ESP32-S31

Technical Reference Manual Pre-release v0.1

PRELIMINARY

ESPRESSIF

About This Document

The **ESP32-S31 Technical Reference Manual** is targeted at developers working on low level software projects that use the ESP32-S31 SoC. It describes the hardware modules listed below for the ESP32-S31 SoC and other products in ESP32-S31 series. The modules detailed in this document provide an overview, list of features, hardware architecture details, any necessary programming procedures, as well as register descriptions.

Navigation in This Document

Here are some tips on navigation through this extensive document:

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

Release Status at a Glance

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

No.	Chapter	Progress
Part I. Microprocessor and Master		
1	Bus Architecture [to be added later]	67%
2	High-Performance CPU [to be added later]	6%
3	RISC-V Trace Encoder (TRACE)	Published
4	Low-Power CPU	Published
5	GDMA Controller (GDMA-AHB, GDMA-AXI)	Published
6	2D-DMA Controller (2D-DMA)	Published
Part II. Memory Organization		
7	System and Memory [to be added later]	83%
8	eFuse Controller (eFuse) [to be added later]	18%
Part III. System Component		
9	GPIO Matrix and IO MUX [to be added later]	67%
10	Reset and Clock [to be added later]	53%
11	Chip Boot Control [to be added later]	76%
12	Interrupt Matrix [to be added later]	17%
13	Event Task Matrix (ETM) [to be added later]	6%
14	Low-Power Management [to be added later]	6%
15	System Timer [to be added later]	27%
16	Timer Group (TIMG) [to be added later]	45%
17	Watchdog Timers (WDT)	Published
18	RTC Timer	Published
19	Permission Control (PMS) [to be added later]	6%
20	System Registers [to be added later]	6%
21	Debug Assistant	Published
22	LP Mailbox	Published
23	Power Supply Detector (PSDET)	Published
Part IV. Cryptography/Security Component		
24	AES Accelerator (AES)	Published
25	ECC Accelerator (ECC)	Published
26	HMAC Accelerator (HMAC)	Published
27	RSA Accelerator (RSA)	Published
28	SHA Accelerator (SHA)	Published
29	RSA Digital Signature Peripheral (RSA_DS)	Published
30	ECDSA Digital Signature Peripheral (ECDSA_DS)	Published
31	Secure Debug Controller (SDC) [to be added later]	45%
32	External Memory Encryption and Decryption (XTS_AES) [to be added later]	17%
33	Random Number Generator (RNG) [to be added later]	94%
34	Key Manager [to be added later]	25%
Part V. Image and Voice Processing		
35	JPEG Codec	Published

No.	Chapter	Progress
36	Pixel-Processing Accelerator (PPA)	Published
37	Audio Sample Rate Converter	Published
38	CORDIC Accelerator (CORDIC) [to be added later]	15%
39	LCD and Camera Controller (LCD_CAM)	Published
Part VI. Connectivity Interface		
40	UART Controller (UART)	Published
41	SPI Controller (SPI) [to be added later]	27%
42	I2C Controller (I2C) [to be added later]	82%
43	I2S Controller (I2S) [to be added later]	55%
44	Pulse Count Controller (PCNT) [to be added later]	64%
45	USB 2.0 High-Speed OTG [to be added later]	36%
46	USB Serial/JTAG Controller	Published
47	Ethernet Media Access Controller (EMAC) [to be added later]	12%
48	Two-Wire Automotive Interface (TWAI) [to be added later]	18%
49	SD/MMC Host Controller (SDHOST) [to be added later]	33%
50	LED PWM Controller (LEDC)	Published
51	Motor Control PWM (MCPWM)	Published
52	Remote Control Peripheral (RMT)	Published
53	Parallel IO Controller (PARLIO)	Published
54	BitScrambler [to be added later]	27%
Part VII. Analog Signal Processing		
55	Touch Sensor (TOUCH)	Published
56	Temperature Sensor (TSENS) [to be added later]	82%
57	SAR ADC Digital Controller (ADC)	Published
58	DAC Controller (DAC) [to be added later]	75%
59	Analog Voltage Comparator	Published

Note:

Check the link or the QR code to make sure that you use the latest version of this document:
https://www.espressif.com/documentation/esp32-s31_technical_reference_manual_en.pdf

Contents

I Microprocessor and Master 30

1 RISC-V Trace Encoder (TRACE) 31
1.1 Terminology 31
1.2 Introduction 32
1.3 Features 33
1.4 Architectural Overview 34
1.5 Functional Description 34
1.5.1 Synchronization 34
1.5.2 Address Mode 35
1.5.3 Optional Sideband Signals 35
1.5.4 Filtering 36
1.5.5 Anchor Tag 36
1.5.6 Memory Writing Mode 37
1.5.7 Automatic Restart 37
1.6 Encoder Output Packets 37
1.6.1 Header 37
1.6.2 Index 37
1.6.3 Payload 38
1.6.3.1 Format 3 Packets 38
1.6.3.2 Format 2 Packets 40
1.6.3.3 Format 1 Packets 40
1.7 Interrupt 42
1.8 Programming Procedures 43
1.8.1 Encoder Option Configuration 43
1.8.2 Filter Configuration 44
1.8.3 Enable Encoder 45
1.8.4 Disable Encoder 46
1.8.5 Notify 46
1.8.6 Decode Data Packets 46
1.8.7 AHB Configuration 47
1.8.8 Software Retention 47
1.9 Register Summary 48
1.10 Registers 49

2 Low-Power CPU 60
2.1 Overview 60
2.2 Features 61
2.3 Configuration and Status Registers (CSRs) 61
2.3.1 Register Summary 61
2.3.2 Registers 62
2.4 Interrupts and Exceptions 72

2.4.1 Interrupts 72
2.4.2 Interrupt Handling 73
2.4.3 Exceptions 73
2.5 Debugging 74
2.5.1 Features 74
2.5.2 Functional Description 74
2.5.3 Register Summary 74
2.5.4 Registers 75
2.6 Hardware Trigger 77
2.6.1 Features 77
2.6.2 Functional Description 77
2.6.3 Trigger Execution Flow 78
2.6.4 Register Summary 78
2.6.5 Registers 78
2.7 Performance Counter 81
2.8 System Access 82
2.8.1 Memory Access 82
2.8.2 Peripheral Access 82
2.9 Event Task Matrix Feature 82
2.10 Sleep and Wake-Up Process 83
2.10.1 Features 83
2.10.2 Process 83
2.10.3 Wake-Up Sources 85
2.10.4 Sleep Rejection 85
2.11 Physical Memory Protection 86
2.11.1 Features 86
2.11.2 Functional Description 86
2.11.3 Register Summary 86
2.11.4 Registers 86

3 GDMA Controller (GDMA-AHB, GDMA-AXI) 90
3.1 Overview 90
3.2 Features 90
3.3 Architecture 91
3.4 Functional Description 92
3.4.1 Linked List 93
3.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 94
3.4.3 Memory-to-Memory Data Transfer 95
3.4.4 Enabling GDMA 96
3.4.5 Linked List Reading Process 96
3.4.6 Interrupt Response Mechanism for GDMA-AXI Linked List Switching 97
3.4.7 EOF 97
3.4.8 Accessing Memory 98
3.4.9 Arbitration 99
3.4.10 Bus Error Response Information Logging 99
3.4.11 Automatic Clock Gating 100

3.5 Event Task Matrix Feature 100
3.6 Interrupts 101
3.7 Programming Procedures 103
3.7.1 Programming Procedures for GDMA's Transmit Channel 103
3.7.2 Programming Procedures for GDMA's Receive Channel 103
3.7.3 Programming Procedures for Memory-to-Memory Transfer 104
3.7.4 Programming Procedures for Channel Priority and Weight 104
3.8 Register Summary 106
3.8.1 GDMA-AHB Register Summary 106
3.8.1.1 GDMA-AHB-HP Register Summary 106
3.8.1.2 GDMA-AHB-LP Register Summary 114
3.8.2 GDMA-AXI Register Summary 117
3.9 Registers 123
3.9.1 GDMA-AHB-HP Registers 123
3.9.2 GDMA-AHB-LP Registers 156
3.9.3 GDMA-AXI Registers 190

4 2D-DMA Controller (2D-DMA) 230
4.1 Overview 230
4.2 Features 230
4.3 Architecture 230
4.4 Functional Description 231
4.4.1 Transfer Mode 231
4.4.2 Linked List 232
4.4.3 Padding in 2D-MOD1 Mode and DSCR-PORT Mode 235
4.4.4 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 236
4.4.5 Memory-to-Memory Data Transfer 237
4.4.6 Macroblock Reordering 237
4.4.6.1 Macroblock Reordering in TX Direction 237
4.4.6.2 Macroblock Reordering in RX Direction 238
4.4.7 Color Space Conversion 239
4.4.8 Enabling 2D-DMA 242
4.4.9 Linked List Reading Process 243
4.4.10 EOF 243
4.4.11 Accessing Memory 244
4.4.12 Arbitration 244
4.5 Event Task Matrix Feature 245
4.6 Interrupts 246
4.7 Programming Procedures 248
4.7.1 General Configurations for 2D-DMA 248
4.7.2 Mode-Specific Configurations for 2D-DMA 248
4.7.3 Configurations for 2D-DMA's Transmit Channel 248
4.7.4 Configurations for 2D-DMA's Receive Channel 249
4.7.5 Configurations for Memory-to-Memory Transfer 249
4.7.6 Configurations for Channel Priority and Weight 249
4.7.7 Resetting 2D-DMA While it Runs 250

4.8 Register Summary 251
4.9 Registers 258

II System Component 302

5 Watchdog Timers (WDT) 303
5.1 Overview 303
5.2 Digital Watchdog Timers 304
5.2.1 Features 304
5.2.2 Functional Description 305
5.2.2.1 Clock Source and 32-Bit Counter 305
5.2.2.2 Stages and Timeout Actions 306
5.2.2.3 Write Protection 306
5.2.2.4 Flash Boot Protection 307
5.3 Super Watchdog 307
5.3.1 Features 307
5.3.2 Super Watchdog Controller 307
5.3.2.1 Structure 308
5.3.2.2 Workflow 308
5.4 Interrupts 308
5.5 Register Summary 309
5.6 Registers 309

6 RTC Timer 318
6.1 Introduction 318
6.2 Feature List 318
6.3 Functional Description 318
6.4 Event Task Matrix Feature 320
6.5 Interrupts 320
6.6 Register Summary 322
6.7 Registers 323

7 Debug Assistant 332
7.1 Overview 332
7.2 Features 332
7.3 Functional Description 332
7.3.1 Region Read/Write Monitoring 332
7.3.2 SP Monitoring 333
7.3.3 PC and SP Logging 333
7.3.4 Lockup Monitoring 333
7.3.5 CPU/DMA Bus Access Logging 333
7.4 Interrupts 333
7.5 Programming Procedures 334
7.5.1 Region Monitoring and SP Monitoring Configuration 334
7.5.2 PC and SP Logging Configuration 335
7.5.3 Lockup Monitoring Configuration 336

7.5.4 CPU/DMA Bus Access Logging Configuration 336
7.5.4.1 Bus Access Logging 1 Configuration 336
7.5.4.2 Bus Access Logging 2 Configuration 340
7.6 Register Summary 343
7.6.1 Bus Logging Configuration Registers 343
7.6.2 Other Registers 344
7.7 Registers 347
7.7.1 Bus Logging Configuration Registers 347
7.7.2 Other Registers 357

8 LP Mailbox 378
8.1 Overview 378
8.2 Features 378
8.3 Functional Description 379
8.3.1 Message Registers 379
8.3.2 Interrupts 379
8.3.3 Inter-Core Communication 380
8.4 Register Summary 381
8.5 Registers 382

9 Power Supply Detector (PSDET) 395
9.1 Overview 395
9.2 Features 395
9.3 Functional Description 395
9.3.1 Architecture 395
9.3.2 Brown-out Detector 396
9.3.3 Voltage Glitch Detectors 397
9.4 Interrupts 398
9.5 Register Summary 400
9.6 Registers 401

III Cryptography/Security Component 408

10 AES Accelerator (AES) 409
10.1 Introduction 409
10.2 Features 409
10.3 Clock and Reset 409
10.4 AES Working Modes 410
10.5 Typical AES Working Mode 411
10.5.1 Key, Plaintext, and Ciphertext 411
10.5.2 Endianness 412
10.5.3 Operation Process 414
10.6 DMA-AES Working Mode 415
10.6.1 Key, Plaintext, and Ciphertext 415
10.6.2 Endianness 416
10.6.3 Standard Incrementing Function 417

10.6.4 Block Number 417
10.6.5 Initialization Vector 417
10.6.6 Block Operation Process 417
10.6.7 GCM Operation Process 418
10.7 GCM Algorithm 419
10.7.1 Hash Subkey 420
10.7.2 J_0 420
10.7.3 Authentication Tag 420
10.7.4 AAD Block Number 421
10.7.5 Number of Effective Bits of Incomplete Blocks 421
10.8 Pseudo-Round Anti-DPA Function 421
10.9 Interrupts 422
10.10 Memory Summary 422
10.11 Register Summary 422
10.12 Registers 424

11 ECC Accelerator (ECC) 432
11.1 Overview 432
11.2 Feature List 432
11.3 ECC Basics 432
11.3.1 Elliptic Curve and Points on the Curves 432
11.3.2 Affine Coordinates and Jacobian Coordinates 433
11.3.3 Memory Blocks 433
11.3.4 Data and Data Block 433
11.3.5 Writing Data 434
11.3.6 Reading Data 434
11.3.7 Standard Calculation and Jacobian Calculation 434
11.4 Function Description 434
11.4.1 Curve Mode 435
11.4.2 Working Modes 435
11.4.2.1 Affine Point Multiplication (Affine Point Multi) 436
11.4.2.2 Affine Point Verification (Affine Point Verif) 436
11.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi) 436
11.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi) 436
11.4.2.5 Point Addition (Point Add) 437
11.4.2.6 Jacobian Point Verification (Jacobian Point Verif) 437
11.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi) 437
11.4.2.8 Mod Addition (Mod Add) 438
11.4.2.9 Mod Subtraction (Mod Sub) 438
11.4.2.10 Mod Multiplication (Mod Multi) 438
11.4.2.11 Mod Division (Mod Div) 439
11.4.3 Enhancing Anti-Attack Performance 439
11.4.4 Clock 439
11.4.5 Reset 440
11.5 Interrupts 440

11.6 Programming Procedures 441
11.7 Register Summary 442
11.8 Registers 443

12 HMAC Accelerator (HMAC) 448
12.1 Overview 448
12.2 Feature List 448
12.3 Functional Description 448
12.3.1 Upstream Mode 449
12.3.2 Downstream Mode - JTAG Enable Feature 449
12.3.3 Downstream Mode - RSA Digital Signature Peripheral and Key Derivation Feature 450
12.4 HMAC eFuse Configuration 450
12.5 HMAC Process (Detailed) 451
12.5.1 Enable HMAC module 451
12.5.2 Configure HMAC keys and key purposes 452
12.5.3 Downstream mode process 452
12.5.4 Upstream mode process 452
12.6 HMAC Algorithm Details 453
12.6.1 Padding Bits 453
12.6.2 HMAC Algorithm Structure 454
12.6.3 Hardware Occupation 455
12.7 Register Summary 456
12.8 Registers 457

13 RSA Accelerator (RSA) 464
13.1 Introduction 464
13.2 Features 464
13.3 Functional Description 464
13.3.1 Definitions and Representations 465
13.3.2 Large-Number Modular Exponentiation 465
13.3.3 Large-Number Modular Multiplication 466
13.3.4 Large-Number Multiplication 467
13.3.5 Options for Additional Acceleration 467
13.4 Interrupts 469
13.5 Memory Summary 470
13.6 Register Summary 470
13.7 Registers 471

14 SHA Accelerator (SHA) 476
14.1 Overview 476
14.2 Features 476
14.3 Working Modes 476
14.4 Function Description 477
14.4.1 Preprocessing 477
14.4.1.1 Padding the Message 478
14.4.1.2 Parsing the Message 478

14.4.1.3 Setting the Initial Hash Value 479
14.4.2 Hash Operation 480
14.4.2.1 Typical SHA Mode Process 480
14.4.2.2 DMA-SHA Mode Process 482
14.4.3 Message Digest 483
14.5 Interrupt 484
14.6 Register Summary 486
14.7 Registers 488

15 RSA Digital Signature Peripheral (RSA_DS) 493
15.1 Overview 493
15.2 Features 493
15.3 Functional Description 493
15.3.1 Overview 493
15.3.2 Private Key Operands 494
15.3.3 Software Prerequisites 494
15.3.4 RSA_DS Operation at the Hardware Level 496
15.3.4.1 Hardware Occupation 496
15.3.5 RSA_DS Operation at the Software Level 497
15.4 Memory Summary 500
15.5 Register Summary 501
15.6 Registers 502

16 ECDSA Digital Signature Peripheral (ECDSA_DS) 506
16.1 Introduction 506
16.2 Features 506
16.3 ECDSA Basics 506
16.3.1 Domain Parameters 506
16.3.2 Key Generation 507
16.3.3 Signature Generation 507
16.3.4 Signature Verification 508
16.4 Functional Description 508
16.4.1 ECDSA_DS Working Modes 508
16.4.2 Key Source 510
16.4.3 Data and Data Block 510
16.4.3.1 Writing Data 510
16.4.3.2 Reading Data 511
16.4.3.3 Padding the Message 511
16.4.3.4 Parsing the Message 511
16.4.4 Security Features 511
16.4.4.1 High Anti-Attack Performance 512
16.4.4.2 State-Dependent Register Access Control 512
16.4.4.3 Hardware Occupation 512
16.5 Programming Procedures 513
16.5.1 ECDSA_DS Process 513
16.5.1.1 IDLE Stage 513

16.5.1.2 PREP Stage 514
16.5.1.3 LOAD Stage 514
16.5.1.4 PROC Stage 515
16.5.1.5 GAIN Stage 515
16.5.1.6 POST Stage 515
16.5.1.7 ECDSA_DS SHA Interface 515
16.5.2 Clock 516
16.5.3 Reset 518
16.6 Interrupts 518
16.7 Memory Blocks 520
16.8 Register Summary 521
16.9 Registers 523

IV Image and Voice Processing 534

17 JPEG Codec 535
17.1 Terminology 535
17.2 Overview 536
17.3 Feature List 537
17.4 Architectural Overview 538
17.5 Functional Description 540
17.5.1 JPEG Encoder 540
17.5.1.1 Pause 540
17.5.1.2 Color Space Conversion 540
17.5.1.3 Configurable Quantization Coefficient Table 541
17.5.1.4 Stuffed Zero Byte 543
17.5.1.5 EOI Marker 543
17.5.1.6 Timeout Detection 543
17.5.2 JPEG Decoder 543
17.5.2.1 Multiple Components 544
17.5.2.2 Parsing RST Marker 545
17.5.2.3 Configurable Quantization Coefficient Table 545
17.5.2.4 Configurable Huffman Table 545
17.5.2.5 Timeout Detection 547
17.6 Interrupts 547
17.7 Programming Procedures 549
17.7.1 JPEG Encoder 549
17.7.2 JPEG Decoder 552
17.7.3 Reset 556
17.8 Register Summary 557
17.9 Registers 559

18 Pixel-Processing Accelerator (PPA) 587
18.1 Overview 587
18.2 Terminology 587
18.3 Features 588

18.4 Architectural Overview 588
18.5 Functional Description 589
18.5.1 PPA Color Space 589
18.5.1.1 SRM Color Space 590
18.5.1.2 BLEND Color Space 592
18.5.2 PPA 2D-DMA Linked List Configuration 593
18.5.2.1 SRM Special Configuration 595
18.5.3 Scaling - Rotation - Mirroring (SRM) 595
18.5.3.1 Basic Functionality 595
18.5.3.2 Pixel Block Reorder 596
18.5.4 Layer Blending (BLEND) 597
18.5.4.1 Basic Functionality 597
18.5.4.2 Filled Image Output 598
18.6 Interrupts 598
18.7 Programming Procedures 599
18.7.1 PPA Clock Reset Configuration 599
18.7.2 SRM Configuration 600
18.7.3 BLEND CLUT Configuration 601
18.7.4 BLEND Configuration 601
18.7.5 BLEND Image Filling Configuration 602
18.7.6 Error Handling 602
18.8 Register Summary 603
18.9 Registers 605

19 Audio Sample Rate Converter 629
19.1 Overview 629
19.2 Terminology 629
19.3 Features 629
19.4 Architectural Overview 630
19.5 Functional Description 631
19.5.1 Clock 631
19.5.2 Reset 631
19.5.3 Resampler 631
19.5.4 Fractional Sample Rate Converter 632
19.5.5 Input/Output Data Length 632
19.5.5.1 No Residual Data 633
19.5.5.2 Residual Data Handling 633
19.5.6 DMA Interface 633
19.6 Interrupts 634
19.7 Configuration Procedure 634
19.7.1 Usage Restrictions and Notes 635
19.8 Application Example 637
19.9 Register Summary 638
19.10 Registers 639

20 LCD and Camera Controller (LCD_CAM) 652

20.1 Overview 652
20.2 Features 652
20.3 Functional Description 653
20.3.1 Block Diagram 653
20.3.2 Signal Description 653
20.3.3 LCD_CAM Module Clocks 654
20.3.3.1 LCD Clock 654
20.3.3.2 Camera Clock 656
20.3.4 LCD_CAM Reset 657
20.3.5 LCD_CAM Data Format Control 657
20.3.5.1 LCD Data Format Control 658
20.3.5.2 Camera Data Format Control 660
20.3.6 YUV-RGB Data Format Conversion 661
20.3.6.1 YUV Formats 662
20.3.6.2 Format Conversion Configuration 662
20.3.7 LCD_CAM Timing 664
20.3.7.1 LCD Timing (RGB Format) 664
20.3.7.2 LCD Timing (I8080/MOTO6800 Format) 665
20.4 Interrupts 665
20.5 Software Configuration Process 666
20.5.1 Configure LCD (RGB Format) as TX Mode 666
20.5.2 Configure LCD (I8080/MOTO6800 Format) as TX Mode 667
20.5.3 Configure Camera as RX Mode 668
20.6 Register Summary 670
20.7 Registers 671

V Connectivity Interface 690

21 UART Controller (UART) 691
21.1 Overview 691
21.2 Features 691
21.3 UART Structure 692
21.4 Functional Description 693
21.4.1 Clock and Reset 693
21.4.2 UART FIFO 694
21.4.3 Baud Rate Generation and Detection 694
21.4.3.1 Baud Rate Generation 694
21.4.3.2 Baud Rate Detection 695
21.4.4 UART Data Frame 696
21.4.5 AT_CMD Character Structure 697
21.4.6 RS485 697
21.4.6.1 Driver Control 697
21.4.6.2 Turnaround Delay 698
21.4.6.3 Bus Snooping 698
21.4.7 IrDA 698
21.4.8 Wakeup 699

21.4.9 Flow Control 700
21.4.9.1 Hardware Flow Control 700
21.4.9.2 Software Flow Control 702
21.4.10 GDMA Mode 702
21.5 Interrupts 703
21.6 Programming Procedures 705
21.6.1 Register Type 705
21.6.2 Detailed Steps 706
21.6.2.1 Initializing UART*n* 706
21.6.2.2 Configuring UART*n* Communication 706
21.6.2.3 Enabling UART*n* 707
21.7 Register Summary 708
21.7.1 UART Register Summary 708
21.7.2 LP UART Register Summary 709
21.7.3 UHCI Register Summary 710
21.8 Registers 712
21.8.1 UART Registers 712
21.8.2 LP UART Registers 735
21.8.3 UHCI Registers 755

22 USB Serial/JTAG Controller 776
22.1 Overview 776
22.2 Feature List 776
22.3 Functional Description 778
22.3.1 CDC-ACM USB Interface Functional Description 778
22.3.2 CDC-ACM Firmware Interface Functional Description 779
22.3.3 USB-to-JTAG Interface: JTAG Command Processor 780
22.3.4 USB-to-JTAG Interface: CMD_REP Usage Example 781
22.3.5 USB-to-JTAG Interface: Response Capture Unit 782
22.3.6 USB-to-JTAG Interface: Control Transfer Requests 782
22.4 Interrupts 783
22.5 Programming Procedures 784
22.6 Register Summary 786
22.7 Registers 788

23 LED PWM Controller (LEDC) 814
23.1 Overview 814
23.2 Features 814
23.3 Architectural Overview 814
23.4 Functional Description 815
23.4.1 Clock 815
23.4.2 Reset 816
23.4.3 Timers 816
23.4.3.1 Clock Divider Configuration 816
23.4.3.2 20-Bit Counter 817
23.4.3.3 Timer Configuration Constraints 819

23.4.4 PWM Generators 819
23.4.4.1 PWM Output Signal Dithering 820
23.4.4.2 Applying the Changed Configuration 820
23.4.5 Duty Cycle Fading 820
23.4.5.1 Linear Duty Cycle Fading 820
23.4.5.2 Gamma Curve Fading 821
23.4.5.3 Suspend and Resume Duty Cycle Fading 822
23.5 Event Task Matrix Feature 822
23.6 Interrupts 824
23.7 Programming Procedures 824
23.8 Memory Blocks 826
23.9 Register Summary 828
23.10 Registers 830

24 Motor Control PWM (MCPWM) 843
24.1 Overview 843
24.2 Features 843
24.3 Modules 846
24.3.1 Overview 846
24.3.1.1 Timer Module 846
24.3.1.2 Operator Module 846
24.3.1.3 Capture Module 847
24.3.1.4 ETM Module 847
24.3.2 PWM Timer Module 848
24.3.2.1 Configurations of the PWM Timer Module 848
24.3.2.2 PWM Timer's Working Modes and Timing Event Generation 848
24.3.2.3 Shadow Register of PWM Timer 854
24.3.2.4 PWM Timer Synchronization and Phase Locking 854
24.3.3 PWM Operator Module 854
24.3.3.1 PWM Generator Module 855
24.3.3.2 Dead Time Generator Module 868
24.3.3.3 PWM Carrier Module 873
24.3.3.4 Fault Detection Module 875
24.3.4 Capture Module 876
24.3.4.1 Introduction 876
24.3.4.2 Capture Timer 877
24.3.4.3 Capture Channel 877
24.3.5 ETM Module 877
24.3.5.1 Overview 877
24.3.5.2 MCPWM-Related ETM Events 877
24.3.5.3 MCPWM-Related ETM Tasks 878
24.4 Interrupts 879
24.5 Register Summary 880
24.6 Registers 883

25 Remote Control Peripheral (RMT) 930

25.1 Overview 930
25.2 Features 930
25.3 Functional Description 931
25.3.1 Architecture 931
25.3.2 RAM 932
25.3.2.1 Structure of RAM 932
25.3.2.2 Use of RAM 933
25.3.2.3 RAM Access 933
25.3.3 Clock 934
25.3.4 Transmitter 934
25.3.4.1 Normal TX Mode 935
25.3.4.2 Wrap TX Mode 935
25.3.4.3 TX Modulation 935
25.3.4.4 Continuous TX Mode 936
25.3.4.5 Simultaneous TX Mode 936
25.3.5 Receiver 936
25.3.5.1 Normal RX Mode 936
25.3.5.2 Wrap RX Mode 937
25.3.5.3 RX Filtering 937
25.3.5.4 RX Demodulation 937
25.3.6 Configuration Update 937
25.4 Interrupts 938
25.5 Register Summary 940
25.6 Registers 942

26 Parallel IO Controller (PARLIO) 957
26.1 Introduction 957
26.2 Glossary 957
26.3 Features 957
26.4 Architectural Overview 959
26.5 Functional Description 959
26.5.1 Clock Generator 959
26.5.2 Clock & Reset Restriction 960
26.5.3 Master-Slave Mode 962
26.5.4 Receive Modes of the RX Unit 963
26.5.4.1 Level Enable Mode 963
26.5.4.2 Pulse Enable Mode 963
26.5.4.3 Software Enable Mode 964
26.5.5 RX Unit GDMA SUC EOF Generation 965
26.5.6 TX Unit EOF Generation 965
26.5.7 RX Unit Timeout 965
26.5.8 Unlimited Length Data Transmission of TX Unit 965
26.5.9 TX Chip Select Function 966
26.5.10 Output Clock Gating of TX Unit 966
26.5.11 Bus Idle Value of TX Unit 966
26.5.12 Data Transfer in a Single Frame 966

26.5.13 Bit Reversal in One Byte 967
26.6 Interrupts 967
26.7 Programming Procedures 968
26.7.1 Data Receiving Operation Process 968
26.7.2 Data Transmitting Operation Process 968
26.8 Application Examples 969
26.8.1 Co-working with SPI 969
26.8.2 Co-working with I2S 971
26.8.3 Co-working with LCD 971
26.9 Register Summary 973
26.10 Registers 974

VI Analog Signal Processing 985

27 Touch Sensor (TOUCH) 986
27.1 Terminology 986
27.2 Feature List 986
27.3 Architectural Overview 987
27.3.1 Touch Panel 987
27.3.2 Capacitive Touch Pin 988
27.3.3 Touch Sensor 988
27.4 Functional Description 990
27.4.1 Touch FSM 990
27.4.2 Sampled Signal Preprocessing 991
27.4.2.1 Measurement Process 991
27.4.2.2 Trigger Source of Measurement 992
27.4.2.3 Scan Mode 993
27.4.2.4 Frequency Hopping 994
27.4.3 Touch Detection 996
27.4.3.1 Sampled Value 996
27.4.3.2 Hardware Touch Detection 997
27.4.4 Proximity Mode 997
27.4.5 Sleep Mode 998
27.4.6 Moisture Tolerance 999
27.4.7 Water Rejection 999
27.5 Interrupts 1000
27.6 Register Summary 1001
27.6.1 Interrupt and Status Register Summary 1001
27.6.2 Configuration Register Summary 1002
27.7 Registers 1005
27.7.1 Interrupt and Status Registers 1005
27.7.2 Configuration Registers 1012

28 SAR ADC Digital Controller (ADC) 1030
28.1 Overview 1030
28.2 Features 1030

28.2.1 ADC Channels 1031
28.2.2 ADC Clock 1031
28.3 Trigger Control 1032
28.3.1 Features 1032
28.3.2 Functional Description 1032
28.3.2.1 Trigger Mode Selection 1032
28.3.2.2 Timer Trigger 1032
28.3.2.3 Software Trigger and Stop 1032
28.4 Pattern Table 1033
28.4.1 Features 1033
28.4.2 Functional Description 1033
28.4.2.1 Pattern Table Structure 1033
28.4.2.2 Scan Length and Pointer 1034
28.4.2.3 Update Timing 1034
28.4.2.4 Channel and Mode Selection 1035
28.5 SAR Control FSM 1035
28.5.1 Features 1035
28.5.2 Functional Description 1035
28.5.2.1 State Machine Flow 1035
28.5.2.2 Continuous Sampling Mode 1035
28.5.2.3 SAR Power Control 1038
28.5.2.4 State Query 1038
28.6 Data Processing 1038
28.6.1 Features 1038
28.6.2 Functional Description 1038
28.6.2.1 Data Arbitration 1038
28.6.2.2 IIR Filter 1039
28.6.2.3 Reading Latest Sample 1039
28.7 Threshold Monitor 1039
28.7.1 Features 1039
28.7.2 Functional Description 1040
28.7.2.1 Threshold Configuration 1040
28.7.2.2 Comparison Logic 1040
28.7.2.3 All-Channel Monitoring Mode 1040
28.8 DMA Control 1040
28.8.1 Features 1040
28.8.2 Functional Description 1040
28.8.2.1 DMA Enable 1040
28.8.2.2 EOF Frame Management 1041
28.8.2.3 DMA Data Format 1041
28.8.2.4 FIFO Overflow 1041
28.9 ETM Events 1041
28.9.1 Features 1041
28.9.2 Functional Description 1041
28.9.2.1 ETM Trigger Inputs 1041
28.9.2.2 ETM Event Outputs 1042

28.9.2.3 ETM Clock 1042
28.10 Interrupts 1042
28.11 Register Summary 1043
28.12 Registers 1044

29 Analog Voltage Comparator 1061
29.1 Introduction 1061
29.2 Feature List 1061
29.3 Architectural Overview 1061
29.4 Functional Description 1062
29.4.1 Reference Voltage Configuration 1062
29.4.2 PAD COMP Controller 1063
29.4.3 Clock 1065
29.4.4 Reset 1066
29.5 Event Task Matrix 1066
29.6 Interrupts 1067
29.7 Programming Procedures 1067
29.8 Register Summary 1069
29.9 Registers 1070

VII Appendix 1082

Related Documentation and Resources 1083

Glossary 1085
Abbreviations for Peripherals 1085
Abbreviations Related to Registers 1086
Access Types for Registers 1087

Programming Reserved Register Field 1089
Introduction 1089
Programming Reserved Register Field 1089

Interrupt Configuration Registers 1090

Revision History 1091

List of Tables

1.3-1 Trace Encoder Parameters 33
1.5-1 Optional sideband signals 35
1.6-1 Header Format 37
1.6-2 Index Format 38
1.6-3 Packet format 3 subformat 0 38
1.6-4 Packet format 3 subformat 1 38
1.6-5 Packet format 3 subformat 3 39
1.6-6 Packet format 2 40
1.6-7 Packet format 1 with address 41
1.6-8 Packet format 1 without address 42
1.7-1 CORE*n*_TRACE's Internal Interrupt Sources 42
1.8-1 Debug module trigger support (the action field of the mcontrol register) 44
1.8-2 Trace Status 46

2.4-1 LP CPU Interrupt Mapping 72
2.4-2 LP CPU Exception Causes 73
2.7-1 Performance Counter 81
2.10-1 Wake Sources 85

3.4-1 GDMA-AHB-HP Selecting Peripherals via Register Configuration 94
3.4-2 GDMA-AHB-LP Selecting Peripherals via Register Configuration 95
3.4-3 GDMA-AXI Selecting Peripherals via Register Configuration 95

4.4-1 Descriptor Field Description 233
4.4-2 Selecting Peripherals via Register Configuration in TX Direction 237
4.4-3 Selecting Peripherals via Register Configuration in RX Direction 237
4.4-4 Recommended Configurations for Macroblock Reordering in TX Direction 238
4.4-5 Recommended Configuration Table for Macroblock Reordering in 2D-DMA Receive Direction (Without Color Format Conversion) 238
4.4-6 Parameter Configuration for Color Space Conversion in TX Direction 240
4.4-7 Parameter Configuration for Color Space Conversion in RX Direction 241
4.4-8 RGBB888 Storage Format 242
4.4-9 YUV444 Storage Format 242
4.4-10 YUV422 Storage Format 242

5.2-1 Timeout Actions 306

6.3-1 Configure RTC Timer to Log the Occurrence Time of Specific Events 319
6.3-2 Target times Configuration 319
6.5-1 RTC Timer's Internal Interrupt Sources 320

7.5-1 HP CPU*n* Packet Format 338
7.5-2 DMA Packet Format 338
7.5-3 LOST Packet Format 338
7.5-4 DMA_0 Packet Format 341

7.5-5 DMA LOST Packet Format 342

8.3-1 LP Mailbox's Internal Interrupt Sources 380

9.3-1 Configuration for Detecting Power Pins 398
9.4-1 Power Supply Detector's Internal Interrupt Source 398

10.4-1 AES Accelerator Working Mode 410
10.4-2 Key Length and Encryption/Decryption 410
10.5-1 Working Status under Typical AES Working Mode 411
10.5-2 Text Endianness Type for Typical AES 412
10.5-3 Key Endianness Type for AES-128 Encryption and Decryption 413
10.5-4 Key Endianness Type for AES-256 Encryption and Decryption 413
10.6-1 Block Cipher Mode 415
10.6-2 Working Status under DMA-AES Working mode 415
10.6-3 TEXT-PADDING 416
10.6-4 Text Endianness for DMA-AES 416
10.9-1 AES's Internal Interrupt Sources 422

11.3-1 ECC Accelerator Memory Blocks 433
11.4-1 ECC Accelerator Curve Mode Selection 435
11.4-2 Working Modes of ECC Accelerator 435

12.4-1 HMAC Purposes and Configuration Value 451

13.3-1 Acceleration Performance 469
13.4-1 RSA's Internal Interrupt Source 469
13.5-1 RSA Accelerator Memory Blocks 470

14.3-1 SHA Accelerator Working Mode 477
14.3-2 SHA Hash Algorithm Selection 477
14.4-4 The Storage and Length of Message Digest from Different Algorithms 484
14.5-1 SHA's Internal Interrupt Sources 484

16.4-1 ECDSA_DS Working Mode 509
16.4-2 ECDSA_DS Elliptic Curve Selection 509
16.4-3 ECDSA_DS SHA Algorithm 509
16.4-4 ECDSA_DS Working State 510
16.7-1 ECDSA_DS Memory Blocks 520

17.1-2 Marker Code Assignments 536
17.2-1 Number of Data Units in the MCU for Different Image Formats 536
17.5-1 Supported Image Formats for Color Space Conversion 541
17.5-2 Position Number of Coefficients in a Quantization Coefficient Table (Example) 541
17.5-3 Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component 542
17.5-4 Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component 542
17.5-5 Sampling Factor and Image Format 544
17.7-1 JPEG Encoder hb and vb Configuration 550
17.7-2 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configuration 551

17.7-3 JPEG Decoder JPEG_HA and JPEG_VA Configuration 552
17.7-4 JPEG Decoder hb and vb Configuration 553
17.7-5 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration 554
17.7-6 2D DMA Color Space Conversion Configuration 555

18.5-1 Pixel Storage Order 589
18.5-2 PPA SRM Alpha Configuration 591
18.5-3 BLEND CLUT Address 593
18.5-4 PPA Linked List Configuration 594
18.5-5 Automatic Padding Block Size Configuration 595
18.5-6 Number of Pixel Block Reorders in SRM Output 597
18.6-1 SRM Parameter Error Types 598
18.6-2 BLEND Parameter Error Types 599

19.6-1 Internal Interrupt Sources of ASRC 634

20.3-1 LCD_CAM Input and Output Signal Descriptions 654
20.3-2 LCD Data Format Control – Stage One 659
20.3-3 LCD Output Data Reordering 660
20.3-4 Camera Data Format Control 661
20.3-5 Conversion Mode Control in 24-bit Mode 663
20.3-6 Conversion Mode Control in Other Modes 663

21.2-1 UART and LP UART Feature Comparison 691
21.4-1 UART_CHAR_WAKEUP Mode Configuration 700

22.3-1 Standard CDC-ACM Control Requests 778
22.3-2 CDC-ACM Settings with RTS and DTR 779
22.3-3 Commands of a Nibble 781
22.3-4 USB-to-JTAG Control Requests 783
22.3-5 JTAG Capability Descriptors 783
22.5-1 Reset SoC into Download Mode 785
22.5-2 Reset SoC into Booting from flash 785

23.4-1 Commonly-used Frequencies and Resolutions 818
23.8-1 LEDC Memory Blocks 826

24.3-1 Functions of Each Submodule in the Operator Module 847
24.3-2 Timing Events Used in PWM Generator 856
24.3-3 Timing Events Priority When PWM Timer Increments 857
24.3-4 Timing Events Priority when PWM Timer Decrements 857
24.3-5 Dead Time Generator Switches Control Fields 869
24.3-6 Typical Dead Time Generator Operating Modes 870
24.3-7 MCPWM-Related ETM Events 877
24.3-8 ETM Related Tasks 878

25.3-1 Configuration Update 938

26.5-1 Operations to Reset AXI Clock Domain with Clock Restrictions 961

26.5-2 Requirements for TX Unit Operating as Slave with Clock Restrictions 962
26.5-3 Requirements for RX Unit Operating as Slave with Clock Restrictions 963
26.6-1 PARLIO's Internal Interrupt Sources 967

27.3-1 ESP32-S31 Capacitive Touch Pins 988
27.3-2 Capacitive Touch Sensor Output Signal 989
27.4-1 touch_smooth_data Calculation 996
27.4-2 benchmark Algorithm 997
27.5-1 Touch Sensor's Internal Interrupt Sources 1000

28.2-1 SAR ADC channels 1031
28.10-1 SAR ADC interrupt sources 1042

29.3-1 Mapping Between PADs and GPIO Pins 1062
29.4-1 V_HYS Values 1063
29.6-1 Internal Interrupt Sources of the Analog Voltage Comparator 1067

29.9-4 Configuration of ENA/RAW/ST Registers 1090

List of Figures

1.0-1 Trace Encoder Overview 31
1.2-1 Trace Flow Overview 32

2.1-1 LP CPU Overview 60
2.10-1 Wake-Up and Sleep Flow of LP CPU 83

3.3-1 GDMA Controller Architecture 92
3.4-1 Structure of a Linked List 93
3.4-2 Relationship among Linked Lists 96

4.3-1 2D-DMA Architecture 231
4.4-1 Structure of a Linked List 232
4.4-2 Padding Illustration (Overall) 236
4.4-3 Padding Illustration (Detail) 236
4.4-4 Macroblock Layout of JPEG Decoder Output Data 239
4.4-5 Color Space Conversion 240

5.1-1 Watchdog Timers Overview 304
5.2-1 Digital Watchdog Timers in ESP32-S31 305
5.3-1 Super Watchdog Controller Structure 308

8.3-1 LP Mailbox Block Diagram 379

9.3-1 Architecture of Power Supply Detector 396
9.3-2 Brown-out Detector Handling 397
9.3-3 Voltage Glitch Detectors 398

10.7-1 GCM Encryption Process 419

12.6-1 HMAC SHA-256 Padding Diagram 454
12.6-2 HMAC Structure Schematic Diagram 454

15.3-1 Software Preparations and Hardware Working Process 495

16.5-1 ECDSA_DS Process 513
16.5-2 clk_ecdsa Diagram 517
16.5-3 clk_apb_ecdsa Diagram 517

17.1-1 JPEG Bitstream Format 536
17.2-1 JPEG Codec System Connection Block Diagram 537
17.4-1 JPEG Encoder Architecture 539
17.4-2 JPEG Decoder Architecture 539

18.1-1 PPA 2D-DMA Connection 587
18.4-1 PPA Architecture 588
18.5-1 PPA SRM Input Pixel Processing 590
18.5-2 PPA SRM Alpha Configuration Logic 591
18.5-3 2D-DMA Linked List 594

18.5-4 SRM Procedure 596

19.4-1 ASRC Module Structure 630
19.8-1 Sample Rate Conversion Relationship 637

20.3-1 LCD_CAM Block Diagram 653
20.3-2 LCD Clock 655
20.3-3 Camera Clock 656
20.3-4 LCD Frame Structure 664
20.3-5 LCD Timing (RGB Format) 665
20.3-6 LCD Timing (I8080/MOTO6800 Format) 665

21.3-1 UART Structure 692
21.4-1 UART Controllers Division 695
21.4-2 The Timing Diagram of Weak UART Signals Along Negative Edges 695
21.4-3 Structure of UART Data Frame 696
21.4-4 AT_CMD Character Structure 697
21.4-5 Driver Control Diagram in RS485 Mode 698
21.4-6 The Timing Diagram of Encoding and Decoding in SIR mode 699
21.4-7 IrDA Encoding and Decoding Diagram 699
21.4-8 Hardware Flow Control Diagram 701
21.4-9 Connection between Hardware Flow Control Signals 701
21.4-10 Data Transfer in GDMA Mode 703
21.6-1 UART Programming Procedures 706

22.2-1 USB Serial/JTAG High Level Diagram 777
22.2-2 USB Serial/JTAG Block Diagram 778

23.3-1 LED PWM Architecture 815
23.3-2 Timer and PWM Generator Block Diagram 815
23.4-1 LEDC_CLK Diagram 816
23.4-2 Frequency Division When LEDC_CLK_DIV is a Non-Integer Value 817
23.4-3 Relationship Between Counter And Resolution 817
23.4-4 LED PWM Output Signal Diagram 819
23.4-5 Output Signal of Linear Duty Cycle Fading 821
23.4-6 Output Signal of Gamma Curve Fading 822

24.2-1 MCPWM Module Overview 844
24.3-1 Timer Module 846
24.3-2 Operator Module 846
24.3-3 Capture Module 847
24.3-4 ETM Module 847
24.3-5 Count-Up Mode Waveform 849
24.3-6 Count-Down Mode Waveforms 849
24.3-7 Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event 849
24.3-8 Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event 850
24.3-9 UTEP and UTEZ Generation in Count-Up Mode 851
24.3-10 DTEP and DTEZ Generation in Count-Down Mode 852

24.3-11 DTEP and UTEZ Generation in Count-Up-Down Mode 852
24.3-12 Block Diagram of A PWM Operator 855
24.3-13 Symmetrical Waveform in Count-Up-Down Mode 859
24.3-14 Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 860
24.3-15 Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A 861
24.3-16 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 862
24.3-17 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary 863
24.3-18 Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B 864
24.3-19 Example of an NCI Software-Force Event on PWM*x*A 866
24.3-20 Example of a CNTU Software-Force Event on PWM*x*B 867
24.3-21 Options for Setting up the Dead Time Generator Module 869
24.3-22 Active High Complementary (AHC) Dead Time Waveforms 870
24.3-23 Active Low Complementary (ALC) Dead Time Waveforms 871
24.3-24 Active High (AH) Dead Time Waveforms 871
24.3-25 Active Low (AL) Dead Time Waveforms 872
24.3-26 Example of Waveforms Showing PWM Carrier Action 873
24.3-27 Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule 874
24.3-28 Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule 875

25.3-1 RMT Architecture 931
25.3-2 Format of Pulse Code in RAM 932

26.4-1 PARLIO Architecture 959
26.5-1 PARLIO Clock Generation 960
26.5-2 Positive Waveform 962
26.5-3 Negative Waveform 962
26.5-4 Level Enable Mode for RX Unit 963
26.5-5 Sub-Modes of Pulse Enable Mode for RX Unit 964
26.5-6 Software Enable Mode for RX Unit 964

27.3-1 Touch Sensor System 987
27.3-2 Touch Sensor Internal Structure 989
27.4-1 Touch FSM Structure and Mechanism 990
27.4-2 TOUCH_OUT Signal Preprocessing 991
27.4-3 Timing Diagram of Scan Mode 994
27.4-4 Touch Sensor Frequency Hopping Sequence Timing 995
27.4-5 Sensing Area 998

28.1-1 SAR ADC digital controller block diagram 1030
28.4-1 ADC_SAR1_PATT_TAB1_REG contains patterns 0–3 1033
28.4-2 ADC_SAR1_PATT_TAB2_REG contains patterns 4–7 1033
28.4-3 ADC_SAR1_PATT_TAB3_REG contains patterns 8–11 1033
28.4-4 ADC_SAR1_PATT_TAB4_REG contains patterns 12–15 1033

28.4-5 Pattern layout (6-bit) 1034
28.5-1 Continuous sampling with channel switching 1036
28.5-2 Continuous sampling with PATT_TYPE = 1 (channel holds) 1037
28.5-3 Non-continuous mode, software-triggered sampling 1037

29.3-1 Analog Voltage Comparator Architecture 1062
29.4-1 Analog Voltage Comparator Full Scan Mode Timing Diagram 1065
29.4-2 Analog Voltage Comparator Clock Diagram 1066

Part I

Microprocessor and Master

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

Chapter 1

RISC-V Trace Encoder (TRACE)

The high-performance CPU (HP CPU) of ESP32-S31 supports the instruction trace interface through the trace encoder. The trace encoder connects to HP CPU's instruction trace interface, compresses the information into smaller packets, and then stores the packets into system memory (see Chapter 3 *System and Memory [to be added later]*).

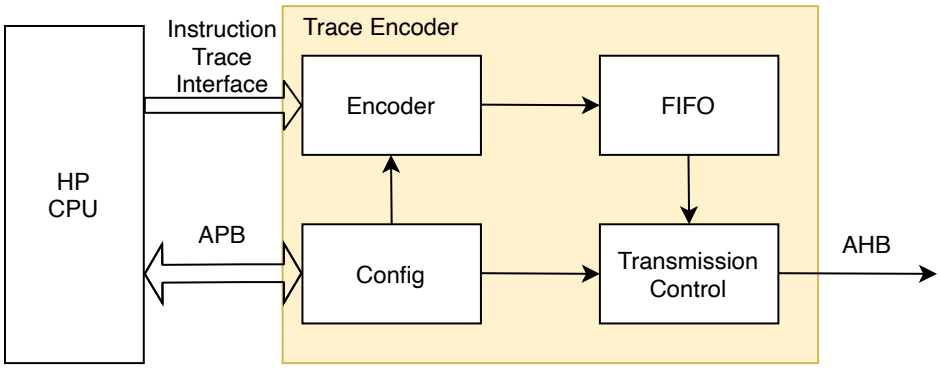


Figure 1.0-1. Trace Encoder Overview

1.1 Terminology

To better illustrate the functions of the RISC-V Trace Encoder, the following terms are used in this chapter.

hart	RISC-V hardware thread
branch	an instruction which conditionally changes the execution flow
uninferable discontinuity (updiscon)	a program counter change that can not be inferred from the program binary alone
delta	a change in the program counter that is other than the difference between two instructions placed consecutively in memory
trap	the transfer of control to a trap handler caused by either an exception or an interrupt
qualification	an instruction that meets the filtering criteria passes the qualification, and will be traced
te_inst	the name of the packet type emitted by the encoder
retire	the final stage of executing an instruction, when the machine state is updated
EPC	exception program counter

1.2 Introduction

In complex systems, understanding program execution flow is not straightforward. This may be due to a number of factors, for example, interactions with other cores, peripherals, real-time events, poor implementations, or some combination of all of the above.

It is hard to use a debugger to monitor the program execution flow of a running system in real time, as this is intrusive and might affect the running state. However, it is important to provide the visibility of program execution.

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

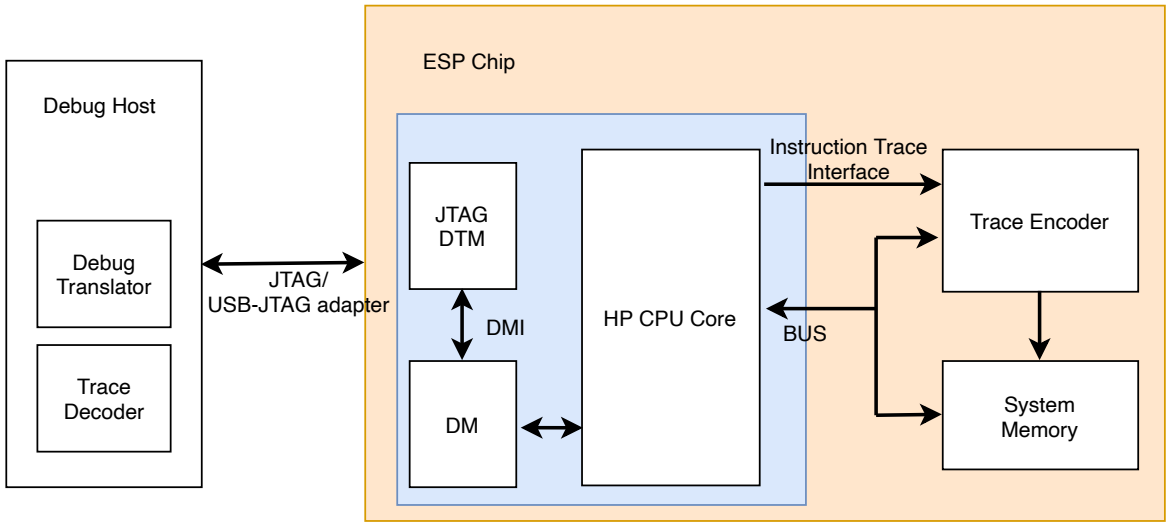


Figure 1.2-1. Trace Flow Overview

Figure 1.2-1 shows the instruction delta trace flow. Each HP core has a private encoder that can work independently, and the trace flow of each core is the same:

- The HP CPU core provides an instruction trace interface that outputs information about instructions executed by the HP CPU. Such information includes instruction address, instruction type, etc. For more details about ESP32-S31 HP CPU's instruction trace interface, please refer to Chapter 2 *High-Performance CPU [to be added later]*.
- The trace encoder collects the relevant instruction trace information from HP CPU's instruction trace interface, compresses the information into lower bandwidth packets, and then stores the packets in system memory.
- The debugger (debug host) can dump the trace packets from the system memory via JTAG or USB Serial/JTAG, and use a decoder to decompress and reconstruct the program execution flow. The trace decoder, usually software on an external PC, takes in the trace packets and reconstructs the program instruction flow with the program binary that runs on the originating hart. This decoding step can be done offline or in real time while the hart is executing.

1.3 Features

- Compatible with Efficient Trace for RISC-V Version 2.0. See Table 1.3-1 for the implemented parameters
- Support for delta address mode and full address mode
- Support for a filter unit
- Support for notifying an instruction address via debug trigger or filter unit
- Support for the following sideband signals to control trace data flow:
  - Support for debugging trigger to start or end encoder
  - When the hart is halted, the encoder can report the last packet and then stop
  - When the hart is reset, the encoder can report the last packet and then stop
  - Support for stalling the hart when FIFO is almost full
- Storing trace data packets within any valid address range
- Configurable synchronization modes:
  - Synchronization counter counts by packet
  - Synchronization counter counts by cycle
  - Synchronization counter can be disabled
- Trace lost status to indicate packet loss
- Automatic restart after packet loss
- Memory writing in the loop or non-loop mode
- Support two interrupts:
  - Triggered when the packet size exceeds the configured memory space
  - Triggered when a packet is lost
- FIFO (128 × 8 bits) to buffer packets
- AHB burst transmission with configurable burst length

Table 1.3-1. Trace Encoder Parameters

Parameter Name	Value	Description
arch_p	0	Initial version
bpred_size_p	0	Branch prediction mode is not supported
cache_size_p	0	Jump target cache mode is not supported
call_counter_size_p	0	Implicit return mode is not supported
ctype_width_p	0	Width of the ctype bus
context_width_p	0	Width of context bus
ecause_width_p	6	Width of exception cause
ecause_choice_p	0	Multiple choice is not supported
f0s_width_p	0	Format 0 packets are not supported
filter_context_p	0	Filtering on context is not supported

Parameter Name	Value	Description
filter_excint_p	1	Filtering on exception cause or interrupt is supported
filter_privilege_p	1	Filtering on privilege is supported
filter_tval_p	1	Filtering on trap value is supported
iaddress_lsb_p	1	Compressed instructions are supported
iaddress_width_p	32	The instruction bus is 32-bit
iretire_width_p	1	Width of the iretire bus
ilastsize_width_p	0	Width of the ilastsize
itype_width_p	3	Width of the itype bus
nocontext_p	1	Exclude context from te_inst packets
notime_p	1	Exclude time from te_inst packets
privilege_width_p	2	Machine, supervisor, and user mode are supported
retires_p	1	Maximum number of instructions that can be retired per block
return_stack_size_p	0	Implicit return mode is not supported
sijump_p	0	Sequentially inferable jump mode is not supported
taken_branches_p	1	Only one instruction retired per cycle
impdef_width_p	0	Not implemented

For detailed descriptions of the above parameters, please refer to the Efficient Trace for RISC-V Version 2.0 > Chapter Parameters and Discovery.

1.4 Architectural Overview

This chapter mainly introduces the implementation details of ESP32-S31's trace encoder.

As shown in Figure 1.0-1, the trace encoder contains an encoder, a FIFO, a register configuration module, and a transmission control module.

The encoder receives the HP CPU's instruction information via the instruction trace interface, compresses it into different packets, and writes it to the internal FIFO.

The transmission control module writes the data in the FIFO to the internal SRAM through the AHB bus.

The FIFO is 128 deep and 8-bit wide. When the memory bandwidth is insufficient, the FIFO may overflow, and packet loss occurs. If a packet is lost, the encoder will send a packet to tell that a packet is lost, stop working until the FIFO is empty, and also trigger an interrupt meanwhile if the interrupt is enabled.

1.5 Functional Description

1.5.1 Synchronization

In order to make the trace robust there must be regular synchronization points within the trace. Synchronization is accomplished by sending a full valued instruction address. When the synchronization counter value reaches the value of the TRACE_RESYNC_PROLONGED field of the TRACE_RESYNC_PROLONGED_REG register, the encoder will send a synchronization packet (format 3 subformat 0, see Section 1.6.3.1).

The synchronization counter is configured via TRACE_RESYNC_MODE:

- 0: Disable the synchronization counter
- 1: Invalid. No effect
- 2: Synchronization counter counts by packet
- 3: Synchronization counter counts by cycle

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

1.5.2 Address Mode

ESP32-S31 supports two address modes: delta address mode and full address mode.

- Delta address mode (default): In delta address mode, addresses are encoded as the difference between the actual address of the current instruction and the actual address of the instruction reported in the previous packet that contained an address. This differential encoding requires fewer bits than the full address, and thus results in more efficient trace compression.
- Full address mode: In full address mode, all addresses in the trace are encoded as absolute addresses instead of in differential form. This kind of encoding is always less efficient, but it can be a useful debugging aid for software decoder developers. This mode is enabled by setting TRACE_FULL_ADDRESS.

1.5.3 Optional Sideband Signals

The Efficient Trace for RISC-V Version 2.0 > Chapter Optional Sideband Signals provides some optional sideband signals for encoder control. ESP32-S31 has implemented the following functions:

Table 1.5-1. Optional sideband signals

Signal	Function
trigger[2:0]	• A pulse on bit 0 will cause the encoder to start tracing, and continue until further notice, subject to other filtering criteria also being met. • A pulse on bit 1 will cause the encoder to stop tracing until further notice. • A pulse on bit 2 will cause the encoder to report a specific address. This function is enabled by setting TRACE_DM_TRIGGER_ENA. The trigger signal is asserted upon a trigger match. For example, if the action field of mcontrol is 2, while the trigger matched it will assert trigger bit0; if the action is 3, while the trigger matched it will assert trigger bit1. For details, please refer to Table 1.8-1.
halted	Hart is halted. Upon assertion, the encoder will output a packet to report the address of the last instruction retired before halting, followed by a support packet to indicate that tracing has stopped. Upon deassertion, the encoder will start tracing again, commencing with a synchronization packet. This function is enabled by setting TRACE_HALT_ENA.

Signal	Function
reset	Hart is in reset. Behavior is as described above for halted.
stall	Stall request to hart. Some applications may require lossless trace, which can be achieved by using this signal to stall the hart if the trace encoder is unable to output a trace packet (internal FIFO almost full). This function is enabled by setting TRACE_STALL_ENA.

1.5.4 Filtering

Filtering provides a mechanism to control the criteria for encoder to produce a trace. For example, it may be desirable to trace:

- When the instruction address is within a particular range
- Starting from one instruction address and continue until a second instruction address
- For one or more specified privilege levels
- Exception and/or interrupt handlers for specified exception causes

The filtering unit has a filter and a comparator unit.

Each comparator unit is actually a pair of comparators (Primary and Secondary, or P, S) allowing a bounded range to be matched with a single unit if required, and offers:

- Input selected from iaddr and tval
- A range of arithmetic options ($<$, $>$, $=$, $!=$, etc.) independently selectable for each comparator
- Secondary match value may be used as a mask for the primary comparator
- The two comparators can be combined in several ways: P, $P\&\&S$, $!(P\&\&S)$, latch (set on P clear on S)
- Each comparator can also be used to explicitly report a particular instruction address

Each filter can specify filtering against instruction from the hart, and offers:

- One run-time selectable comparator unit to match
- Multiple choice selection for priv and cause inputs
- Select matching for interrupt

1.5.5 Anchor Tag

Since the length of data packets is variable, in order to identify boundaries between data packets when packed packets are written to memory, the ESP32-S31 encoder inserts zero bytes between data packets:

- The maximum packet length is 13 bytes, so a sequence of at least 14 zero bytes cannot occur within a normal packet. Therefore, the first non-zero byte seen after a sequence of at least 14 zero bytes must be the first byte of a packet.
- Every time when 128 packets are transmitted, the encoder writes 14 zero bytes to the memory partition boundary as anchor tags.

1.5.6 Memory Writing Mode

When writing the system 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 system memory or not by configuring the memory writing mode:

- Loop mode: When the size of the trace packets exceeds the capacity of the system memory (namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG), the system 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 system memory, the system memory is not wrapped around. The encoder stops at TRACE_MEM_END_ADDR_REG, and old data will be retained.

1.5.7 Automatic Restart

When packets are lost due to FIFO overflow, the encoder will stop working and need to be resumed by software manually or restarted by hardware automatically. If the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG is set, once the FIFO is empty, the encoder can automatically be restarted.

If the automatic restart feature is enabled, the encoder will be restarted in any case. Therefore, to disable the encoder, the automatic restart feature must be disabled first by clearing the TRACE_RESTART_ENA bit of the TRACE_TRIGGER_REG register.

1.6 Encoder Output Packets

This section mainly introduces ESP32-S31 trace encoder output packet format.

A packet includes header, index and payload. Header, index and payload are transmitted sequentially in bit stream form, from the fields listed at the top of tables below to the fields listed at the bottom. If a field consists of multiple bits, then the least significant bit is transmitted first.

1.6.1 Header

Header is 1-byte long. The format of header is shown in Table 1.6-1.

Table 1.6-1. Header Format

Field	Bits	Description	Value
length	5	Length of whole packet	4 ~ 13
flow	2	Reserved	0
timestamp	1	Reserved	0

1.6.2 Index

Index has 2 bytes. The format of index is shown in Table 1.6-2.

Table 1.6-2. Index Format

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

1.6.3 Payload

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

1.6.3.1 Format 3 Packets

Format 3 packets are used for synchronization, and report supporting information. There are 4 subformats defined in the specification. ESP32-S31 only supports 3 of them.

Format 3 Subformat 0 - Synchronization

This packet contains all the information the decoder needs to fully identify an instruction. It is sent for the first traced instruction (unless that instruction also happens to be the first one in an exception handler), and when synchronization has been scheduled by expiry of the synchronization timer. The payload length is 5 bytes.

Table 1.6-3. Packet format 3 subformat 0

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	00 (start): Start of tracing, or resync
branch	1	Set to 0 if the address points to a branch instruction, and the branch was taken. Set to 1 if the instruction is not a branch or if the branch is not taken.
privilege	2	The privilege level of the reported instruction
address	31	Full instruction address. The address must be left shifted 1 bit in order to recreate the original byte address.
sign_extend	2	Reserved

Format 3 Subformat 1 - Exception

This packet also contains all the information the decoder needs to fully identify an instruction. It is sent following an exception or interrupt, and includes the cause, the 'trap value' (for exceptions), and the address of the trap handler or of the exception itself. The length is variable: if the interrupt field is 1, the tvalepc field is omitted and the length is 6 bytes, otherwise the length is 10 bytes.

Table 1.6-4. Packet format 3 subformat 1

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	01 (exception): Exception cause and trap handler address

Field name	Bits	Description
branch	1	Set to 0 if the address points to a branch instruction, and the branch was taken. Set to 1 if the instruction is not a branch or if the branch is not taken.
privilege	2	The privilege level of the reported instruction
ecause	6	Exception cause
interrupt	1	Interrupt
theaddr	1	When set to 1, the address field points to the trap handler address. When set to 0, the address field points to the EPC (exception program counter) for an exception at the target for an updiscon, and is undefined for other exceptions and interrupts.
address	31	Full instruction address. The value of this field must be left shifted 1 bit in order to recreate the original byte address.
tvalepc	32	Value from appropriate utval/stval/mtval CSR (control/status register). Omitted if the interrupt is 1
sign_extend	2	Reserved

Format 3 Subformat 3 - Support

This packet provides supporting information to aid the decoder. It is issued when the trace is ended, filter match is failed, or a packet is lost. The length is 2 bytes.

Table 1.6-5. Packet format 3 subformat 3

Field name	Bits	Description
format	2	11 (sync): Synchronization
subformat	2	11 (support): Supporting information for the decoder
enable	1	Indicates if the encoder is enabled
encoder_mode	1	Always 0. Only supported by branch trace
qual_status	2	Indicates qualification status: • 00 (no_change): No change to filter qualification • 01 (ended_rep): Qualification ended, preceding instruction sent explicitly to indicate last qualification instruction • 10 (trace lost): One or more packets lost • 11 (ended_upd): Qualification ended, preceding te_inst would have been sent anyway due to an updiscon, even if wasn't the last qualified instruction

Field name	Bits	Description
ioptions	6	Indicates optional modes: • sequentially inferred jumps (Bits 5): When set to 1, the targets of sequentially inferable jumps will not be reported. (This feature is not implemented, so bit 5 should always be 0) • branch prediction (Bit 4): When set to 1, the branch prediction is enabled. (This feature is not implemented, so bit 4 should always be 0) • jump target cache (Bit 3): When set to 1, the jump target cache is enabled. (This feature is not implemented, so bit 3 should always be 0) • full address (Bit 2): – 0: delta address mode – 1: full address mode • implicit exception (Bit 1): Exclude address from format 3 subformat 1 packet if trap vector can be determined from ecause. (This feature is not implemented, so bit 1 should always be 0) • implicit return (Bit 0): When set to 1, function return addresses will not be reported. (This feature is not implemented, bit 0 should always be 0)
sign_extend	2	Reserved

1.6.3.2 Format 2 Packets

This packet contains only an instruction address, and is used when the address of an instruction must be reported (for example if the instruction is the target for an updiscon, or is the last instruction before exception), and there is no reported branch information.

The length is variable if delta address mode is enabled, ranging from 5 bytes to 8 bytes. The address field can be 8/16/24/32 bits, inferred from the packet length. For example, if the header length is n bytes, the address bits are $n-4$ bytes. A sign bit is required to extend the address field to 32 bits.

Table 1.6-6. Packet format 2

Field name	Bits	Description
format	2	10 (addr-only): No branch information
notify	1	If the value of this bit is different from the MSB of address, it indicates that this packet is reporting an instruction that is not the target of an uninferable discontinuity because a notification was requested via trigger[2] or a filter matched.
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.
zero_extend	4	Reserved
address	Variable	Delta instruction address. These bits can be 8/16/24/32.

1.6.3.3 Format 1 Packets

This packet includes branch information and is used in either of the following cases:

- When branch information must be reported (for example, when the branch map is full)
- When an instruction address must be reported after at least one branch has occurred since the previous packet.

The packet length is variable if delta address mode is enabled, ranging from 5 bytes to 12 bytes. The address field can be 8/16/24/32 bits, inferred from the packet length. For example:

- The header length is n bytes
- The value of the format field is 1
- The value of the branches field is:
  - 0: the packet is format 1 without address
  - 1: branch_map is 1 bit wide, zero_ext is 6 bits wide, and address is $n - 5$ bits wide
  - 2 ~ 3: branch_map is 3 bits wide, zero_ext is 4 bits wide, and address is $n - 5$ bits wide
  - 4 ~ 7: branch_map is 7 bits wide, zero_ext is 0 bits wide, and address is $n - 5$ bits wide
  - 8 ~ 15: branch_map is 15 bits wide, zero_ext is 0 bits wide, and address is $n - 6$ bits wide
  - 16 ~ 31: branch_map is 31 bits wide, zero_ext is 0 bits wide, and address is $n - 8$ bits wide

Format 1 - address, branch_map

The length is variable.

Table 1.6-7. Packet format 1 with address

Field name	Bits	Description
format	2	01: Includes branch information
branches	5	Number of valid bits branch_map. The number of bits of branch_map is determined as follows: • 0: (cannot occur for this format) • 1: 1 bit • 2 ~ 3: 3 bits • 4 ~ 7: 7 bits • 8 ~ 15: 15 bits • 16 ~ 31: 31 bits For example if branches = 12, branch_map is 15-bit long, and the 12 LSBs are valid.
branch_map	Variable	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken The field Bits is variable and determined by the branches field.
notify	1	If the value of this bit is different from the MSB of address, it indicates that this packet is reporting an instruction that is not the target of an uninferable discontinuity because a notification was requested via trigger[2] or a filter matched.

Field name	Bits	Description
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.
zero_ext	Variable	The length of this field is determined by branches as follows: • 1: 6 bits • 2 ~ 3: 4 bits • 4 ~ 31: 0 bits
address	Variable	Delta instruction address. These bits can be 8/16/24/32

Format 1 - no address, branch_map

The length is 5 bytes.

Table 1.6-8. Packet format 1 without address

Field name	Bits	Description
format	2	01: includes branch information
branches	5	Number of valid bits in branch_map. Can only be 0, indicating the length of branch_map is 31 bits.
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

1.7 Interrupt

ESP32-S31's CORE*n*_TRACE (i.e., the trace encoder of core 0 or core 1) can generate the CORE*n*_TRACE_INTR interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from CORE*n*_TRACE that can generate the above interrupt signal. The interrupt sources from CORE*n*_TRACE are listed with their trigger conditions and the resulted interrupt signals in Table 1.7-1.

Table 1.7-1. CORE*n*_TRACE's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TRACE_MEM_FULL_INTR	Triggered when the packet size exceeds the capacity of the system memory, namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG. If necessary, this interrupt can be enabled to notify the HP CPU for processing, such as applying for a new memory space again	CORE*n*_TRACE_INTR

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TRACE_FIFO_OVERFLOW_INTR	Triggered when the internal FIFO overflows and one or more packets have been lost.	CORE*n*_TRACE_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

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 1.9 *Register Summary*.

1.8 Programming Procedures

1.8.1 Encoder Option Configuration

The encoder has some optional configuration options.

- Full address mode
  - The encoder uses delta address mode by default. Users can set TRACE_FULL_ADDRESS to enable full address mode for debugging.
- Restart after packet loss
  - Once the trace packet is lost due to FIFO overflow, the encoder automatically stops. If TRACE_RESTART_ENA is set, the encoder will automatically restart once the FIFO is empty.
- Stall hart while FIFO is almost full
  - When the FIFO is almost full with TRACE_STALL_ENA set, the encoder asserts a stall request to stall the hart until the request is deasserted.
- Stop tracing when the hart is halted

  When the hart halts with TRACE_HALT_ENA set, the encoder outputs an address packet for the last instruction executed before halting, followed by a support packet indicating that tracing has stopped. Upon the deassertion of the halted signal, the encoder will start tracing again, commencing with a synchronization packet.
- Stop tracing when the hart is in reset

  The encoder will be reset separately, if TRACE_RESET_ENA is set. When the hart is in reset, the encoder will output a packet to report the address of the last instruction before reset, followed by a support packet to indicate that tracing has stopped. Once the hart exits from reset, the encoder will start tracing again, commencing with a synchronization packet.
- Using trigger outputs from the Debug Module

  The debug module of the RISC-V hart has a trigger unit. This defines a match control register (mcontrol) containing a 4-bit action field, and reserves codes 2 ~ 4 of this field for trace use. ESP32-S31 has implemented this feature. The values of the action field are listed in Table 1.8-1.

Table 1.8-1. Debug module trigger support (the action field of the mcontrol register)

Value	Description
2	Trace-on: start trace
3	Trace-off: end trace
4	Trace-notify: notify the encoder to report an address

For details about the debug module trigger unit, please refer to Chapter 2 *High-Performance CPU [to be added later]*.

1.8.2 Filter Configuration

The filter unit described in Section 1.5.4 can be configured as follows:

1. Select one of the following match modes:
   - Set TRACE_MATCH_PRIVILEGE to enable privilege match mode:
     - Configure TRACE_MATCH_CHOICE_PRIVILEGE to specify the privilege level.
   - Set TRACE_MATCH_ECAUSE to enable ecause match mode:
     - Configure TRACE_MATCH_CHOICE_ECAUSE to specify the ecause value
   - Set TRACE_MATCH_INTERRUPT to enable interrupt match mode:
     - Configure TRACE_MATCH_VALUE_INTERRUPT to specify interrupt level
   - Set TRACE_MATCH_COMP to enable comparator match mode:
     - Configure the primary comparator:
       * Configure TRACE_P_INPUT to choose the input
       * Configure TRACE_P_FUNCTION to select the comparator function
       * Configure TRACE_FILTER_P_COMPARATOR_MATCH_REG to define match value
     - Configure the secondary comparator if needed:
       * Configure TRACE_S_INPUT to choose the input
       * Configure TRACE_S_FUNCTION to select the comparator function
       * Configure TRACE_FILTER_S_COMPARATOR_MATCH_REG to define match value
     - Choose which comparator to match via TRACE_MATCH_MODE:
       * 0: only the primary comparator matches, and the secondary comparator has no effect
       * 1: both the primary comparator and the secondary comparator match ($P\&\&S$)
       * 2: the primary comparator and secondary comparator do not both match $!(P\&\&S)$
       * 3: start filtering when primary matches up until secondary matches

   If TRACE_P_NOTIFY or TRACE_S_NOTIFY is set, the comparator will report the matched address as a notification, the instructions will not be filtered. For example:

  - If the TRACE_MATCH_MODE is 0, and TRACE_P_NOTIFY is set, then all instructions will be traced, and the matched primary comparator will be notified.
  - If the TRACE_MATCH_MODE is 0, and TRACE_S_NOTIFY is set, the instruction will be filtered by the primary comparator, and the instruction that matches the secondary comparator will be notified.

2. Set TRACE_FILTER_EN to enable filter unit.

1.8.3 Enable Encoder

- Configure the memory address space for storing trace packets via TRACE_MEM_START_ADDR_REG and TRACE_MEM_END_ADDR_REG. The encoder can access system memory TCM and PSRAM:
  - TCM: 0x2F000000 ~ 0x2F07FFFF
  - PSRAM: 0x50000000 ~ 0x53FFFFFF
- Update the value of TRACE_MEM_CURRENT_ADDR_REG to the value of TRACE_MEM_START_ADDR_REG by setting TRACE_MEM_CURRENT_ADDR_UPDATE
- (Optional) Configure the memory writing mode via the TRACE_MEM_LOOP bit of TRACE_TRIGGER_REG
  - 0: Non-loop mode
  - 1: Loop mode (default)
- Configure the synchronization mode via the TRACE_RESYNC_MODE bit of TRACE_RESYNC_PROLONGED_REG
  - 0: Disable the synchronization counter (default)
  - 1: Invalid. No effect
  - 2: Synchronization counter counts by packet
  - 3: Synchronization counter counts by cycle
- (Optional) Configures the threshold for the synchronization counter (default value is 128) via TRACE_RESYNC_PROLONGED_REG
- (Optional) Enable Interrupt
  - Set the corresponding bit of TRACE_INTR_ENA_REG to enable the corresponding interrupt
  - Set the corresponding bit of TRACE_INTR_CLR_REG to clear the corresponding interrupt
  - Read TRACE_INTR_RAW_REG to know which interrupt occurs
- (Optional) Enable automatic restart by setting the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG. This function is enabled by default
- There are 2 ways to enable the trace encoder:
  - Setting the TRACE_TRIGGER_ON bit of TRACE_TRIGGER_REG
  - Through the Debug module Trace-on to start the encoder. When the trigger is matched and the action is 2, it will start the encoder. For trigger configurations, please refer to the RISC-V Debug Specification.

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

1.8.4 Disable Encoder

There are two ways to end output trace packets:

1. Set TRACE_TRIGGER_OFF of TRACE_TRIGGER_REG to end the encoder
2. Through the Debug module Trace-off to end the encoder. When the trigger is matched and the action is 3, it will end the encoder. For trigger configurations, please refer to the RISC-V Debug Specification.

Due to the internal FIFO, the packets are not written to the memory immediately after the end of the trace. It is recommended to query the TRACE_FIFO_STATUS_REG to confirm that the data is all written to memory.

Table 1.8-2. Trace Status

Field	Description
TRACE_FIFO_EMPTY	A value of 1 indicates that the FIFO is empty and all data has been written to memory.
TRACE_WORK_STATUS	Encoder's work status: • 0: idle state, the encoder is not started • 1: the encoder is normally operating and outputting output packets • 2: the encoder is not outputting packets, the hart is halted or in reset • 3: the encoder is not outputting packets, it occurs while a packet is lost, and is waiting for FIFO empty to restart.

1.8.5 Notify

Users may want to report a special address, even if it is not the target of an uninferable discontinuity. There are two ways to notify and report a special address:

1. The debug trigger unit matched, and the action is 4
2. The filter comparator matched

When a notification is requested, the encoder will report a format 2 or format 1 with an address packet, determined by the branch_map. If branch_map is 0, report format 2, otherwise report format 1. The notify field shows whether the packet is reporting a notification address. If it is the target of an uninferable discontinuity, even if it is a notification, the notify field will not be asserted, otherwise the decoder cannot reconstruct the instruction stream correctly.

1.8.6 Decode Data Packets

- Find the first address to decode
  - Read the TRACE_MEM_FULL_INTR_RAW bit of the TRACE_INTR_RAW_REG register to know if the system 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 1.6, 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

1.8.7 AHB Configuration

ESP32-S31 encoder supports the AHB bus. There are some optional configurations to control the transmission bandwidth of the encoder.

If there are many DMA masters, users can increase the bandwidth of Trace by setting AHB burst TRACE_AHB_CONFIG_REG.

- The TRACE_HBURST supports the following configurations:
  - 0x0: SINGLE
  - 0x1: INCR (length not defined)
  - 0x2: INCR4
  - 0x4: INCR8
  - Others: reserved

  Burst transfer means that once arbitration is successful, multiple words can be transferred continuously. For SINGLE, the length is 1; for INCR, the length is undefined; for INCR4, the length is 4; for INCR8, the length is 8. The longer the burst length, the better the bandwidth.
- When configured as INCR transfer, since INCR is a burst of indeterminate length, in order to avoid Trace occupying the bus for a long time, it will end the INCR transfer after the transfer length up to TRACE_MAX_INCR, and restart bus arbitration.

This is the configuration used to adjust the bandwidth. When the bandwidth is sufficient, it's not required to do the configuration, and users can use the default configuration.

1.8.8 Software Retention

The TRACE_WORK_STATUS represents the state of the encoder. When the encoder is not in the IDLE state, it must backup and restore the configuration when the chip enters sleep.

1.9 Register Summary

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

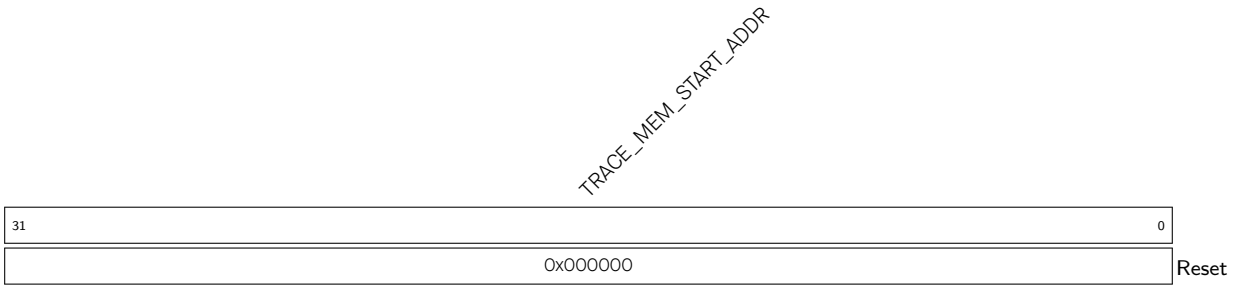
Name	Description	Address	Access
Trace memory configuration registers			
TRACE_MEM_START_ADDR_REG	Memory start address register	0x0000	R/W
TRACE_MEM_END_ADDR_REG	Memory end address register	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	Memory current address register	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	Memory address update register	0x000C	WT
Trace fifo status register			
TRACE_FIFO_STATUS_REG	FIFO status register	0x0010	RO
Trace interrupt configuration 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 register			
TRACE_TRIGGER_REG	Trace trigger register	0x0020	varies
TRACE_CONFIG_REG	Trace configuration register	0x0024	R/W
TRACE_FILTER_CONTROL_REG	Filter control register	0x0028	R/W
TRACE_FILTER_MATCH_CONTROL_REG	Filter match control register	0x002C	R/W
TRACE_FILTER_COMPARATOR_CONTROL_REG	Filter comparator control register	0x0030	R/W
TRACE_FILTER_P_COMPARATOR_MATCH_REG	Primary comparator match value register	0x0034	R/W
TRACE_FILTER_S_COMPARATOR_MATCH_REG	Secondary comparator match value register	0x0038	R/W
TRACE_RESYNC_PROLONGED_REG	Resynchronization configuration register	0x003C	R/W
TRACE_AHB_CONFIG_REG	AHB configuration register	0x0040	R/W
Clock Gate Configuration register			
TRACE_CLOCK_GATE_REG	Clock gating control register	0x0044	R/W
Version register			
TRACE_DATE_REG	Version control register	0x03FC	R/W

1.10 Registers

The addresses in this section are relative to RISC-V Trace Encoder base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

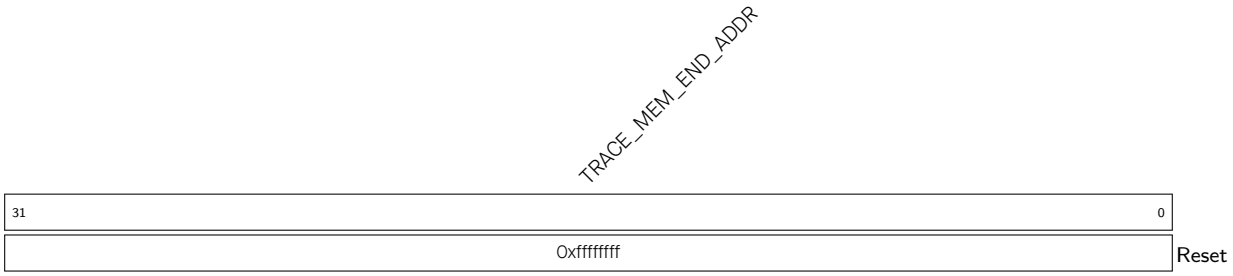
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 1.1. TRACE_MEM_START_ADDR_REG (0x0000)



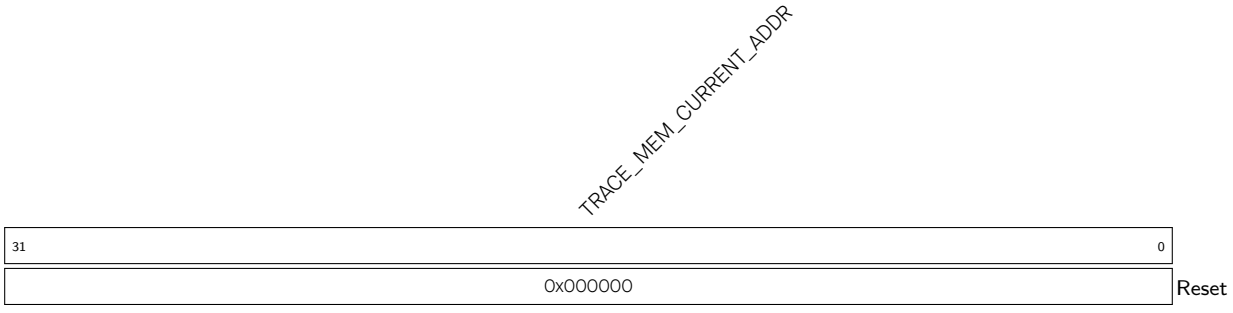
TRACE_MEM_START_ADDR Configures the start address of the trace memory. (R/W)

Register 1.2. TRACE_MEM_END_ADDR_REG (0x0004)



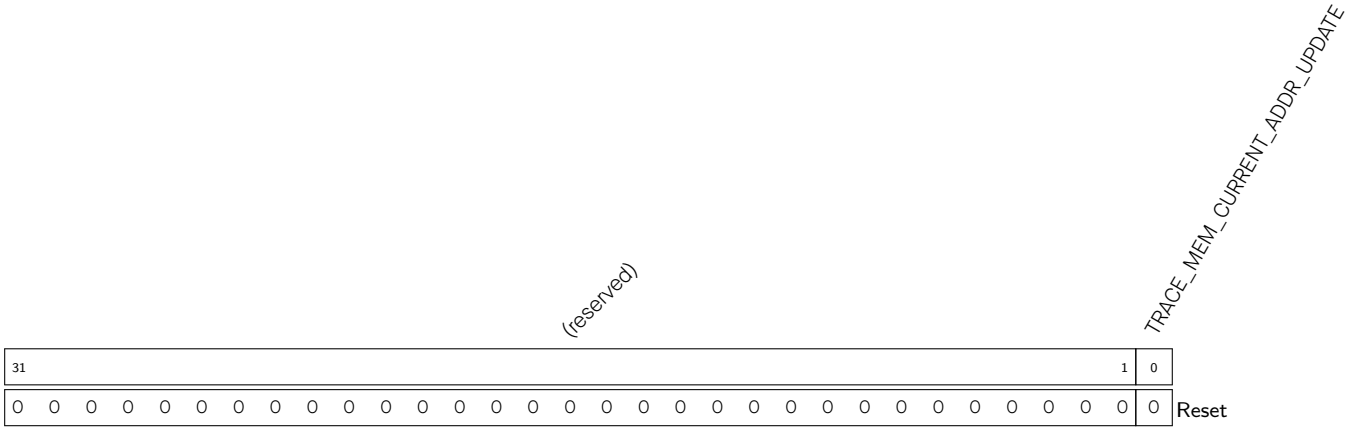
TRACE_MEM_END_ADDR Configures the end address of the trace memory. (R/W)

Register 1.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



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

Register 1.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)

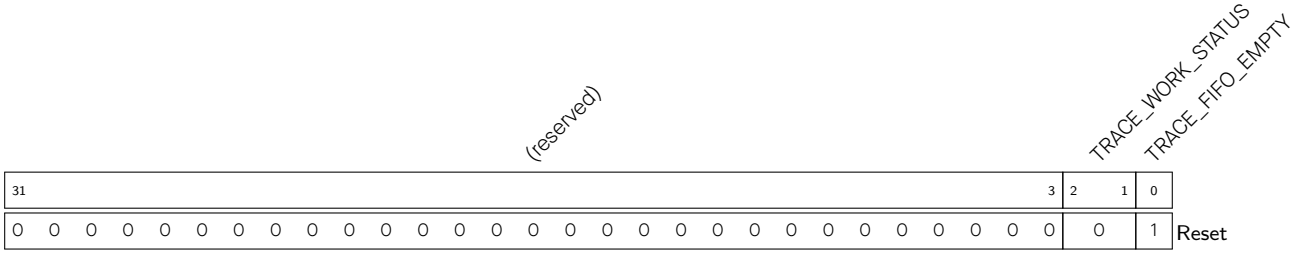


TRACE_MEM_CURRENT_ADDR_UPDATE Configures whether to update the value of TRACE_MEM_CURRENT_ADDR to TRACE_MEM_START_ADDR.

0: Not update

1: Update (WT)

Register 1.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY Represents whether the FIFO is empty.

0: Not empty

1: Empty (RO)

TRACE_WORK_STATUS Represents the state of the encoder.

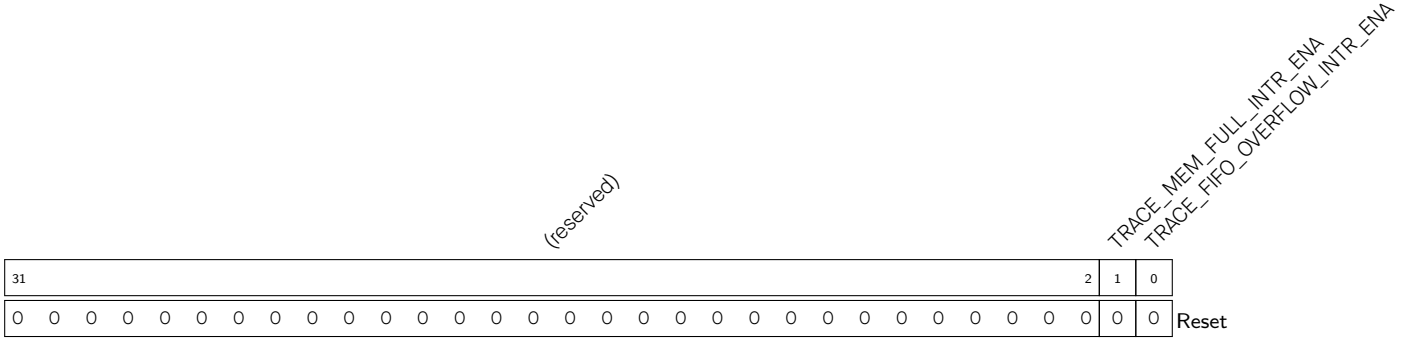
0: Idle state

1: Working state

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

3: Lost state (RO)

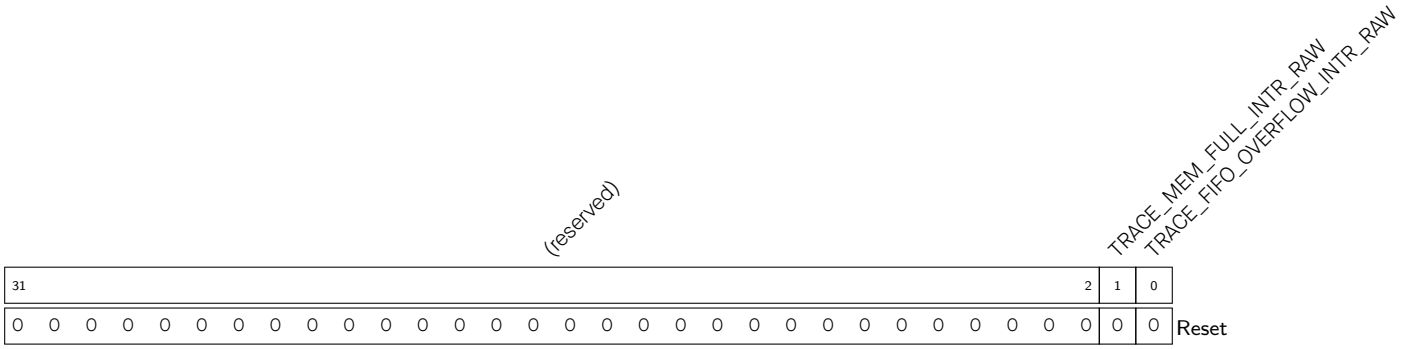
Register 1.6. TRACE_INTR_ENA_REG (0x0014)



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)

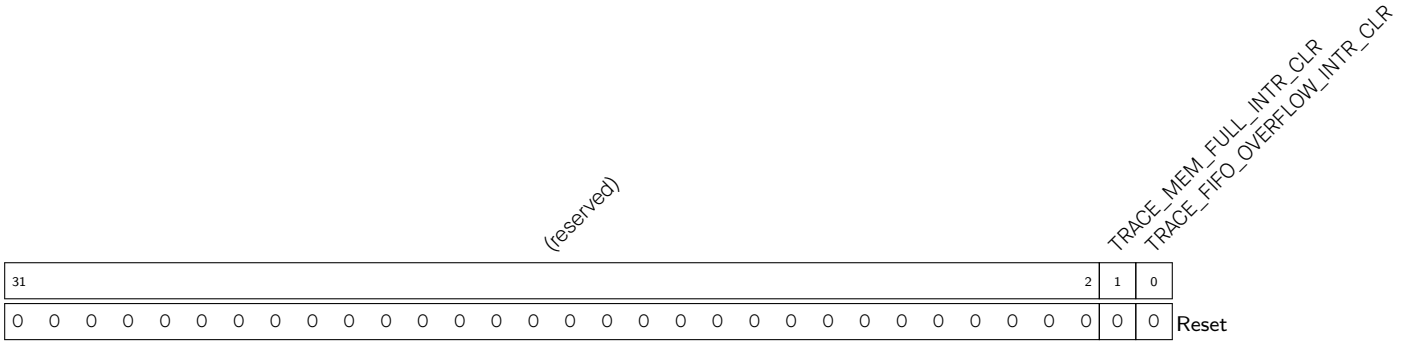
Register 1.7. TRACE_INTR_RAW_REG (0x0018)



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)

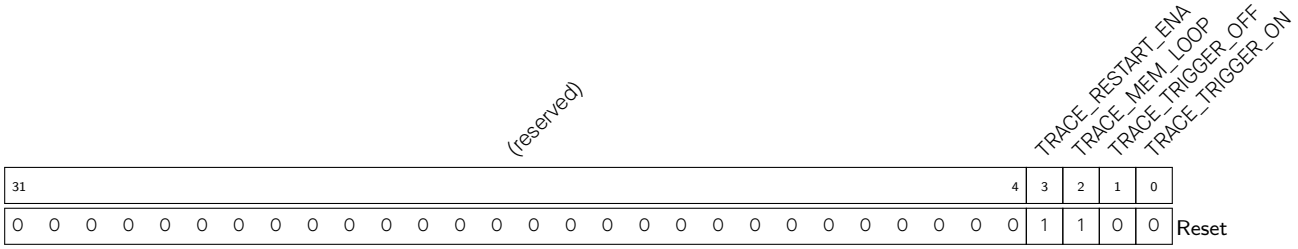
Register 1.8. TRACE_INTR_CLR_REG (0x001C)



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 1.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON Configures whether to enable the encoder.

0: Invalid. No effect

1: Enable (WT)

TRACE_TRIGGER_OFF Configures whether to disable the encoder.

0: Invalid. No effect

1: Disable (WT)

TRACE_MEM_LOOP Configures the memory writing mode.

0: Non-loop mode

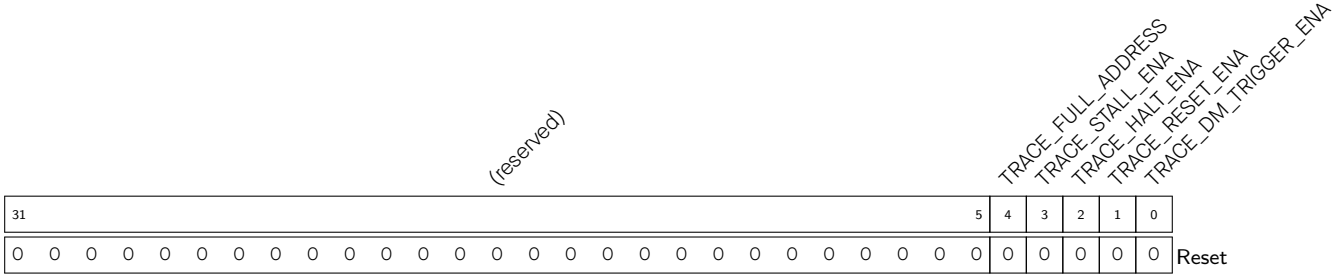
1: Loop mode (R/W)

TRACE_RESTART_ENA Configures whether to enable automatic restart.

0: Disable

1: Enable (R/W)

Register 1.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA Configures whether to enable the trigger signal.

0: Disable

1: Enable (R/W)

TRACE_RESET_ENA Configures whether to enable the reset signal.

0: Disable

1: Enable (R/W)

TRACE_HALT_ENA Configures whether to enable the halted signal.

0: Disable

1: Enable (R/W)

TRACE_STALL_ENA Configures whether to enable the stall signal.

0: Disable

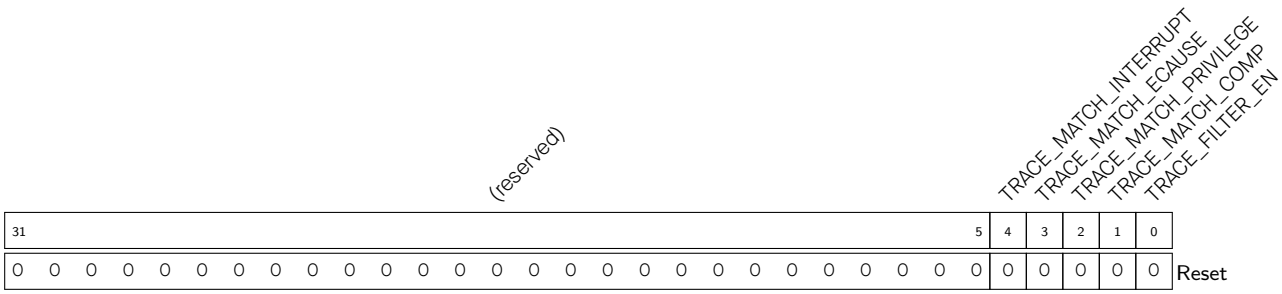
1: Enable (R/W)

TRACE_FULL_ADDRESS Configures the address mode.

0: Delta address mode

1: Full address mode (R/W)

Register 1.11. TRACE_FILTER_CONTROL_REG (0x0028)



TRACE_FILTER_EN Configures whether to enable filtering.

0: Disable

1: Enable (R/W)

TRACE_MATCH_COMP Configures whether to enable the comparator match mode.

0: Disable

1: Enable (R/W)

TRACE_MATCH_PRIVILEGE Configures whether to enable the privilege match mode.

0: Disable

1: Enable (R/W)

TRACE_MATCH_ECAUSE Configures whether to enable the ecause match mode.

0: Disable

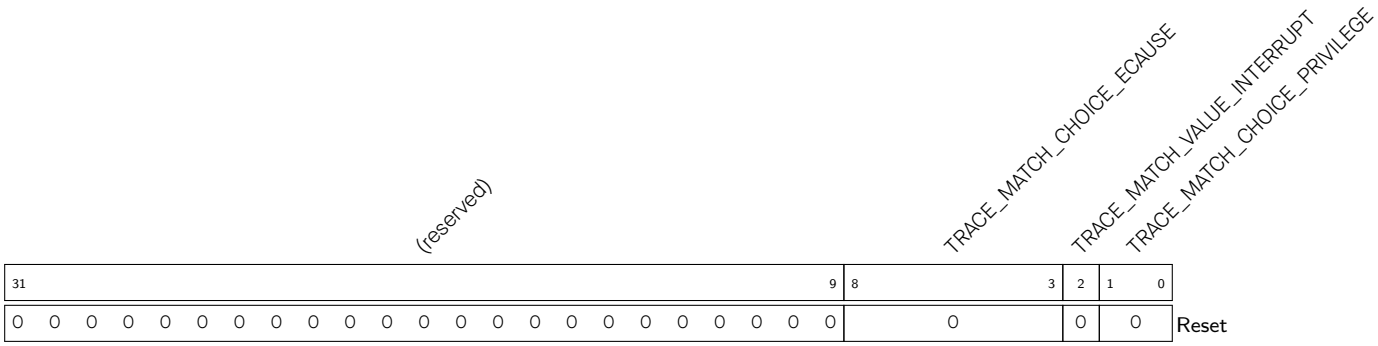
1: Enable (R/W)

TRACE_MATCH_INTERRUPT Configures whether to enable the interrupt match mode.

0: Disable

1: Enable (R/W)

Register 1.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)



TRACE_MATCH_CHOICE_PRIVILEGE Configures the privilege level for matching.

0: User mode

1: Supervisor mode

2: Reserved

3: Machine mode (R/W)

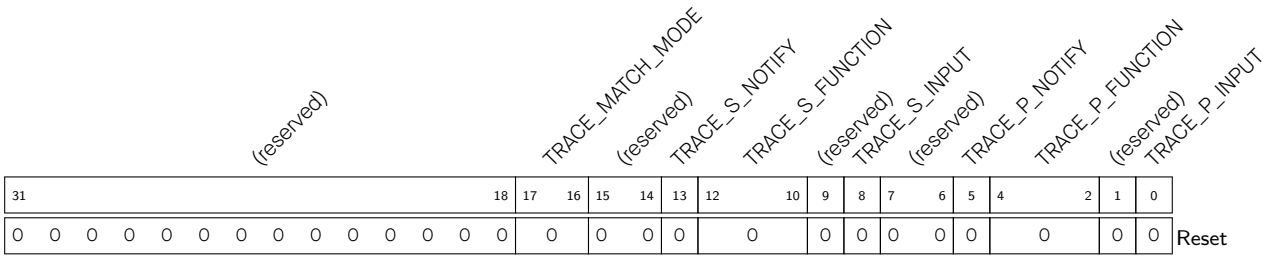
TRACE_MATCH_VALUE_INTERRUPT Configures the interrupt level for matching. Valid only when TRACE_MATCH_INTERRUPT is 1.

0: itype=1

1: itype=2 (R/W)

TRACE_MATCH_CHOICE_ECAUSE Configures the ecause code for matching. (R/W)

Register 1.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)



TRACE_P_INPUT Configures the input of the primary comparator for matching.

0: iaddr

1: tval (R/W)

TRACE_P_FUNCTION Configures the function for the primary comparator.

0: Equal

1: Not equal

2: Less than

3: Less than or equal

4: Greater than

5: Greater than or equal

Others: Always match (R/W)

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

0: Not report

1: Report (R/W)

TRACE_S_INPUT Configures the input of the secondary comparator for matching.

0: iaddr

1: tval (R/W)

TRACE_S_FUNCTION Configures the function for the secondary comparator.

0: Equal

1: Not equal

2: Less than

3: Less than or equal

4: Greater than

5: Greater than or equal

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

Others: Always match (R/W)

TRACE_S_NOTIFY Configures whether to explicitly report an instruction address matched against the secondary comparator.

0: Not report

1: Report (R/W)

Continued on the next page...

Register 1.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

Continued from the previous page...

TRACE_MATCH_MODE Configures the comparator match condition.

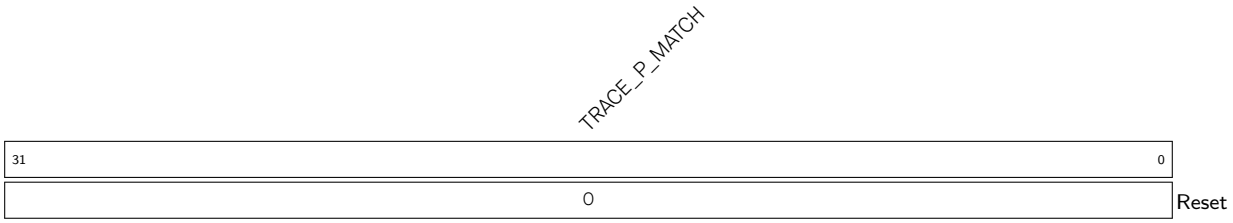
0: Only the primary comparator matches

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

2: The primary and secondary comparators do not both match !(P&&S)

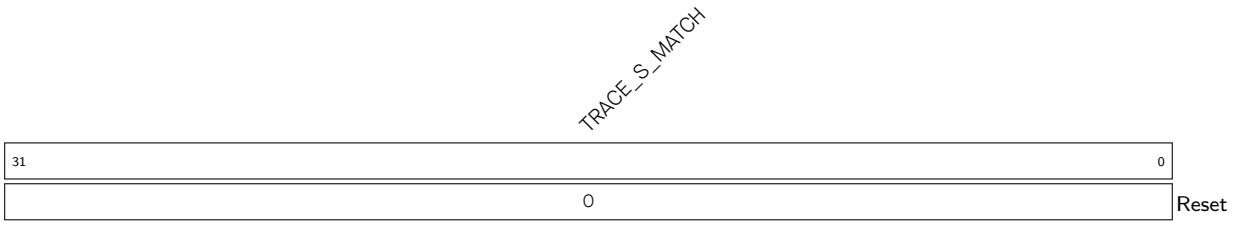
3: Start filtering when the primary comparator matches and stop filtering when the secondary comparator matches (R/W)

Register 1.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



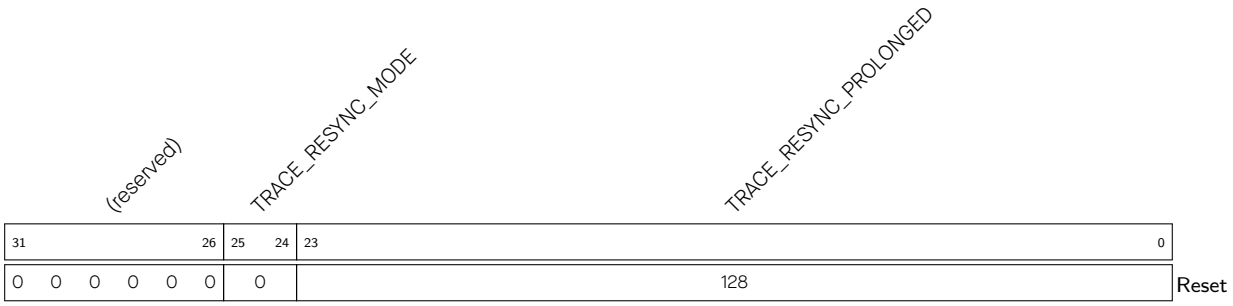
TRACE_P_MATCH Configures the match value for the primary comparator. (R/W)

Register 1.15. TRACE_FILTER_S_COMPARATOR_MATCH_REG (0x0038)



TRACE_S_MATCH Configures the match value for the secondary comparator. (R/W)

Register 1.16. TRACE_RESYNC_PROLONGED_REG (0x003C)



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

TRACE_RESYNC_MODE Configures the synchronization mode.

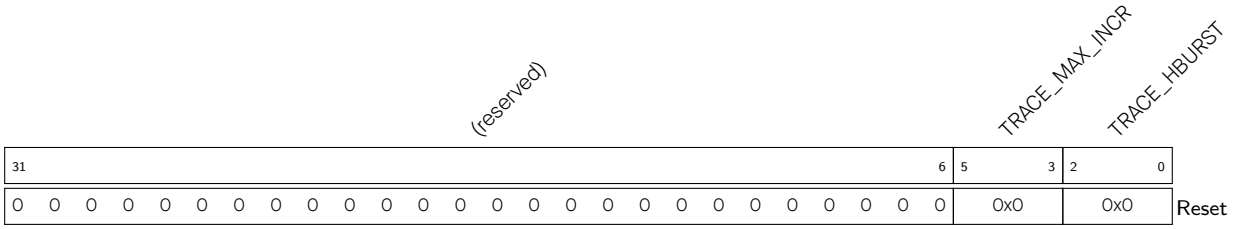
0: Disable the synchronization counter

1: Invalid. No effect

2: Synchronization counter counts by packet

3: Synchronization counter counts by cycle (R/W)

Register 1.17. TRACE_AHB_CONFIG_REG (0x0040)



TRACE_HBURST Configures the AHB burst mode.

0: SINGLE

1: INCR (length not defined)

2: INCR4

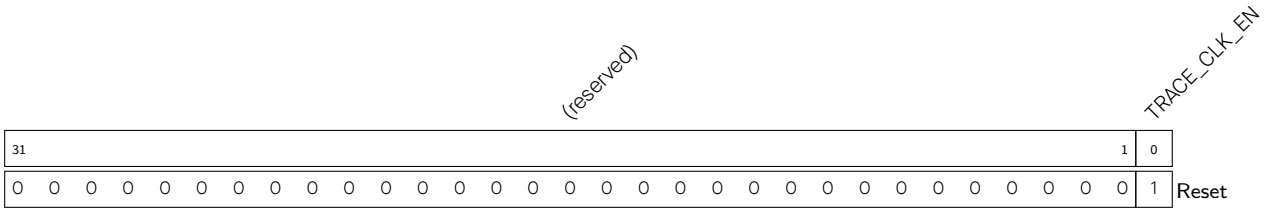
3: Invalid. No effect

4: INCR8

Others: Invalid. No effect (R/W)

TRACE_MAX_INCR Configures the maximum burst length for INCR mode. (R/W)

Register 1.18. TRACE_CLOCK_GATE_REG (0x0044)



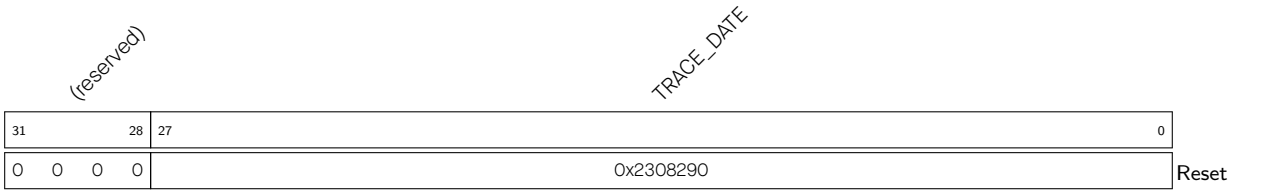
TRACE_CLK_EN Configures register clock gating.

0: Support clock only when the application writes registers to save power

1: Always force the clock on for registers

This field does not affect register access. (R/W)

Register 1.19. TRACE_DATE_REG (0x03FC)



TRACE_DATE Version control register. (R/W)

Chapter 2

Low-Power CPU

2.1 Overview

The ESP32-S31 Low-Power CPU (LP CPU) is a 32-bit processor based on RISC-V ISA and supports the integer (I), multiplication/division (M), atomic (A), and compressed (C) standard extensions. It features ultra-low power consumption and uses a 2-stage, in-order, and scalar pipeline. The LP CPU core complex has an interrupt controller (INTC), a debug module (DM), and system bus (SYS BUS) interfaces for memory and peripheral access.

The LP CPU is in sleep mode by default (see Section 2.10). It can remain powered on when the chip enters Deep-sleep mode (see Chapter 11 *Low-Power Management [to be added later]* for details) and can access most peripherals and memories (see Chapter 3 *System and Memory [to be added later]* for details). The LP CPU has two application scenarios:

- Power-insensitive scenario: When the High-Performance CPU (HP CPU) is active, the LP CPU can assist it with some speed- and efficiency-insensitive controls and computations.
- Power-sensitive scenario: When the HP CPU is in the power-down state to save power, the LP CPU can be woken up to handle some external wake-up events.

Figure 2.1-1 shows the resources accessible to the HP CPU and the LP CPU.

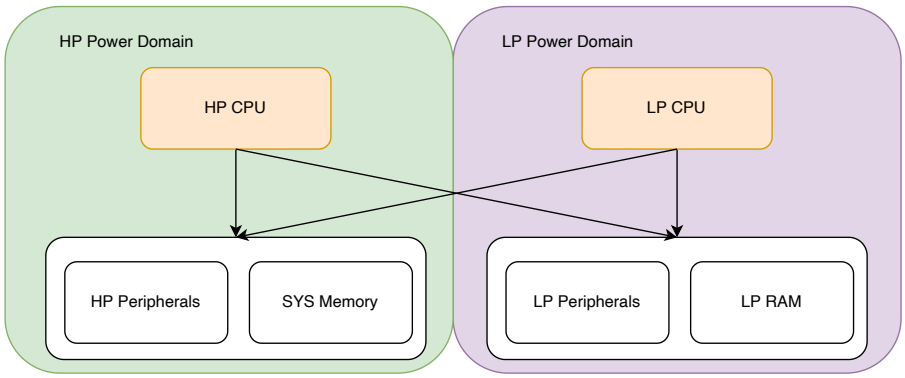


Figure 2.1-1. LP CPU Overview

In the figure above, the HP Power Domain and LP Power Domain can be powered on and off independently. When one domain is powered down, the CPU in the other domain cannot access the resources of that domain. In other words, only when the domain is powered on can its resources be accessed. For more information about power, please refer to Chapter 11 *Low-Power Management [to be added later]*.

2.2 Features

The LP CPU has the following features:

- Operating clock frequency up to 40 MHz
- 18 vector interrupts
- Debug module compliant with RISC-V External Debug Support Version 0.13 with external debugger support over an industry-standard JTAG/USB port
- Hardware trigger compliant with RISC-V External Debug Support Version 0.13 with up to 2 breakpoints/watchpoints
- Core performance metric events
- Waking up the HP CPU and sending interrupts to the HP CPU
- Access to HP memory and LP memory
- Access to the entire peripheral address space
- Standard physical memory protection (PMP) compatible with the RISC-V specification

2.3 Configuration and Status Registers (CSRs)

2.3.1 Register Summary

Below is a list of CSRs that are supported by the LP CPU. Except for the custom performance counter CSRs, all the implemented CSRs follow the standard mapping of bit fields as described in the RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. It must be noted that even among the standard CSRs, not all bit fields have been implemented, limited by the subset of features implemented in the CPU. For a detailed description of the subset of fields implemented under each of these CSRs, please refer to Section 2.3.2.

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

Name	Description	Address	Access
Machine Information CSR			
mhartid	Machine Hart ID	0xF14	RO
Machine Trap Setup CSRs			
mstatus	Machine Mode Status	0x300	R/W
misa [1]	Machine ISA	0x301	R/W
mie	Machine Interrupt Enable	0x304	R/W
mtvec [2]	Machine Trap Vector	0x305	R/W
Machine Trap Handling CSRs			
mscratch	Machine Scratch	0x340	R/W
mepc	Machine Trap Program Counter	0x341	R/W

[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.

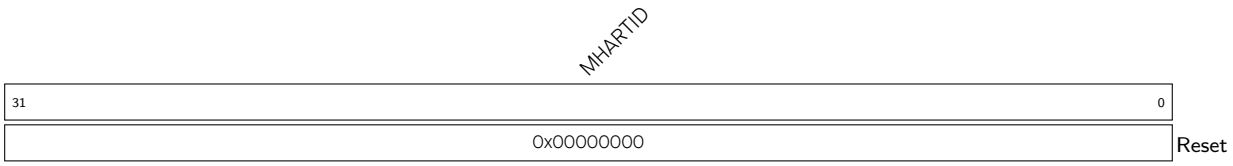
Name	Description	Address	Access
mcause [3]	Machine Trap Cause	0x342	R/W
mtval	Machine Trap Value	0x343	R/W
mip	Machine Interrupt Pending	0x344	R/W
Trigger Module CSRs (shared with Debug Mode)			
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
Debug Mode CSRs			
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W
Machine Counter/Timer CSRs			
mcycle	Machine Clock Cycle Counter	0xB00	R/W
minstret	Machine Retired Instruction Counter	0xB02	R/W
mhpmcounter*n*(*n*:3-12)	Machine Performance Monitor Counter	0xB00+*n*	R/W
mcycleh	The higher 32 bits of mcycle	0xB80	R/W
minstreth	The higher 32 bits of minstret	0xB82	R/W
mhpmcounter*n*h(*n*:3-12)	The higher 32 bits of mhpmcounter*n*(*n*:3-12)	0xB80+*n*	R/W
Machine Counter Setup CSR			
mcountinhibit	Machine Counter Control	0x320	R/W
Physical Memory Protection (PMP) CSR			
pmpcfg0	Physical Memory Protection Configuration	0x3A0	R/W
pmpcfg1	Physical Memory Protection Configuration	0x3A1	R/W
pmpcfg2	Physical Memory Protection Configuration	0x3A2	R/W
pmpcfg3	Physical Memory Protection Configuration	0x3A3	R/W
pmpaddr*n* (*n*: 0-15)	Physical Memory Protection Address Register	0x3B0+0x1**n*	R/W

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

2.3.2 Registers

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

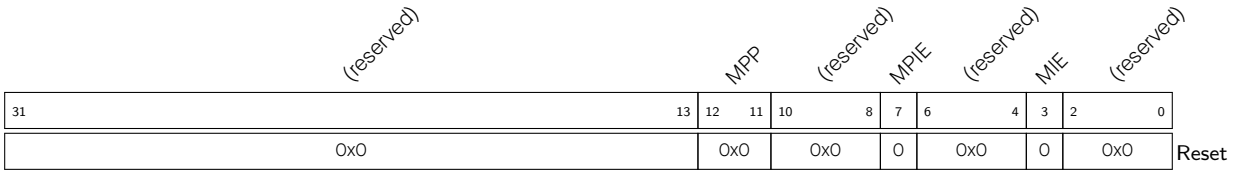
Register 2.1. mhartid (0xF14)



MHARTID Represents hart ID. The LP CPU hart ID is 0. (RO)

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

Register 2.2. mstatus (0x300)



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

MPIE Stores the interrupt-enable bit value that was active before a trap occurred. (R/W)

MPP Configures machine previous privilege mode (before trap).

0x1: User mode

0x3: Machine mode

Other values: Invalid

Note: Only the lower bit is writable. Any write to the higher bit is ignored as it is directly tied to the lower bit.

(R/W)

Register 2.3. misa (0x301)

MXL	(reserved)	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
31 30	29 26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1

Reset

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

Z Reserved = 0. (RO)

Y Reserved = 0. (RO)

X Non-standard extensions present = 0. (RO)

W Reserved = 0. (RO)

V Reserved = 0. (RO)

U User mode implemented = 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)

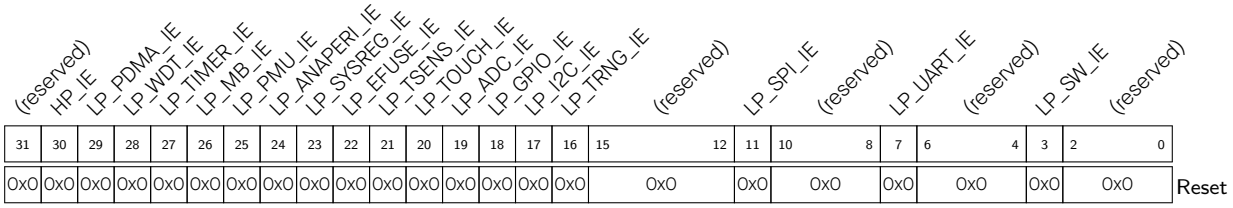
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

A Atomic Standard Extension = 1. (RO)

Register 2.4. mie (0x304)



LP_SW_IE Write 1 to enable LP software interrupt. (R/W)

LP_UART_IE Write 1 to enable LP UART interrupt. (R/W)

LP_SPI_IE Write 1 to enable LP SPI interrupt. (R/W)

LP_I2C_IE Write 1 to enable I2C interrupt. (R/W)

LP_TRNG_IE Write 1 to enable TRNG interrupt. (R/W)

LP_GPIO_IE Write 1 to enable LP GPIO interrupt. (R/W)

LP_ADC_IE Write 1 to enable LP ADC interrupt. (R/W)

LP_TOUCH_IE Write 1 to enable LP TOUCH interrupt. (R/W)

LP_TSENS_IE Write 1 to enable LP TSENS interrupt. (R/W)

LP_EFUSE_IE Write 1 to enable LP eFuse interrupt. (R/W)

LP_SYSREG_IE Write 1 to enable SYSREG interrupt. (R/W)

LP_ANAPERI_IE Write 1 to enable ANAPERI interrupt. (R/W)

LP_PMU_IE Write 1 to enable PMU interrupt. (R/W)

LP_MB_IE Write 1 to enable Mailbox interrupt. (R/W)

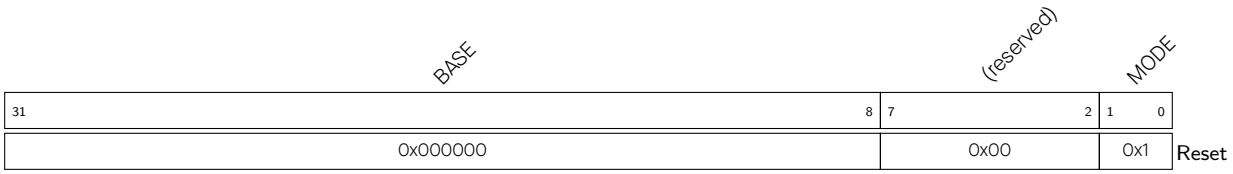
LP_TIMER_IE Write 1 to enable LP Timer interrupt. (R/W)

LP_WDT_IE Write 1 to enable WDT interrupt. (R/W)

LP_PDMA_IE Write 1 to enable PDMA interrupt. (R/W)

HP_IE Write 1 to enable HP interrupt. (R/W)

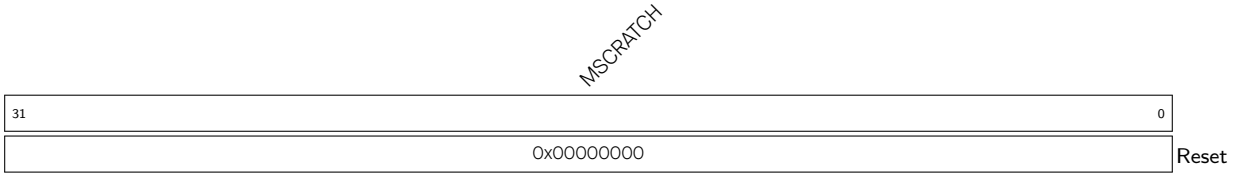
Register 2.5. mtvec (0x305)



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

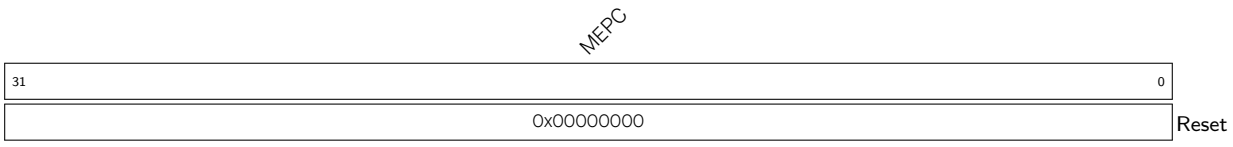
BASE Configures the higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 2.6. mscratch (0x340)



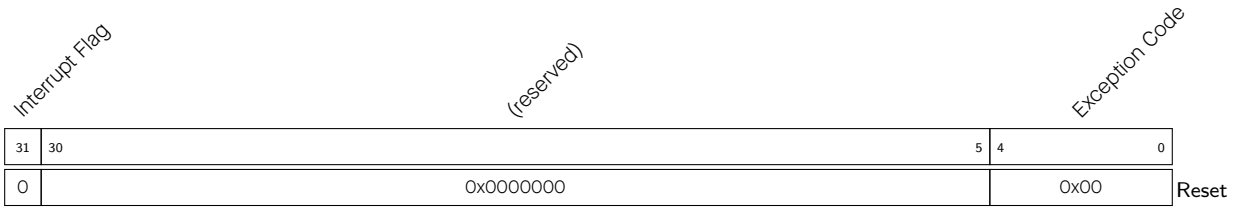
MSCRATCH Contains machine scratch information for custom use. (R/W)

Register 2.7. mepc (0x341)



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

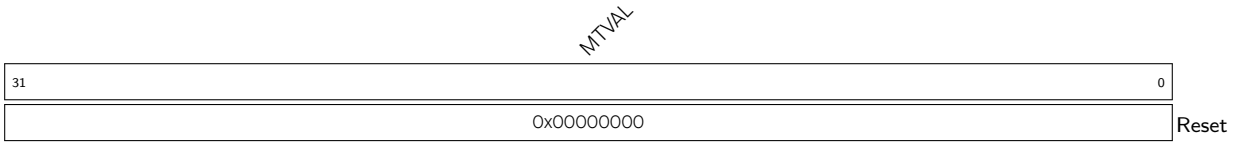
Register 2.8. mcause (0x342)



Exception Code This field is automatically updated with the unique ID of the most recent exception or interrupt due to which CPU entered trap. Possible exception IDs are:
0x1: Instruction access fault 0x2: Illegal instruction
0x3: Hardware breakpoint/watchpoint or EBREAK
0x5: Load access fault
0x6: Misaligned atomic instructions
0x7: Store access fault
Note: Exception ID 0x0 (instruction access misaligned) is not present because the CPU always masks the lowest bit of the address during instruction fetch.
(R/W)

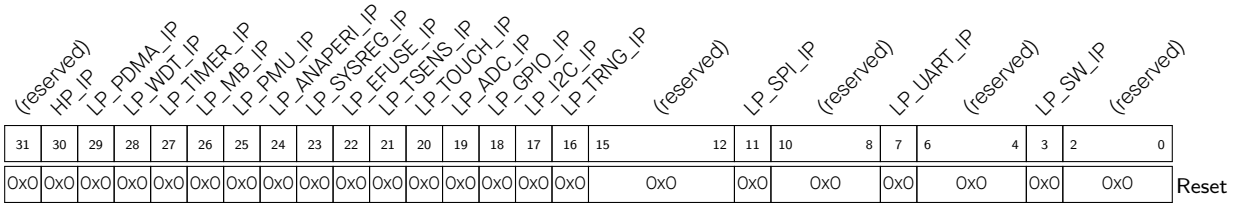
Interrupt Flag This flag is automatically updated when CPU enters trap. If this is found to be set, it indicates that the latest trap occurred due to an interrupt. For exceptions it remains unset. (R/W)

Register 2.9. mtval (0x343)



MTVAL Configures machine trap value. This is automatically updated with an exception dependent data which may be useful for handling that exception. Data is to be interpreted depending upon exception IDs:
0x1: Faulting address of instruction
0x2: Faulting instruction opcode
0x5: Faulting data address of load operation
0x7: Faulting data address of store operation
Note: The value of this register is not valid for other exception IDs and interrupts.
(R/W)

Register 2.10. mip (0x344)



LP_SW_IP Configures the pending status of the LP software interrupt.

0: Not pending

1: Pending

(R/W)

LP_UART_IP Configures the pending status of the LP UART interrupt.

0: Not pending

1: Pending

(R/W)

LP_SPI_IP Configures the pending status of the LP SPI interrupt.

0: Not pending

1: Pending

(R/W)

LP_TRNG_IP Configures the pending status of the LP TRNG interrupt.

0: Not pending

1: Pending

(R/W)

LP_I2C_IP Configures the pending status of the LP I2C interrupt.

0: Not pending

1: Pending

(R/W)

LP_GPIO_IP Configures the pending status of the LP GPIO interrupt.

0: Not pending

1: Pending

(R/W)

LP_ADC_IP Configures the pending status of the LP ADC interrupt.

0: Not pending

1: Pending

(R/W)

LP_TOUCH_IP Configures the pending status of the LP TOUCH interrupt.

0: Not pending

1: Pending

(R/W)

LP_TSENS_IP Configures the pending status of the LP TSENS interrupt.

0: Not pending

1: Pending

(R/W)

Continued on the next page...

Register 2.10. mip (0x344)

Continued from the previous page...

LP_EFUSE_IP Configures the pending status of the LP eFuse interrupt.
0: Not pending
1: Pending
(R/W)

LP_SYSREG_IP Configures the pending status of the LP SYSREG interrupt.
0: Not pending
1: Pending
(R/W)

LP_ANAPERI_IP Configures the pending status of the LP ANAPERI interrupt.
0: Not pending
1: Pending
(R/W)

LP_PMU_IP Configures the pending status of the PMU interrupt.
0: Not pending
1: Pending
(R/W)

LP_MB_IP Configures the pending status of the mailbox interrupt.
0: Not pending
1: Pending
(R/W)

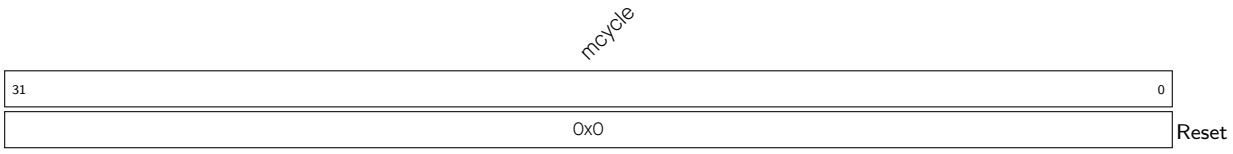
LP_TIMER_IP Configures the pending status of the LP Timer interrupt.
0: Not pending
1: Pending
(R/W)

LP_WDT_IP Configures the pending status of the LP WDT interrupt.
0: Not pending
1: Pending
(R/W)

LP_PDMA_IP Configures the pending status of the LP PDMA interrupt.
0: Not pending
1: Pending
(R/W)

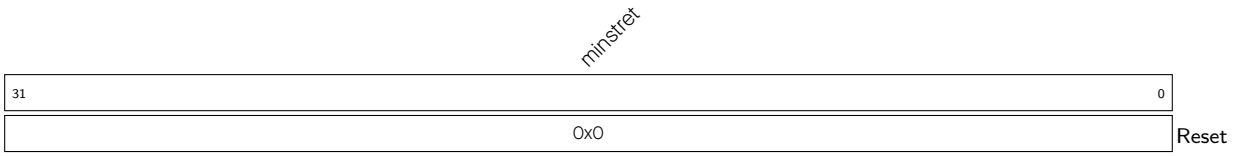
HP_IP Configures the pending status of the HP interrupt.
0: Not pending
1: Pending
(R/W)

Register 2.11. mcycle (0xB00)



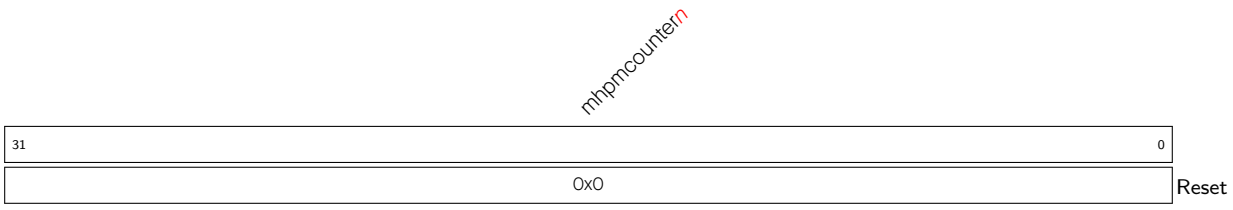
MCYCLE Configures the lower 32 bits of the clock cycle counter. (R/W)

Register 2.12. minstret (0xB02)



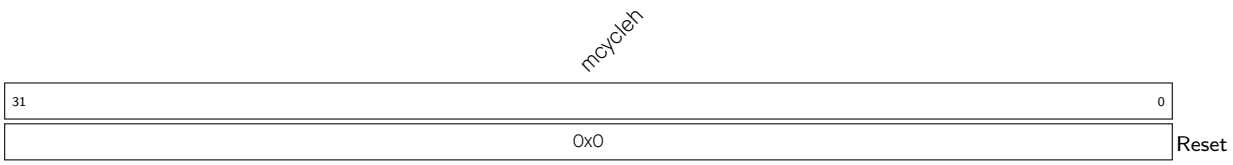
MINSTRET Configures the lower 32 bits of the instruction counter. (R/W)

Register 2.13. mhpmcounter*n*(*n*: 3-12) (0xB00+*n*)



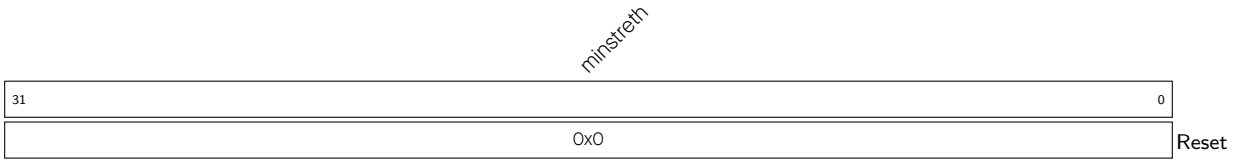
MHPMCOUNTER*n* Configures the lower 32 bits of the performance counter *n*. (R/W)

Register 2.14. mcycleh (0xB80)



MCYCLEH Configures the higher 32 bits of the clock cycle counter. (R/W)

Register 2.15. minstreth (0xB82)



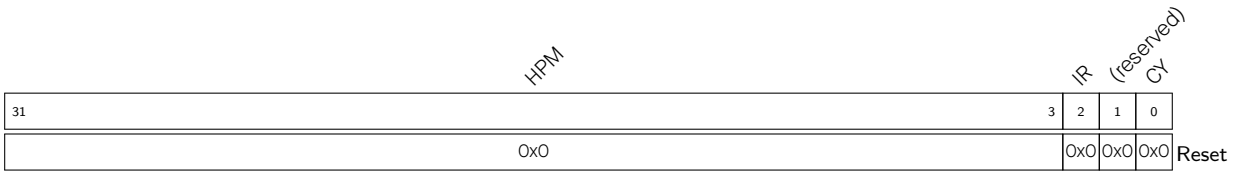
MINSTRETH Configures the higher 32 bits of the instruction counter. (R/W)

Register 2.16. mhpmcounter*n*(*n*: 3-12)h (0xB80+*n*)



MHPMCOUNTER*n*h Configures the higher 32 bits of the performance counter *n*. (R/W)

Register 2.17. mcountinhibit (0x320)



HPM Configures whether the performance counter *n*(*n*:3-12) increments.

0: The counter does not count

1: The counter increments

(R/W)

IR Configures whether the instruction counter increments.

0: The counter does not count

1: The counter increments

(R/W)

CY Configures whether the clock cycle counter increments.

0: The counter does not count

1: The counter increments

(R/W)

2.4 Interrupts and Exceptions

The LP CPU handles interrupts and exceptions according to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. After entering an interrupt/exception handler, the CPU:

- Saves the current program counter (PC) value to the mepc CSR
- Copies the state of MIE of mstatus to MPIE of mstatus
- Saves the current privileged mode to MPP of mstatus
- Clears MIE of mstatus
- Toggles the privileged mode to machine mode (M mode)
- Jumps to the handler address
  - For exceptions, the handler address is the base address of the vector table in the mtvec CSR
  - For interrupts, the handler address is $mtvec + 4 \times ID$, where ID is the interrupt ID. For more information, see Section 2.4.1
- After the mret instruction is executed, the core jumps to the PC saved in the mepc CSR, restores the value of MPIE of mstatus to MIE of mstatus, and restores the privileged mode to that indicated by MPP of mstatus

When the core boots, the base address of the vector table is initialized to the boot address 0x2E000000. After boot, the base address can be changed by writing the mtvec CSR. For details on configuration and status registers, see Section 2.3.1.

After a reset, the core starts to fetch instructions from the memory address specified by the LP_SYS_LP_CORE_BOOT_ADDR_REG register with an offset of 0x80.

2.4.1 Interrupts

The LP CPU in the ESP32-S31 supports 18 interrupt inputs. Each interrupt corresponds to an ID, which is denoted as n (where n = 3, 7, 11, 16 to 30), and has a specific entry address of $mtvec + 4 \times n$. These interrupts are associated with specific peripheral sources. For the mapping between peripheral sources and interrupt IDs, please refer to 2.4-1.

The read-only register LPINTR_STATUS_REG indicates the current status of the LP CPU external interrupt sources. For the mapping between this register and the interrupt sources, please refer to the description of this register. For details on the interrupt sources, see Section 9.

Table 2.4-1. LP CPU Interrupt Mapping

Interrupt ID	Peripheral Source
3	LP_SW_INTR, LP_HUK_INTR
7	LP_UART_INTR
11	LP_SPI_INTR
16	LP_TRNG_INTR
17	LP_I2C_INTR
18	LP_GPIO_INTR

Interrupt ID	Peripheral Source
19	LP_ADC_INTR, LP_DAC_INTR
20	LP_TOUCH_INTR
21	LP_TSENS_INTR
22	LP_EFUSE_INTR
23	LP_SYSREG_INTR, LP_APM_INTR, LP_PERI_PMS_INTR, LP_PERI_TIMEOUT_INTR
24	LP_ANAPERI_INTR
25	PMU_REG_0_INTR, PMU_REG_1_INTR
26	MB_HP_INTR, MB_LP_INTR
27	LP_TIMER_REG_0_INTR, LP_TIMER_REG_1_INTR
28	LP_WDT_INTR
29	LP_PDMA_INTR
30	HP_INTR

2.4.2 Interrupt Handling

By default, interrupts are disabled globally because the MIE bit in mstatus has a reset value of 0. Software must set this bit to enable global interrupts.

Assuming the interrupt to enable is Interrupt n:

1. Enable Interrupt n:
   - To enable interrupts globally, write 1 to the MIE bit of mstatus.
   - To enable Interrupt n, write 1 to the nth bit of mie CSR.
2. After Interrupt n is enabled, the LP CPU can respond to it. Meanwhile, the corresponding peripheral interrupt should be configured to send an interrupt signal to the LP CPU.
3. After the interrupt is triggered, the LP CPU jumps to the $mtvec + 4 \times n$ address.
4. To clear Interrupt n, the peripheral needs to clear the interrupt signal.

2.4.3 Exceptions

The LP CPU supports the RISC-V standard exceptions and can trigger the following exceptions:

Table 2.4-2. LP CPU Exception Causes

Exception ID	Description
1	Instruction access fault
2	Illegal instructions
3	Breakpoints (EBREAK)
5	Load access fault
6	Misaligned atomic instructions
7	Store access fault

2.5 Debugging

This section describes how to debug and test the LP CPU. Debug support is provided through standard JTAG pins and complies with RISC-V External Debug Support Version 0.13.

For ESP32-S31 system debugging overview, please refer to Chapter *2 High-Performance CPU [to be added later]* > Figure *34 DAC Controller (DAC) [to be added later]*.

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), which in turn connects to Debug Transport Hardware (e.g., ESP-Prog adapter). The Debug Transport Hardware connects the Debug Host to the CPU's Debug Transport Module (DTM) through a standard JTAG interface. The DTM provides access to the debug module (DM) using the Debug Module Interface (DMI).

The LP CPU implements four registers for core debugging: dcsr, dpc, dscratch0, and dscratch1. All of those registers can only be accessed from debug mode. If software attempts to access them when the LP CPU is not in debug mode, an illegal instruction exception will be triggered.

2.5.1 Features

The LP CPU debug module has the following debugging features:

- Access to necessary information about the implementation
- CPU core halt and resume
- Read and write of CPU core registers (including CSRs)
- CPU core reset
- CPU halted on software breakpoint (planted breakpoint instruction)
- Hardware single-stepping
- Two hardware triggers (which can be used as breakpoints/watchpoints). See Section 2.6 for details

2.5.2 Functional Description

The debugging mechanism adheres to the specification RISC-V External Debug Support Version 0.13. For a detailed description of the debugging features, refer to the specification.

2.5.3 Register Summary

The following table lists the debug CSRs supported for the LP CPU.

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

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

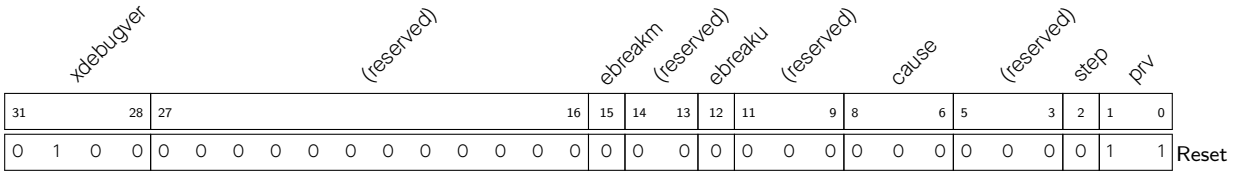
All debug module registers are implemented in accordance with the specification RISC-V External Debug Support Version 0.13. For more information, refer to the specification.

2.5.4 Registers

The following is a detailed description of the debug CSR supported by the LP CPU.

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

Register 2.18. dcsr (0x7B0)



xdebugver Represents the debug version.

4: External debug support exists

(RO)

ebreakm Configures execution of the EBREAK instruction in machine mode.

0: Trigger an exception with mcause = 3

1: Enter debug mode

(R/W)

ebreaku Configures execution of the EBREAK instruction in user mode.

0: Trigger an exception with mcause = 3 as described in the Privileged Architecture specification

1: Enter debug mode

(R/W)

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

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

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

3: haltreq was set. (priority 2)

4: The CPU core single stepped because step was set. (priority 1)

Other values: reserved for future use

(RO)

step When set and not in Debug Mode, the core will only execute a single instruction and then enter Debug Mode.

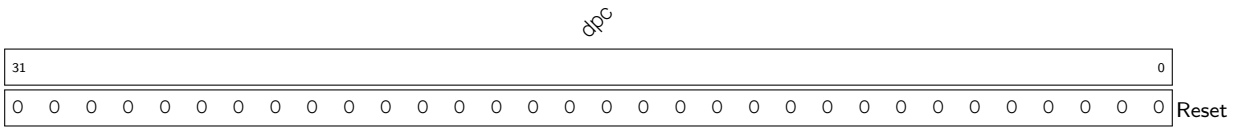
If the instruction does not complete due to an exception, the core will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.

Setting this bit does not mask interrupts. This is a deviation from the RISC-V External Debug Support Specification Version 0.13.

(R/W)

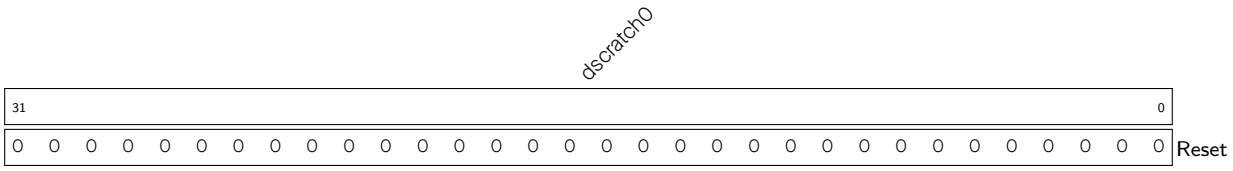
prv Contains the privilege level the core is operating in when debug mode is entered. A debugger can change this value to change the core's privilege level when exiting debug mode. Only **0x3** (machine mode) and **0x0** (user mode) are supported. (R/W)

Register 2.19. dpc (0x7B1)



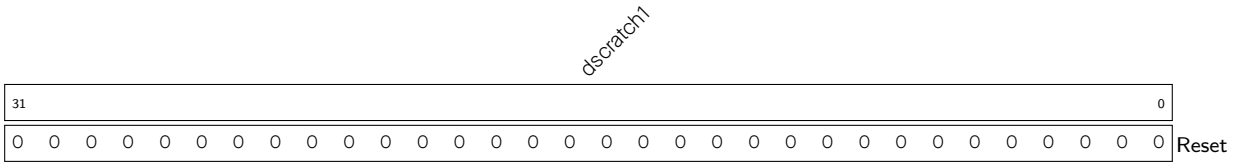
dpc Upon entry to debug mode, dpc is written with the address of the next instruction that will be executed. When resuming, the CPU core's PC is updated to the address stored in dpc. In debug mode, dpc can be modified. This field can be accessed in debug mode. (R/W)

Register 2.20. dscratch0 (0x7B2)



dscratch0 Used by the debug module internally. (R/W)

Register 2.21. dscratch1 (0x7B3)



dscratch1 Used by the debug module internally. (R/W)

2.6 Hardware Trigger

2.6.1 Features

The hardware trigger module provides breakpoint and watchpoint capability for debugging. It has the following features:

- Two independent trigger units
- Configurable unit to match the address of the program counter
- Able to halt execution and transfer control to the debugger

2.6.2 Functional Description

The hardware trigger module provides five CSRs. See Section 2.6.4 for details. Among them, tdata1 and tdata2 are abstract CSRs, which means they are shadow registers for accessing internal registers in the trigger units, one at a time.

To select a specific trigger unit, the corresponding number (0-1) needs to be written to the tselect CSR. When a valid value is written, the abstract CSRs, tdata1 and tdata2, automatically match the internal registers of the trigger unit. Each trigger unit has two internal registers, namely mcontrol and maddress, which are mapped to tdata1 and tdata2, respectively.

Writing a value more than 1 to tselect will set tselect to 1.

Since software or debugger may need to know the type of the selected trigger to correctly interpret tdata1 and tdata2, the 4-bit field (31-28) of tdata1 encodes the type of the selected trigger. This type field is read-only and always provides a value of 0x2 for every trigger, which stands for support for address and data matching. Hence, it is inferred that tdata1 and tdata2 are to be interpreted as fields of mcontrol and maddress, respectively. The specification RISC-V External Debug Support Version 0.13 provides information on other possible values, but the trigger module only supports the 0x2 type.

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

2.6.3 Trigger Execution Flow

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

- dpc is set to the current PC in the decoding phase
- The cause field in dcsr is set to 2, which means halt due to trigger

2.6.4 Register Summary

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

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

Name	Description	Address	Access
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
mcontrol	tdata1 Shadow Register	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
maddress	tdata2 Shadow Register	0x7A2	R/W

2.6.5 Registers

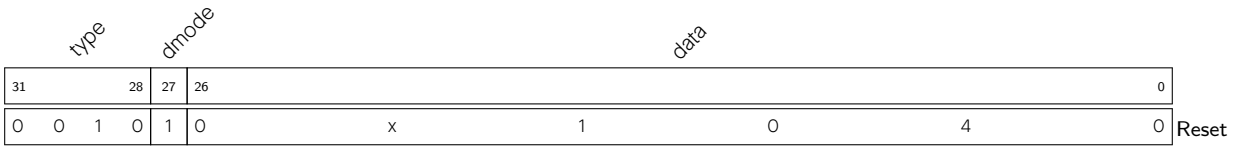
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 2.22. tselect (0x7A0)



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

Register 2.23. tdata1 (0x7A1)



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

dmode This is set to 1 if a trigger is being used by the debugger. This field is reserved since it is only supported in debug mode. (RO)

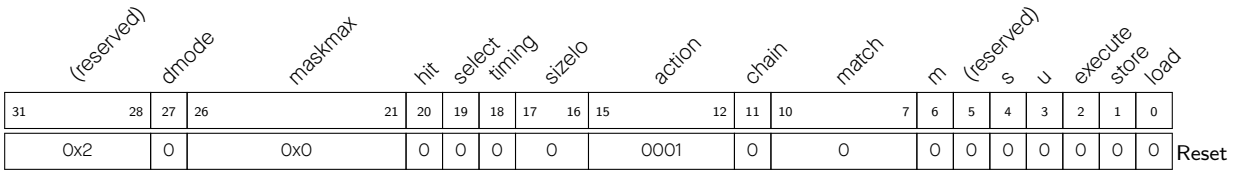
data Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

Register 2.24. tdata2 (0x7A2)



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

Register 2.25. mcontrol (0x7A1)



dmode Same as dmode in tdata1. (RO)

maskmax Represents the maximum NAPOT range.

0: A byte. Only exact match is supported.

Other values: Not supported.

(RO)

hit Not implemented in hardware. This field remains 0. (RO)

select Configures to select between an address match or a data match.

0: Perform a match on the virtual address

1: Perform a match on the data value loaded or stored, or the instruction executed

Note: Only address match is implemented. This field remains 0.

(RO)

timing Configures when the trigger will take action.

0: Take action before the instruction is executed

1: Take action after the instruction is executed

Note: The field remains 0.

(RO)

sizelo Only match of any size is supported. This field remains 0. (RO)

action Configure action of the selected trigger after it is triggered.

0x0: Cause a breakpoint exception

0x1: Enter debug mode (Valid only when dmode = 1)

Note: Only entering debug mode is supported. This field remains 1.

(RO)

chain Not implemented in hardware. This field remains 0. (RO)

match Configures the trigger to perform the matching operation of the lower data/instruction address.

0x0: Exact match. Namely, the address corresponding to a certain byte during the access must exactly match the value of maddress.

0x1: NAPOT match. Namely, at least one byte during the access is in the NAPOT region specified in maddress.

Note: Only exact byte match is supported. This field remains 0.

(R/W)

m Set this field to make the selected trigger operate in machine mode. (RO)

s Set this field to make the selected trigger operate in supervisor mode. Operation in supervisor mode is not supported. This field is always 0. (RO)

Continued on the next page...

Register 2.25. mcontrol (0x7A1)

Continued from the previous page...

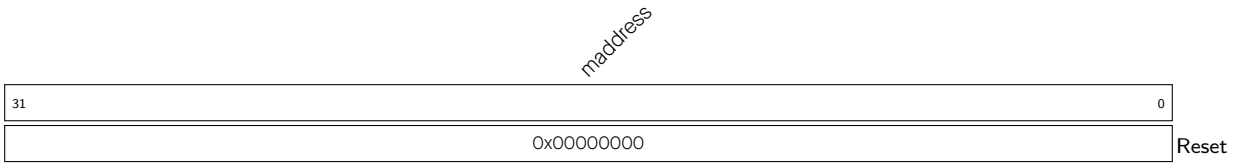
u Set this field to make the selected trigger operate in user mode. Operation in user mode is not supported. This field is always 0. (RO)

execute Configures whether to enable the selected trigger to match the virtual address of instructions.
0: Not enable
1: Enable
(R/W)

store Set this field to make the selected trigger match the virtual address of the memory write operation. Not supported by hardware. This field is always 0. (RO)

load Set this field to make the selected trigger match the virtual address of a memory read operation. Not supported by hardware. This field is always 0. (RO)

Register 2.26. maddress (0x7A2)



maddress Configures the address used by the selected trigger when performing match operation. (R/W)

2.7 Performance Counter

The LP CPU implements one clock cycle counter mcycle(h), one instruction counter minstret(h), and ten event counters mhpmcounter*n*(*n*:3-12). The clock cycle counter and instruction counter are always available and each is 64-bit wide. The performance counters are 40-bit wide each.

By default, all counters are enabled after reset. A counter can be enabled or disabled individually via the corresponding bit in the mcountinhibit CSR.

As shown in Table 2.7-1, each counter is dedicated to counting a particular event.

Table 2.7-1. Performance Counter

Counter	Counted Event
mcycle	Clock cycles
minstret	The number of instructions
mhpmcounter3	Wait cycles for memory access

Counter	Counted Event
mhpmcounter4	Wait cycles for fetching instructions
mhpmcounter5	The number of memory read operations. An unaligned read is counted as two.
mhpmcounter6	The number of memory write operations. An unaligned write is counted as two.
mhpmcounter7	The number of unconditional jump instructions (jal, jr, jalr)
mhpmcounter8	The number of branch instructions
mhpmcounter9	The number of taken branch instructions
mhpmcounter10	The number of compressed instructions
mhpmcounter11	Wait cycles for multiplication instructions
mhpmcounter12	Wait cycles for division instructions

2.8 System Access

The LP CPU has access to both memory and peripherals. When accessing an illegal address, i.e., outside the memory's or peripherals' address space, it will trigger the corresponding access error exception based on the type of access.

2.8.1 Memory Access

The ESP32-S31 LP CPU can access LP SRAM and HP SRAM. For more information, please refer to Section 3 *System and Memory [to be added later]*.

- LP SRAM: 32 KB starting from 0x2E00_0000 to 0x2E00_7FFF. It is used for instruction fetch, data read, data write, etc.
- HP SRAM: 512 KB starting from 0x2F00_0000 to 0x2F07_FFFF. It is used for instruction fetch, data read, data write, etc.

The LP CPU supports the atomic instruction set. Both the LP CPU and the HP CPU can access memory through atomic instructions, thus achieving atomicity of memory access. For details on the atomic instruction set, please refer to RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2.

Note: Only LP SRAM supports atomic accesses by both the HP CPU and the LP CPU.

2.8.2 Peripheral Access

Table 3 in Chapter 3 *System and Memory [to be added later]* lists the peripherals accessible by the LP CPU and their base addresses.

2.9 Event Task Matrix Feature

The LP CPU on ESP32-S31 supports the Event Task Matrix (ETM) function, which allows LP CPU's ETM tasks to be triggered by any peripherals' ETM events, or LP CPU's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the LP CPU. For more information, please refer to Chapter 10 Event Task Matrix (ETM) [to be added later].

LP CPU can receive the following ETM task:

- ULP_TASK_WAKEUP_CPU: Wakes up the LP CPU.
- ULP_TASK_INT_CPU: Triggers an LP CPU interrupt.

LP CPU can generate the following ETM events:

- ULP_EVT_ERR_INTR: Indicates that an LP CPU exception occurs.
- ULP_EVT_START_INTR: Indicates that the LP CPU clock is turned on.
- ULP_EVT_HALT: Indicates that the LP CPU enters the sleep state

2.10 Sleep and Wake-Up Process

2.10.1 Features

Here are the sleep and wake-up features supported by the LP CPU.

- Independent operation, sleeping, and waking-up in the low-power system when the HP CPU is sleeping
- Actively configuring registers to enter the sleep state based on software operating status
- Multiple wake-up sources
- Sleep entry rejection by peripherals

2.10.2 Process

The LP CPU is in sleep by default and its wake-up module follows the process below to wake it up for work and make it sleep.

To configure wake-up sources, please refer to Table 2.10-1.

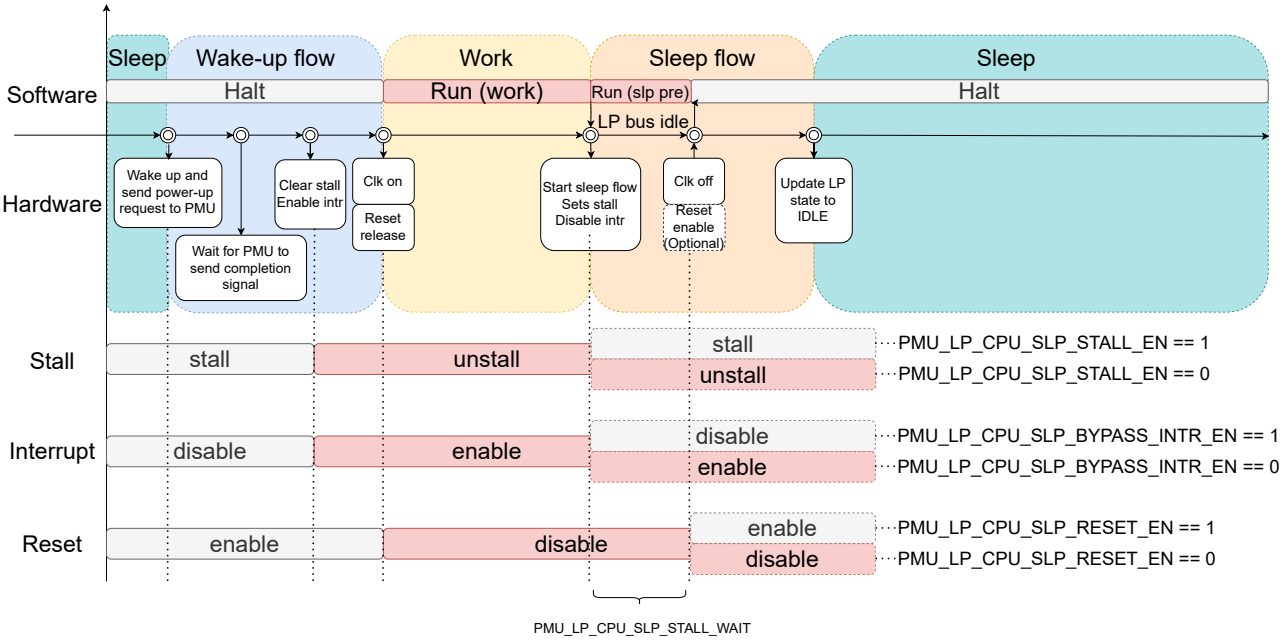


Figure 2.10-1. Wake-Up and Sleep Flow of LP CPU

The first startup of the LP CPU after power-up depends on the wake-up enable and wake-up source configuration by the HP CPU.

- Initialization of the LP CPU
  - Initialize the LP memory.
  - Start the LP CPU. Since the startup of the LP CPU depends on the wake-up process, it is recommended to use the PMU_HP_TRIGGER_LP register to start the initialization of the LP CPU in the following way:
    * Set PMU_LP_CPU_WAKEUP_EN to 0x00000002
    * Set PMU_HP_TRIGGER_LP to 0x1
    * The LP CPU will go through the wake-up process to start running
- Wake-up process:
  - The wake-up module receives a wake-up signal and sends a power-up request to the PMU.
  - If the current power consumption state (clock, power supply, etc.) meets the requirements of the LP CPU, the PMU will immediately reply with the completion signal. Otherwise, it will adjust the power consumption state before replying with the completion signal.
  - The wake-up module disables the STALL state of the LP CPU and enables interrupt receiving.
  - The wake-up module starts the clock, releases reset (ignore this step if reset is not enabled for sleep), and starts working.
- Sleep process:
  - The LP CPU configures the PMU_LP_CPU_SLEEP_REQ register to enable the wake-up module to start the sleep process.
  - If PMU_LP_CPU_SLP_STALL_EN is 1, the wake-up module enables the STALL state of the LP CPU. If it is 0, the module does not enable that state. If PMU_LP_CPU_SLP_BYPASS_INTR_EN is 1, the module masks all the LP CPU's interrupts. If it is 0, the module does not mask them.
  - The wake-up module waits for PMU_LP_CPU_SLP_STALL_WAIT LP CPU clock cycles, and then turns off the LP CPU clock. If PMU_LP_CPU_SLP_RESET_EN is 1, the module enables reset of the LP CPU.
  - The wake-up module changes the state of the LP CPU to IDLE and the PMU can enter a deeper level of sleep state.

2.10.3 Wake-Up Sources

Table 2.10-1. Wake Sources

Register Value [1]	Wake-Up Source	Description
BIT(1)	PMU_HP_TRIGGER_LP register	The HP CPU sets the register PMU_HP_TRIGGER_LP to wake up the LP CPU, and sets REG_HP_SW_TRIGGER_INT_CLR to clear this wake-up source.
BIT(9)	LP IO	The LP CPU can be woken up by the LP IO interrupt status register signal. For more information, please refer to Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.
BIT(10)	LP UART	The LP CPU can be woken up when the LP UART receives a certain number of RX pulses. REG_UART_WAKEUP_EN needs to be enabled. For more information, please refer to Chapter 21 *UART Controller (UART)*.
BIT(13)	RTC timer	The LP CPU can be woken up by the RTC timer target 0 timeout interrupt. For more information, please refer to Chapter 11 *Low-Power Management [to be added later]*.
BIT(14)	Brownout detection	The LP CPU can be woken up when a brownout occurs. For more information, please refer to Chapter 9 *Power Supply Detector (PSDET)*.
BIT(17)	ETM	The LP CPU can be woken up by the ETM task. For more information, please refer to Chapter 10 *Event Task Matrix (ETM) [to be added later]*.
BIT(18)	RTC timer	The LP CPU can be woken up by the RTC timer target 1 timeout interrupt. For more information, please refer to Chapter 11 *Low-Power Management [to be added later]*.

[1] Value of the PMU_LP_CPU_WAKEUP_EN register

2.10.4 Sleep Rejection

When the LP CPU is preparing to enter the sleep state, it can be rejected by peripheral events. In other words, peripheral events have the capability to prevent the LP CPU from entering sleep mode. Below is the configuration process for the rejection:

1. The LP CPU configures PMU_SLP_REJECT_EN to enable the functionality that allows peripherals to reject the LP CPU from entering the sleep mode.
2. The LP CPU configures PMU_SLEEP_REJECT_ENA to enable the corresponding peripheral event to prevent the LP CPU from entering the sleep state.

The peripheral sources that can prevent the LP CPU from sleeping are the same as the wake-up sources.

2.11 Physical Memory Protection

The physical memory protection unit (PMP) provides access control for the physical address space and can be configured in machine mode. The LP CPU core includes a PMP unit, which is fully compliant with RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. For information not defined in this chapter, please refer to the RISC-V specification.

2.11.1 Features

In addition to the features supported in the Privileged Architecture specification, the LP CPU has the following implementation-level features:

- 16 PMP entries
- Minimum configurable region granularity per entry: 128 bytes
- Three address-matching modes: OFF, TOR (Top of Range), and NAPOT (Naturally Aligned Power-of-Two)

2.11.2 Functional Description

The PMP mechanism follows the RISC-V specification, so configure PMP according to the RISC-V specification. After configuration, requests that fail PMP checks will raise instruction access fault, load access fault, or store access fault exceptions.

2.11.3 Register Summary

The following table lists the PMP CSRs, which can be read and written only in machine mode.

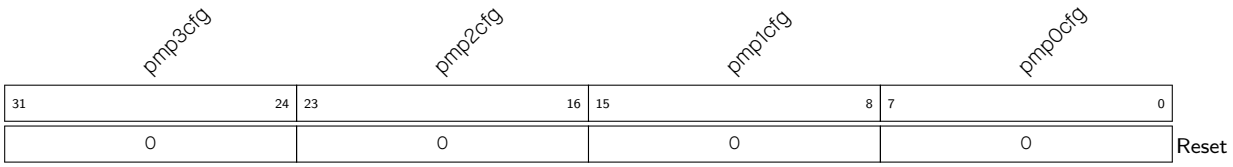
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
pmpcfg0	Physical memory protection configuration register	0x3A0	R/W
pmpcfg1	Physical memory protection configuration register	0x3A1	R/W
pmpcfg2	Physical memory protection configuration register	0x3A2	R/W
pmpcfg3	Physical memory protection configuration register	0x3A3	R/W
pmpaddr*n* (*n*: 0-15)	Physical memory protection address register	0x3B0+0x1**n*	R/W

2.11.4 Registers

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

Register 2.27. pmpcfg0 (0x3A0)



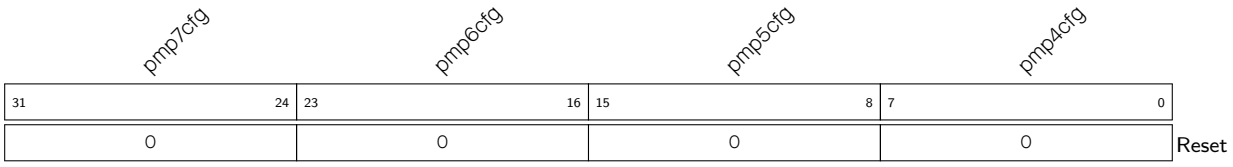
pmp3cfg Configuration register for PMP entry 3. (R/W)

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

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

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

Register 2.28. pmpcfg1 (0x3A1)



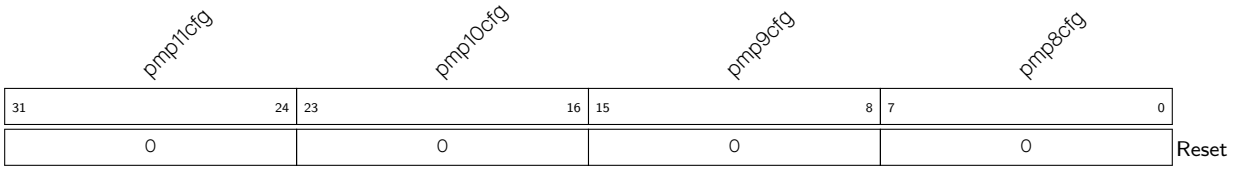
pmp7cfg Configuration register for PMP entry 7. (R/W)

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

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

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

Register 2.29. pmpcfg2 (0x3A2)



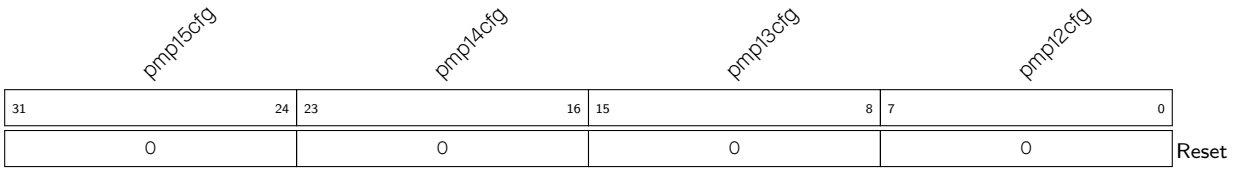
pmp11cfg Configuration register for PMP entry 11. (R/W)

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

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

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

Register 2.30. pmpcfg3 (0x3A3)



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

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

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

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

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

Register 2.31. pmpXcfg



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

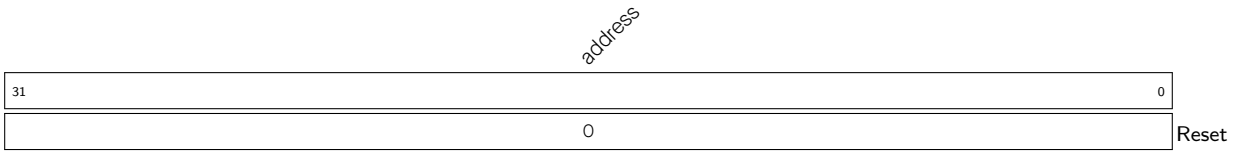
A Configures the address matching mode.
0: Off
1: TOR (top of range)
2: Not supported
3: NAPOT (naturally aligned power-of-two region >= 128 bytes)
(R/W)

X Configures execute permissions. After it is set, execution from the address range is permitted. (R/W)

W Configures write permissions. Once set, memory writes to the address range are permitted. (R/W)

R Configures read permissions. Once set, memory can be read from the address range. (R/W)

Register 2.32. pmpaddr*n* (*n*: 0-15) (0x3B0+0x1n*)**



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

Chapter 3

GDMA Controller (GDMA-AHB, GDMA-AXI)

3.1 Overview

General Direct Memory Access (GDMA) provides high-speed data transfer between peripherals and memory, and between memory and memory. With GDMA, software can transfer data without CPU intervention, reducing CPU load and improving system efficiency.

ESP32-S31 has two types of GDMA controllers that directly access the AHB bus and the AXI bus, hereinafter referred to as GDMA-AHB and GDMA-AXI. Depending on the system, GDMA-AHB is further divided into GDMA-AHB-HP and GDMA-AHB-LP for high-performance and low-power systems. GDMA channels are shared among peripherals that support GDMA. Users can assign a channel to any such peripheral.

- GDMA-AHB-HP: 10 independent channels, including 5 transmit channels and 5 receive channels. Supported peripherals include UHCI, I2S, RMT, Audio Sample Rate Converter (ASRC), and CORDIC accelerator.
- GDMA-AHB-LP: 4 independent channels, including 2 transmit channels and 2 receive channels. Supported peripherals include ADC and DAC.
- GDMA-AXI: 6 independent channels, including 3 transmit channels and 3 receive channels. Supported peripherals include LCD, CAM, two general-purpose SPIs, PARLIO, AES, and SHA.

GDMA-AHB-HP and GDMA-AHB-LP are functionally identical, differing only in the number of channels and supported peripherals. The following description uses GDMA-AHB-HP as the reference, and everything applies equally to GDMA-AHB-LP by replacing the register prefix `AHB_DMA_` with `LP_AHB_DMA_`. Unless otherwise specified, GDMA-AHB refers to both GDMA-AHB-HP and GDMA-AHB-LP, and GDMA refers to both GDMA-AHB and GDMA-AXI.

On ESP32-S31, both GDMA-AHB and GDMA-AXI can access internal and external memory. GDMA-AHB and GDMA-AXI use configurable priority and weight arbitration to meet the bandwidth requirements of different peripherals.

3.2 Features

GDMA-AHB and GDMA-AXI have the following common features:

- Data transfer in bytes with software-programmable transfer length
- Access via any address and size
- Linked list
- Software-configurable peripheral selection for each channel

- Configurable channel priority and weight arbitration
- Memory-to-memory transfer

GDMA-AHB features:

- Architecture: AHB bus architecture
- Channels: GDMA-AHB-HP has five transmit channels and five receive channels; GDMA-AHB-LP has two transmit channels and two receive channels
- Alignment requirements:
  - Descriptor address: 1-word aligned
  - Data address and length:
    * Internal memory and non-encrypted external memory: no requirements
    * Encrypted external memory (only accessed by GDMA-AHB-HP, not GDMA-AHB-LP): 16-byte aligned
- INCR4, INCR8, or INCR16 burst transfer when accessing memory

GDMA-AXI features:

- Architecture: AXI bus architecture, supporting out-of-order transfers with a depth of 8 and outstanding transfers with a depth of 8
- Channels: three transmit channels and three receive channels
- Alignment requirements:
  - Descriptor address: 2-word aligned
  - Data address and length:
    * Internal memory and non-encrypted external memory: no requirements
    * Encrypted external memory: 16-byte aligned
- Burst transfer when accessing memory, with burst length configurable as 1, 2, 4, 8, or 16 beats
- Interrupt response mechanism for linked list switching

3.3 Architecture

The GDMA controller and the CPU data bus use the same address space to access memory. All modules on ESP32-S31 that require high-speed data transfer support GDMA. Figure 3.3-1 shows the basic architecture of the GDMA controller.

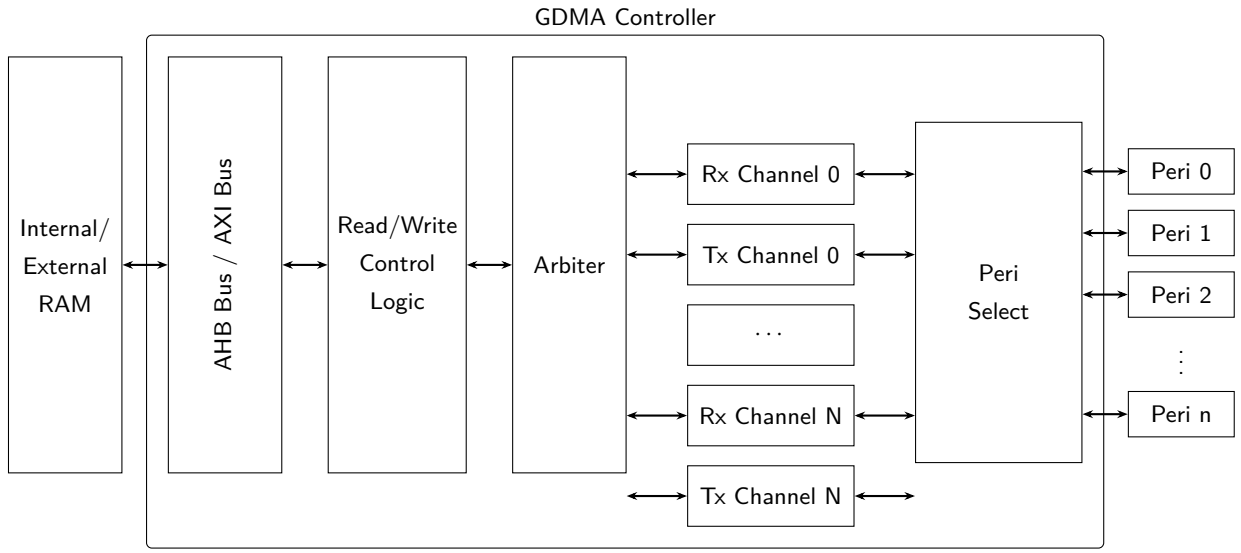


Figure 3.3-1. GDMA Controller Architecture

Both GDMA-AHB and GDMA-AXI have a fixed number of transfer channels. The number of transmit channels equals the number of receive channels. Each channel can connect to a different peripheral, so channel resources are shared among peripherals.

GDMA-AHB and GDMA-AXI read data from or write data to internal or external memory via the AHB bus or the AXI bus respectively. Before transferring data over the bus, GDMA arbitrates read and write requests from each channel using a configurable arbitration mechanism. For the memory address ranges available to GDMA, see Chapter 3 *System and Memory [to be added later]*.

Software uses the GDMA controller through linked lists. GDMA-AHB linked lists are stored in internal memory; GDMA-AXI linked lists are stored in internal or external memory. These linked lists include outlink*n* and inlink*n*. In this chapter, *n* denotes the physical channel number.

GDMA reads linked lists from memory, then sends data from the corresponding memory according to outlink*n*, or stores received data in the specified memory address space according to inlink*n*.

3.4 Functional Description

3.4.1 Linked List

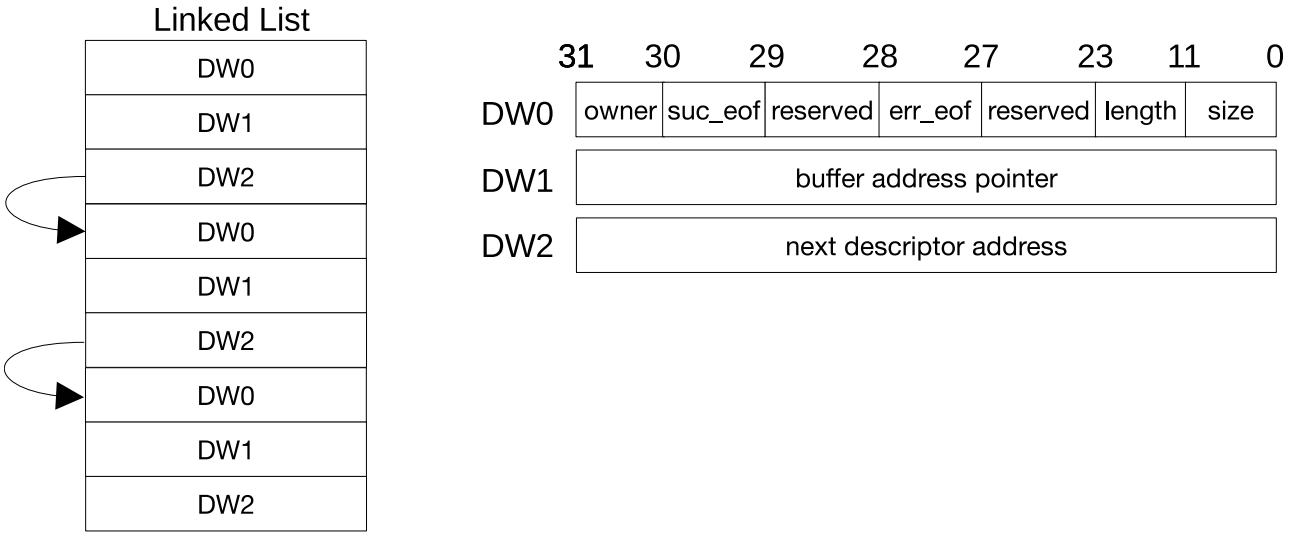


Figure 3.4-1. Structure of a Linked List

Figure 3.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list consists of one or more descriptors; each descriptor has three words. Store linked lists in memory so that GDMA can use them. The descriptor fields are defined as follows:

- owner (DW0) [31]: Specifies who may access the buffer that this descriptor points to.
  0: The CPU may access the buffer.
  1: The GDMA controller may access the buffer.
  After GDMA finishes using the buffer, hardware automatically clears this bit in a receive descriptor. For a transmit descriptor, hardware clears this bit automatically only if AHB/AXI_DMA_OUT_AUTO_WRBACK_CH*n* is set to 1. Software can disable automatic clearing by setting AHB/AXI_DMA_OUT_LOOP_TEST_CH*n* or AHB/AXI_DMA_IN_LOOP_TEST_CH*n*. When software loads a linked list, set this bit to 1.
  Note: In this chapter, registers prefixed with AHB/AXI_DMA_OUT belong to transmit channels; registers prefixed with AHB/AXI_DMA_IN belong to receive channels.
- suc_eof (DW0) [30]: Specifies whether the AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT or AHB/AXI_DMA_OUT_EOF_CH*n*_INT interrupt is triggered after the data corresponding to this descriptor is transferred.
  0: No interrupt is triggered after the current descriptor is transferred successfully.
  1: An interrupt is triggered after the current descriptor is transferred successfully.
  For receive descriptors, software clears this bit to 0; hardware sets it to 1 after receiving data that contains the EOF flag.
  For transmit descriptors, software sets this bit to 1 as needed.
  If software sets this bit to 1, GDMA includes the EOF flag in the data sent to the peripheral, indicating that this data segment has ended.
- 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 applies only when UHCI or PARLIO receives data via GDMA. Hardware sets this bit to 1 when it detects an error in the data segment corresponding to a descriptor.

- reserved (DW0) [27:24]: Reserved.
- length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. In a transmit descriptor, software writes this field to indicate how many bytes to read from the buffer. In a receive descriptor, hardware writes this field automatically to indicate how many valid bytes are stored in the buffer.
- size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
- buffer address pointer (DW1): Address of the buffer.
- next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value is 0. For GDMA-AHB, this field can only point to the internal memory space; for GDMA-AXI, this field can point to the internal or external memory space.

> **Note:**
> GDMA-AHB descriptor addresses must be 4-byte aligned; GDMA-AXI descriptor addresses must be 8-byte aligned.

When GDMA receives data, if the received data length is smaller than the buffer size specified by the current descriptor, the next descriptor does not use the remaining space in that buffer.

3.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer

The GDMA controller supports peripheral-to-memory and memory-to-peripheral data transfer, corresponding to receive (RX) and transmit (TX) respectively. A TX channel transfers data from the specified memory region to a peripheral transmitter via outlink*n*. An RX channel transfers data received by a peripheral to the specified memory region via inlink*n*.

Each TX or RX channel can connect to any peripheral that supports GDMA. Table 3.4-1, Table 3.4-2, and Table 3.4-3 show how to map channels to peripherals via registers. The value "Dummy-*n*" is used for memory-to-memory transfer. When one channel is already connected to a peripheral, no other channel can connect to that peripheral.

Table 3.4-1. GDMA-AHB-HP Selecting Peripherals via Register Configuration

AHB_DMA_PERI_IN_SEL_CH*n* AHB_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	UHCI0
1	I2S0-0
2	I2S0-1
3	I2S0-2
4	I2S0-3
5	I2S1-0
6	I2S1-1
7	I2S1-2

8	I2S1-3
9	Dummy-9
10	RMT
11	Audio Sample Rate Converter-0
12	Audio Sample Rate Converter-1
13	CORDIC accelerator
14 ~ 63	Invalid value

Table 3.4-2. GDMA-AHB-LP Selecting Peripherals via Register Configuration

LP_AHB_DMA_PERI_IN_SEL_CH*n* LP_AHB_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	ADC
1	DAC
2 ~ 63	Dummy

Table 3.4-3. GDMA-AXI Selecting Peripherals via Register Configuration

AXI_DMA_PERI_IN_SEL_CH*n* AXI_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	LCD or CAM
1	SPI2
2	SPI3
3	PARLIO
4	AES
5	SHA
6 ~ 15	Dummy-6 ~ 15
16 ~ 63	Invalid value

3.4.3 Memory-to-Memory Data Transfer

The GDMA controller supports memory-to-memory data transfer. Such data transfer is performed by setting AHB/AXI_DMA_MEM_TRANS_EN_CH*n* to connect the output of TX channel *n* to the input of RX channel *n*.

A TX channel is paired only with the RX channel of the same number for memory-to-memory transfer. AHB/AXI_DMA_PERI_IN_SEL_CH*n* and AHB/AXI_DMA_PERI_OUT_SEL_CH*n* must be configured to the same value corresponding to "Dummy".

All 5 channels of GDMA-AHB-HP and 2 channels of GDMA-AHB-LP support memory-to-memory data transfers. The ID recorded by AHB_DMA_AHBINF_RESP_ERR_ID is fixed: ID 16 indicates memory-to-memory transfer on GDMA-AHB channel 0; ID 17 indicates memory-to-memory transfer on GDMA-AHB channel 1, and so on.

3.4.4 Enabling GDMA

Software uses the GDMA controller through linked lists.

For data reception, software loads a receive linked list and prepares to receive data. Configure AHB/AXI_DMA_INLINK_ADDR_CH*n* to point to the first receive descriptor, then set AHB/AXI_DMA_INLINK_START_CH*n* to start GDMA.

For data transmission, software loads a transmit linked list and prepares data to transmit. Configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* to point to the first transmit descriptor, then set AHB/AXI_DMA_OUTLINK_START_CH*n* to start GDMA. Hardware automatically clears AHB/AXI_DMA_INLINK_START_CH*n* and AHB/AXI_DMA_OUTLINK_START_CH*n*.

For GDMA-AXI channels, set AXI_DMA_IN_CMD_DISABLE_CH*n* or AXI_DMA_OUT_CMD_DISABLE_CH*n* to pause data transfer on the corresponding channel.

Sometimes software must append more descriptors after a DMA transfer has already started. One approach is to clear the EOF bit of the last descriptor in the loaded list and set its next descriptor address pointer (DW2) to the first descriptor of the new list. This approach fails if the DMA transfer has already finished or is about to finish.

The GDMA controller provides dedicated logic so that a transfer can continue or restart. If the transfer is still in progress, the controller processes the appended descriptors. If the transfer has finished, the controller restarts with the new descriptors. This logic is implemented by the Restart function.

When using the Restart function, software updates the last descriptor in the loaded list so that DW2 points to the first descriptor of the new list. Software then sets AHB/AXI_DMA_INLINK_RESTART_CH*n* or AHB/AXI_DMA_OUTLINK_RESTART_CH*n* (hardware automatically clears these bits). As shown in Figure 3.4-2, when hardware reads the last descriptor in the loaded list, it obtains the address of the new list and continues processing it.

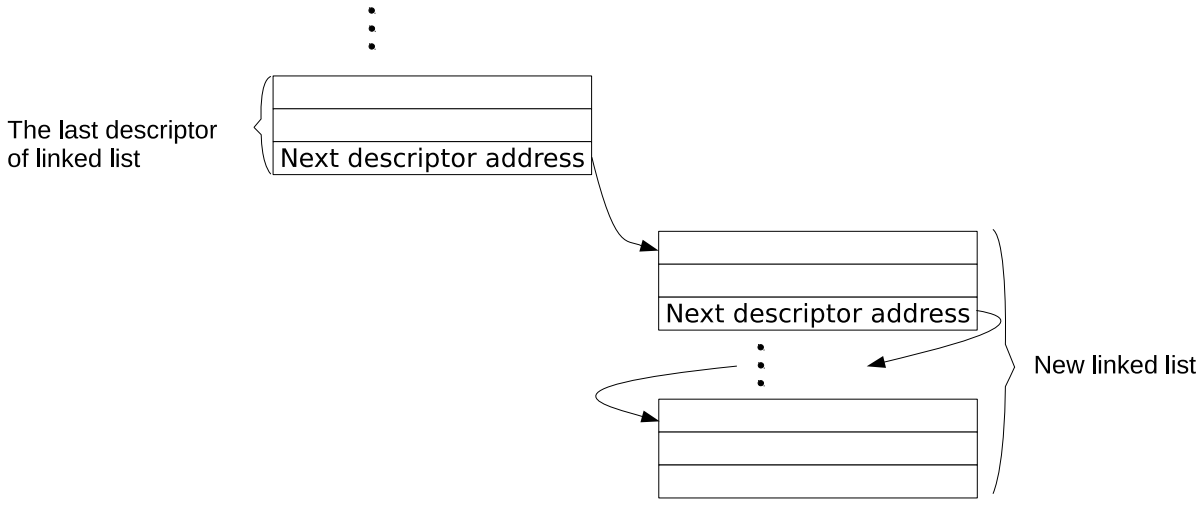


Figure 3.4-2. Relationship among Linked Lists

3.4.5 Linked List Reading Process

After software configures and starts GDMA, GDMA reads the linked list from memory and checks whether the descriptors are valid. A channel starts data transfer only when all descriptors pass the checks. If any descriptor

fails a check, hardware triggers a descriptor error interrupt (AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT or AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT), and the channel stops.

The descriptor checks include:

- Owner bit check when AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the CPU owns the buffer and the check fails. If AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is 0, hardware does not check the owner bit.
- Descriptor address check: the descriptor must lie within the configured memory address space.
  - GDMA-AHB: The descriptor address must be within AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR.
  - GDMA-AXI: The descriptor address must be within AXI_DMA_ACCESS_INTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_INTR_MEM_END_ADDR, or within AXI_DMA_ACCESS_EXTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_EXTR_MEM_END_ADDR.

  See Section 3.4.8 for details.

After software detects a descriptor error interrupt, reset the corresponding channel and set AHB/AXI_DMA_OUTLINK_START_CH*n* or AHB/AXI_DMA_INLINK_START_CH*n* to restart GDMA.

3.4.6 Interrupt Response Mechanism for GDMA-AXI Linked List Switching

GDMA-AXI can generate an interrupt when linked list switching completes. Set AXI_DMA_LINK_SWITCH_STATE_CH*n* to 1 to enable linked list switching. After switching succeeds, if AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT is enabled, GDMA-AXI generates this interrupt and clears AXI_DMA_LINK_SWITCH_STATE_CH*n*, indicating that the linked list has been switched.

This feature serves only as a status indication and does not affect normal GDMA-AXI data transfer.

3.4.7 EOF

Note: In this chapter, EOF of transmit descriptors refers to suc_eof (bit 30 of DW0); EOF of receive descriptors can be suc_eof (bit 30 of DW0) or err_eof (bit 28 of DW0).

GDMA uses end-of-frame (EOF) flags to indicate that the data segment corresponding to a descriptor has been transferred.

For data transmission, the GDMA generates two types of EOF interrupts:

- AHB/AXI_DMA_OUT_EOF_CH*n*_INT: Generated when any descriptor in the linked list has the suc_eof flag set and the data corresponding to that descriptor has been transmitted. This interrupt is enabled by setting AHB/AXI_DMA_OUT_EOF_CH*n*_INT_ENA.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT: Generated when the last descriptor in the linked list has the suc_eof flag set and the data corresponding to that descriptor has been transmitted. This interrupt is enabled by setting AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA.

For data reception, the GDMA also generates two types of EOF interrupts:

- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT: Generated when a data segment with an EOF flag has been received. This interrupt is enabled by setting AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA.

- (UHCI and PARLIO only) AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT: Generated when the data segment corresponding to a descriptor has been received but contains errors. This interrupt is enabled by setting the AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA bit, and is valid only when the channel is connected to UHCI or PARLIO.

When software detects an AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT or AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt, read AHB/AXI_DMA_OUT_EOF_DES_ADDR_CH*n* or AHB/AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* to obtain the address of the descriptor whose transfer has completed. Software can then identify which descriptors have been used and reclaim them.

For RX, the descriptor address can also be recorded when an AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT interrupt is triggered, via AHB/AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n*.

When AHB/AXI_DMA_INLINK_AUTO_RET_CH*n* is set, if the peripheral pushes data with the err_eof flag to GDMA, GDMA returns to the start descriptor address of the current frame and repeats the receive transfer for that frame.

3.4.8 Accessing Memory

The TX/RX channels of GDMA-AHB can access internal and external memory. The accessible address space is determined by AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR.

The TX/RX channels of GDMA-AXI can access:

- Internal memory space configured by AXI_DMA_ACCESS_INTR_MEM_START_ADDR and AXI_DMA_ACCESS_INTR_MEM_END_ADDR;
- External memory space configured by AXI_DMA_ACCESS_EXTR_MEM_START_ADDR and AXI_DMA_ACCESS_EXTR_MEM_END_ADDR.

To accelerate data transfer, GDMA supports burst transfer mode. For GDMA-AHB, burst transfer is disabled by default. Set AHB_DMA_INDSCR_BURST_EN_CH*n* to enable burst transfer on RX channels, and set AHB_DMA_OUTDSCR_BURST_EN_CH*n* to enable burst transfer on TX channels. Select the burst transfer mode via AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* and AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n*. The valid burst transfer modes are:

- 0: SINGLE transfer (4 bytes)
- 1: INCR4 transfer (16 bytes)
- 2: INCR8 transfer (32 bytes)
- 3: INCR16 transfer (64 bytes)

For GDMA-AXI, burst transfer is enabled by default and cannot be disabled. Software can configure the amount of data transferred in each AXI burst via AXI_DMA_IN_BURST_SIZE_SEL_CH*n* and AXI_DMA_OUT_BURST_SIZE_SEL_CH*n*. Valid burst lengths are 8, 16, 32, 64, or 128 bytes.

When GDMA-AHB and GDMA-AXI access the configured internal memory and non-encrypted external memory space, there are no alignment requirements for linked list descriptor parameters.

When accessing the encrypted external memory space (see Chapter 19 *External Memory Encryption and Decryption (XTS_AES) [to be added later]*), the buffer address pointer and corresponding data must be 16-byte aligned; other parameters have no alignment requirements.

Note that encrypted external memory space cannot be accessed by GDMA-AHB-LP.

3.4.9 Arbitration

To ensure timely response to high-speed, low-latency peripherals such as SPI, GDMA implements channel priority arbitration and channel weight arbitration.

- Priority arbitration: Each channel can be assigned a priority from 0 ~ 15. The larger the number, the higher the priority. If multiple channels request data transfer at the same time, GDMA responds to higher-priority channels first.
- Weight arbitration:
  - Each channel is assigned a weight (token count) from 0 ~ 15. GDMA divides the bus (AHB/AXI) clock period into time slots. Within each time slot, each channel may perform as many transfer requests as its token count allows; each request consumes one token.

    If a channel exhausts its tokens within a time slot, GDMA stops responding to its requests until the slot ends or all channels exhaust their tokens. GDMA then refreshes token counts and starts the next time slot.
  - If a channel has a non-zero token count, GDMA waits for all channels to exhaust their tokens within each time slot, even when some channels have no transfer requests. This wastes bandwidth when idle channels hold non-zero token counts.

    GDMA provides a weight arbitration optimization switch. When enabled, channels with no transfer requests are treated as having zero tokens and do not participate in arbitration. When no channel has transfer requests, the arbiter exits the current time slot early, improving response speed and transfer rate.

Regardless of how channel priority is configured, descriptor transfers always have higher priority than data transfers. However, descriptor transfers are still subject to the weight arbitration. Once the configured tokens are exhausted, arbitration can only be requested again in the next time slot.

Note:

- If both priority and weight arbitration are enabled and priorities differ but weights are the same, higher-priority channels finish first when each channel transfers a small and equal amount of data.
- If both priority and weight differ and each channel transfers a large and equal amount of data, channels with larger weight receive more bandwidth and finish first.

3.4.10 Bus Error Response Information Logging

During GDMA-AHB data transfer, if a slave on the bus returns an error response, the data transfer is not interrupted; GDMA-AHB continues to transfer data according to the existing linked list. As a result, the actual data received after transfer may not match the expected data. To facilitate troubleshooting, if an error response occurs during GDMA-AHB data transfer, the following internal registers record relevant transfer information:

- AHB_DMA_AHBINF_RESP_ERR_ADDR: Records the 32-bit address corresponding to the current bus transfer.

- AHB_DMA_AHBINF_RESP_ERR_CH_ID: Records the GDMA-AHB channel used for the current bus transfer.
- AHB_DMA_AHBINF_RESP_ERR_ID: Records the transfer ID for the current peripheral-to-memory, memory-to-peripheral, or memory-to-memory transfer on the bus.
- AHB_DMA_AHBINF_RESP_ERR_WR: Records the type of transfer for the current bus transfer.

When GDMA-AHB initiates an AHB bus transfer as the master and receives an error response, it also generates an error response interrupt to notify the CPU. After the interrupt is generated, GDMA-AHB does not interrupt the data transfer.

In addition, GDMA-AHB does not latch register information for the first AHB bus transfer that triggers an error response. If multiple consecutive error response transfers occur, the error response logging registers are updated in real time; the registers retain information about the last AHB bus transfer that triggered an error response before the registers are read.

3.4.11 Automatic Clock Gating

When GDMA-AHB is operating, some scenarios require only part of the channels; in this case, clocks for unused channels can be disabled to reduce power consumption. Similarly, the internal arbitration and bus interface modules need to operate only during data transfer, and their clocks can be disabled during idle periods to save power. Currently, GDMA-AHB has the following internal modules:

- Bus interface processing module
- Arbitration module
- RX data processing module
- RX descriptor processing module
- TX data processing module
- TX descriptor processing module
- Register configuration synchronization module

Clocks of the above modules are automatically disabled when idle to minimize power consumption. These clocks can also be forced on via register configuration. For example, setting AHB_DMA_AHBINF_CLK_EN keeps the bus interface processing module clock always on; configurations for keeping other module clocks always on are similar.

3.5 Event Task Matrix Feature

The GDMA controller on ESP32-S31 supports the Event Task Matrix (ETM) function, which allows GDMA's ETM tasks to be triggered by any of the peripherals' ETM events, or GDMA's ETM events to trigger any of the 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 (ETM) [to be added later].

GDMA can receive the following ETM tasks:

- GDMA_AHB/AXI_TASK_IN_START_CH*n*: When triggered, starts data transfer on GDMA receive channel *n*.

- GDMA_AHB/AXI_TASK_OUT_START_CH*n*: When triggered, starts data transfer on GDMA transmit channel *n*.

Note:
The above two ETM tasks provide the same functionality as setting AHB/AXI_DMA_INLINK_START_CH*n* and AHB/AXI_DMA_OUTLINK_START_CH*n* via the CPU.

When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB/AXI_DMA_OUT_ETM_EN_CH*n* is 1, only ETM can start GDMA in the corresponding direction and channel. When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB/AXI_DMA_OUT_ETM_EN_CH*n* is 0, only the CPU can enable the corresponding GDMA channel.

GDMA can generate the following ETM events:

- GDMA_AHB/AXI_EVT_IN_DONE_CH*n*: Indicates that data pointed to by a receive descriptor on channel *n* has been received.
- GDMA_AHB/AXI_EVT_IN_SUC_EOF_CH*n*: Indicates that data pointed to by a receive descriptor with EOF bit set to 1 on channel *n* has been received.
- GDMA_AHB/AXI_EVT_IN_FIFO_EMPTY_CH*n*: Indicates that the RX FIFO on channel *n* changed from non-empty to empty.
- GDMA_AHB/AXI_EVT_IN_FIFO_FULL_CH*n*: Indicates that the RX FIFO on channel *n* changed from non-full to full.
- GDMA_AHB/AXI_EVT_OUT_DONE_CH*n*: Indicates that data pointed to by a transmit descriptor on channel *n* has been transmitted.
- GDMA_AHB/AXI_EVT_OUT_SUC_EOF_CH*n*: Indicates that data pointed to by a transmit descriptor with EOF bit set to 1 on channel *n* has been transmitted.
- GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that data pointed to by the last transmit descriptor with EOF bit set to 1 on channel *n* has been transmitted.
- GDMA_AHB/AXI_EVT_OUT_FIFO_EMPTY_CH*n*: Indicates that the TX FIFO on channel *n* changed from non-empty to empty.
- GDMA_AHB/AXI_EVT_OUT_FIFO_FULL_CH*n*: Indicates that the TX FIFO on channel *n* changed from non-full to full.

In practical applications, GDMA's ETM events can trigger GDMA's own ETM tasks. For example, the GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH0 event can trigger the GDMA_AHB/AXI_TASK_IN_START_CH1 task to start a new round of GDMA operations.

3.6 Interrupts

ESP32-S31's GDMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- GDMA-AHB-HP:
  - AHB_PDMA_IN/OUT_CH0_INTR
  - AHB_PDMA_IN/OUT_CH1_INTR

  - AHB_PDMA_IN/OUT_CH2_INTR
  - AHB_PDMA_IN/OUT_CH3_INTR
  - AHB_PDMA_IN/OUT_CH4_INTR
- GDMA-AHB-LP:
  - AHB_PDMA_IN/OUT_CH0_INTR
  - AHB_PDMA_IN/OUT_CH1_INTR
- GDMA-AXI:
  - AXI_PDMA_IN/OUT_CH0_INTR
  - AXI_PDMA_IN/OUT_CH1_INTR
  - AXI_PDMA_IN/OUT_CH2_INTR

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

The AHB/AXI_PDMA_IN_CH*n*_INTR signals are generated by the following interrupt sources:

- AHB/AXI_DMA_IN_DONE_CH*n*_INT: For receive channel *n*, triggered when data corresponding to a receive descriptor has been received.
- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT: For receive channel *n*, triggered when data corresponding to a receive descriptor has been received and the descriptor's suc_eof is 1.
- AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT: For receive channel *n*, triggered when an error occurs in the data segment corresponding to a received descriptor. (This interrupt is valid only when the peripheral is UHCI or PARLIO.)
- AHB/AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT: For receive channel *n*, triggered when the buffer size pointed to by a receive descriptor is smaller than the length of data to be received.
- AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT: For receive channel *n*, triggered when a receive descriptor fails any descriptor check.
- AHB_DMA_INFIFO_OVF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the GDMA-AHB receive buffer or GDMA-AXI receive L1, L2, or L3 buffer overflows.
- AHB/AXI_DMA_INFIFO_UDF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the GDMA-AHB receive buffer or GDMA-AXI receive L1, L2, or L3 buffer has no data.
- AHB_DMA_IN_AHBINF_RESP_ERR_CH*n*_INT: Triggered when GDMA-AHB receive channel *n* performs a data transfer and an error response occurs on the AHB bus.

The AHB/AXI_PDMA_OUT_CH*n*_INTR signals are generated by the following interrupt sources:

- AHB/AXI_DMA_OUT_DONE_CH*n*_INT: For transmit channel *n*, triggered when data corresponding to a transmit descriptor has been sent.
- AHB/AXI_DMA_OUT_EOF_CH*n*_INT: For transmit channel *n*, triggered when the EOF bit of a transmit descriptor is 1. The trigger timing depends on AHB/AXI_DMA_OUT_EOF_MODE_CH*n*:
  - When set to 0: triggered when the last byte of data corresponding to this descriptor enters the GDMA TX channel from the bus;

  - When set to 1: triggered when the last byte of data corresponding to this descriptor leaves the GDMA TX channel.

- AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT: For transmit channel *n*, triggered when a transmit descriptor fails any descriptor check.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT: For transmit channel *n*, triggered when the last descriptor in the linked list has suc_eof configured and the data corresponding to that descriptor has been transmitted.
- AHB_DMA_OUTFIFO_OVF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the GDMA-AHB transmit buffer or GDMA-AXI transmit L1, L2, or L3 buffer overflows.
- AHB_DMA_OUTFIFO_UDF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the GDMA-AHB transmit buffer or GDMA-AXI transmit L1, L2, or L3 buffer has no data.
- AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT: Triggered when GDMA-AXI transmit channel *n* switches from one linked list to the next.
- AHB_DMA_OUT_AHBINF_RESP_ERR_CH*n*_INT: Triggered when GDMA-AHB transmit channel *n* performs a data transfer and an error response occurs on the AHB bus.

3.7 Programming Procedures

Configure GDMA clock gating via HP_SYS_CLKRST_AHB/AXI_PDMA_SYS_CLK_EN; clock gating is enabled by default. Reset GDMA by configuring HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA.

3.7.1 Programming Procedures for GDMA's Transmit Channel

To transmit data using GDMA, configure the GDMA TX channel as follows:

1. Set AHB/AXI_DMA_OUT_RST_CH*n* to 1 and then to 0 to reset the GDMA TX channel state machine and FIFO pointer.
2. Load a transmit linked list and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* to point to the first transmit descriptor.
3. Configure AHB/AXI_DMA_PERI_OUT_SEL_CH*n* to the peripheral number, as shown in Table 3.4-1, Table 3.4-2, and Table 3.4-3.
4. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to start GDMA TX channel data transfer.
5. Configure and enable the corresponding peripheral. See the corresponding peripheral chapter for details.
6. Wait for the AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT interrupt, which indicates that data transfer is complete.

3.7.2 Programming Procedures for GDMA's Receive Channel

To receive data using GDMA, configure the GDMA RX channel as follows:

1. Set AHB/AXI_DMA_IN_RST_CH*n* to 1 and then to 0 to reset the GDMA RX channel state machine and FIFO pointer.

2. Load a receive linked list and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* to point to the first receive descriptor.
3. Configure AHB/AXI_DMA_PERI_IN_SEL_CH*n* to the peripheral number, as shown in Table 3.4-1, Table 3.4-2, and Table 3.4-3.
4. Set AHB/AXI_DMA_INLINK_START_CH*n* to start GDMA RX channel data transfer.
5. Configure and enable the corresponding peripheral. See the corresponding peripheral chapter for details.

3.7.3 Programming Procedures for Memory-to-Memory Transfer

To transfer data from one memory location to another using GDMA, configure as follows:

1. Set AHB/AXI_DMA_OUT_RST_CH*n* to 1 and then to 0 to reset the GDMA TX channel state machine and FIFO pointer.
2. Set AHB/AXI_DMA_IN_RST_CH*n* to 1 and then to 0 to reset the GDMA RX channel state machine and FIFO pointer.
3. Load a transmit linked list and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* to point to the first transmit descriptor.
4. Load a receive linked list and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* to point to the first receive descriptor.
5. Set AHB/AXI_DMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
6. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to start GDMA TX channel data transfer.
7. Set AHB/AXI_DMA_INLINK_START_CH*n* to start GDMA RX channel data transfer.
8. If the suc_eof bit is set in a transmit descriptor, an AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt is triggered when the data segment corresponding to that descriptor has been transmitted.

3.7.4 Programming Procedures for Channel Priority and Weight

Priority arbitration can be configured as follows:

1. Configure AHB/AXI_DMA_TX_PRI_CH*n* and AHB/AXI_DMA_RX_PRI_CH*n* as the priority for TX and RX channels.

Weight arbitration can be configured as follows:

1. Configure the weight arbitration time slot length:
   - GDMA-AHB: Configure AHB_DMA_ARB_TIMEOUT, which applies to both TX and RX directions;
   - GDMA-AXI: Configure TX and RX directions independently via AXI_DMA_ARB_TIMEOUT_TX and AXI_DMA_ARB_TIMEOUT_RX.
2. Configure AHB/AXI_DMA_TX_CH_ARB_WEIGH_CH*n* and AHB/AXI_DMA_RX_CH_ARB_WEIGH_CH*n* as the token count for each channel.
3. Clear AHB/AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* and AHB/AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* to enable weight arbitration optimization for TX and RX directions.

4. Enable weight arbitration:
   - GDMA-AHB: Set AHB_DMA_WEIGHT_EN to enable both TX and RX directions;
   - GDMA-AXI: Set AXI_DMA_WEIGHT_EN_TX and AXI_DMA_WEIGHT_EN_RX to enable TX and RX directions respectively.

3.8 Register Summary

3.8.1 GDMA-AHB Register Summary

3.8.1.1 GDMA-AHB-HP Register Summary

The addresses in this section are relative to GDMA-AHB-HP base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Interrupt Registers			
AHB_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
AHB_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
AHB_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
AHB_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0010	R/WTC/SS
AHB_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX channel 2 raw interrupt status register	0x0020	R/WTC/SS
AHB_DMA_IN_INT_ST_CH2_REG	RX channel 2 masked interrupt status register	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX channel 2 interrupt enable register	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX channel 2 interrupt clear register	0x002C	WT
AHB_DMA_IN_INT_RAW_CH3_REG	RX channel 3 raw interrupt status register	0x0030	R/WTC/SS
AHB_DMA_IN_INT_ST_CH3_REG	RX channel 3 masked interrupt status register	0x0034	RO
AHB_DMA_IN_INT_ENA_CH3_REG	RX channel 3 interrupt enable register	0x0038	R/W
AHB_DMA_IN_INT_CLR_CH3_REG	RX channel 3 interrupt clear register	0x003C	WT
AHB_DMA_IN_INT_RAW_CH4_REG	RX channel 4 raw interrupt status register	0x0040	R/WTC/SS
AHB_DMA_IN_INT_ST_CH4_REG	RX channel 4 masked interrupt status register	0x0044	RO
AHB_DMA_IN_INT_ENA_CH4_REG	RX channel 4 interrupt enable register	0x0048	R/W

Name	Description	Address	Access
AHB_DMA_IN_INT_CLR_CH4_REG	RX channel 4 interrupt clear register	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0050	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x005C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x0060	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x0064	RO
AHB_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x0068	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x006C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX channel 2 raw interrupt status register	0x0070	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH2_REG	TX channel 2 masked interrupt status register	0x0074	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX channel 2 interrupt enable register	0x0078	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX channel 2 interrupt clear register	0x007C	WT
AHB_DMA_OUT_INT_RAW_CH3_REG	TX channel 3 raw interrupt status register	0x0080	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH3_REG	TX channel 3 masked interrupt status register	0x0084	RO
AHB_DMA_OUT_INT_ENA_CH3_REG	TX channel 3 interrupt enable register	0x0088	R/W
AHB_DMA_OUT_INT_CLR_CH3_REG	TX channel 3 interrupt clear register	0x008C	WT
AHB_DMA_OUT_INT_RAW_CH4_REG	TX channel 4 raw interrupt status register	0x0090	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH4_REG	TX channel 4 masked interrupt status register	0x0094	RO
AHB_DMA_OUT_INT_ENA_CH4_REG	TX channel 4 interrupt enable register	0x0098	R/W
AHB_DMA_OUT_INT_CLR_CH4_REG	TX channel 4 interrupt clear register	0x009C	WT
Debug Registers			
AHB_DMA_AHB_TEST_REG	Reserved register	0x00A0	R/W
Configuration Registers			
AHB_DMA_MISC_CONF_REG	Miscellaneous configuration register	0x00A4	R/W
AHB_DMA_IN_CONF0_CH0_REG	RX channel 0 configuration 0 register	0x0100	R/W
AHB_DMA_IN_CONF1_CH0_REG	RX channel 0 configuration 1 register	0x0104	R/W
AHB_DMA_IN_POP_CH0_REG	RX channel 0 FIFO pop register	0x010C	varies
AHB_DMA_IN_LINK_CH0_REG	RX channel 0 descriptor control register	0x0110	varies

Name	Description	Address	Access
AHB_DMA_IN_LINK_ADDR_CH0_REG	RX channel 0 descriptor address register	0x0114	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX channel 0 arbitration weight register	0x013C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX channel 0 weight optimization register	0x0140	R/W
AHB_DMA_OUT_CONF0_CH0_REG	TX channel 0 configuration 0 register	0x0180	R/W
AHB_DMA_OUT_CONF1_CH0_REG	TX channel 0 configuration 1 register	0x0184	R/W
AHB_DMA_OUT_PUSH_CH0_REG	TX channel 0 FIFO push register	0x018C	varies
AHB_DMA_OUT_LINK_CH0_REG	TX channel 0 descriptor control register	0x0190	varies
AHB_DMA_OUT_LINK_ADDR_CH0_REG	TX channel 0 descriptor address register	0x0194	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX channel 0 arbitration weight register	0x01BC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX channel 0 weight optimization register	0x01C0	R/W
AHB_DMA_IN_CONF0_CH1_REG	RX channel 1 configuration 0 register	0x0200	R/W
AHB_DMA_IN_CONF1_CH1_REG	RX channel 1 configuration 1 register	0x0204	R/W
AHB_DMA_IN_POP_CH1_REG	RX channel 1 FIFO pop register	0x020C	varies
AHB_DMA_IN_LINK_CH1_REG	RX channel 1 descriptor control register	0x0210	varies
AHB_DMA_IN_LINK_ADDR_CH1_REG	RX channel 1 descriptor address register	0x0214	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX channel 1 arbitration weight register	0x023C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX channel 1 weight optimization register	0x0240	R/W
AHB_DMA_OUT_CONF0_CH1_REG	TX channel 1 configuration 0 register	0x0280	R/W
AHB_DMA_OUT_CONF1_CH1_REG	TX channel 1 configuration 1 register	0x0284	R/W
AHB_DMA_OUT_PUSH_CH1_REG	TX channel 1 FIFO push register	0x028C	varies
AHB_DMA_OUT_LINK_CH1_REG	TX channel 1 descriptor control register	0x0290	varies
AHB_DMA_OUT_LINK_ADDR_CH1_REG	TX channel 1 descriptor address register	0x0294	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX channel 1 arbitration weight register	0x02BC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX channel 1 weight optimization register	0x02C0	R/W
AHB_DMA_IN_CONF0_CH2_REG	RX channel 2 configuration 0 register	0x0300	R/W
AHB_DMA_IN_CONF1_CH2_REG	RX channel 2 configuration 1 register	0x0304	R/W
AHB_DMA_IN_POP_CH2_REG	RX channel 2 FIFO pop register	0x030C	varies
AHB_DMA_IN_LINK_CH2_REG	RX channel 2 descriptor control register	0x0310	varies
AHB_DMA_IN_LINK_ADDR_CH2_REG	RX channel 2 descriptor address register	0x0314	R/W

Name	Description	Address	Access
AHB_DMA_RX_CH_ARB_WEIGH_CH2_REG	RX channel 2 arbitration weight register	0x033C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH2_REG	RX channel 2 weight optimization register	0x0340	R/W
AHB_DMA_OUT_CONF0_CH2_REG	TX channel 2 configuration 0 register	0x0380	R/W
AHB_DMA_OUT_CONF1_CH2_REG	TX channel 2 configuration 1 register	0x0384	R/W
AHB_DMA_OUT_PUSH_CH2_REG	TX channel 2 FIFO push register	0x038C	varies
AHB_DMA_OUT_LINK_CH2_REG	TX channel 2 descriptor control register	0x0390	varies
AHB_DMA_OUT_LINK_ADDR_CH2_REG	TX channel 2 descriptor address register	0x0394	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH2_REG	TX channel 2 arbitration weight register	0x03BC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH2_REG	TX channel 2 weight optimization register	0x03C0	R/W
AHB_DMA_IN_CONF0_CH3_REG	RX channel 3 configuration 0 register	0x0400	R/W
AHB_DMA_IN_CONF1_CH3_REG	RX channel 3 configuration 1 register	0x0404	R/W
AHB_DMA_IN_POP_CH3_REG	RX channel 3 FIFO pop register	0x040C	varies
AHB_DMA_IN_LINK_CH3_REG	RX channel 3 descriptor control register	0x0410	varies
AHB_DMA_IN_LINK_ADDR_CH3_REG	RX channel 3 descriptor address register	0x0414	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH3_REG	RX channel 3 arbitration weight register	0x043C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH3_REG	RX channel 3 weight optimization register	0x0440	R/W
AHB_DMA_OUT_CONF0_CH3_REG	TX channel 3 configuration 0 register	0x0480	R/W
AHB_DMA_OUT_CONF1_CH3_REG	TX channel 3 configuration 1 register	0x0484	R/W
AHB_DMA_OUT_PUSH_CH3_REG	TX channel 3 FIFO push register	0x048C	varies
AHB_DMA_OUT_LINK_CH3_REG	TX channel 3 descriptor control register	0x0490	varies
AHB_DMA_OUT_LINK_ADDR_CH3_REG	TX channel 3 descriptor address register	0x0494	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH3_REG	TX channel 3 arbitration weight register	0x04BC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH3_REG	TX channel 3 weight optimization register	0x04C0	R/W
AHB_DMA_IN_CONF0_CH4_REG	RX channel 4 configuration 0 register	0x0500	R/W
AHB_DMA_IN_CONF1_CH4_REG	RX channel 4 configuration 1 register	0x0504	R/W
AHB_DMA_IN_POP_CH4_REG	RX channel 4 FIFO pop register	0x050C	varies
AHB_DMA_IN_LINK_CH4_REG	RX channel 4 descriptor control register	0x0510	varies
AHB_DMA_IN_LINK_ADDR_CH4_REG	RX channel 4 descriptor address register	0x0514	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH4_REG	RX channel 4 arbitration weight register	0x053C	R/W

Name	Description	Address	Access
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH4_REG	RX channel 4 weight optimization register	0x0540	R/W
AHB_DMA_OUT_CONF0_CH4_REG	TX channel 4 configuration 0 register	0x0580	R/W
AHB_DMA_OUT_CONF1_CH4_REG	TX channel 4 configuration 1 register	0x0584	R/W
AHB_DMA_OUT_PUSH_CH4_REG	TX channel 4 FIFO push register	0x058C	varies
AHB_DMA_OUT_LINK_CH4_REG	TX channel 4 descriptor control register	0x0590	varies
AHB_DMA_OUT_LINK_ADDR_CH4_REG	TX channel 4 descriptor address register	0x0594	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH4_REG	TX channel 4 arbitration weight register	0x05BC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH4_REG	TX channel 4 weight optimization register	0x05C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	Accessible start address register	0x0600	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	Accessible end address register	0x0604	R/W
AHB_DMA_ARB_TIMEOUT_REG	Arbitration timeout register	0x0608	R/W
AHB_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0610	R/W
AHB_DMA_MODULE_CLK_EN_REG	Module clock force on register	0x0618	R/W
Version Registers			
AHB_DMA_DATE_REG	Version control register	0x00A8	R/W
Status Registers			
AHB_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status register	0x0108	RO
AHB_DMA_IN_STATE_CH0_REG	RX channel 0 status register	0x0118	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX channel 0 successful EOF address register	0x011C	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX channel 0 error EOF address register	0x0120	RO
AHB_DMA_IN_DONE_DES_ADDR_CH0_REG	RX channel 0 done descriptor address register	0x0124	RO
AHB_DMA_IN_DSCR_CH0_REG	RX channel 0 next descriptor address register	0x0128	RO
AHB_DMA_IN_DSCR_BF0_CH0_REG	RX channel 0 current descriptor address register	0x012C	RO
AHB_DMA_IN_DSCR_BF1_CH0_REG	RX channel 0 previous descriptor address register	0x0130	RO
AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status register	0x0188	RO
AHB_DMA_OUT_STATE_CH0_REG	TX channel 0 status register	0x0198	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX channel 0 EOF address register	0x019C	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX channel 0 previous EOF address register	0x01A0	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG	TX channel 0 done descriptor address register	0x01A4	RO

GoBack

Name	Description	Address	Access
AHB_DMA_OUT_DSCR_CH0_REG	TX channel 0 next descriptor address register	0x01A8	RO
AHB_DMA_OUT_DSCR_BF0_CH0_REG	TX channel 0 current descriptor address register	0x01AC	RO
AHB_DMA_OUT_DSCR_BF1_CH0_REG	TX channel 0 previous descriptor address register	0x01B0	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status register	0x0208	RO
AHB_DMA_IN_STATE_CH1_REG	RX channel 1 status register	0x0218	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX channel 1 successful EOF address register	0x021C	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX channel 1 error EOF address register	0x0220	RO
AHB_DMA_IN_DONE_DES_ADDR_CH1_REG	RX channel 1 done descriptor address register	0x0224	RO
AHB_DMA_IN_DSCR_CH1_REG	RX channel 1 next descriptor address register	0x0228	RO
AHB_DMA_IN_DSCR_BF0_CH1_REG	RX channel 1 current descriptor address register	0x022C	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	RX channel 1 previous descriptor address register	0x0230	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 1 FIFO status register	0x0288	RO
AHB_DMA_OUT_STATE_CH1_REG	TX channel 1 status register	0x0298	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX channel 1 EOF address register	0x029C	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX channel 1 previous EOF address register	0x02A0	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG	TX channel 1 done descriptor address register	0x02A4	RO
AHB_DMA_OUT_DSCR_CH1_REG	TX channel 1 next descriptor address register	0x02A8	RO
AHB_DMA_OUT_DSCR_BF0_CH1_REG	TX channel 1 current descriptor address register	0x02AC	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	TX channel 1 previous descriptor address register	0x02B0	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status register	0x0308	RO
AHB_DMA_IN_STATE_CH2_REG	RX channel 2 status register	0x0318	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX channel 2 successful EOF address register	0x031C	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX channel 2 error EOF address register	0x0320	RO
AHB_DMA_IN_DONE_DES_ADDR_CH2_REG	RX channel 2 done descriptor address register	0x0324	RO
AHB_DMA_IN_DSCR_CH2_REG	RX channel 2 next descriptor address register	0x0328	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	RX channel 2 current descriptor address register	0x032C	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	RX channel 2 previous descriptor address register	0x0330	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status register	0x0388	RO
AHB_DMA_OUT_STATE_CH2_REG	TX channel 2 status register	0x0398	RO

Name	Description	Address	Access
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX channel 2 EOF address register	0x039C	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX channel 2 previous EOF address register	0x03A0	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH2_REG	TX channel 2 done descriptor address register	0x03A4	RO
AHB_DMA_OUT_DSCR_CH2_REG	TX channel 2 next descriptor address register	0x03A8	RO
AHB_DMA_OUT_DSCR_BF0_CH2_REG	TX channel 2 current descriptor address register	0x03AC	RO
AHB_DMA_OUT_DSCR_BF1_CH2_REG	TX channel 2 previous descriptor address register	0x03B0	RO
AHB_DMA_INFIFO_STATUS_CH3_REG	RX channel 3 FIFO status register	0x0408	RO
AHB_DMA_IN_STATE_CH3_REG	RX channel 3 status register	0x0418	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH3_REG	RX channel 3 successful EOF address register	0x041C	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH3_REG	RX channel 3 error EOF address register	0x0420	RO
AHB_DMA_IN_DONE_DES_ADDR_CH3_REG	RX channel 3 done descriptor address register	0x0424	RO
AHB_DMA_IN_DSCR_CH3_REG	RX channel 3 next descriptor address register	0x0428	RO
AHB_DMA_IN_DSCR_BF0_CH3_REG	RX channel 3 current descriptor address register	0x042C	RO
AHB_DMA_IN_DSCR_BF1_CH3_REG	RX channel 3 previous descriptor address register	0x0430	RO
AHB_DMA_OUTFIFO_STATUS_CH3_REG	TX channel 3 FIFO status register	0x0488	RO
AHB_DMA_OUT_STATE_CH3_REG	TX channel 3 status register	0x0498	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH3_REG	TX channel 3 EOF address register	0x049C	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH3_REG	TX channel 3 previous EOF address register	0x04A0	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH3_REG	TX channel 3 done descriptor address register	0x04A4	RO
AHB_DMA_OUT_DSCR_CH3_REG	TX channel 3 next descriptor address register	0x04A8	RO
AHB_DMA_OUT_DSCR_BF0_CH3_REG	TX channel 3 current descriptor address register	0x04AC	RO
AHB_DMA_OUT_DSCR_BF1_CH3_REG	TX channel 3 previous descriptor address register	0x04B0	RO
AHB_DMA_INFIFO_STATUS_CH4_REG	RX channel 4 FIFO status register	0x0508	RO
AHB_DMA_IN_STATE_CH4_REG	RX channel 4 status register	0x0518	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH4_REG	RX channel 4 successful EOF address register	0x051C	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH4_REG	RX channel 4 error EOF address register	0x0520	RO
AHB_DMA_IN_DONE_DES_ADDR_CH4_REG	RX channel 4 done descriptor address register	0x0524	RO
AHB_DMA_IN_DSCR_CH4_REG	RX channel 4 next descriptor address register	0x0528	RO
AHB_DMA_IN_DSCR_BF0_CH4_REG	RX channel 4 current descriptor address register	0x052C	RO

Name	Description	Address	Access
AHB_DMA_IN_DSCR_BF1_CH4_REG	RX channel 4 previous descriptor address register	0x0530	RO
AHB_DMA_OUTFIFO_STATUS_CH4_REG	TX channel 4 FIFO status register	0x0588	RO
AHB_DMA_OUT_STATE_CH4_REG	TX channel 4 status register	0x0598	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH4_REG	TX channel 4 EOF address register	0x059C	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH4_REG	TX channel 4 previous EOF address register	0x05A0	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH4_REG	TX channel 4 done descriptor address register	0x05A4	RO
AHB_DMA_OUT_DSCR_CH4_REG	TX channel 4 next descriptor address register	0x05A8	RO
AHB_DMA_OUT_DSCR_BF0_CH4_REG	TX channel 4 current descriptor address register	0x05AC	RO
AHB_DMA_OUT_DSCR_BF1_CH4_REG	TX channel 4 previous descriptor address register	0x05B0	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG	AHB response error status register 0	0x0620	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG	AHB response error status register 1	0x0624	RO
Priority Registers			
AHB_DMA_IN_PRI_CH0_REG	RX channel 0 priority register	0x0134	R/W
AHB_DMA_OUT_PRI_CH0_REG	TX channel 0 priority register	0x01B4	R/W
AHB_DMA_IN_PRI_CH1_REG	RX channel 1 priority register	0x0234	R/W
AHB_DMA_OUT_PRI_CH1_REG	TX channel 1 priority register	0x02B4	R/W
AHB_DMA_IN_PRI_CH2_REG	RX channel 2 priority register	0x0334	R/W
AHB_DMA_OUT_PRI_CH2_REG	TX channel 2 priority register	0x03B4	R/W
AHB_DMA_IN_PRI_CH3_REG	RX channel 3 priority register	0x0434	R/W
AHB_DMA_OUT_PRI_CH3_REG	TX channel 3 priority register	0x04B4	R/W
AHB_DMA_IN_PRI_CH4_REG	RX channel 4 priority register	0x0534	R/W
AHB_DMA_OUT_PRI_CH4_REG	TX channel 4 priority register	0x05B4	R/W
Peripheral Select Registers			
AHB_DMA_IN_PERI_SEL_CH0_REG	RX channel 0 peripheral selection register	0x0138	R/W
AHB_DMA_OUT_PERI_SEL_CH0_REG	TX channel 0 peripheral selection register	0x01B8	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	RX channel 1 peripheral selection register	0x0238	R/W
AHB_DMA_OUT_PERI_SEL_CH1_REG	TX channel 1 peripheral selection register	0x02B8	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	RX channel 2 peripheral selection register	0x0338	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	TX channel 2 peripheral selection register	0x03B8	R/W

Name	Description	Address	Access
AHB_DMA_IN_PERI_SEL_CH3_REG	RX channel 3 peripheral selection register	0x0438	R/W
AHB_DMA_OUT_PERI_SEL_CH3_REG	TX channel 3 peripheral selection register	0x04B8	R/W
AHB_DMA_IN_PERI_SEL_CH4_REG	RX channel 4 peripheral selection register	0x0538	R/W
AHB_DMA_OUT_PERI_SEL_CH4_REG	TX channel 4 peripheral selection register	0x05B8	R/W

3.8.1.2 GDMA-AHB-LP Register Summary

The addresses in this section are relative to GDMA-AHB-LP base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Interrupt Registers			
LP_AHB_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
LP_AHB_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
LP_AHB_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
LP_AHB_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
LP_AHB_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0010	R/WTC/SS
LP_AHB_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x0014	RO
LP_AHB_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0018	R/W
LP_AHB_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x001C	WT
LP_AHB_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0050	R/WTC/SS
LP_AHB_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x0054	RO
LP_AHB_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0058	R/W
LP_AHB_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x005C	WT
LP_AHB_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x0060	R/WTC/SS
LP_AHB_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x0064	RO
LP_AHB_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x0068	R/W
LP_AHB_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x006C	WT
Debug Registers			

GoBack

Name	Description	Address	Access
LP_AHB_DMA_AHB_TEST_REG	Reserved register	0x00A0	R/W
Configuration Registers			
LP_AHB_DMA_MISC_CONF_REG	Miscellaneous configuration register	0x00A4	R/W
LP_AHB_DMA_IN_CONF0_CH0_REG	RX channel 0 configuration 0 register	0x0100	R/W
LP_AHB_DMA_IN_CONF1_CH0_REG	RX channel 0 configuration 1 register	0x0104	R/W
LP_AHB_DMA_IN_POP_CH0_REG	RX channel 0 FIFO pop register	0x010C	varies
LP_AHB_DMA_IN_LINK_CH0_REG	RX channel 0 descriptor control register	0x0110	varies
LP_AHB_DMA_IN_LINK_ADDR_CH0_REG	RX channel 0 descriptor address register	0x0114	R/W
LP_AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX channel 0 arbitration weight register	0x013C	R/W
LP_AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX channel 0 weight optimization register	0x0140	R/W
LP_AHB_DMA_OUT_CONF0_CH0_REG	TX channel 0 configuration 0 register	0x0180	R/W
LP_AHB_DMA_OUT_CONF1_CH0_REG	TX channel 0 configuration 1 register	0x0184	R/W
LP_AHB_DMA_OUT_PUSH_CH0_REG	TX channel 0 FIFO push register	0x018C	varies
LP_AHB_DMA_OUT_LINK_CH0_REG	TX channel 0 descriptor control register	0x0190	varies
LP_AHB_DMA_OUT_LINK_ADDR_CH0_REG	TX channel 0 descriptor address register	0x0194	R/W
LP_AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX channel 0 arbitration weight register	0x01BC	R/W
LP_AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX channel 0 weight optimization register	0x01C0	R/W
LP_AHB_DMA_IN_CONF0_CH1_REG	RX channel 1 configuration 0 register	0x0200	R/W
LP_AHB_DMA_IN_CONF1_CH1_REG	RX channel 1 configuration 1 register	0x0204	R/W
LP_AHB_DMA_IN_POP_CH1_REG	RX channel 1 FIFO pop register	0x020C	varies
LP_AHB_DMA_IN_LINK_CH1_REG	RX channel 1 descriptor control register	0x0210	varies
LP_AHB_DMA_IN_LINK_ADDR_CH1_REG	RX channel 1 descriptor address register	0x0214	R/W
LP_AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX channel 1 arbitration weight register	0x023C	R/W
LP_AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX channel 1 weight optimization register	0x0240	R/W
LP_AHB_DMA_OUT_CONF0_CH1_REG	TX channel 1 configuration 0 register	0x0280	R/W
LP_AHB_DMA_OUT_CONF1_CH1_REG	TX channel 1 configuration 1 register	0x0284	R/W
LP_AHB_DMA_OUT_PUSH_CH1_REG	TX channel 1 FIFO push register	0x028C	varies
LP_AHB_DMA_OUT_LINK_CH1_REG	TX channel 1 descriptor control register	0x0290	varies
LP_AHB_DMA_OUT_LINK_ADDR_CH1_REG	TX channel 1 descriptor address register	0x0294	R/W

Name	Description	Address	Access
LP_AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX channel 1 arbitration weight register	0x02BC	R/W
LP_AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX channel 1 weight optimization register	0x02C0	R/W
LP_AHB_DMA_INTR_MEM_START_ADDR_REG	Accessible start address register	0x0600	R/W
LP_AHB_DMA_INTR_MEM_END_ADDR_REG	Accessible end address register	0x0604	R/W
LP_AHB_DMA_ARB_TIMEOUT_REG	Arbitration timeout register	0x0608	R/W
LP_AHB_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0610	R/W
LP_AHB_DMA_MODULE_CLK_EN_REG	Module clock force on register	0x0618	R/W
Version Registers			
LP_AHB_DMA_DATE_REG	Version control register	0x00A8	R/W
Status Registers			
LP_AHB_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status register	0x0108	RO
LP_AHB_DMA_IN_STATE_CH0_REG	RX channel 0 status register	0x0118	RO
LP_AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX channel 0 successful EOF address register	0x011C	RO
LP_AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX channel 0 error EOF address register	0x0120	RO
LP_AHB_DMA_IN_DONE_DES_ADDR_CH0_REG	RX channel 0 done descriptor address register	0x0124	RO
LP_AHB_DMA_IN_DSCR_CH0_REG	RX channel 0 next descriptor address register	0x0128	RO
LP_AHB_DMA_IN_DSCR_BF0_CH0_REG	RX channel 0 current descriptor address register	0x012C	RO
LP_AHB_DMA_IN_DSCR_BF1_CH0_REG	RX channel 0 previous descriptor address register	0x0130	RO
LP_AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status register	0x0188	RO
LP_AHB_DMA_OUT_STATE_CH0_REG	TX channel 0 status register	0x0198	RO
LP_AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX channel 0 EOF address register	0x019C	RO
LP_AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX channel 0 previous EOF address register	0x01A0	RO
LP_AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG	TX channel 0 done descriptor address register	0x01A4	RO
LP_AHB_DMA_OUT_DSCR_CH0_REG	TX channel 0 next descriptor address register	0x01A8	RO
LP_AHB_DMA_OUT_DSCR_BF0_CH0_REG	TX channel 0 current descriptor address register	0x01AC	RO
LP_AHB_DMA_OUT_DSCR_BF1_CH0_REG	TX channel 0 previous descriptor address register	0x01B0	RO
LP_AHB_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status register	0x0208	RO
LP_AHB_DMA_IN_STATE_CH1_REG	RX channel 1 status register	0x0218	RO
LP_AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX channel 1 successful EOF address register	0x021C	RO

Name	Description	Address	Access
LP_AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX channel 1 error EOF address register	0x0220	RO
LP_AHB_DMA_IN_DONE_DES_ADDR_CH1_REG	RX channel 1 done descriptor address register	0x0224	RO
LP_AHB_DMA_IN_DSCR_CH1_REG	RX channel 1 next descriptor address register	0x0228	RO
LP_AHB_DMA_IN_DSCR_BF0_CH1_REG	RX channel 1 current descriptor address register	0x022C	RO
LP_AHB_DMA_IN_DSCR_BF1_CH1_REG	RX channel 1 previous descriptor address register	0x0230	RO
LP_AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 1 FIFO status register	0x0288	RO
LP_AHB_DMA_OUT_STATE_CH1_REG	TX channel 1 status register	0x0298	RO
LP_AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX channel 1 EOF address register	0x029C	RO
LP_AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX channel 1 previous EOF address register	0x02A0	RO
LP_AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG	TX channel 1 done descriptor address register	0x02A4	RO
LP_AHB_DMA_OUT_DSCR_CH1_REG	TX channel 1 next descriptor address register	0x02A8	RO
LP_AHB_DMA_OUT_DSCR_BF0_CH1_REG	TX channel 1 current descriptor address register	0x02AC	RO
LP_AHB_DMA_OUT_DSCR_BF1_CH1_REG	TX channel 1 previous descriptor address register	0x02B0	RO
LP_AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG	AHB response error status register 0	0x0620	RO
LP_AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG	AHB response error status register 1	0x0624	RO
Priority Registers			
LP_AHB_DMA_IN_PRI_CH0_REG	RX channel 0 priority register	0x0134	R/W
LP_AHB_DMA_OUT_PRI_CH0_REG	TX channel 0 priority register	0x01B4	R/W
LP_AHB_DMA_IN_PRI_CH1_REG	RX channel 1 priority register	0x0234	R/W
LP_AHB_DMA_OUT_PRI_CH1_REG	TX channel 1 priority register	0x02B4	R/W
Peripheral Select Registers			
LP_AHB_DMA_IN_PERI_SEL_CH0_REG	RX channel 0 peripheral selection register	0x0138	R/W
LP_AHB_DMA_OUT_PERI_SEL_CH0_REG	TX channel 0 peripheral selection register	0x01B8	R/W
LP_AHB_DMA_IN_PERI_SEL_CH1_REG	RX channel 1 peripheral selection register	0x0238	R/W
LP_AHB_DMA_OUT_PERI_SEL_CH1_REG	TX channel 1 peripheral selection register	0x02B8	R/W

3.8.2 GDMA-AXI Register Summary

The addresses in this section are relative to GDMA-AXI base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Interrupt Registers			
AXI_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
AXI_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
AXI_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
AXI_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
AXI_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0068	R/WTC/SS
AXI_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x006C	RO
AXI_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0070	R/W
AXI_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x0074	WT
AXI_DMA_IN_INT_RAW_CH2_REG	RX channel 2 raw interrupt status register	0x00D0	R/WTC/SS
AXI_DMA_IN_INT_ST_CH2_REG	RX channel 2 masked interrupt status register	0x00D4	RO
AXI_DMA_IN_INT_ENA_CH2_REG	RX channel 2 interrupt enable register	0x00D8	R/W
AXI_DMA_IN_INT_CLR_CH2_REG	RX channel 2 interrupt clear register	0x00DC	WT
AXI_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0138	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x013C	RO
AXI_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0140	R/W
AXI_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x0144	WT
AXI_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x01A0	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x01A4	RO
AXI_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x01A8	R/W
AXI_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x01AC	WT
AXI_DMA_OUT_INT_RAW_CH2_REG	TX channel 2 raw interrupt status register	0x0208	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH2_REG	TX channel 2 masked interrupt status register	0x020C	RO
AXI_DMA_OUT_INT_ENA_CH2_REG	TX channel 2 interrupt enable register	0x0210	R/W
AXI_DMA_OUT_INT_CLR_CH2_REG	TX channel 2 interrupt clear register	0x0214	WT
Configuration Registers			
AXI_DMA_IN_CONF0_CH0_REG	RX channel 0 configuration 0 register	0x0010	R/W

Name	Description	Address	Access
AXI_DMA_IN_CONF1_CH0_REG	RX channel 0 configuration 1 register	0x0014	R/W
AXI_DMA_IN_POP_CH0_REG	RX channel 0 FIFO pop register	0x001C	varies
AXI_DMA_IN_LINK1_CH0_REG	RX channel 0 descriptor control register	0x0020	varies
AXI_DMA_IN_LINK2_CH0_REG	RX channel 0 descriptor address register	0x0024	R/W
AXI_DMA_IN_CONF0_CH1_REG	RX channel 1 configuration 0 register	0x0078	R/W
AXI_DMA_IN_CONF1_CH1_REG	RX channel 1 configuration 1 register	0x007C	R/W
AXI_DMA_IN_POP_CH1_REG	RX channel 1 FIFO pop register	0x0084	varies
AXI_DMA_IN_LINK1_CH1_REG	RX channel 1 descriptor control register	0x0088	varies
AXI_DMA_IN_LINK2_CH1_REG	RX channel 1 descriptor address register	0x008C	R/W
AXI_DMA_IN_CONF0_CH2_REG	RX channel 2 configuration 0 register	0x00E0	R/W
AXI_DMA_IN_CONF1_CH2_REG	RX channel 2 configuration 1 register	0x00E4	R/W
AXI_DMA_IN_POP_CH2_REG	RX channel 2 FIFO pop register	0x00EC	varies
AXI_DMA_IN_LINK1_CH2_REG	RX channel 2 descriptor control register	0x00F0	varies
AXI_DMA_IN_LINK2_CH2_REG	RX channel 2 descriptor address register	0x00F4	R/W
AXI_DMA_OUT_CONF0_CH0_REG	TX channel 0 configuration 0 register	0x0148	R/W
AXI_DMA_OUT_CONF1_CH0_REG	TX channel 0 configuration 1 register	0x014C	R/W
AXI_DMA_OUT_PUSH_CH0_REG	TX channel 0 FIFO push register	0x0154	varies
AXI_DMA_OUT_LINK1_CH0_REG	TX channel 0 descriptor control register	0x0158	varies
AXI_DMA_OUT_LINK2_CH0_REG	TX channel 0 descriptor address register	0x015C	R/W
AXI_DMA_OUT_CONF0_CH1_REG	TX channel 1 configuration 0 register	0x01B0	R/W
AXI_DMA_OUT_CONF1_CH1_REG	TX channel 1 configuration 1 register	0x01B4	R/W
AXI_DMA_OUT_PUSH_CH1_REG	TX channel 1 FIFO push register	0x01BC	varies
AXI_DMA_OUT_LINK1_CH1_REG	TX channel 1 descriptor control register	0x01C0	varies
AXI_DMA_OUT_LINK2_CH1_REG	TX channel 1 descriptor address register	0x01C4	R/W
AXI_DMA_OUT_CONF0_CH2_REG	TX channel 2 configuration 0 register	0x0218	R/W
AXI_DMA_OUT_CONF1_CH2_REG	TX channel 2 configuration 1 register	0x021C	R/W
AXI_DMA_OUT_PUSH_CH2_REG	TX channel 2 FIFO push register	0x0224	varies
AXI_DMA_OUT_LINK1_CH2_REG	TX channel 2 descriptor control register	0x0228	varies
AXI_DMA_OUT_LINK2_CH2_REG	TX channel 2 descriptor address register	0x022C	R/W

Name	Description	Address	Access
AXI_DMA_ARB_TIMEOUT_REG	Arbitration timeout register	0x0270	R/W
AXI_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0274	R/W
AXI_DMA_IN_MEM_CONF_REG	Memory configuration register	0x0278	R/W
AXI_DMA_INTR_MEM_START_ADDR_REG	Accessible start address register	0x027C	R/W
AXI_DMA_INTR_MEM_END_ADDR_REG	Accessible end address register	0x0280	R/W
AXI_DMA_EXTR_MEM_START_ADDR_REG	Accessible start address register	0x0284	R/W
AXI_DMA_EXTR_MEM_END_ADDR_REG	Accessible end address register	0x0288	R/W
AXI_DMA_IN_RESET_AVAIL_CH0_REG	RX channel 0 reset status register	0x028C	RO
AXI_DMA_IN_RESET_AVAIL_CH1_REG	RX channel 1 reset status register	0x0290	RO
AXI_DMA_IN_RESET_AVAIL_CH2_REG	RX channel 2 reset status register	0x0294	RO
AXI_DMA_OUT_RESET_AVAIL_CH0_REG	TX channel 0 reset status register	0x0298	RO
AXI_DMA_OUT_RESET_AVAIL_CH1_REG	TX channel 1 reset status register	0x029C	RO
AXI_DMA_OUT_RESET_AVAIL_CH2_REG	TX channel 2 reset status register	0x02A0	RO
AXI_DMA_MISC_CONF_REG	Miscellaneous configuration register	0x02A8	R/W
Status Registers			
AXI_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status register	0x0018	RO
AXI_DMA_IN_STATE_CH0_REG	RX channel 0 status register	0x0028	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX channel 0 successful EOF address register	0x002C	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX channel 0 error EOF address register	0x0030	RO
AXI_DMA_IN_DSCR_CH0_REG	RX channel 0 next descriptor address register	0x0034	RO
AXI_DMA_IN_DSCR_BF0_CH0_REG	RX channel 0 current descriptor address register	0x0038	RO
AXI_DMA_IN_DSCR_BF1_CH0_REG	RX channel 0 previous descriptor address register	0x003C	RO
AXI_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status register	0x0080	RO
AXI_DMA_IN_STATE_CH1_REG	RX channel 1 status register	0x0090	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX channel 1 successful EOF address register	0x0094	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX channel 1 error EOF address register	0x0098	RO
AXI_DMA_IN_DSCR_CH1_REG	RX channel 1 next descriptor address register	0x009C	RO
AXI_DMA_IN_DSCR_BF0_CH1_REG	RX channel 1 current descriptor address register	0x00A0	RO
AXI_DMA_IN_DSCR_BF1_CH1_REG	RX channel 1 previous descriptor address register	0x00A4	RO

Name	Description	Address	Access
AXI_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status register	0x00E8	RO
AXI_DMA_IN_STATE_CH2_REG	RX channel 2 status register	0x00F8	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX channel 2 successful EOF address register	0x00FC	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX channel 2 error EOF address register	0x0100	RO
AXI_DMA_IN_DSCR_CH2_REG	RX channel 2 next descriptor address register	0x0104	RO
AXI_DMA_IN_DSCR_BF0_CH2_REG	RX channel 2 current descriptor address register	0x0108	RO
AXI_DMA_IN_DSCR_BF1_CH2_REG	RX channel 2 previous descriptor address register	0x010C	RO
AXI_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status register	0x0150	RO
AXI_DMA_OUT_STATE_CH0_REG	TX channel 0 status register	0x0160	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX channel 0 EOF address register	0x0164	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX channel 0 previous EOF address register	0x0168	RO
AXI_DMA_OUT_DSCR_CH0_REG	TX channel 0 next descriptor address register	0x016C	RO
AXI_DMA_OUT_DSCR_BF0_CH0_REG	TX channel 0 current descriptor address register	0x0170	RO
AXI_DMA_OUT_DSCR_BF1_CH0_REG	TX channel 0 previous descriptor address register	0x0174	RO
AXI_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 1 FIFO status register	0x01B8	RO
AXI_DMA_OUT_STATE_CH1_REG	TX channel 1 status register	0x01C8	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX channel 1 EOF address register	0x01CC	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX channel 1 previous EOF address register	0x01D0	RO
AXI_DMA_OUT_DSCR_CH1_REG	TX channel 1 next descriptor address register	0x01D4	RO
AXI_DMA_OUT_DSCR_BF0_CH1_REG	TX channel 1 current descriptor address register	0x01D8	RO
AXI_DMA_OUT_DSCR_BF1_CH1_REG	TX channel 1 previous descriptor address register	0x01DC	RO
AXI_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status register	0x0220	RO
AXI_DMA_OUT_STATE_CH2_REG	TX channel 2 status register	0x0230	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX channel 2 EOF address register	0x0234	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX channel 2 previous EOF address register	0x0238	RO
AXI_DMA_OUT_DSCR_CH2_REG	TX channel 2 next descriptor address register	0x023C	RO
AXI_DMA_OUT_DSCR_BF0_CH2_REG	TX channel 2 current descriptor address register	0x0240	RO
AXI_DMA_OUT_DSCR_BF1_CH2_REG	TX channel 2 previous descriptor address register	0x0244	RO
AXI_DMA_WRESP_CNT_REG	AXI write response counter register	0x02B8	RO

Name	Description	Address	Access
AXI_DMA_RRESP_CNT_REG	AXI read response counter register	0x02BC	RO
AXI_DMA_INFIFO_STATUS1_CH0_REG	RX channel 0 FIFO status 1 register	0x02C0	RO
AXI_DMA_INFIFO_STATUS1_CH1_REG	RX channel 1 FIFO status 1 register	0x02C4	RO
AXI_DMA_INFIFO_STATUS1_CH2_REG	RX channel 2 FIFO status 1 register	0x02C8	RO
AXI_DMA_OUTFIFO_STATUS1_CH0_REG	TX channel 0 FIFO status 1 register	0x02CC	RO
AXI_DMA_OUTFIFO_STATUS1_CH1_REG	TX channel 1 FIFO status 1 register	0x02D0	RO
AXI_DMA_OUTFIFO_STATUS1_CH2_REG	TX channel 2 FIFO status 1 register	0x02D4	RO
AXI_DMA_LINK_SWITCH_STATE_REG	Link switch register	0x02DC	R/W
Priority Registers			
AXI_DMA_IN_PRI_CH0_REG	RX channel 0 priority register	0x0040	R/W
AXI_DMA_IN_PRI_CH1_REG	RX channel 1 priority register	0x00A8	R/W
AXI_DMA_IN_PRI_CH2_REG	RX channel 2 priority register	0x0110	R/W
AXI_DMA_OUT_PRI_CH0_REG	TX channel 0 priority register	0x0178	R/W
AXI_DMA_OUT_PRI_CH1_REG	TX channel 1 priority register	0x01E0	R/W
AXI_DMA_OUT_PRI_CH2_REG	TX channel 2 priority register	0x0248	R/W
Peripheral Select Registers			
AXI_DMA_IN_PERI_SEL_CH0_REG	RX channel 0 peripheral selection register	0x0044	R/W
AXI_DMA_IN_PERI_SEL_CH1_REG	RX channel 1 peripheral selection register	0x00AC	R/W
AXI_DMA_IN_PERI_SEL_CH2_REG	RX channel 2 peripheral selection register	0x0114	R/W
AXI_DMA_OUT_PERI_SEL_CH0_REG	TX channel 0 peripheral selection register	0x017C	R/W
AXI_DMA_OUT_PERI_SEL_CH1_REG	TX channel 1 peripheral selection register	0x01E4	R/W
AXI_DMA_OUT_PERI_SEL_CH2_REG	TX channel 2 peripheral selection register	0x024C	R/W
Debug Registers			
AXI_DMA_RDN_RESULT_REG	Reserved register	0x02AC	varies
AXI_DMA_RDN_ECO_HIGH_REG	Reserved register	0x02B0	R/W
AXI_DMA_RDN_ECO_LOW_REG	Reserved register	0x02B4	R/W
Version Registers			
AXI_DMA_DATE_REG	Version control register	0x02D8	R/W

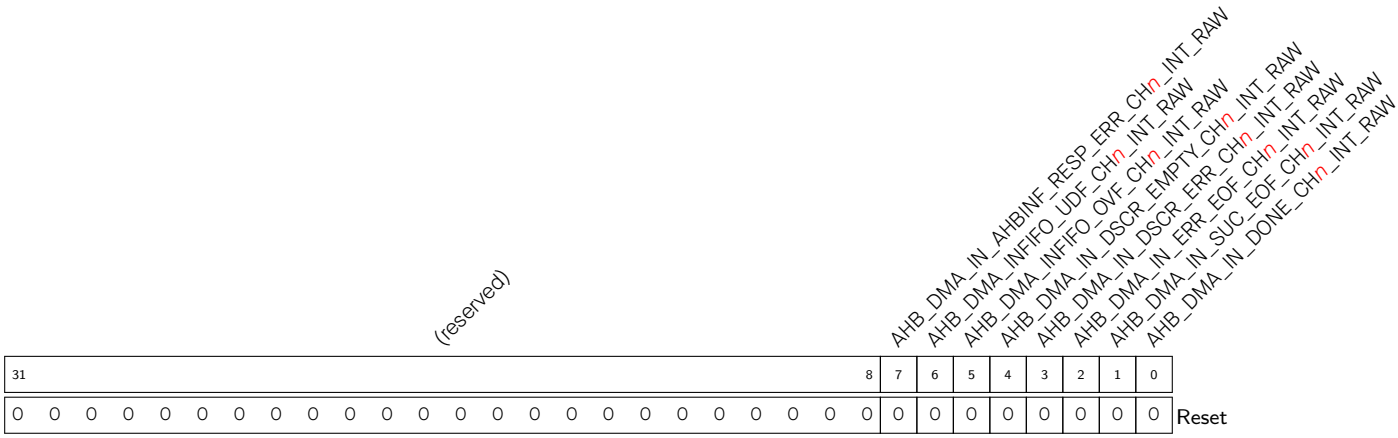
3.9 Registers

3.9.1 GDMA-AHB-HP Registers

The addresses in this section are relative to GDMA-AHB-HP base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 3.1. AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-4) (0x0000+0x10n*)**



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

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

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

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

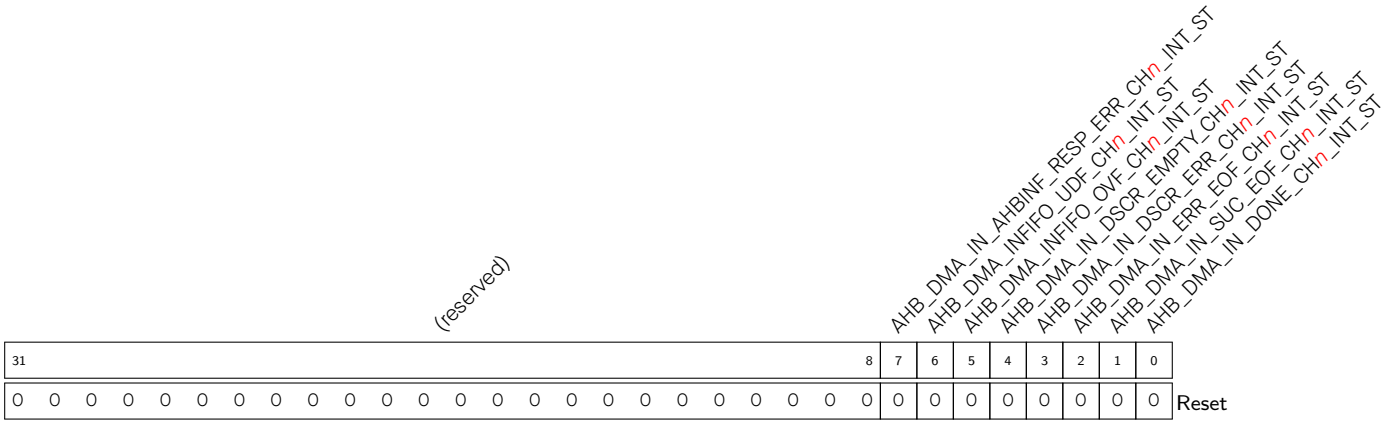
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.2. AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-4) (0x0004+0x10**n*)



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

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

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

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

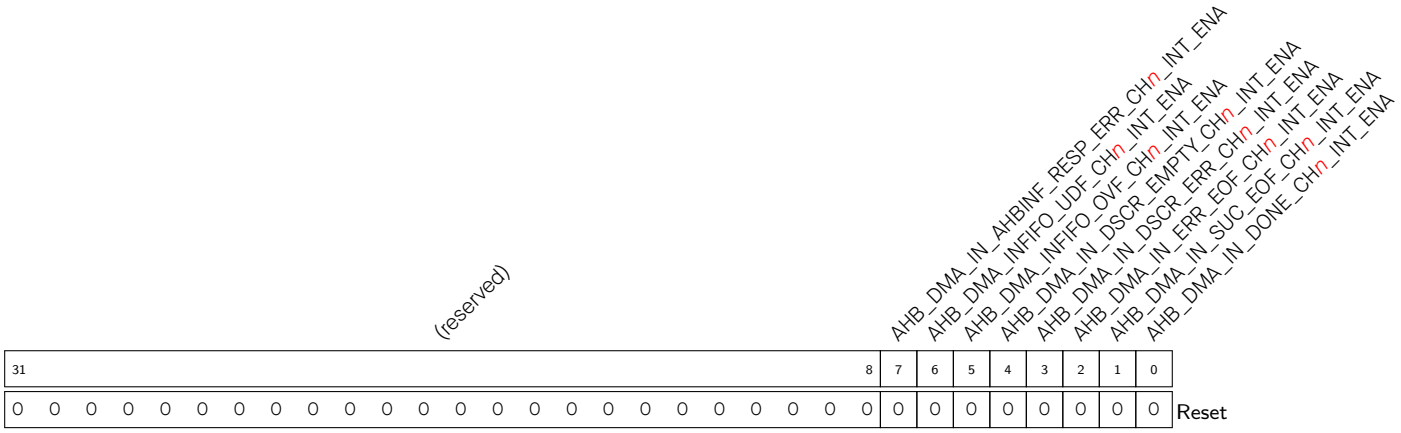
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (RO)

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

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

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

Register 3.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-4) (0x0008+0x10n*)**



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

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

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

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

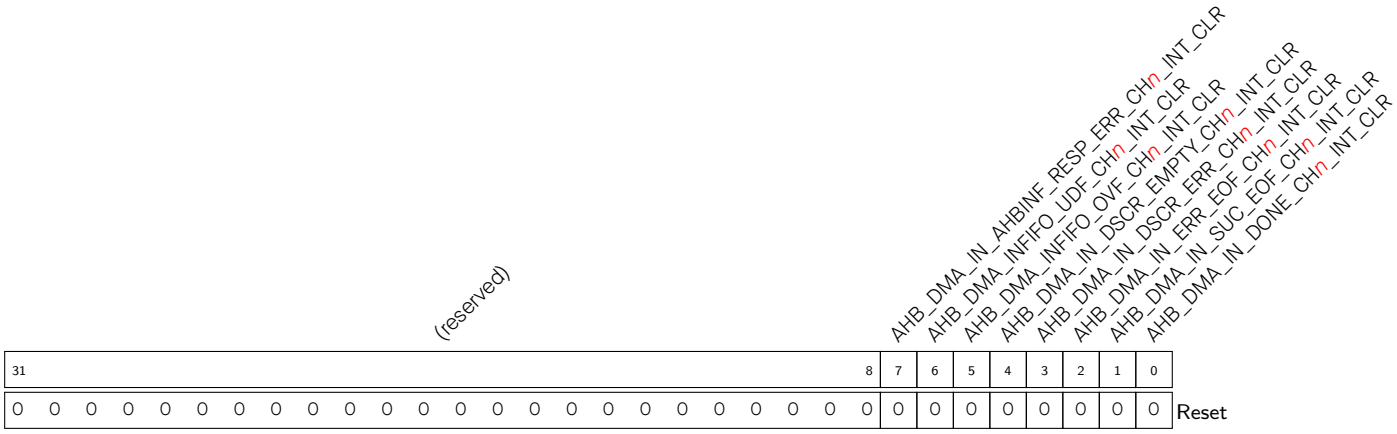
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

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

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

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

Register 3.4. AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-4) (0x000C+0x10n*)**



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

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

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

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

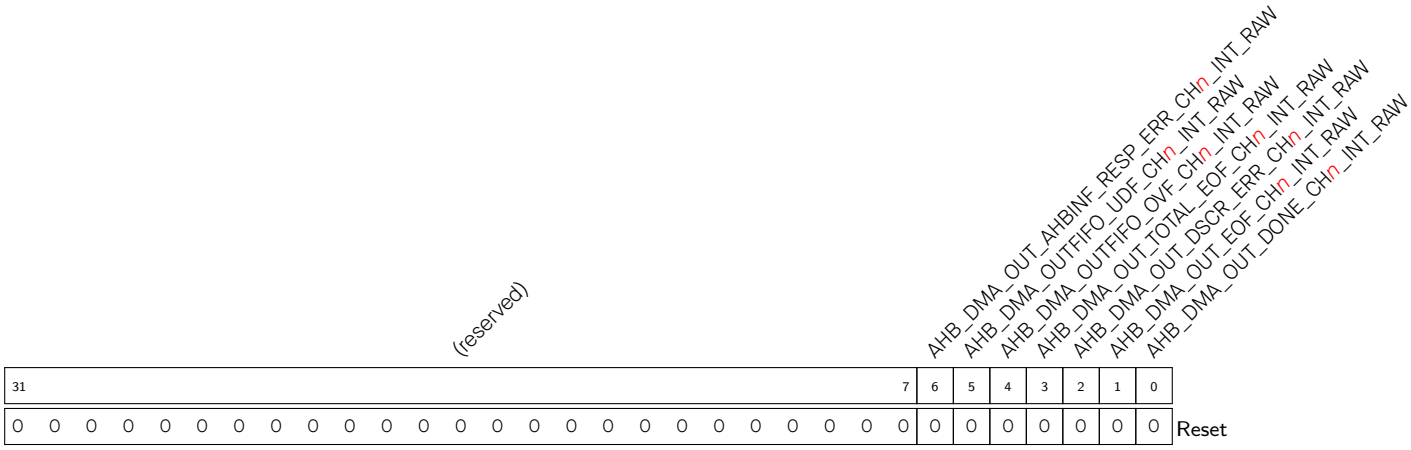
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (WT)

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

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

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

Register 3.5. AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-4) (0x0050+0x10n*)**



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

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

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

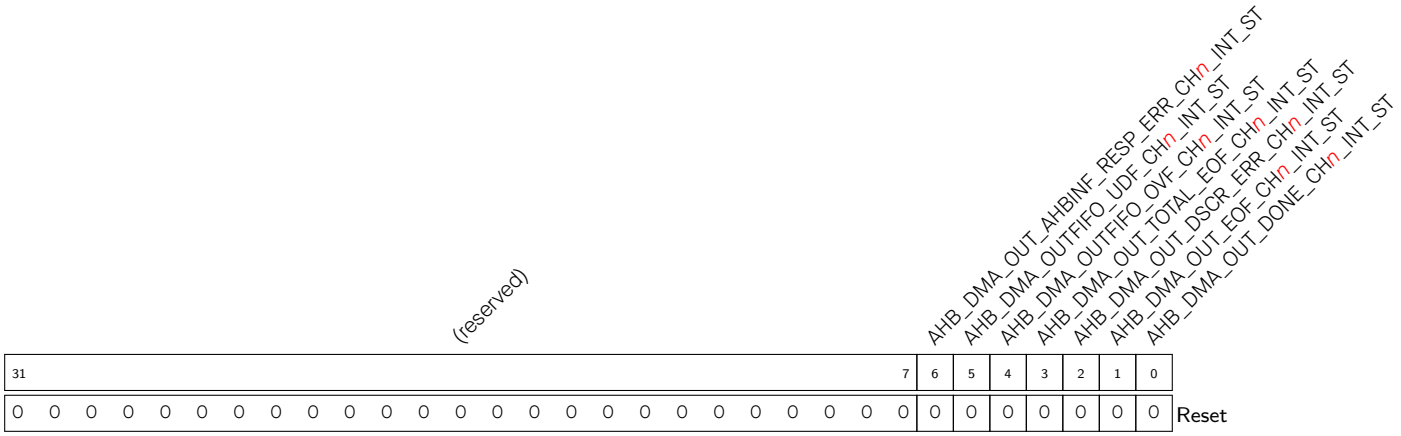
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.6. AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-4) (0x0054+0x10n*)**



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

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

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

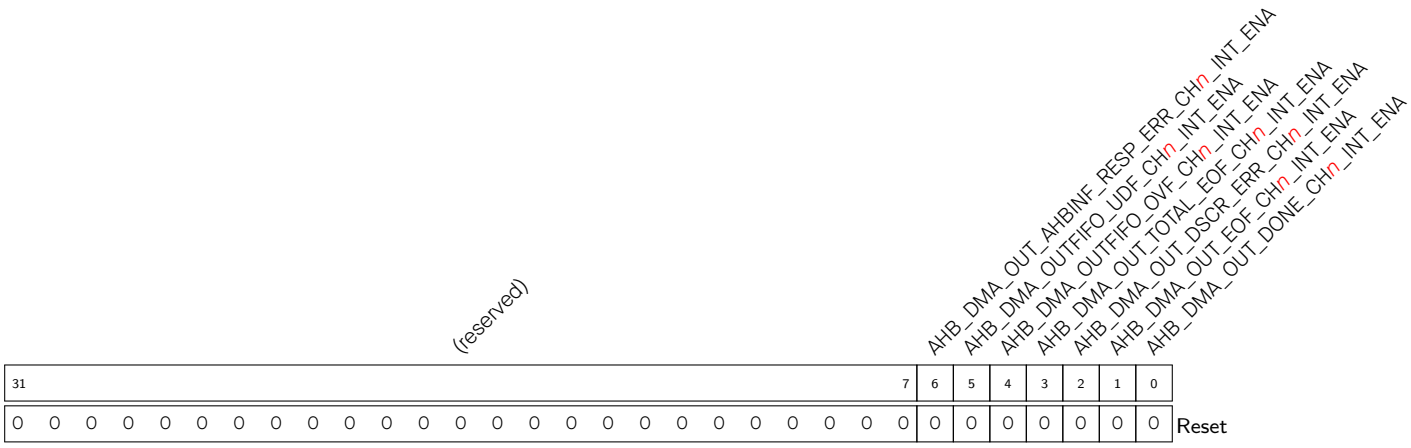
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (RO)

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

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

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

Register 3.7. AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-4) (0x0058+0x10**n*)



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

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

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

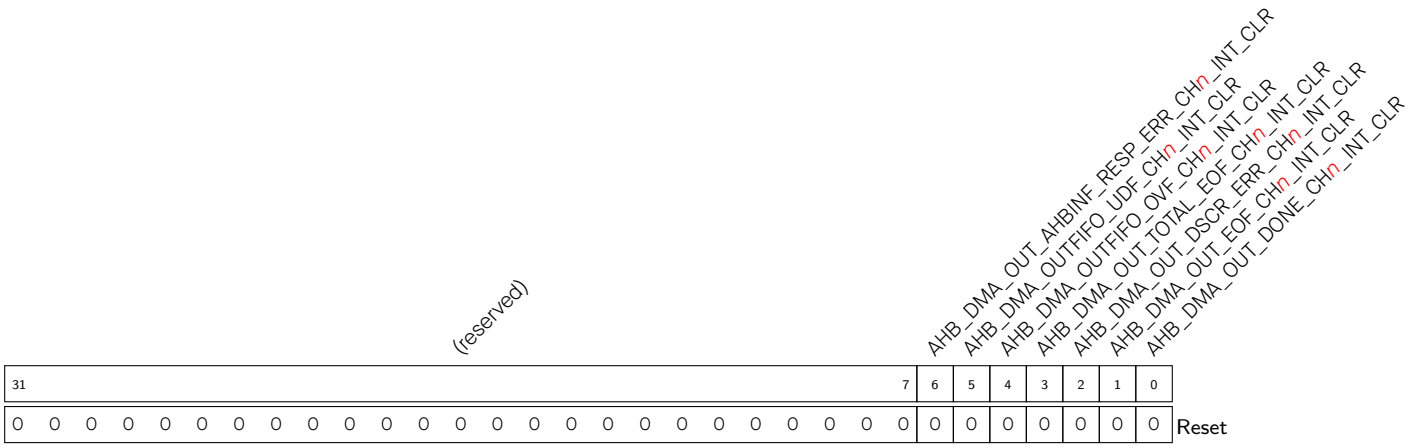
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

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

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

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

Register 3.8. AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-4) (0x005C+0x10**n*)



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

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

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

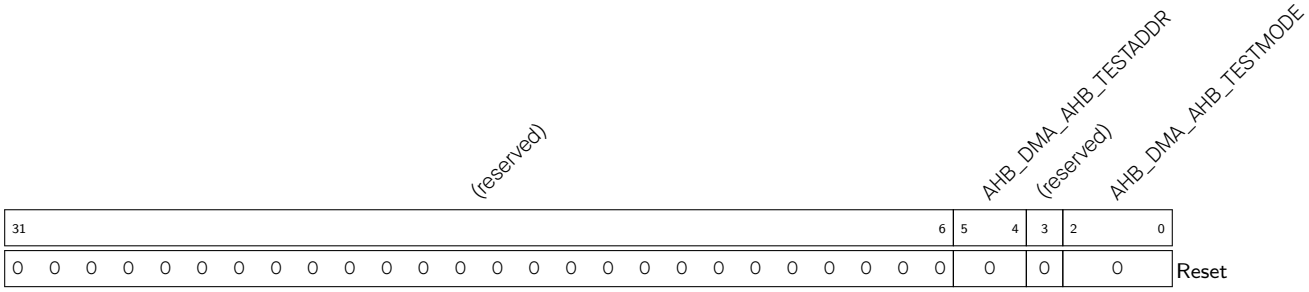
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (WT)

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

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

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

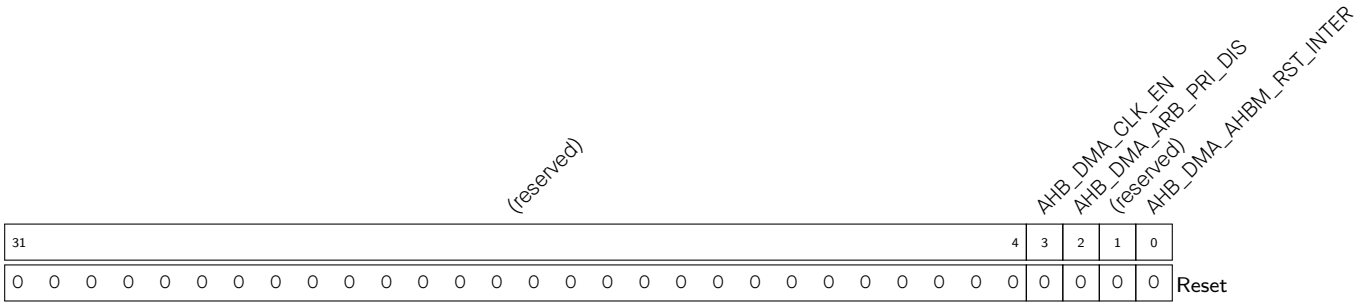
Register 3.9. AHB_DMA_AHB_TEST_REG (0x00A0)



AHB_DMA_AHB_TESTMODE Reserved. (R/W)

AHB_DMA_AHB_TESTADDR Reserved. (R/W)

Register 3.10. AHB_DMA_MISC_CONF_REG (0x00A4)



AHB_DMA_AHBM_RST_INTER Configures reset state of the internal AHB FSM.

0: Release reset

1: Reset

(R/W)

AHB_DMA_ARB_PRI_DIS Configures whether to disable fixed-priority channel arbitration.

0: Enable

1: Disable

(R/W)

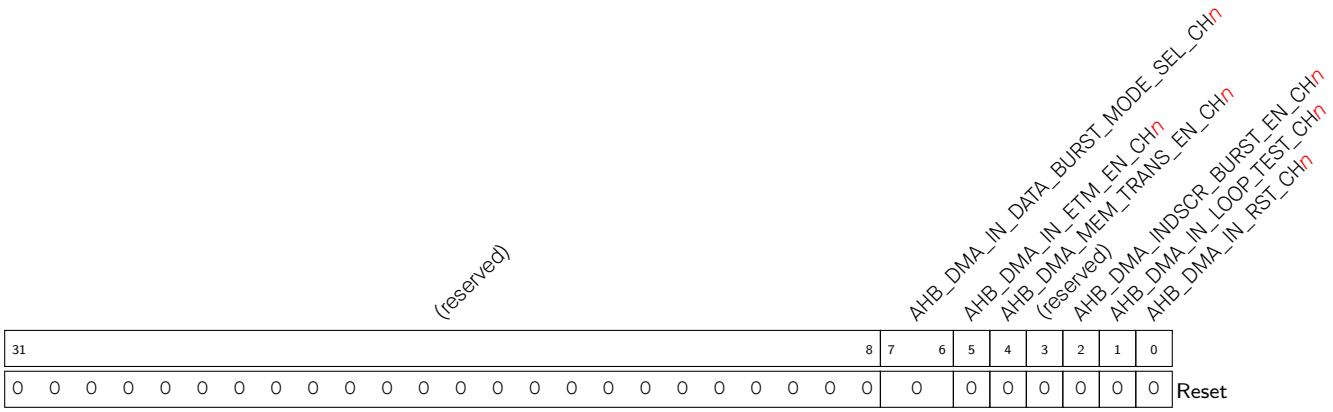
AHB_DMA_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 3.11. AHB_DMA_IN_CONFO_CH*n*_REG (*n*: 0-4) (0x0100+0x100n*)**



AHB_DMA_IN_RST_CH*n* Configures reset state of AHB DMA RX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

AHB_DMA_IN_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for RX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

AHB_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read receive descriptors.

0: Disable

1: Enable

(R/W)

AHB_DMA_MEM_TRANS_EN_CH*n* Configures whether to enable memory-to-memory data transfer.

0: Disable

1: Enable

(R/W)

AHB_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel *n*.

0: Disable

1: Enable

(R/W)

AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* Configures burst transfer mode for RX channel *n*.

0: Single

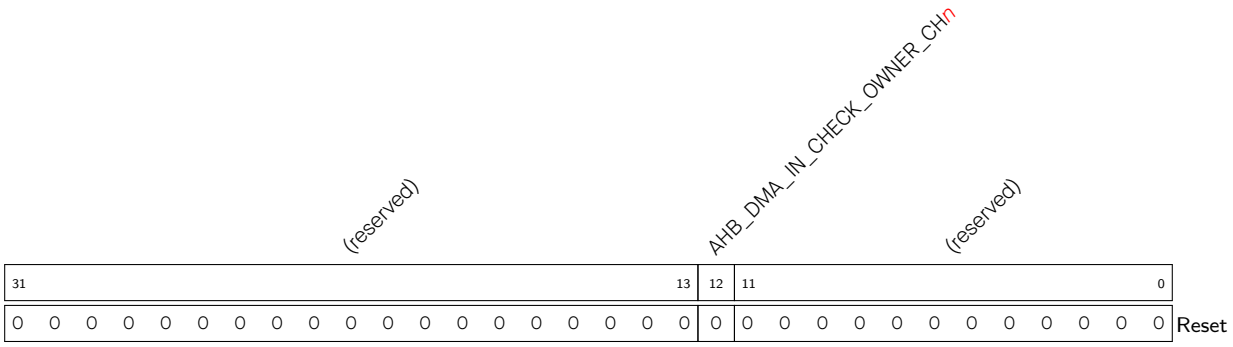
1: INCR4

2: INCR8

3: INCR16

(R/W)

Register 3.12. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-4) (0x0104+0x100**n*)



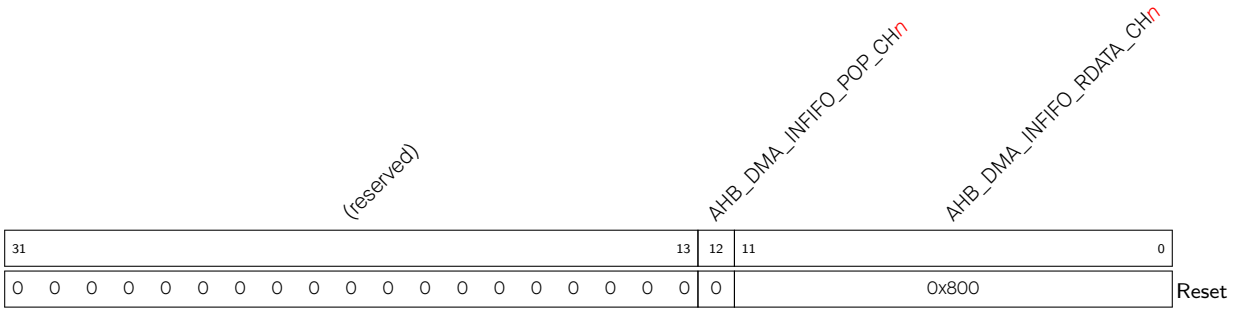
AHB_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.13. AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-4) (0x010C+0x100**n*)



AHB_DMA_INFIFO_RDATA_CH*n* Data received from AHB DMA FIFO. (RO)

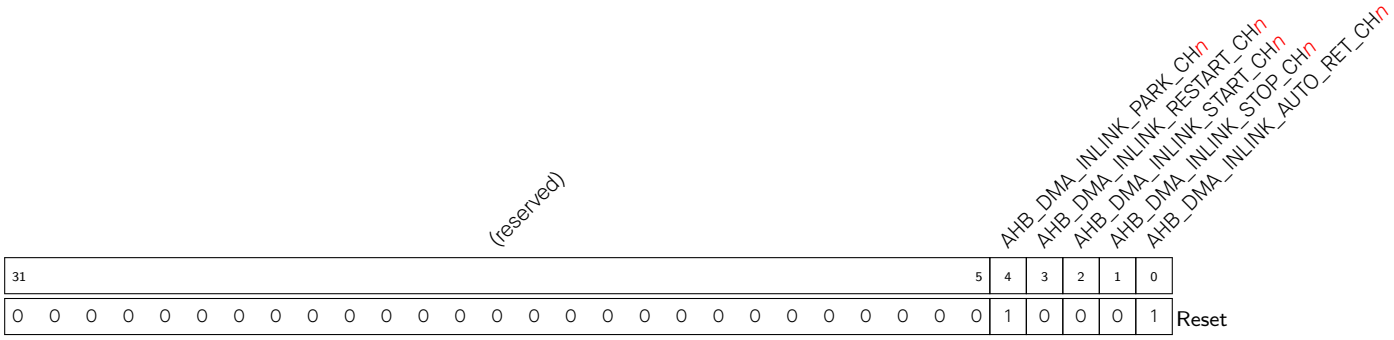
AHB_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AHB DMA FIFO for RX channel *n*.

0: Invalid. No effect

1: Pop

(WT)

Register 3.14. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-4) (0x0110+0x100n*)**



AHB_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor address when an error occurs in the current received data for RX channel *n*.

0: Not return

1: Return

(R/W)

AHB_DMA_INLINK_STOP_CH*n* Configures whether to stop processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_INLINK_START_CH*n* Configures whether to start processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Start

(WT)

AHB_DMA_INLINK_RESTART_CH*n* Configures whether to restart descriptor processing for RX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

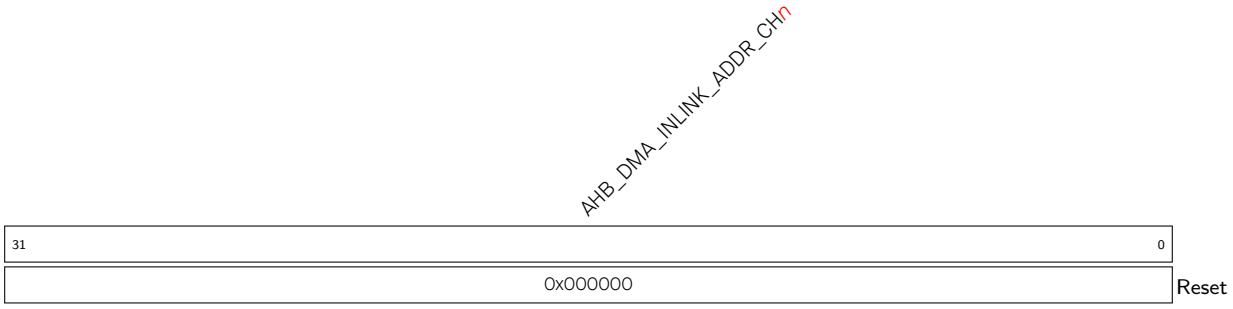
AHB_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor FSM for RX channel *n*.

0: Running

1: Idle

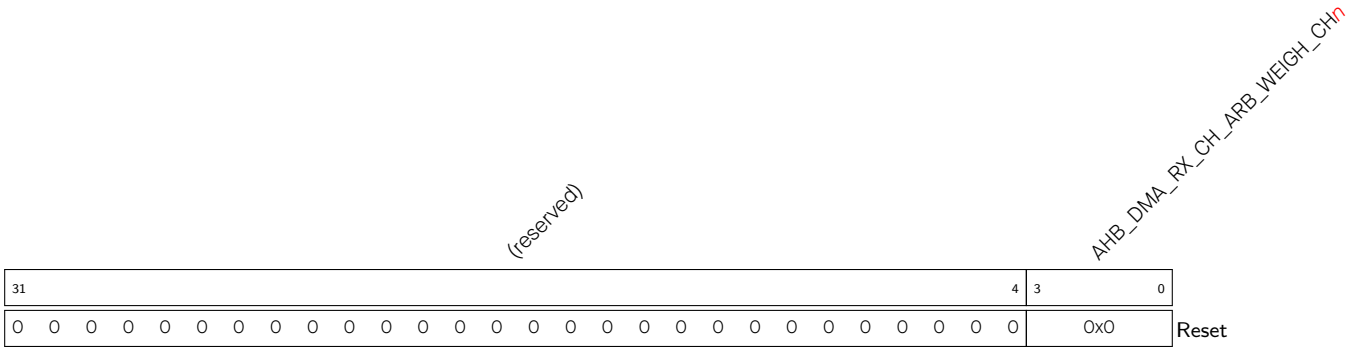
(RO)

Register 3.15. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-4) (0x0114+0x100n*)**



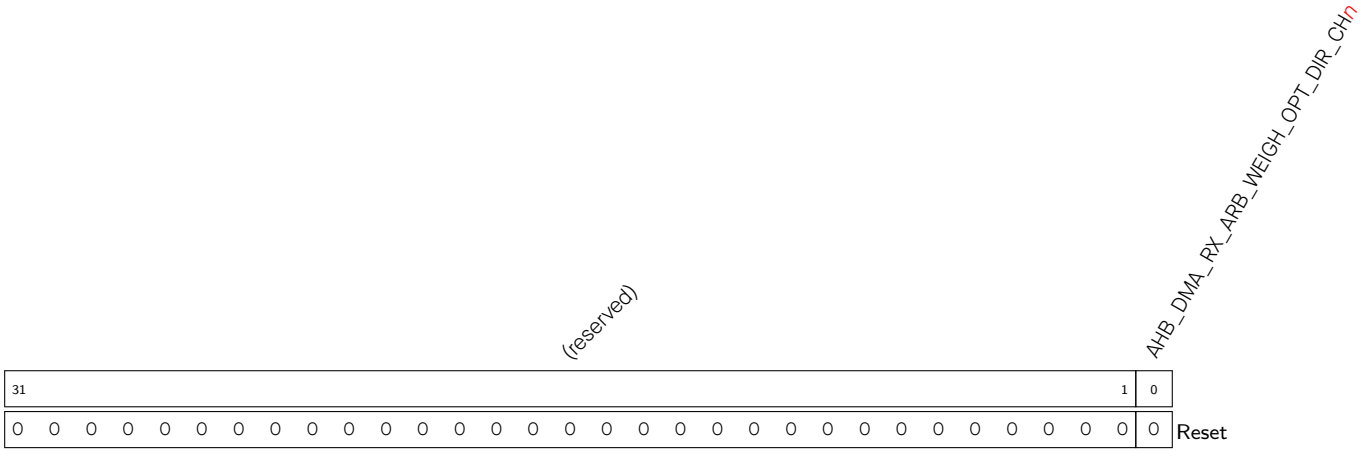
AHB_DMA_INLINK_ADDR_CH*n* Configures the first receive descriptor address for RX channel *n*. (R/W)

Register 3.16. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-4) (0x013C+0x100n*)**



AHB_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of RX channel *n*. Measurement unit: token. (R/W)

Register 3.17. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-4) (0x0140+0x100n*)**



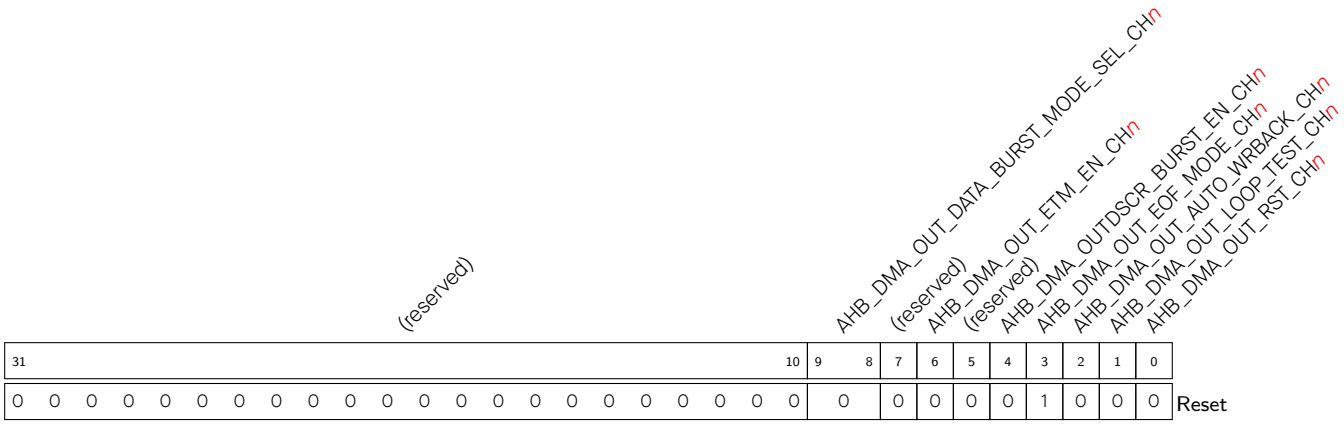
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for RX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.18. AHB_DMA_OUT_CONFO_CH*n*_REG (*n*: 0-4) (0x0180+0x100n*)**



AHB_DMA_OUT_RST_CH*n* Configures reset state of AHB DMA TX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for TX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

AHB_DMA_OUT_AUTO_WRBACK_CH*n* Configures whether to enable automatic outlink write-back when all data in the TX FIFO has been transmitted for TX channel *n*.

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_EOF_MODE_CH*n* Configures when to generate the EOF flag for TX channel *n*.

0: Generate when data to be transmitted has been pushed into the FIFO.

1: Generate when data to be transmitted has been popped from the FIFO.

(R/W)

AHB_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* to read transmit descriptors.

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_ETM_EN_CH*n* Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n* Configures burst transfer mode for TX channel *n*.

0: Single

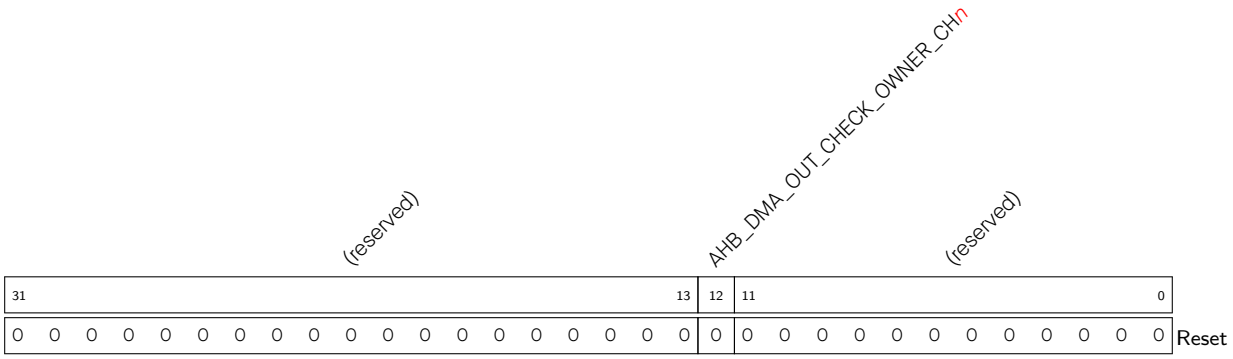
1: INCR4

2: INCR8

3: INCR16

(R/W)

Register 3.19. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-4) (0x0184+0x100**n*)



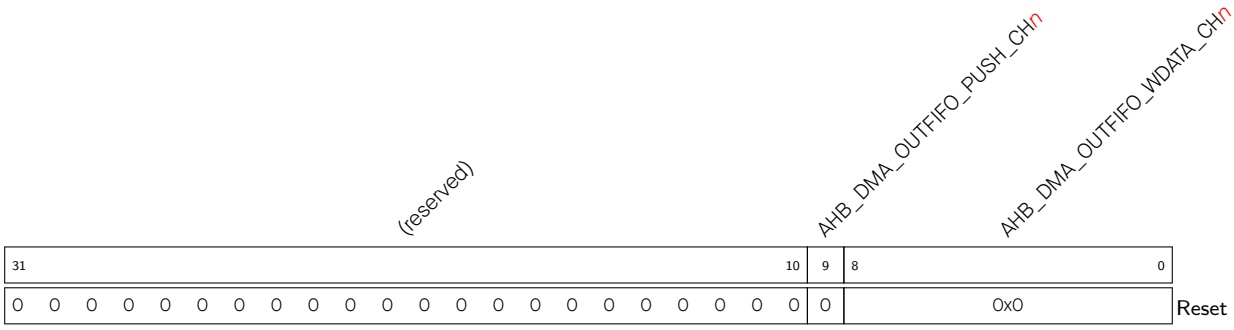
AHB_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.20. AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-4) (0x018C+0x100**n*)



AHB_DMA_OUTFIFO_WDATA_CH*n* Data to be transmitted to AHB DMA FIFO. (R/W)

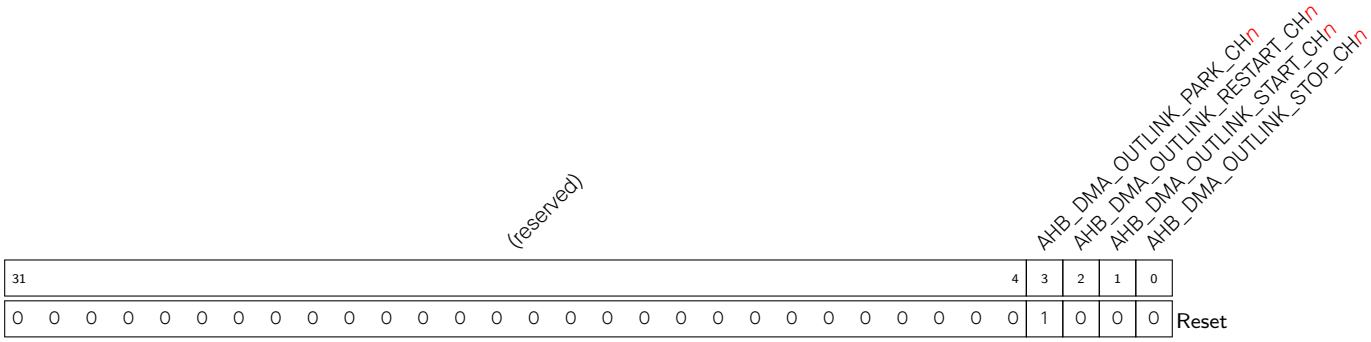
AHB_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AHB DMA FIFO for TX channel *n*.

0: Invalid. No effect

1: Push

(WT)

Register 3.21. AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-4) (0x0190+0x100n*)**



AHB_DMA_OUTLINK_STOP_CH*n* Configures whether to stop processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_OUTLINK_START_CH*n* Configures whether to start processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Start

(WT)

AHB_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart descriptor processing for TX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

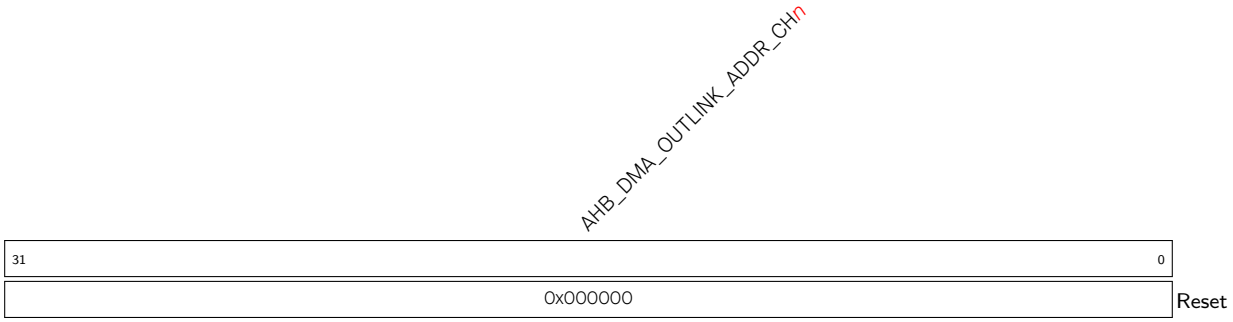
AHB_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor FSM for TX channel *n*.

0: Running

1: Idle

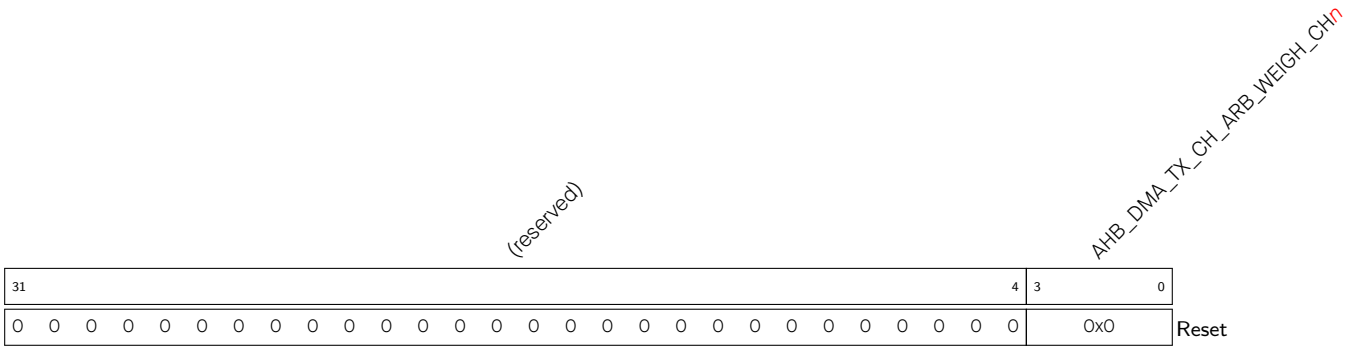
(RO)

Register 3.22. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-4) (0x0194+0x100n*)**



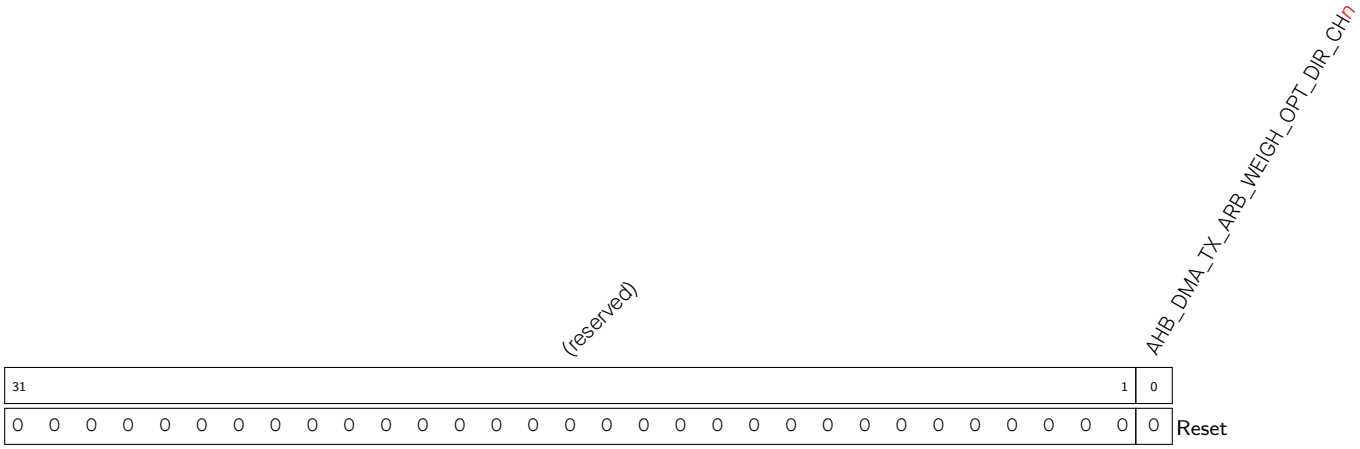
AHB_DMA_OUTLINK_ADDR_CH*n* Configures the first transmit descriptor address for TX channel *n*. (R/W)

Register 3.23. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-4) (0x01BC+0x100n*)**



AHB_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of TX channel *n*. Measurement unit: token. (R/W)

Register 3.24. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-4) (0x01C0+0x100**n*)



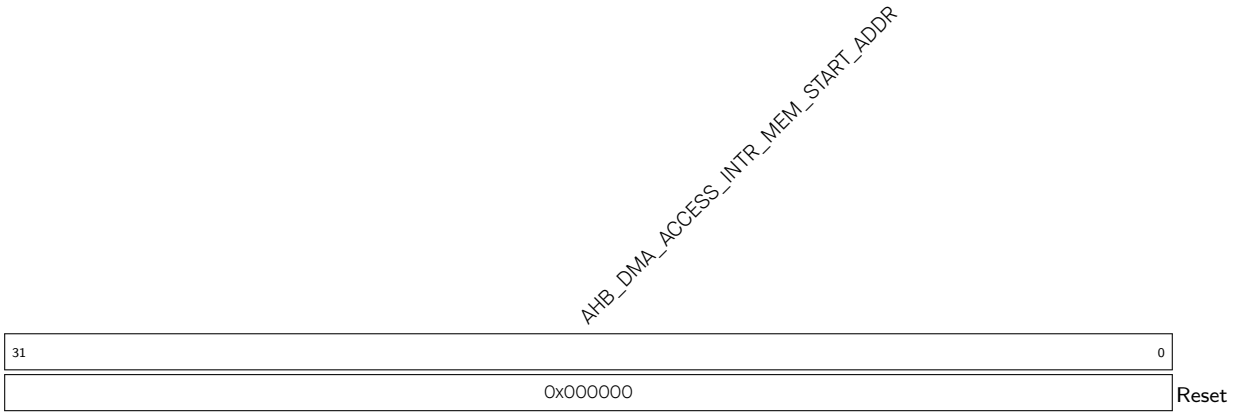
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.25. AHB_DMA_INTR_MEM_START_ADDR_REG (0x0600)



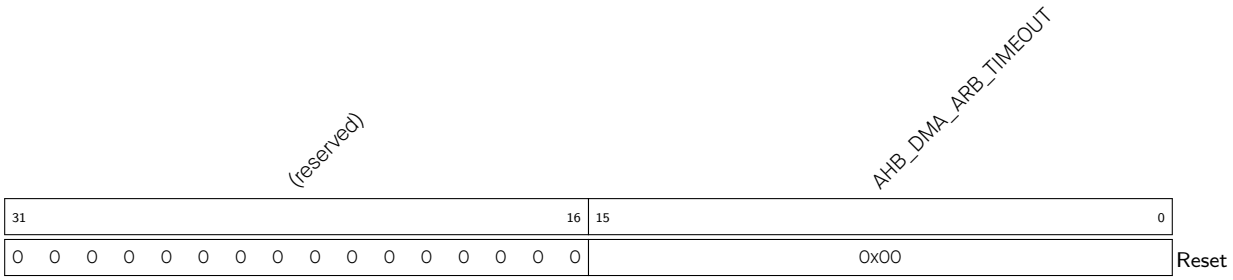
AHB_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible memory space. (R/W)

Register 3.26. AHB_DMA_INTR_MEM_END_ADDR_REG (0x0604)



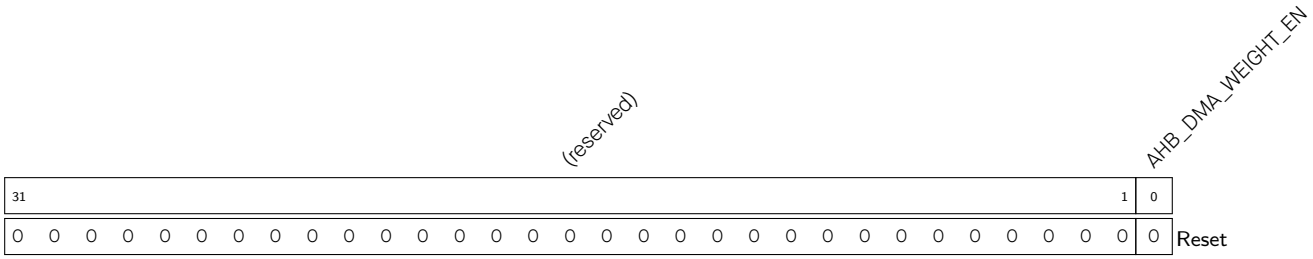
AHB_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible memory space. (R/W)

Register 3.27. AHB_DMA_ARB_TIMEOUT_REG (0x0608)



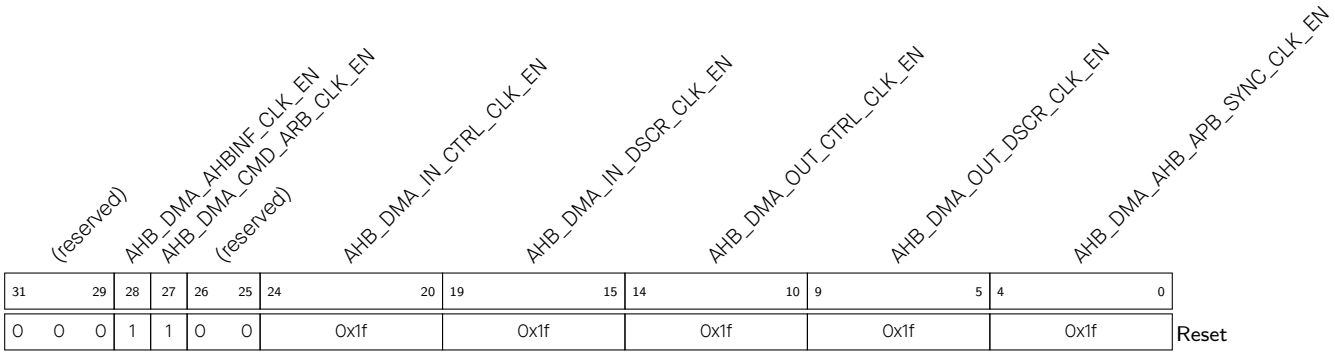
AHB_DMA_ARB_TIMEOUT Configures arbitration time slot.
Measurement unit: bus clock cycle. (R/W)

Register 3.28. AHB_DMA_WEIGHT_EN_REG (0x0610)



AHB_DMA_WEIGHT_EN Configures whether to enable weight arbitration.
0: Disable
1: Enable
(R/W)

Register 3.29. AHB_DMA_MODULE_CLK_EN_REG (0x0618)



AHB_DMA_AHB_APB_SYNC_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA register configuration synchronization module.

Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_OUT_DSCR_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA TX descriptor processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_OUT_CTRL_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA TX data processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_IN_DSCR_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA RX descriptor processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

AHB_DMA_IN_CTRL_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA RX data processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

Continued on the next page...

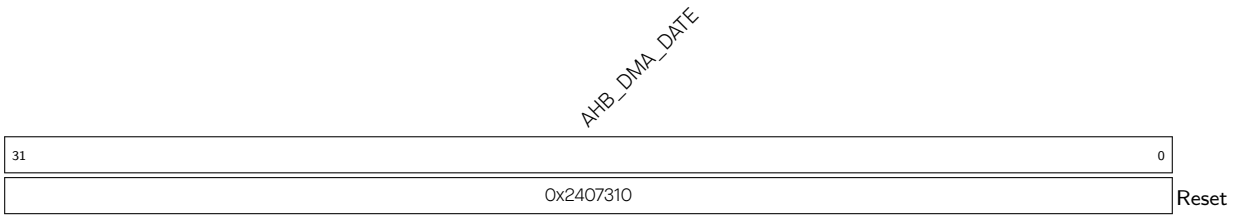
Register 3.29. AHB_DMA_MODULE_CLK_EN_REG (0x0618)

Continued from the previous page...

AHB_DMA_CMD_ARB_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA arbitration module.
Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.
0: Not forcibly enable
1: Forcibly enable
(R/W)

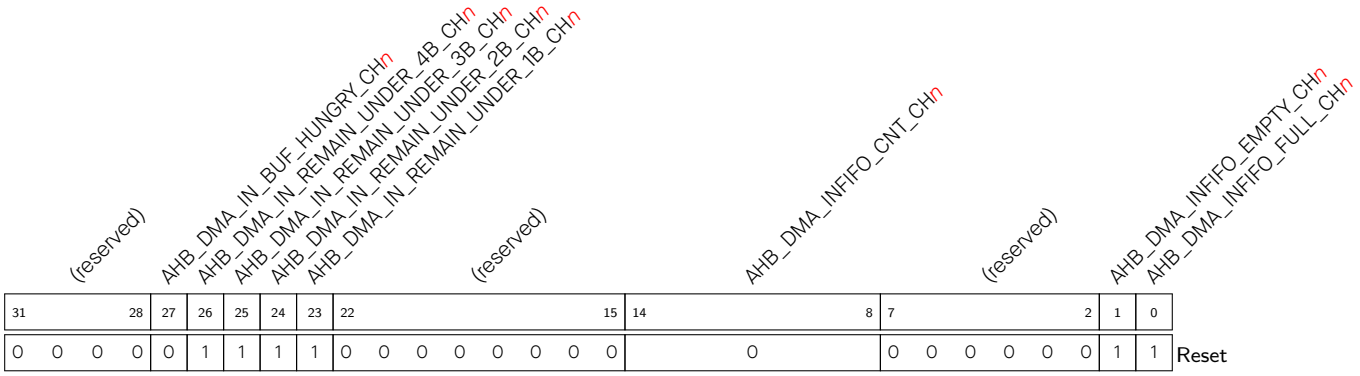
AHB_DMA_AHBINF_CLK_EN Configures whether to forcibly enable the clock for the AHB DMA bus interface processing module.
Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.
0: Not forcibly enable
1: Forcibly enable
(R/W)

Register 3.30. AHB_DMA_DATE_REG (0x00A8)



AHB_DMA_DATE Version control register. (R/W)

Register 3.31. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-4) (0x0108+0x100n*)**



AHB_DMA_INFIFO_FULL_CH*n* Represents whether RX FIFO is full for RX channel *n*.

0: Not full

1: Full

(RO)

AHB_DMA_INFIFO_EMPTY_CH*n* Represents whether RX FIFO is empty for RX channel *n*.

0: Not empty

1: Empty

(RO)

AHB_DMA_INFIFO_CNT_CH*n* Represents the number of data bytes in RX FIFO for RX channel *n*.

Measurement unit: byte. (RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

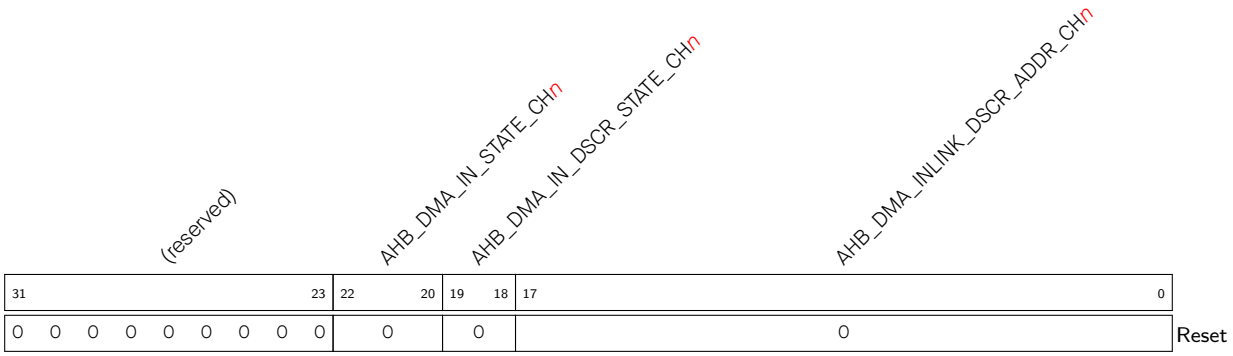
AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

AHB_DMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 3.32. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-4) (0x0118+0x100**n*)

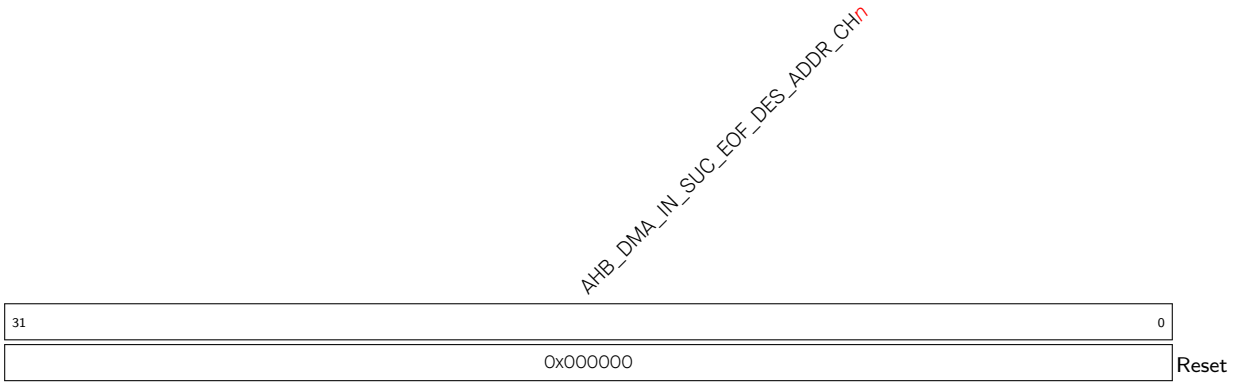


AHB_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address to be processed. (RO)

AHB_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

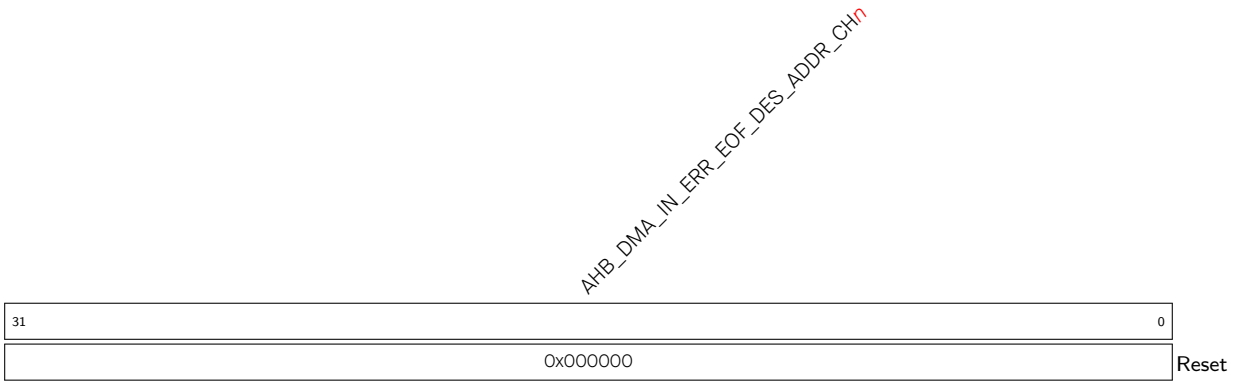
AHB_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.33. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x011C+0x100**n*)



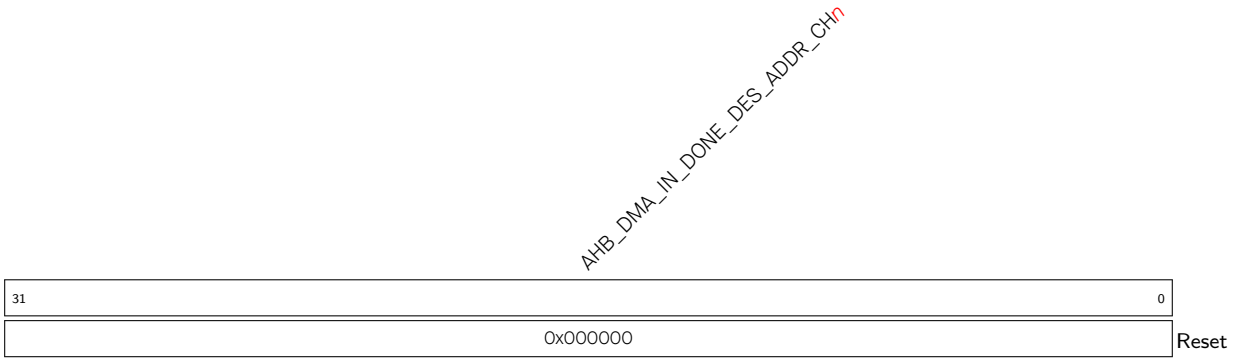
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when this descriptor has an EOF flag for RX channel *n*. (RO)

Register 3.34. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x0120+0x100**n*)



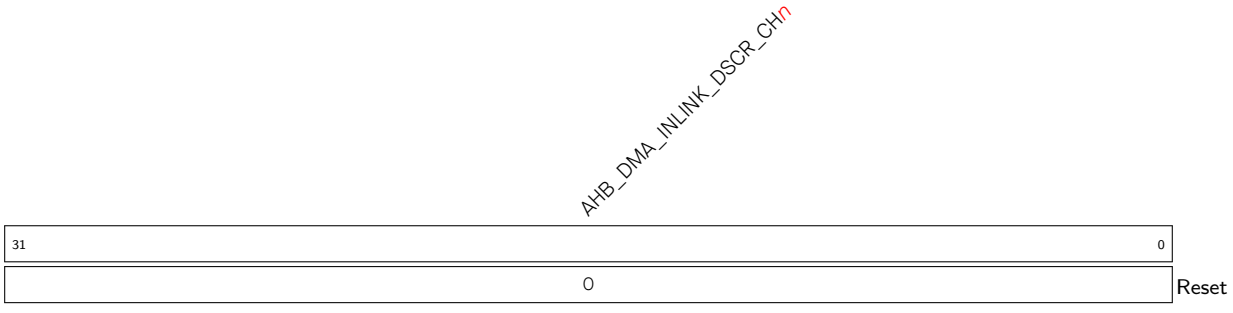
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when errors occur in the currently received data for RX channel *n*. (RO)

Register 3.35. AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x0124+0x100**n*)



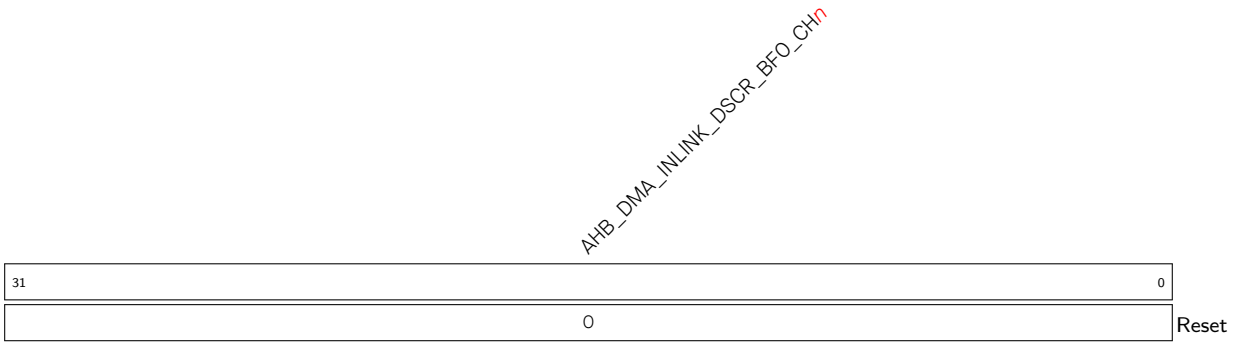
AHB_DMA_IN_DONE_DES_ADDR_CH*n* Represents the descriptor address when this descriptor is completed for RX channel *n*. (RO)

Register 3.36. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-4) (0x0128+0x100**n*)



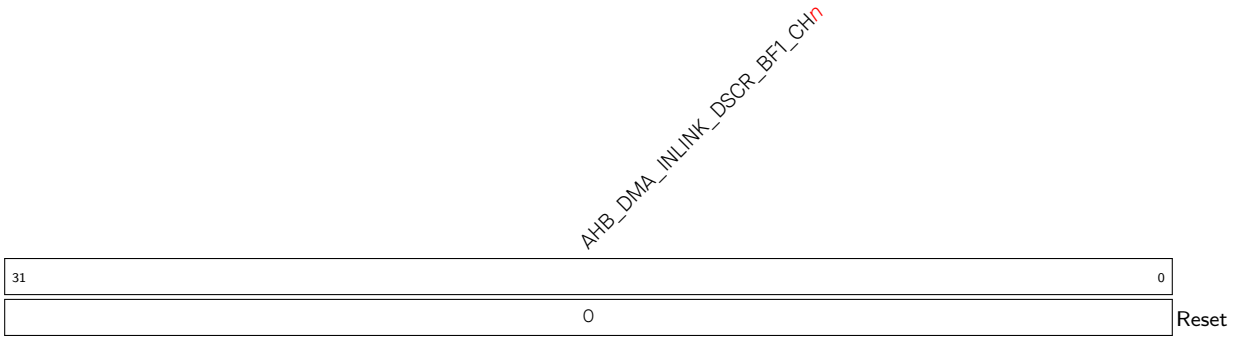
AHB_DMA_INLINK_DSCR_CH*n* Represents the next receive descriptor address pointed to by the current receive descriptor that has already been fetched. (RO)

Register 3.37. AHB_DMA_IN_DSCR_BFO_CH*n*_REG (*n*: 0-4) (0x012C+0x100n*)**



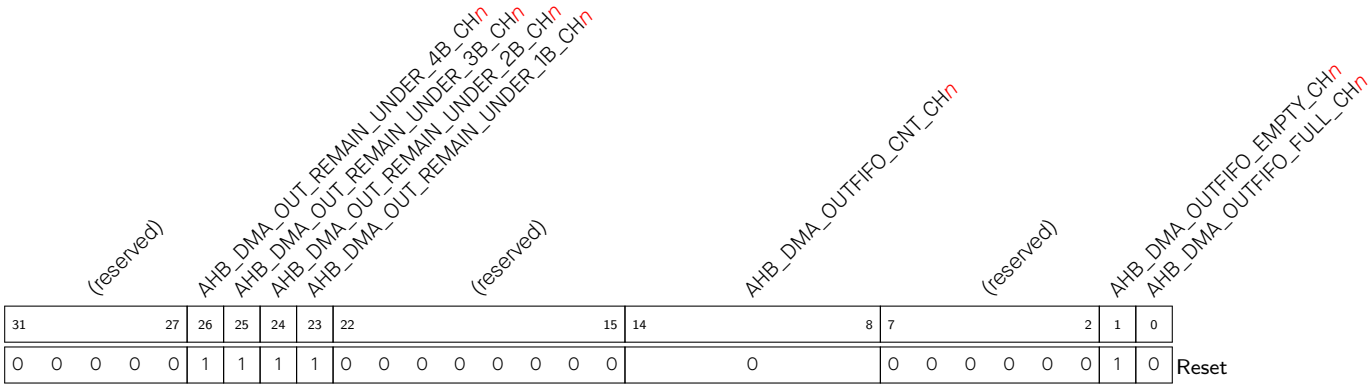
AHB_DMA_INLINK_DSCR_BFO_CH*n* Represents the current receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.38. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-4) (0x0130+0x100n*)**



AHB_DMA_INLINK_DSCR_BF1_CH*n* Represents the previous receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.39. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-4) (0x0188+0x100*n)



AHB_DMA_OUTFIFO_FULL_CH*n* Represents whether TX FIFO is full for TX channel *n*.

0: Not full

1: Full

(RO)

AHB_DMA_OUTFIFO_EMPTY_CH*n* Represents whether TX FIFO is empty for TX channel *n*.

0: Not empty

1: Empty

(RO)

AHB_DMA_OUTFIFO_CNT_CH*n* Represents the number of data bytes in TX FIFO for TX channel *n*.

Measurement unit: byte. (RO)

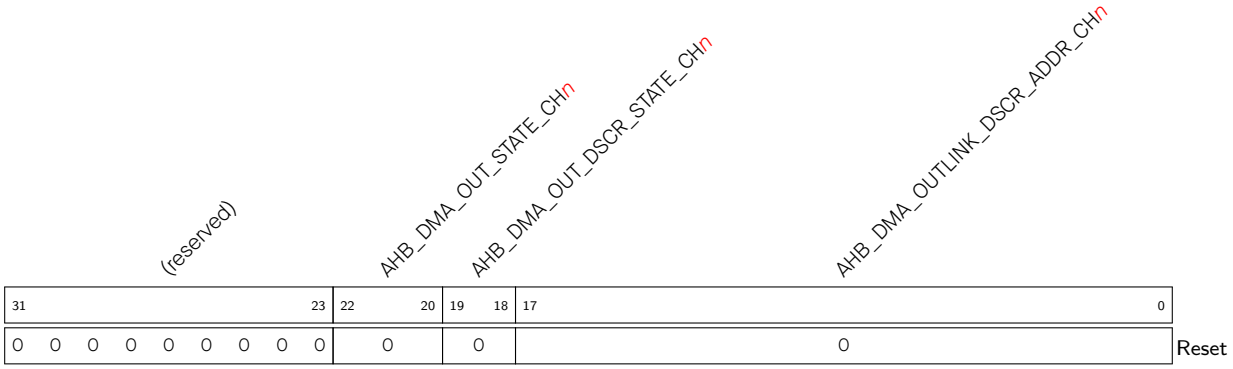
AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

Register 3.40. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-4) (0x0198+0x100**n*)

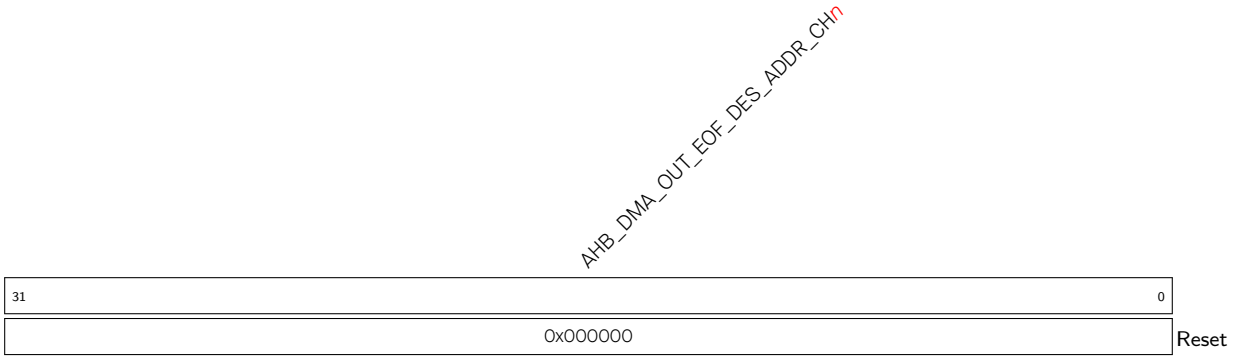


AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address to be processed. (RO)

AHB_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

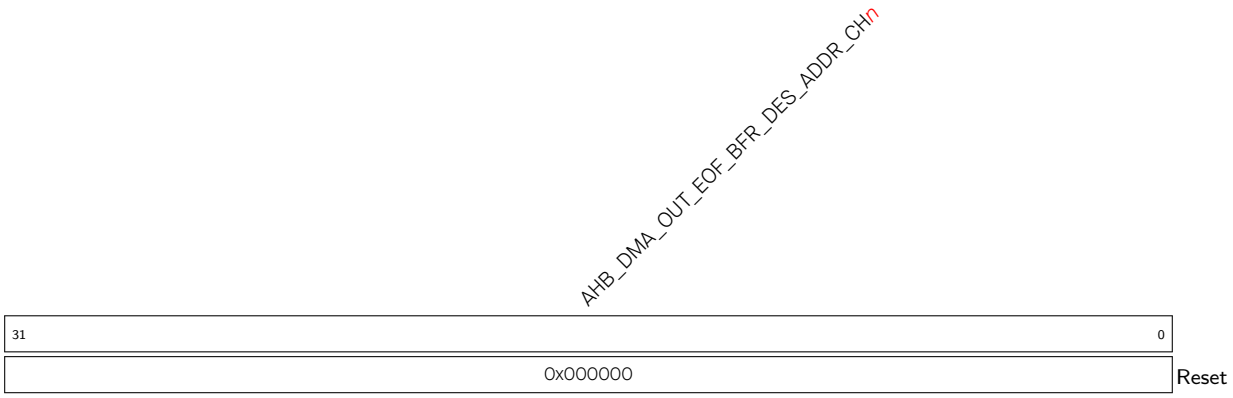
AHB_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.41. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x019C+0x100**n*)



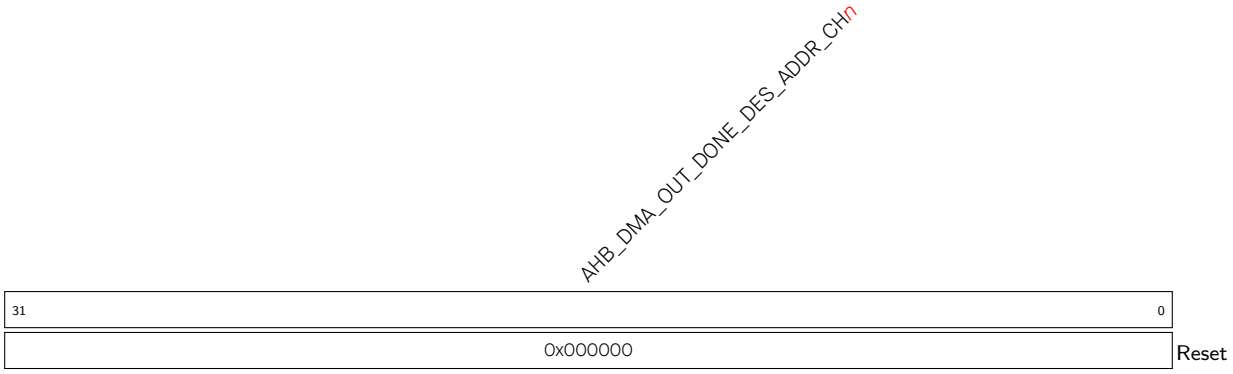
AHB_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the transmit descriptor address when this descriptor has an EOF flag for TX channel *n*. (RO)

Register 3.42. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x01A0+0x100*n*)



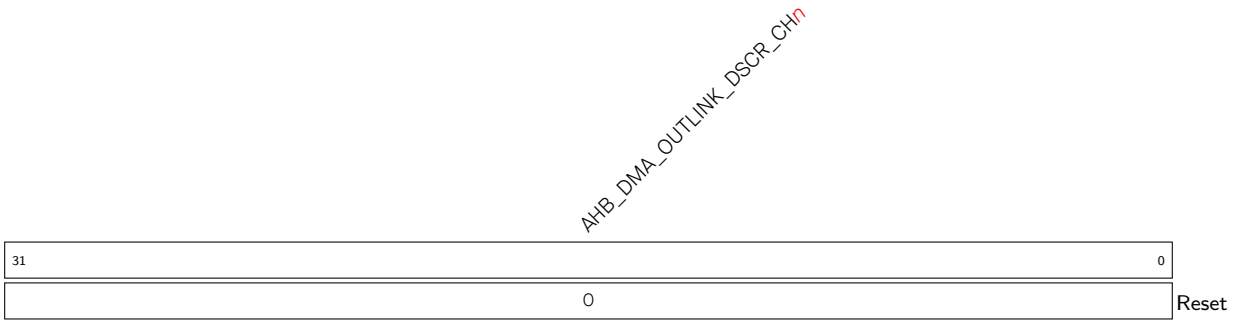
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the transmit descriptor address before the last transmit descriptor for TX channel *n*. (RO)

Register 3.43. AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG (*n*: 0-4) (0x01A4+0x100*n*)



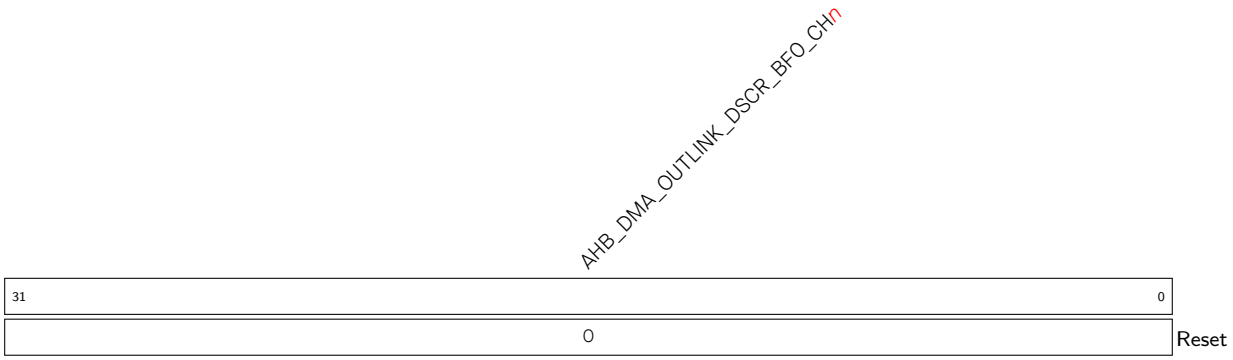
AHB_DMA_OUT_DONE_DES_ADDR_CH*n* Represents the descriptor address when this descriptor is completed for TX channel *n*. (RO)

Register 3.44. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-4) (0x01A8+0x100*n*)



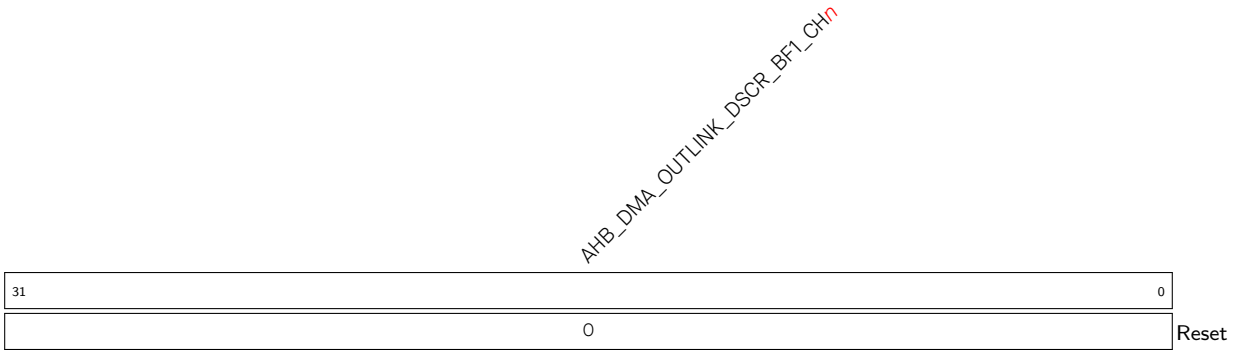
AHB_DMA_OUTLINK_DSCR_CH*n* Represents the next transmit descriptor address pointed to by the current transmit descriptor that has already been fetched. (RO)

Register 3.45. AHB_DMA_OUT_DSCR_BFO_CH*n*_REG (*n*: 0-4) (0x01AC+0x100**n*)



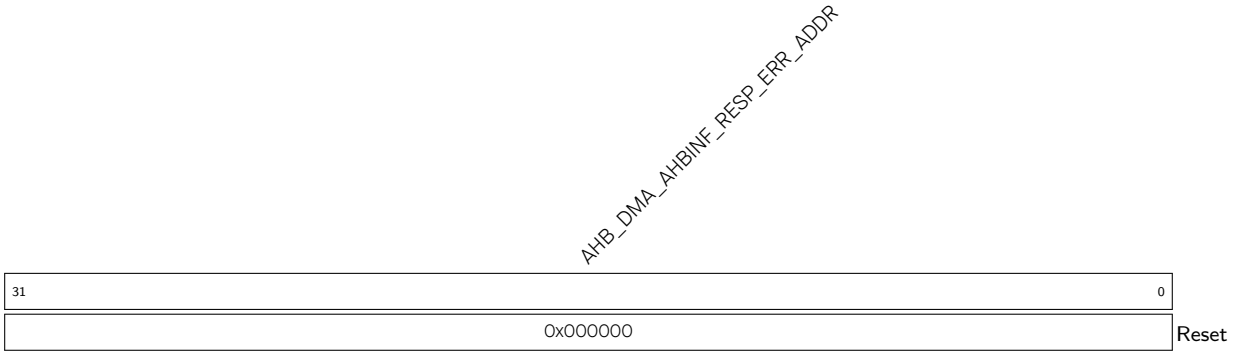
AHB_DMA_OUTLINK_DSCR_BFO_CH*n* Represents the current transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.46. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-4) (0x01B0+0x100**n*)



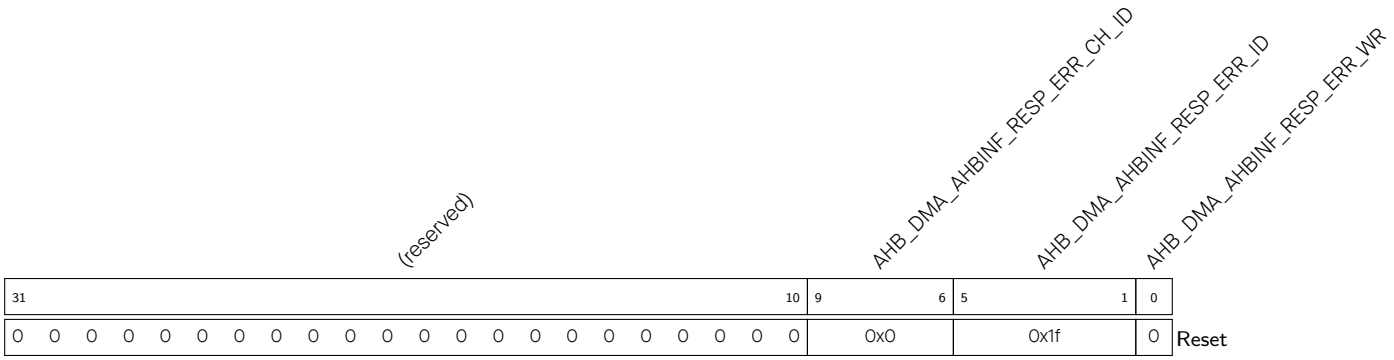
AHB_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the previous transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.47. AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG (0x0620)



AHB_DMA_AHBINF_RESP_ERR_ADDR Represents the address of the current AHB bus transfer that triggers an error response. (RO)

Register 3.48. AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG (0x0624)



AHB_DMA_AHBINF_RESP_ERR_WR Represents whether the AHB response error is caused by a write request.

0: Read request

1: Write request

(RO)

AHB_DMA_AHBINF_RESP_ERR_ID Represents the AHB response error request ID.

0: UHCI0

1: I2S0-0

2: I2S0-1

3: I2S0-2

4: I2S0-3

5: I2S1-0

6: I2S1-1

7: I2S1-2

8: I2S1-3

10: RMT

11: ASRC-0

12: ASRC-1

13: CORDIC

16: GDMA-AHB-HP channel 0 memory to memory transfer

17: GDMA-AHB-HP channel 1 memory to memory transfer

18: GDMA-AHB-HP channel 2 memory to memory transfer

19: GDMA-AHB-HP channel 3 memory to memory transfer

20: GDMA-AHB-HP channel 4 memory to memory transfer

Others: Invalid

(RO)

AHB_DMA_AHBINF_RESP_ERR_CH_ID Represents the AHB response error request channel ID.

Bit[3]: Represents the channel direction

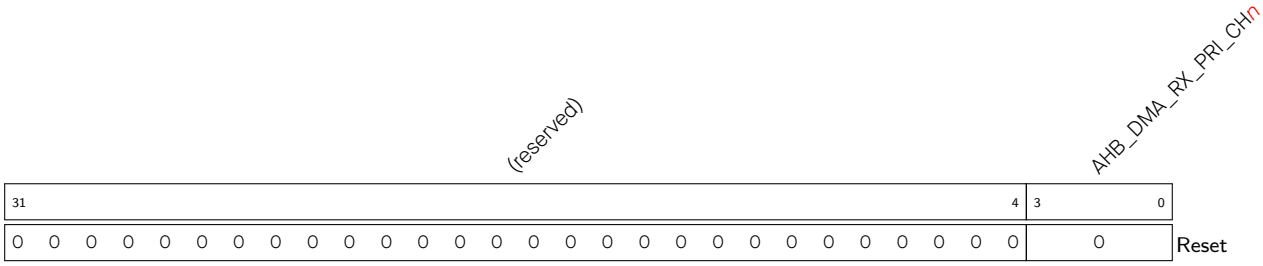
0: RX channel

1: TX channel

Bit[0:2]: Represents the channel number. Value range: 0 ~4

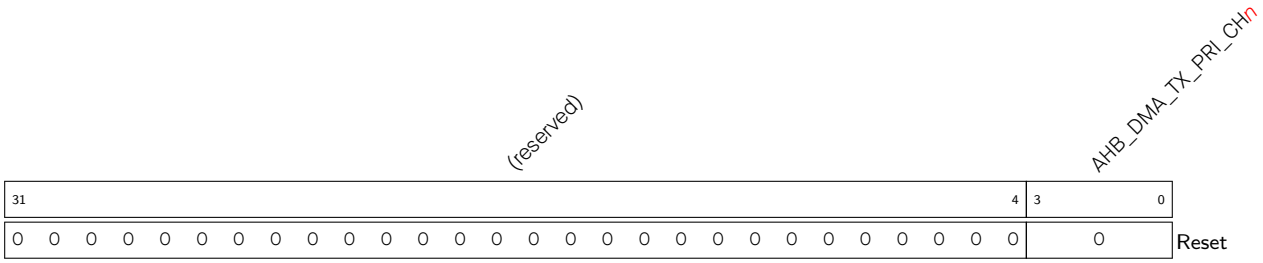
(RO)

Register 3.49. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-4) (0x0134+0x100**n*)



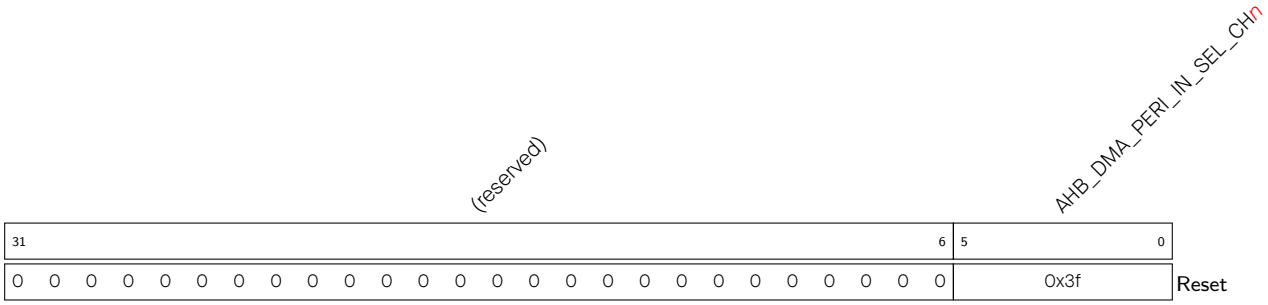
AHB_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

Register 3.50. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-4) (0x01B4+0x100**n*)



AHB_DMA_TX_PRI_CH*n* Configures the priority of TX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

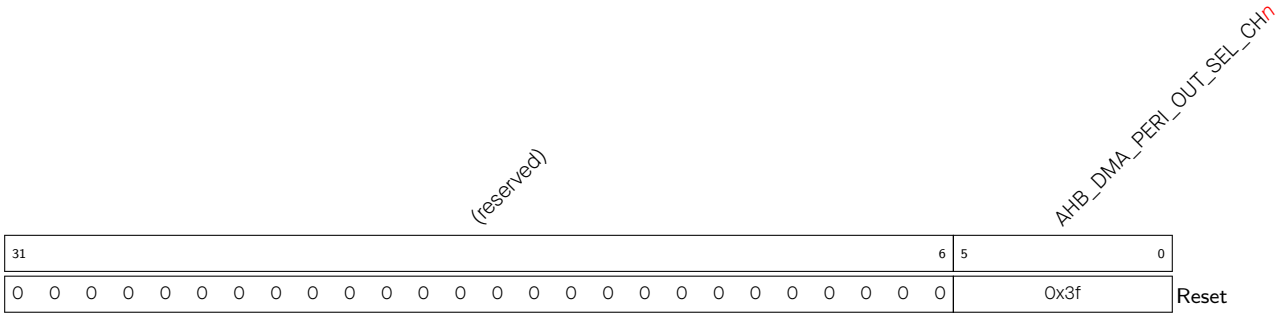
Register 3.51. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-4) (0x0138+0x100n*)**



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

0: UHCI0
1: I2S0-0
2: I2S0-1
3: I2S0-2
4: I2S0-3
5: I2S1-0
6: I2S1-1
7: I2S1-2
8: I2S1-3
9: Dummy-9
10: RMT
11: ASRC-0
12: ASRC-1
13: CORDIC
14 ~ 63: Invalid value
(R/W)

Register 3.52. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-4) (0x01B8+0x100**n*)



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

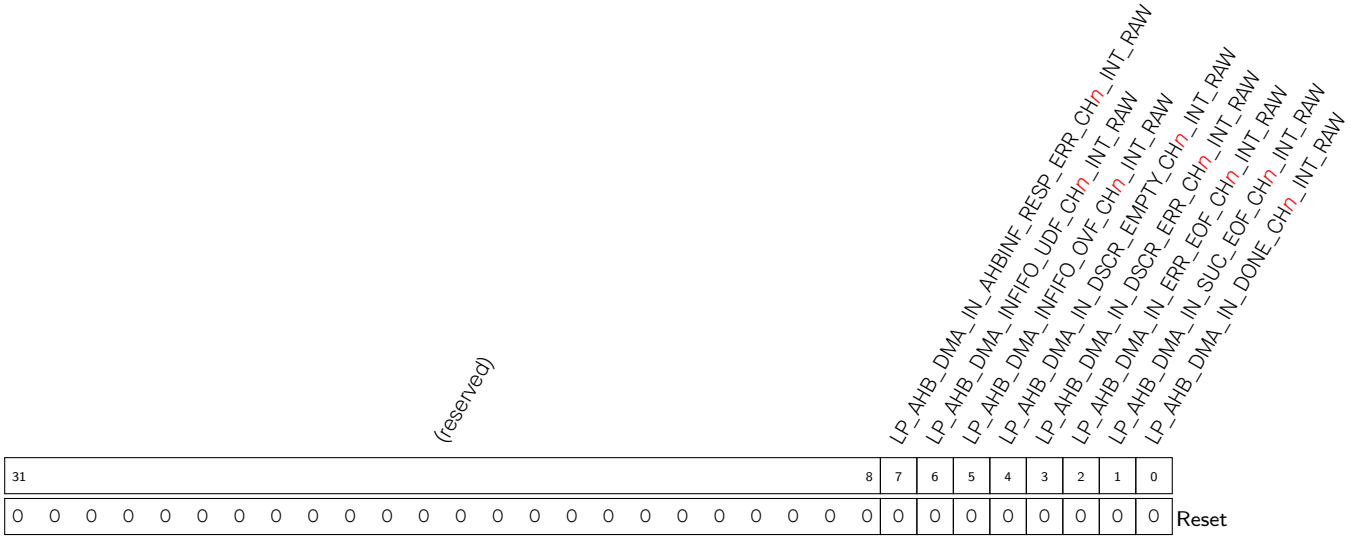
0: UHCI0
1: I2S0-0
2: I2S0-1
3: I2S0-2
4: I2S0-3
5: I2S1-0
6: I2S1-1
7: I2S1-2
8: I2S1-3
9: Dummy-9
10: RMT
11: ASRC-0
12: ASRC-1
13: CORDIC
14 ~ 63: Invalid value
(R/W)

3.9.2 GDMA-AHB-LP Registers

The addresses in this section are relative to GDMA-AHB-LP base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 3.53. LP_AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-1) (0x0000+0x10n*)**



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

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

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

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

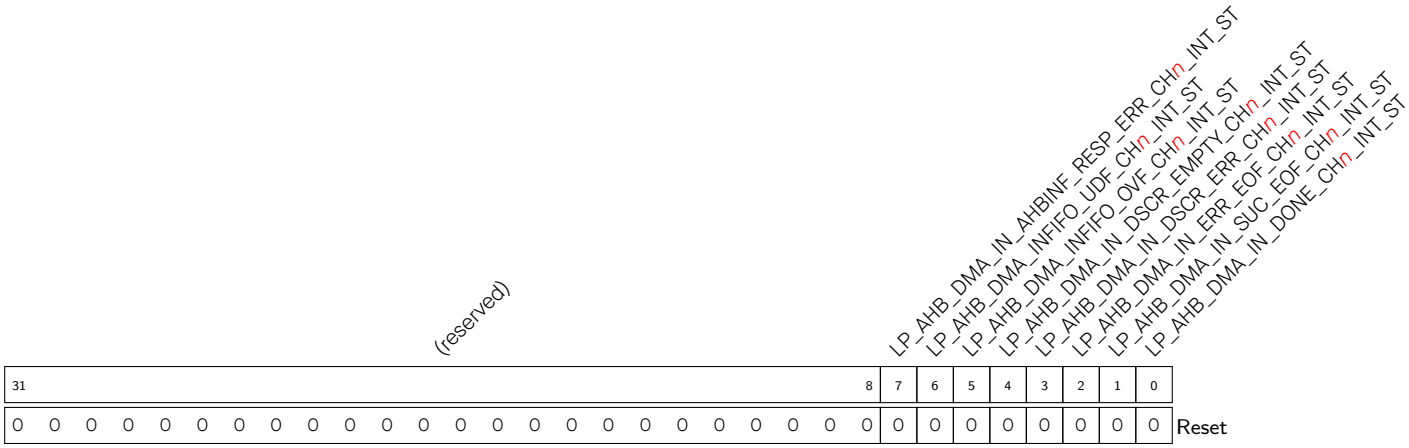
LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.54. LP_AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-1) (0x0004+0x10n*)**



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

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

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

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

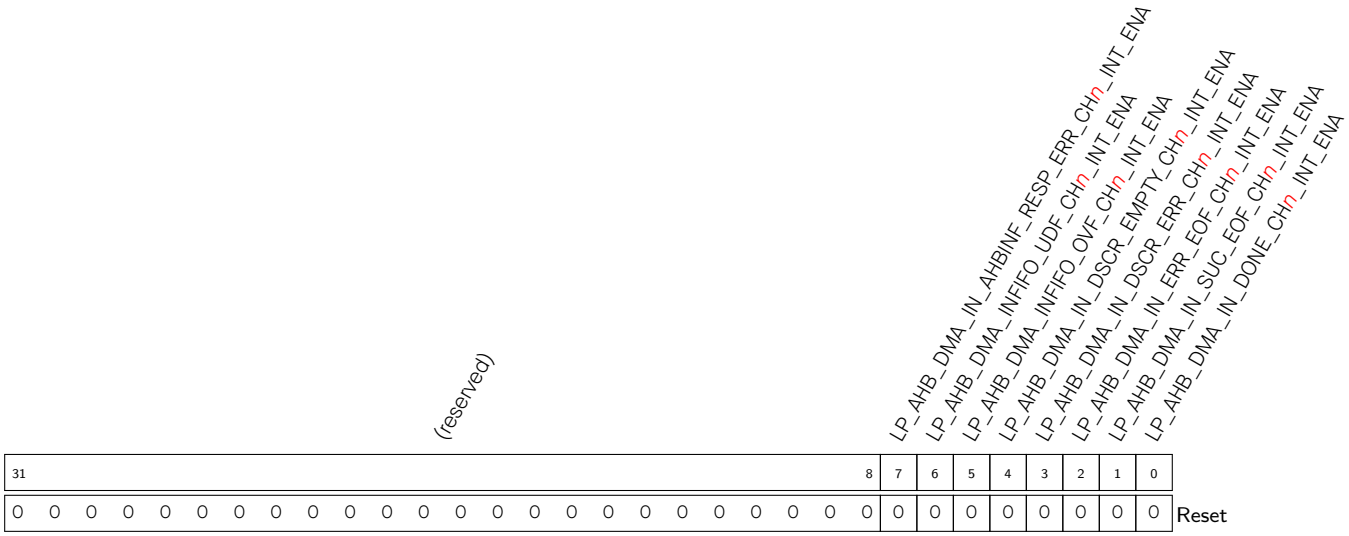
LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (RO)

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

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

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

Register 3.55. LP_AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-1) (0x0008+0x10**n*)



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

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

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

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

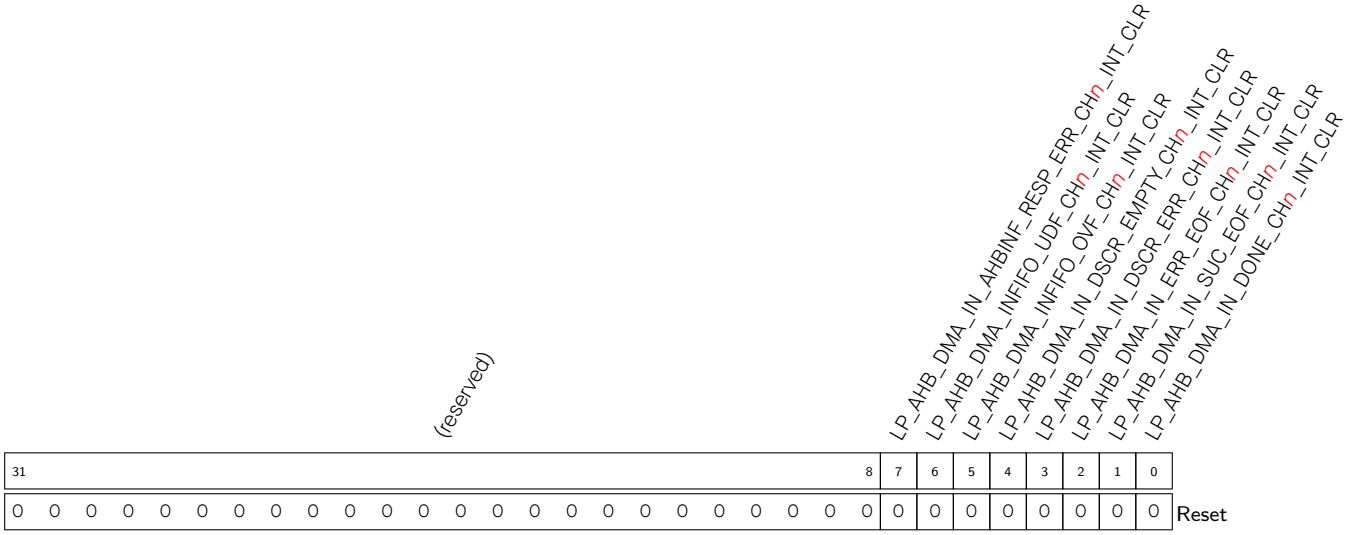
LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

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

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

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

Register 3.56. LP_AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-1) (0x000C+0x10n*)**



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

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

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

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

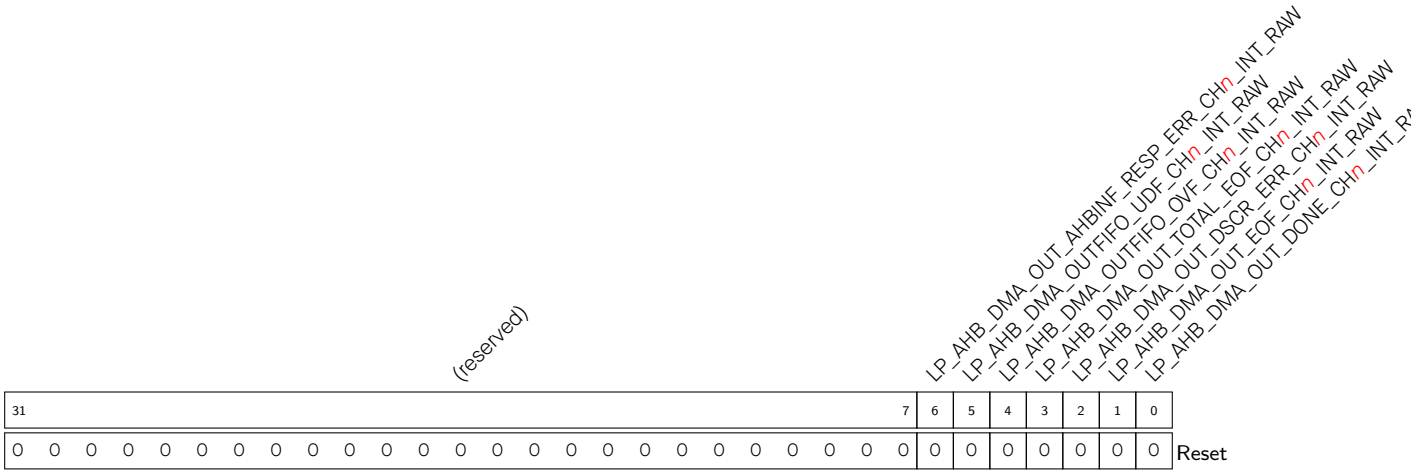
LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR Write 1 to clear LP_AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT. (WT)

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

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

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

Register 3.57. LP_AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-1) (0x0050+0x10**n*)



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

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

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

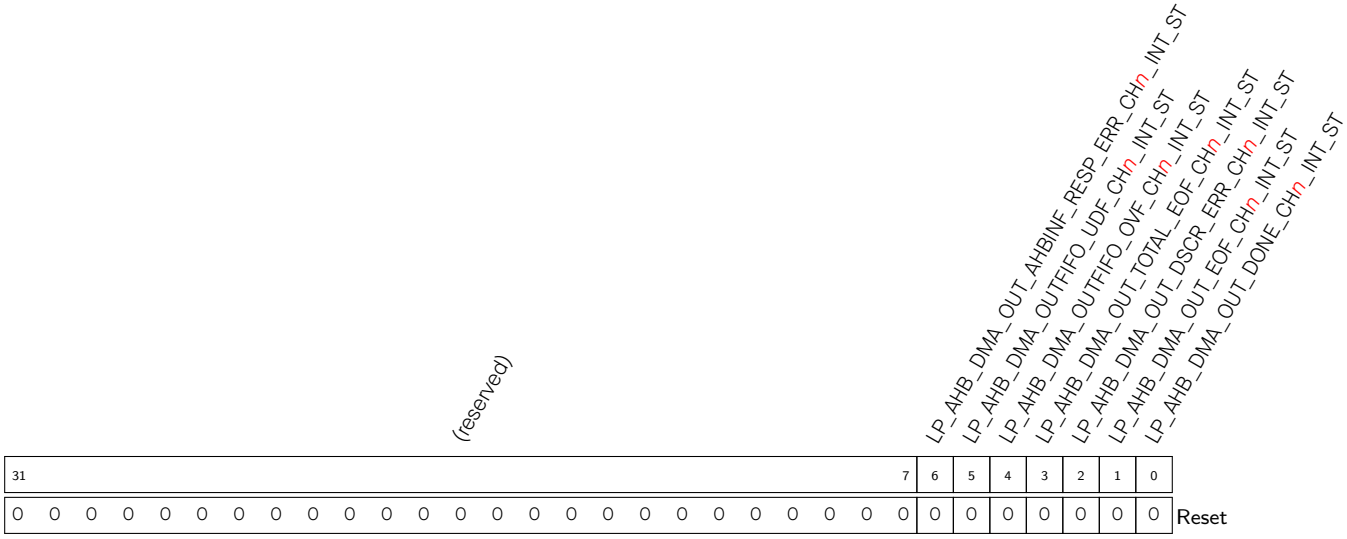
LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

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

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

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

Register 3.58. LP_AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-1) (0x0054+0x10n*)**



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

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

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

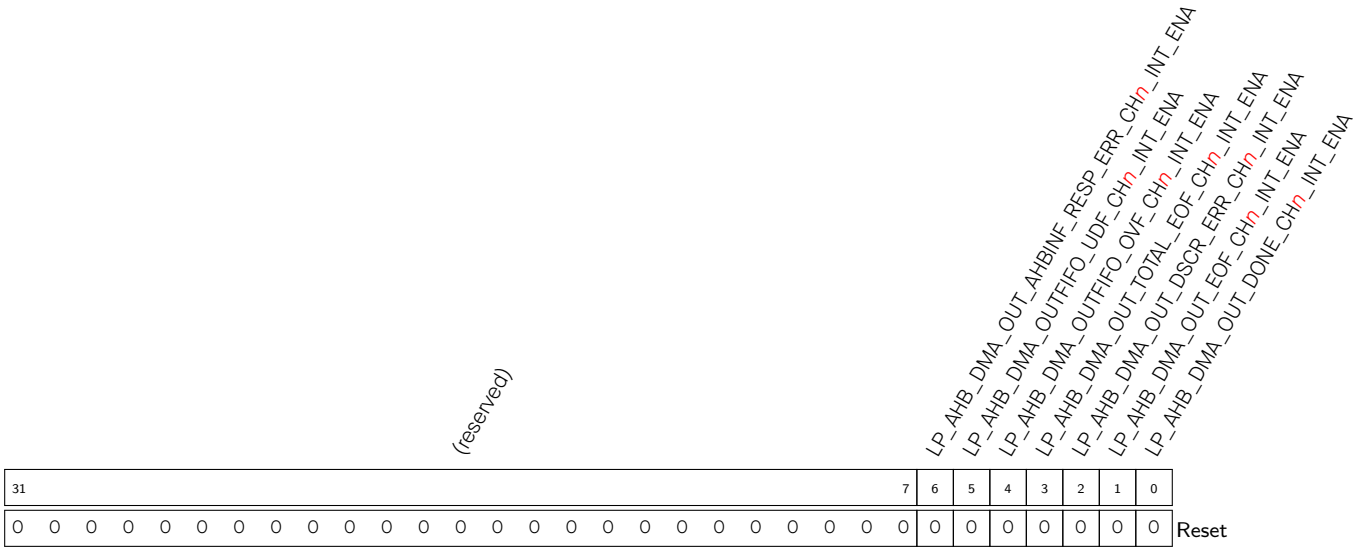
LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (RO)

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

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

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

Register 3.59. LP_AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-1) (0x0058+0x10n*)**



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

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

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

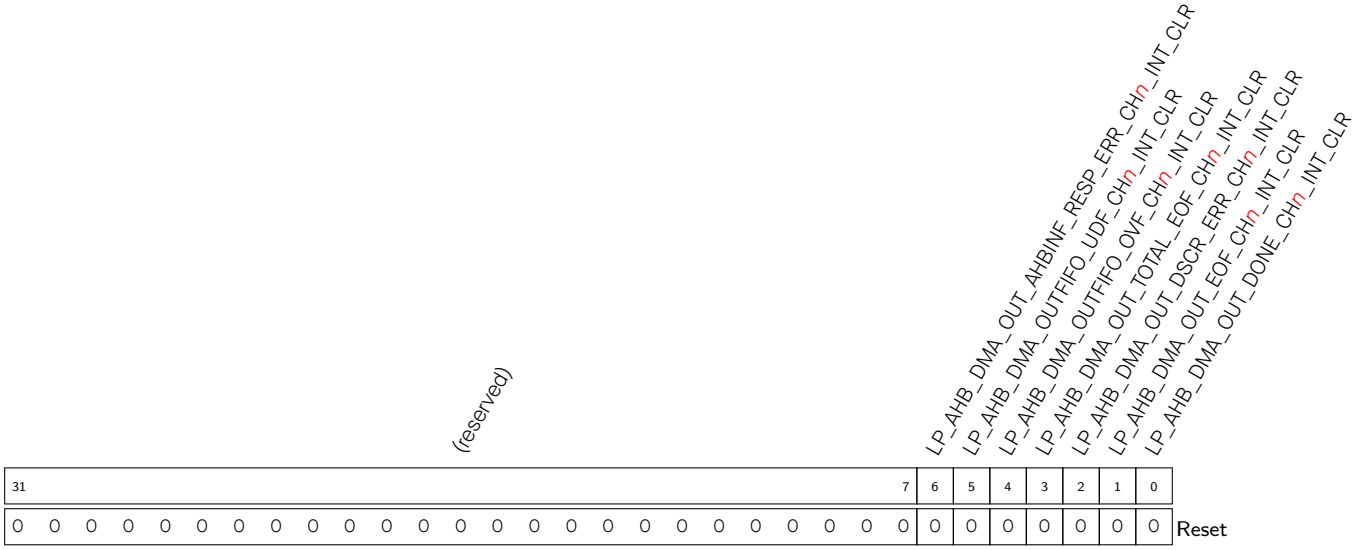
LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

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

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

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

Register 3.60. LP_AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-1) (0x005C+0x10*n*)



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

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

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

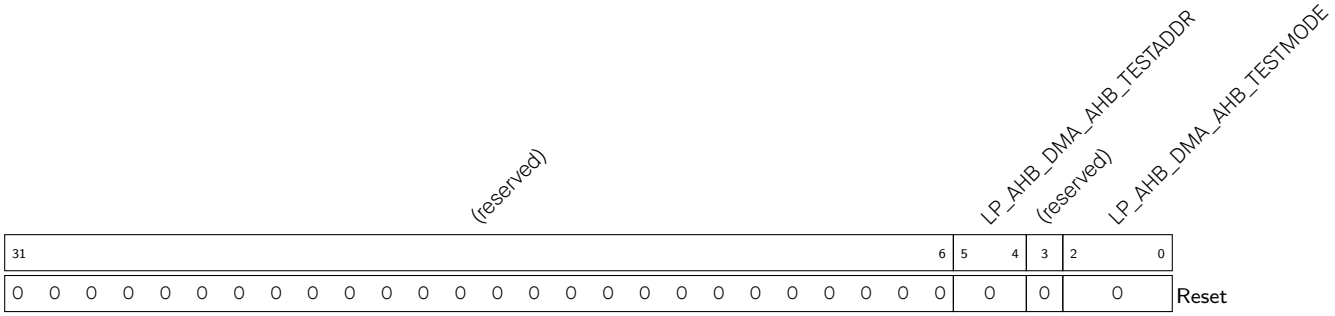
LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear LP_AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT. (WT)

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

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

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

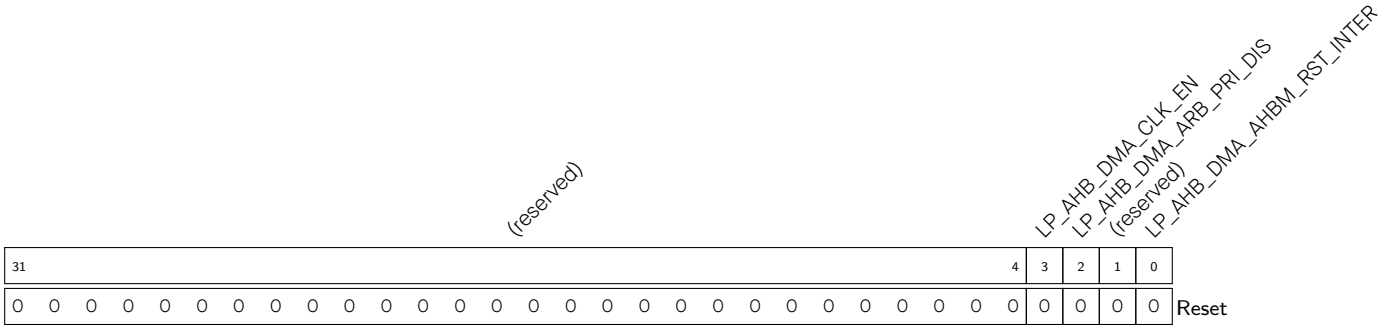
Register 3.61. LP_AHB_DMA_AHB_TEST_REG (0x00A0)



LP_AHB_DMA_AHB_TESTMODE Reserved. (R/W)

LP_AHB_DMA_AHB_TESTADDR Reserved. (R/W)

Register 3.62. LP_AHB_DMA_MISC_CONF_REG (0x00A4)



LP_AHB_DMA_AHBM_RST_INTER Configures reset state of the internal LP AHB FSM.

0: Release reset

1: Reset

(R/W)

LP_AHB_DMA_ARB_PRI_DIS Configures whether to disable fixed-priority channel arbitration.

0: Enable

1: Disable

(R/W)

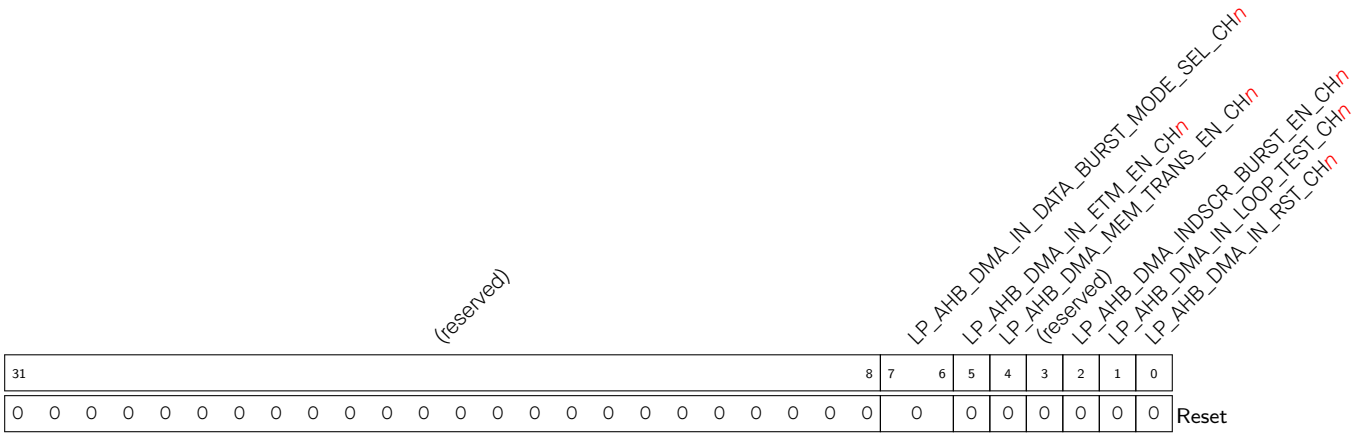
LP_AHB_DMA_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 3.63. LP_AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-1) (0x0100+0x100n*)**



LP_AHB_DMA_IN_RST_CH*n* Configures reset state of LP AHB DMA RX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

LP_AHB_DMA_IN_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for RX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

LP_AHB_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read receive descriptors.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_MEM_TRANS_EN_CH*n* Configures whether to enable memory-to-memory data transfer.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel *n*.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* Configures burst transfer mode for RX channel *n*.

0: Single

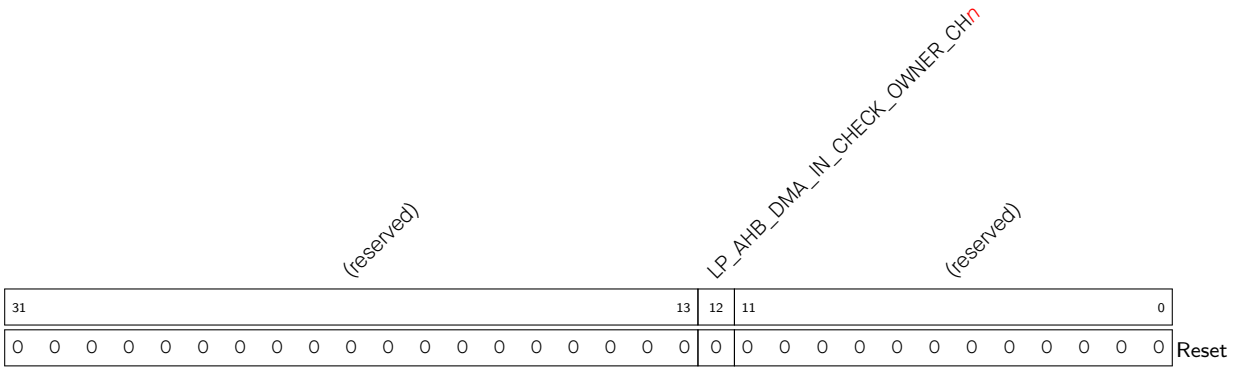
1: INCR4

2: INCR8

3: INCR16

(R/W)

Register 3.64. LP_AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-1) (0x0104+0x100**n*)



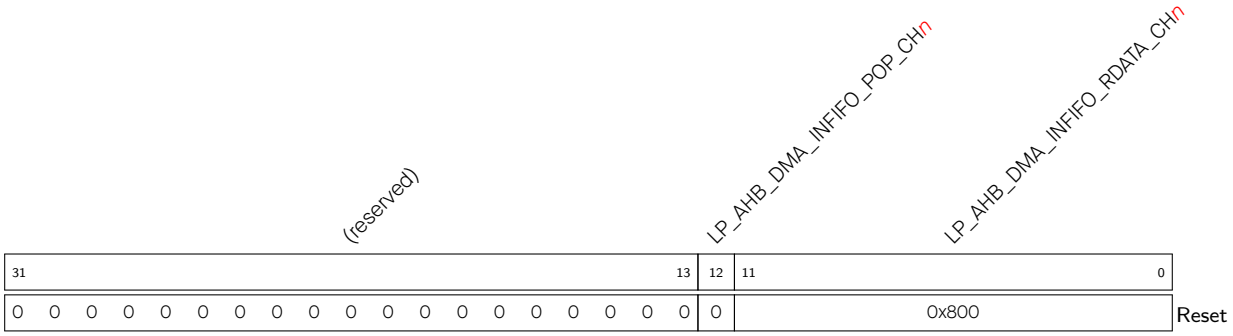
LP_AHB_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.65. LP_AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-1) (0x010C+0x100**n*)



LP_AHB_DMA_INFIFO_RDATA_CH*n* Data received from LP AHB DMA FIFO. (RO)

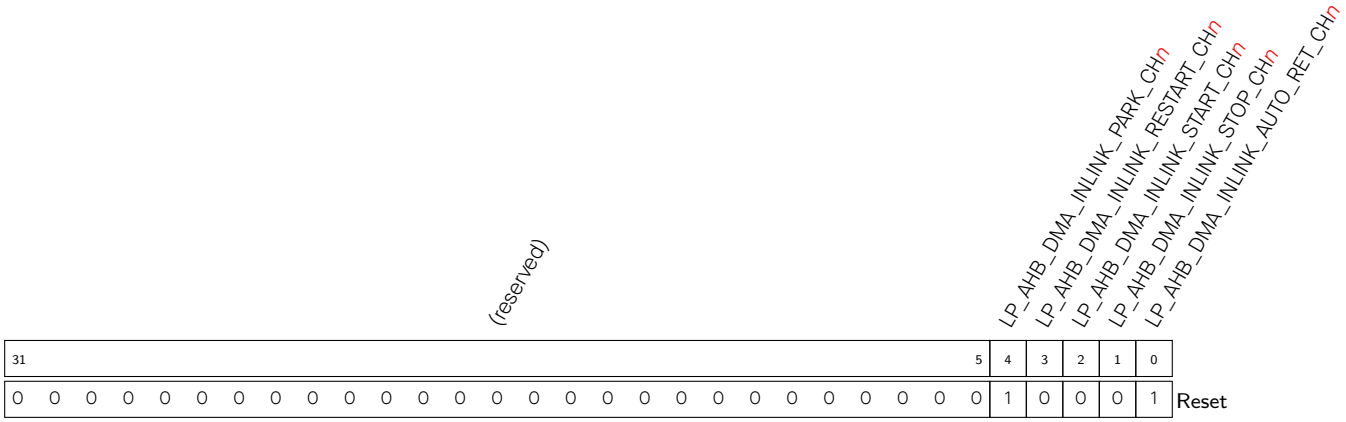
LP_AHB_DMA_INFIFO_POP_CH*n* Configures whether to pop data from LP AHB DMA FIFO for RX channel *n*.

0: Invalid. No effect

1: Pop

(WT)

Register 3.66. LP_AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-1) (0x0110+0x100n*)**



LP_AHB_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor address when an error occurs in the current received data for RX channel *n*.

0: Not return

1: Return

(R/W)

LP_AHB_DMA_INLINK_STOP_CH*n* Configures whether to stop processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

LP_AHB_DMA_INLINK_START_CH*n* Configures whether to start processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Start

(WT)

LP_AHB_DMA_INLINK_RESTART_CH*n* Configures whether to restart descriptor processing for RX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

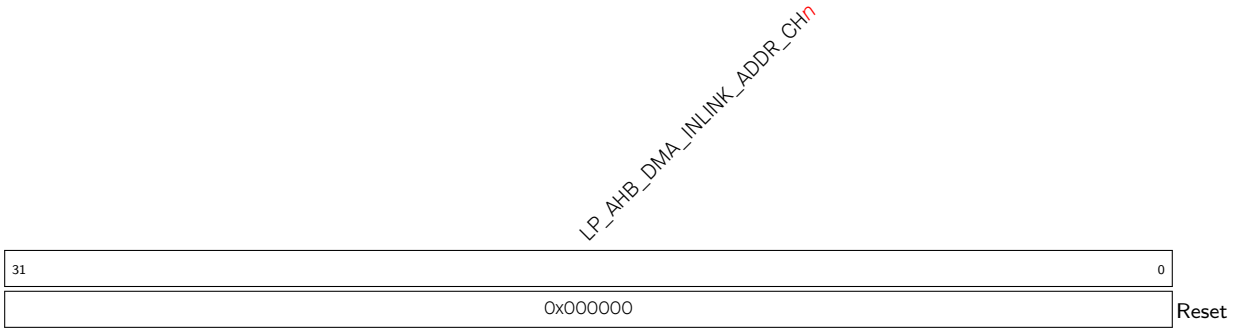
LP_AHB_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor FSM for RX channel *n*.

0: Running

1: Idle

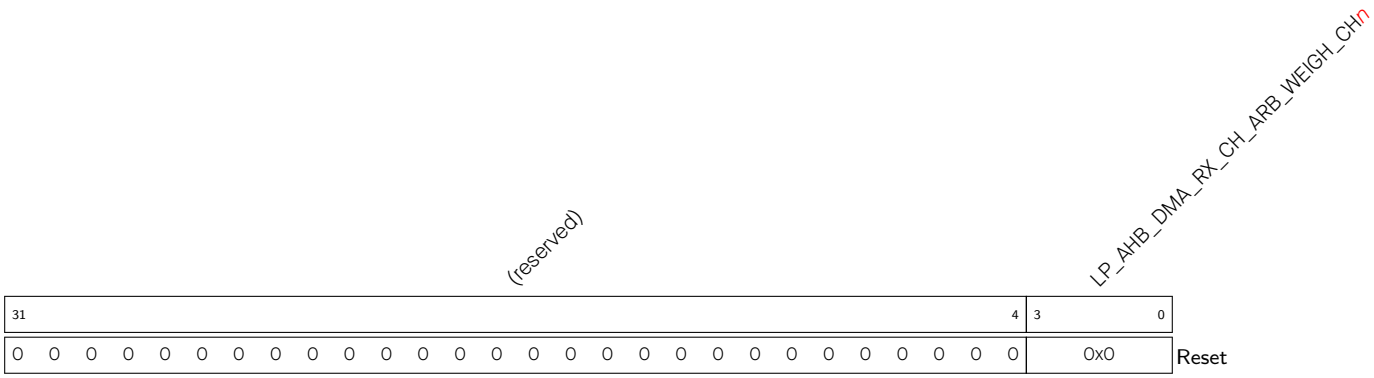
(RO)

Register 3.67. LP_AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-1) (0x0114+0x100n*)**



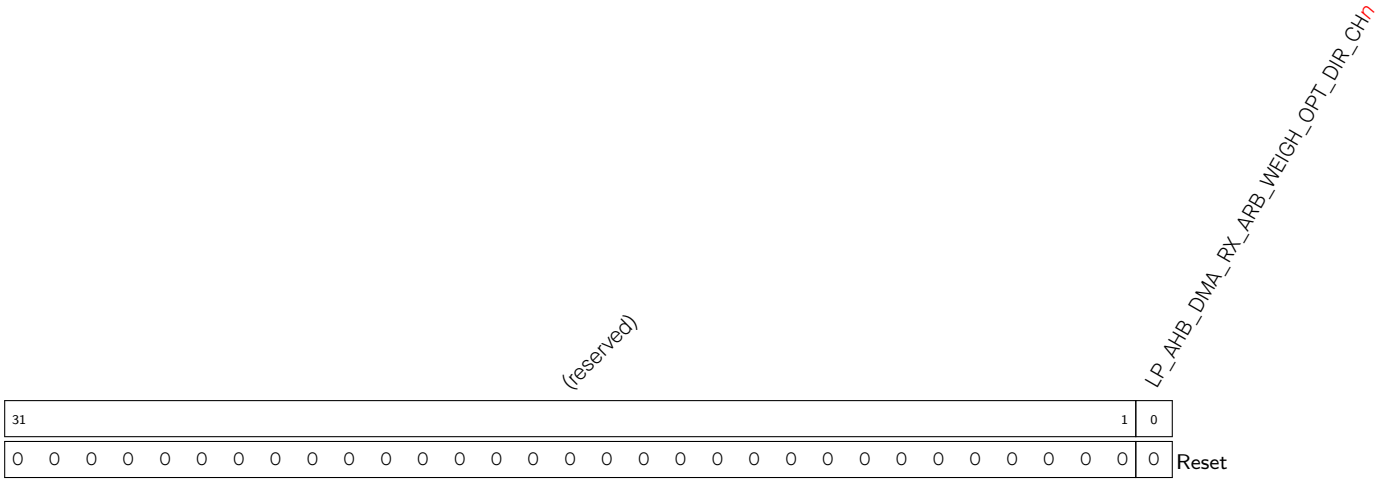
LP_AHB_DMA_INLINK_ADDR_CH*n* Configures the first receive descriptor address for RX channel *n*. (R/W)

Register 3.68. LP_AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-1) (0x013C+0x100n*)**



LP_AHB_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of RX channel *n*. Measurement unit: token. (R/W)

Register 3.69. LP_AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-1) (0x0140+0x100n*)**



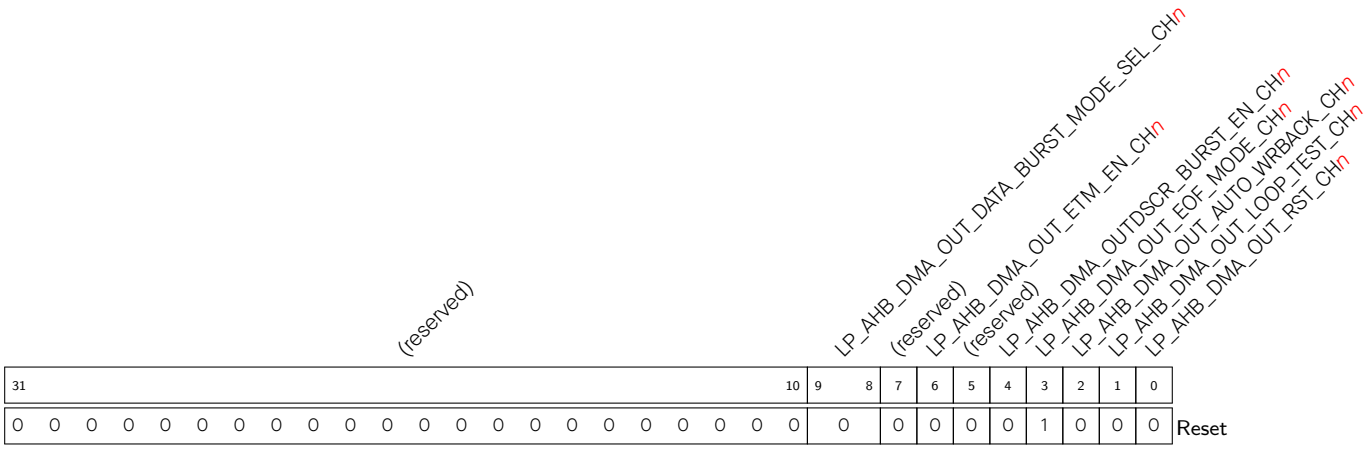
LP_AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for RX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.70. LP_AHB_DMA_OUT_CONFO_CH*n*_REG (*n*: 0-1) (0x0180+0x100**n*)



LP_AHB_DMA_OUT_RST_CH*n* Configures reset state of LP AHB DMA TX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

LP_AHB_DMA_OUT_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for TX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

LP_AHB_DMA_OUT_AUTO_WRBACK_CH*n* Configures whether to enable automatic outlink write-back when all data in the TX FIFO has been transmitted for TX channel *n*.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_OUT_EOF_MODE_CH*n* Configures when to generate the EOF flag for TX channel *n*.

0: Generate when data to be transmitted has been pushed into the FIFO.

1: Generate when data to be transmitted has been popped from the FIFO.

(R/W)

LP_AHB_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* to read transmit descriptors.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_OUT_ETM_EN_CH*n* Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

LP_AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n* Configures burst transfer mode for TX channel *n*.

0: Single

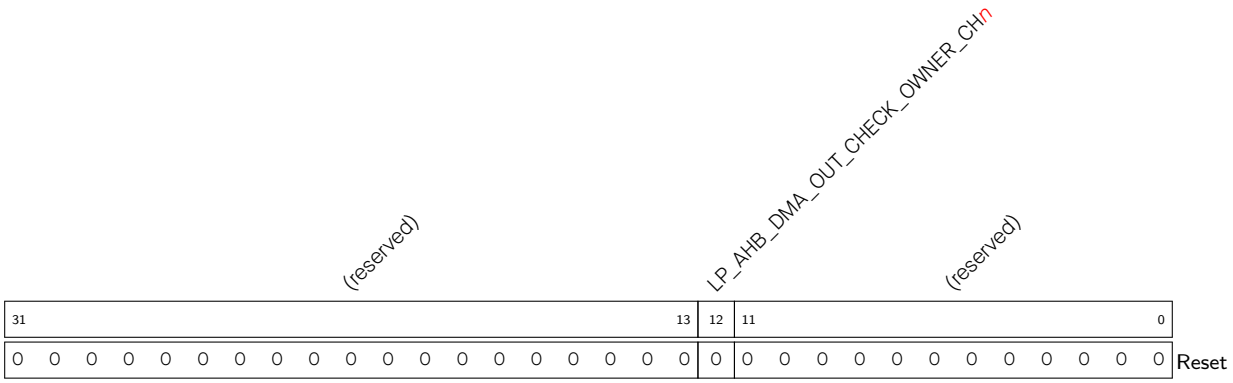
1: INCR4

2: INCR8

3: INCR16

(R/W)

Register 3.71. LP_AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-1) (0x0184+0x100**n*)



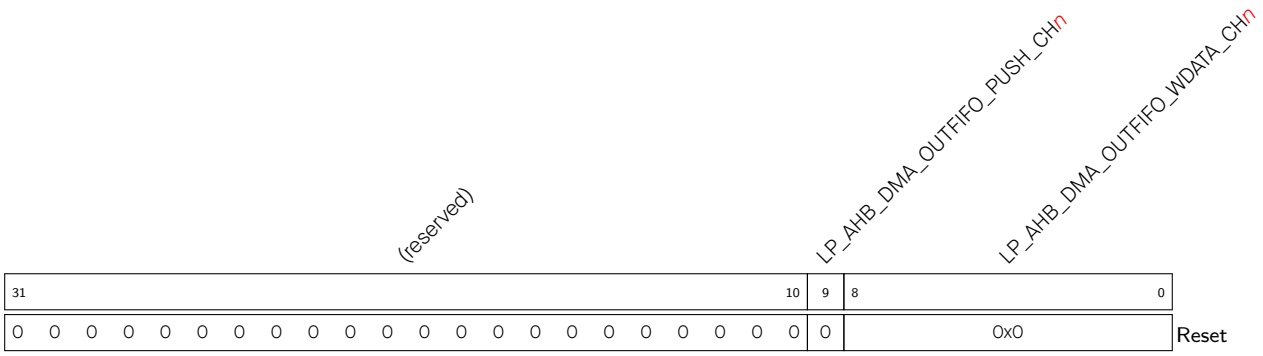
LP_AHB_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.72. LP_AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-1) (0x018C+0x100**n*)



LP_AHB_DMA_OUTFIFO_WDATA_CH*n* Data to be transmitted to LP AHB DMA FIFO. (R/W)

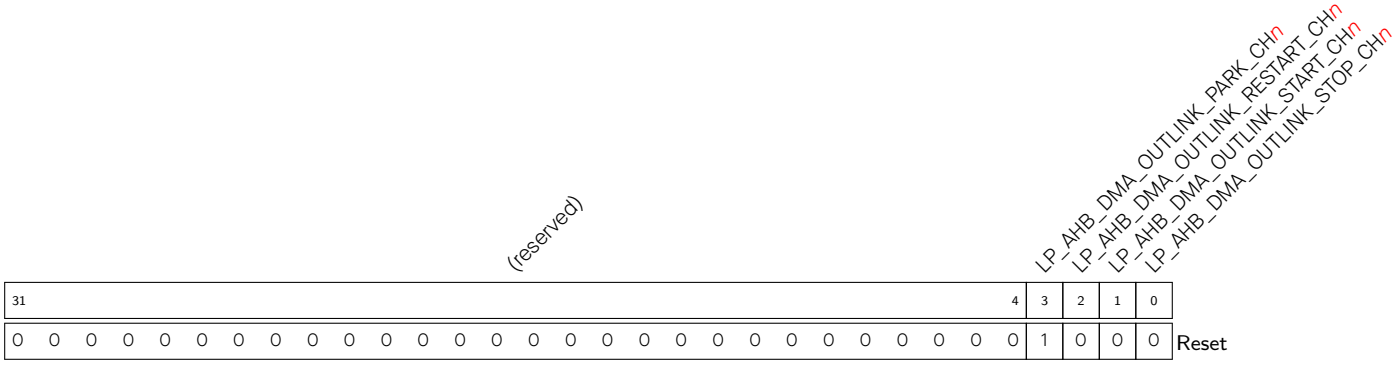
LP_AHB_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into LP AHB DMA FIFO for TX channel *n*.

0: Invalid. No effect

1: Push

(WT)

Register 3.73. LP_AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-1) (0x0190+0x100**n*)



LP_AHB_DMA_OUTLINK_STOP_CH*n* Configures whether to stop processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

LP_AHB_DMA_OUTLINK_START_CH*n* Configures whether to start processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Start

(WT)

LP_AHB_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart descriptor processing for TX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

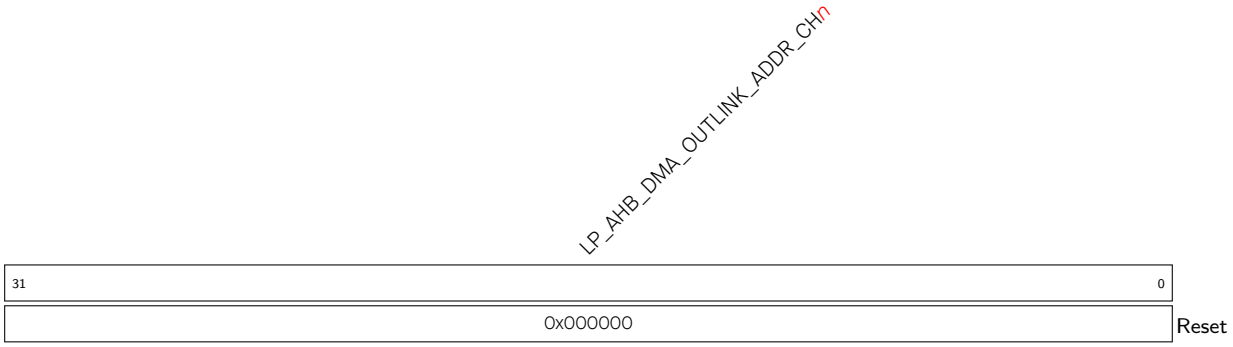
LP_AHB_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor FSM for TX channel *n*.

0: Running

1: Idle

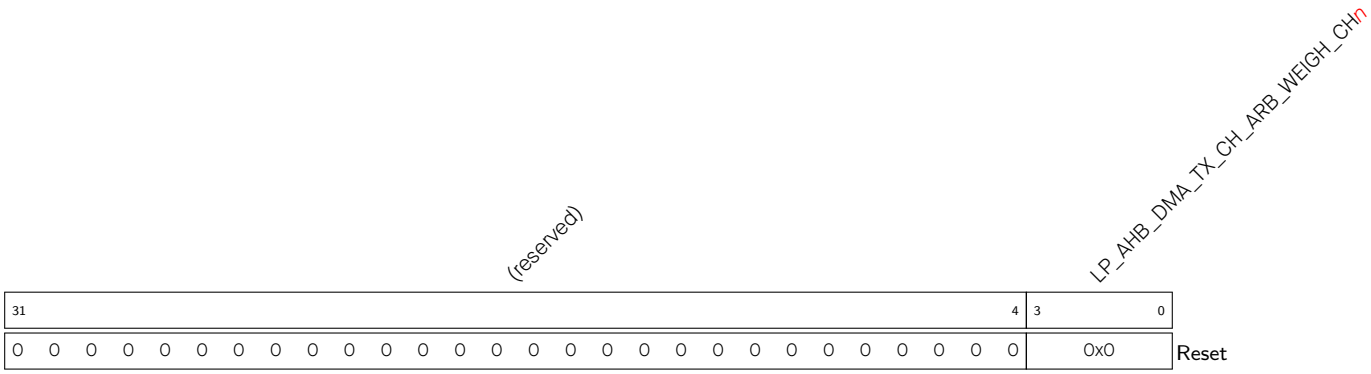
(RO)

Register 3.74. LP_AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-1) (0x0194+0x100n*)**



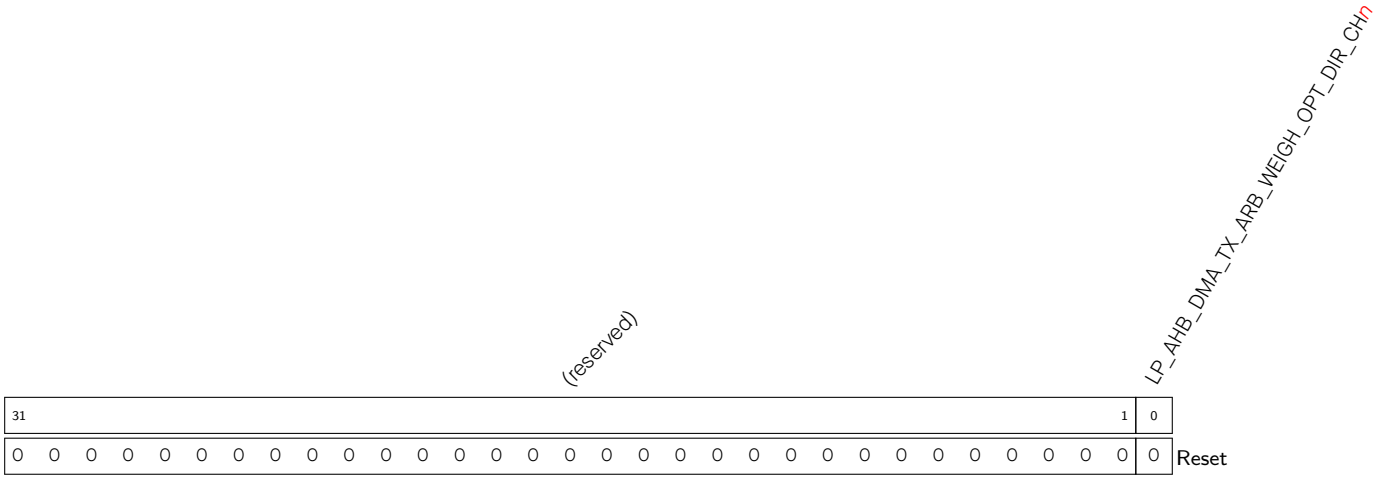
LP_AHB_DMA_OUTLINK_ADDR_CH*n* Configures the first transmit descriptor address for TX channel *n*. (R/W)

Register 3.75. LP_AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-1) (0x01BC+0x100n*)**



LP_AHB_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of TX channel *n*. Measurement unit: token. (R/W)

Register 3.76. LP_AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-1) (0x01C0+0x100n*)**



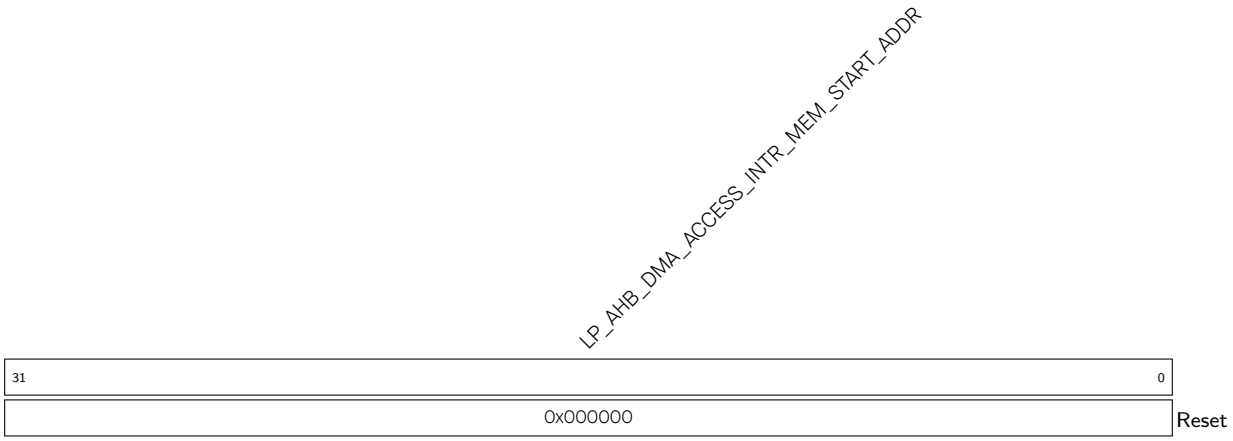
LP_AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.77. LP_AHB_DMA_INTR_MEM_START_ADDR_REG (0x0600)



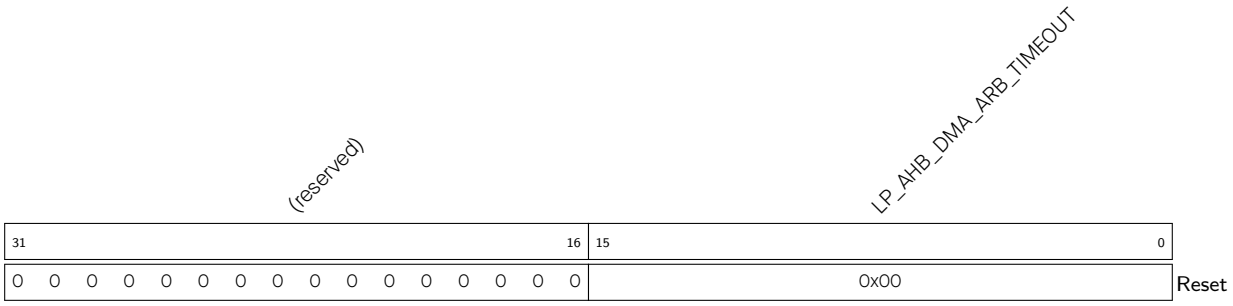
LP_AHB_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible memory space. (R/W)

Register 3.78. LP_AHB_DMA_INTR_MEM_END_ADDR_REG (0x0604)



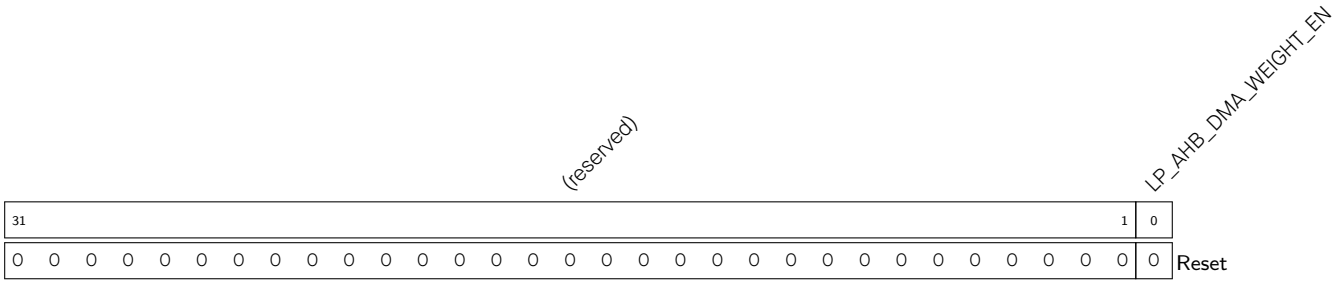
LP_AHB_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible memory space. (R/W)

Register 3.79. LP_AHB_DMA_ARB_TIMEOUT_REG (0x0608)



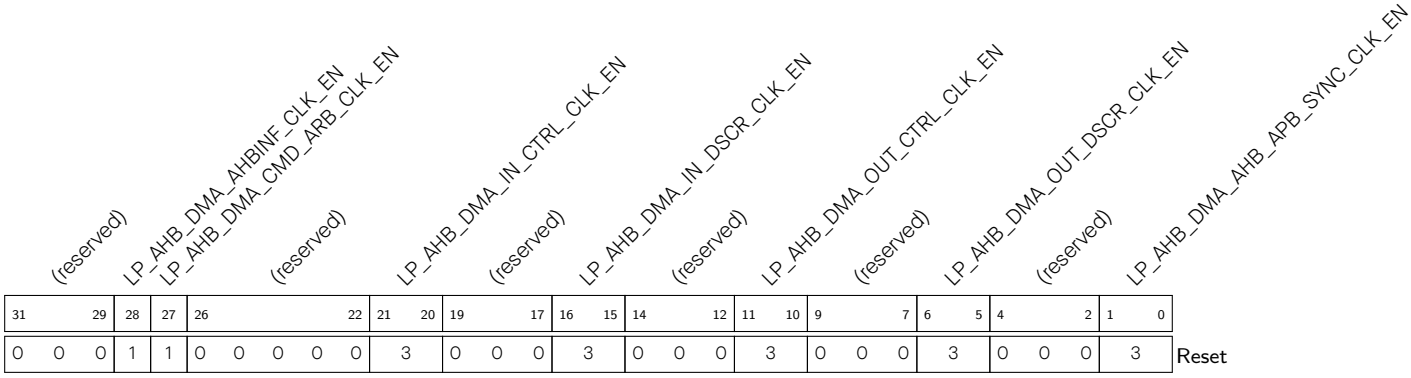
LP_AHB_DMA_ARB_TIMEOUT Configures arbitration time slot.
Measurement unit: bus clock cycle. (R/W)

Register 3.80. LP_AHB_DMA_WEIGHT_EN_REG (0x0610)



LP_AHB_DMA_WEIGHT_EN Configures whether to enable weight arbitration.
0: Disable
1: Enable
(R/W)

Register 3.81. LP_AHB_DMA_MODULE_CLK_EN_REG (0x0618)



LP_AHB_DMA_AHB_APB_SYNC_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA register configuration synchronization module.

Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

LP_AHB_DMA_OUT_DSCR_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA TX descriptor processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

LP_AHB_DMA_OUT_CTRL_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA TX data processing module.

Bit 0 corresponds to TX channel 0, bit 1 corresponds to TX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

LP_AHB_DMA_IN_DSCR_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA RX descriptor processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

LP_AHB_DMA_IN_CTRL_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA RX data processing module.

Bit 0 corresponds to RX channel 0, bit 1 corresponds to RX channel 1, and so on.

0: Not forcibly enable

1: Forcibly enable

(R/W)

Continued on the next page...

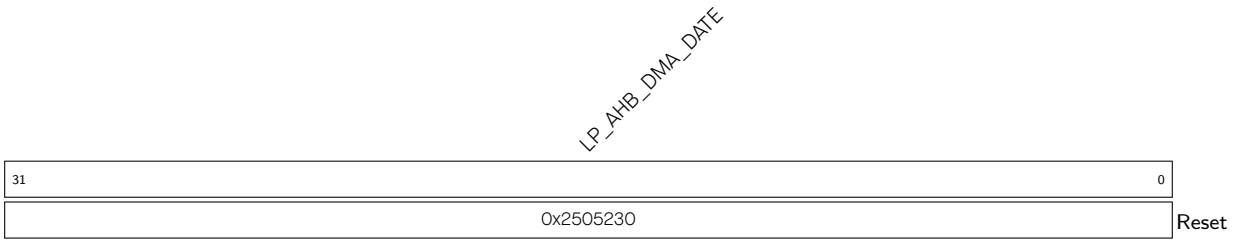
Register 3.81. LP_AHB_DMA_MODULE_CLK_EN_REG (0x0618)

Continued from the previous page...

LP_AHB_DMA_CMD_ARB_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA arbitration module.
Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.
0: Not forcibly enable
1: Forcibly enable
(R/W)

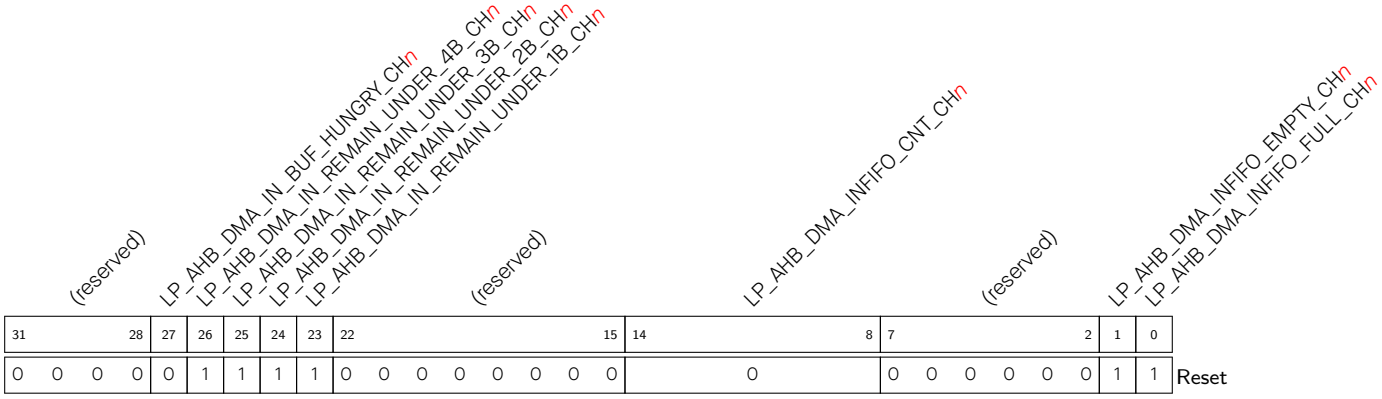
LP_AHB_DMA_AHBINF_CLK_EN Configures whether to forcibly enable the clock for the LP AHB DMA bus interface processing module.
Bit 0 corresponds to TX and RX channel 0, bit 1 corresponds to channel TX and RX 1, and so on.
0: Not forcibly enable
1: Forcibly enable
(R/W)

Register 3.82. LP_AHB_DMA_DATE_REG (0x00A8)



LP_AHB_DMA_DATE Version control register. (R/W)

Register 3.83. LP_AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-1) (0x0108+0x100n*)**



LP_AHB_DMA_INFIFO_FULL_CH*n* Represents whether RX FIFO is full for RX channel *n*.

0: Not full

1: Full

(RO)

LP_AHB_DMA_INFIFO_EMPTY_CH*n* Represents whether RX FIFO is empty for RX channel *n*.

0: Not empty

1: Empty

(RO)

LP_AHB_DMA_INFIFO_CNT_CH*n* Represents the number of data bytes in RX FIFO for RX channel *n*.

Measurement unit: byte. (RO)

LP_AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

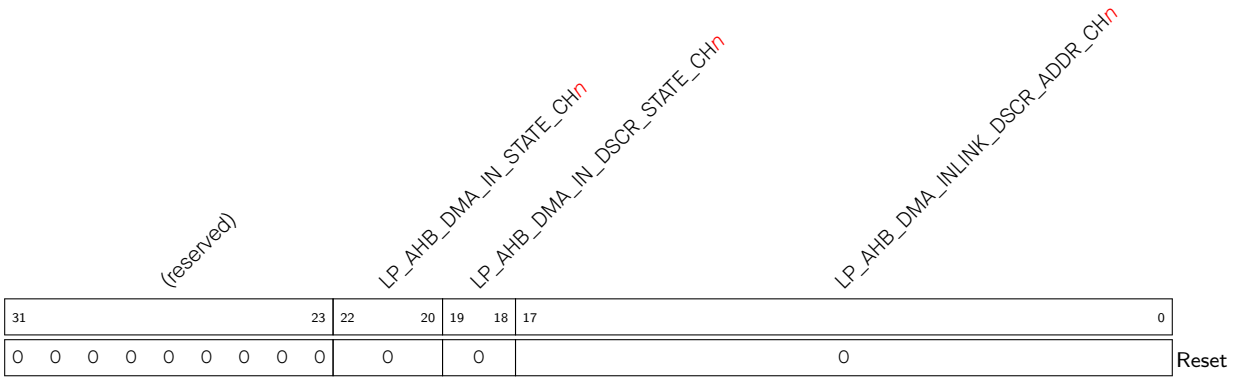
LP_AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

LP_AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

LP_AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

LP_AHB_DMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 3.84. LP_AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-1) (0x0118+0x100**n*)

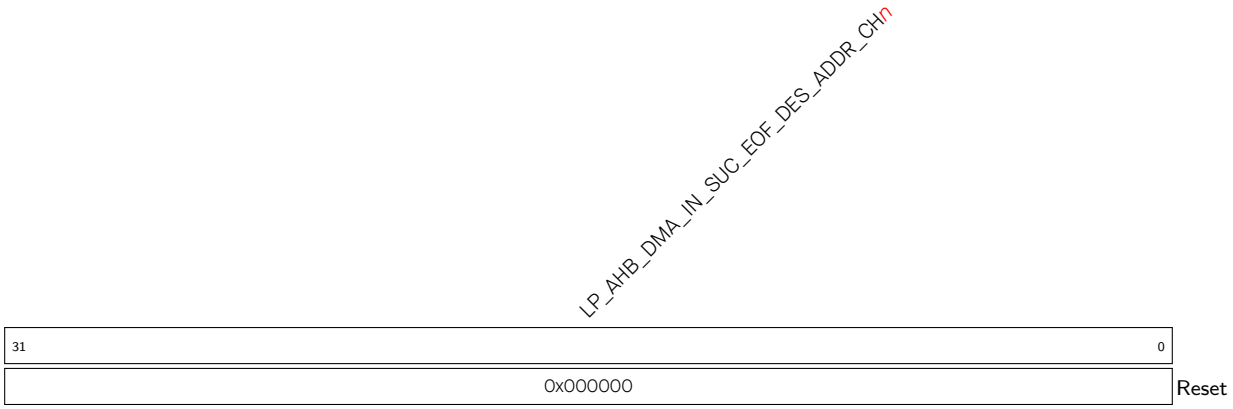


LP_AHB_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address to be processed. (RO)

LP_AHB_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

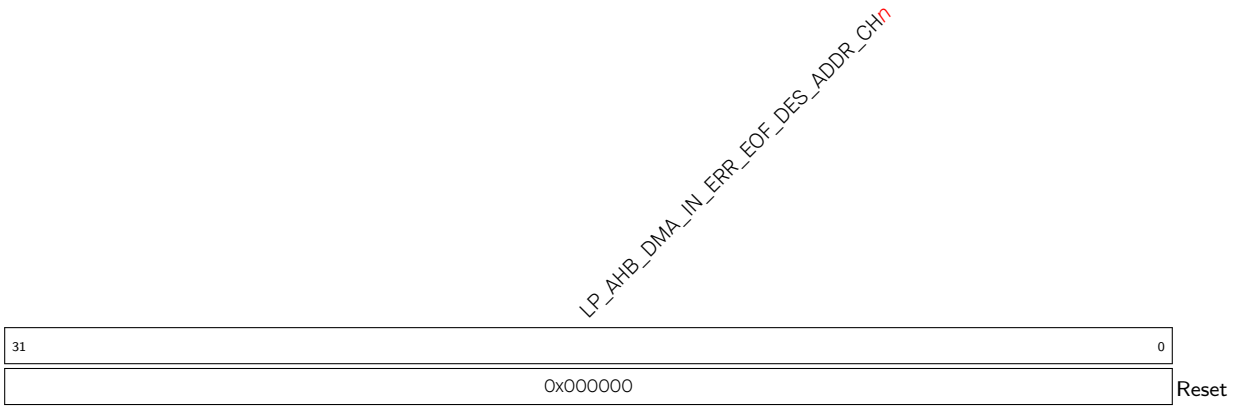
LP_AHB_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.85. LP_AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x011C+0x100**n*)



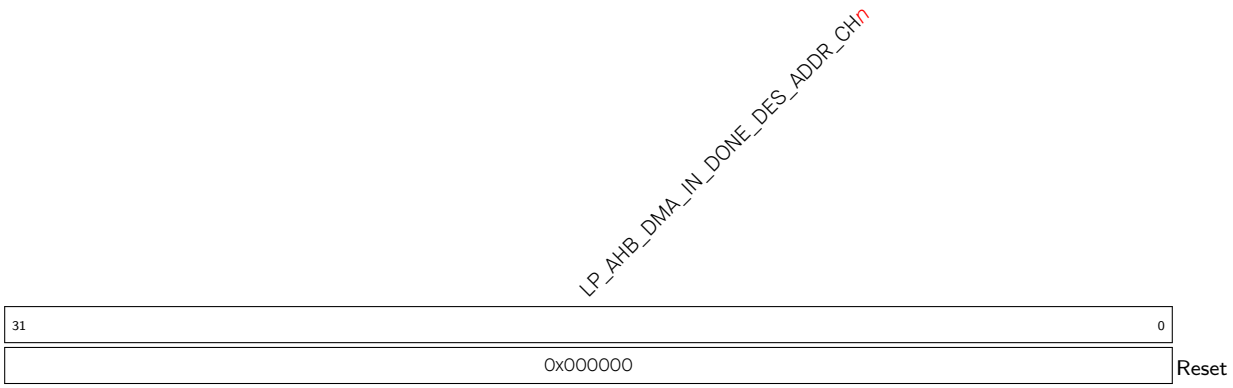
LP_AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when this descriptor has an EOF flag for RX channel *n*. (RO)

Register 3.86. LP_AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x0120+0x100*n*)



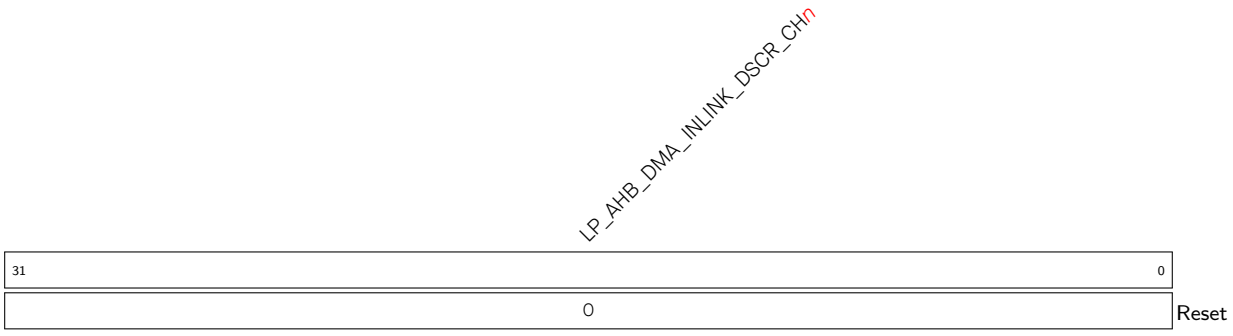
LP_AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when errors occur in the currently received data for RX channel *n*. (RO)

Register 3.87. LP_AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x0124+0x100*n*)



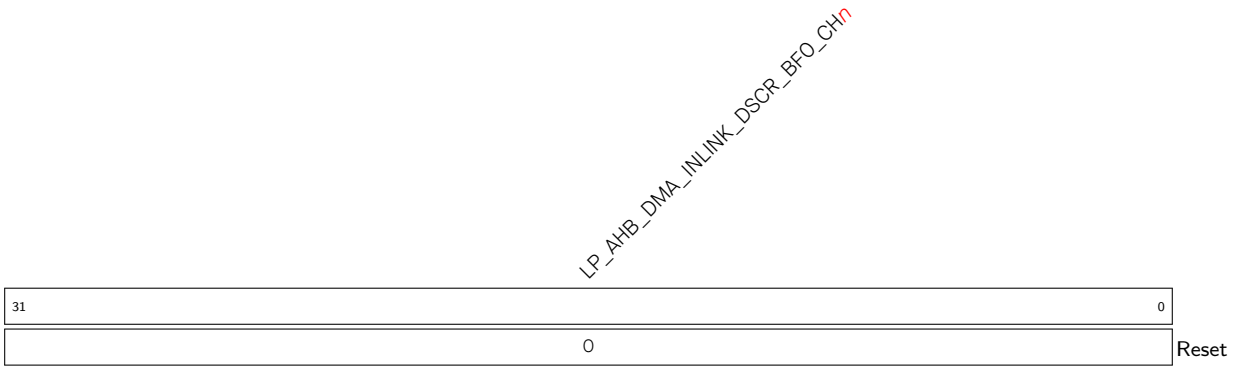
LP_AHB_DMA_IN_DONE_DES_ADDR_CH*n* Represents the descriptor address when this descriptor is completed for RX channel *n*. (RO)

Register 3.88. LP_AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-1) (0x0128+0x100*n*)



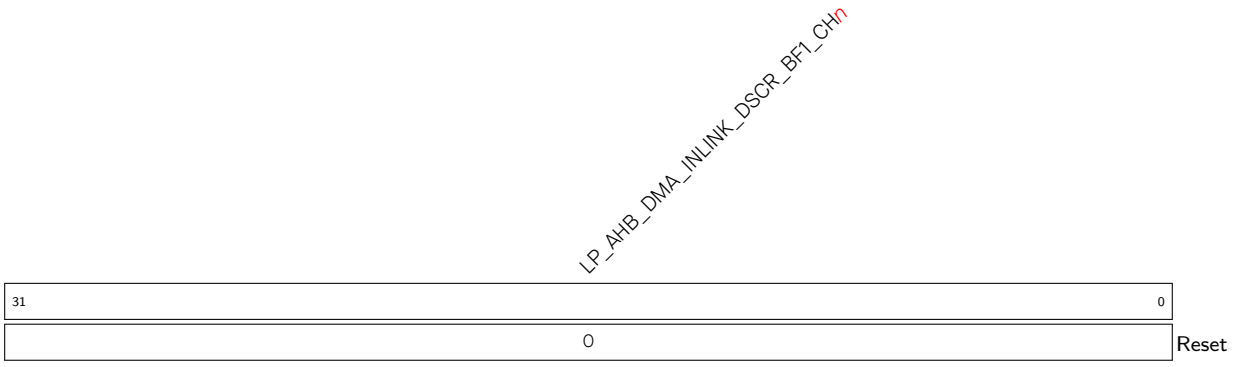
LP_AHB_DMA_INLINK_DSCR_CH*n* Represents the next receive descriptor address pointed to by the current receive descriptor that has already been fetched. (RO)

Register 3.89. LP_AHB_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-1) (0x012C+0x100**n*)



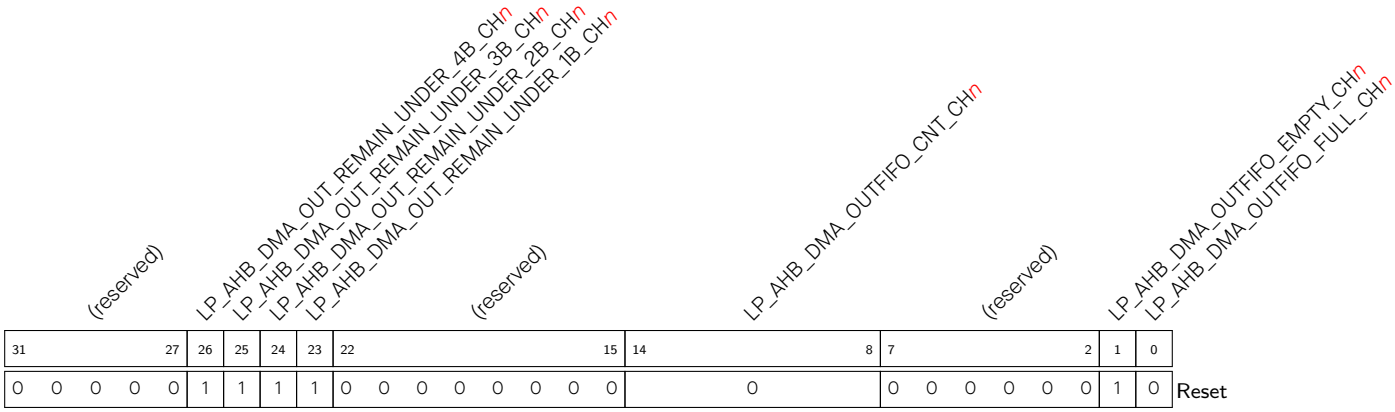
LP_AHB_DMA_INLINK_DSCR_BF0_CH*n* Represents the current receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.90. LP_AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-1) (0x0130+0x100**n*)



LP_AHB_DMA_INLINK_DSCR_BF1_CH*n* Represents the previous receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.91. LP_AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-1) (0x0188+0x100*n*)



LP_AHB_DMA_OUTFIFO_FULL_CH*n* Represents whether TX FIFO is full for TX channel *n*.

0: Not full

1: Full

(RO)

LP_AHB_DMA_OUTFIFO_EMPTY_CH*n* Represents whether TX FIFO is empty for TX channel *n*.

0: Not empty

1: Empty

(RO)

LP_AHB_DMA_OUTFIFO_CNT_CH*n* Represents the number of data bytes in TX FIFO for TX channel *n*.

Measurement unit: byte. (RO)

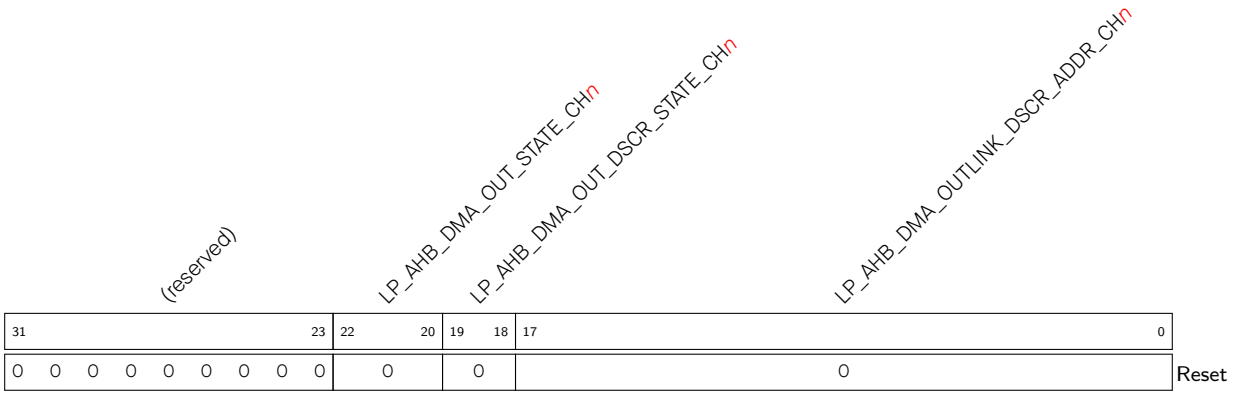
LP_AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

LP_AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

LP_AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

LP_AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

Register 3.92. LP_AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-1) (0x0198+0x100**n*)

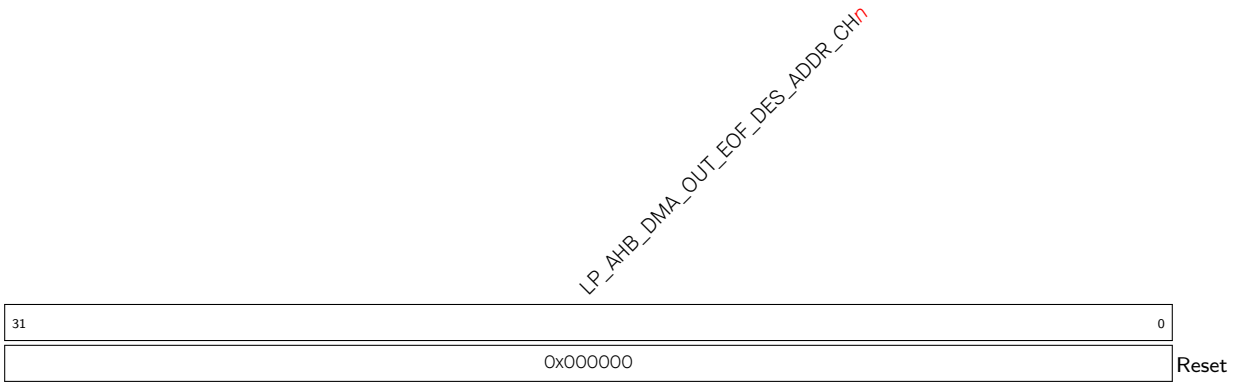


LP_AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address to be processed. (RO)

LP_AHB_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

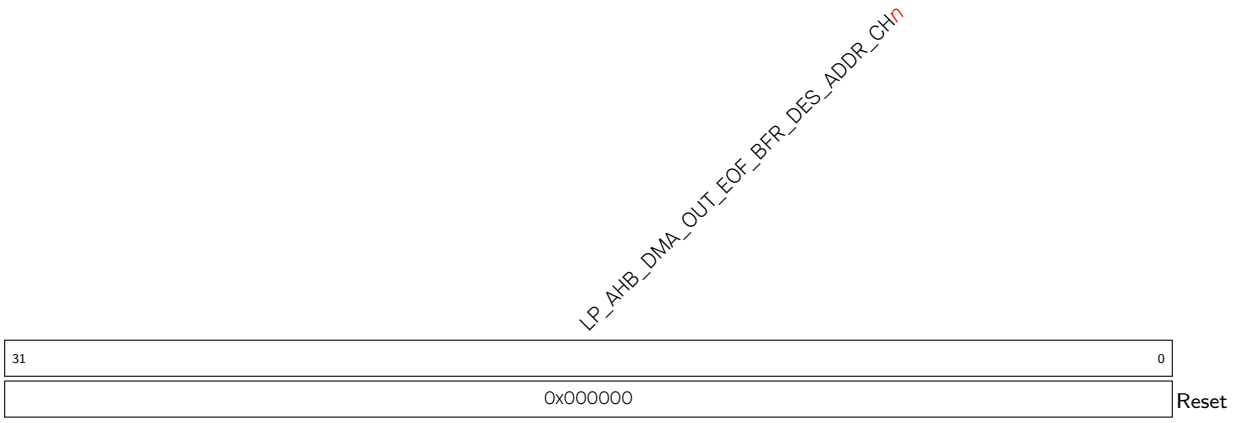
LP_AHB_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.93. LP_AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x019C+0x100**n*)



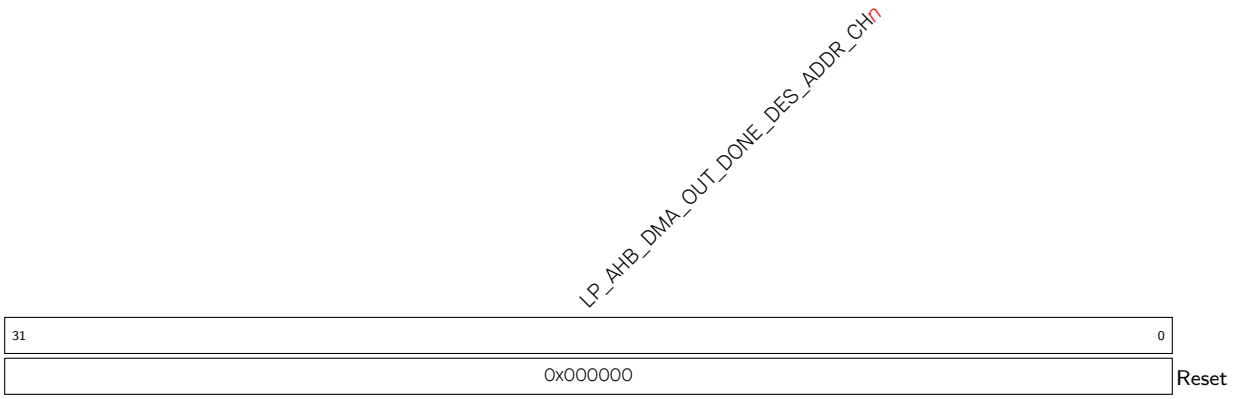
LP_AHB_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the transmit descriptor address when this descriptor has an EOF flag for TX channel *n*. (RO)

Register 3.94. LP_AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x01A0+0x100*n*)



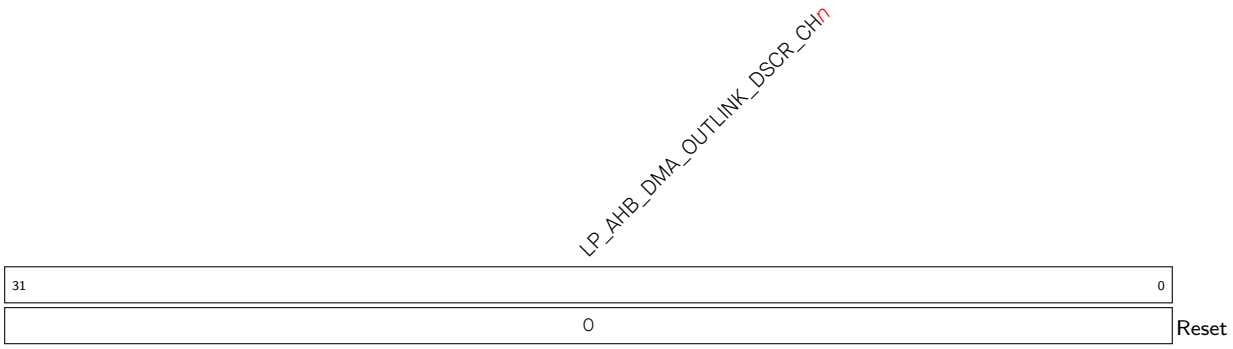
LP_AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the transmit descriptor address before the last transmit descriptor for TX channel *n*. (RO)

Register 3.95. LP_AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x01A4+0x100*n*)



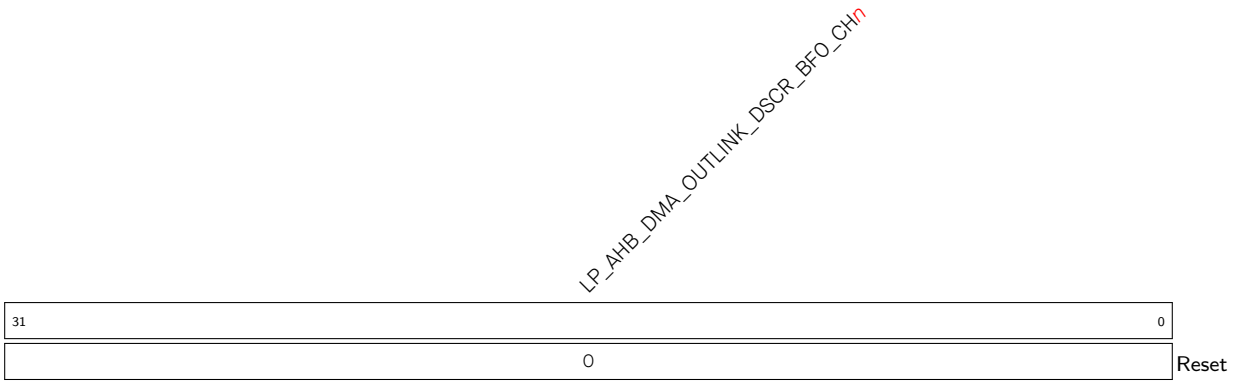
LP_AHB_DMA_OUT_DONE_DES_ADDR_CH*n* Represents the descriptor address when this descriptor is completed for TX channel *n*. (RO)

Register 3.96. LP_AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-1) (0x01A8+0x100**n*)



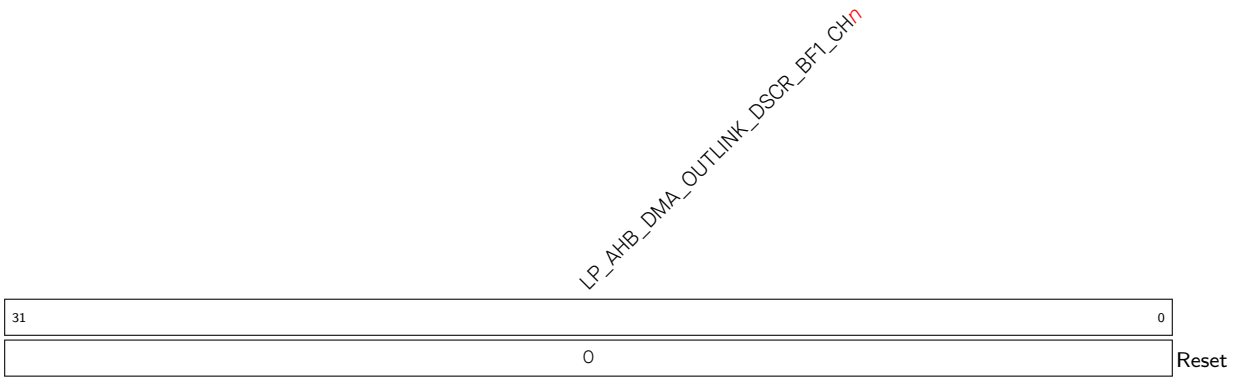
LP_AHB_DMA_OUTLINK_DSCR_CH*n* Represents the next transmit descriptor address pointed to by the current transmit descriptor that has already been fetched. (RO)

Register 3.97. LP_AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-1) (0x01AC+0x100**n*)



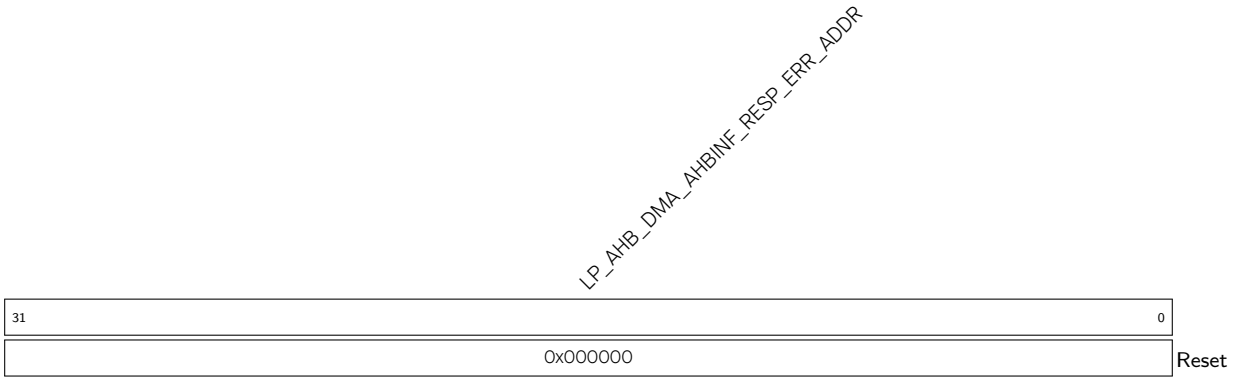
LP_AHB_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the current transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.98. LP_AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-1) (0x01B0+0x100**n*)



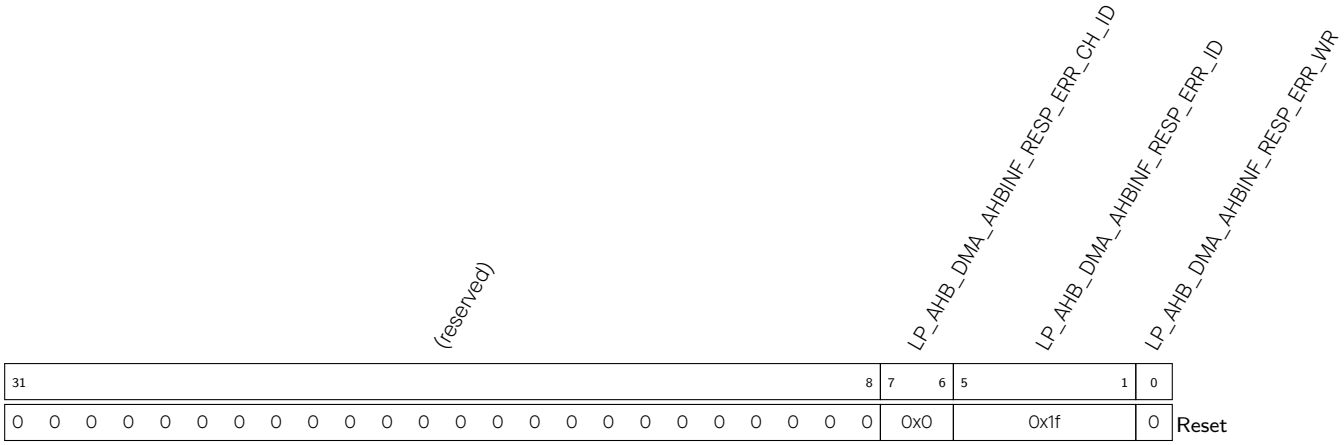
LP_AHB_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the previous transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.99. LP_AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG (0x0620)



LP_AHB_DMA_AHBINF_RESP_ERR_ADDR Represents the address of the current LP AHB bus transfer that triggers an error response. (RO)

Register 3.100. LP_AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG (0x0624)



LP_AHB_DMA_AHBINF_RESP_ERR_WR Represents whether the LP AHB response error is caused by a write request.

0: Read request

1: Write request

(RO)

LP_AHB_DMA_AHBINF_RESP_ERR_ID Represents the LP AHB response error request ID.

0: ADC

1: DAC

16: GDMA-AHB-LP channel 0 memory to memory transfer

17: GDMA-AHB-LP channel 1 memory to memory transfer

Others: Invalid

(RO)

LP_AHB_DMA_AHBINF_RESP_ERR_CH_ID Represents the LP AHB response error request channel ID.

Bit[1]: Represents the channel direction

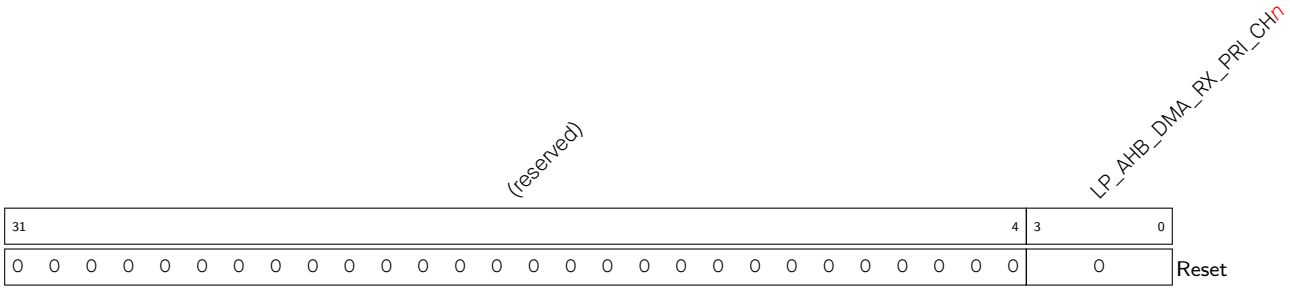
0: RX channel

1: TX channel

Bit[0]: Represents the channel number. Value range: 0 ~1

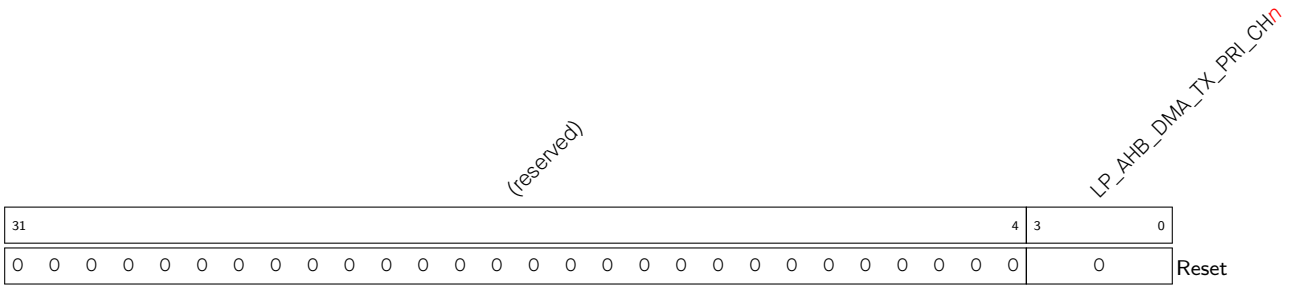
(RO)

Register 3.101. LP_AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-1) (0x0134+0x100**n*)



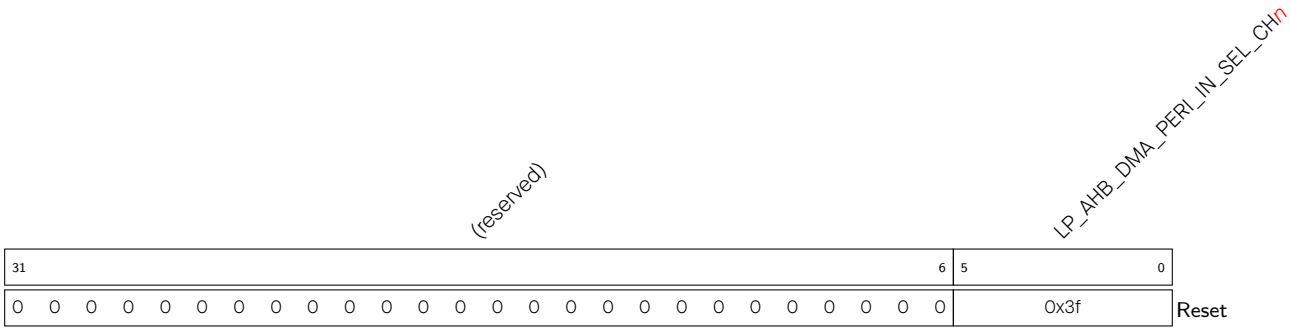
LP_AHB_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

Register 3.102. LP_AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-1) (0x01B4+0x100**n*)



LP_AHB_DMA_TX_PRI_CH*n* Configures the priority of TX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

Register 3.103. LP_AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-1) (0x0138+0x100**n*)



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

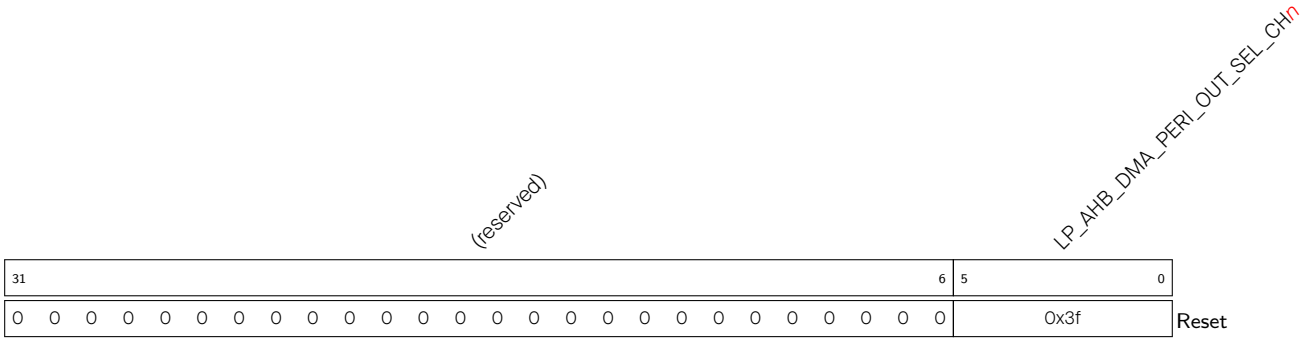
0: ADC

1: DAC

2~63: Dummy

(R/W)

Register 3.104. LP_AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-1) (0x01B8+0x100n*)**



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

0: ADC

1: DAC

2~63: Dummy

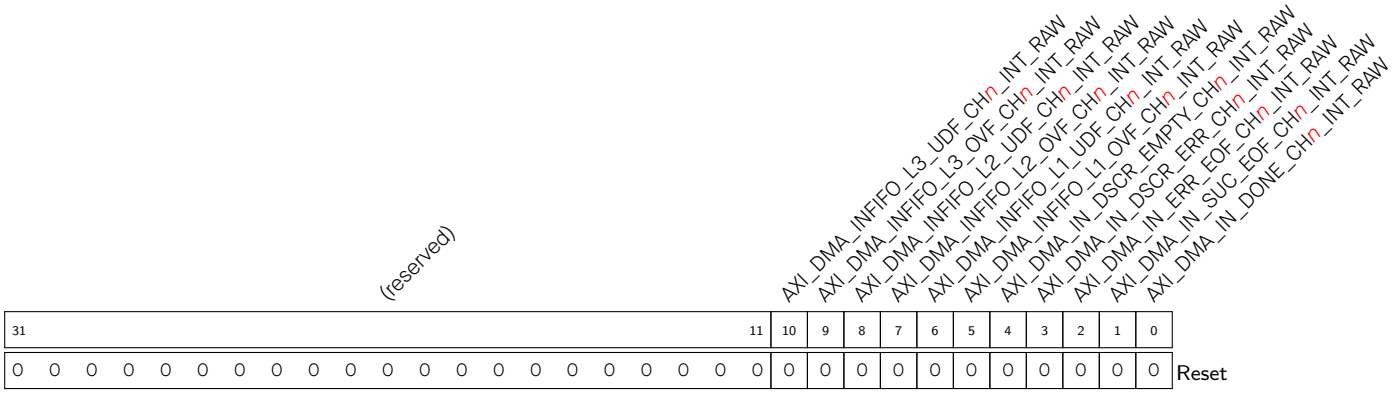
(R/W)

3.9.3 GDMA-AXI Registers

The addresses in this section are relative to GDMA-AXI base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 3.105. AXI_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x68n*)**



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

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

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

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

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

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

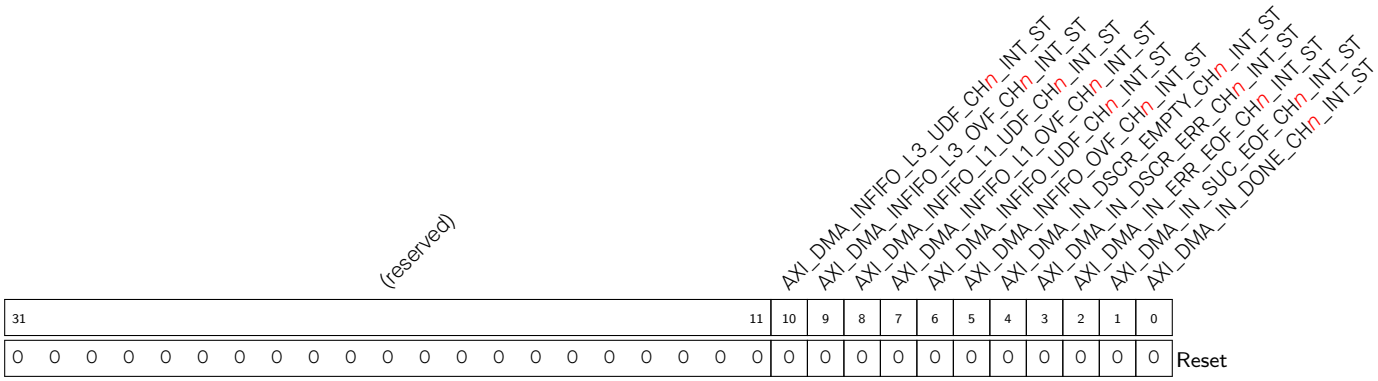
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

Register 3.106. AXI_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x68*n*)**



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

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

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

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

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

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

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

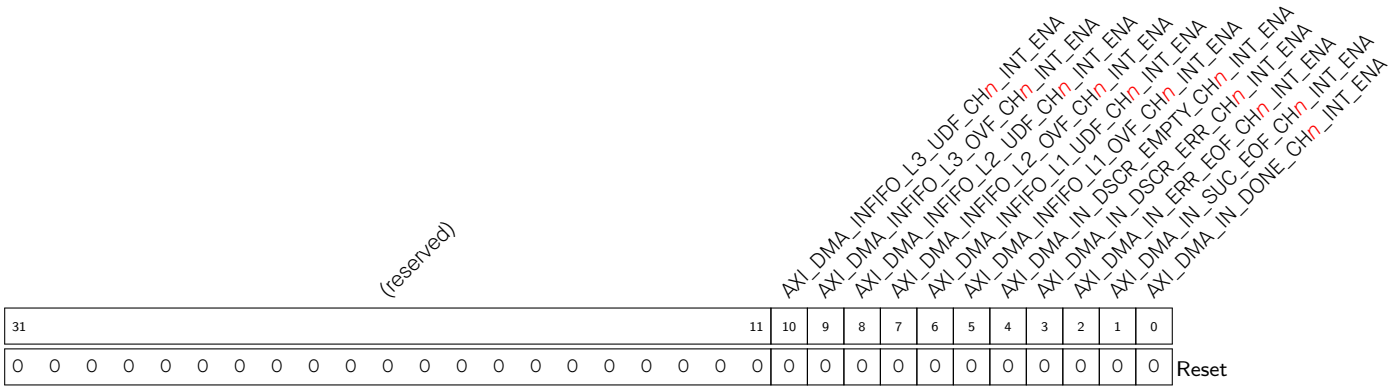
AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (RO)

Register 3.107. AXI_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x68**n*)



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

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

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

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

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

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (R/W)

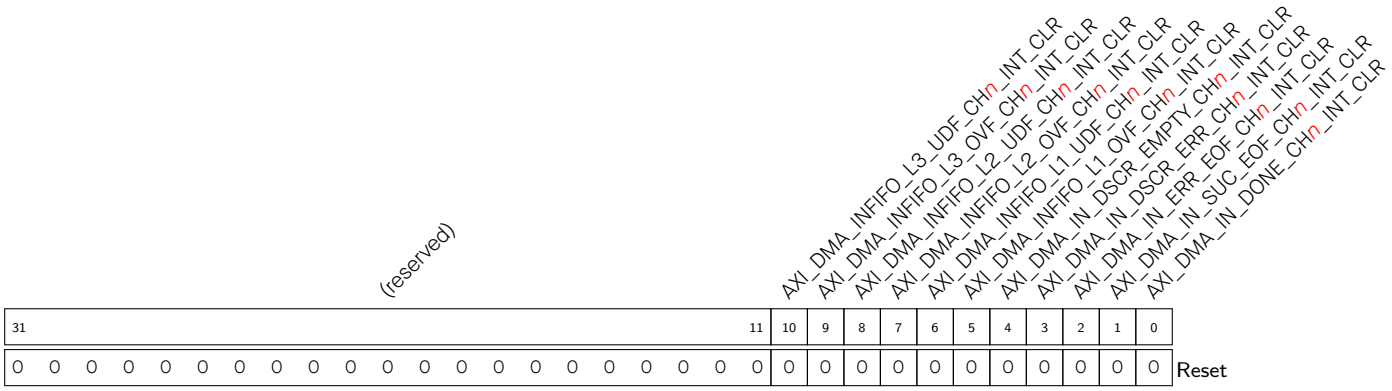
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (R/W)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (R/W)

Register 3.108. AXI_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x68n*)**



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

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

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

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

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

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (WT)

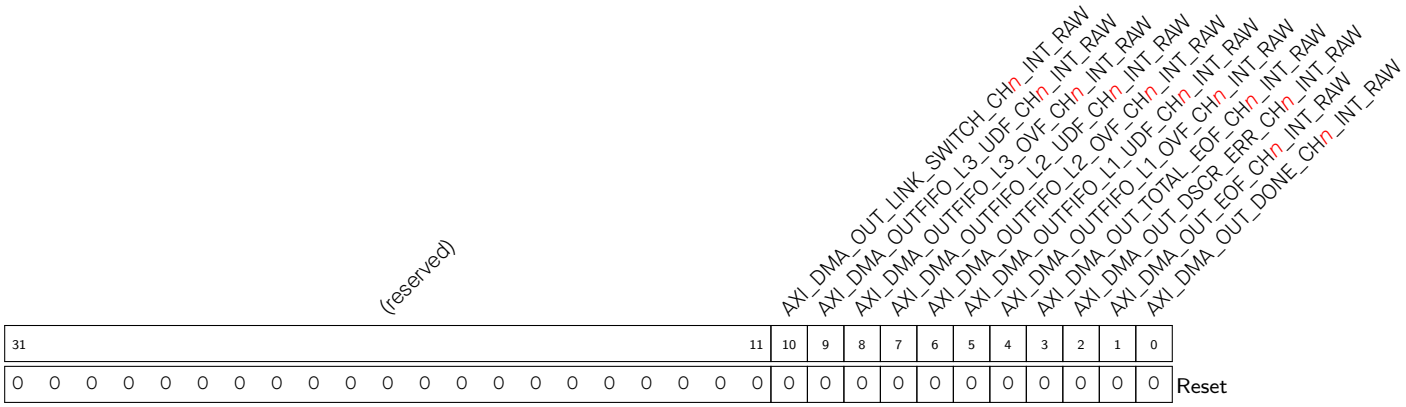
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (WT)

Register 3.109. AXI_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0138+0x68*n*)**



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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

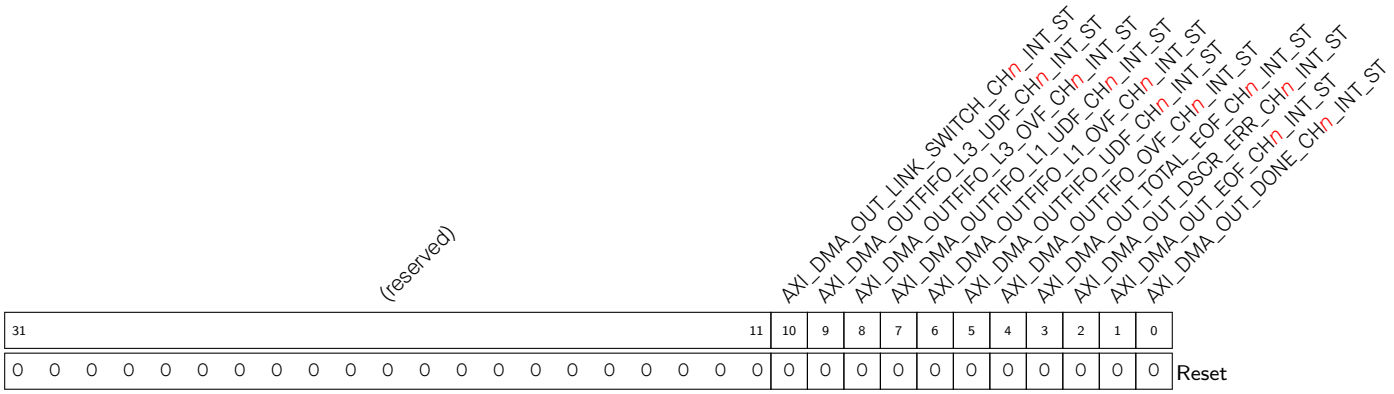
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

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

Register 3.110. AXI_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x013C+0x68n*)**



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

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

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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (RO)

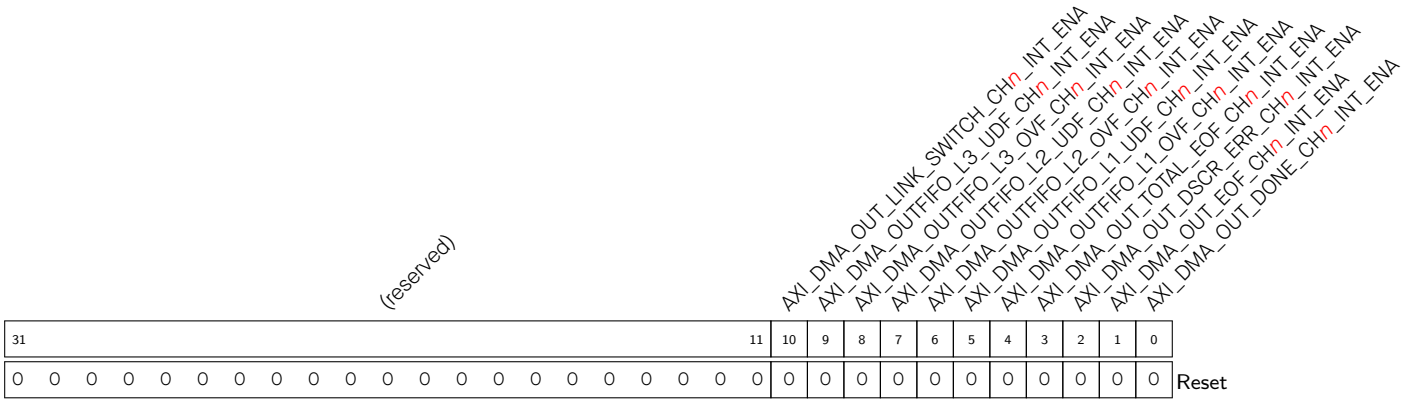
AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (RO)

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

Register 3.111. AXI_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0140+0x68**n*)



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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/W)

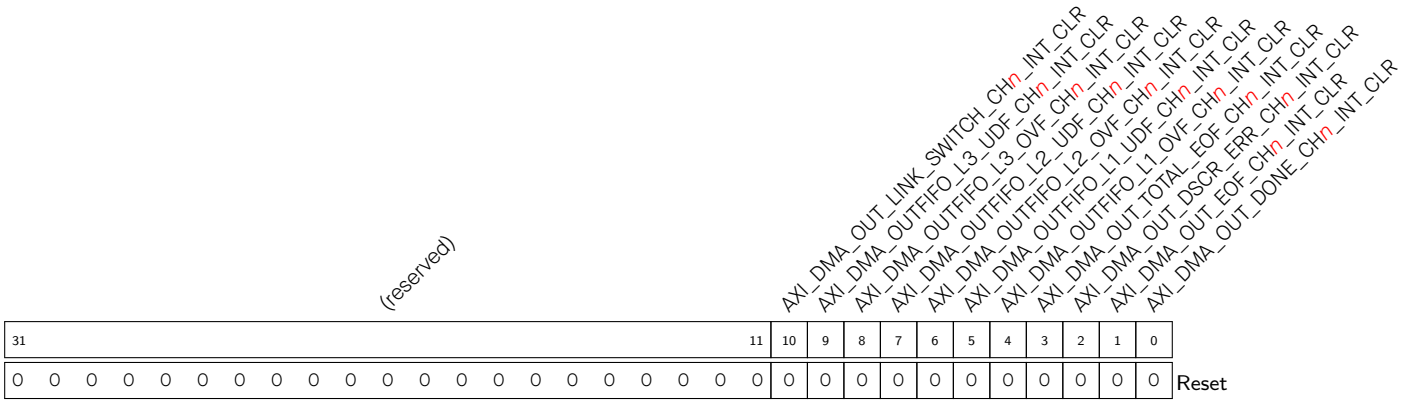
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (R/W)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (R/W)

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

Register 3.112. AXI_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x0144+0x68**n*)



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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (WT)

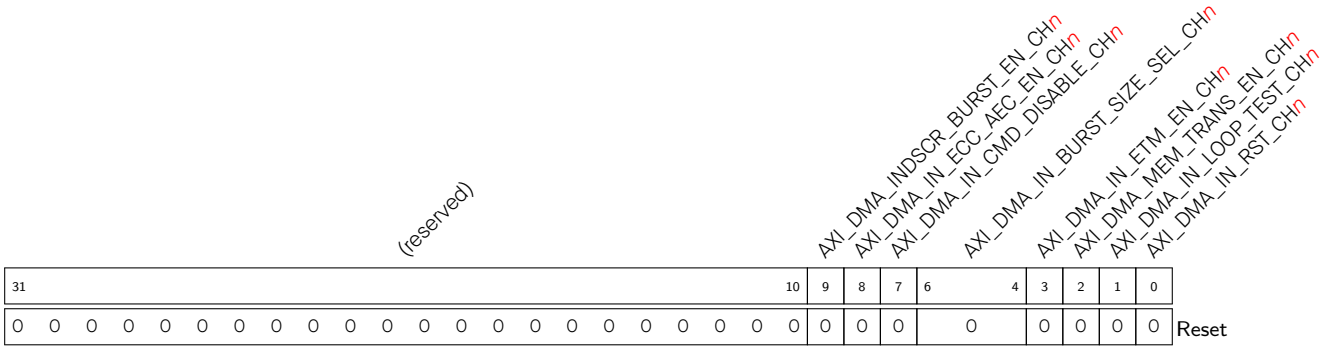
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (WT)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (WT)

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

Register 3.113. AXI_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0010+0x68**n*)



AXI_DMA_IN_RST_CH*n* Configures reset state of AXI DMA RX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

AXI_DMA_IN_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for RX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

AXI_DMA_MEM_TRANS_EN_CH*n* Configures whether to enable memory-to-memory data transfer.

0: Disable

1: Enable

(R/W)

AXI_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel *n*.

0: Disable

1: Enable

(R/W)

AXI_DMA_IN_BURST_SIZE_SEL_CH*n* Configures burst length for RX channel *n*.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid

(R/W)

AXI_DMA_IN_CMD_DISABLE_CH*n* Configures whether to disable commands for RX channel *n*.

0: Enable commands

1: Disable commands

(R/W)

Continued on the next page...

Register 3.113. AXI_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0010+0x68**n*)

Continued from the previous page...

AXI_DMA_IN_ECC_AEC_EN_CH*n* Configures whether to access the ECC or AES domain for RX channel *n*.

0: Disable

1: Enable

(R/W)

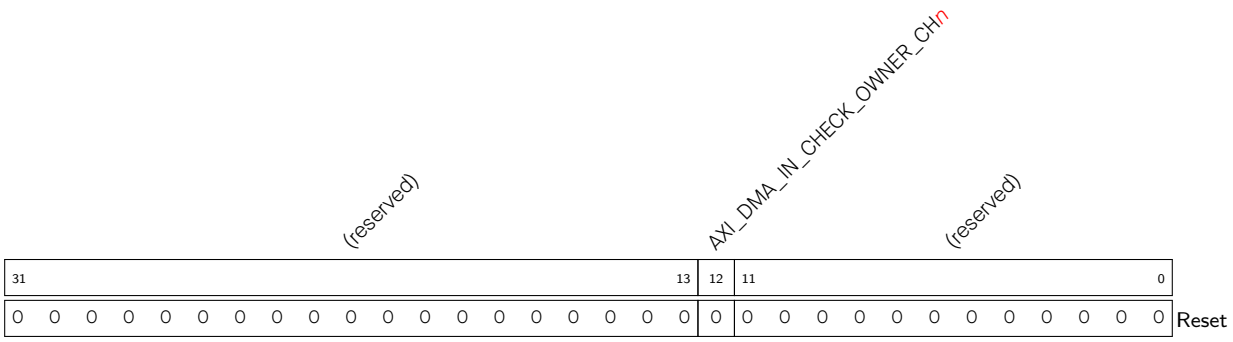
AXI_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read receive descriptors.

0: Disable

1: Enable

(R/W)

Register 3.114. AXI_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0014+0x68**n*)



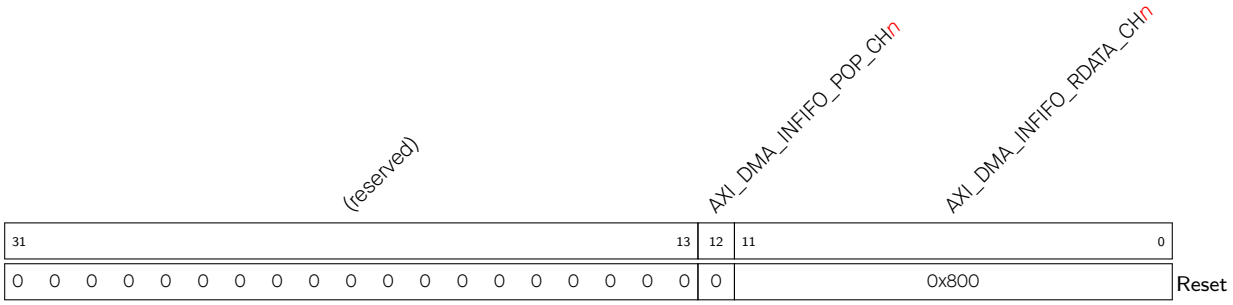
AXI_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.115. AXI_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x001C+0x68n*)**



AXI_DMA_INFIFO_RDATA_CH*n* Data received from AXI DMA FIFO. (RO)

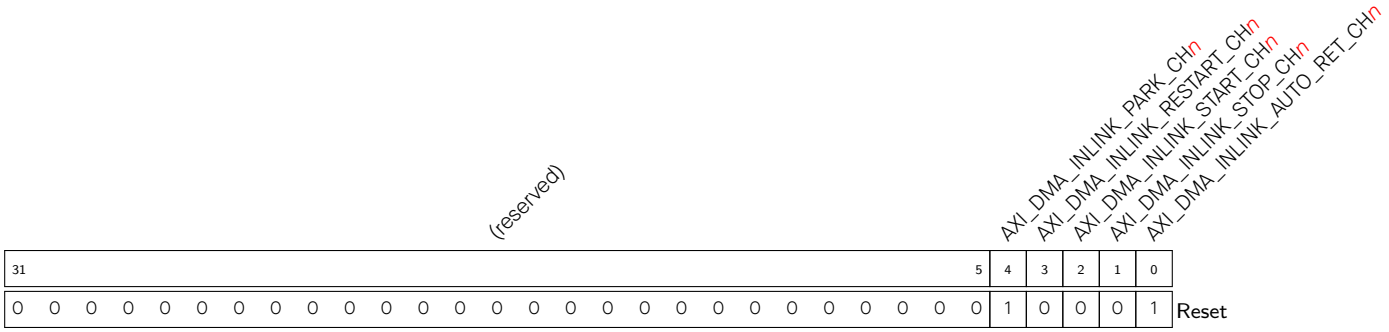
AXI_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AXI DMA FIFO for RX channel *n*.

0: Invalid. No effect

1: Pop

(WT)

Register 3.116. AXI_DMA_IN_LINK1_CH*n*_REG (*n*: 0-2) (0x0020+0x68n*)**



AXI_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor address when an error occurs in the current received data for RX channel *n*.

0: Not return

1: Return

(R/W)

AXI_DMA_INLINK_STOP_CH*n* Configures whether to stop processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_INLINK_START_CH*n* Configures whether to start processing descriptors for RX channel *n*.

0: Invalid. No effect

1: Start

(WT)

AXI_DMA_INLINK_RESTART_CH*n* Configures whether to restart descriptor processing for RX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

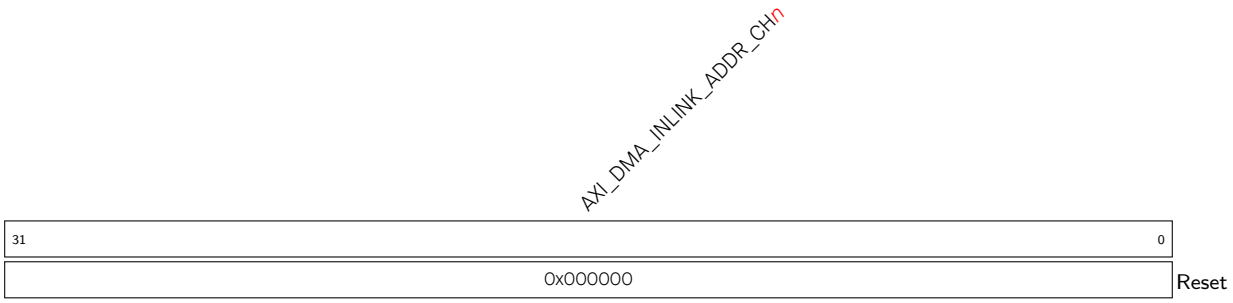
AXI_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor FSM for RX channel *n*.

0: Running

1: Idle

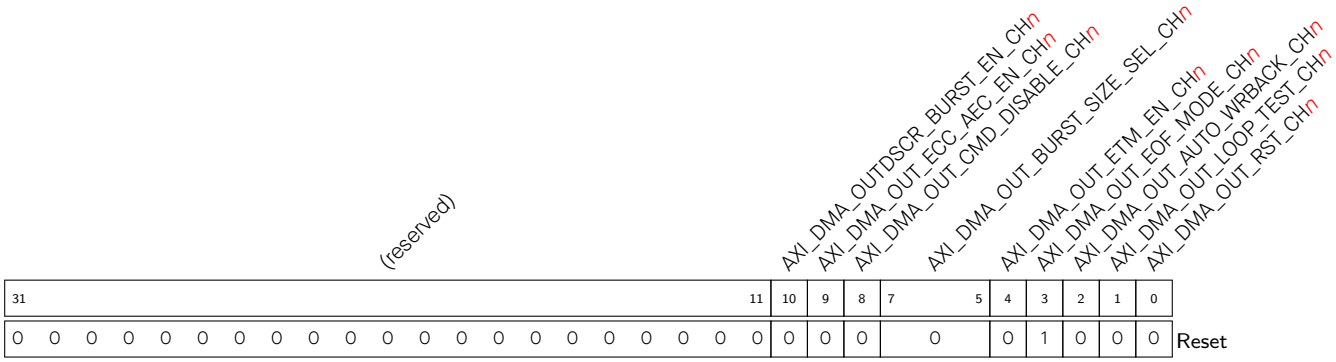
(RO)

Register 3.117. AXI_DMA_IN_LINK2_CH*n*_REG (*n*: 0-2) (0x0024+0x68n*)**



AXI_DMA_INLINK_ADDR_CH*n* Configures the first receive descriptor address for RX channel *n*. (R/W)

Register 3.118. AXI_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x0148+0x68*n*)**



AXI_DMA_OUT_RST_CH*n* Configures reset state of AXI DMA TX channel *n* FSM and FIFO pointer.

0: Release reset

1: Reset

(R/W)

AXI_DMA_OUT_LOOP_TEST_CH*n* Configures whether to disable automatic clearing of owner bit for TX channel *n*.

0: Invalid. No effect

1: Disable

(R/W)

AXI_DMA_OUT_AUTO_WRBACK_CH*n* Configures whether to enable automatic outlink write-back when all data in the TX FIFO has been transmitted for TX channel *n*.

0: Disable

1: Enable

(R/W)

AXI_DMA_OUT_EOF_MODE_CH*n* Configures when to generate the EOF flag for TX channel *n*.

0: Generate when data to be transmitted has been pushed into the FIFO.

1: Generate when data to be transmitted has been popped from the FIFO.

(R/W)

AXI_DMA_OUT_ETM_EN_CH*n* Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

AXI_DMA_OUT_BURST_SIZE_SEL_CH*n* Configures burst transfer size for TX channel *n*.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid

(R/W)

Continued on the next page...

Register 3.118. AXI_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x0148+0x68**n*)

Continued from the previous page...

AXI_DMA_OUT_CMD_DISABLE_CH*n* Configures whether to disable commands for TX channel *n*.

0: Enable commands

1: Disable commands

(R/W)

AXI_DMA_OUT_ECC_AEC_EN_CH*n* Configures whether to access the ECC or AES domain for TX channel *n*.

0: Disable

1: Enable

(R/W)

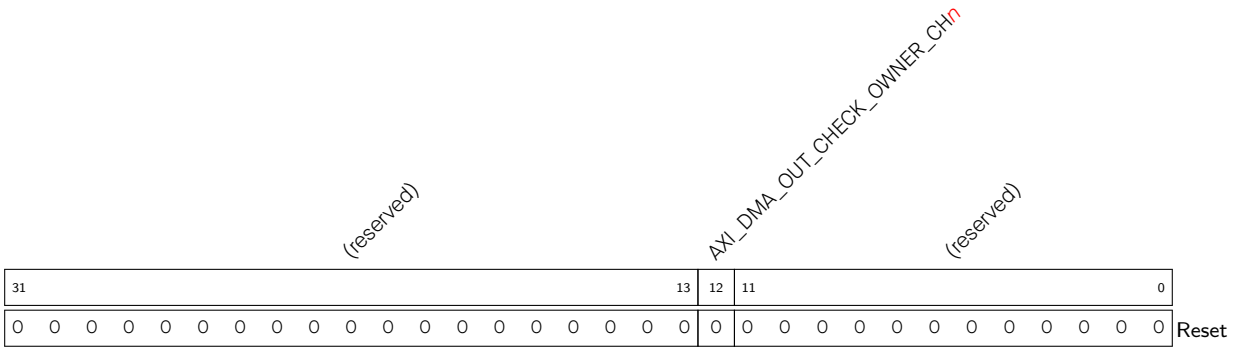
AXI_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* to read transmit descriptors.

0: Disable

1: Enable

(R/W)

Register 3.119. AXI_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x014C+0x68**n*)



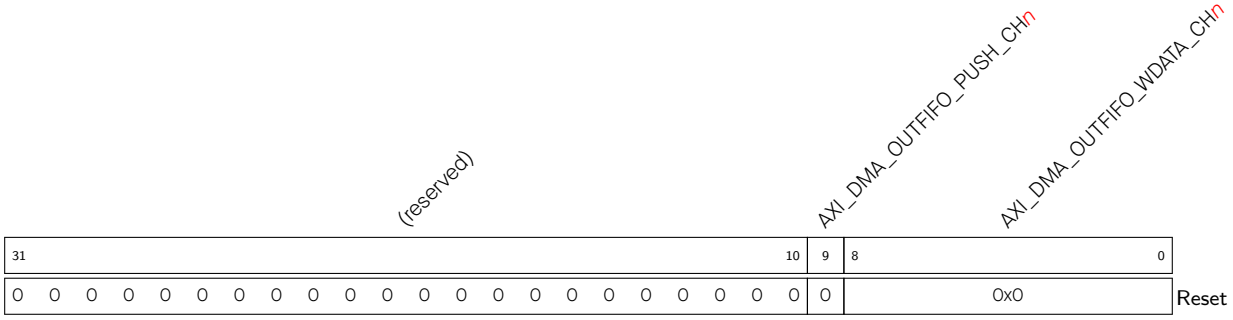
AXI_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.120. AXI_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x0154+0x68n*)**



AXI_DMA_OUTFIFO_WDATA_CH*n* Data to be transmitted to AXI DMA FIFO. (R/W)

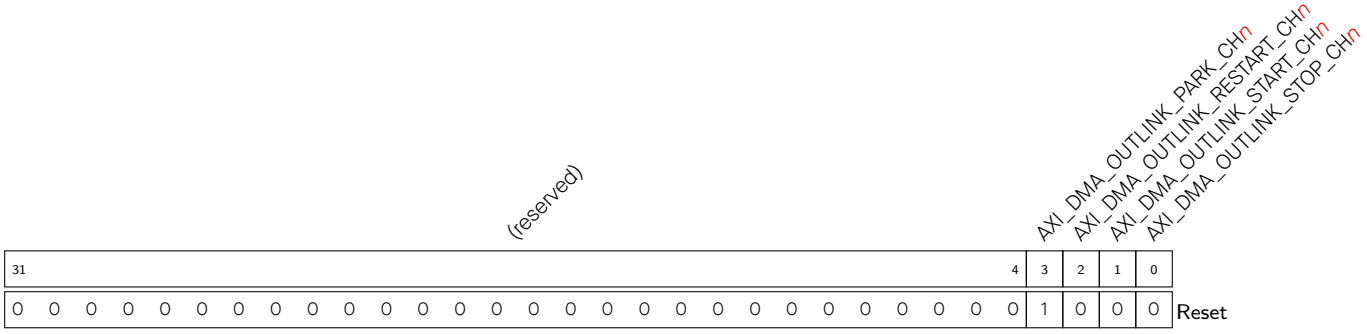
AXI_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AXI DMA FIFO for TX channel *n*.

0: Invalid. No effect

1: Push

(WT)

Register 3.121. AXI_DMA_OUT_LINK1_CH*n*_REG (*n*: 0-2) (0x0158+0x68*n*)**



AXI_DMA_OUTLINK_STOP_CH*n* Configures whether to stop processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_OUTLINK_START_CH*n* Configures whether to start processing descriptors for TX channel *n*.

0: Invalid. No effect

1: Start

(WT)

AXI_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart descriptor processing for TX channel *n*.

0: Invalid. No effect

1: Restart

(WT)

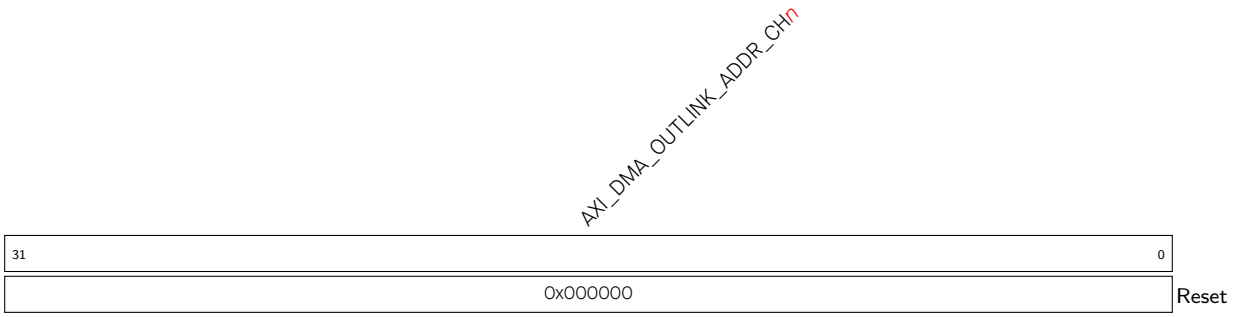
AXI_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor FSM for TX channel *n*.

0: Running

1: Idle

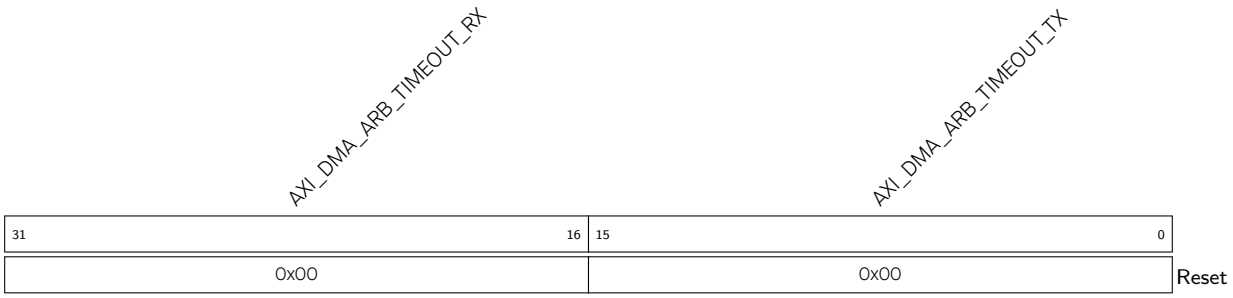
(RO)

Register 3.122. AXI_DMA_OUT_LINK2_CH*n*_REG (*n*: 0-2) (0x015C+0x68n*)**



AXI_DMA_OUTLINK_ADDR_CH*n* Configures the first transmit descriptor address for TX channel *n*. (R/W)

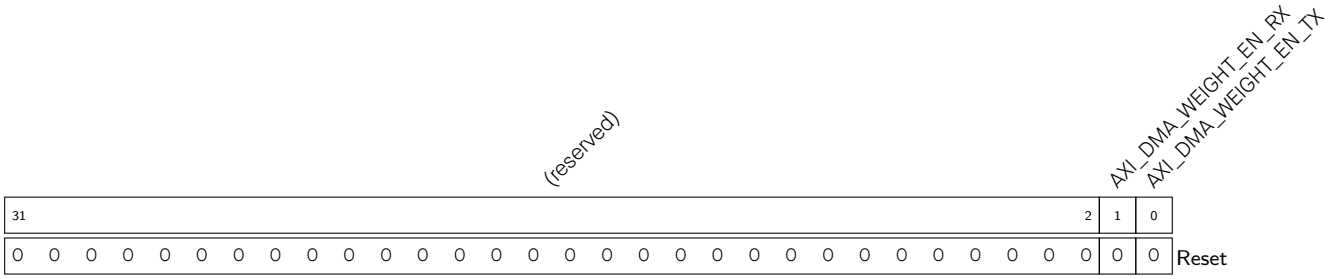
Register 3.123. AXI_DMA_ARB_TIMEOUT_REG (0x0270)



AXI_DMA_ARB_TIMEOUT_TX Configures arbitration time slot for TX. Measurement unit: bus clock cycle. (R/W)

AXI_DMA_ARB_TIMEOUT_RX Configures arbitration time slot for RX. Measurement unit: bus clock cycle. (R/W)

Register 3.124. AXI_DMA_WEIGHT_EN_REG (0x0274)



AXI_DMA_WEIGHT_EN_TX Configures whether to enable weight arbitration for TX channels.

0: Disable

1: Enable

(R/W)

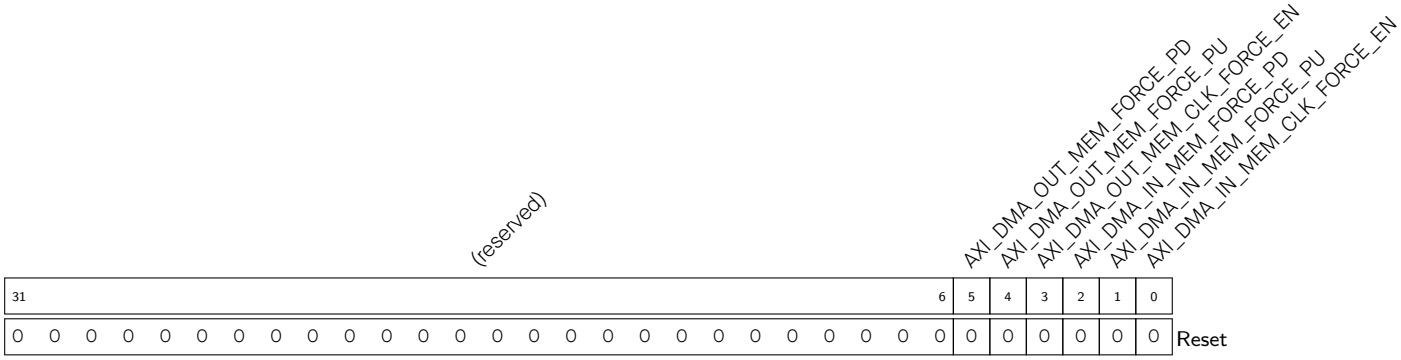
AXI_DMA_WEIGHT_EN_RX Configures whether to enable weight arbitration for RX channels.

0: Disable

1: Enable

(R/W)

Register 3.125. AXI_DMA_IN_MEM_CONF_REG (0x0278)



AXI_DMA_IN_MEM_CLK_FORCE_EN Configures whether to force on the RX RAM clock.

0: Not force on

1: Force on

(R/W)

AXI_DMA_IN_MEM_FORCE_PU Configures whether to force power up RX RAM.

0: Not force power up

1: Force power up

(R/W)

AXI_DMA_IN_MEM_FORCE_PD Configures whether to force power down RX RAM.

0: Not force power down

1: Force power down

(R/W)

AXI_DMA_OUT_MEM_CLK_FORCE_EN Configures whether to force on the TX RAM clock.

0: Not force on

1: Force on

(R/W)

AXI_DMA_OUT_MEM_FORCE_PU Configures whether to force power up TX RAM.

0: Not force power up

1: Force power up

(R/W)

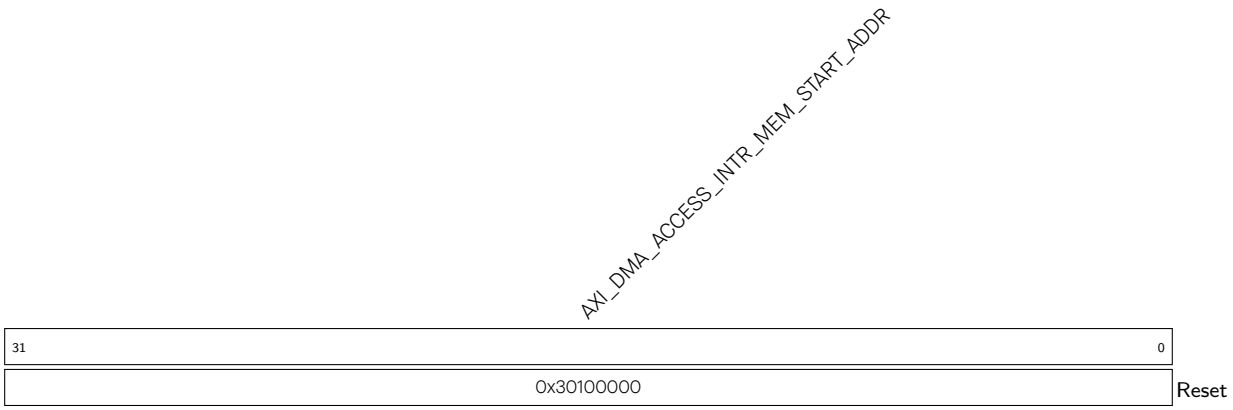
AXI_DMA_OUT_MEM_FORCE_PD Configures whether to force power down TX RAM.

0: Not force power down

1: Force power down

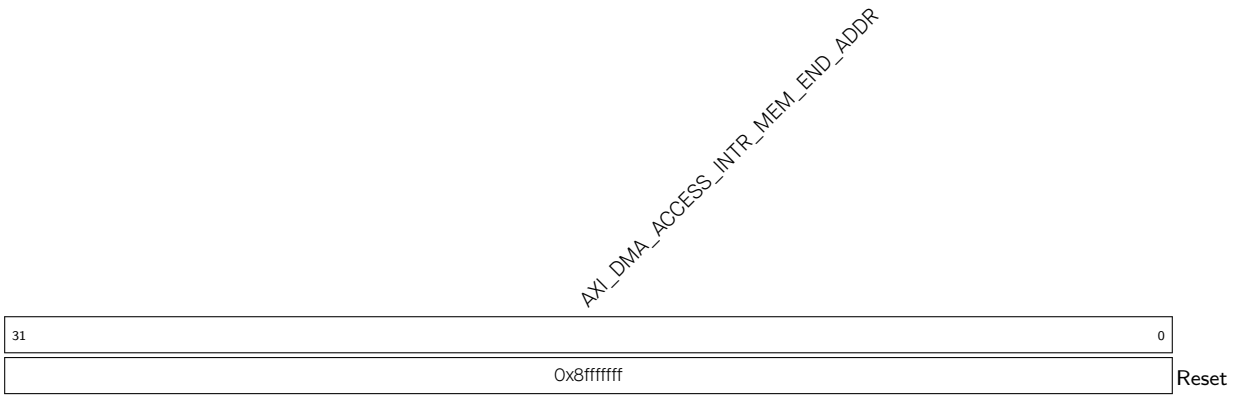
(R/W)

Register 3.126. AXI_DMA_INTR_MEM_START_ADDR_REG (0x027C)



AXI_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible address space for internal memory. (R/W)

Register 3.127. AXI_DMA_INTR_MEM_END_ADDR_REG (0x0280)



AXI_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible address space for internal memory. (R/W)

Register 3.128. AXI_DMA_EXTR_MEM_START_ADDR_REG (0x0284)



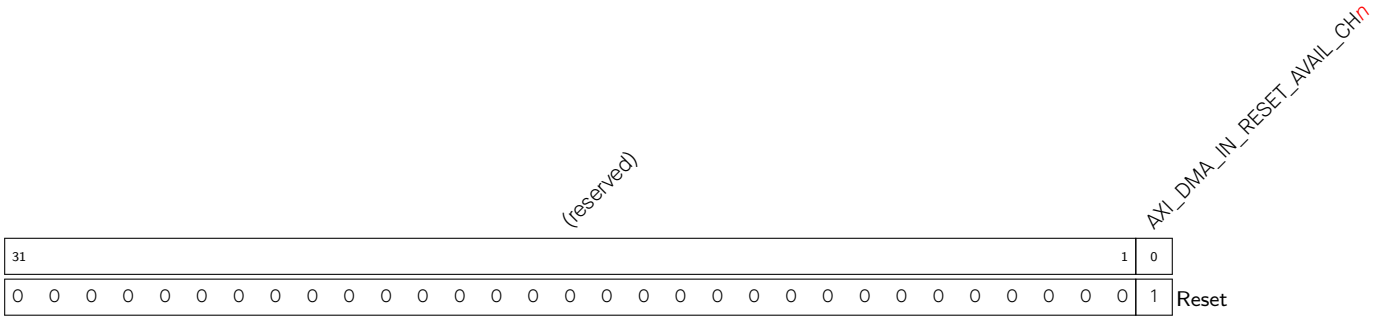
AXI_DMA_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible address space for external memory. (R/W)

Register 3.129. AXI_DMA_EXTR_MEM_END_ADDR_REG (0x0288)



AXI_DMA_ACCESS_EXTR_MEM_END_ADDR Configures the end address of the accessible address space for external memory. (R/W)

Register 3.130. AXI_DMA_IN_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x028C+0x4n*)**



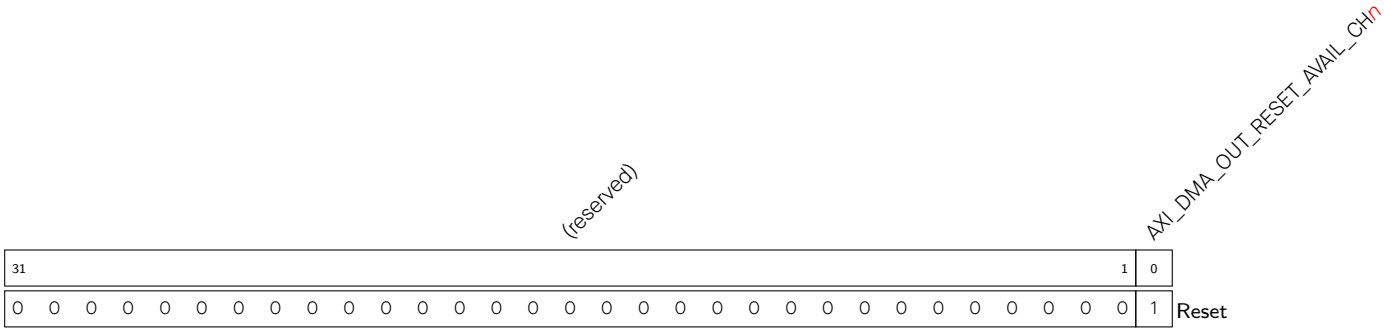
AXI_DMA_IN_RESET_AVAIL_CH*n* Represents whether reset is valid for RX channel *n*.

0: Invalid

1: Valid

(RO)

Register 3.131. AXI_DMA_OUT_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x0298+0x4n*)**



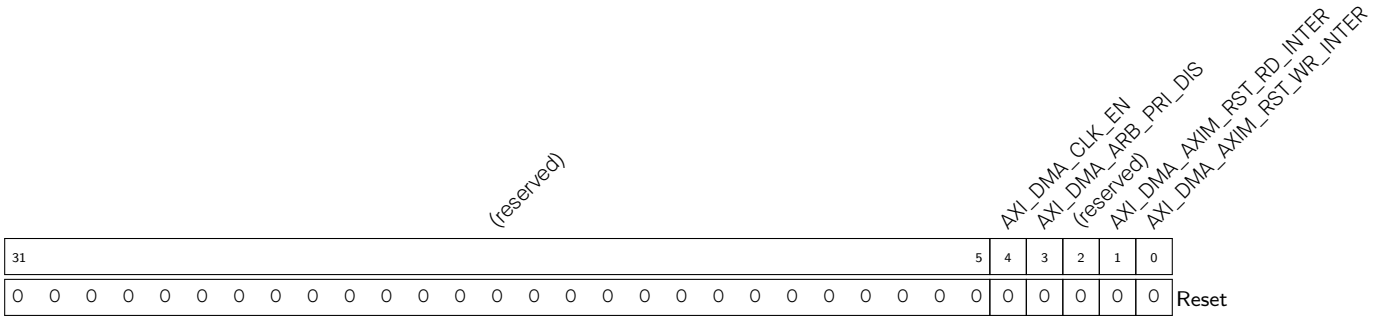
AXI_DMA_OUT_RESET_AVAIL_CH*n* Represents whether reset is valid for TX channel *n*.

0: Invalid

1: Valid

(RO)

Register 3.132. AXI_DMA_MISC_CONF_REG (0x02A8)



AXI_DMA_AXIM_RST_WR_INTER Configures reset state of the internal AXI write FSM.

0: Release reset

1: Reset

(R/W)

AXI_DMA_AXIM_RST_RD_INTER Configures reset state of the internal AXI read FSM.

0: Release reset

1: Reset

(R/W)

AXI_DMA_ARB_PRI_DIS Configures whether to disable fixed-priority channel arbitration.

0: Enable

1: Disable

(R/W)

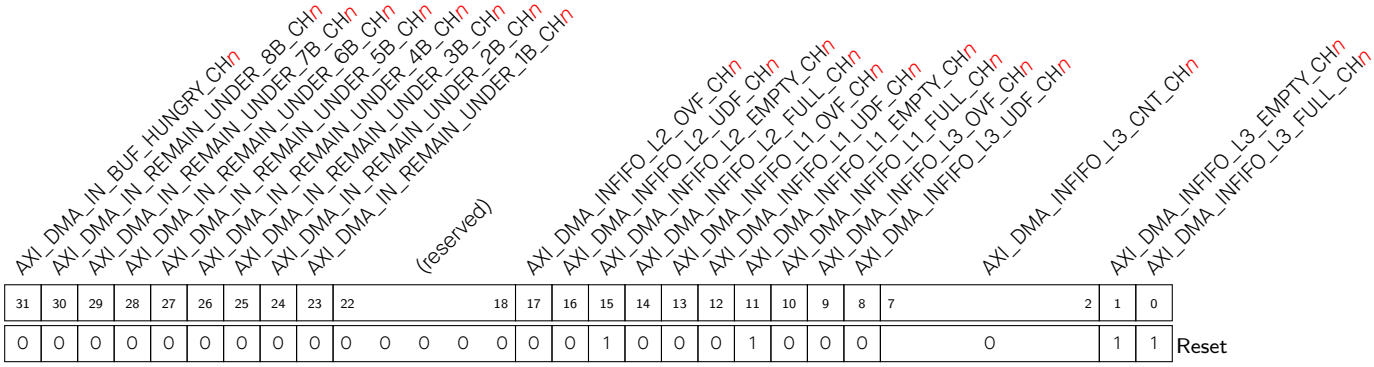
AXI_DMA_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 3.133. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68*n*)**



AXI_DMA_INFIFO_L3_FULL_CH*n* Represents whether L3 RX FIFO is full for RX channel *n*.

0: Not full

1: Full

(RO)

AXI_DMA_INFIFO_L3_EMPTY_CH*n* Represents whether L3 RX FIFO is empty for RX channel *n*.

0: Not empty

1: Empty

(RO)

AXI_DMA_INFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 RX FIFO for RX channel *n*.

Measurement unit: byte. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n* Represents whether L3 RX FIFO underflows for RX channel *n*.

0: Not underflow

1: Underflow

(RO)

AXI_DMA_INFIFO_L3_OVF_CH*n* Represents whether L3 RX FIFO overflows for RX channel *n*.

0: Not overflow

1: Overflow

(RO)

AXI_DMA_INFIFO_L1_FULL_CH*n* Represents whether L1 RX FIFO is full for RX channel *n*.

0: Not full

1: Full

(RO)

AXI_DMA_INFIFO_L1_EMPTY_CH*n* Represents whether L1 RX FIFO is empty for RX channel *n*.

0: Not empty

1: Empty

(RO)

Continued on the next page...

Register 3.133. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68**n*)

Continued from the previous page...

AXI_DMA_INFIFO_L1_UDF_CH*n* Represents whether L1 RX FIFO underflows for RX channel *n*.
0: Not underflow
1: Underflow
(RO)

AXI_DMA_INFIFO_L1_OVF_CH*n* Represents whether L1 RX FIFO overflows for RX channel *n*.
0: Not overflow
1: Overflow
(RO)

AXI_DMA_INFIFO_L2_FULL_CH*n* Represents whether L2 RX FIFO is full for RX channel *n*.
0: Not full
1: Full
(RO)

AXI_DMA_INFIFO_L2_EMPTY_CH*n* Represents whether L2 RX FIFO is empty for RX channel *n*.
0: Not empty
1: Empty
(RO)

AXI_DMA_INFIFO_L2_UDF_CH*n* Represents whether L2 RX FIFO underflows for RX channel *n*.
0: Not underflow
1: Underflow
(RO)

AXI_DMA_INFIFO_L2_OVF_CH*n* Represents whether L2 RX FIFO overflows for RX channel *n*.
0: Not overflow
1: Overflow
(RO)

AXI_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

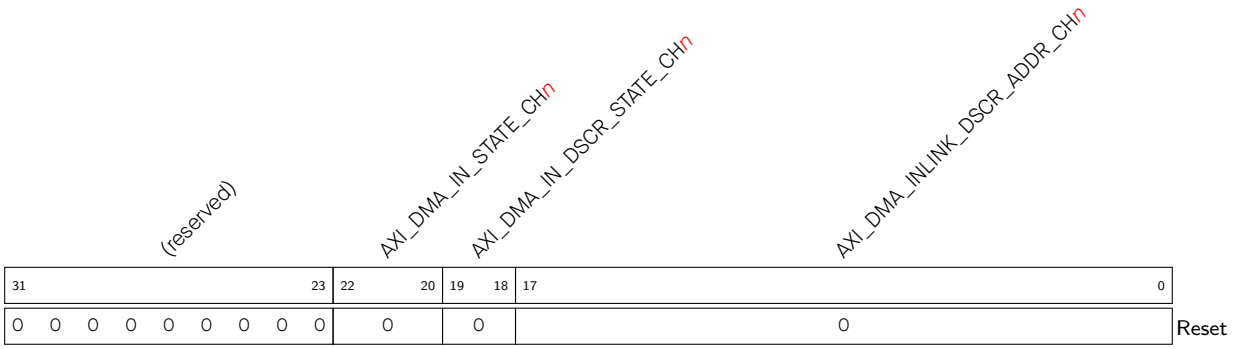
AXI_DMA_IN_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

AXI_DMA_IN_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

AXI_DMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 3.134. AXI_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0028+0x68**n*)

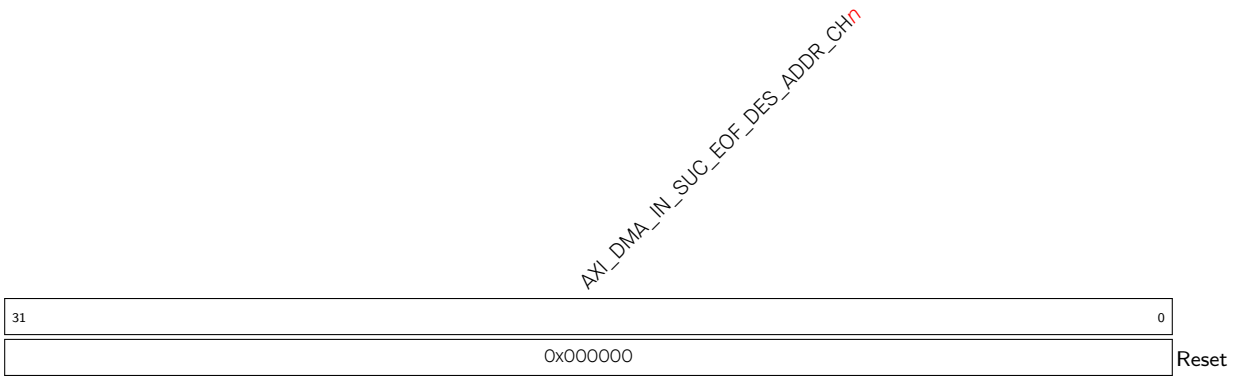


AXI_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address to be processed. (RO)

AXI_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

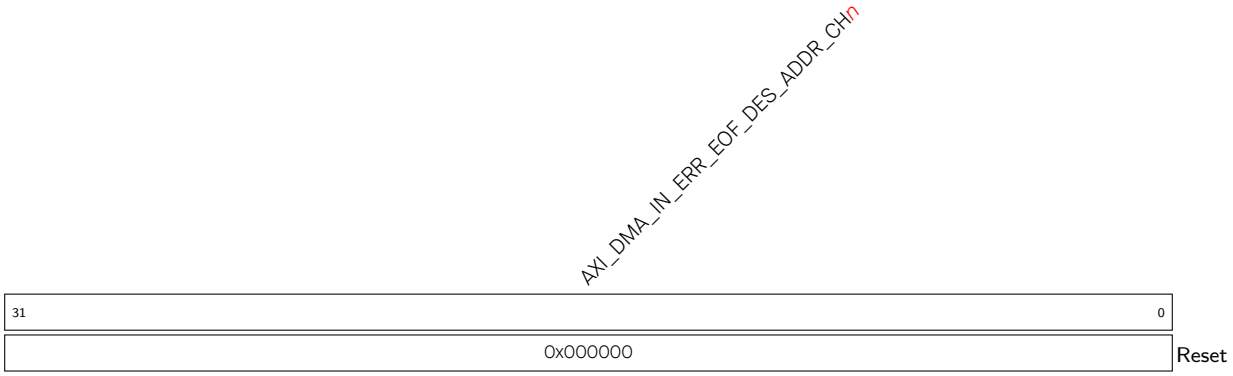
AXI_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.135. AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x002C+0x68**n*)



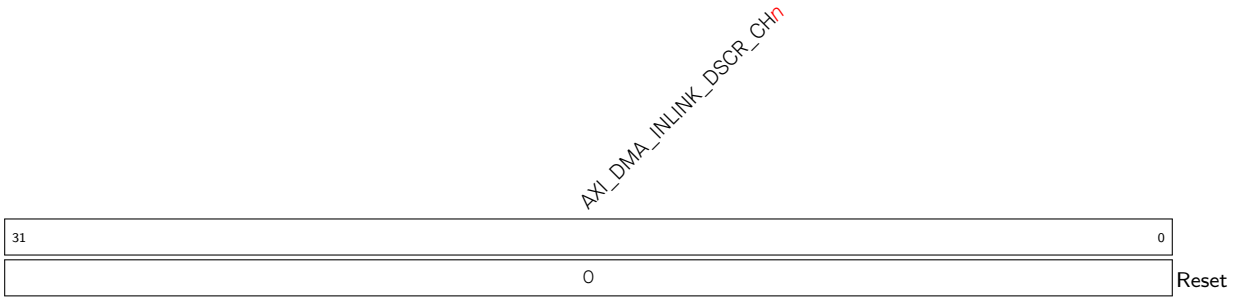
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when this descriptor has an EOF flag for RX channel *n*. (RO)

Register 3.136. AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0030+0x68n*)**



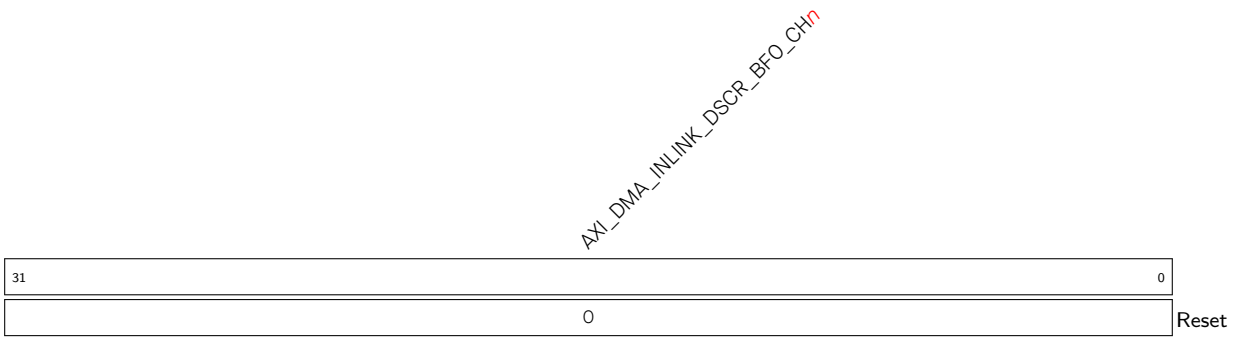
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the receive descriptor address when errors occur in the currently received data for RX channel *n*. (RO)

Register 3.137. AXI_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0034+0x68n*)**



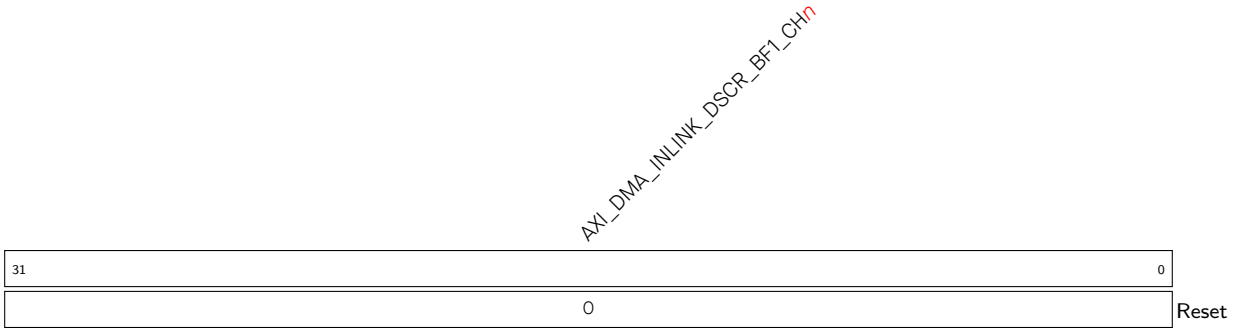
AXI_DMA_INLINK_DSCR_CH*n* Represents the next receive descriptor address pointed to by the current receive descriptor that has already been fetched. (RO)

Register 3.138. AXI_DMA_IN_DSCR_BFO_CH*n*_REG (*n*: 0-2) (0x0038+0x68n*)**



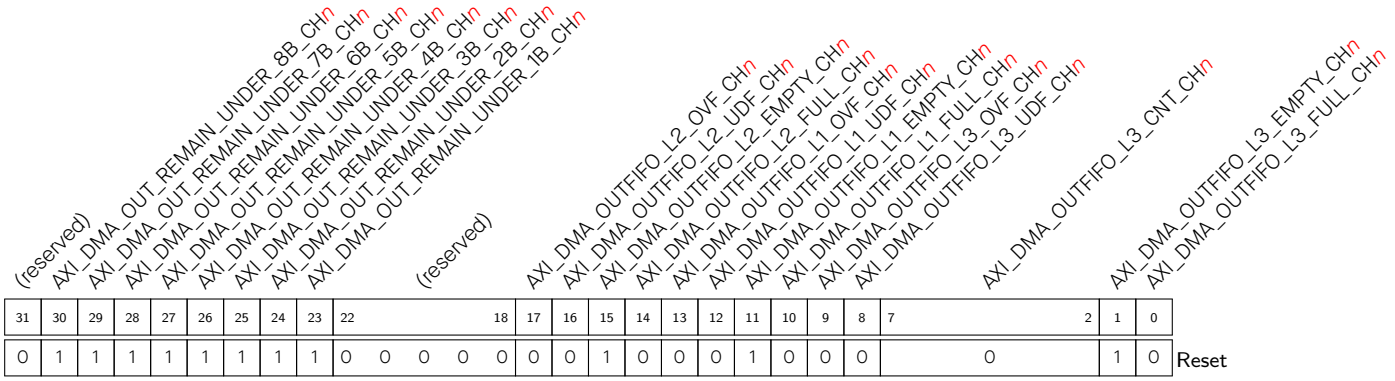
AXI_DMA_INLINK_DSCR_BFO_CH*n* Represents the current receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.139. AXI_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x003C+0x68*n*)**



AXI_DMA_INLINK_DSCR_BF1_CH*n* Represents the previous receive descriptor address for RX channel *n* that has already been fetched. (RO)

Register 3.140. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68n*)**



AXI_DMA_OUTFIFO_L3_FULL_CH*n* Represents whether L3 TX FIFO is full for TX channel *n*.

0: Not full

1: Full

(RO)

AXI_DMA_OUTFIFO_L3_EMPTY_CH*n* Represents whether L3 TX FIFO is empty for TX channel *n*.

0: Not empty

1: Empty

(RO)

AXI_DMA_OUTFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 TX FIFO for TX channel *n*.

Measurement unit: byte. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n* Represents whether L3 TX FIFO underflows for TX channel *n*.

0: Not underflow

1: Underflow

(RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n* Represents whether L3 TX FIFO overflows for TX channel *n*.

0: Not overflow

1: Overflow

(RO)

AXI_DMA_OUTFIFO_L1_FULL_CH*n* Represents whether L1 TX FIFO is full for TX channel *n*.

0: Not full

1: Full

(RO)

AXI_DMA_OUTFIFO_L1_EMPTY_CH*n* Represents whether L1 TX FIFO is empty for TX channel *n*.

0: Not empty

1: Empty

(RO)

Continued on the next page...

Register 3.140. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68**n*)

Continued from the previous page...

AXI_DMA_OUTFIFO_L1_UDF_CH*n* Represents whether L1 TX FIFO underflows for TX channel *n*.
0: Not underflow
1: Underflow
(RO)

AXI_DMA_OUTFIFO_L1_OVF_CH*n* Represents whether L1 TX FIFO overflows for TX channel *n*.
0: Not overflow
1: Overflow
(RO)

AXI_DMA_OUTFIFO_L2_FULL_CH*n* Represents whether L2 TX FIFO is full for TX channel *n*.
0: Not full
1: Full
(RO)

AXI_DMA_OUTFIFO_L2_EMPTY_CH*n* Represents whether L2 TX FIFO is empty for TX channel *n*.
0: Not empty
1: Empty
(RO)

AXI_DMA_OUTFIFO_L2_UDF_CH*n* Represents whether L2 TX FIFO underflows for TX channel *n*.
0: Not underflow
1: Underflow
(RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n* Represents whether L2 TX FIFO overflows for TX channel *n*.
0: Not overflow
1: Overflow
(RO)

AXI_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

AXI_DMA_OUT_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

Register 3.141. AXI_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x0160+0x68**n*)

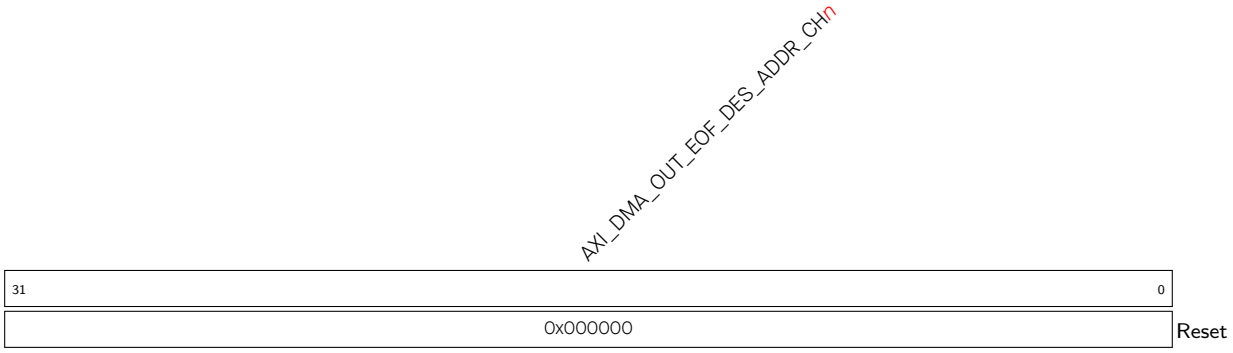


AXI_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address to be processed. (RO)

AXI_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

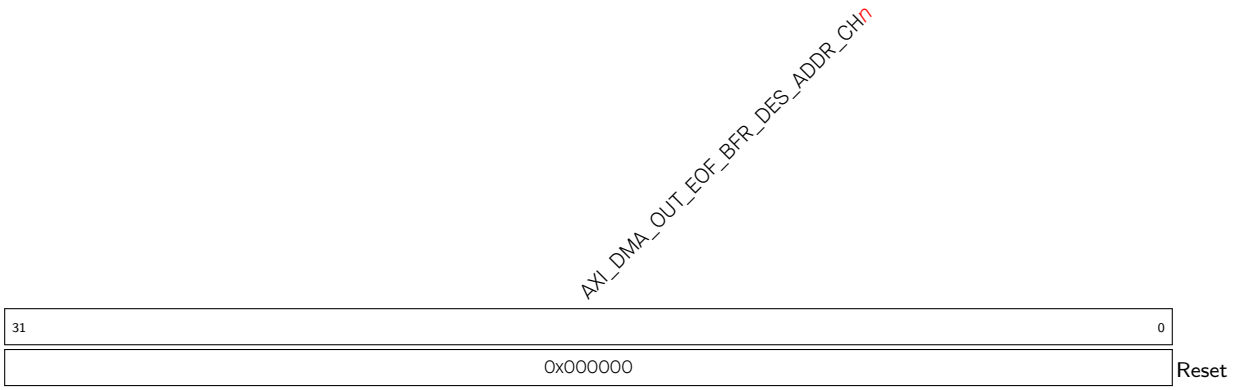
AXI_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.142. AXI_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0164+0x68**n*)



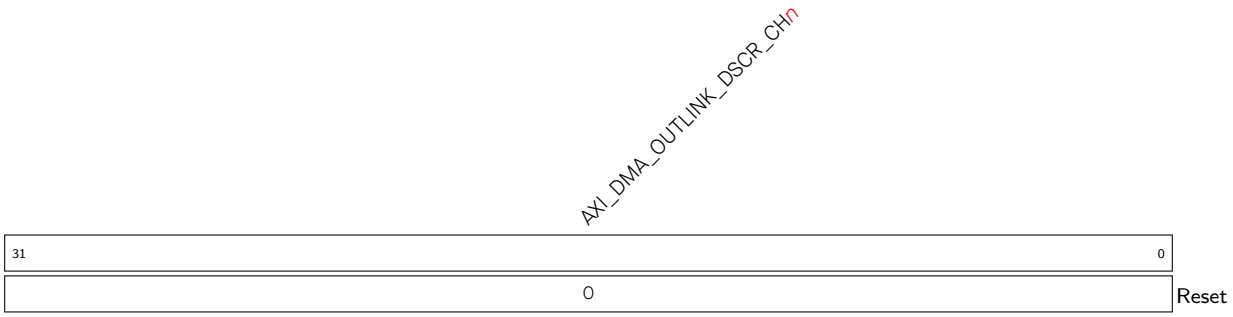
AXI_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the transmit descriptor address when this descriptor has an EOF flag for TX channel *n*. (RO)

Register 3.143. AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0168+0x68**n*)



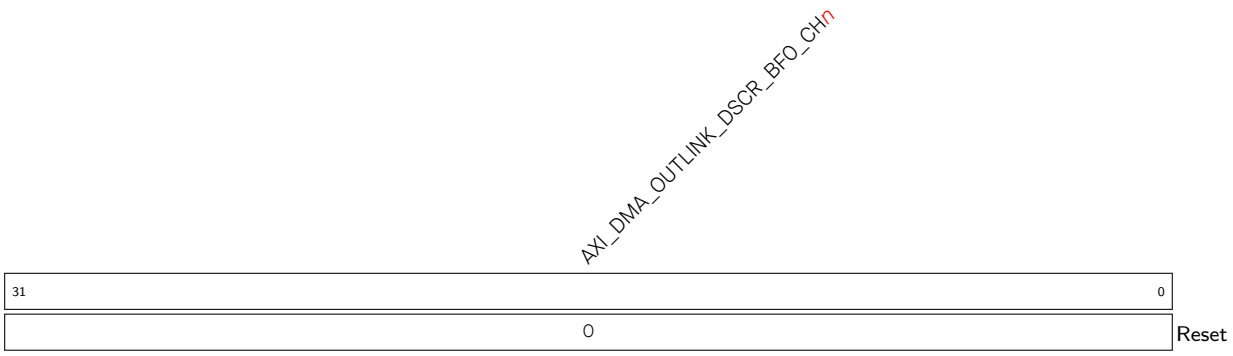
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the transmit descriptor address before the last transmit descriptor for TX channel *n*. (RO)

Register 3.144. AXI_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x016C+0x68**n*)



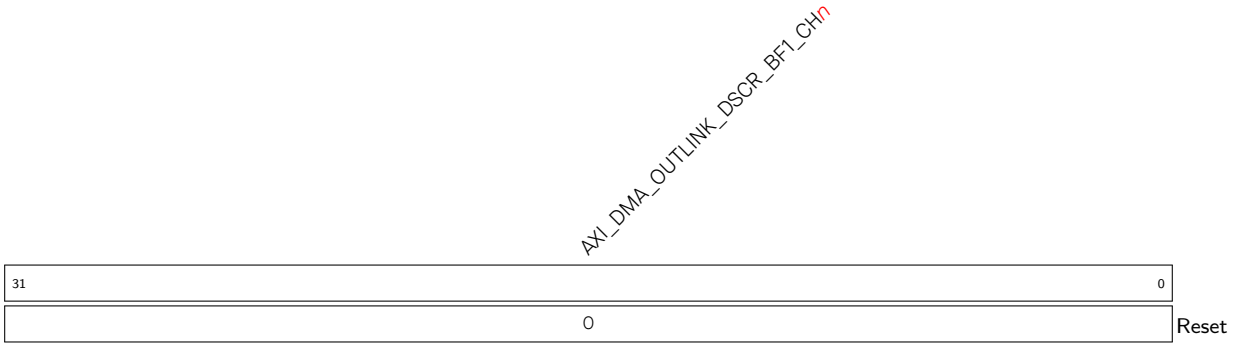
AXI_DMA_OUTLINK_DSCR_CH*n* Represents the next transmit descriptor address pointed to by the current transmit descriptor that has already been fetched. (RO)

Register 3.145. AXI_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0170+0x68**n*)



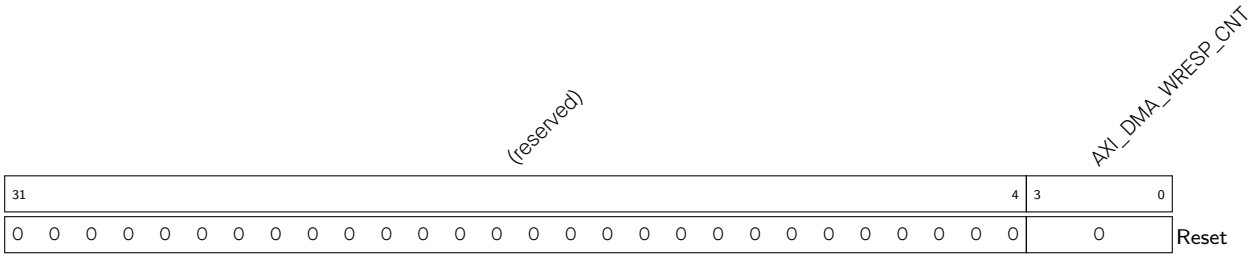
AXI_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the current transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.146. AXI_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0174+0x68n*)**



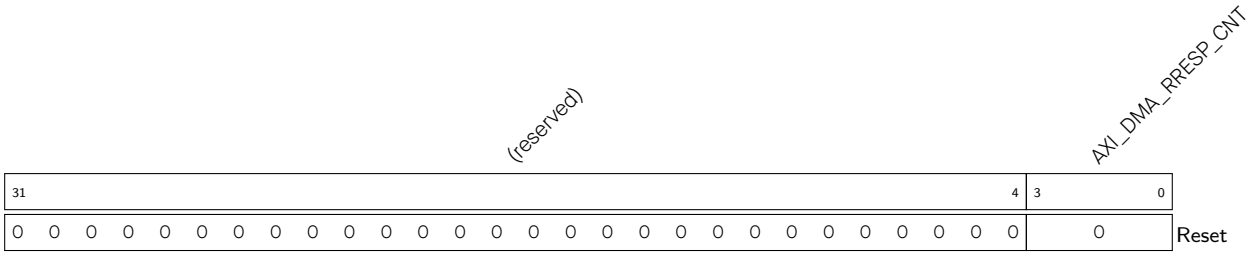
AXI_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the previous transmit descriptor address for TX channel *n* that has already been fetched. (RO)

Register 3.147. AXI_DMA_WRESP_CNT_REG (0x02B8)



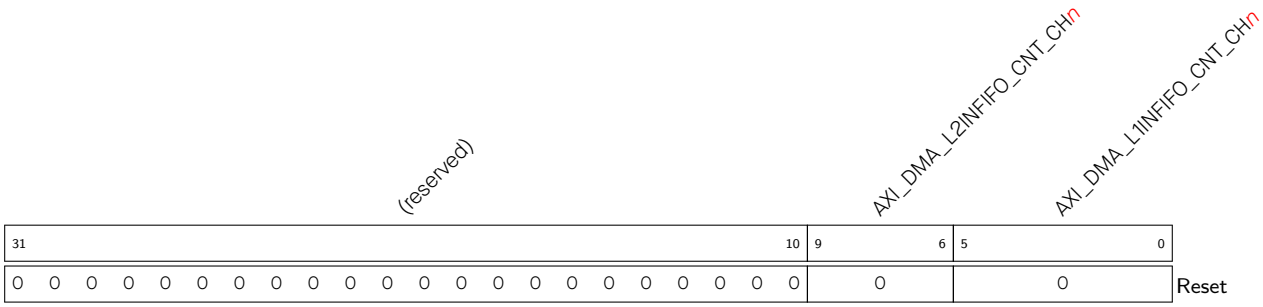
AXI_DMA_WRESP_CNT Represents the AXI write response count. (RO)

Register 3.148. AXI_DMA_RRESP_CNT_REG (0x02BC)



AXI_DMA_RRESP_CNT Represents the AXI read response count. (RO)

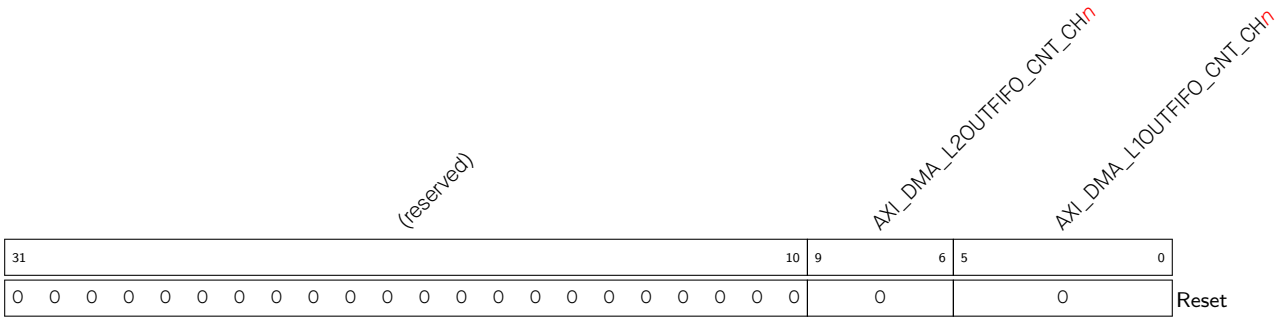
Register 3.149. AXI_DMA_INFIFO_STATUS1_CH*n*_REG (*n*: 0-2) (0x02C0+0x4*n*)



AXI_DMA_L1INFIFO_CNT_CH*n* Represents the number of data bytes in RX FIFO for RX channel *n*.
Measurement unit: byte. (RO)

AXI_DMA_L2INFIFO_CNT_CH*n* Represents the number of data bytes in RX FIFO for RX channel *n*.
Measurement unit: byte. (RO)

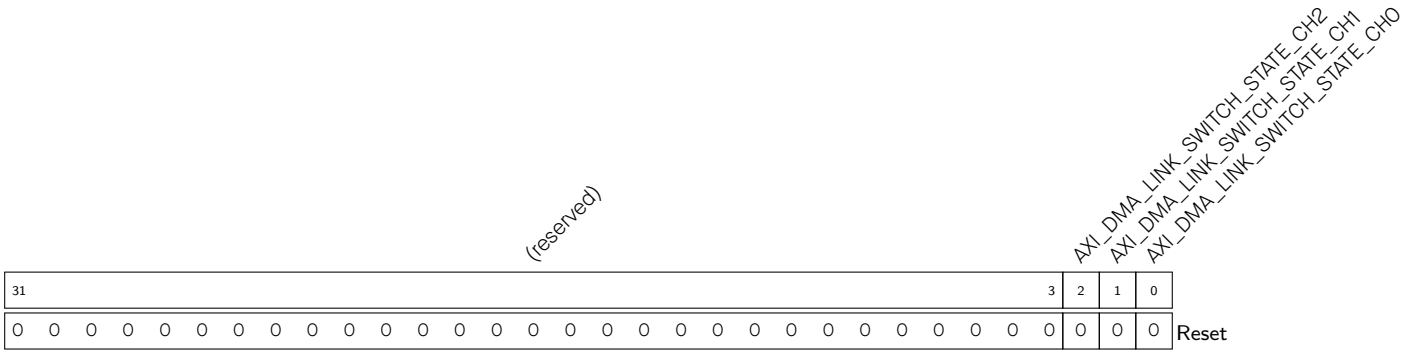
Register 3.150. AXI_DMA_OUTFIFO_STATUS1_CH*n*_REG (*n*: 0-2) (0x02CC+0x4*n*)



AXI_DMA_L1OUTFIFO_CNT_CH*n* Represents the number of data bytes in TX FIFO for TX channel *n*.
Measurement unit: byte. (RO)

AXI_DMA_L2OUTFIFO_CNT_CH*n* Represents the number of data bytes in TX FIFO for TX channel *n*.
Measurement unit: byte. (RO)

Register 3.151. AXI_DMA_LINK_SWITCH_STATE_REG (0x02DC)



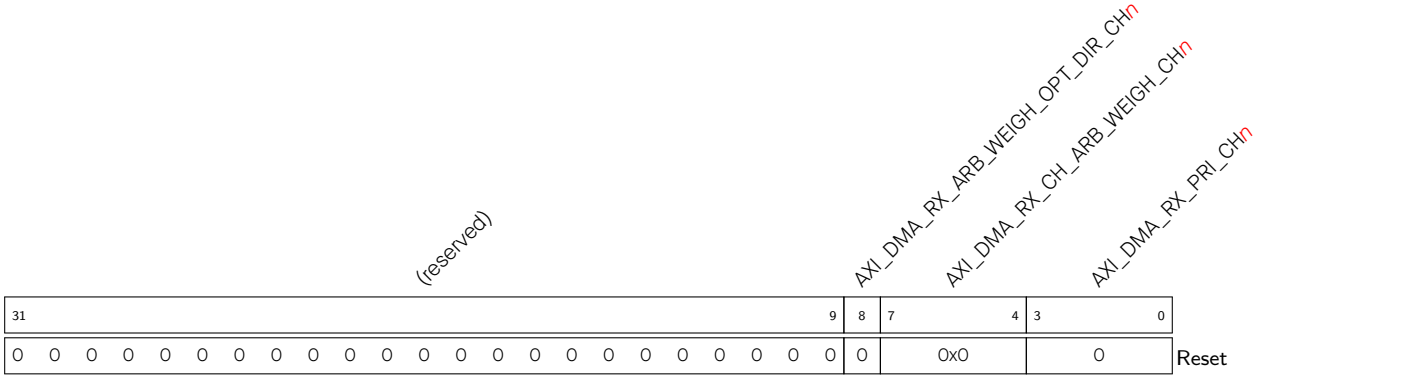
AXI_DMA_LINK_SWITCH_STATE_CH*n* (*n*: 0-2) Configures whether to enable linked list switch for channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.152. AXI_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x0040+0x68**n*)



AXI_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

AXI_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of RX channel *n*. Measurement unit: token. (R/W)

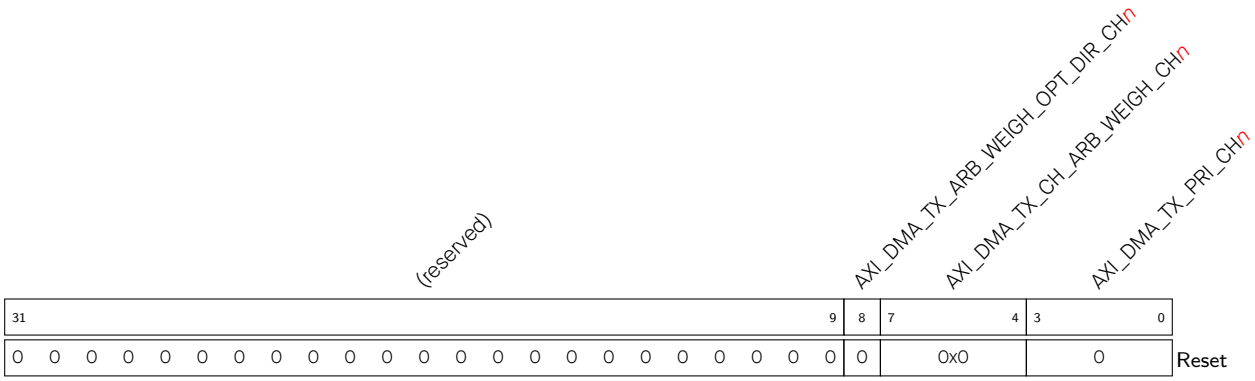
AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for RX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.153. AXI_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x0178+0x68n*)**



AXI_DMA_TX_PRI_CH*n* Configures the priority of TX channel *n*. A larger value means a higher priority. Value range: 0 ~ 15. (R/W)

AXI_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the arbitration weight of TX channel *n*. Measurement unit: token. (R/W)

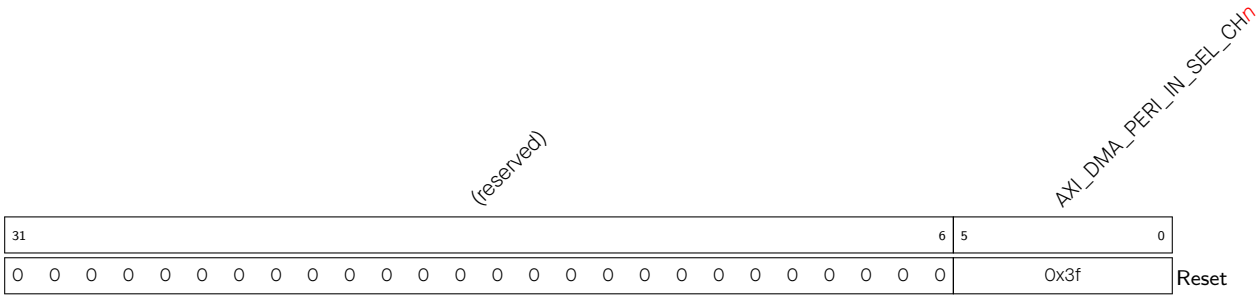
AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight arbitration optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 3.154. AXI_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0044+0x68n*)**



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

0: LCD or CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

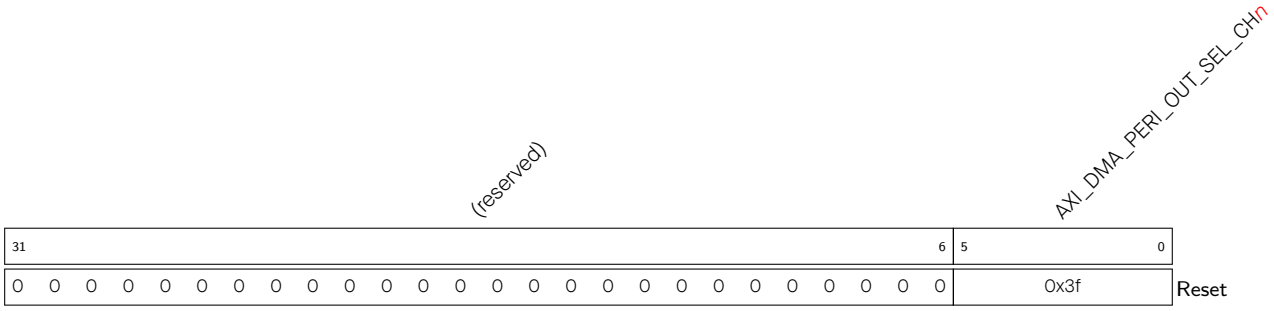
5: SHA

6 ~ 15: Dummy-6 ~ 15

16 ~ 63: Invalid value

(R/W)

Register 3.155. AXI_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x017C+0x68**n*)



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

0: LCD or CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

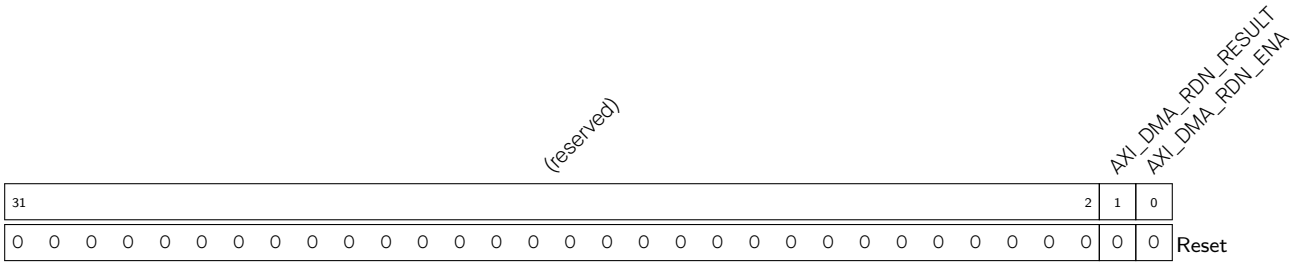
5: SHA

6 ~ 15: Dummy-6 ~ 15

16 ~ 63: Invalid value

(R/W)

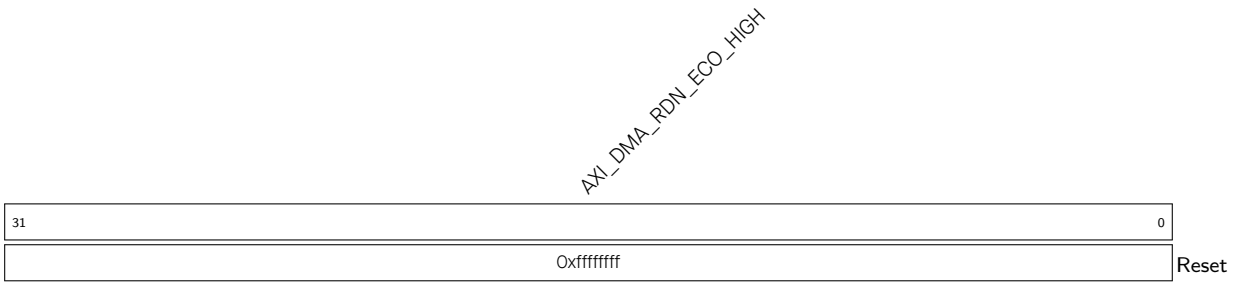
Register 3.156. AXI_DMA_RDN_RESULT_REG (0x02AC)



AXI_DMA_RDN_ENA Reserved. (R/W)

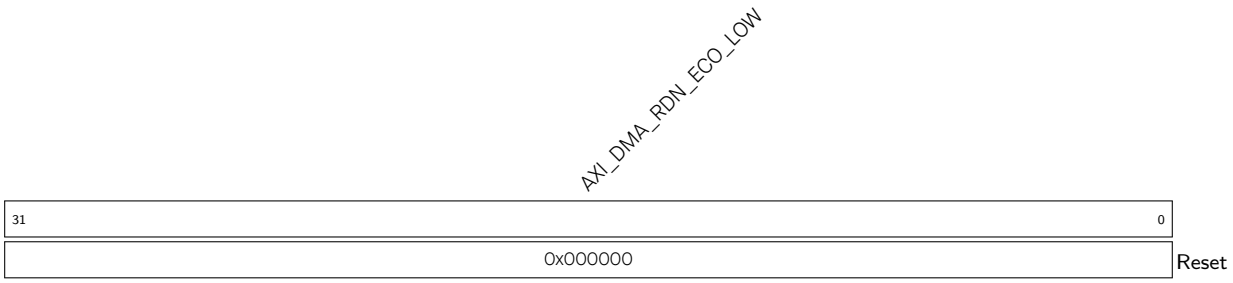
AXI_DMA_RDN_RESULT Reserved. (RO)

Register 3.157. AXI_DMA_RDN_ECO_HIGH_REG (0x02B0)



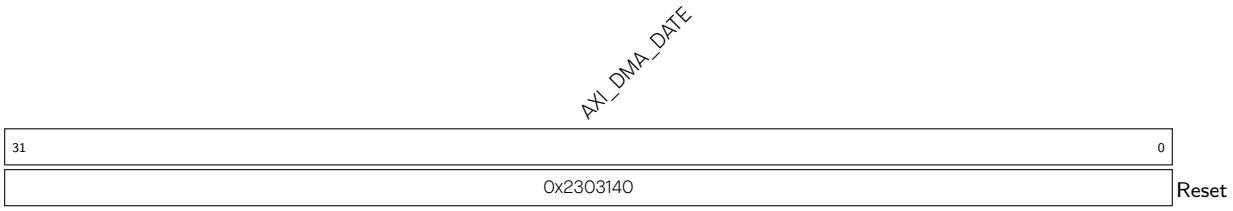
AXI_DMA_RDN_ECO_HIGH Reserved. (R/W)

Register 3.158. AXI_DMA_RDN_ECO_LOW_REG (0x02B4)



AXI_DMA_RDN_ECO_LOW Reserved. (R/W)

Register 3.159. AXI_DMA_DATE_REG (0x02D8)



AXI_DMA_DATE Version control register. (R/W)

Chapter 4

2D-DMA Controller (2D-DMA)

4.1 Overview

The 2D-DMA controller is a DMA (Direct Memory Access) dedicated to two-dimensional image processing. In addition to all the features of GDMA-AXI, it includes support for macroblock reordering and color space conversion (CSC) to better meet the data transfer requirements from JPEG and PPA. Notably, the 2D-DMA facilitates memory-to-memory transfers, enabling the movement of macroblocks between different segments of memory address space while concurrently performing color space conversion.

4.2 Features

- One AXI master interface
- Data transfer with unaligned starting addresses
- Memory-to-memory, peripheral-to-memory (RX), and memory-to-peripheral (TX) data transfer
- Four memory-to-peripheral channels, and three peripheral-to-memory channels
- Support for PPA and JPEG Codec
- Macroblock reordering
- Color space conversion
- Configurable channel priority and weight

4.3 Architecture

In ESP32-S31, only JPEG Codec and PPA support 2D-DMA. The 2D-DMA controller and CPU data bus have access to the same address space in memory. Figure 4.3-1 shows the basic architecture of the 2D-DMA controller.

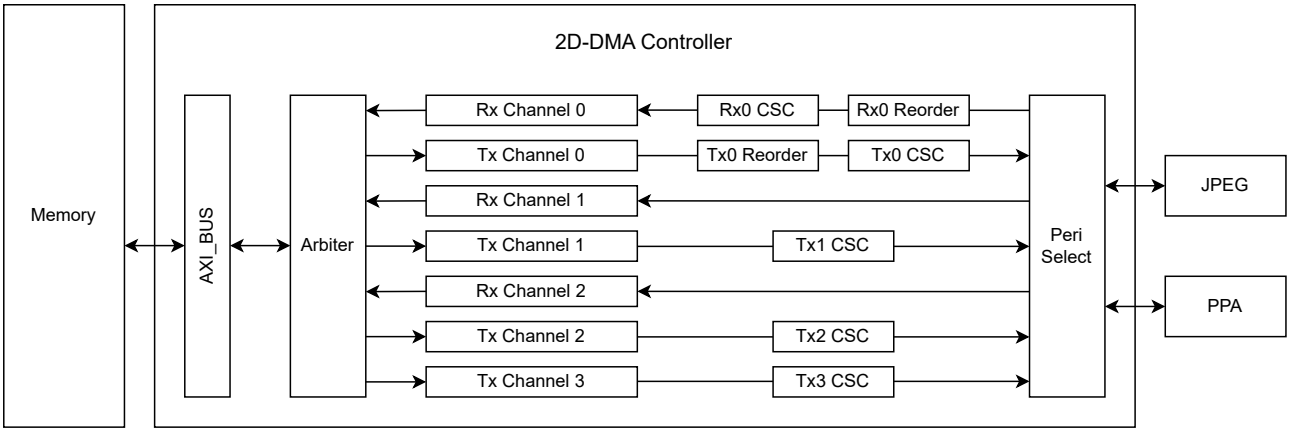


Figure 4.3-1. 2D-DMA Architecture

The 2D-DMA has seven independent channels: four transmit channels and three receive channels. Every channel can be connected to different peripherals, or in other words shared by peripherals.

The macroblock reordering and color space conversion features of the 2D-DMA are not supported by all channels. Among the channels, transmit channel 0 and receive channel 0 support macroblock reordering, while transmit channel 0 ~ 3 and receive channel 0 support color space conversion.

The 2D-DMA reads data from or writes data to the internal or external memory via AXI_BUS. Before the data transfer, the 2D-DMA uses configurable arbitration schemes for channels requesting read or write access. For the available address range of internal and external memory, please see Chapter 3 *System and Memory [to be added later]*.

The software can use the 2D-DMA through linked lists stored in internal or external memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number. For outlink*n*, *n* ranges from 0 to 3; for inlink*n*, *n* ranges from 0 to 2. The 2D-DMA reads an outlink*n* (i.e., a linked list of transmit descriptors) from memory and transmit data in corresponding memory according to the outlink*n*, or read an inlink*n* (i.e., a linked list of receive descriptors) and store received data into specific address space in memory according to the inlink*n*.

4.4 Functional Description

4.4.1 Transfer Mode

The 2D-DMA supports four transfer modes:

- 1D mode: enabled by setting the 2DEN field of DW0 and the mod field of DW2 to 0. In this mode, the 2D-DMA has the same functionalities as GDMA-AXI: it performs memory read/write operations based on the data length and memory space configured in the linked list, and the operation addresses are contiguous.
- 2D-MOD0 mode: enabled by setting the 2DEN field of DW0 to 1 and the mod field of DW2 to 0. In this mode, the 2D-DMA reads or writes a hb*vb macroblock starting from the (X, Y) coordinates of the HA*VA image. After the last data in each row has been read or written, the address jumps to the address of the first data in the next row to be read or written.

- 2D-MOD1 mode: enabled by setting the 2DEN field of DW0 to 1 and the mod field of DW2 to 1. In this mode, the 2D-DMA reads or writes multiple macroblocks from the starting address of the HA*VA image until the HA*VA image has been fully read or written. For JPEG, it is necessary to configure DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* or DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* to choose a macroblock size out of the three available options (horizontal × vertical): 8 pixels × 8 pixels, 8 pixels × 16 pixels, and 16 pixels × 16 pixels. For more information about macroblock size, see Chapter 17 *JPEG Codec*. For correspondence between macroblock size, hb, and vb of descriptors, see Table 4.4-4 and Table 4.4-5.
- DSCR-PORT mode: enabled by setting the 2DEN field of DW0 to 1, the mod field of DW2 to 0, and the DMA2D_OUT_DSCR_PORT_EN_CH*n* bit or the DMA2D_IN_DSCR_PORT_EN_CH*n* bit to 1. This mode is specifically designed for PPA. For details, see Chapter 18 *Pixel-Processing Accelerator (PPA)*.

1D Mode, 2D-MOD0 Mode, 2D-MOD1 Mode, and DSCR-PORT support both peripheral-to-memory and memory-to-peripheral data transfers. Only 1D Mode and 2D-MOD0 Mode support memory-to-memory data transfers.

4.4.2 Linked List

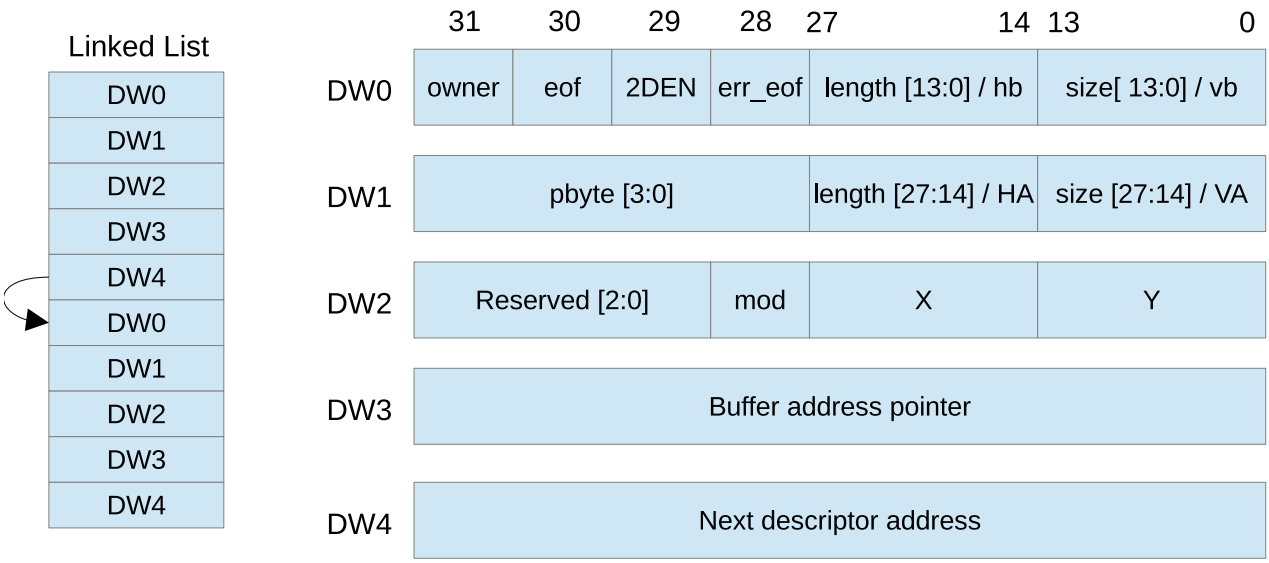


Figure 4.4-1. Structure of a Linked List

Figure 4.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of five words. Linked lists should be stored in the memory for the 2D-DMA to be able to use them. The meanings of a descriptor's fields are shown in Table 4.4-1.

Note: DMA2D_OUT is the prefix of transmit channel registers, and DMA2D_IN is the prefix of receive channel registers.

Table 4.4-1. Descriptor Field Description

Bits	Name	Description
DW0[31]	owner	Specifies who is allowed to access the buffer that this descriptor points to. • 0: CPU can access the buffer • 1: The 2D-DMA controller can access the buffer When the 2D-DMA controller stops using the buffer, this bit in a receive descriptor is automatically cleared by hardware, while this bit in a transmit descriptor can only be automatically cleared by hardware if DMA2D_OUT_AUTO_WRBACK_CH*n* is set to 1. When software loads a linked list, this bit should be set to 1.
DW0[30]	eof	Specifies the end of frame. • 0: The current descriptor is not the last descriptor of an image • 1: The current descriptor is the last descriptor of an image For receive descriptors, software needs to write 0 to this bit first, and hardware would set this bit to 1 after receiving a complete image. For transmit descriptors, software should set this bit in the last descriptor of an image to 1.
DW0[29]	2DEN	Specifies whether to enable 2D functions. • 0: 2D-DMA operates in 1D mode • 1: 2D-DMA operates in 2D mode
DW0[28]	err_eof	Specifies whether the received data has errors (JPEG only). • 0: The received data does not have errors • 1: The received data has errors For receive descriptors, the hardware sets this bit to 1 after receiving a complete image and detecting errors in the received data.
DW0[27:14]	length[13:0]/hb	**hb** in 2D mode – Specifies the horizontal width of the macroblocks to be moved in the unit of pixels. **length[13:0]** in 1D mode – Specifies the lower 14 bits of length, which is the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored in the buffer.
DW0[13:0]	size[13:0]/vb	**vb** in 2D mode – Specifies the vertical height of the macroblocks to be moved in the unit of pixels. **size[13:0]** in 1D mode – Specifies the lower 14 bits of size, which is the capacity of the buffer that this descriptor points to in the unit of bytes.

Bits	Name	Description
DW1[31:28]	pbyte	In 2D mode, specifies the number of bytes per pixel of the image to be transferred to/from the memory. (If color space conversion is applied, this field represents the number of bytes per pixel of the image stored in the memory.) • 0: 0.5 byte/pixel • 1: 1 byte/pixel • 2: 1.5 bytes/pixel • 3: 2 bytes/pixel • 4: 3 bytes/pixel • 5: 4 bytes/pixel In 1D mode, this field is not used and can be any value.
DW1[27:14]	length[27:14]/HA	**HA** in 2D mode – Specifies the horizontal width of the original image corresponding to the macroblocks to be moved in the unit of pixels. **length[27:14]** in 1D mode – Specifies the higher 14 bits of length.
DW1[13:0]	size[27:14]/VA	**VA** in 2D mode – Specifies the vertical height of the original image corresponding to the macroblocks to be moved in the unit of pixels. **size[27:14]** in 1D mode – Specifies the higher 14 bits of size.
DW2[31:29]	Reserved	Reserved. Irrelevant.
DW2[28]	mod	Specifies the macroblock read mode. • 0: 2D-DMA reads or writes a single macroblock with horizontal width hb and vertical height vb only once • 1: 2D-DMA reads or writes multiple macroblocks with horizontal width hb and vertical height vb until the image data with horizontal width HA and vertical height VA is fully read or written In 1D mode, this bit must be 0. **Note:** hb/vb/HA/VA are in the unit of pixels.
DW2[27:14]	X	In 2D mode, specifies the X coordinate of the hb*vb image block within the HA*VA image in the unit of pixels. In 1D mode, this field is not used and can be any value.
DW2[13:0]	Y	In 2D mode, specifies the Y coordinate of the hb*vb image block within the HA*VA image in the unit of pixels. In 1D mode, this field is not used and can be any value.
DW3	Buffer address pointer	In 2D mode, specifies the address of the HA*VA image block origin in the buffer (the address of the original image's first byte in the memory). In 1D mode, specifies the starting address of the buffer.

Bits	Name	Description
DW4	Next descriptor address	Address of the next descriptor. If the current descriptor is the last descriptor in the linked list, this value can be 0. This address can point to the address space of internal or external memory. **Note:** The starting address for storing 2D-DMA descriptors must be 8-byte aligned.

4.4.3 Padding in 2D-MOD1 Mode and DSCR-PORT Mode

This section only applies to the TX direction of 2D-DMA transfers in 2D-MOD1 and DSCR-PORT mode.

For JPEG decoding, the basic unit of image data input is a block of 8 × 8, 8 × 16, or 16 × 16 (pixels). In 2D-MOD1 mode, when the length and width of the image are not integer multiples of the basic unit, the image needs to pad out to integer multiples of the basic unit based on the previous or the current macroblock.

For PPA, the basic unit for transmit channels in DSCR-PORT mode is 36 × 36 (YUV420), 36 × 34 (YUV422) and 34 × 34 (horizontal × vertical). The width and height of a basic unit for different transmit channels are configured using DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* and DMA2D_OUT_DSCR_PORT_BLK_V_CH*n*. When the length and width of the image are not integer multiples of the basic unit, the image needs to pad out.

Below are the padding rules rules:

- Horizontally, the padding depends on the pbyte field of DW1.
  - When pbyte is 0 (0.5 byte/pixel), the remaining pixels must be even, and the padding is the last 0.5 byte.
  - When pbyte is 1 (1 byte/pixel), the padding is the last 1 byte.
  - When pbyte is 2 (1.5 bytes/pixel), the remaining pixels must be even, and the padding is the last 3 bytes.
  - When pbyte is 3 (2 bytes/pixel), the padding is the last 2 bytes.
  - When pbyte is 4 (3 bytes/pixel), the padding is the last 3 bytes.
  - When pbyte is 5 (4 bytes/pixel), the padding is the last 4 bytes.
- Vertically, the padding is the last row.

Figure 4.4-2 and Figure 4.4-3 illustrate the padding method.

In Figure 4.4-2, a blue grid represents a basic unit (a block of 8 × 8, 8 × 16, or 16 × 16 pixels), and a yellow box (incomplete basic unit) along with a green box (padding) forms a basic unit.

In Figure 4.4-3, a grid represents one pixel, and the numbers inside each grid represent pixel values.

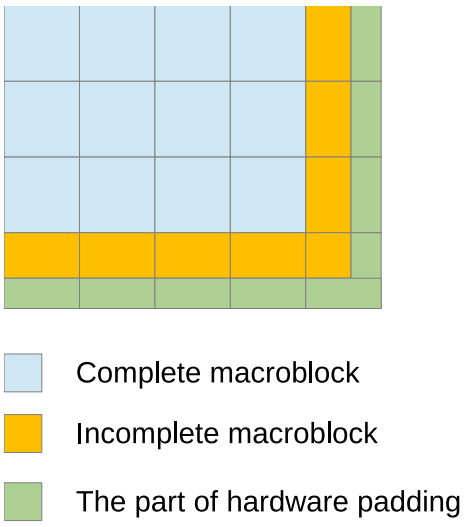


Figure 4.4-2. Padding Illustration (Overall)

1	2	3	3	3	3	3	3
4	5	6	6	6	6	6	6
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9

Figure 4.4-3. Padding Illustration (Detail)

4.4.4 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer

When DMA2D_MEM_TRANS_EN_CH*n* is 0, the 2D-DMA controller enters peripheral transfer mode. The transmit channel reads data from the memory and transmits it to peripherals, while the receive channel receives data from peripherals and stores it into the memory.

The transmit direction refers to data transfer from memory to peripheral, and the receive direction refers data transfer from peripheral to memory. A transmit channel transfers data in the specified memory location to a peripheral's transmitter via an outlink*n*, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlink*n*.

Every transmit and receive channel can be connected to any peripheral that supports 2D-DMA. Table 4.4-2 and Table 4.4-3 illustrate how to select the peripheral to be connected via registers. "Dummy-*n*" corresponds to register values for memory-to-memory data transfer. When a channel is connected to a peripheral, the rest

of the channels cannot be connected to that peripheral.

Table 4.4-2. Selecting Peripherals via Register Configuration in TX Direction

DMA2D_PERI_OUT_SEL_CH*n*	Peripheral
0	JPEG
1	PPA's SRM module
2	PPA's BLEND0 module
3	PPA's BLEND1 module
4 ~ 7	Dummy-4 ~ 7

Table 4.4-3. Selecting Peripherals via Register Configuration in RX Direction

DMA2D_PERI_IN_SEL_CH*n*	Peripheral
0	JPEG
1	PPA's SRM module
2	PPA's BLEND channel
3 ~ 7	Dummy-3 ~ 7

4.4.5 Memory-to-Memory Data Transfer

The 2D-DMA controller allows memory-to-memory data transfer only in 1D mode and 2D-MOD0 mode. Such data transfer can be enabled by setting DMA2D_IN_MEM_TRANS_EN_CH*n*, which connects the output of transmit channel *n* to the input of receive channel *n*. Note that a transmit channel can only be connected to the receive channel with the same number (*n*), and DMA2D_IN_PERI_SEL_CH*n* and DMA2D_OUT_PERI_SEL_CH*n* should be configured to any value corresponding to "Dummy".

Memory-to-memory data transfer can be used in combination with color space conversion so that a macroblock can be moved from one segment of memory address space to another with its color space converted.

4.4.6 Macroblock Reordering

4.4.6.1 Macroblock Reordering in TX Direction

When the 2D-DMA accesses external memory via SPI, in order to improve the bandwidth utilization of SPI, it is necessary to initiate AXI transfers with a larger burst length. In 2D-MOD1 mode, the 2D-DMA can increase the burst length of AXI transfers by fetching multiple macroblocks at once. For example, the 2D-DMA can fetch five macroblocks of 8x8 pixels in one operation, and in this case, the hb field in the descriptor should be 40 and vb should be 8. However, the JPEG requires the 2D-DMA to send data in the order of macroblocks. This means that the 2D-DMA needs to reorder the fetched macroblocks to meet the requirements of JPEG.

Recommended configurations for macroblock reordering are listed in Table 4.4-4.

Table 4.4-4. Recommended Configurations for Macroblock Reordering in TX Direction

Byte/Pixel	Macroblock Size (Pixel)	Optimal Macroblock Count	hb	vb
3	8× 8	5	40	8
3	16 × 8	2	32	8
3	16 × 16	2	32	16
2	8 × 8	8	64	8
2	16 × 8	4	64	8
2	16 × 16	3	48	16
1	8 × 8	16	128	8

The macroblock size in TX direction should be configured via the DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* field.

4.4.6.2 Macroblock Reordering in RX Direction

Currently, in RX direction the 2D-DMA can only reorder the data decoded by JPEG (the bitstream is 1 scan, see Chapter 17 *JPEG Codec*). The 2D-DMA breaks down color component data decoded by JPEG into pixels, assembles them into larger macroblocks, and writes these blocks into the memory.

The data formats decoded by JPEG include GRAY, YUV444, YUV422, and YUV420. Recommended configurations for macroblock reordering are listed in Table 4.4-5.

Table 4.4-5. Recommended Configuration Table for Macroblock Reordering in 2D-DMA Receive Direction (Without Color Format Conversion)

Format	Macroblock Size (pixels)	Optimal Macroblock Count	hb	vb
GRAY	8 × 8	12	96	8
YUV444	8 × 8	5	40	8
YUV422	16 × 8	4	64	8
YUV420	16 × 16	3	48	16

The macroblock size in TX direction should be configured via the DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* field.

Figure 4.4-4 shows the layout of macroblocks outputted by the JPEG decoder. Each blue grid in this figure represents 8x8 bytes of continuous data, and for each color format the figure displays four macroblocks of 8 × 8 pixels.

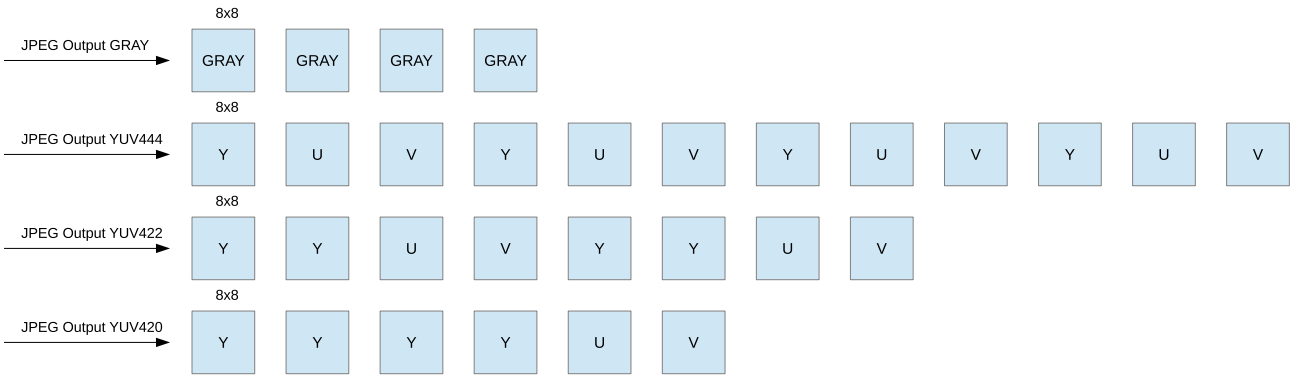


Figure 4.4-4. Macroblock Layout of JPEG Decoder Output Data

4.4.7 Color Space Conversion

On the transmit side, the 2D-DMA converts the color space after assembling the data read from memory into pixels. The 2D-DMA supports the following conversion except YUV420:

- RGB888 to RGB565
- RGB565 to RGB888
- RGB888 to YUV444
- RGB888 to YUV422 (the number of pixels per row in RGB format must be even)
- YUV444 to RGB888
- YUV422 to RGB888

On the receive side, only pixels received from JPEG requires color space conversion. The 2D-DMA supports the following conversion:

- YUV444 to RGB888/RGB565/YUV422/YUV420
- YUV422 to YUV444 to RGB888/RGB565/YUV420
- YUV420 to YUV444 to RGB888/RGB565

Note:

- For JPEG decoding, i.e., the receive side of 2D-DMA, the hardware will forcibly convert YUV422 and YUV420 to the YUV444 format. Software can optionally further convert the format to RGB888/RGB565 with the color space conversion feature.

Figure 4.4-5 shows the structure of the color space conversion module.

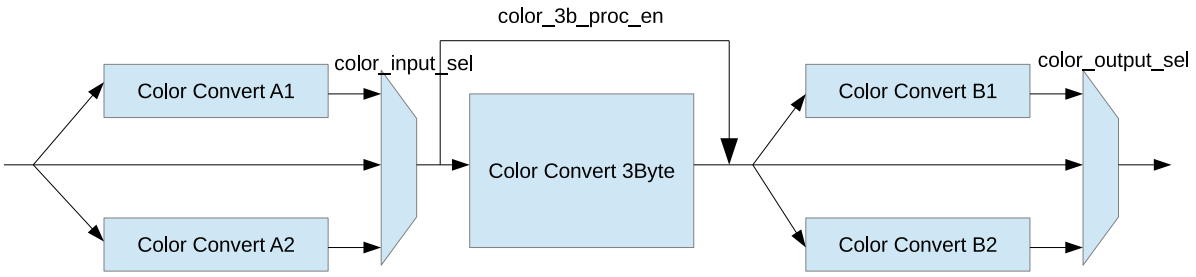


Figure 4.4-5. Color Space Conversion

Color space conversion is performed in 3 stages. The first stage converts a specific 16-bit color format (2 bytes/pixel) to a 24-bit color format (3 bytes/pixel). The source color formats can be configured via the DMA2D_OUT_COLOR_INPUT_SEL_CH*n* or DMA2D_IN_COLOR_INPUT_SEL_CH0 field. (See the supported color format conversion in the first stage by clicking the fields)

The second stage further converts the output data of the first stage using the following formula:

$$256 * Q = A[9:0] * x + B[10:0] * y + C[9:0] * z + D[17:0]$$

In this formula, Q is the output data; x, y, and z are the most significant, middle, and the least significant byte of the input data respectively; A, B, C, and D are the corresponding multiplication or addition coefficients, which are signed numbers. At most 3 bytes of data can be processed in one go, and therefore 3 * 4 = 12 coefficients need to be configured via the following fields:

- DMA2D_OUT_COLOR_PARAM_H0_CH*n*/DMA2D_IN_COLOR_PARAM_H0_CH0
- DMA2D_OUT_COLOR_PARAM_H1_CH*n*/DMA2D_IN_COLOR_PARAM_H1_CH0
- DMA2D_OUT_COLOR_PARAM_M0_CH*n*/DMA2D_IN_COLOR_PARAM_M0_CH0
- DMA2D_OUT_COLOR_PARAM_M1_CH*n*/DMA2D_IN_COLOR_PARAM_M1_CH0
- DMA2D_OUT_COLOR_PARAM_L0_CH*n*/DMA2D_IN_COLOR_PARAM_L0_CH0
- DMA2D_OUT_COLOR_PARAM_L1_CH*n*/DMA2D_IN_COLOR_PARAM_L1_CH0

PARAM_H*x*, PARAM_M*x*, and PARAM_L*x* (*x* is 0 or 1) in field names represent the parameters for the three bytes of data respectively. Take PARAM_H*x* as examples, the lower 10 bits of the 21-bit PARAM_H0 is coefficient A, the higher 11 bits of PARAM_H0 is coefficient B, the lower 10 bits of the 28-bit PARAM_H1 is coefficient C, and the higher 18 bits of PARAM_H1 is coefficient D.

The third stage, similar to the first stage, outputs the color format converted in the second stage directly, or further converts the 24-bit color format (3 bytes/pixel) to a 16-bit color format. This is configured using DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* or DMA2D_IN_COLOR_OUTPUT_SEL_CH0.

The following table lists the parameter configurations for commonly used color space conversions:

Table 4.4-6. Parameter Configuration for Color Space Conversion in TX Direction

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
No conversion	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Only scramble order	2/3	0	2	N/A	N/A	N/A	N/A	N/A	N/A

Cont'd on next page

Table 4.4-6 – cont'd from previous page

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
RGB888 → RGB565	3	0	0	N/A	N/A	N/A	N/A	N/A	N/A
RGB565 → RGB888	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
RGB888 → (601) YUV444	3	1	2	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV444	3	1	2	2742F	400010	1EABE6	2000070	1E6870	20003F6
RGB888 → (601) YUV422-MIPI	3	1	1	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV422-MIPI	3	1	1	2742F	400010	1EABE6	2000070	1E6870	20003F6
YUV444 → (601) RGB888	3	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (709) RGB888	3	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV422-MIPI → (601) RGB888	1	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422-MIPI → (709) RGB888	1	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Table 4.4-7. Parameter Configuration for Color Space Conversion in RX Direction

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
No conversion	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Only scramble order	1/2	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV420 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV422	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV420 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (601) RGB888/RGB565	2	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV420 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Cont'd on next page

Table 4.4-7 – cont'd from previous page

Conversion	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
YUV422 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV444 → (709) RGB888/RGB565	2	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

Additionally, the color space conversion module can scramble the order of at most three bytes per pixel. This can be done before the color space conversion in TX direction, and before or after the color space conversion in RX direction. Table 4.4-8, Table 4.4-9, and Table 4.4-10 show the byte order per pixel required by the color space conversion module. For the JPEG decoder, if the byte order per pixel is not scrambled after the macroblock reordering in RX direction, it will conform to the requirements of color space conversion. Assuming that the order of an input pixel is represented from MSB to LSD as 2, 1, 0, the order can be scrambled and outputted as 210 (default), 201, 102, 120, 021, or 012. In RX direction, this feature can be configured before or after the color space conversion via DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 orDMA2D_IN_SCRAMBLE_SEL_POST_CH0 respectively; in TX rection, this feature can be configured before the color space conversion via DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n*.

Table 4.4-8. RGBB888 Storage Format

	LSB								MSB
LINE	B1	G1	R1	B2	G2	R2	B3	G3	R3

Table 4.4-9. YUV444 Storage Format

	LSB								MSB
LINE	V1	U1	Y1	V2	U2	Y2	V3	U3	Y3

Table 4.4-10. YUV422 Storage Format

	LSB							MSB
LINE	U1	Y1	V1	Y2	U3	Y3	V3	Y4

4.4.8 Enabling 2D-DMA

The software uses the 2D-DMA controller through linked lists. For receive channels, software loads an inlink, configures the DMA2D_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets the DMA2D_INLINK_START_CH*n* bit to enable the 2D-DMA. For transmit channels, software loads an outlink, prepares data to be transmitted, configures the DMA2D_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets the DMA2D_OUTLINK_START_CH*n* bit to enable the 2D-DMA. The DMA2D_INLINK_START_CH*n* bit and DMA2D_OUTLINK_START_CH*n* bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a 2D-DMA transfer that is already started. Naively, it would seem to be possible to do this by clearing the eof bit of the final descriptor in the existing list and setting its next descriptor address pointer field (DW4) to the first descriptor of the to-be-added list. However, this strategy fails if the existing 2D-DMA transfer is almost or entirely finished. Instead, the 2D-DMA controller

has specialized logic to make sure a transfer can be continued or restarted: if the transfer is ongoing, the controller will make sure to take the appended descriptors into account; if the transfer has already finished, the controller will restart with the new descriptors. This is implemented by the Restart function.

When using the Restart function, software needs to

1. Set DMA2D_INLINK_STOP_CH*n* or DMA2D_OUTLINK_STOP_CH*n* to stop the 2D-DMA when it finishes processing the current descriptor
2. Rewrite the address of the first descriptor in the new list to DW4 of the last descriptor in the loaded list
3. Set DMA2D_INLINK_RESTART_CH*n* or DMA2D_OUTLINK_RESTART_CH*n* (these two bits are cleared automatically by hardware)

By doing so, hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

4.4.9 Linked List Reading Process

Once configured and enabled by software, the 2D-DMA controller starts to read the linked list from the memory. The 2D-DMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, the corresponding 2D-DMA channel will start data transfer. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either DMA2D_IN_DSCR_ERR_CH*n*_INT or DMA2D_OUT_DSCR_ERR_CH*n*_INT), and the channel will remain idle. This channel can be restarted by setting DMA2D_OUTLINK_START_CH*n* or DMA2D_INLINK_START_CH*n* again.

The checks performed on descriptors are:

- Owner bit check when DMA2D_IN_CHECK_OWNER_CH*n* or DMA2D_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the buffer is accessed by the CPU. In this case, the owner bit fails the check. The owner bit will not be checked if DMA2D_IN_CHECK_OWNER_CH*n* or DMA2D_OUT_CHECK_OWNER_CH*n* is 0.
- Descriptor address check, which checks if the descriptor address is located in configured memory space. If a 2D-DMA descriptor points to DMA2D_ACCESS_INTR_MEM_START_ADDR ~ DMA2D_ACCESS_INTR_MEM_END_ADDR (the value should be located in internal memory) or DMA2D_ACCESS_EXTR_MEM_START_ADDR ~ DMA2D_ACCESS_EXTR_MEM_END_ADDR (the value should be located in external memory), it passes the check. For details, please refer to Section 4.4.11).
- HA/VA/hb/vb/SIZE check, which checks if any of these fields is 0. If any field is 0, it fails the check. HA/VA/hb/vb fields are not checked in 1D mode, while SIZE is checked only in 1D mode.

After the software detects a descriptor error interrupt, it must reset the corresponding channel, and restart the 2D-DMA DMA by setting the DMA2D_OUTLINK_START_CH*n* or DMA2D_INLINK_START_CH*n* bit.

To quickly obtain receive or transmit descriptors from the memory, AXI burst transfers for 2D-DMA can be enabled by setting DMA2D_INDSCR_BURST_EN_CH*n* or DMA2D_OUTDSCR_BURST_EN_CH*n*.

4.4.10 EOF

Note: In this chapter, EOF of transmit descriptors refers to eof (i.e., bit 30 of DW0), while EOF of receive descriptors refers to both eof and err_eof (i.e., bit 28 of DW0).

The 2D-DMA controller uses EOF (end of frame) flags to indicate the end of the image transmission.

For data transmission, the 2D-DMA generates two types of EOF interrupts:

- DMA2D_OUT_EOF_CH*n*_INT, generated when the eof bit of any descriptor in the linked list is set, and the data corresponding to this descriptor has been transmitted. Usually, the eof bit is set in the last descriptor for the transmission of an image. This interrupt is enabled by setting the DMA2D_OUT_EOF_CH*n*_INT_ENA bit.
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT, generated when the eof bit of the last descriptor in the linked list is set, and the data corresponding to the last descriptor has been transmitted. This interrupt is enabled by setting the DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA bit.

For data reception, the 2D-DMA also generates two types of EOF interrupts:

- DMA2D_IN_SUC_EOF_CH*n*_INT, generated when an image has been received. This interrupt is enabled by setting the DMA2D_IN_SUC_EOF_CH*n*_INT_ENA bit.
- (JPEG only) DMA2D_IN_ERR_CH*n*_EOF_INT, generated when an image has been received with errors. This interrupt is enabled by setting the DMA2D_IN_ERR_EOF_CH*n*_INT_ENA bit, and is valid only when the channel is connected to JPEG.

When detecting an DMA2D_OUT_TOTAL_EOF_CH*n*_INT or DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, software can read the value of the DMA2D_OUT_EOF_DES_ADDR_CH*n* or DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* field, which stores the address of the finished descriptor. Therefore, the software can tell which descriptors have been used and reclaim them as needed. In RX direction, the address of the descriptor can also be stored to the DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* field when an DMA2D_IN_ERR_EOF_CH*n*_INT interrupt is triggered.

4.4.11 Accessing Memory

Any transmit and receive channels of the 2D-DMA can access the internal memory address space configured by DMA2D_ACCESS_INTR_MEM_START_ADDR and DMA2D_ACCESS_INTR_MEM_END_ADDR, and the external memory address space configured by DMA2D_ACCESS_EXTR_MEM_START_ADDR and DMA2D_ACCESS_EXTR_MEM_END_ADDR.

The 2D-DMA can send data in the internal memory in burst mode via the AXI bus. Burst transfer on the AXI bus is enabled by default, and cannot be disabled. Users can configure the number of data bytes in a single AXI burst (burst length) to 8, 16, 32, 64, or 128 bytes via DMA2D_IN_MEM_BURST_LENGTH_CH*n* and DMA2D_OUT_MEM_BURST_LENGTH_CH*n*.

When the 2D-DMA accesses the encrypted external memory space (see Chapter 19 *External Memory Encryption and Decryption (XTS_AES) [to be added later]*), buffer address pointer and corresponding data should be 16-byte aligned; when it accesses the configured internal memory and non-encrypted external memory space, there are no requirements for descriptor field alignment.

4.4.12 Arbitration

To ensure timely response to peripherals running at a high speed with low latency, the 2D-DMA controller implements two arbitration schemes, based on channel priority and channel weight respectively.

- Priority arbitration: Each channel can be assigned a priority from 0 ~15 (in total 16 levels). The larger the number, the higher the priority. When several channels perform data transfers at the same time, the

2D-DMA would respond to the transfer requests according to priority levels, and the channel with higher priority would get a response more timely.

- Weight arbitration:
  - Each channel is assigned a weight (i.e., the number of tokens) from 0 ~ 15. The 2D-DMA divides the AXI bus clock period into multiple time slots, and in each time slot, the number of transfers performed by a channel is determined by the number of its tokens. Every time a channel performs a data transfer, one of its tokens is spent. If all of its tokens have been spent in a time slot, then this channel's transfer request will no longer be responded to until the time slot expires, or until tokens of all channels have been spent and the new time slot starts.
  - If the number of tokens assigned to a channel is not zero, the 2D-DMA waits for this channel's tokens to be spent even if it does not have data transfer requests, and will not exit from this time slot ahead of expiration. This leads to a waste of bandwidth. Therefore, the 2D-DMA provides weight arbitration optimization. When this feature is enabled, a channel without data transfer requests will not be involved in the arbitration and its tokens will be ignored. When all channels no longer have transfer requests, the 2D-DMA arbiter exits from the current time slot before expiration. This way, the arbitration response is accelerated and the transfer efficiency is improved.

Note:

- If channels have the same weight but different priorities, on condition that the number of bytes to be transferred is the same and not large, channels with higher priorities would finish data transfers first.
- If channels have different priorities and weights, on condition that the number of bytes to be transferred is the same, channels with higher weights would be allocated more bandwidth and finish data transfers first.

4.5 Event Task Matrix Feature

The 2D-DMA controller on ESP32-S31 supports the Event Task Matrix (ETM) function, which allows 2D-DMA's ETM tasks to be triggered by any peripherals' ETM events, or 2D-DMA's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to 2D-DMA. For more information, please refer to Chapter 10 Event Task Matrix (ETM) [to be added later].

2D-DMA can receive the following ETM tasks:

- DMA2D_TASK_IN_START_CH*n*: Enables the corresponding receive channel *n* for data transfer.
- DMA2D_TASK_OUT_START_CH*n*: Enables the corresponding transmit channel *n* for data transfer.

  Note:
  Above ETM tasks can achieve the same functions as CPU configuring DMA2D_INLINK_START_CH*n* and DMA2D_OUTLINK_START_CH*n*. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 1, only ETM tasks can be used to configure the transfer direction and enable the corresponding 2D-DMA channel. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 0, only CPU can be used to enable the corresponding 2D-DMA channel.

- DMA2D_TASK_IN_DSCR_READY_CH*n*: Allows receive channel *n* to process the next descriptor.

- DMA2D_TASK_OUT_DSCR_READY_CH*n*: Allows transmit channel *n* to process the next descriptor.

> **Note:**
> In normal conditions, after finishing with the current descriptor if the next descriptor is available (i.e., the next descriptor address field is not zero), the 2D-DMA would process the next descriptor directly and reads/writes to the memory according to the descriptor's configuration until all descriptors are processed. When DMA2D_IN_ETM_LOOP_EN_CH*n* or DMA2D_OUT_ETM_LOOP_EN_CH*n* is set, after finishing with the current descriptor, the 2D-DMA would not start processing the next descriptor until receiving the DMA2D_TASK_IN_DSCR_READY_CH*n* or
> DMA2D_TASK_OUT_DSCR_READY_CH*n* task.
> The DMA2D_TASK_IN_DSCR_READY_CH*n* or DMA2D_TASK_OUT_DSCR_READY_CH*n* task can be stored in the task buffer, which can cache at most four tasks. When the 2D-DMA receives this task but the current descriptor is not yet processed, it can cache the task. The depth of the task buffer is determined by DMA2D_IN_DSCR_TASK_MAX_CH*n* or DMA2D_OUT_DSCR_TASK_MAX_CH*n*.
> When the task buffer exceeds its maximum capacity, it will trigger an DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT or DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT interrupt.

2D-DMA can generate the following ETM events:

- DMA2D_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.
- DMA2D_EVT_IN_SUC_EOF_CH*n*: Indicates that the data corresponding to a receive descriptor has been received via channel *n* and the Eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- DMA2D_EVT_OUT_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that the data corresponding to the last transmit descriptors has been sent via transmit channel *n* and the eof bit of this descriptor is 1.

In practical applications, 2D-DMA's ETM events can trigger its own ETM tasks. For example, the DMA2D_EVT_OUT_TOTAL_EOF_CH0 event can trigger the DMA2D_TASK_IN_START_CH1 task, and in this way trigger a new round of 2D-DMA operations.

4.6 Interrupts

ESP32-S31's 2D-DMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- DMA2D_IN_CH0_INTR
- DMA2D_IN_CH1_INTR
- DMA2D_IN_CH2_INTR
- DMA2D_OUT_CH0_INTR
- DMA2D_OUT_CH1_INTR

- DMA2D_OUT_CH2_INTR
- DMA2D_OUT_CH3_INTR

There are several internal interrupt sources from 2D-DMA that can generate the above interrupt signals.

The DMA2D_IN_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- DMA2D_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.
- DMA2D_IN_SUC_EOF_CH*n*_INT: Triggered when the eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received (i.e., when an image corresponding to an inlink has been received) via receive channel *n*.
- DMA2D_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the image received via receive channel *n*. This interrupt is used only for JPEG.
- DMA2D_IN_DSCR_EMPTY_CH*n*_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel *n*.
- DMA2D_IN_DSCR_ERR_CH*n*_INT: Triggered when a receive descriptor on receive channel *n* fails any of the two descriptor checks.
- DMA2D_INFIFO_OVF_CH*n*_INT: Triggered when the RX FIFO of 2D-DMA overflows.
- DMA2D_INFIFO_UDF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the RX FIFO of 2D-DMA underflows.
- DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT: Triggered when the number of cached DMA2D_TASK_IN_DSCR_READY_CH*n* task exceeds the maximum capacity of the task buffer.
- DMA2D_INFIFO_RO_OVF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering in RX direction overflows.
- DMA2D_INFIFO_RO_UDF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering in RX direction underflows.

The DMA2D_OUT_CH*n*_INTR (*n* is 0 ~ 3) signals are generated by the following interrupt sources:

- DMA2D_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- DMA2D_OUT_EOF_CH*n*_INT: Triggered when the eof bit in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If DMA2D_OUT_EOF_MODE_CH*n* is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters 2D-DMA's transmit channel via the bus; if DMA2D_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from 2D-DMA's transmit channel.
- DMA2D_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on transmit channel *n* fails any of the two descriptor checks.
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel *n*.
- DMA2D_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the TX FIFO of 2D-DMA overflows.
- DMA2D_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the TX FIFO of 2D-DMA underflows.

- DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT: Triggered when the number of cached DMA2D_TASK_OUT_DSCR_READY_CH*n* task exceeds the maximum capacity of the task buffer.
- DMA2D_OUTFIFO_RO_OVF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering or color space conversion in TX direction overflows.
- DMA2D_OUTFIFO_RO_UDF_CH0_INT (only channel 0): Triggered when the buffer for macroblock reordering or color space conversion in TX direction underflows.

4.7 Programming Procedures

The clock gating for the 2D-DMA can be configured via HP_SYS_CLKRST_DMA2D_SYS_CLK_EN, and is enabled by default. The 2D-DMA can be reset globally by configuring HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA.

4.7.1 General Configurations for 2D-DMA

Configure the accessible address space of internal and external memory and burst length according to Section 4.4.11.

4.7.2 Mode-Specific Configurations for 2D-DMA

- 1D Mode: No specific configuration required.
- 2D-MOD0 Mode: No specific configuration required.
- 2D-MOD1 Mode: For JPEG, configure DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* or DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* to choose a macroblock size out of the three available options (horizontal × vertical): 8 pixels × 8 pixels, 8 pixels × 16 pixels, and 16 pixels × 16 pixels. For more information about macroblock size, see Chapter 17 *JPEG Codec*. For correspondence between macroblock size, hb, and vb of descriptors, see Table 4.4-4 and Table 4.4-5.
- DSCR-PORT Mode: Set DMA2D_OUT_DSCR_PORT_EN_CH*n* to enter this mode. This mode is specifically designed for PPA. For details, see Chapter 18 *Pixel-Processing Accelerator (PPA)*.

4.7.3 Configurations for 2D-DMA's Transmit Channel

To transmit data, 2D-DMA's transmit channel should be configured by software as follows:

1. Set DMA2D_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's transmit channel and FIFO pointer.
2. Load an outlink, and configure DMA2D_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure DMA2D_OUT_PERI_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 4.4-2.
4. Set DMA2D_OUTLINK_START_CH*n* to enable 2D-DMA's transmit channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.
6. Wait for the DMA2D_OUT_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

4.7.4 Configurations for 2D-DMA's Receive Channel

To receive data, 2D-DMA's receive channel should be configured by software as follows:

1. Set DMA2D_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's receive channel and FIFO pointer.
2. Load an inlink, and configure DMA2D_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure DMA2D_IN_PERI_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 4.4-3.
4. Set DMA2D_INLINK_START_CH*n* to enable 2D-DMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral. See details in individual chapters of these peripherals.
6. Wait for DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, which indicates that an image has been received.

4.7.5 Configurations for Memory-to-Memory Transfer

To transfer data from one memory location to another, 2D-DMA should be configured by software as follows:

1. Set DMA2D_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's transmit channel and FIFO pointer.
2. Set DMA2D_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of 2D-DMA's receive channel and FIFO pointer.
3. Load an outlink, and configure DMA2D_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure DMA2D_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Configure DMA2D_PERI_OUT_SEL_CH*n* with any value corresponding to Dummy-*n* (see Table 4.4-2).
6. Configure DMA2D_PERI_IN_SEL_CH*n* with any value corresponding to Dummy-*n* (see Table 4.4-3).
7. Set DMA2D_IN_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
8. Set DMA2D_OUTLINK_START_CH*n* to enable 2D-DMA's transmit channel for data transfer.
9. Set DMA2D_INLINK_START_CH*n* to enable 2D-DMA's receive channel for data transfer.
10. Wait for DMA2D_IN_SUC_EOF_CH*n*_INT interrupt, which indicates that a data transaction has been completed.

4.7.6 Configurations for Channel Priority and Weight

The priority arbitration can be configured as follows:

1. Configure the channel priority for TX and RX respectively via DMA2D_OUT_ARB_PRIORITY_CH*n* and DMA2D_IN_ARB_PRIORITY_CH*n*.

The weight arbitration can be configured as follows:

1. Configure the time slot for TX and RX respectively via DMA2D_OUT_ARB_TIMEOUT_NUM and DMA2D_IN_ARB_TIMEOUT_NUM.

2. Configure the number of tokens for TX and RX respectively via DMA2D_OUT_ARB_TOKEN_NUM_CH*n* and DMA2D_IN_ARB_TOKEN_NUM_CH*n*.
3. Enable weight arbitration optimization for TX and RX respectively by clearing DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH_CH*n* and DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n*.
4. Enable the arbitration for TX and RX respectively via DMA2D_OUT_WEIGHT_EN and DMA2D_IN_WEIGHT_EN.

4.7.7 Resetting 2D-DMA While it Runs

When the 2D-DMA is running, namely when it is still processing descriptors, it must be stopped first before being reset fully via HP_SYS_CLKRST_RST_EN_DMA2D or partially (only reset AXI functions) via DMA2D_AXIM_RD_RST and DMA2D_AXIM_WR_RST. The procedures are as follows:

1. Set DMA2D_IN_CMD_DISABLE_CH*n* and DMA2D_OUT_CMD_DISABLE_CH*n* to stop the 2D-DMA.
2. Reset the 2D-DMA when DMA2D_OUT_RESET_AVAIL_CH*n* and DMA2D_IN_RESET_AVAIL_CH*n* turn to 1.
3. Clear DMA2D_IN_CMD_DISABLE_CH*n* and DMA2D_OUT_CMD_DISABLE_CH*n* to 0 after the 2D-DMA is reset.

4.8 Register Summary

The addresses in this section are relative to 2D-DMA base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Registers			
DMA2D_OUT_CONF0_CH0_REG	Configures the TX channel 0	0x0000	R/W
DMA2D_OUT_PUSH_CH0_REG	Configures the FIFO of TX channel 0	0x0018	varies
DMA2D_OUT_LINK_CONF_CH0_REG	Configures the transmit descriptor operations of TX channel 0	0x001C	varies
DMA2D_OUT_LINK_ADDR_CH0_REG	Configures the transmit descriptor address of TX channel 0	0x0020	R/W
DMA2D_OUT_ARB_CH0_REG	Configures the arbitration of TX channel 0	0x003C	R/W
DMA2D_OUT_RO_PD_CONF_CH0_REG	Configures the macroblock reordering memory of TX channel 0	0x0044	R/W
DMA2D_OUT_COLOR_CONVERT_CH0_REG	Configures the color space conversion of TX channel 0	0x0048	R/W
DMA2D_OUT_SCRAMBLE_CH0_REG	Configures the byte scrambling of TX channel 0	0x004C	R/W
DMA2D_OUT_COLOR_PARAM0_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0050	R/W
DMA2D_OUT_COLOR_PARAM1_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0054	R/W
DMA2D_OUT_COLOR_PARAM2_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0058	R/W
DMA2D_OUT_COLOR_PARAM3_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x005C	R/W
DMA2D_OUT_COLOR_PARAM4_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0060	R/W
DMA2D_OUT_COLOR_PARAM5_CH0_REG	Configures the color space conversion parameter of TX channel 0	0x0064	R/W
DMA2D_OUT_ETM_CONF_CH0_REG	Configures the ETM of TX channel 0	0x0068	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH0_REG	Configures the block size for TX channel 0 in DSCR-PORT mode	0x006C	R/W
DMA2D_OUT_CONF0_CH1_REG	Configures the TX channel 1	0x0100	R/W
DMA2D_OUT_PUSH_CH1_REG	Configures the FIFO of TX channel 1	0x0118	varies
DMA2D_OUT_LINK_CONF_CH1_REG	Configures the transmit descriptor operations of TX channel 1	0x011C	varies
DMA2D_OUT_LINK_ADDR_CH1_REG	Configures the transmit descriptor address of TX channel 1	0x0120	R/W
DMA2D_OUT_ARB_CH1_REG	Configures the arbitration of TX channel 1	0x013C	R/W
DMA2D_OUT_COLOR_CONVERT_CH1_REG	Configures the color space conversion of TX channel 1	0x0148	R/W
DMA2D_OUT_SCRAMBLE_CH1_REG	Configures the byte scrambling of TX channel 1	0x014C	R/W

Name	Description	Address	Access
DMA2D_OUT_COLOR_PARAM0_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0150	R/W
DMA2D_OUT_COLOR_PARAM1_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0154	R/W
DMA2D_OUT_COLOR_PARAM2_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0158	R/W
DMA2D_OUT_COLOR_PARAM3_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x015C	R/W
DMA2D_OUT_COLOR_PARAM4_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0160	R/W
DMA2D_OUT_COLOR_PARAM5_CH1_REG	Configures the color space conversion parameter of TX channel 1	0x0164	R/W
DMA2D_OUT_ETM_CONF_CH1_REG	Configures the ETM of TX channel 1	0x0168	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH1_REG	Configures the block size for TX channel 1 in DSCR-PORT mode	0x016C	R/W
DMA2D_OUT_CONF0_CH2_REG	Configures the TX channel 2	0x0200	R/W
DMA2D_OUT_PUSH_CH2_REG	Configures the FIFO of TX channel 2	0x0218	varies
DMA2D_OUT_LINK_CONF_CH2_REG	Configures the transmit descriptor operations of TX channel 2	0x021C	varies
DMA2D_OUT_LINK_ADDR_CH2_REG	Configures the transmit descriptor address of TX channel 2	0x0220	R/W
DMA2D_OUT_ARB_CH2_REG	Configures the arbitration of TX channel 2	0x023C	R/W
DMA2D_OUT_COLOR_CONVERT_CH2_REG	Configures the color space conversion of TX channel 2	0x0248	R/W
DMA2D_OUT_SCRAMBLE_CH2_REG	Configures the byte scrambling of TX channel 2	0x024C	R/W
DMA2D_OUT_COLOR_PARAM0_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0250	R/W
DMA2D_OUT_COLOR_PARAM1_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0254	R/W
DMA2D_OUT_COLOR_PARAM2_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0258	R/W
DMA2D_OUT_COLOR_PARAM3_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x025C	R/W
DMA2D_OUT_COLOR_PARAM4_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0260	R/W
DMA2D_OUT_COLOR_PARAM5_CH2_REG	Configures the color space conversion parameter of TX channel 2	0x0264	R/W
DMA2D_OUT_ETM_CONF_CH2_REG	Configures the ETM of TX channel 2	0x0268	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH2_REG	Configures the block size for TX channel 2 in DSCR-PORT mode	0x026C	R/W
DMA2D_OUT_CONF0_CH3_REG	Configures the TX channel 3	0x0300	R/W
DMA2D_OUT_PUSH_CH3_REG	Configures the FIFO of TX channel 3	0x0318	varies
DMA2D_OUT_LINK_CONF_CH3_REG	Configures the transmit descriptor operations of TX channel 3	0x031C	varies
DMA2D_OUT_LINK_ADDR_CH3_REG	Configures the transmit descriptor address of TX channel 3	0x0320	R/W
DMA2D_OUT_ARB_CH3_REG	Configures the arbitration of TX channel 3	0x033C	R/W
DMA2D_OUT_COLOR_CONVERT_CH3_REG	Configures the color space conversion of TX channel 3	0x0348	R/W

GoBack

Name	Description	Address	Access
DMA2D_OUT_SCRAMBLE_CH3_REG	Configures the byte scrambling of TX channel 3	0x034C	R/W
DMA2D_OUT_COLOR_PARAM0_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0350	R/W
DMA2D_OUT_COLOR_PARAM1_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0354	R/W
DMA2D_OUT_COLOR_PARAM2_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0358	R/W
DMA2D_OUT_COLOR_PARAM3_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x035C	R/W
DMA2D_OUT_COLOR_PARAM4_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0360	R/W
DMA2D_OUT_COLOR_PARAM5_CH3_REG	Configures the color space conversion parameter of TX channel 3	0x0364	R/W
DMA2D_OUT_ETM_CONF_CH3_REG	Configures the ETM of TX channel 3	0x0368	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH3_REG	Configures the block size for TX channel 3 in DSCR-PORT mode	0x036C	R/W
DMA2D_IN_CONF0_CH0_REG	Configures the RX channel 0	0x0500	R/W
DMA2D_IN_POP_CH0_REG	Configures the FIFO of RX channel 0	0x0518	varies
DMA2D_IN_LINK_CONF_CH0_REG	Configures the receive descriptor operations of RX channel 0	0x051C	varies
DMA2D_IN_LINK_ADDR_CH0_REG	Configures the receive descriptor address of RX channel 0	0x0520	R/W
DMA2D_IN_ARB_CH0_REG	Configures the arbitration of RX channel 0	0x0540	R/W
DMA2D_IN_RO_PD_CONF_CH0_REG	Configures the macroblock reordering memory of RX channel 0	0x0548	R/W
DMA2D_IN_COLOR_CONVERT_CH0_REG	Configures the color space conversion of RX channel 0	0x054C	R/W
DMA2D_IN_SCRAMBLE_CH0_REG	Configures the byte scrambling of RX channel 0	0x0550	R/W
DMA2D_IN_COLOR_PARAM0_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0554	R/W
DMA2D_IN_COLOR_PARAM1_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0558	R/W
DMA2D_IN_COLOR_PARAM2_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x055C	R/W
DMA2D_IN_COLOR_PARAM3_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0560	R/W
DMA2D_IN_COLOR_PARAM4_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0564	R/W
DMA2D_IN_COLOR_PARAM5_CH0_REG	Configures the color space conversion parameter of RX channel 0	0x0568	R/W
DMA2D_IN_ETM_CONF_CH0_REG	Configures the ETM of RX channel 0	0x056C	R/W
DMA2D_IN_CONF0_CH1_REG	Configures the RX channel 1	0x0600	R/W
DMA2D_IN_POP_CH1_REG	Configures the FIFO of RX channel 1	0x0618	varies
DMA2D_IN_LINK_CONF_CH1_REG	Configures the receive descriptor operations of RX channel 1	0x061C	varies
DMA2D_IN_LINK_ADDR_CH1_REG	Configures the receive descriptor address of RX channel 1	0x0620	R/W
DMA2D_IN_ARB_CH1_REG	Configures the arbitration of RX channel 1	0x0640	R/W

Name	Description	Address	Access
DMA2D_IN_ETM_CONF_CH1_REG	Configures the ETM of RX channel 1	0x0648	R/W
DMA2D_IN_CONF0_CH2_REG	Configures the RX channel 2	0x0700	R/W
DMA2D_IN_POP_CH2_REG	Configures the FIFO of RX channel 2	0x0718	varies
DMA2D_IN_LINK_CONF_CH2_REG	Configures the receive descriptor operations of RX channel 2	0x071C	varies
DMA2D_IN_LINK_ADDR_CH2_REG	Configures the receive descriptor address of RX channel 2	0x0720	R/W
DMA2D_IN_ARB_CH2_REG	Configures the arbitration of RX channel 2	0x0740	R/W
DMA2D_IN_ETM_CONF_CH2_REG	Configures the ETM of RX channel 2	0x0748	R/W
DMA2D_RST_CONF_REG	Configures the reset of AXI	0x0A04	R/W
DMA2D_INTR_MEM_START_ADDR_REG	The start address of accessible internal memory address space	0x0A08	R/W
DMA2D_INTR_MEM_END_ADDR_REG	The end address of accessible internal memory address space	0x0A0C	R/W
DMA2D_EXTR_MEM_START_ADDR_REG	The start address of accessible external memory address space	0x0A10	R/W
DMA2D_EXTR_MEM_END_ADDR_REG	The end address of accessible external memory address space	0x0A14	R/W
DMA2D_OUT_ARB_CONFIG_REG	Configures the arbitration in TX direction	0x0A18	R/W
DMA2D_IN_ARB_CONFIG_REG	Configures the arbitration in RX direction	0x0A1C	R/W
Interrupt Registers			
DMA2D_OUT_INT_RAW_CH0_REG	Raw interrupt status of TX channel 0	0x0004	R/WTC/SS
DMA2D_OUT_INT_ENA_CH0_REG	Interrupt enable bits of TX channel 0	0x0008	R/W
DMA2D_OUT_INT_ST_CH0_REG	Masked interrupt status of TX channel 0	0x000C	RO
DMA2D_OUT_INT_CLR_CH0_REG	Interrupt clear bits of TX channel 0	0x0010	WT
DMA2D_OUT_INT_RAW_CH1_REG	Raw interrupt status of TX channel 1	0x0104	R/WTC/SS
DMA2D_OUT_INT_ENA_CH1_REG	Interrupt enable bits of TX channel 1	0x0108	R/W
DMA2D_OUT_INT_ST_CH1_REG	Masked interrupt status of TX channel 1	0x010C	RO
DMA2D_OUT_INT_CLR_CH1_REG	Interrupt clear bits of TX channel 1	0x0110	WT
DMA2D_OUT_INT_RAW_CH2_REG	Raw interrupt status of TX channel 2	0x0204	R/WTC/SS
DMA2D_OUT_INT_ENA_CH2_REG	Interrupt enable bits of TX channel 2	0x0208	R/W
DMA2D_OUT_INT_ST_CH2_REG	Masked interrupt status of TX channel 2	0x020C	RO
DMA2D_OUT_INT_CLR_CH2_REG	Interrupt clear bits of TX channel 2	0x0210	WT
DMA2D_OUT_INT_RAW_CH3_REG	Raw interrupt status of TX channel 3	0x0304	R/WTC/SS
DMA2D_OUT_INT_ENA_CH3_REG	Interrupt enable bits of TX channel 3	0x0308	R/W

GoBack

Name	Description	Address	Access
DMA2D_OUT_INT_ST_CH3_REG	Masked interrupt status of TX channel 3	0x030C	RO
DMA2D_OUT_INT_CLR_CH3_REG	Interrupt clear bits of TX channel 3	0x0310	WT
DMA2D_IN_INT_RAW_CH0_REG	Raw interrupt status of RX channel 0	0x0504	R/WTC/SS
DMA2D_IN_INT_ENA_CH0_REG	Interrupt enable bits of RX channel 0	0x0508	R/W
DMA2D_IN_INT_ST_CH0_REG	Masked interrupt status of RX channel 0	0x050C	RO
DMA2D_IN_INT_CLR_CH0_REG	Interrupt clear bits of RX channel 0	0x0510	WT
DMA2D_IN_INT_RAW_CH1_REG	Raw interrupt status of RX channel 1	0x0604	R/WTC/SS
DMA2D_IN_INT_ENA_CH1_REG	Interrupt enable bits of RX channel 1	0x0608	R/W
DMA2D_IN_INT_ST_CH1_REG	Masked interrupt status of RX channel 1	0x060C	RO
DMA2D_IN_INT_CLR_CH1_REG	Interrupt clear bits of RX channel 1	0x0610	WT
DMA2D_IN_INT_RAW_CH2_REG	Raw interrupt status of RX channel 2	0x0704	R/WTC/SS
DMA2D_IN_INT_ENA_CH2_REG	Interrupt enable bits of RX channel 2	0x0708	R/W
DMA2D_IN_INT_ST_CH2_REG	Masked interrupt status of RX channel 2	0x070C	RO
DMA2D_IN_INT_CLR_CH2_REG	Interrupt clear bits of RX channel 2	0x0710	WT
Status Registers			
DMA2D_OUTFIFO_STATUS_CH0_REG	Represents the status of the FIFO of TX channel 0	0x0014	RO
DMA2D_OUT_STATE_CH0_REG	Represents the working status of the transmit descriptor of TX channel 0	0x0024	RO
DMA2D_OUT_EOF_DES_ADDR_CH0_REG	Represents the transmit descriptor address when EOF occurs on TX channel 0	0x0028	RO
DMA2D_OUT_DSCR_CH0_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x002C	RO
DMA2D_OUT_DSCR_BF0_CH0_REG	Represents the address of the current pre-read transmit descriptor on TX channel 0	0x0030	RO
DMA2D_OUT_DSCR_BF1_CH0_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 0	0x0034	RO
DMA2D_OUTFIFO_STATUS_CH1_REG	Represents the status of the FIFO of TX channel 1	0x0114	RO
DMA2D_OUT_STATE_CH1_REG	Represents the working status of the transmit descriptor of TX channel 1	0x0124	RO
DMA2D_OUT_EOF_DES_ADDR_CH1_REG	Represents the transmit descriptor address when EOF occurs on TX channel 1	0x0128	RO

Name	Description	Address	Access
DMA2D_OUT_DSCR_CH1_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x012C	RO
DMA2D_OUT_DSCR_BF0_CH1_REG	Represents the address of the current pre-read transmit descriptor on TX channel 1	0x0130	RO
DMA2D_OUT_DSCR_BF1_CH1_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 1	0x0134	RO
DMA2D_OUTFIFO_STATUS_CH2_REG	Represents the status of the FIFO of TX channel 2	0x0214	RO
DMA2D_OUT_STATE_CH2_REG	Represents the working status of the transmit descriptor of TX channel 2	0x0224	RO
DMA2D_OUT_EOF_DES_ADDR_CH2_REG	Represents the transmit descriptor address when EOF occurs on TX channel 2	0x0228	RO
DMA2D_OUT_DSCR_CH2_REG	Represents the address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x022C	RO
DMA2D_OUT_DSCR_BF0_CH2_REG	Represents the address of the current pre-read transmit descriptor on TX channel 2	0x0230	RO
DMA2D_OUT_DSCR_BF1_CH2_REG	Represents the address of the previous pre-read transmit descriptor on TX channel 2	0x0234	RO
DMA2D_INFIFO_STATUS_CH0_REG	Represents the status of the FIFO of RX channel 0	0x0514	RO
DMA2D_IN_STATE_CH0_REG	Represents the working status of the receive descriptor of RX channel 0	0x0524	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH0_REG	Represents the eceive descriptor address when EOF occurs on RX channel 0	0x0528	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH0_REG	Represents the receive descriptor address when errors occur on RX channel 0	0x052C	RO
DMA2D_IN_DSCR_CH0_REG	Represents the address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0530	RO
DMA2D_IN_DSCR_BF0_CH0_REG	Represents address of the current pre-read receive descriptor on RX channel 0	0x0534	RO
DMA2D_IN_DSCR_BF1_CH0_REG	Represents the address of the previous pre-read receive descriptor on RX channel 0	0x0538	RO
DMA2D_INFIFO_STATUS_CH1_REG	Represents the status of the FIFO of RX channel 1	0x0614	RO

Name	Description	Address	Access
DMA2D_IN_STATE_CH1_REG	Represents the working status of the receive descriptor of RX channel 1	0x0624	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH1_REG	Represents the eceive descriptor address when EOF occurs on RX channel 1	0x0628	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH1_REG	Represents the receive descriptor address when errors occur on RX channel 1	0x062C	RO
DMA2D_IN_DSCR_CH1_REG	Represents the address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x0630	RO
DMA2D_IN_DSCR_BF0_CH1_REG	Represents address of the current pre-read receive descriptor on RX channel 1	0x0634	RO
DMA2D_IN_DSCR_BF1_CH1_REG	Represents the address of the previous pre-read receive descriptor on RX channel 1	0x0638	RO
DMA2D_AXI_ERR_REG	Represents the status of the AXI bus	0x0A00	RO
DMA2D_DATE_REG	Version register	0x0A2C	R/W
Peripheral Select Registers			
DMA2D_OUT_PERI_SEL_CH0_REG	Configures the peripheral connected to TX channel 0	0x0038	R/W
DMA2D_OUT_PERI_SEL_CH1_REG	Configures the peripheral connected to TX channel 1	0x0138	R/W
DMA2D_OUT_PERI_SEL_CH2_REG	Configures the peripheral connected to TX channel 2	0x0238	R/W
DMA2D_OUT_PERI_SEL_CH3_REG	Configures the peripheral connected to TX channel 3	0x0338	R/W
DMA2D_IN_PERI_SEL_CH0_REG	Configures the peripheral connected to RX channel 0	0x053C	R/W
DMA2D_IN_PERI_SEL_CH1_REG	Configures the peripheral connected to RX channel 1	0x063C	R/W
DMA2D_IN_PERI_SEL_CH2_REG	Configures the peripheral connected to RX channel 2	0x073C	R/W

4.9 Registers

The addresses in this section are relative to 2D-DMA base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 4.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100n*)**

Bits	Field	Reset
31–27	(reserved)	0 0 0 0 0
26	DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH*n*	0
25	DMA2D_OUT_CMD_DISABLE_CH*n*	0
24	DMA2D_OUT_RST_CH*n*	0
23–17	(reserved)	0 0 0 0 0 0 0
16	DMA2D_OUT_REORDER_EN_CH0	0
15–12	(reserved)	0 0 0 0
11	DMA2D_OUT_DSCR_PORT_EN_CH*n*	0
10–9	DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*	0x0
8–6	DMA2D_OUT_MEM_BURST_LENGTH_CH*n*	0x0
5	(reserved)	0
4	DMA2D_OUT_CHECK_OWNER_CH*n*	0
3	DMA2D_OUT_ECC_AES_EN_CH*n*	0
2	DMA2D_OUTDSCR_BURST_EN_CH*n*	0
1	DMA2D_OUT_EOF_MODE_CH*n*	1
0	DMA2D_OUT_AUTO_WRBACK_CH*n*	0

DMA2D_OUT_AUTO_WRBACK_CH*n* Configures whether to enable automatic outlink write-back when all the data in TX FIFO has been transmitted.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_EOF_MODE_CH*n* Configures when to generate EOF flag.
0: EOF flag for TX channel *n* is generated when data need to read has been pushed into FIFO of 2DDMA
1: EOF flag for TX channel *n* is generated when data need to read has been popped from FIFO of 2DDMA
(R/W)

DMA2D_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* reading descriptors.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_ECC_AES_EN_CH*n* Configures to enable the access to external memory space for ECC and AES (encrypted) via TX channel *n*.
0: Disable
1: Enable
In this case, the starting address of the space and the corresponding data should be 16-byte aligned.
(R/W)

Continued on the next page...

Register 4.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

Continued from the previous page...

DMA2D_OUT_CHECK_OWNER_CH*n* Configures whether to enable the owner bit check for TX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_MEM_BURST_LENGTH_CH*n* Configures the burst length for TX channel *n*.
0: 8 bytes
1: 16 bytes
2: 32 bytes
3: 64 bytes
4: 128 bytes
Others: Invalid
(R/W)

DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configures macroblock size for TX channel *n*. Used only in 2D-MOD1 mode.
0: 8 * 8 pixels
1: 8 * 16 pixels
2: 16 *16 pixels
3: No macroblock
(R/W)

DMA2D_OUT_DSCR_PORT_EN_CH*n* Configures whether to enable the DSCR-PORT mode for TX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_REORDER_EN_CH0 Configures whether to enable macroblock reordering for TX channel 0.
0: Disable
1: Enable
(R/W)

DMA2D_OUT_RST_CH*n* Configures whether to reset TX channel *n*.
0: Reset release
1: Reset
(R/W)

Continued on the next page...

Register 4.1. DMA2D_OUT_CONFO_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

Continued from the previous page...

DMA2D_OUT_CMD_DISABLE_CH*n* Configures reset command for TX channel *n*.

0: Set 0 after reset release

1: Pause transfers before reset

(R/W)

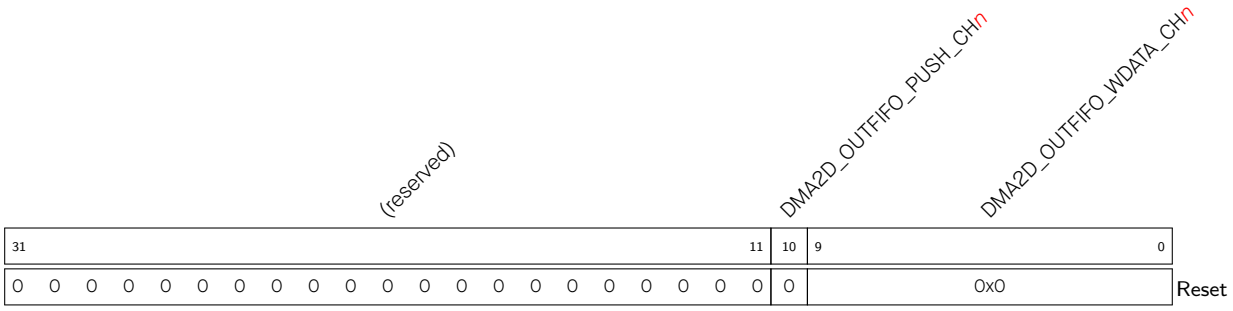
DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH*n* Configures whether to enable weight optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 4.2. DMA2D_OUT_PUSH_CH*n*_REG (*n*: 0-3) (0x0018+0x100**n*)



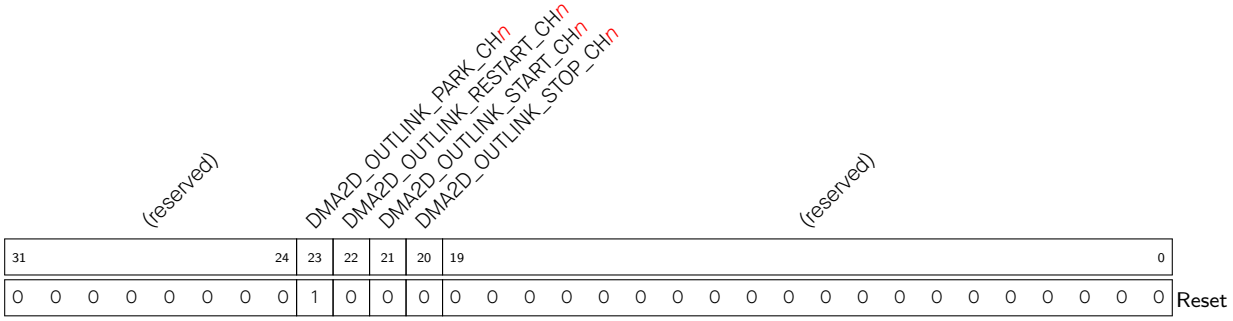
DMA2D_OUTFIFO_WDATA_CH*n* Configures the data to be pushed into the 2D-DMA TX FIFO. (R/W)

DMA2D_OUTFIFO_PUSH_CH*n* Configures whether to push data into the 2D-DMA TX FIFO.

0: Not push 1: Push

(R/W/SC)

Register 4.3. DMA2D_OUT_LINK_CONF_CH*n*_REG (*n*: 0-3) (0x001C+0x100*n)



DMA2D_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(R/W/SC)

DMA2D_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(R/W/SC)

DMA2D_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(R/W/SC)

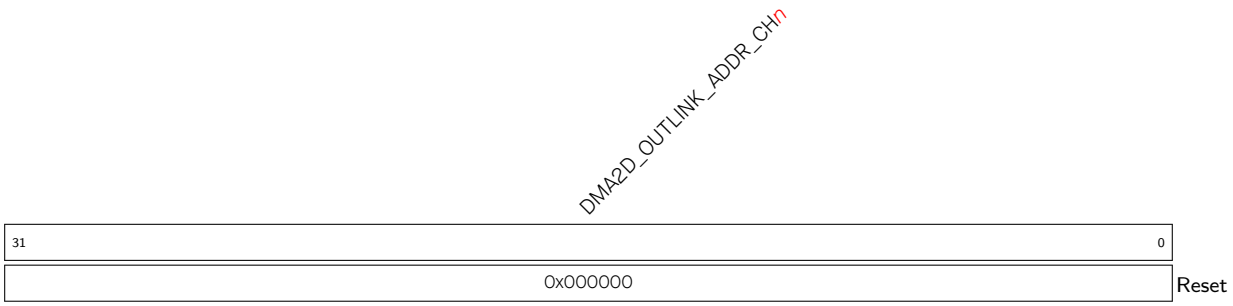
DMA2D_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

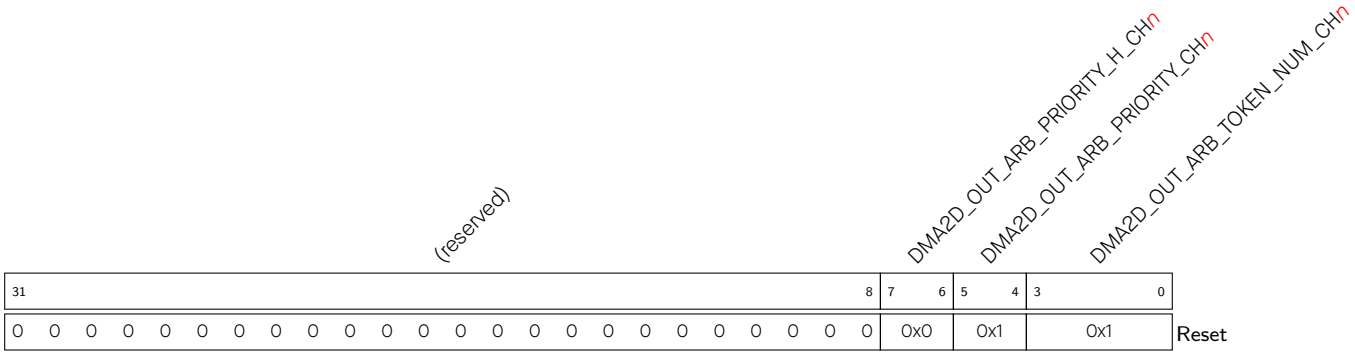
(RO)

Register 4.4. DMA2D_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-3) (0x0020+0x100*n)



DMA2D_OUTLINK_ADDR_CH*n* Represents the first transmit descriptor's address. (R/W)

Register 4.5. DMA2D_OUT_ARB_CH*n*_REG (*n*: 0-3) (0x003C+0x100**n*)



DMA2D_OUT_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

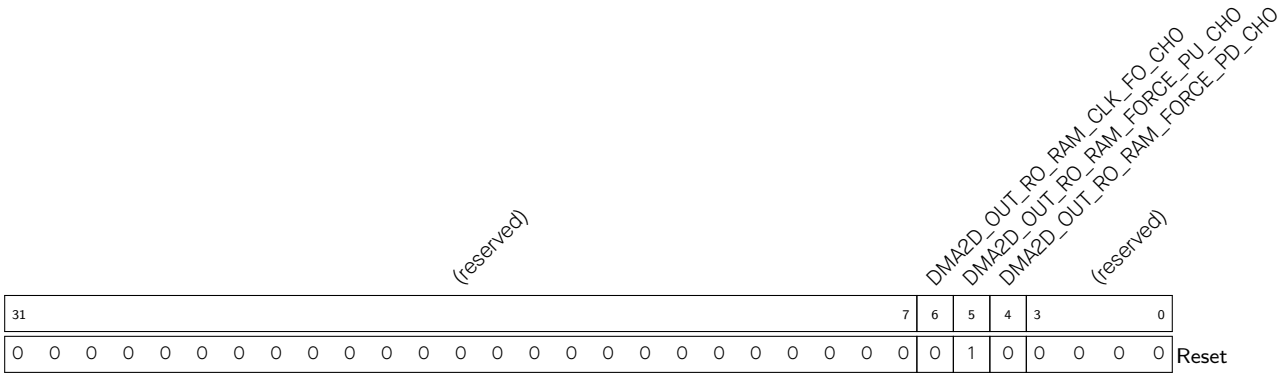
DMA2D_OUT_ARB_PRIORITY_CH*n* Configures the lower two bits of TX channel *n* priority.

Together with DMA2D_OUT_ARB_PRIORITY_H_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

DMA2D_OUT_ARB_PRIORITY_H_CH*n* Configures the higher two bits of TX channel *n* priority.

Together with DMA2D_OUT_ARB_PRIORITY_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

Register 4.6. DMA2D_OUT_RO_PD_CONF_CH0_REG (0x0044)



DMA2D_OUT_RO_RAM_FORCE_PD_CH0 Configures whether to force power down the macroblock reordering RAM for TX channel 0.

0: Not force power down

1: Force power down

(R/W)

DMA2D_OUT_RO_RAM_FORCE_PU_CH0 Configures whether to force power up the macroblock reordering RAM for TX channel 0.

0: Not force power up

1: Force power up

(R/W)

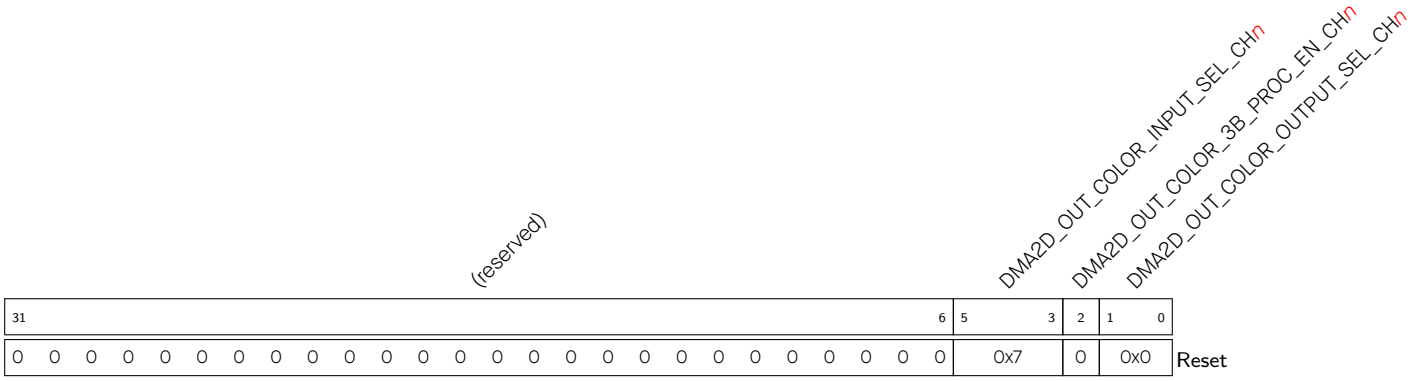
DMA2D_OUT_RO_RAM_CLK_FO_CH0 Configures the clock gating when TX channel 0 accessing the 2D-DMA RAM.

0: Use the clock gate

1: Force the clock on and bypass the clock gate

(R/W)

Register 4.7. DMA2D_OUT_COLOR_CONVERT_CH*n*_REG (*n*: 0-3) (0x0048+0x100n*)**



DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* Configures the third stage output of color space conversion for TX channel *n*.

0: RGB888 to RGB565

1: YUV444 to YUV422

2: Output directly

Others: Invalid

(R/W)

DMA2D_OUT_COLOR_3B_PROC_EN_CH*n* Configures whether to enable the second stage of color space conversion for TX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_OUT_COLOR_INPUT_SEL_CH*n* Configures the first stage output of color space conversion for TX channel *n*.

0: RGB565 to RGB888

1: YUV422 to YUV444

2: Other 2 bytes/pixel color format

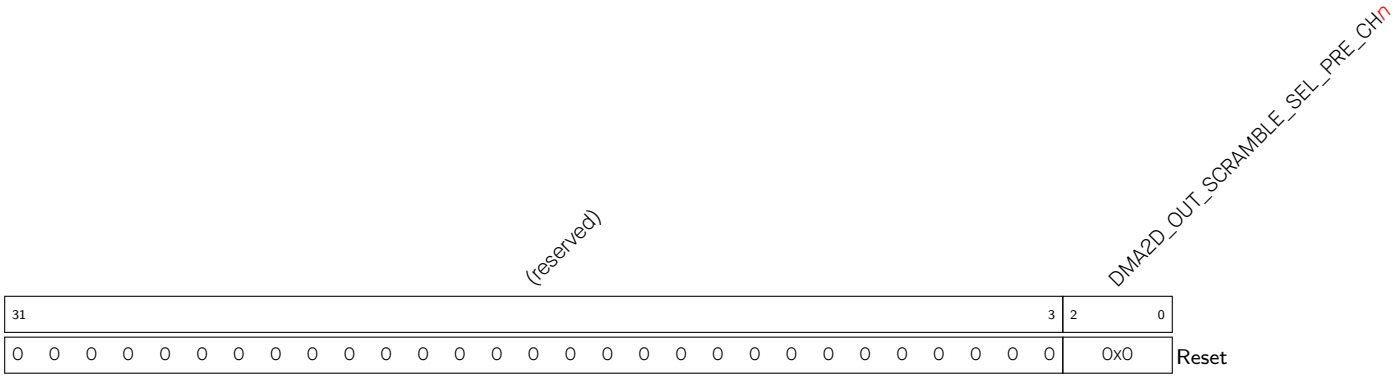
3: Other 3 bytes/pixel color format

7: Disable color space conversion

Others: Invalid

(R/W)

Register 4.8. DMA2D_OUT_SCRAMBLE_CH*n*_REG (*n*: 0-3) (0x004C+0x100**n*)



DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n* Configures the byte order scrambling before the color space conversion for TX channel *n*.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

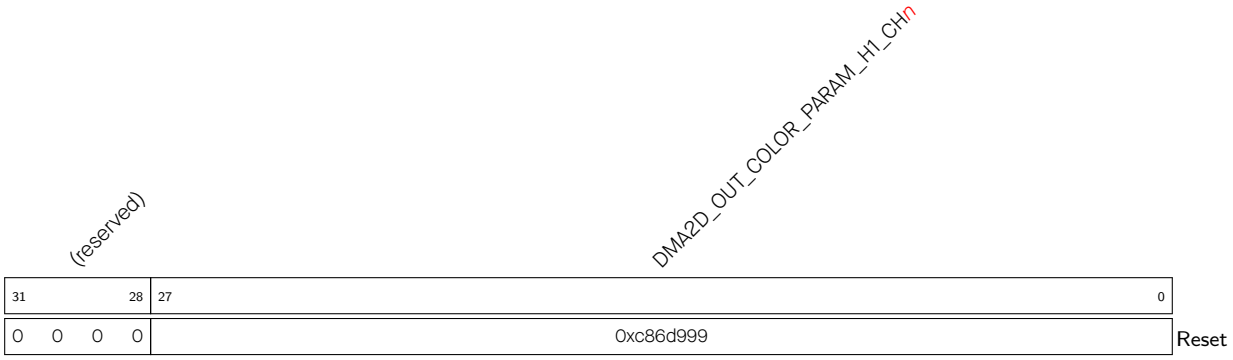
(R/W)

Register 4.9. DMA2D_OUT_COLOR_PARAM0_CH*n*_REG (*n*: 0-3) (0x0050+0x100**n*)



DMA2D_OUT_COLOR_PARAM_H0_CH*n* Configures the coefficient A and B for the most significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.10. DMA2D_OUT_COLOR_PARAM1_CH*n*_REG (*n*: 0-3) (0x0054+0x100**n*)



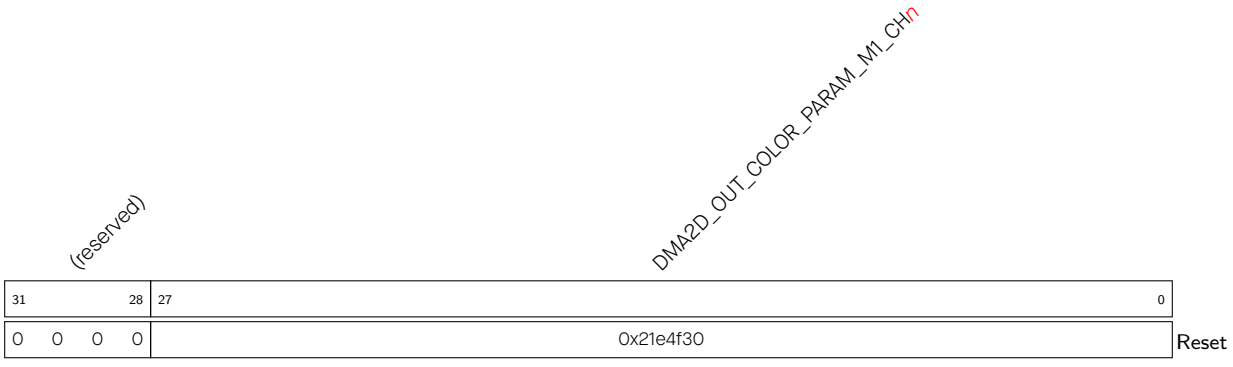
DMA2D_OUT_COLOR_PARAM_H1_CH*n* Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.11. DMA2D_OUT_COLOR_PARAM2_CH*n*_REG (*n*: 0-3) (0x0058+0x100**n*)



DMA2D_OUT_COLOR_PARAM_M0_CH*n* Configures the coefficient A and B for the middle byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.12. DMA2D_OUT_COLOR_PARAM3_CH*n*_REG (*n*: 0-3) (0x005C+0x100**n*)



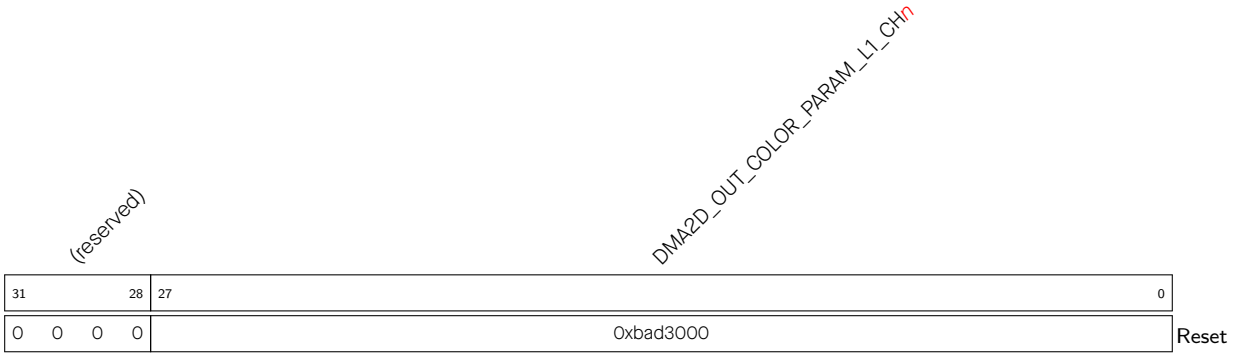
DMA2D_OUT_COLOR_PARAM_M1_CH*n* Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.13. DMA2D_OUT_COLOR_PARAM4_CH*n*_REG (*n*: 0-3) (0x0060+0x100n*)**



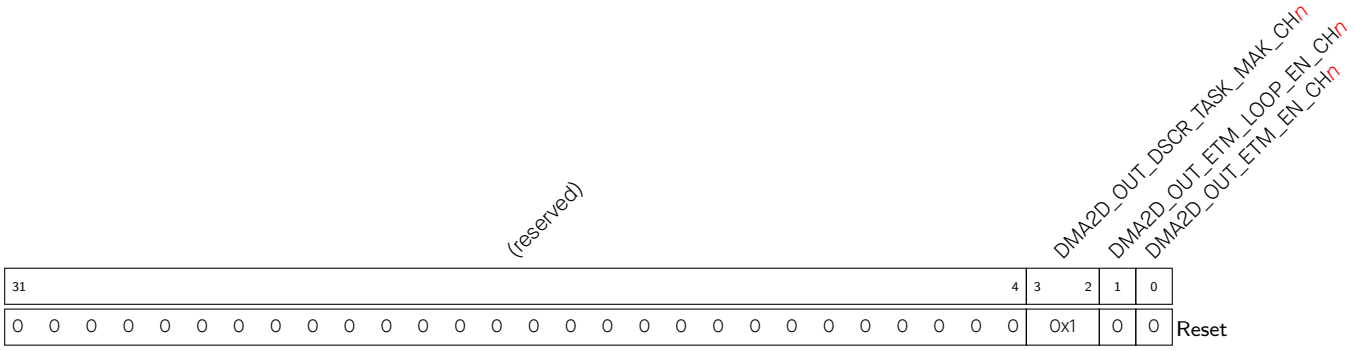
DMA2D_OUT_COLOR_PARAM_LO_CH*n* Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.14. DMA2D_OUT_COLOR_PARAM5_CH*n*_REG (*n*: 0-3) (0x0064+0x100n*)**



DMA2D_OUT_COLOR_PARAM_L1_CH*n* Configures the coefficient C and D for the least significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.15. DMA2D_OUT_ETM_CONF_CH*n*_REG (*n*: 0-3) (0x0068+0x100n*)**



DMA2D_OUT_ETM_EN_CH*n* Configures whether to enable the ETM function for TX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_OUT_ETM_LOOP_EN_CH*n* Configures whether to use the ETM task to indicate the processing of the next descriptor for TX channel *n*.

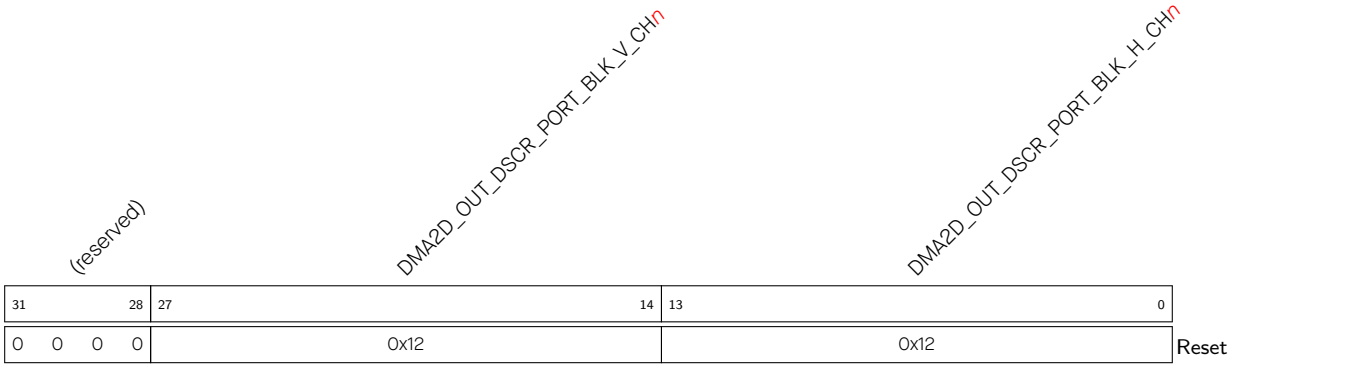
0: Not use ETM task

1: Use ETM task

(R/W)

DMA2D_OUT_DSCR_TASK_MAK_CH*n* Configures the maximum number of tasks that can be cached for TX channel *n*. (R/W)

Register 4.16. DMA2D_OUT_DSCR_PORT_BLK_CH*n*_REG (*n*: 0-3) (0x006C+0x100n*)**



DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* Configures the horizontal width of the basic unit for TX channel *n* in DSCR-PORT mode.

Measurement unit: pixels. (R/W)

DMA2D_OUT_DSCR_PORT_BLK_V_CH*n* Configures the vertical height of the basic unit for TX channel *n* in DSCR-PORT mode.

Measurement unit: pixels. (R/W)

Register 4.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100n*)**

Bits	Field	Reset
31–27	(reserved)	0 0 0 0 0
26	DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n*	0
25	DMA2D_IN_CMD_DISABLE_CH*n*	0
24	DMA2D_IN_RST_CH*n*	0
23–17	(reserved)	0 0 0 0 0 0 0
16	DMA2D_IN_REORDER_EN_CH0	0
15–12	(reserved)	0 0 0 0
11	DMA2D_IN_DSCR_PORT_EN_CH*n*	0
10–9	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*	0x0
8–6	DMA2D_IN_MEM_BURST_LENGTH_CH*n*	0x0
5	reserved	0
4	DMA2D_IN_CHECK_OWNER_CH*n*	0
3	DMA2D_IN_ECC_AES_EN_CH*n*	0
2	DMA2D_INDSCR_BURST_EN_CH*n*	0
1	(reserved)	0
0	DMA2D_IN_MEM_TRANS_EN_CH*n*	0

DMA2D_IN_MEM_TRANS_EN_CH*n* Configures whether to enable memory-to-memory data transfer for TX and RX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read descriptors when accessing internal memory.
0: Disable
1: Enable
(R/W)

DMA2D_IN_ECC_AES_EN_CH*n* Configures whether to enable the access to external memory space for ECC and AES (encrypted) via RX channel *n*.
0: Disable
1: Enable. In this case, the starting address of the space and the corresponding data should be 16-byte aligned.
(R/W)

DMA2D_IN_CHECK_OWNER_CH*n* Configures whether to enable the owner bit check for RX channel *n*.
0: Disable
1: Enable
(R/W)

DMA2D_IN_MEM_BURST_LENGTH_CH*n* Configures the burst length for RX channel *n*.
0: 8 bytes
1: 16 bytes
2: 32 bytes
3: 64 bytes
4: 128 bytes
Others: Invalid
(R/W)

Continued on the next page...

Register 4.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100**n*)

Continued from the previous page...

DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configures macroblock size for RX channel *n*. Used only in 2D-MOD1 mode.
0: 8 * 8 pixels
1: 8 * 16 pixels
2: 16 * 16 pixels
3: No macroblock
(R/W)

DMA2D_IN_DSCR_PORT_EN_CH*n* Configures whether to enable the DSCR-PORT mode for RX channel *n*.
0: Disable
1: Enable
(R/W)

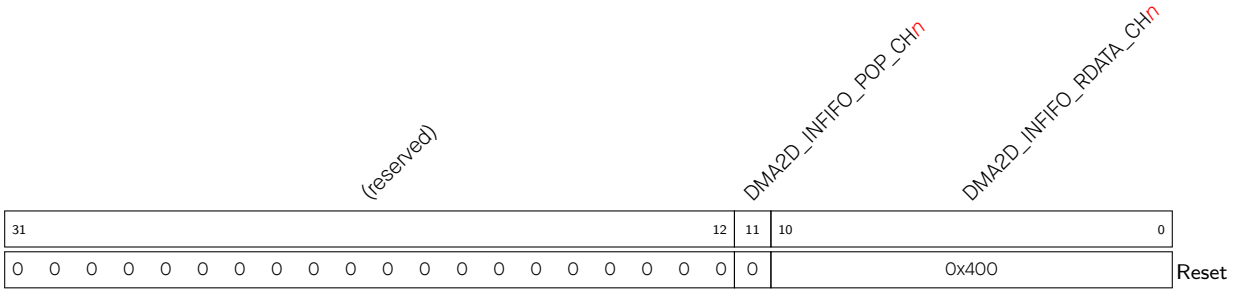
DMA2D_IN_REORDER_EN_CH0 Configures whether to enable macroblock reordering for RX channel 0.
0: Disable
1: Enable
(R/W)

DMA2D_IN_RST_CH*n* Configures whether to reset RX channel *n*.
0: Invalid
1: Reset
(R/W)

DMA2D_IN_CMD_DISABLE_CH*n* Configures reset command for RX channel *n*.
0: Set 0 after reset release
1: Pause transfers before reset
(R/W)

DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n* Configures whether to enable weight optimization for RX channel *n*.
0: Enable
1: Disable
(R/W)

Register 4.18. DMA2D_IN_POP_CH*n*_REG (*n*: 0-2) (0x0518+0x100n*)**



DMA2D_INFIFO_RDATA_CH*n* Represents the data popped from 2D-DMA RX FIFO. (RO)

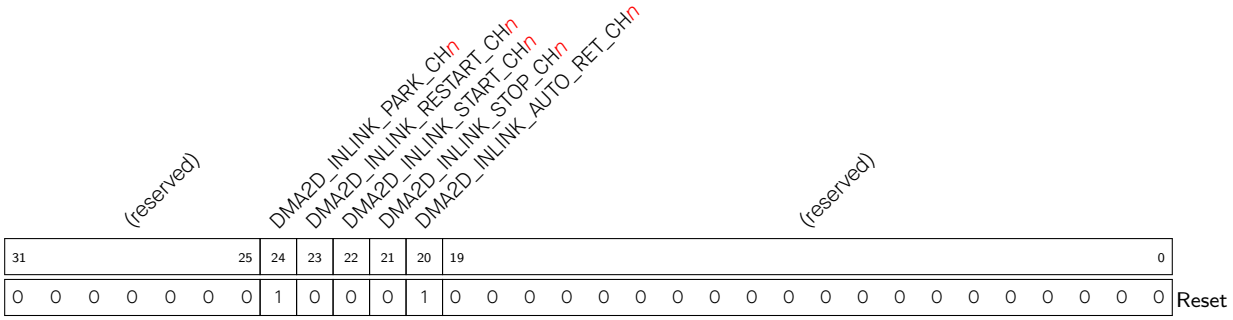
DMA2D_INFIFO_POP_CH*n* Configures whether to pop data from 2D-DMA RX FIFO.

0: Not pop

1: Pop

(R/W/SC)

Register 4.19. DMA2D_IN_LINK_CONF_CH*n*_REG (*n*: 0-2) (0x051C+0x100n*)**



DMA2D_INLINK_AUTO_RET_CH*n* Configure the value of the owner bit written back to the receive descriptor.

0: Write back 0

1: Write back 1

(R/W)

DMA2D_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(R/W/SC)

DMA2D_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(R/W/SC)

DMA2D_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for 2D-DMA transfer.

0: Invalid. No effect

1: Restart

(R/W/SC)

DMA2D_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

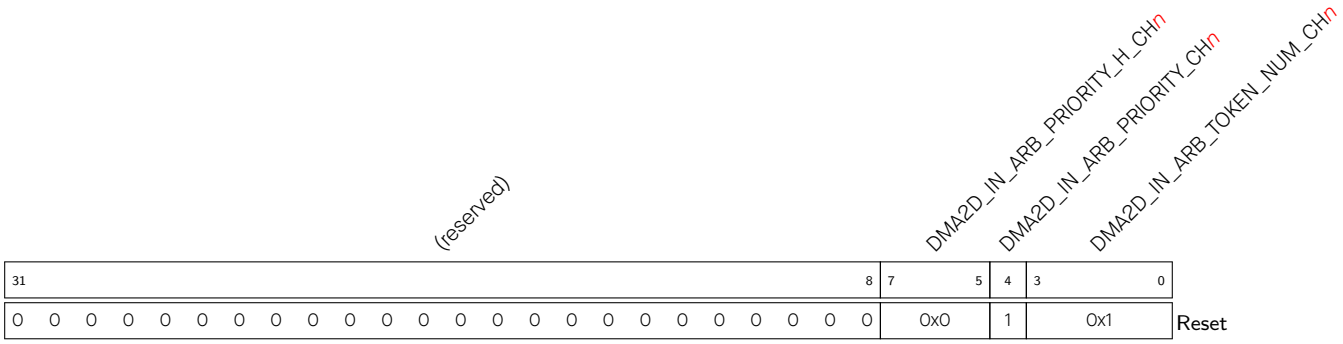
(RO)

Register 4.20. DMA2D_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x0520+0x100n*)**



DMA2D_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 4.21. DMA2D_IN_ARB_CH*n*_REG (*n*: 0-2) (0x0540+0x100n*)**



DMA2D_IN_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

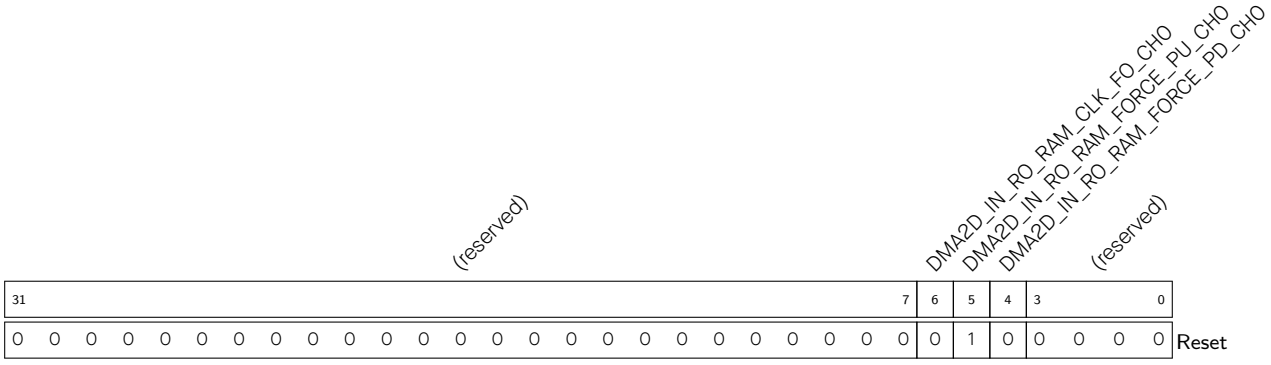
DMA2D_IN_ARB_PRIORITY_CH*n* Configures the lower two bits of RX channel *n* priority.

Together with DMA2D_IN_ARB_PRIORITY_H_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

DMA2D_IN_ARB_PRIORITY_H_CH*n* Configures the higher two bits of TX channel *n* priority.

Together with DMA2D_IN_ARB_PRIORITY_CH*n*, this field forms a 4-bit priority value for TX channel *n*, ranging from 0 (lowest) to 15 (highest). (R/W)

Register 4.22. DMA2D_IN_RO_PD_CONF_CH0_REG (0x0548)



DMA2D_IN_RO_RAM_FORCE_PD_CH0 Configures whether to force power down the macroblock reordering RAM for RX channel 0.

0: Not force power down

1: Force power down

(R/W)

DMA2D_IN_RO_RAM_FORCE_PU_CH0 Configures whether to force power up the macroblock reordering RAM for RX channel 0.

0: Not force power up

1: Force power up

(R/W)

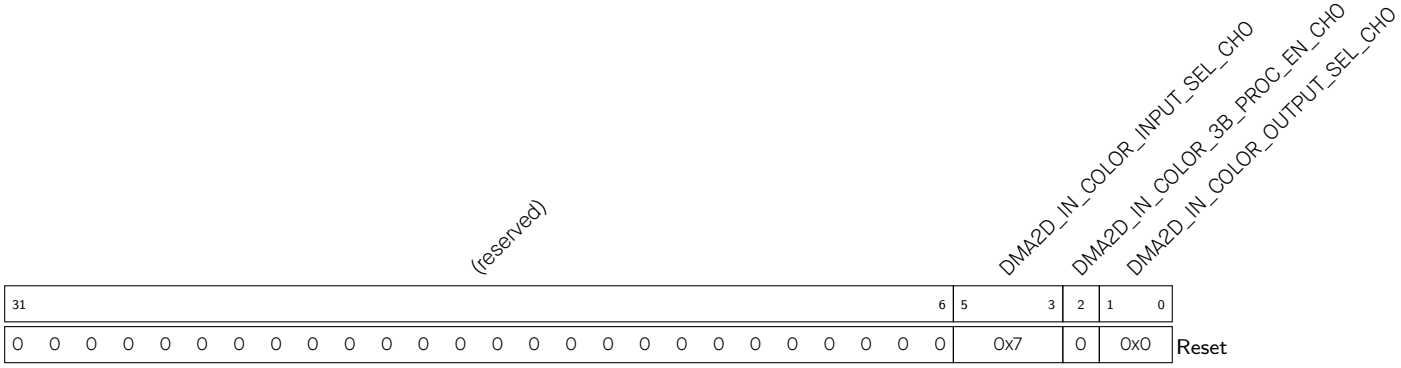
DMA2D_IN_RO_RAM_CLK_FO_CH0 Configures the clock gating when RX channel 0 accessing the 2D-DMA RAM.

0: Use the clock gate

1: Force the clock on and bypass the clock gate

(R/W)

Register 4.23. DMA2D_IN_COLOR_CONVERT_CH0_REG (0x054C)



DMA2D_IN_COLOR_OUTPUT_SEL_CH0 Configures the third stage output of color space conversion for RX channel 0.

0: RGB888 to RGB565

1: Output directly

(R/W)

DMA2D_IN_COLOR_3B_PROC_EN_CH0 Configures whether to enable the second stage of color space conversion for RX channel 0.

0: Disable

1: Enable

(R/W)

DMA2D_IN_COLOR_INPUT_SEL_CH0 Configures the first stage output of color space conversion for RX channel 0.

0: YUV422/420 to YUV444

1: YUV422

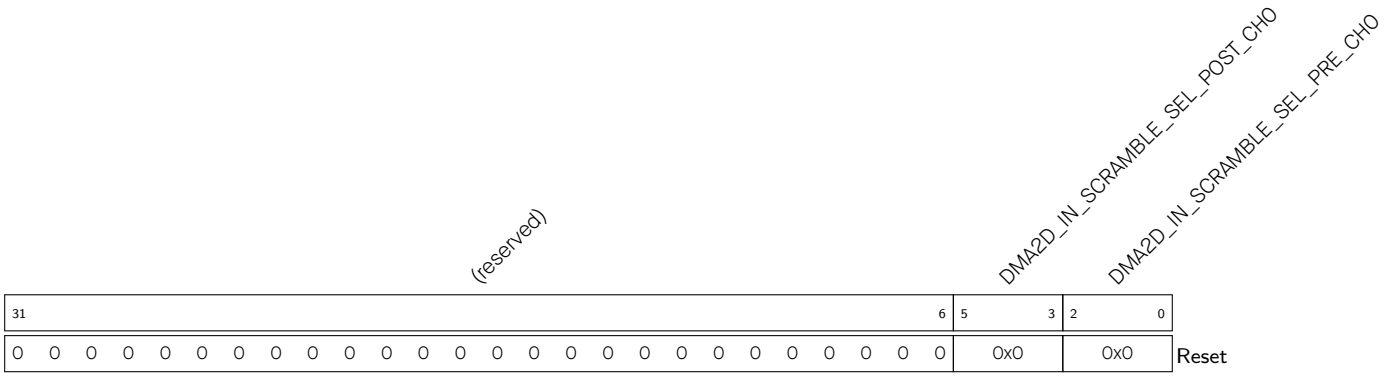
2: YUV444/420

7: Disable color space conversion

Others: Invalid

(R/W)

Register 4.24. DMA2D_IN_SCRAMBLE_CH0_REG (0x0550)



DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 Configures the byte order scrambling before the color space conversion for RX channel 0.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

(R/W)

DMA2D_IN_SCRAMBLE_SEL_POST_CH0 Configures the byte order scrambling after the color space conversion for RX channel 0.

0: BYTE2-1-0

1: BYTE2-0-1

2: BYTE1-0-2

3: BYTE1-2-0

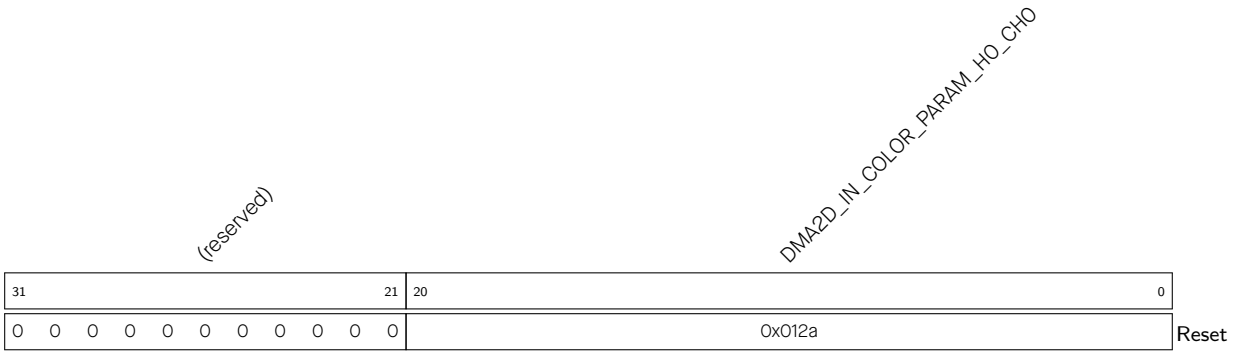
4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

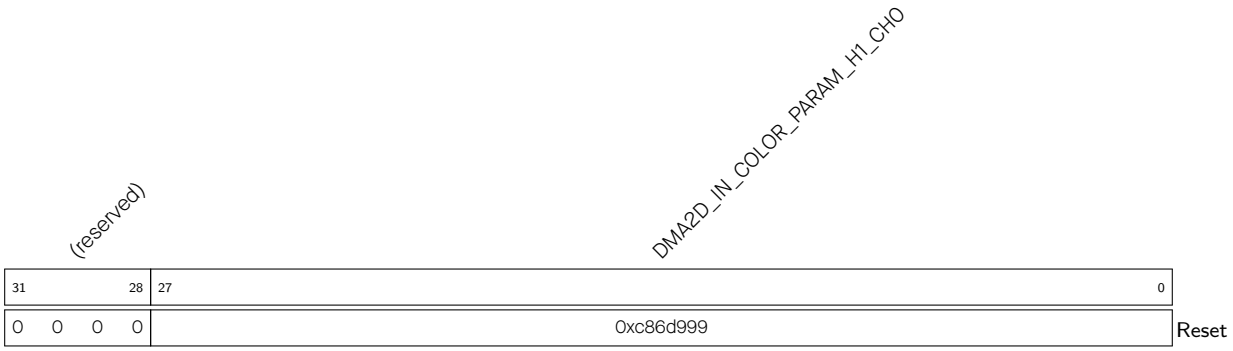
(R/W)

Register 4.25. DMA2D_IN_COLOR_PARAM0_CH0_REG (0x0554)



DMA2D_IN_COLOR_PARAM_H0_CH0 Configures the coefficient A and B for the most significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.26. DMA2D_IN_COLOR_PARAM1_CH0_REG (0x0558)



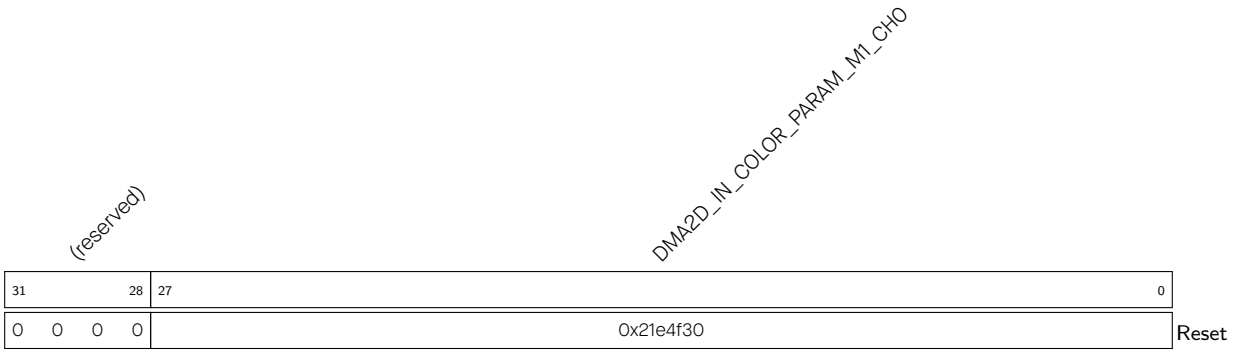
DMA2D_IN_COLOR_PARAM_H1_CH0 Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.27. DMA2D_IN_COLOR_PARAM2_CH0_REG (0x055C)



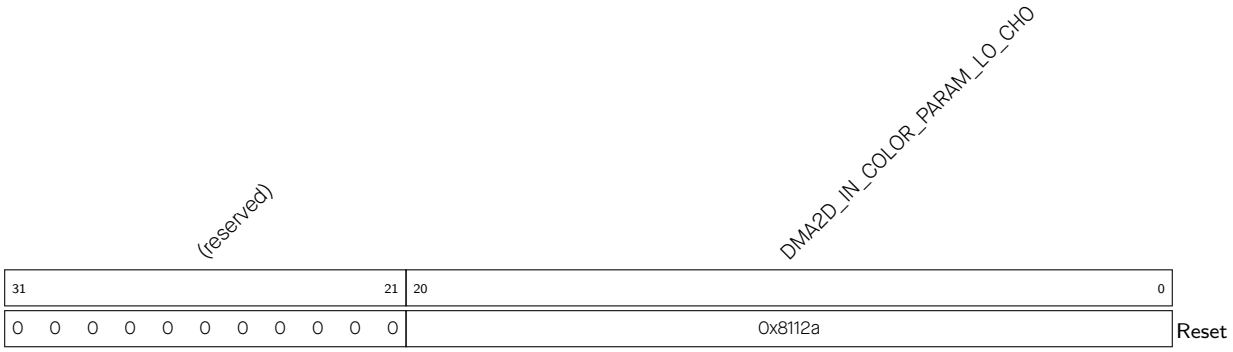
DMA2D_IN_COLOR_PARAM_M0_CH0 Configures the coefficient A and B for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.28. DMA2D_IN_COLOR_PARAM3_CH0_REG (0x0560)



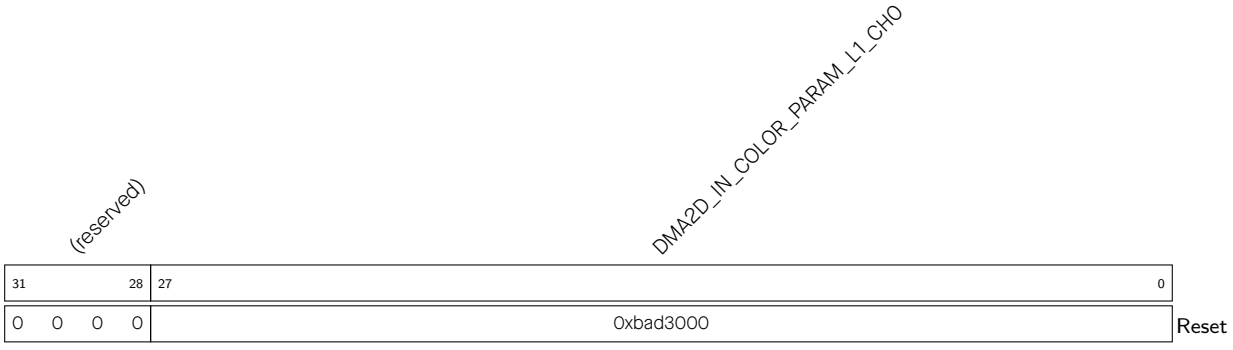
DMA2D_IN_COLOR_PARAM_M1_CH0 Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.29. DMA2D_IN_COLOR_PARAM4_CH0_REG (0x0564)



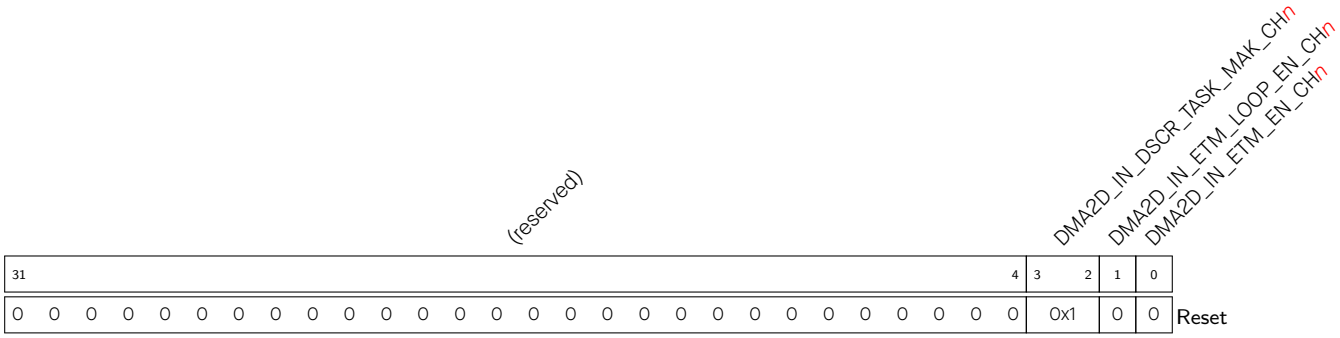
DMA2D_IN_COLOR_PARAM_L0_CH0 Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.30. DMA2D_IN_COLOR_PARAM5_CH0_REG (0x0568)



DMA2D_IN_COLOR_PARAM_L1_CH0 Configures the coefficient C and D for the least significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.31. DMA2D_IN_ETM_CONF_CH*n*_REG (*n*: 0-2) (0x056C+0x100n*)**



DMA2D_IN_ETM_EN_CH*n* Configures whether to enable the ETM function for RX channel *n*.

0: Disable

1: Enable

(R/W)

DMA2D_IN_ETM_LOOP_EN_CH*n* Configures whether to use the ETM task to indicate the processing of the next descriptor for RX channel *n*.

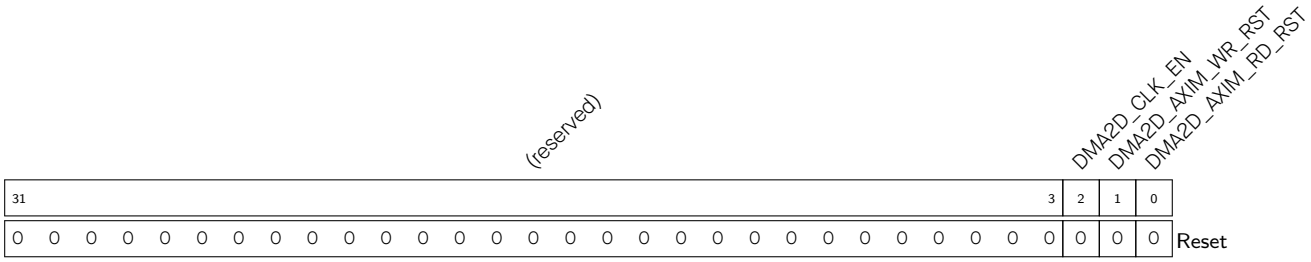
0: Not use ETM task

1: Use ETM task

(R/W)

DMA2D_IN_DSCR_TASK_MAK_CH*n* Configures the maximum number of tasks that can be cached for RX channel *n*. (R/W)

Register 4.32. DMA2D_RST_CONF_REG (0x0A04)



DMA2D_AXIM_RD_RST Configures whether to reset the AXI master read data FIFO.

0: No effect

1: Reset

(R/W)

DMA2D_AXIM_WR_RST Configures whether to reset the AXI master write data FIFO.

0: No effect

1: Reset

(R/W)

DMA2D_CLK_EN Configures the clock for registers.

0: Support clock only when application writes registers

1: Force clock on for register

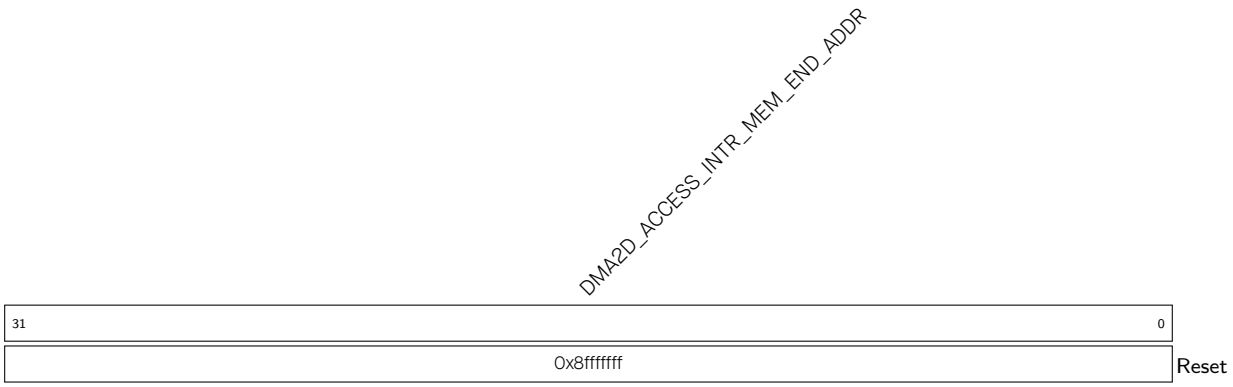
(R/W)

Register 4.33. DMA2D_INTR_MEM_START_ADDR_REG (0x0A08)



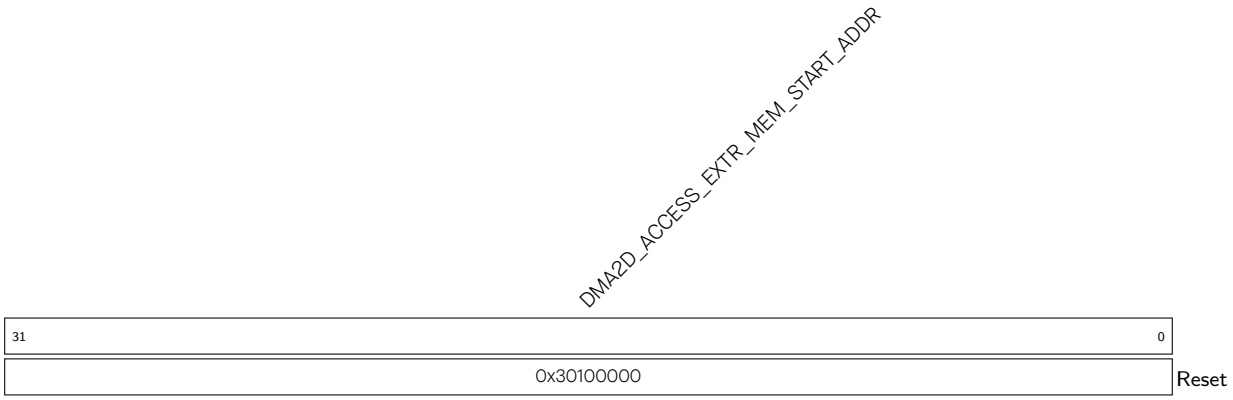
DMA2D_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible internal address space. (R/W)

Register 4.34. DMA2D_INTR_MEM_END_ADDR_REG (0x0A0C)



DMA2D_ACCESS_INTR_MEM_END_ADDR Configures the end address of the accessible internal address space. Accessing an address beyond this range will result in a descriptor error. (R/W)

Register 4.35. DMA2D_EXTR_MEM_START_ADDR_REG (0x0A10)



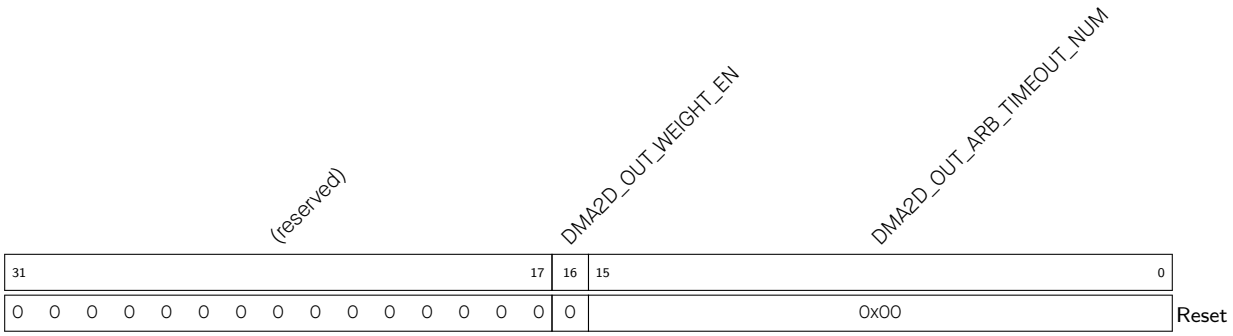
DMA2D_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible external address space. (R/W)

Register 4.36. DMA2D_EXTR_MEM_END_ADDR_REG (0x0A14)



DMA2D_ACCESS_EXTR_MEM_END_ADDR Configures the end address of the accessible external address space. Accessing an address beyond this range will result in a descriptor error. (R/W)

Register 4.37. DMA2D_OUT_ARB_CONFIG_REG (0x0A18)



DMA2D_OUT_ARB_TIMEOUT_NUM Configures the time slot for TX. Measurement unit: AXI bus clock cycle. (R/W)

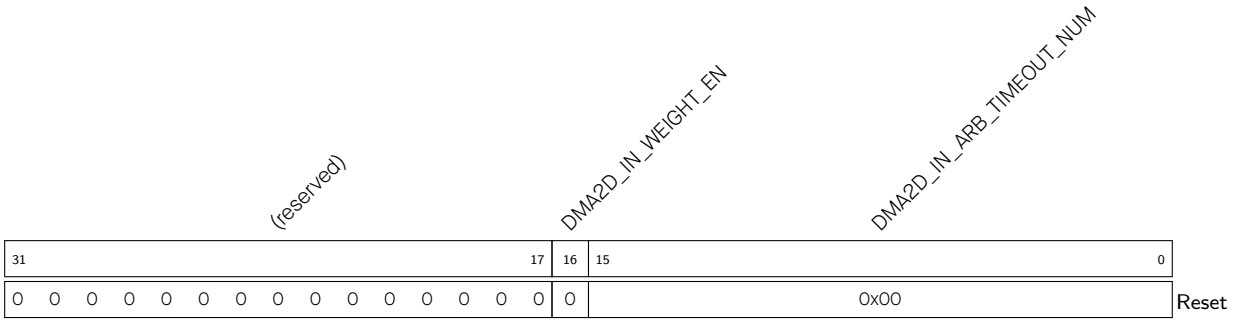
DMA2D_OUT_WEIGHT_EN Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

Register 4.38. DMA2D_IN_ARB_CONFIG_REG (0x0A1C)



DMA2D_IN_ARB_TIMEOUT_NUM Configures the time slot for RX. Measurement unit: AXI bus clock cycle. (R/W)

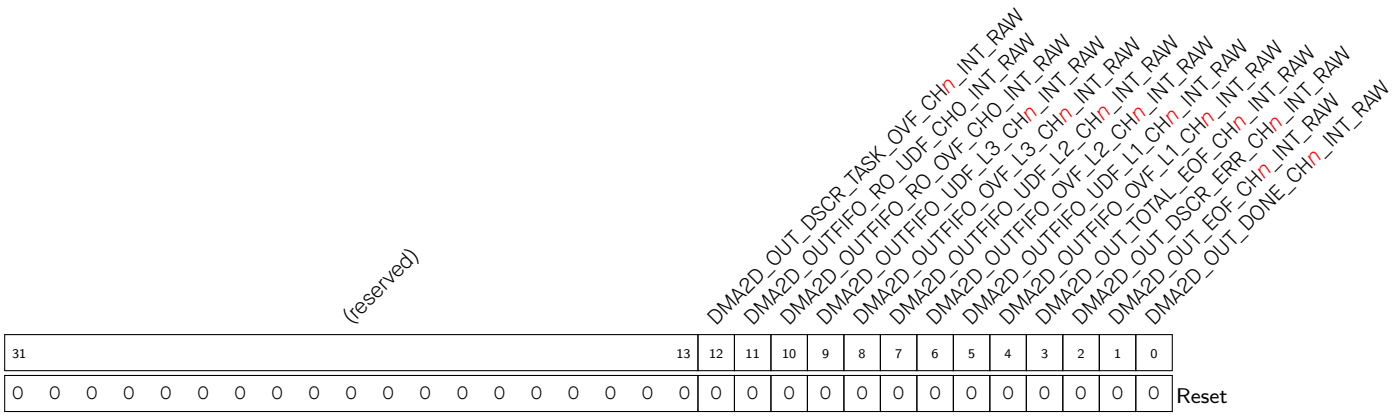
DMA2D_IN_WEIGHT_EN Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

(R/W)

Register 4.39. DMA2D_OUT_INT_RAW_CH*n*_REG (*n*: 0-3) (0x0004+0x100n*)**



DMA2D_OUT_DONE_CH*n*_INT_RAW The raw interrupt status bit of DMA2D_OUT_DONE_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (R/WTC/SS)

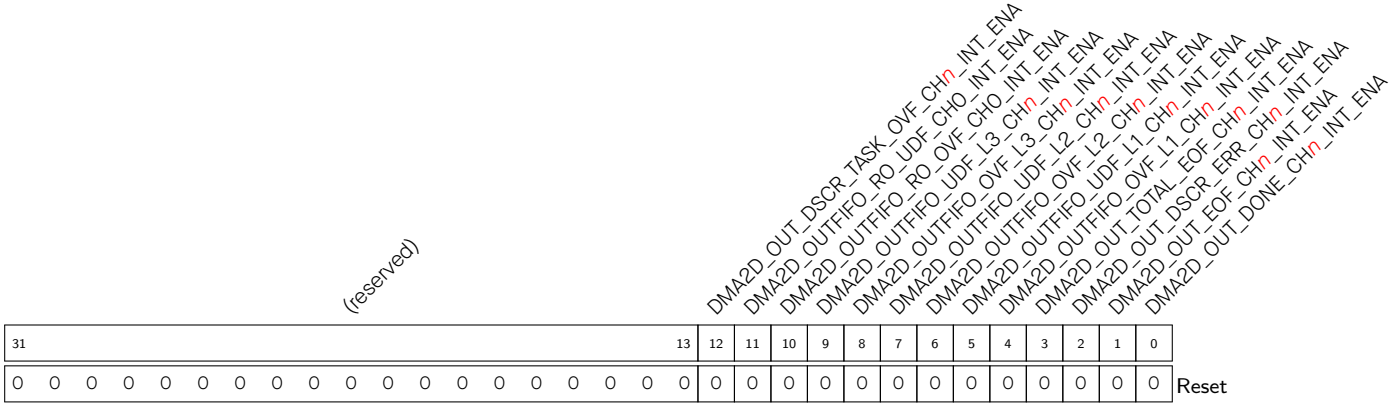
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_OVF_CHO_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_UDF_CHO_INT. (R/WTC/SS)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (R/WTC/SS)

Register 4.40. DMA2D_OUT_INT_ENA_CH*n*_REG (*n*: 0-3) (0x0008+0x100**n*)



DMA2D_OUT_DONE_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DONE_CH*n*_INT. (R/W)

DMA2D_OUT_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_EOF_CH*n*_INT. (R/W)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DSCR_ERR_CH*n*_INT. (R/W)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (R/W)

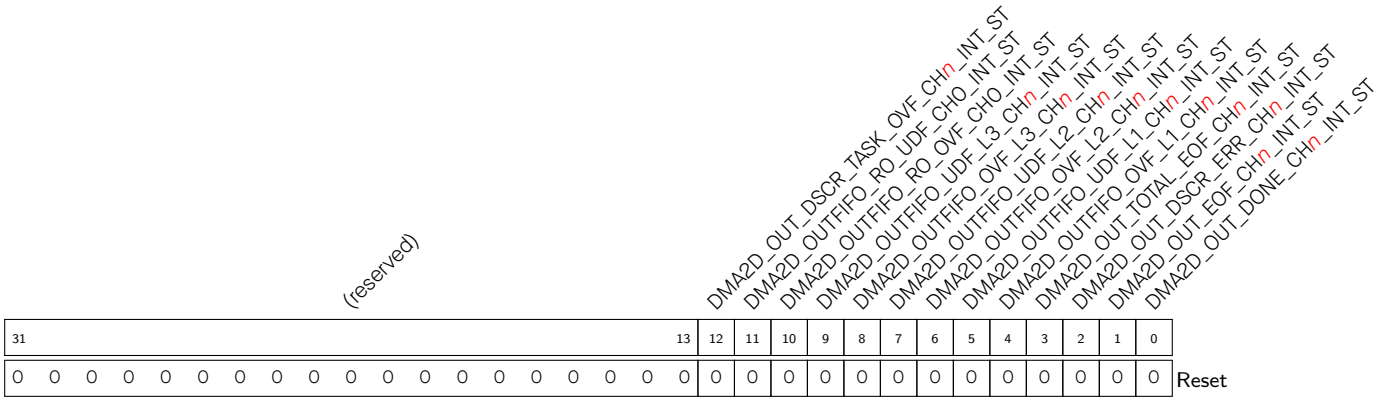
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/W)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_RO_OVF_CHO_INT. (R/W)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_RO_UDF_CHO_INT. (R/W)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (R/W)

Register 4.41. DMA2D_OUT_INT_ST_CH*n*_REG (*n*: 0-3) (0x000C+0x100n*)**



DMA2D_OUT_DONE_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DONE_CH*n*_INT. (RO)

DMA2D_OUT_EOF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_EOF_CH*n*_INT. (RO)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DSCR_ERR_CH*n*_INT. (RO)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (RO)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (RO)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (RO)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (RO)

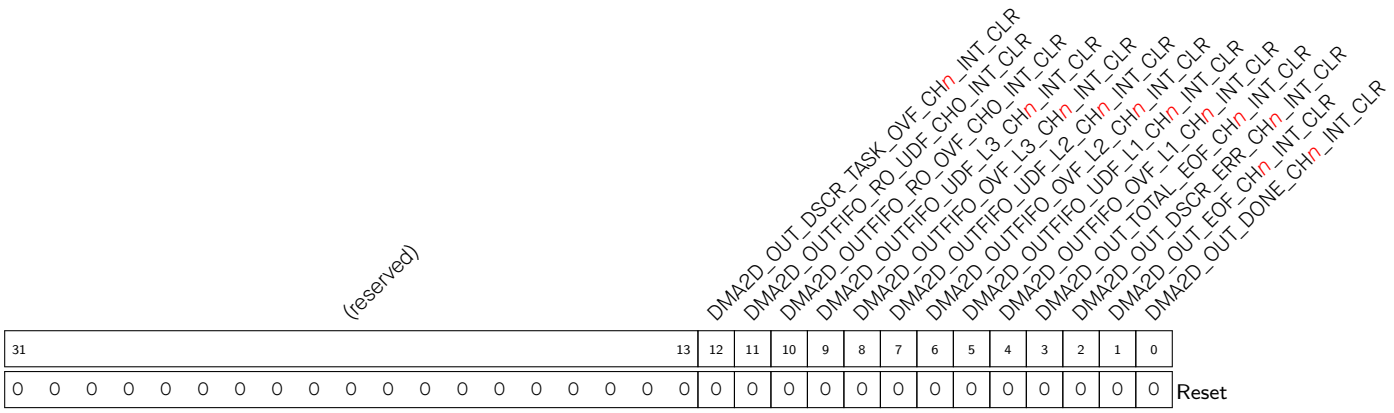
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (RO)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_OVF_CHO_INT. (RO)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_UDF_CHO_INT. (RO)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ST The masked interrupt status of DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (RO)

Register 4.42. DMA2D_OUT_INT_CLR_CH*n*_REG (*n*: 0-3) (0x0010+0x100**n*)



DMA2D_OUT_DONE_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DONE_CH*n*_INT. (WT)

DMA2D_OUT_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_EOF_CH*n*_INT. (WT)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DSCR_ERR_CH*n*_INT. (WT)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_TOTAL_EOF_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L1_CH*n*_INT. (WT)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L1_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L2_CH*n*_INT. (WT)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L2_CH*n*_INT. (WT)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_OVF_L3_CH*n*_INT. (WT)

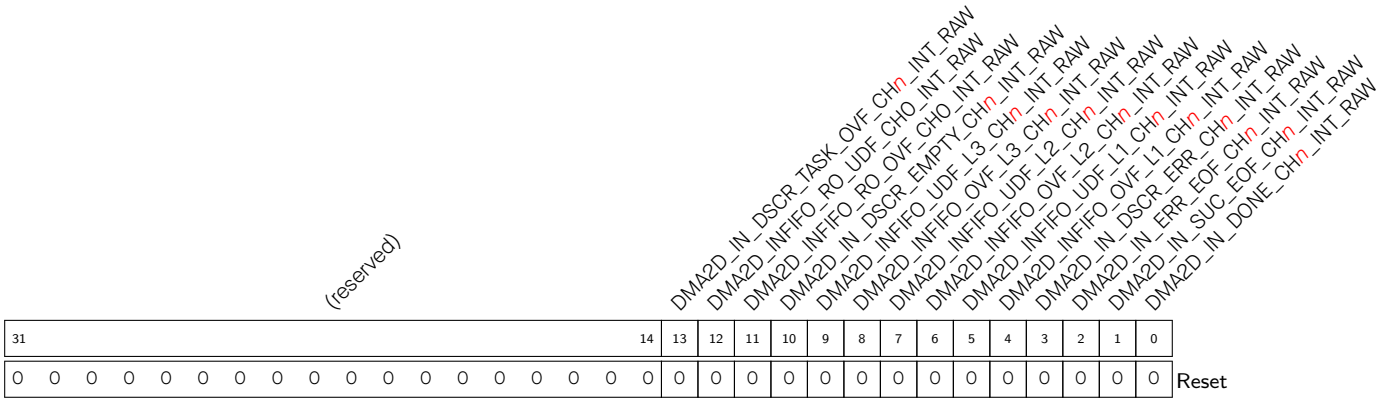
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (WT)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_RO_OVF_CHO_INT. (WT)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_RO_UDF_CHO_INT. (WT)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT. (WT)

Register 4.43. DMA2D_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0504+0x100n*)**



DMA2D_IN_DONE_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DONE_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_SUC_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_ERR_EOF_CH*n*_INT. (R/WTC/SS)

DMA2D_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L1_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L2_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_OVF_L3_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_INFIFO_UDF_L3_CH*n*_INT. (R/WTC/SS)

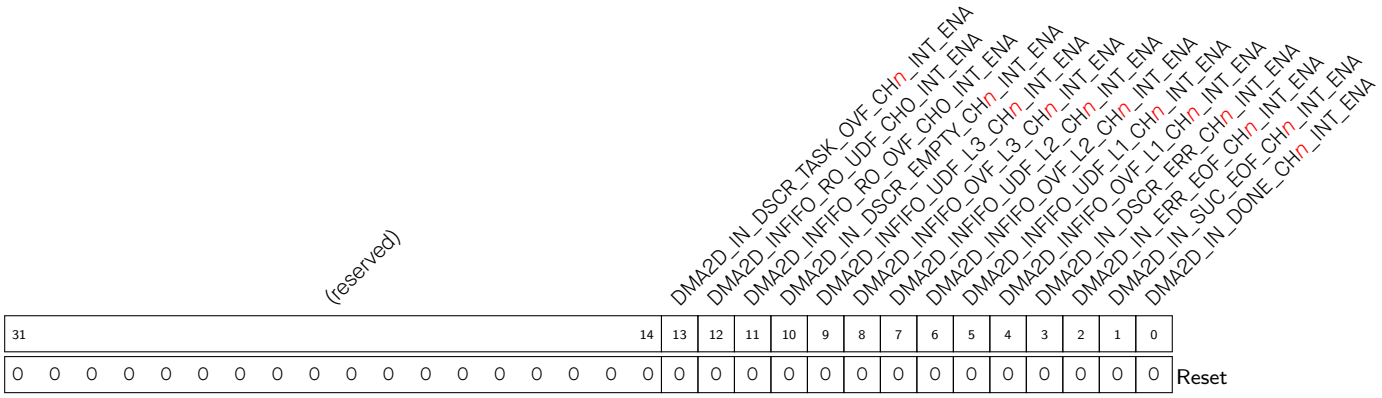
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_OVF_CHO_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_OVF_CHO_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_UDF_CHO_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_UDF_CHO_INT. (R/WTC/SS)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (R/WTC/SS)

Register 4.44. DMA2D_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0508+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DONE_CH*n*_INT. (R/W)

DMA2D_IN_SUC_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_SUC_EOF_CH*n*_INT. (R/W)

DMA2D_IN_ERR_EOF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_ERR_EOF_CH*n*_INT. (R/W)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_ERR_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L1_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L1_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L2_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L2_CH*n*_INT. (R/W)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_OVF_L3_CH*n*_INT. (R/W)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_INFIFO_UDF_L3_CH*n*_INT. (R/W)

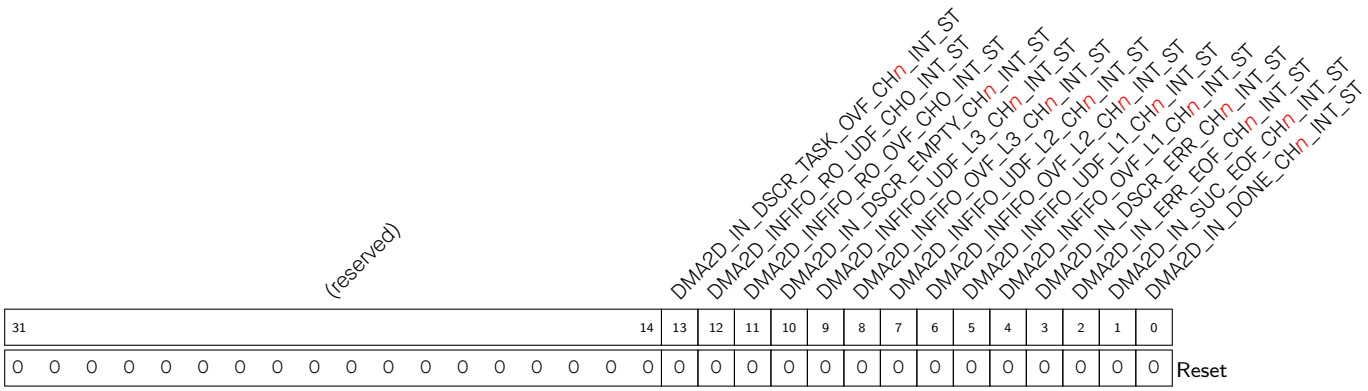
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

DMA2D_INFIFO_RO_OVF_CH0_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_OVF_CH0_INT. (R/W)

DMA2D_INFIFO_RO_UDF_CH0_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_UDF_CH0_INT. (R/W)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (R/W)

Register 4.45. DMA2D_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x050C+0x100*n)



DMA2D_IN_DONE_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DONE_CH*n*_INT. (RO)

DMA2D_IN_SUC_EOF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_SUC_EOF_CH*n*_INT. (RO)

DMA2D_IN_ERR_EOF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_ERR_EOF_CH*n*_INT. (RO)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_ERR_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L1_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L1_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L2_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L2_CH*n*_INT. (RO)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_OVF_L3_CH*n*_INT. (RO)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ST The masked interrupt status of the DMA2D_INFIFO_UDF_L3_CH*n*_INT. (RO)

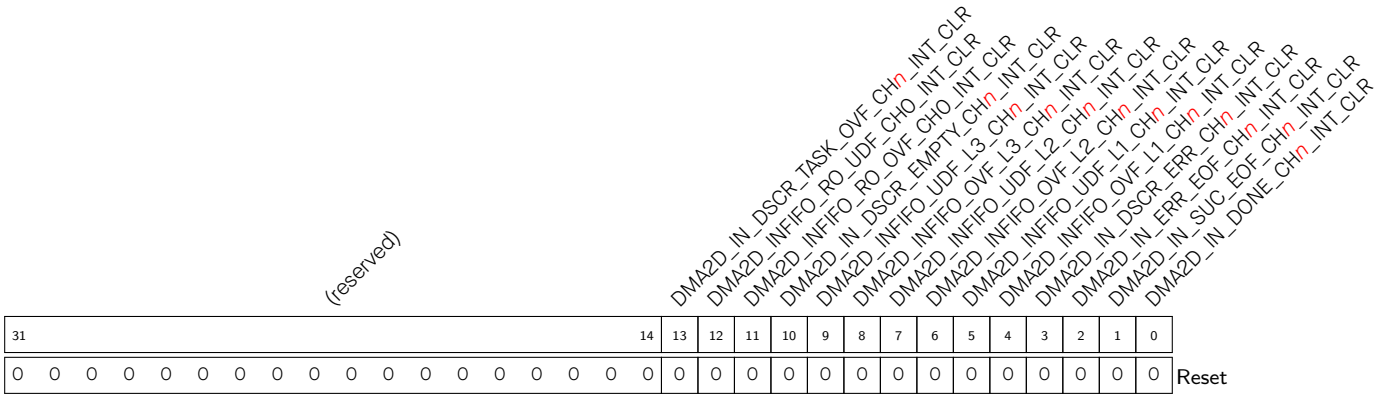
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (RO)

DMA2D_INFIFO_RO_OVF_CHO_INT_ST The masked interrupt status of the DMA2D_INFIFO_RO_OVF_CHO_INT. (RO)

DMA2D_INFIFO_RO_UDF_CHO_INT_ST The masked interrupt status of the DMA2D_INFIFO_RO_UDF_CHO_INT. (RO)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (RO)

Register 4.46. DMA2D_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x0510+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_DONE_CH*n*_INT. (WT)

DMA2D_IN_SUC_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_SUC_EOF_CH*n*_INT. (WT)

DMA2D_IN_ERR_EOF_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_ERR_EOF_CH*n*_INT. (WT)

DMA2D_IN_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_ERR_CH*n*_INT. (WT)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L1_CH*n*_INT. (WT)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L1_CH*n*_INT. (WT)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L2_CH*n*_INT. (WT)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L2_CH*n*_INT. (WT)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L3_CH*n*_INT. (WT)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L3_CH*n*_INT. (WT)

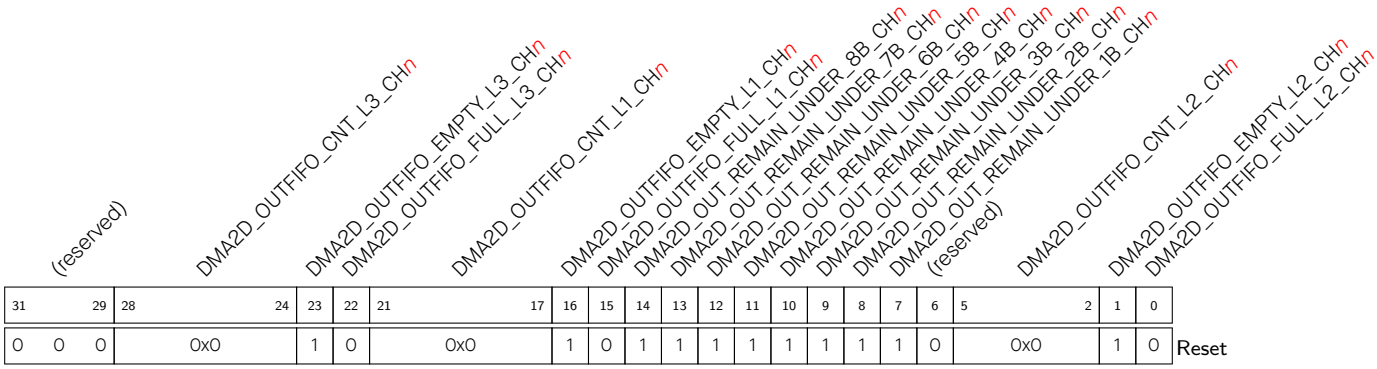
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (WT)

DMA2D_INFIFO_RO_OVF_CH0_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_OVF_CH0_INT. (WT)

DMA2D_INFIFO_RO_UDF_CH0_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_UDF_CH0_INT. (WT)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT. (WT)

Register 4.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100n*)**



DMA2D_OUTFIFO_FULL_L2_CH*n* Represents whether the L2 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L2_CH*n* Represents whether the L2 TX FIFO for TX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L2_CH*n* Represents the number of data bytes in L2 TX FIFO for TX channel *n*. (RO)

DMA2D_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

DMA2D_OUT_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

DMA2D_OUTFIFO_FULL_L1_CH*n* Represents whether the L1 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L1_CH*n* Represents whether the L1 TX FIFO for TX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L1_CH*n* Represents the number of data bytes in L1 TX FIFO for TX channel *n*. (RO)

Continued on the next page...

Register 4.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100**n*)

Continued from the previous page...

DMA2D_OUTFIFO_FULL_L3_CH*n* Represents whether the L3 TX FIFO for TX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_OUTFIFO_EMPTY_L3_CH*n* Represents whether the L3 TX FIFO for TX channel *n* is empty.

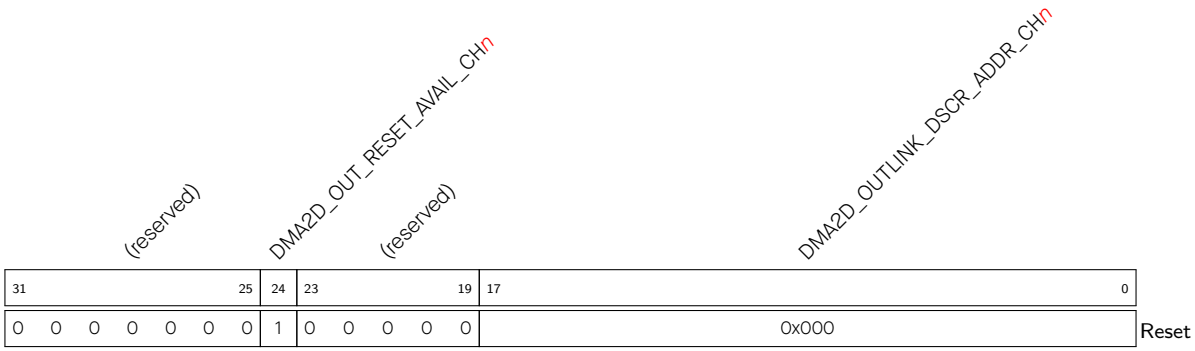
0: Not empty

1: Empty

(RO)

DMA2D_OUTFIFO_CNT_L3_CH*n* Represents the number of data bytes in L3 TX FIFO for TX channel *n*. (RO)

Register 4.48. DMA2D_OUT_STATE_CH*n*_REG (*n*: 0-3) (0x0024+0x100**n*)



DMA2D_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

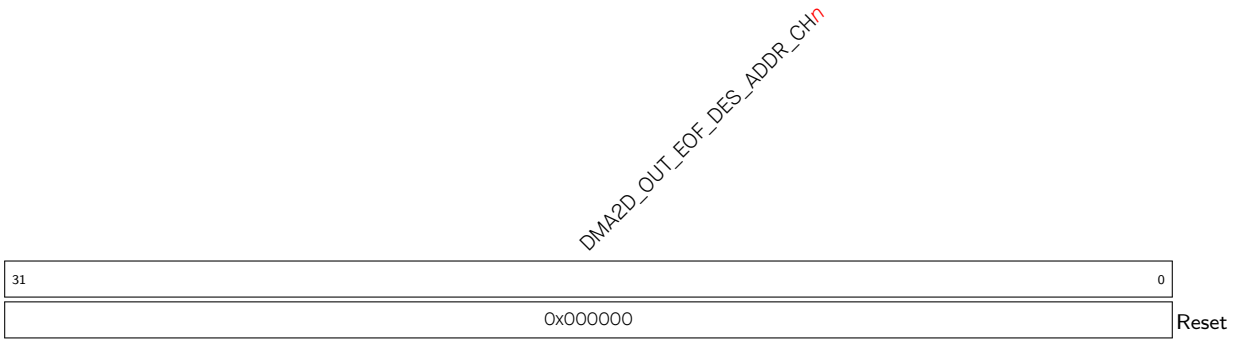
DMA2D_OUT_RESET_AVAIL_CH*n* Represents whether it is safe to reset TX channel *n*.

0: Not safe

1: Safe

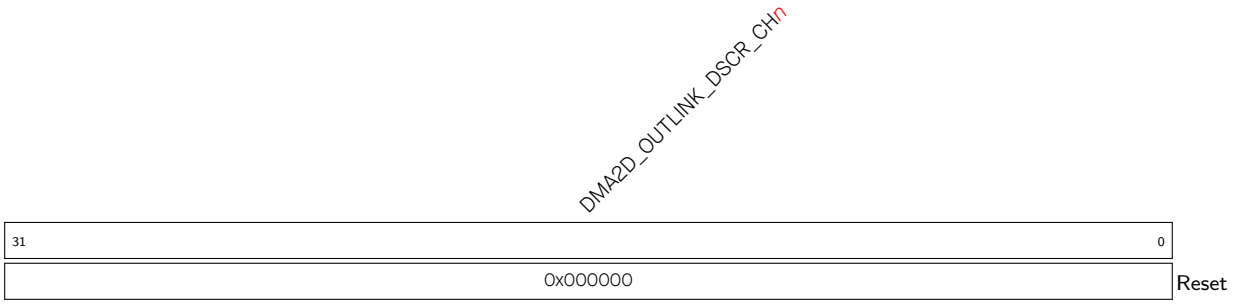
(RO)

Register 4.49. DMA2D_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-3) (0x0028+0x100**n*)



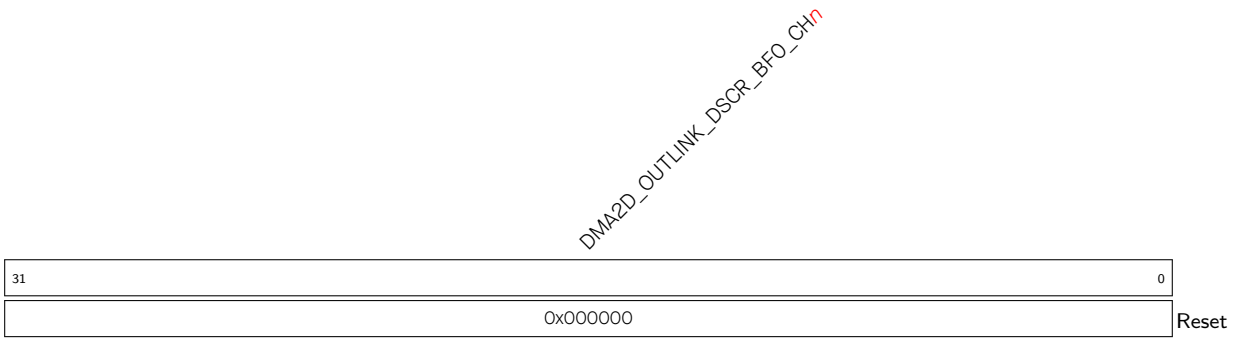
DMA2D_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the eof bit in this descriptor is 1. (RO)

Register 4.50. DMA2D_OUT_DSCR_CH*n*_REG (*n*: 0-3) (0x002C+0x100**n*)



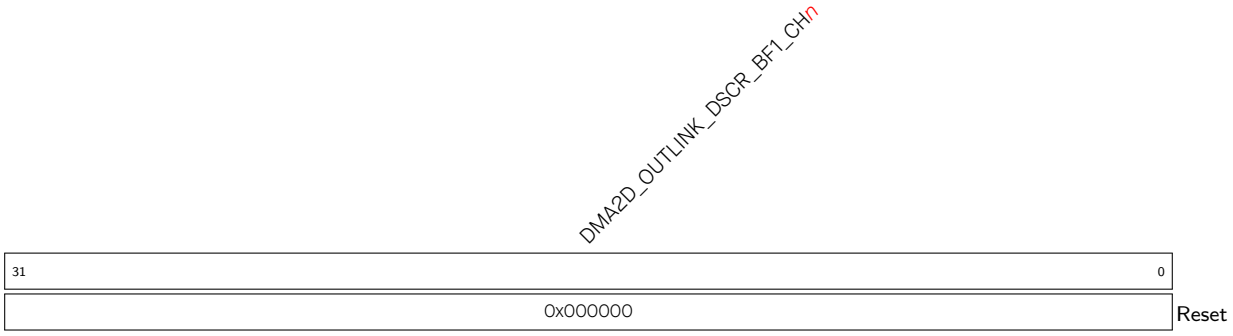
DMA2D_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor pointed by the current transmit descriptor that is pre-read. (RO)

Register 4.51. DMA2D_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-3) (0x0030+0x100**n*)



DMA2D_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor that is pre-read. (RO)

Register 4.52. DMA2D_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-3) (0x0034+0x100n*)**



DMA2D_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor that is pre-read. (RO)

Register 4.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100n*)**



DMA2D_INFIFO_FULL_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L2_CH*n* Represents the number of data bytes in the L2 RX FIFO for RX channel *n*. (RO)

DMA2D_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

DMA2D_INFIFO_FULL_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L1_CH*n* Represents the number of data bytes in the L1 RX FIFO for RX channel *n*. (RO)

Continued on the next page...

Register 4.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100**n*)

Continued from the previous page...

DMA2D_INFIFO_FULL_L3_CH*n* Represents whether the L3 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L3_CH*n* Represents whether the L3 RX FIFO for RX channel *n* is empty.

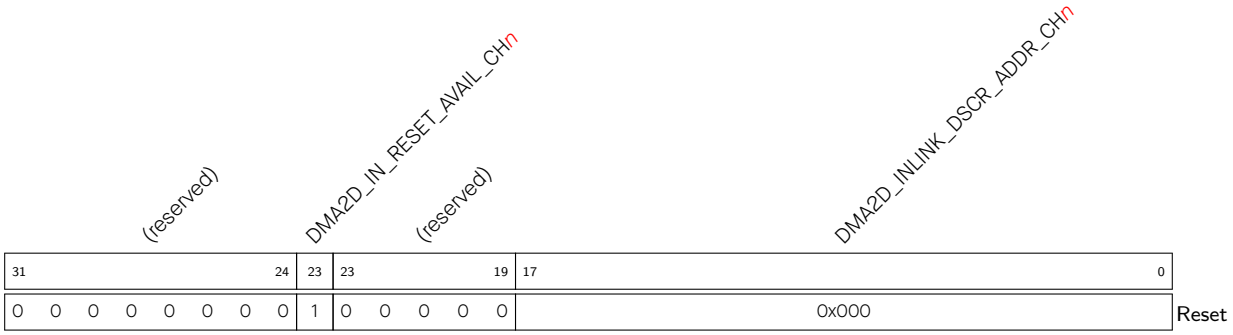
0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L3_CH*n* Represents the number of data bytes in the L3 RX FIFO for RX channel *n*. (RO)

Register 4.54. DMA2D_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0524+0x100**n*)



DMA2D_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

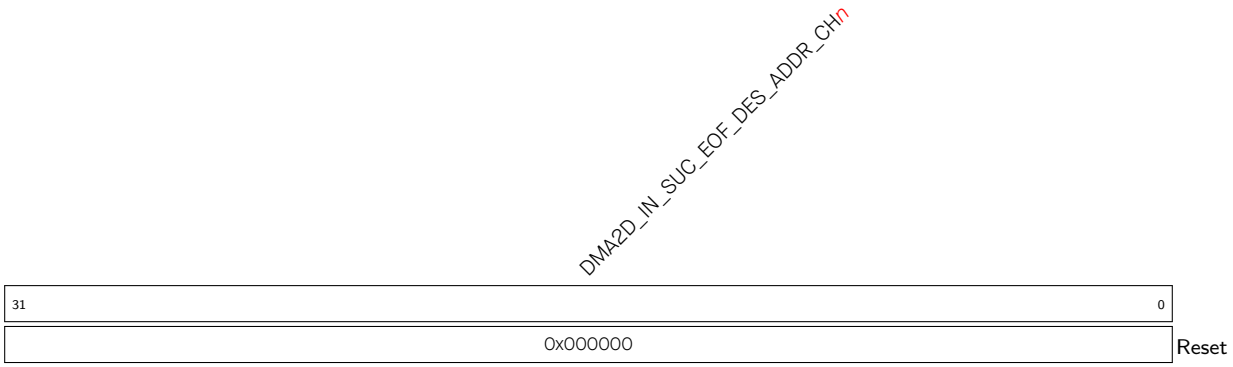
DMA2D_IN_RESET_AVAIL_CH*n* Represents whether it is safe to reset RX channel *n*.

0: Not safe

1: Safe

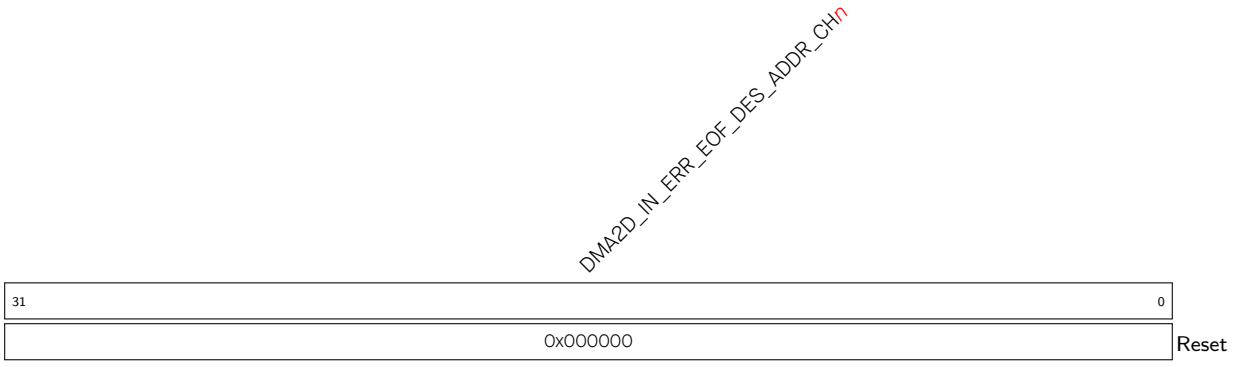
(RO)

Register 4.55. DMA2D_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0528+0x100**n*)



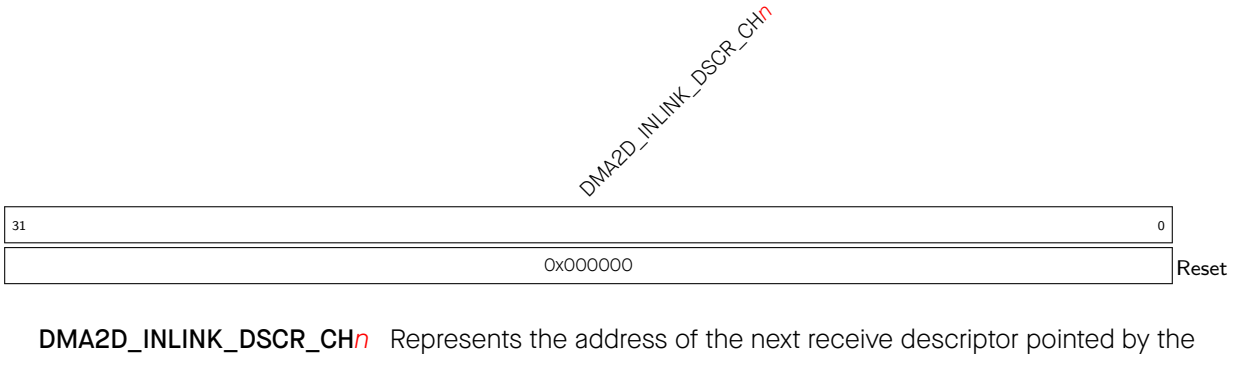
DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the eof bit in this descriptor is 1. (RO)

Register 4.56. DMA2D_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x052C+0x100**n*)



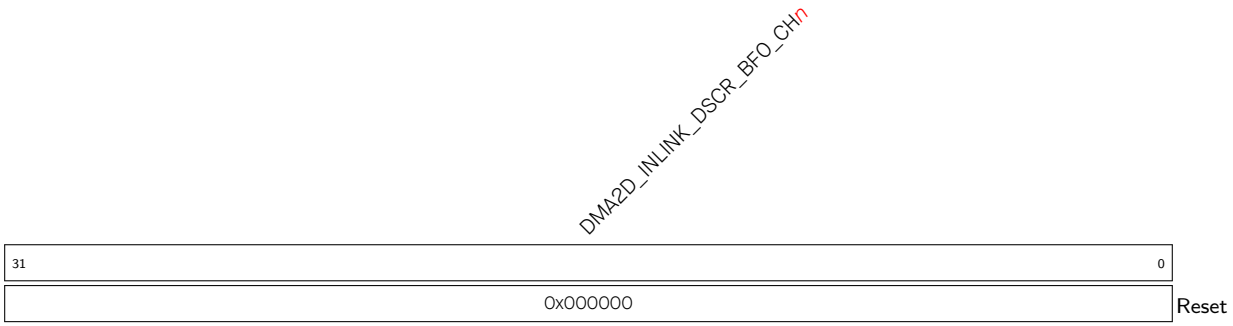
DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for JPEG. (RO)

Register 4.57. DMA2D_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0530+0x100**n*)



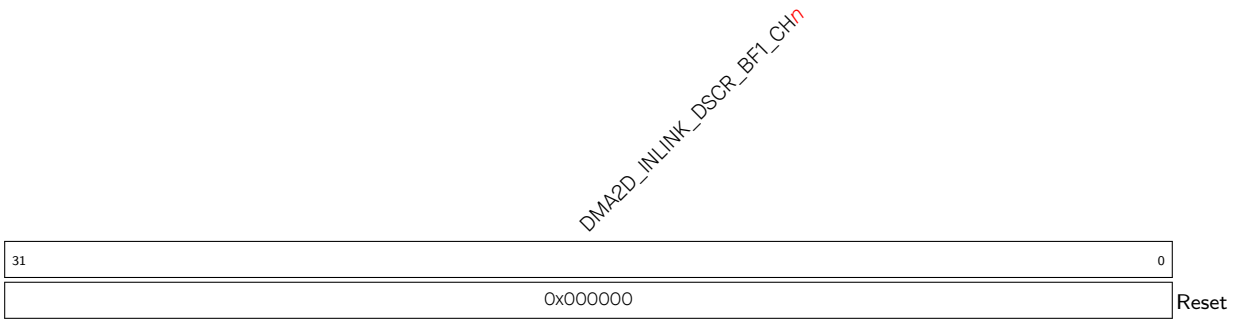
DMA2D_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor pointed by the current receive descriptor that is pre-read. (RO)

Register 4.58. DMA2D_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0534+0x100n*)**



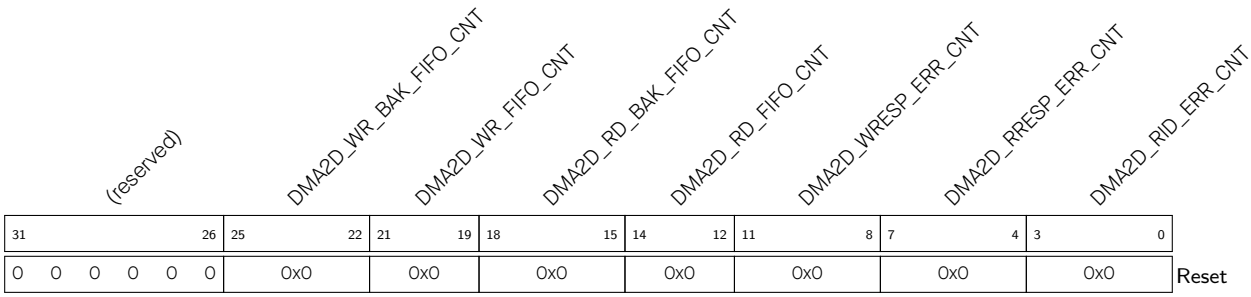
DMA2D_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor that is pre-read. (RO)

Register 4.59. DMA2D_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0538+0x100n*)**



DMA2D_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor that is pre-read. (RO)

Register 4.60. DMA2D_AXI_ERR_REG (0x0A00)



DMA2D_RID_ERR_CNT Represents the number of AXI read ID errors. (RO)

DMA2D_RRESP_ERR_CNT Represents the number of AXI read response errors. (RO)

DMA2D_WRESP_ERR_CNT Represents the number of AXI write response errors. (RO)

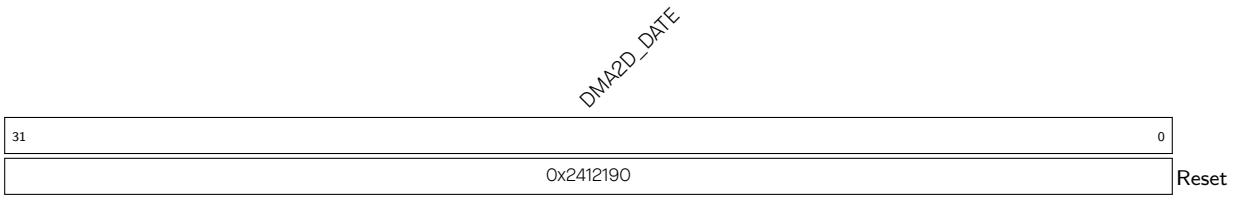
DMA2D_RD_FIFO_CNT Represents the number of remaining commands in the AXI read command FIFO. (RO)

DMA2D_RD_BAK_FIFO_CNT Represents the number of remaining commands in the AXI read backup command FIFO. (RO)

DMA2D_WR_FIFO_CNT Represents the number of remaining commands in the AXI write command FIFO. (RO)

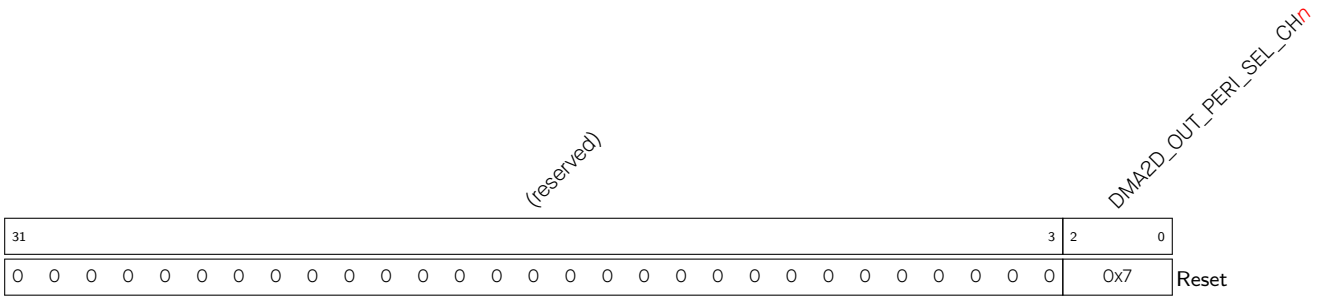
DMA2D_WR_BAK_FIFO_CNT Represents the number of remaining commands in the AXI write backup command FIFO. (RO)

Register 4.61. DMA2D_DATE_REG (0x0A2C)



DMA2D_DATE Version control register. (R/W)

Register 4.62. DMA2D_OUT_PERI_SEL_CH*n*_REG (*n*: 0-3) (0x0038+0x100n*)**



DMA2D_OUT_PERI_SEL_CH*n* Configures the peripheral connected to TX channel *n*.

0: JPEG

1: SRM module of PPA

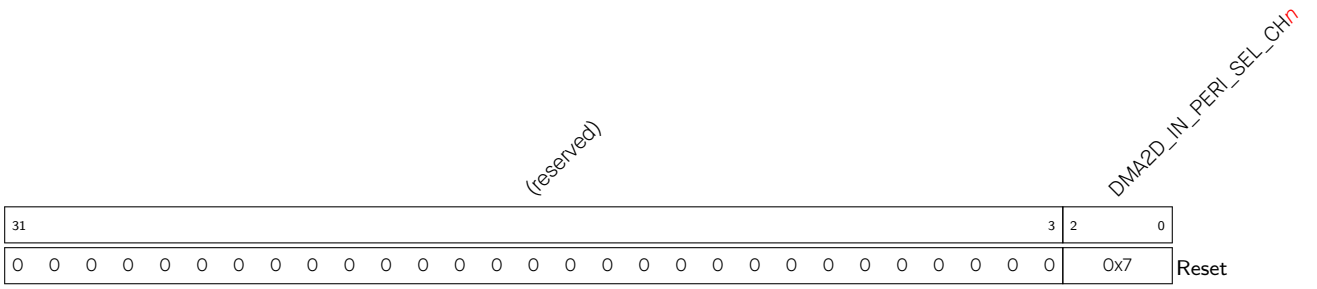
2: BLEND0 channel of PPA

3: BLEND1 channel of PPA

4 ~ 7: Dummy 4 ~ 7

(R/W)

Register 4.63. DMA2D_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x053C+0x100n*)**



DMA2D_IN_PERI_SEL_CH*n* Configures the peripheral connected to RX channel *n*.

0: JPEG

1: SRM module of PPA

2: BLEND module of PPA

3 ~ 7: Dummy 3 ~ 7

(R/W)

Part II

System Component

Encompassing a range of system-level functionalities, this part describes components related to system boot, clocks, GPIO, timers, watchdogs, debug assistance, event and interrupt handling, low-power management, and various system registers.

Chapter 5

Watchdog Timers (WDT)

5.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g., is stuck in a software loop or in overdue events) will fail to feed the watchdog, thus the watchdog timer will time out and trigger an interrupt or reset, directing the system to a known state for exception handling or a restart. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

As shown in Figure 5.1-1, ESP32-S31 contains three digital watchdog timers: one in each of the two timer groups in Chapter 13 *Timer Group (TIMG) [to be added later]* (called Main System Watchdog Timers, or MWDT) and one in the LP system (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages, and each stage can be programmed to take one action upon timeout, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, HP CPU reset, and HP core reset, while RWDT supports four timeout actions: interrupt, HP CPU reset, HP core reset, and system reset (see details in Section 5.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

In SPI Boot mode, RWDT and the MWDT in timer group 0 are enabled automatically in order to detect errors that may occur during the flash boot process and facilitate recovery.

ESP32-S31 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

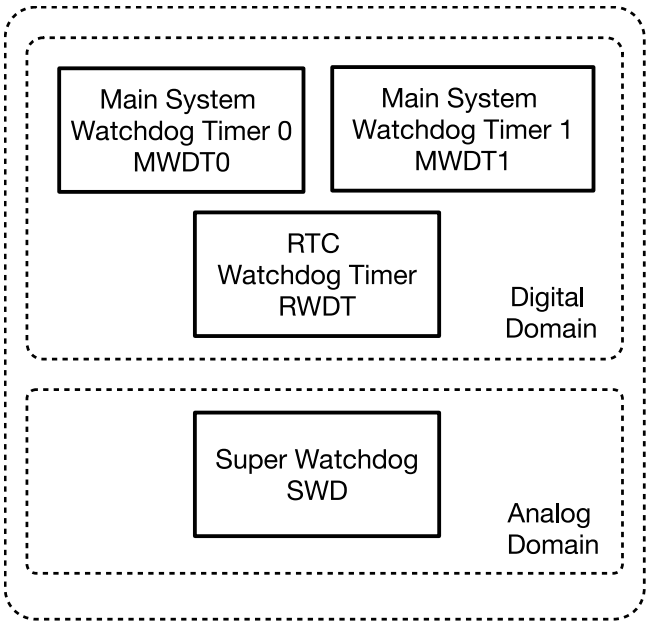


Figure 5.1-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer's, MWDT register descriptions are detailed in Chapter 13 *Timer Group (TIMG) [to be added later]*, and the RWDT and SWD register descriptions are detailed in Section 5.5 *Register Summary*.

5.2 Digital Watchdog Timers

5.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWDT: interrupt, HP CPU reset, HP core reset
  - RWDT: interrupt, HP CPU reset, HP core reset, system reset
- Flash boot protection under SPI Boot mode at stage 0:
  - MWDT0: HP core reset upon timeout
  - RWDT: system reset upon timeout
- Write protection that makes WDT register read only unless unlocked
- 32-bit timeout counter
- Clock source:
  - MWDT: PLL_F80M_CLK, RC_FAST_CLK or XTAL_CLK
  - RWDT: LP_DYN_SLOW_CLK

5.2.2 Functional Description

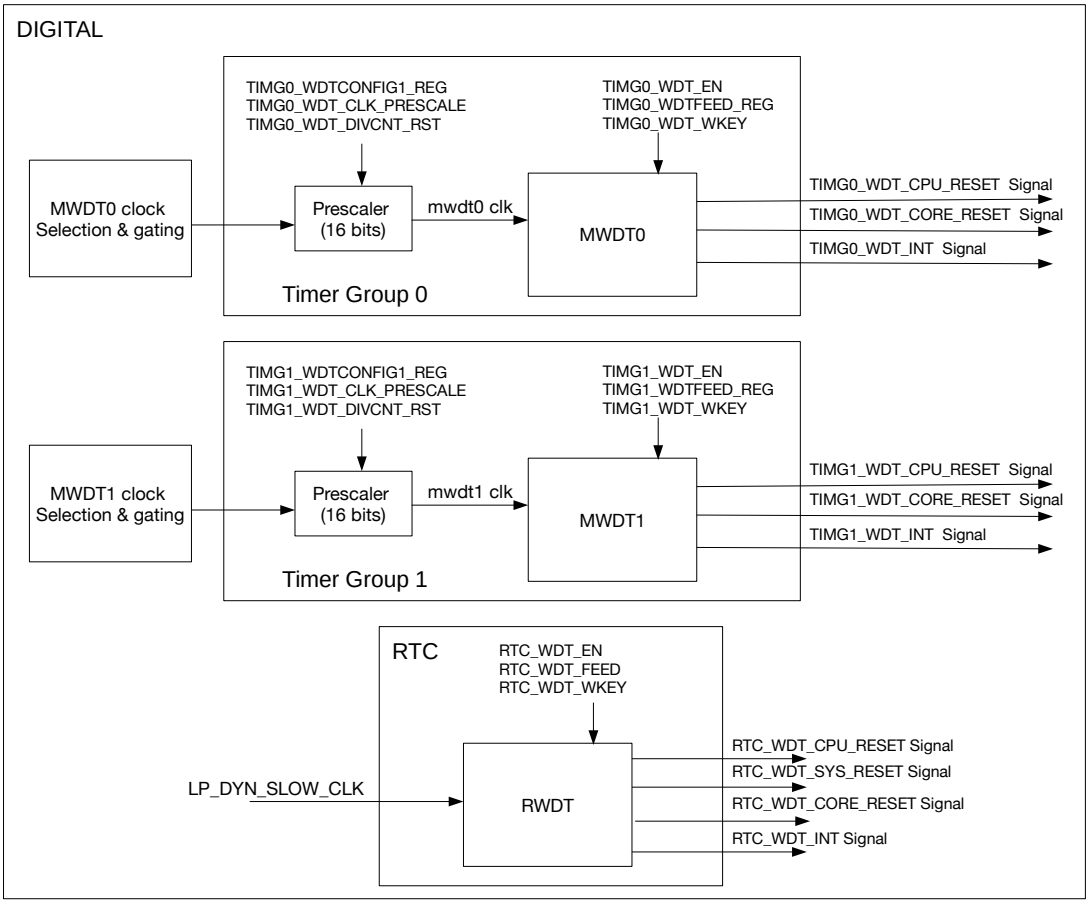


Figure 5.2-1. Digital Watchdog Timers in ESP32-S31

Figure 5.2-1 shows the three watchdog timers in ESP32-S31 digital systems.

5.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

Take MWDT0 as an example:

- MWDT0 can select between the PLL_F80M_CLK, RC_FAST_CLK or XTAL_CLK (external) clock as its clock source by setting the HP_SYS_TIMERGRP0_WDT_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register.
- The selected clock is switched on by setting HP_SYS_TIMERGRP0_WDT_CLK_EN field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register to 1 and switched off by setting it to 0. Then the selected clock is divided by a 16-bit configurable prescaler. See more details in Table **??** of Chapter 7.
- The 16-bit prescaler for MWDT is configured via the TIMG_WDT_CLK_PRESCALE field of TIMG_WDTCONFIG1_REG. When TIMG_WDT_DIVCNT_RST field is set, the prescaler is reset and it can be re-configured at once.

In contrast, the clock source of RWDT is derived directly from LP_DYN_SLOW_CLK (see details in Chapter 7

Reset and Clock [to be added later]).

MWDT and RWDT are enabled by setting the TIMG_WDT_EN and RTC_WDT_EN fields respectively. When enabled, the 32-bit counters of the watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e., timeout of the current stage). When this occurs, the current counter value is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to TIMG_WDTFEED_REG for MDWT and by writing 1 to RTC_WDT_FEED for RWDT.

5.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout action. When one stage times out, the timeout action is triggered, the counter value is reset to zero, and the next stage becomes active.

MWDT/RWDT offers four stages (referred to as stages 0 to 3). The watchdog timers will progress through each stage in a loop (i.e., from stage 0 to 3, then back to stage 0).

Timeout values of each stage for MWDT are configured in TIMG_WDTCONFIG*i*_REG (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured using RTC_WDT_STG*j*_HOLD field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT ($Thold_0$) is determined by the combination of the
EFUSE_WDT_DELAY_SEL field of eFuse register EFUSE_RD_REPEAT_DATA0_REG and RTC_WDT_STG0_HOLD field. The relationship is as follows:

$$T_{hold0} = RTC\_WDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

where << is a left-shift operator. For example, if RTC_WDT_STG0_HOLD is configured as 100 and EFUSE_WDT_DELAY_SEL is 1, the $Thold_0$ will be 400 cycles.

Upon the timeout of each stage, one of the following timeout actions will be executed:

Table 5.2-1. Timeout Actions

Timeout Action	Description
Interrupt	Trigger an interrupt
HP CPU reset	Reset the HP CPU0 and HP CPU1
HP core reset	HP core reset resets HP CPU0, HP CPI1, HP peripherals, HP GPIO, etc.
System reset	Reset the whole digital system, including the LP system.
Disabled	No effect on the system

For MWDT, the timeout action of all stages is configured in TIMG_WDTCONFIG0_REG. Likewise for RWDT, the timeout action is configured in RTC_WDT_CONFIG0_REG.

5.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g., due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write

protection mechanism that prevent the watchdogs from being disabled or tampered with due to an accidental write.

The write protection mechanism is implemented using a write-key field for each timer (TIMG_WDT_WKEY for MWDT, RTC_WDT_WKEY for RWDT). The value 0x50D83AA1 must be written to the watchdog timer's write-key field before any other register of the same watchdog timer can be changed. Any attempts to write to a watchdog timer's registers (other than the write-key field itself) whilst the write-key field's value is not 0x50D83AA1 will be ignored. The recommended procedure for accessing a watchdog timer is as follows:

1. Disable the write protection by writing the value 0x50D83AA1 to the timer's write-key field.
2. Make the required modification of the watchdog such as feeding or changing its configuration.
3. Re-enable write protection by writing any value other than 0x50D83AA1 to the timer's write-key field.

5.2.2.4 Flash Boot Protection

Under the SPI boot mode, MWDT0 as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT0 is automatically configured as HP core reset action upon timeout, known as HP core reset. Likewise, stage 0 for RWDT is configured to system reset, which resets the main system and RTC when it times out. After SPI booting, TIMG_WDT_FLASHBOOT_MOD_EN and RTC_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for both MWDT0 and RWDT respectively. After this, MWDT0 and RWDT can be configured by software.

5.3 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.

5.3.1 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

5.3.2 Super Watchdog Controller

5.3.2.1 Structure

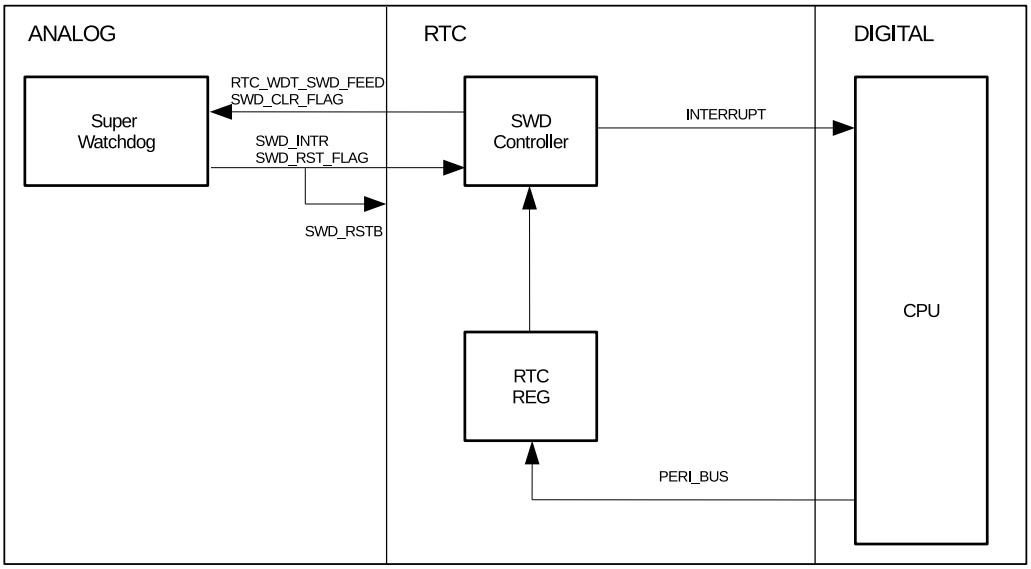


Figure 5.3-1. Super Watchdog Controller Structure

5.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU.
- Main CPU can feed SWD directly by setting RTC_WDT_SWD_FEED.
- When trying to feed SWD, CPU needs to disable SWD controller's write protection by writing 0x50D83AA1 to RTC_WDT_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_WDT_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU.

After reset:

- Check LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] for the cause of HP CPU reset.
  If LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] == 0x12, it indicates that the cause is SWD reset.
- Set RTC_WDT_SWD_RST_FLAG_CLR to clear the SWD reset flag.

5.4 Interrupts

For watchdog timer interrupts, please refer to Section **?? ??** in Chapter 13 *Timer Group (TIMG) [to be added later]*.

5.5 Register Summary

The addresses in this section are relative to RTC_WDT base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

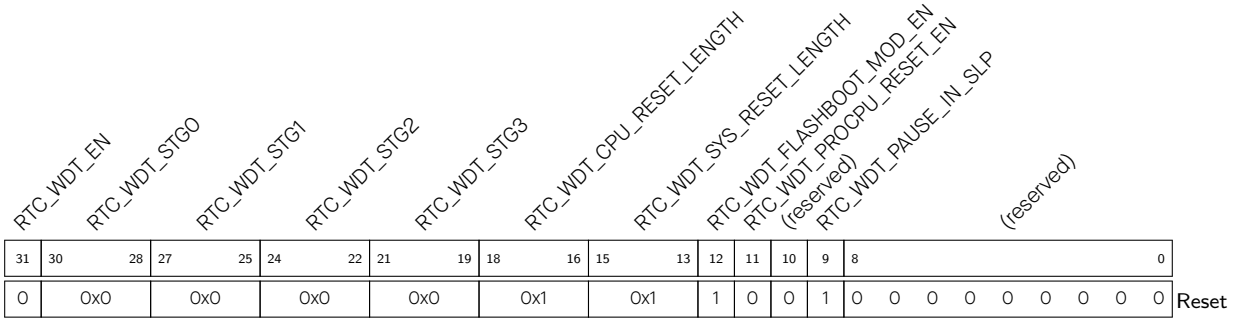
Name	Description	Address	Access
configuration register			
RTC_WDT_CONFIG0_REG	Configure the RWDT operation	0x0000	R/W
RTC_WDT_CONFIG1_REG	Configure the RWDT timeout time of stage0	0x0004	R/W
RTC_WDT_CONFIG2_REG	Configure the RWDT timeout time of stage1	0x0008	R/W
RTC_WDT_CONFIG3_REG	Configure the RWDT timeout time of stage2	0x000C	R/W
RTC_WDT_CONFIG4_REG	Configure the RWDT timeout time of stage3	0x0010	R/W
RTC_WDT_FEED_REG	Configure the feed function of RWDT	0x0014	WT
RTC_WDT_WPROTECT_REG	Configure the lock function of RWDT	0x0018	R/W
RTC_WDT_SWD_CONFIG_REG	Configure the SWD operation	0x001C	varies
RTC_WDT_SWD_WPROTECT_REG	Configure the lock function of SWD	0x0020	R/W
RTC_WDT_INT_RAW_REG	Configure whether to generate timeout interrupt	0x0024	R/WTC/SS
RTC_WDT_INT_ST_REG	The interrupt status register of WDT	0x0028	RO
RTC_WDT_INT_ENA_REG	The interrupt enable register of WDT	0x002C	R/W
RTC_WDT_INT_CLR_REG	The interrupt clear register of WDT	0x0030	WT
RTC_WDT_DATE_REG	Version control register	0x03FC	R/W

5.6 Registers

MWDT registers are part of the timer submodule and are described in Section **?? ??** in Chapter 13 *Timer Group (TIMG) [to be added later]*.

The addresses of RWDT and SWD registers in this section are relative to RTC_WDT base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Register 5.1. RTC_WDT_CONFIG0_REG (0x0000)



RTC_WDT_PAUSE_IN_SLP Configure whether or not pause RWDT when chip is in sleep mode.

0: Enable

1: Disable

(R/W)

RTC_WDT_PROCPU_RESET_EN Configure whether or not to enable RWDT to reset CPU.

0: Disable

1: Enable

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN Configure whether or not to enable RWDT when chip is in SPI boot mode.

0: Disable

1: Enable

(R/W)

RTC_WDT_SYS_RESET_LENGTH Configure the HP core reset time.

Measurement unit: LP_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH Configure the HP CPU reset time.

Measurement unit: LP_DYN_FAST_CLK

(R/W)

RTC_WDT_STG3 Configure the timeout action of stage3.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

Continued on the next page...

Register 5.1. RTC_WDT_CONFIG0_REG (0x0000)

Continued from the previous page...

RTC_WDT_STG2 Configure the timeout action of stage2.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

RTC_WDT_STG1 Configure the timeout action of stage1.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

RTC_WDT_STG0 Configure the timeout action of stage0.

0: No operation

1: Generate interrupt

2: Generate HP CPU reset

3: Generate HP core reset

4: Generate system reset

(R/W)

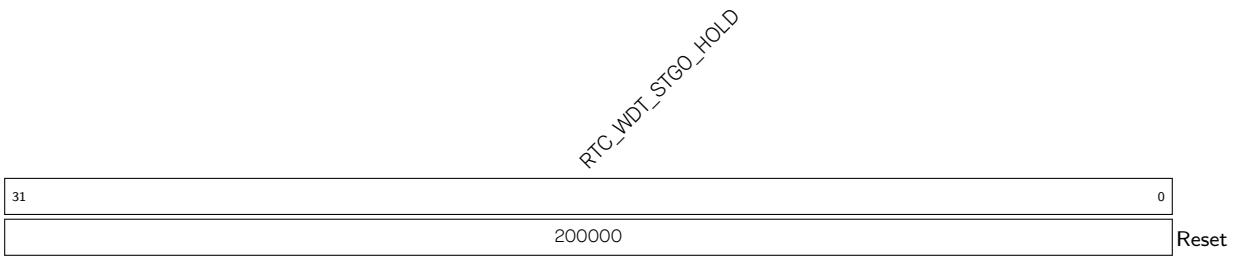
RTC_WDT_EN Configure whether or not enable RWDT.

0: Disable RWDT

1: Enable RWDT

(R/W)

Register 5.2. RTC_WDT_CONFIG1_REG (0x0004)

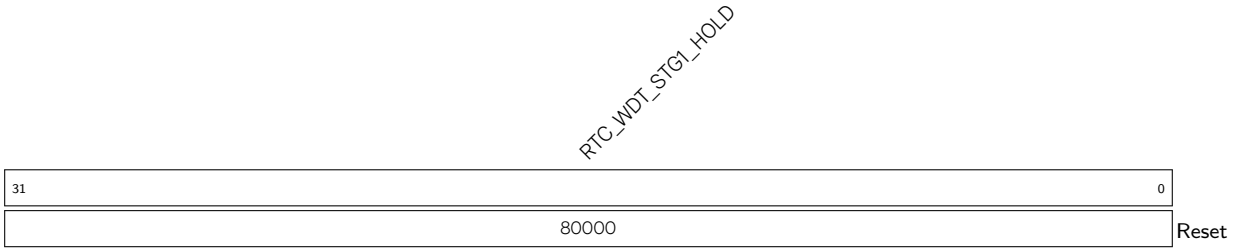


RTC_WDT_STG0_HOLD Configure the timeout time for stage0.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 5.3. RTC_WDT_CONFIG2_REG (0x0008)

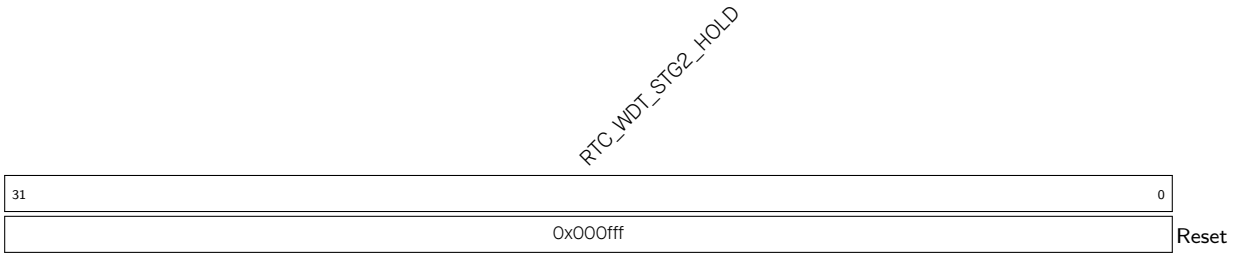


RTC_WDT_STG1_HOLD Configure the timeout time for stage1.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 5.4. RTC_WDT_CONFIG3_REG (0x000C)

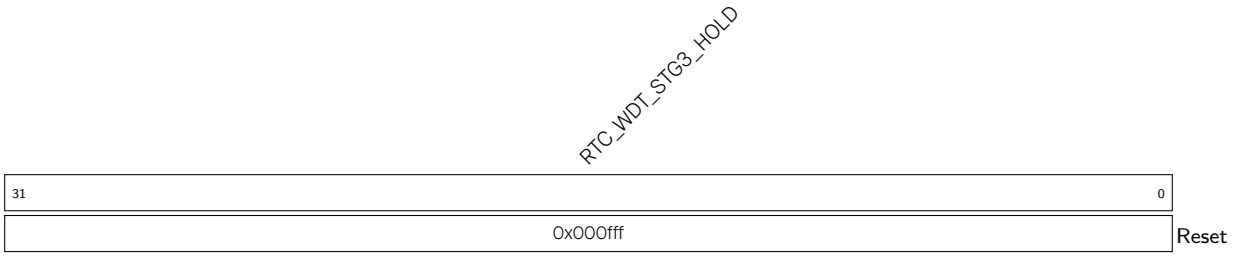


RTC_WDT_STG2_HOLD Configure the timeout time for stage2.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 5.5. RTC_WDT_CONFIG4_REG (0x0010)

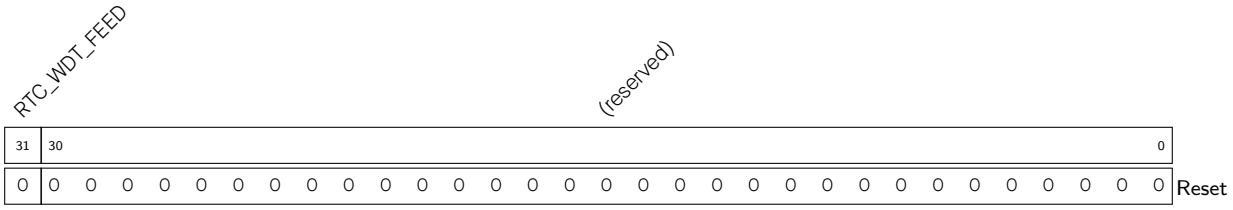


RTC_WDT_STG3_HOLD Configure the timeout time for stage3.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 5.6. RTC_WDT_FEED_REG (0x0014)



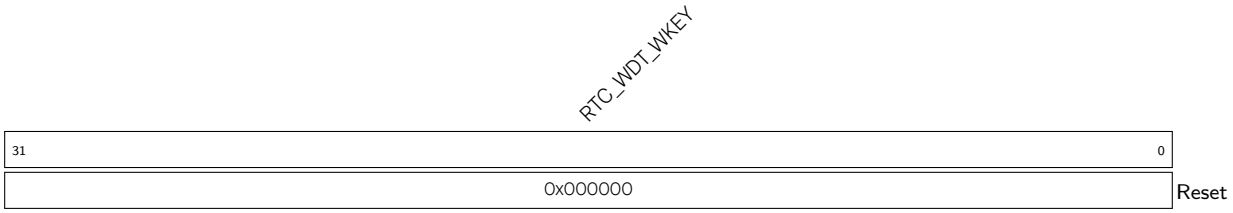
RTC_WDT_FEED Configure this bit to feed the RWDT.

0: Invalid

1: Feed RWDT

(WT)

Register 5.7. RTC_WDT_WPROTECT_REG (0x0018)



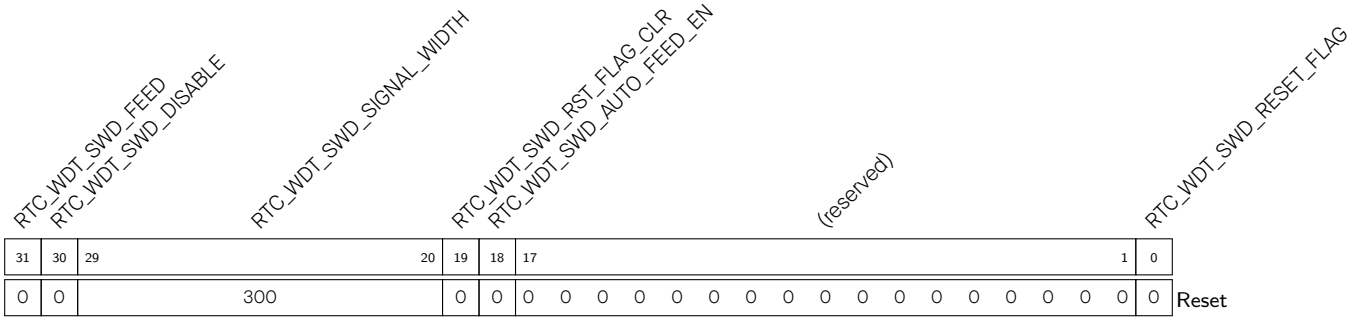
RTC_WDT_WKEY Configure this field to lock or unlock RWDT's configuration registers.

0x50D83AA1: unlock the RWDT configuration register

Others value: lock the RWDT configuration register which can't be modified by software.

(R/W)

Register 5.8. RTC_WDT_SWD_CONFIG_REG (0x001C)



RTC_WDT_SWD_RESET_FLAG Represents the SWD whether has generated the reset signal

0: No

1: Yes

(RO)

RTC_WDT_SWD_AUTO_FEED_EN Configure this bit to enable to feed SWD automatically by hardware.

0: Disable

1: Enable

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR Configure this bit to clear SWD reset flag

0: Invalid

1: Clear the reset flag

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH Configure the SWD signal length that output to analog circuit. Measurement unit: LP_DYN_FAST_CLK (R/W)

RTC_WDT_SWD_DISABLE Configure this bit to disable the SWD.

0: Enable the SWD

1: Disable the SWD

(R/W)

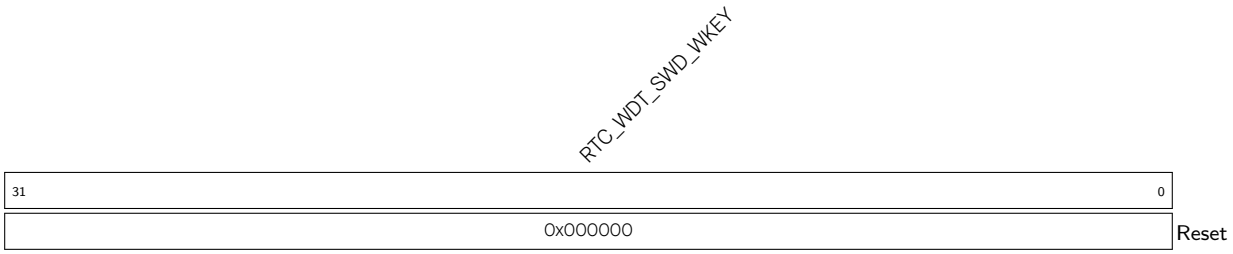
RTC_WDT_SWD_FEED Configure this bit to feed the SWD.

0: Invalid

1: Feed SWD

(WT)

Register 5.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)



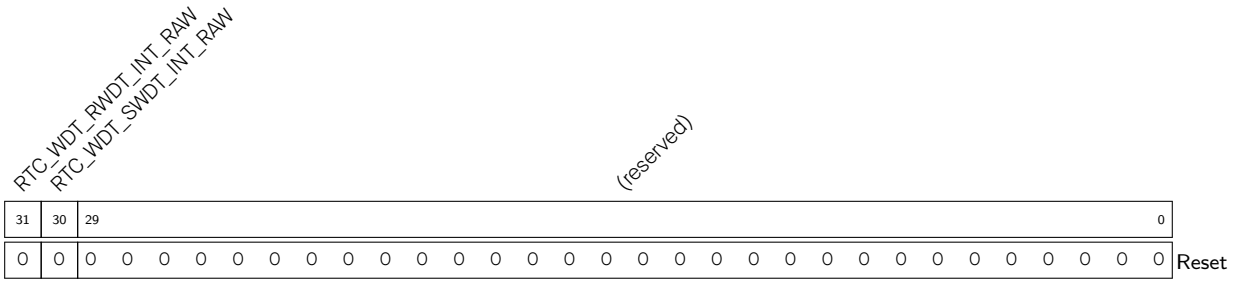
RTC_WDT_SWD_WKEY Configure this field to lock or unlock SWD's configuration registers.

0x50D83AA1: unlock the SWD configuration register.

Others value: lock the SWD configuration register which can't be modified by the software.

(R/W)

Register 5.10. RTC_WDT_INT_RAW_REG (0x0024)



RTC_WDT_SWDT_INT_RAW Represents the SWD whether or not has generated timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

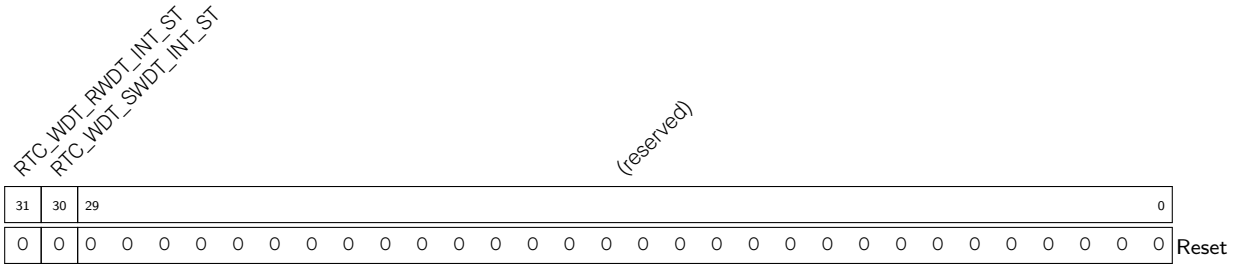
RTC_WDT_RWDT_INT_RAW Represents the RWDT whether or not generates timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

Register 5.11. RTC_WDT_INT_ST_REG (0x0028)



RTC_WDT_SWDT_INT_ST Represents the SWD whether or not has generated and sent timeout interrupt to CPU.

0: No

1: Yes

(RO)

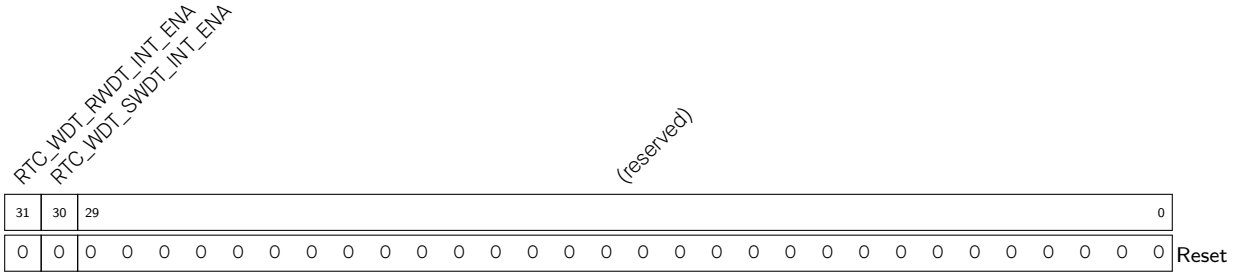
RTC_WDT_RWDT_INT_ST Represents the RWDT whether or not generates and sends timeout interrupt to CPU.

0: No

1: Yes

(RO)

Register 5.12. RTC_WDT_INT_ENA_REG (0x002C)



RTC_WDT_SWDT_INT_ENA Configure whether or not to enable the SWD to send timeout interrupt.

0: Disable

1: Enable

(R/W)

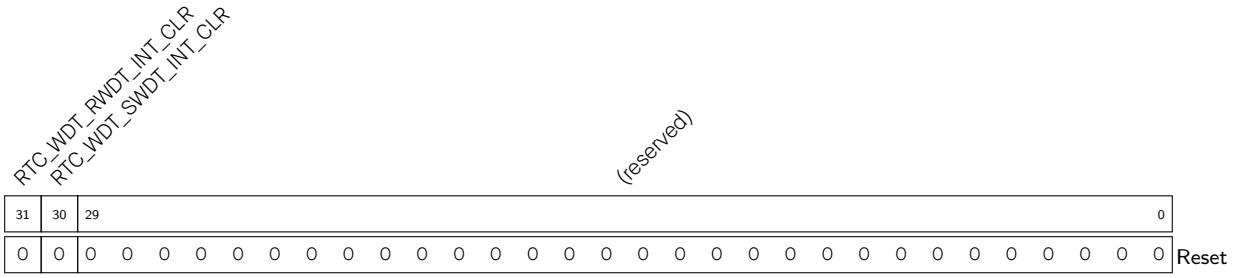
RTC_WDT_RWDT_INT_ENA Configure whether or not to enable the RWDT to send timeout interrupt.

0: Disable

1: Enable

(R/W)

Register 5.13. RTC_WDT_INT_CLR_REG (0x0030)



RTC_WDT_SWDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by SWD to CPU.

0: No

1: Yes

(WT)

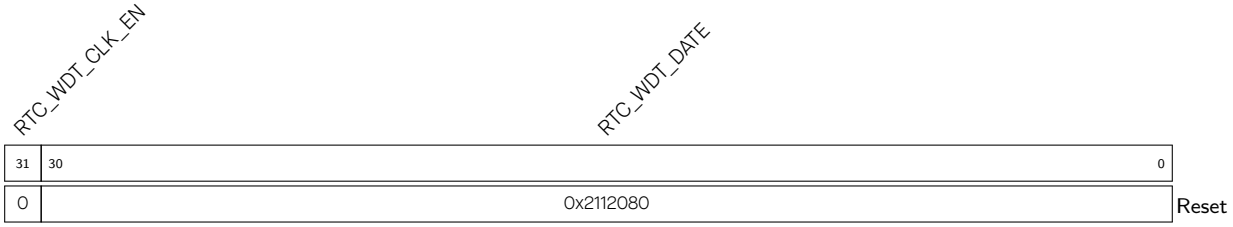
RTC_WDT_RWDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by RWDT to CPU.

0: No

1: Yes

(WT)

Register 5.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE Version control register. (R/W)

RTC_WDT_CLK_EN Reserved. (R/W)

Chapter 6

RTC Timer

6.1 Introduction

RTC Timer is an important module for implementing low power management of ESP32-S31. Based on a 48-bit readable counter, RTC Timer is mainly used as a system timer in low power mode when the timer peripheral in the HP system is unavailable. It also allows for configuring timer interrupts and logging the time when specific events happen in the system. The supported events are described in the 6.3 Functional Description.

6.2 Feature List

RTC Timer has the following features:

- 48-bit counter
- Time logging when one of the following events happens:
  - HP system reset
  - CPU enters stall state
  - CPU exits stall state
  - Crystal powers up
  - Crystal powers down
- Time logging through register configuration
- Occurrence time cached of the most recent two specific events
- Generation of interrupts at target times, which are configurable. It is also possible to configure two target times simultaneously.
- Uninterrupted operation during any reset or sleep mode, except for power-on reset of LP system.

6.3 Functional Description

- 48-bit counter
  - The implementation of the RTC Timer is based on a 48-bit counter, driven by the RC_SLOW_CLK in the AlwaysOn power domain. It uses continuous loop counting (except during LP system reset), and an overflow interrupt is generated when the 48-bit counter overflows.
- Log the occurrence time of specific events
  - RTC Timer supports logging the occurrence time of three types of events:

    * HP system reset
    * CPU enters or exits stall state
    * Crystal powers up or down
  - The above three types of events have their own independent configuration registers. When the configuration registers are enabled, if the corresponding events occur, the hardware pulse generated will trigger the RTC Timer to log the current time. The corresponding configuration registers are shown in the table below.

Table 6.3-1. Configure RTC Timer to Log the Occurrence Time of Specific Events

Configuration Register	Description
RTC_TIMER_MAIN_TIMER_SYS_RST	Enable the RTC Timer to log the time of system reset.
RTC_TIMER_MAIN_TIMER_SYS_STALL	Enable the RTC Timer to log the time when the CPU enters or exits the stall state.
RTC_TIMER_MAIN_TIMER_XTAL_OFF	Enable the RTC Timer to log the time when the crystal powers up or down.

- Log the current time through software configuration.
  - In addition to the above three specific events that can trigger the RTC Timer to log the time, the RTC Timer can also record the current time through the configuration register RTC_TIMER_MAIN_TIMER_UPDATE.
  - A pulse signal will be generated by configuring the RTC_TIMER_MAIN_TIMER_UPDATE, which will trigger the RTC Timer to record the current time.
- Generate counting interrupts at target times
  - RTC Timer supports configuring two target times simultaneously, referred to as target time 0 and target time 1. When the timer reaches target time 0, it will trigger the RTC_TIMER_CMP0_INT interrupt. Similarly, when the timer reaches Target Moment 1, it will trigger the RTC_TIMER_CMP1_INT interrupt.
  - Configure the target times through the registers below:

Table 6.3-2. Target times Configuration

Configuration Register	Description
RTC_TIMER_MAIN_TIMER_TAR_EN*n*	*n* = 0 or 1, enable target time configuration.
RTC_TIMER_MAIN_TIMER_TAR_LOW*n*	*n* = 0 or 1, configure the low 32 bits of the target time.
RTC_TIMER_MAIN_TIMER_TAR_HIGH*n*	*n* = 0 or 1, configure the high 16 bits of the target time.

- Read the time cached in the RTC Timer
  - There are two sets of registers used to cache the occurrence time of specific events, namely register group 0 and register group 1.
  - Register group 0 includes register RTC_TIMER_MAIN_BUF0_LOW_REG and RTC_TIMER_MAIN_BUF1_HIGH_REG which are used to cache the count value of the RTC Timer under the current trigger.

- Register group 1 includes register RTC_TIMER_MAIN_BUF1_LOW_REG and RTC_TIMER_MAIN_BUF1_HIGH_REG, which are used to cache the count value of the RTC Timer from the last trigger.
- On a new trigger, the record from the previous trigger will be moved from register group 0 to register group 1 (and the previous record in register group 1 will be overwritten), and the record of this trigger will be stored in register group 0. Therefore, only the last two triggers can be logged at any time.

6.4 Event Task Matrix Feature

The RTC Timer on ESP32-S31 supports the Event Task Matrix (ETM) function, which allows RTC Timer's ETM tasks to be triggered by any peripherals' ETM events, or RTC Timer's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to RTC Timer. For more information, please refer to Chapter 10 Event Task Matrix (ETM) [to be added later].

RTC Timer doesn't receive any ETM tasks:

RTC Timer can generate the following ETM events:

- RTC_EVT_CMP: Indicate that the RTC Timer reaches the target time 0 configured by RTC_TIMER_MAIN_TIMER_TAR_LOW0 and RTC_TIMER_MAIN_TIMER_TAR_HIGH0.
- RTC_EVT_OVF: Indicate the 48-bit counter overflow.
- RTC_EVT_TICK: Indicates the event that occurs when the RTC timer increments by 1.

6.5 Interrupts

ESP32-S31's RTC Timer can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- LP_TIMER_REG_0_INTR
- LP_TIMER_REG_1_INTR

There are several internal interrupt sources from RTC Timer that can generate the above interrupt signal(s). The interrupt sources from RTC Timer are listed with their trigger conditions and the resulted interrupt signal(s) in Table 6.5-1.

Table 6.5-1. RTC Timer's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
RTC_TIMER_CMP0_INT	RTC Timer reaches the target time 0	LP_TIMER_REG_0_INTR
RTC_TIMER_CMP1_INT	RTC Timer reaches the target time 1	LP_TIMER_REG_1_INTR
RTC_TIMER_OVERFLOW_INT	48-bit counter overflow	LP_TIMER_REG_0_INTR LP_TIMER_REG_1_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

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 6.6 *Register Summary*.

6.6 Register Summary

The addresses in this section are relative to RTC Timer base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

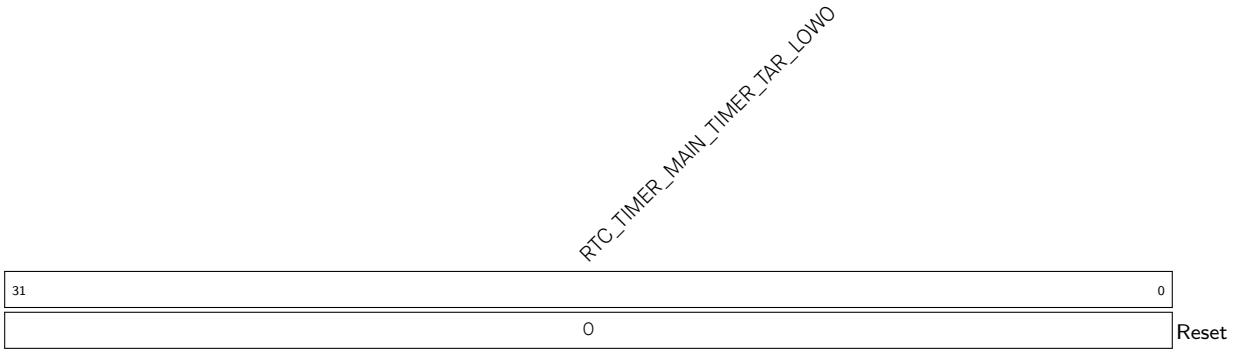
Name	Description	Address	Access
Configuration Registers			
RTC_TIMER_TAR0_LOW_REG	Configure the low 32 bits of the target time 0	0x0000	R/W
RTC_TIMER_TAR0_HIGH_REG	Configure the high 16 bits of the target time 0	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	Configure the low 32 bits of the target time 1	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	Configure the high 16 bits of the target time 1	0x000C	varies
RTC_TIMER_UPDATE_REG	Configure to enable the RTC Timer to latch current value	0x0010	varies
RTC_TIMER_MAIN_BUF0_LOW_REG	The low 32 bits of the cached value 0 in RTC Timer	0x0014	RO
RTC_TIMER_MAIN_BUF0_HIGH_REG	The high 16 bits of the cached value 0 in RTC Timer	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	The low 32 bits of the cached value 1 in RTC Timer	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	The high 16 bits of the cached value 1 in RTC Timer	0x0020	RO
Interrupt Registers			
RTC_TIMER_INT_RAW_REG	The interrupt raw status register for the RTC Timer reaching the target time 0	0x0028	RO
RTC_TIMER_INT_ST_REG	The interrupt status register for the RTC Timer reaching the target time 0	0x002C	RO
RTC_TIMER_INT_ENA_REG	The interrupt enable register for the RTC Timer reaching the target time 0	0x0030	R/W
RTC_TIMER_INT_CLR_REG	The interrupt clear register for the RTC Timer reaching the target time 0	0x0034	WO
RTC_TIMER_LP_INT_RAW_REG	The interrupt raw status register for RTC Timer reaching the target time 1	0x0038	RO
RTC_TIMER_LP_INT_ST_REG	The interrupt status register for the RTC Timer reaching the target time 1	0x003C	RO
RTC_TIMER_LP_INT_ENA_REG	The interrupt enable register for the RTC Timer reaching the target time 1	0x0040	R/W
RTC_TIMER_LP_INT_CLR_REG	The interrupt clear register for the RTC Timer reaching the target time 1	0x0044	WO
Version Register			
RTC_TIMER_DATE_REG	Version control register	0x03FC	R/W

6.7 Registers

The addresses in this section are relative to RTC Timer base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

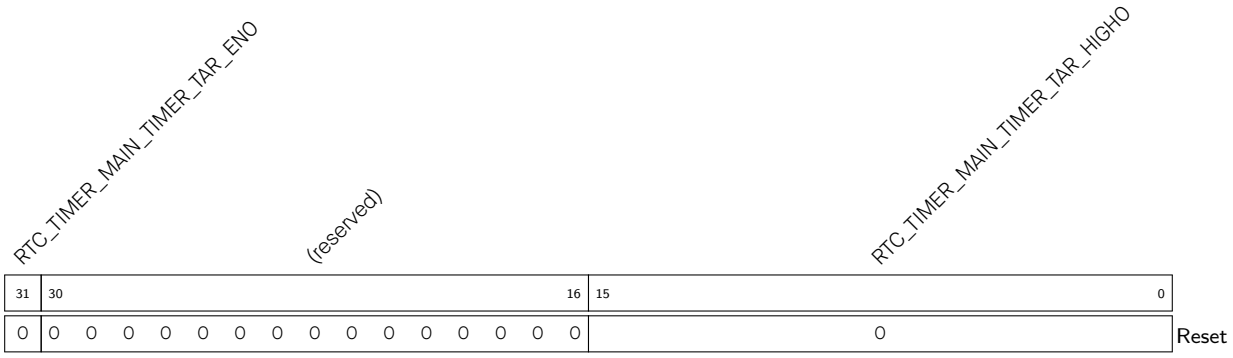
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 6.1. RTC_TIMER_TAR0_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 Configure the low 32 bits of the target time 0 of the RTC Timer. (R/W)

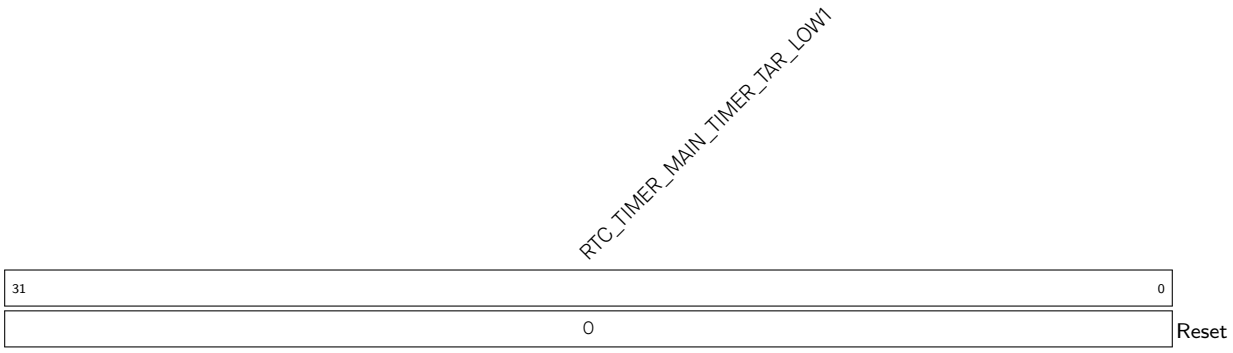
Register 6.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_HIGH0 Configure the high 16 bits of the target time 0. (R/W)

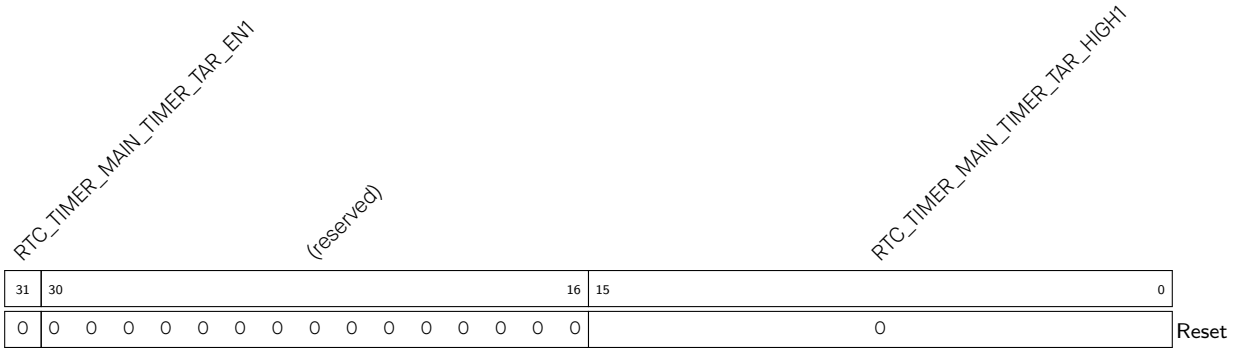
RTC_TIMER_MAIN_TIMER_TAR_EN0 Configure to enable the target time 0 of the RTC Timer. (WO)

Register 6.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 Configure the low 32 bits of the target time 1. (R/W)

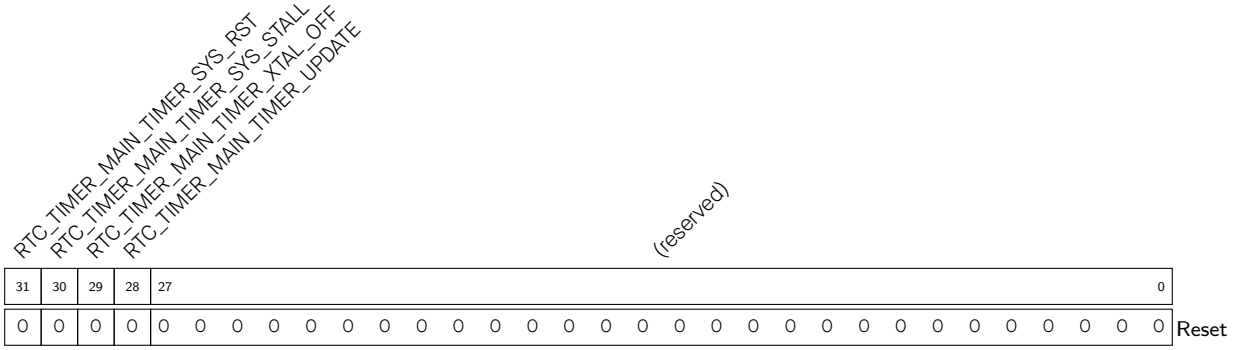
Register 6.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)



RTC_TIMER_MAIN_TIMER_TAR_HIGH1 Configure the high 16 bits of the target time 1. (R/W)

RTC_TIMER_MAIN_TIMER_TAR_EN1 Configure to enable the target time 1 of the RTC Timer. (WO)

Register 6.5. RTC_TIMER_UPDATE_REG (0x0010)



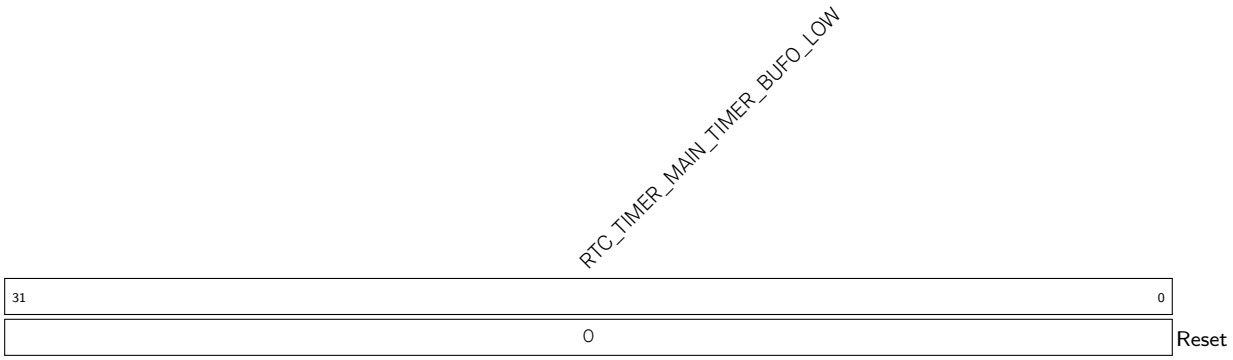
RTC_TIMER_MAIN_TIMER_UPDATE Configure to log the current time through software configuration. (WO)

RTC_TIMER_MAIN_TIMER_XTAL_OFF Configure to enable RTC Timer to log the time when the crystal powers up or down. (R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL Configure to enable the RTC Timer to log the time when the CPU enters or exits the stall state. (R/W)

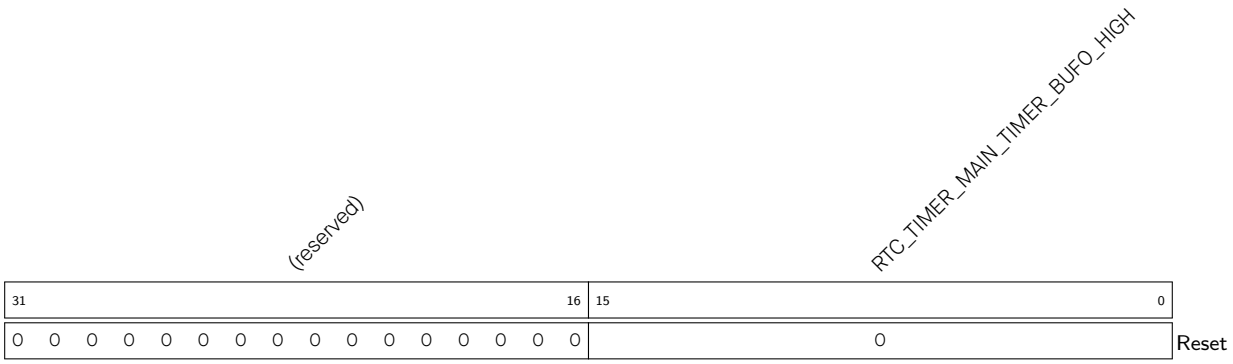
RTC_TIMER_MAIN_TIMER_SYS_RST Configure to enable RTC Timer to log the time of system reset. (R/W)

Register 6.6. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



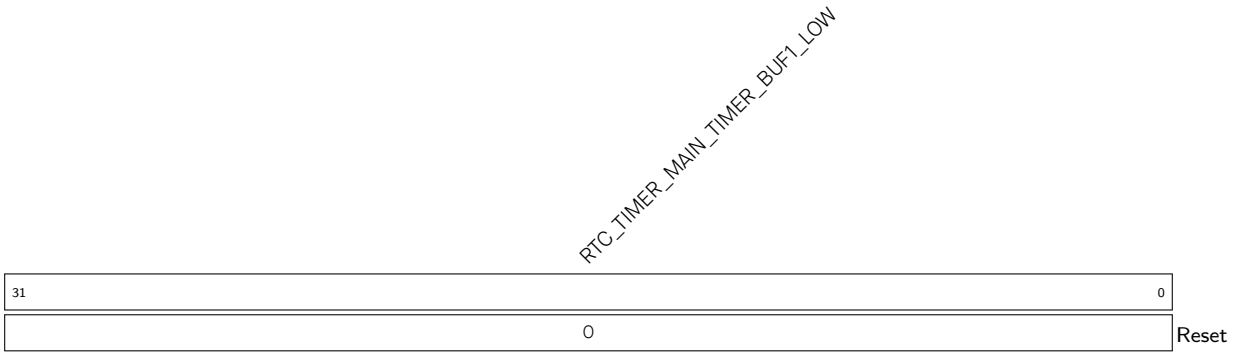
RTC_TIMER_MAIN_TIMER_BUF0_LOW Represent the low 32 bits of the cached value 0 in RTC Timer. (RO)

Register 6.7. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



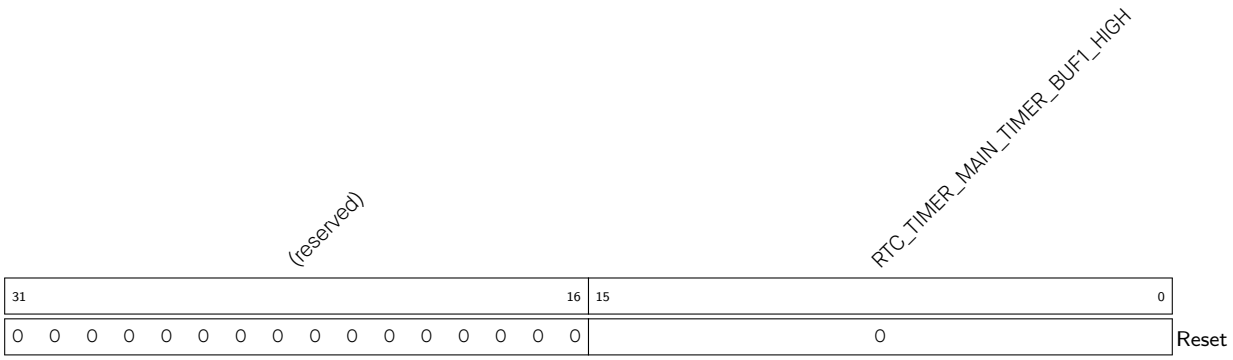
RTC_TIMER_MAIN_TIMER_BUF0_HIGH Represent the high 16 bits of the cached value 0 in RTC Timer. (RO)

Register 6.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



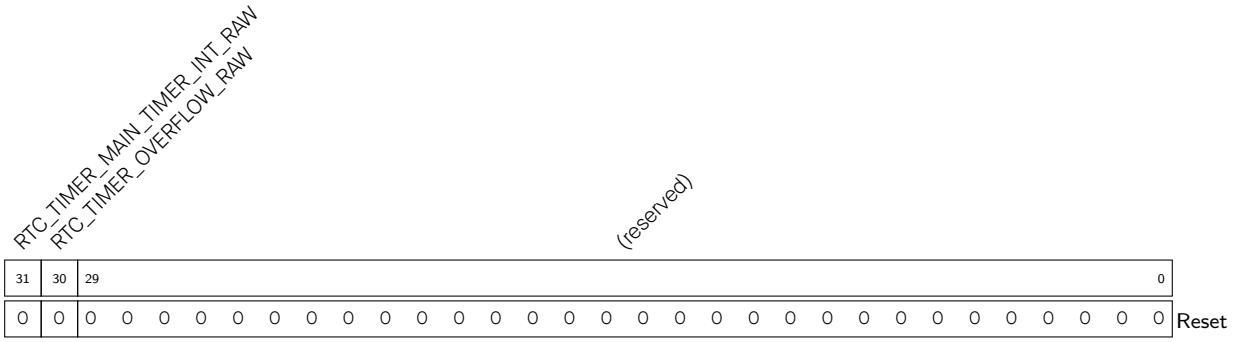
RTC_TIMER_MAIN_TIMER_BUF1_LOW Represent the low 32 bits of the cached value 1 in RTC Timer. (RO)

Register 6.9. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_TIMER_BUF1_HIGH Represent the high 16 bits of the cached value 1 in RTC Timer. (RO)

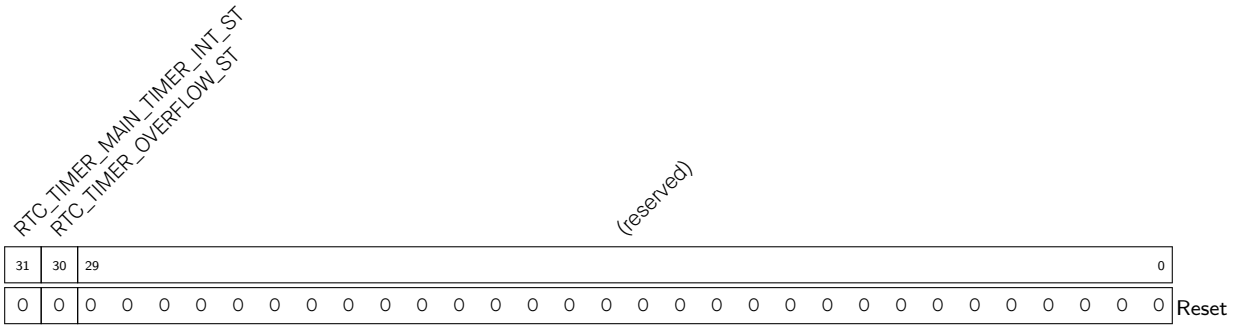
Register 6.10. RTC_TIMER_INT_RAW_REG (0x0028)



RTC_TIMER_OVERFLOW_RAW The interrupt raw status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_INT_RAW The interrupt raw status for the RTC Timer reaching the target time 0. (RO)

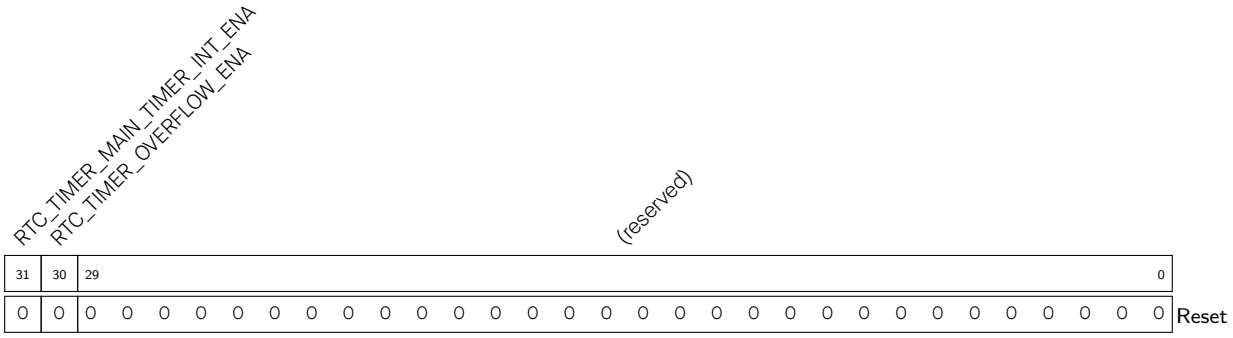
Register 6.11. RTC_TIMER_INT_ST_REG (0x002C)



RTC_TIMER_OVERFLOW_ST The interrupt status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_INT_ST The interrupt status for the RTC Timer reaching the target time 0. (RO)

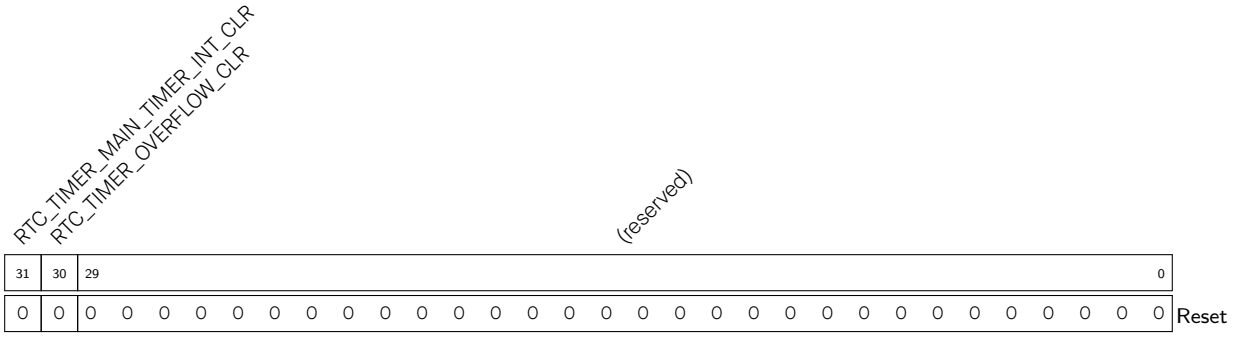
Register 6.12. RTC_TIMER_INT_ENA_REG (0x0030)



RTC_TIMER_OVERFLOW_ENA Configure to enable the interrupt for 48-bit counter overflow. (R/W)

RTC_TIMER_MAIN_TIMER_INT_ENA Configure to enable the interrupt for the RTC Timer reaching the target time 0. (R/W)

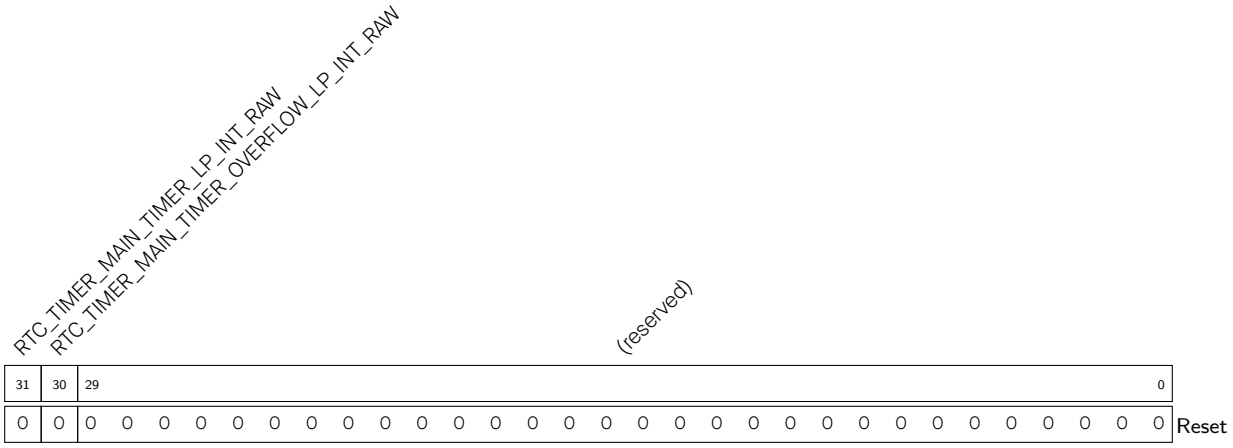
Register 6.13. RTC_TIMER_INT_CLR_REG (0x0034)



RTC_TIMER_OVERFLOW_CLR Configure to clear the interrupt for 48-bit counter overflow. (WO)

RTC_TIMER_MAIN_TIMER_INT_CLR Configure to clear the interrupt for the RTC Timer reaching the target time 0. (WO)

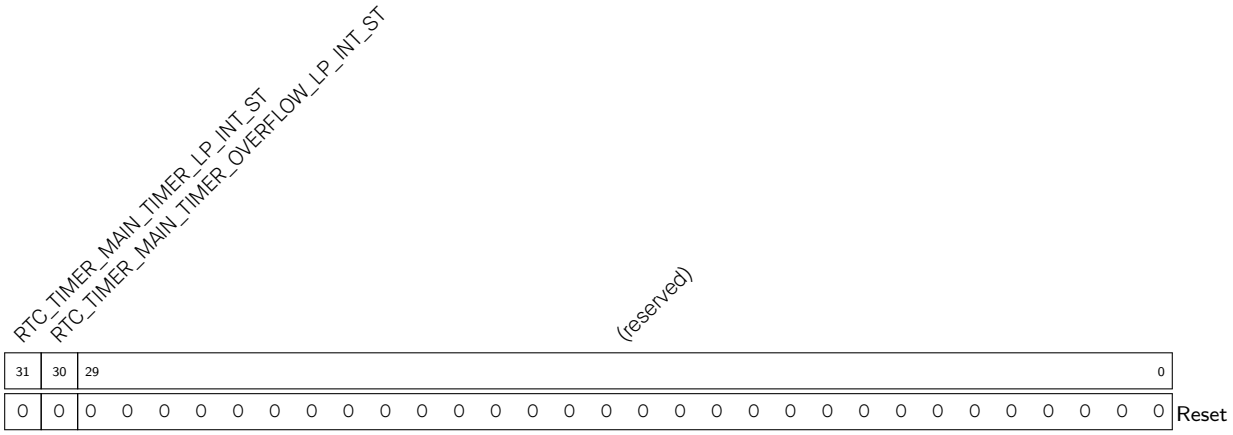
Register 6.14. RTC_TIMER_LP_INT_RAW_REG (0x0038)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_RAW The interrupt raw status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_LP_INT_RAW The interrupt raw status for the RTC Timer reaching the target time 1.(RO)

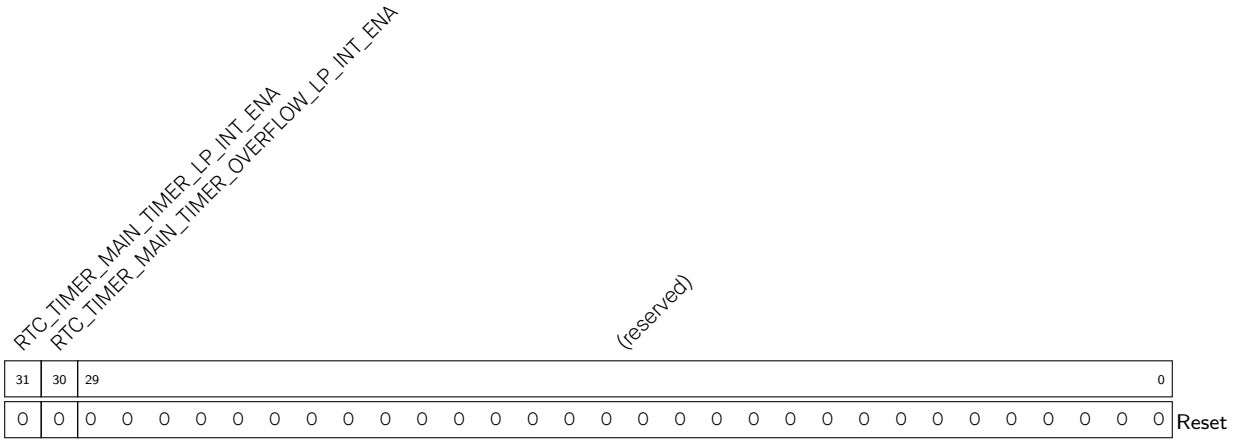
Register 6.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ST The interrupt status for the 48-bit counter overflow. (RO)

RTC_TIMER_MAIN_TIMER_LP_INT_ST The interrupt status for the RTC Timer reaching the target time 1.(RO)

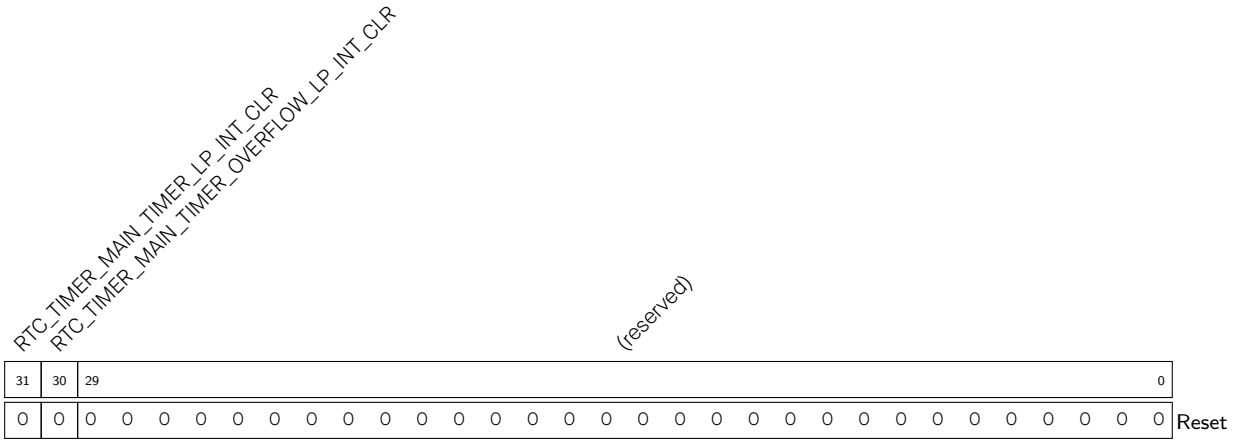
Register 6.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ENA Configure to enable the interrupt for the 48-bit counter overflow. (R/W)

RTC_TIMER_MAIN_TIMER_LP_INT_ENA Configure to enable the interrupt for the RTC Timer reaching the target time 1. (R/W)

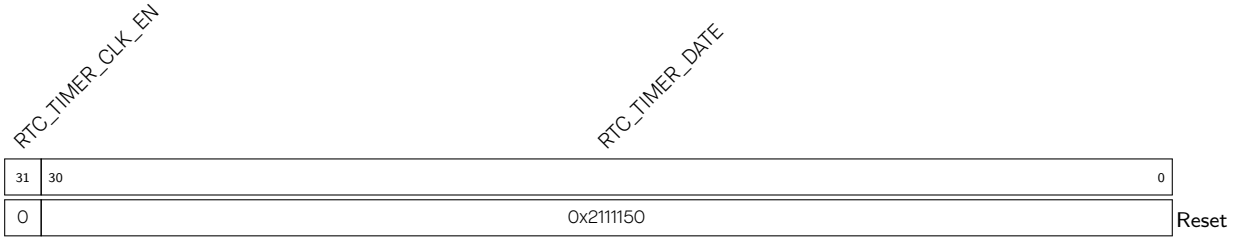
Register 6.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR Configure to clear the interrupt for 48-bit counter overflow. (WO)

RTC_TIMER_MAIN_TIMER_LP_INT_CLR Configure to clear the interrupt for the RTC Timer reaching the target time 1. (WO)

Register 6.18. RTC_TIMER_DATE_REG (0x03FC)



RTC_TIMER_DATE Version control register. (R/W)

RTC_TIMER_CLK_EN Configure to enable the register read/write clock. (R/W)

Chapter 7

Debug Assistant

7.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

7.2 Features

The Debug Assistant module has the following features:

- **Region read/write monitoring**: Monitors whether the bus of the High-Performance dual-core processor (High-Performance CPU, HP CPU0 and HP CPU1) reads from or writes to a specified memory address range. A detected read or write triggers an interrupt.
- **Stack pointer (SP) monitoring**: Monitors whether the SP exceeds the specified address space. A bounds violation triggers an interrupt.
- **Program counter (PC) and SP logging**: Records the PC value and SP value. The last PC value and SP value at the most recent reset of the HP CPU*n* (Below, *n* = 0, 1) can be retrieved.
- **Lockup monitoring**: Monitors whether HP CPU*n* encounters a lockup, and records the exception cause (cause), trap value (tval), instruction PC (iaddr), and current privilege level (priv) when the first and second exceptions occur.
- **Bus access logging**: Records write operation information of the HP CPU*n* Data bus and the DMA bus. Supports write monitoring and monitoring of specific written values by word, halfword, or byte; records the bus type, address, and the PC when the write is performed by the HP CPU*n*, then stores this information in the HP SRAM in a specific format. This feature is implemented through Bus Access Logging 1 (TCM_MEM_MONITOR, HP SRAM or external RAM) and Bus Access Logging 2 (PSRAM_MEM_MONITOR, external RAM).

7.3 Functional Description

7.3.1 Region Read/Write Monitoring

The Debug Assistant module monitors reads and writes performed by the HP CPU*n* bus in a specified address space, i.e., memory region. Whenever the bus reads from or writes to this address space, an interrupt is triggered. Refer to Table **??** in Chapter *2 High-Performance CPU [to be added later]*.

In this chapter, when the HP CPU*n* accesses the data space, the bus is referred to as the **Data bus**. When the HP CPU*n* accesses the AHB peripheral space, the bus is referred to as the **Peripheral bus**. The Debug

Assistant module can simultaneously monitor two address ranges on the Data bus, referred to as HP CPU*n* Data bus region 0 and HP CPU*n* Data bus region 1, and two address ranges on the Peripheral bus, referred to as HP CPU*n* Peripheral bus region 0 and HP CPU*n* Peripheral bus region 1. Region 0 and region 1 are configured as needed. Read and write monitoring for each region can be enabled independently.

7.3.2 SP Monitoring

The Debug Assistant module monitors the SP to prevent stack overflow or erroneous push/pop operations in the HP CPU*n*. When the HP CPU*n* SP exceeds the upper bound or falls below the lower bound of the monitored region, the module records the PC and generates an interrupt. The bounds are configured by software.

7.3.3 PC and SP Logging

The Debug Assistant module records the PC and SP values at the last reset of HP CPU*n*. Software can read these values for debugging.

7.3.4 Lockup Monitoring

When a CPU exception occurs, execution jumps to the exception handler. During the handling of the first exception, if another exception is triggered, the CPU enters a lockup. Refer to Section **?? ??** in Chapter 2 *High-Performance CPU [to be added later]*. When a lockup occurs on HP CPU*n*, the exception context generated by the later exception overwrites the context of the first exception, which makes debugging more difficult for software developers.

The Debug Assistant module records the exception cause, trap value, instruction PC, and current privilege level of HP CPU*n* at the first and second exceptions to assist in debugging.

7.3.5 CPU/DMA Bus Access Logging

The Debug Assistant module records write operations of the HP CPU*n* Data bus and the DMA bus in real time. When a write operation occurs or a specific value is written to a specified address space, the module records the bus type, address, PC (only when the write is performed by the HP CPU*n*), and other information. It then stores the data in the HP SRAM in a specific format. The specified address range must fall within the permitted memory regions of HP SRAM or external RAM.

7.4 Interrupts

The Debug Assistant can generate the BUS_MONITOR_INTR interrupt signal that is sent to the Interrupt Matrix. The following interrupt sources can generate this interrupt signal (Below, *n* = 0, 1).

- BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_INT: Triggered when HP CPU*n* reads in HP CPU*n* Data bus region 0.
- BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_INT: Triggered when HP CPU*n* writes in HP CPU*n* Data bus region 0.
- BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_INT: Triggered when HP CPU*n* reads in HP CPU*n* Data bus region 1.

- BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_WR_INT: Triggered when HP CPU*n* writes in HP CPU*n* Data bus region 1.
- BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_INT: Triggered when HP CPU*n* reads in HP CPU*n* Peripheral bus region 0.
- BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_INT: Triggered when HP CPU*n* writes in HP CPU*n* Peripheral bus region 0.
- BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_INT: Triggered when HP CPU*n* reads in HP CPU*n* Peripheral bus region 1.
- BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_INT: Triggered when HP CPU*n* writes in HP CPU*n* Peripheral bus region 1.
- BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_INT: Triggered when the HP CPU*n* SP falls below the lower bound of the monitored region.
- BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_INT: Triggered when the HP CPU*n* SP exceeds the upper bound of the monitored region.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 7.6 *Register Summary*.

7.5 Programming Procedures

7.5.1 Region Monitoring and SP Monitoring Configuration

The configuration process for region monitoring and SP monitoring is as follows (Below, *n* = 0, 1):

1. Configure the monitored region and SP bounds.
   - Set the HP CPU*n* Data bus region 0 with BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_MIN_REG and BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_MAX_REG.
   - Set the HP CPU*n* Data bus region 1 with BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_MIN_REG and BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_MAX_REG.
   - Set the HP CPU*n* Peripheral bus region 0 with BUS_MONITOR_CORE_*n*_AREA_PIF_0_MIN_REG and BUS_MONITOR_CORE_*n*_AREA_PIF_0_MAX_REG.
   - Set the HP CPU*n* Peripheral bus region 1 with BUS_MONITOR_CORE_*n*_AREA_PIF_1_MIN_REG and BUS_MONITOR_CORE_*n*_AREA_PIF_1_MAX_REG.
   - Set the HP CPU*n* SP bounds with BUS_MONITOR_CORE_*n*_SP_MIN_REG and BUS_MONITOR_CORE_*n*_SP_MAX_REG.
2. Set BUS_MONITOR_CORE_*n*_INTR_ENA_REG to enable interrupts for different monitoring modes.

3. Set BUS_MONITOR_CORE_*n*_MONTR_ENA_REG to enable the monitoring mode(s). You can enable multiple monitoring modes simultaneously.
4. Read BUS_MONITOR_CORE_*n*_INTR_RAW_REG to obtain the raw interrupt status of a monitoring mode.
5. Set BUS_MONITOR_CORE_*n*_INTR_CLR_REG to clear the interrupt for a monitoring mode.

For example, to monitor whether the HP CPU0 Data bus writes to the address space [A ~ B], enable monitoring in HP CPU0 Data bus region 0 or region 1. The following configuration uses region 0:

1. Set BUS_MONITOR_CORE_0_RCD_PDEBUGEN to 1 to enable the HP CPU0 to update the PC signals to the Debug Assistant module.
2. Set BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN_REG to Address A.
3. Set BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX_REG to Address B.
4. Set BUS_MONITOR_CORE_0_INTR_ENA_REG bit[1] to enable the interrupt for write operations by the HP CPU0 Data bus in region 0.
5. Set BUS_MONITOR_CORE_0_MONTR_ENA_REG bit[1] to enable monitoring of write operations by the HP CPU0 Data bus in region 0.
6. Configure the interrupt matrix to map BUS_MONITOR_INTR into the HP CPU0 interrupt (refer to Chapter 9 *Interrupt Matrix [to be added later]*).
7. After the interrupt is triggered:
   - Read BUS_MONITOR_CORE_0_INTR_RAW_REG to identify the interrupt source.
   - If the interrupt is triggered by the HP CPU0 region monitoring, read BUS_MONITOR_CORE_0_AREA_PC for the HP CPU0 PC value and BUS_MONITOR_CORE_0_AREA_SP for the HP CPU0 SP.
   - If the interrupt is triggered by SP monitoring, read BUS_MONITOR_CORE_0_SP_PC for the HP CPU0 PC value.
   - Write 1 to the corresponding bit of BUS_MONITOR_CORE_0_INTR_CLR_REG to clear the interrupt.

7.5.2 PC and SP Logging Configuration

Configure BUS_MONITOR_CORE_0_RCD_PDEBUGEN to 1 to enable the HP CPU0 to update the PC signals to the Debug Assistant module. If BUS_MONITOR_CORE_0_RCD_RECORDEN is also set to 1, BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG will record the HP CPU0's PC value, and BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG will record the HP CPU0's SP value. Otherwise, these two registers will retain their original values.

When the HP CPU0 resets, BUS_MONITOR_CORE_0_RCD_EN_REG will reset, while BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG and BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG will not. Therefore, these two registers will retain the PC and SP values during the HP CPU0 reset.

The same applies to HP CPU1.

7.5.3 Lockup Monitoring Configuration

Set BUS_MONITOR_CORE_0_TRACE_LOCKUP_ENA in BUS_MONITOR_CORE_0_MONTR_ENA_REG to enable HP CPU0 lockup monitoring mode.

When a lockup occurs on HP CPU0:

- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_CAUSE_1_REG for the exception cause of the first lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_TVAL_1_REG for the trap value of the first lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_IADDR_1_REG for the instruction PC at the first lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_PRIV_1_REG for the privilege level at the first lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_CAUSE_0_REG for the exception cause of the second lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_TVAL_0_REG for the trap value of the second lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_IADDR_0_REG for the instruction PC at the second lockup.
- Read BUS_MONITOR_CORE_0_TRACE_LOCKUP_PRIV_0_REG for the privilege level at the second lockup.

The registers that record lockup information are not reset when HP CPU0 is reset, so the recorded information can be retained for developers to read for debugging.

The same applies to HP CPU1.

7.5.4 CPU/DMA Bus Access Logging Configuration

7.5.4.1 Bus Access Logging 1 Configuration

Bus access logging 1 includes HP CPU*n* bus access logging (HP SRAM and external RAM monitoring) and DMA bus access logging (HP SRAM monitoring).

The base address of its configuration registers is the base address of Memory Access Monitor 1 (TCM_MEM_MONITOR). For the specific base address, refer to Table 3 in Chapter 3 *System and Memory [to be added later]*. Since Bus Access Logging 1 only uses the DMA_0, DMA_1, and DMA_2 channels, the DMA_3-related registers in the register list can be ignored. The configuration process is described below.

1. Configure the monitored address space:
    - Set MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to specify the monitored address space. For HP CPU*n*, the monitored address space should be in the range of HP SRAM (0x2F00_0000 ~ 0x2F07_FFFF) or external RAM (0x5000_0000 ~ 0x53FF_FFFF). For DMA, the monitored address space should be in the range of HP SRAM (0x2F00_0000 ~ 0x2F07_FFFF).
    - Configure the corresponding register fields of MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG and MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG to synchronize the address range in MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to the monitoring range of the corresponding HP CPU*n* or DMA bus.

2. Configure the monitoring mode with MEM_MONITOR_LOG_MODE:
   - 1: Write monitoring (detects bus write operations)
   - 2: Word monitoring (detects write operations of a specific word)
   - 4: Halfword monitoring (detects write operations of a specific halfword)
   - 8: Byte monitoring (detects write operations of a specific byte)
3. Configure the specific values to monitor:
   - In word monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
   - MEM_MONITOR_LOG_DATA_MASK_REG masks the byte(s) specified in MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if MEM_MONITOR_LOG_CHECK_DATA_REG is set to 0x01020304 and MEM_MONITOR_LOG_DATA_MASK_REG is set to 0x1, then any writes matching the 0x010203XX pattern will be recorded.
4. Configure the storage space for recorded data:
   - MEM_MONITOR_LOG_MEM_START_REG and MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x2F00_0000 ~ 0x2F07_FFFF.
   - Set MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the Debug Assistant module's permission to access HP SRAM. The Debug Assistant module can access HP SRAM only when the access permission is enabled. For more information, refer to Chapter 14 *Permission Control (PMS) [to be added later]*.
5. Configure the writing mode for recorded data as loop mode or non-loop mode:
   - In loop mode, writing to the specified address space occurs in loops. When writing reaches the end address, it returns to the starting address and continues, overwriting previously recorded data. Set MEM_MONITOR_LOG_MEM_LOOP_ENABLE to 1 to enable loop mode.
     For example, if there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access, after the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the data written by the 1st to 5th operations.
   - In non-loop mode, when writing reaches the end address, it stops at the end address and discards the remaining data. The previously recorded data will not be overwritten. Clear MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable non-loop mode.
     For example, if there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access, after the 5th operation writes to address 4, the 6th to 10th write operations will stop at address 4 and will not be performed. Therefore, address 0 ~ 4 will store the values written by the 1st to 5th operations, while the values of the 6th to 10th operations will be discarded.

6. Configure bus enable registers.
    - Enable HP CPU*n* bus access logging via MEM_MONITOR_LOG_CORE_ENA. Enable DMA_0 bus access logging via MEM_MONITOR_LOG_DMA_0_ENA. Enable DMA_1 bus access logging via MEM_MONITOR_LOG_DMA_1_ENA. Enable DMA_2 bus access logging via MEM_MONITOR_LOG_DMA_2_ENA. They can be enabled at the same time.

The Debug Assistant module first writes the recorded data to internal buffer A, then fetches the data from the buffer and writes it to the configured memory space. When monitored behaviors are triggered continuously, the generated recording packets may fill the buffer, preventing it from accepting new packets. In such cases, the module discards incoming packets that cannot be cached in time and, when the buffer is not full, buffers a LOST packet instead. The LOST packet contains information about the bus type of the discarded packets but not the number of discarded packets.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data consists of seven packet formats: HP CPU0 packet, HP CPU1 packet, DMA_0 channel 0 packet, DMA_0 channel 1 packet, DMA_1 channel 0 packet, DMA_2 channel 0 packet, and LOST packet. The packet formats are shown in Tables 7.5-1, 7.5-2, and 7.5-3.

Table 7.5-1. HP CPU*n* Packet Format

Bit[63:33]	Bit[32]	Bit[31:4]	Bit[3:1]	Bit[0]
pc_offset	anchored(1)	addr_offset	format	anchored(0)

Table 7.5-2. DMA Packet Format

Bit[31:5]	Bit[4]	Bit[3:1]	Bit[0]
addr_offset	reserved	format	anchored(0)

Table 7.5-3. LOST Packet Format

Bit[31:12]	Bit[11:4]	Bit[3:1]	Bit[0]
reserved	lost_source	format	anchored(0)

The data packet formats show that the HP CPU*n* packet is 64 bits, while the other packets are 32 bits each. These packets have the following fields:

- **format** – indicates the packet type:
    - 0: HP CPU0 packet
    - 1: HP CPU1 packet
    - 2: DMA_0 channel 0 packet
    - 3: DMA_0 channel 1 packet
    - 4: DMA_1 channel 0 packet
    - 5: DMA_2 channel 0 packet
    - 6: LOST packet

- **pc_offset** - indicates the offset of the HP CPU PC register at the time of access. Actual PC = pc_offset + 0x0000_0000.
- **addr_offset** - the address offset of a write operation. Actual address = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}.
- **lost_source** - indicates which packets were discarded when the LOST packet was generated.
  - Bit[5:4]:
    * 0: HP CPU0 packets were not discarded
    * 3: HP CPU0 packets were discarded
    * Other values: reserved
  - Bit[7:6]:
    * 0: HP CPU1 packets were not discarded
    * 3: HP CPU1 packets were discarded
    * Other values: reserved
  - Bit[8]:
    * 0: DMA_0 channel 0 packets were not discarded
    * 1: DMA_0 channel 0 packets were discarded
  - Bit[9]:
    * 0: DMA_0 channel 1 packets were not discarded
    * 1: DMA_0 channel 1 packets were discarded
  - Bit[10]:
    * 0: DMA_1 channel 0 packets were not discarded
    * 1: DMA_1 channel 0 packets were discarded
  - Bit[11]:
    * 0: DMA_2 channel 0 packets were not discarded
    * 1: DMA_2 channel 0 packets were discarded
- **anchored** - indicates the location of the 32 bits in the data packet:
  - 0: Lower 32 bits
  - 1: Higher 32 bits

The module's internal buffer is 32 bits wide. If HP CPU0, HP CPU1, DMA_0 channel 0, DMA_0 channel 1, DMA_1 channel 0, and DMA_2 channel 0 bus access logging are enabled simultaneously and data is recorded at the same time, the HP CPU0 packets are first stored in the buffer, followed by the HP CPU1 packets, then the DMA_0 channel 0, DMA_0 channel 1, and DMA_1 channel 0 packets, and finally the DMA_2 channel 0 packets. The Debug Assistant module automatically fetches the buffered data and stores it in the specified memory space with a 32-bit data width.

In loop mode, data looping multiple times in the storage memory may cause residual data, which can interfere with packet parsing. For example, the lower 32 bits of an HP CPU*n* packet may be overwritten, leaving higher 32-bit residual data. Therefore, users must filter out possible residual data when determining the starting position of the first valid packet. Read MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to identify the starting position of the packet, then check the anchored bit value of the packet. If it is 0, retain the data. If it is 1, discard it.

The packet parsing process is described below.

- Read MEM_MONITOR_LOG_MEM_FULL_FLAG to determine whether there is a data overflow.
  - If no, the address space to read is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is enabled, the address space is MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG and MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is not enabled, the address space is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address, 32 bits at a time.

After completing packet parsing, clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG to 1.

7.5.4.2 Bus Access Logging 2 Configuration

Bus access logging 2 includes DMA bus access logging (external RAM monitoring).

The base address of its configuration registers is the base address of Memory Access Monitor 2 (PSRAM_MEM_MONITOR). For the specific base address, refer to Table 3 in Chapter 3 *System and Memory [to be added later]*. Since Bus Access Logging 2 only uses the DMA_0 channel, the registers related to DMA_1, DMA_2, and DMA_3 in the register list can be ignored. The configuration process is described below.

1. Configure monitored address space:
   - Set MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to specify the monitored address space, which should range from 0x5000_0000 to 0x53FF_FFFF.
   - Configure the corresponding register fields of MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG and MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG to synchronize the address range in MEM_MONITOR_LOG_MIN_REG and MEM_MONITOR_LOG_MAX_REG to the monitoring range of the corresponding DMA bus.
2. Configure the monitoring mode with MEM_MONITOR_LOG_MODE:
   - 1: Write monitoring (detects bus write operations)
   - 2: Word monitoring (detects write operations of a specific word)
   - 4: Halfword monitoring (detects write operations of a specific halfword)
   - 8: Byte monitoring (detects write operations of a specific byte)
3. Configure the specific values to monitor:

- In word monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
- In halfword monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
- In byte monitoring mode, MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
- Use MEM_MONITOR_LOG_DATA_MASK_REG to mask the byte specified in MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, if MEM_MONITOR_LOG_CHECK_DATA_REG is set to 0x01020304 and MEM_MONITOR_LOG_DATA_MASK_REG is set to 0x1, then any writes matching the 0x010203XX pattern will be recorded.

4. Configure the storage space for recorded data:
   - MEM_MONITOR_LOG_MEM_START_REG and MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data, which must be in the range of 0x2F00_0000 ~ 0x2F07_FFFF.
   - Set MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the Debug Assistant module's permission to access the HP SRAM. The Debug Assistant module can access the HP SRAM only when access permission is enabled. For more information, refer to Chapter 14 *Permission Control (PMS) [to be added later]*.
5. Configure the writing mode for recorded data as loop mode or non-loop mode:
   - In loop mode, writing to the specified address space occurs in loops. When writing reaches the end address, it returns to the starting address and continues, overwriting previously recorded data. Set MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
   - In non-loop mode, when writing reaches the end address, it stops and discards the remaining data, without overwriting previously recorded data. Clear MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable non-loop mode.
   - See examples in Section 7.5.4.1 > step 5.
6. Set MEM_MONITOR_LOG_DMA_0_ENA to enable DMA_0 channel group bus access logging.

The Debug Assistant module first writes the DMA recorded data to the internal buffer B, then fetches the data from the buffer and writes it to the configured memory space. When monitored behaviors are triggered continuously, the generated recording packets may fill the buffer, preventing it from accepting new packets. In such cases, the module discards incoming packets that cannot be cached in time and, when the buffer is not full, buffers a DMA LOST packet instead. The DMA LOST packet contains information about the bus type of the discarded packets, but not the number of discarded packets.

When bus access logging is finished, read the recorded data from memory for decoding. The recorded data is in two packet formats: DMA_0 packet (corresponding to DMA_0 Data bus) and DMA LOST packet. The packet formats are shown in Table 7.5-4 and 7.5-5.

Table 7.5-4. DMA_0 Packet Format

Bit[31:6]	Bit[5:2]	Bit[1]	Bit[0]
addr_offset	reserved	format	anchored(0)

Table 7.5-5. DMA LOST Packet Format

Bit[31:3]	Bit[2]	Bit[1]	Bit[0]
reserved	lost_source	format	anchored(0)

The data packet formats show that the DMA_0 and DMA LOST packets are both 32 bits. These packets contain the following fields:

- **format** – indicates the packet type:
  - 0: DMA_0 packet
  - 1: DMA LOST packet
- **addr_offset** – indicates the address offset of a write operation. Actual address = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}.
- **lost_source** – indicates which packets were discarded when the DMA LOST packet was generated:
  - 0: DMA_0 packets were not discarded
  - 1: DMA_0 packets were discarded
- **anchored** – indicates the location of the 32 bits in the data packet:
  - 0: Lower 32 bits
  - 1: Higher 32 bits

The internal buffer of the module is 32 bits wide. When the bus access logging of the DMA_0 channel groups is enabled and the record data is generated, the DMA_0 data packets are buffered. The Debug Assistant module automatically fetches the buffered data and stores it in 32-bit width in the specified memory space.

The packet parsing process is described below.

- Read MEM_MONITOR_LOG_MEM_FULL_FLAG to determine whether there is a data overflow.
  - If no, the address space to read is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is enabled, the address space is MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG and MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG − 4.
  - If yes and loop mode is not enabled, the address space is MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address. Read 32 bits at a time.

After packet parsing is complete, clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG to 1.

7.6 Register Summary

The addresses of **bus logging configuration registers** in Section 7.6.1 are offsets (relative addresses) from the base addresses of **TCM_MEM_MONITOR** and **PSRAM_MEM_MONITOR**. The addresses of other registers in Section 7.6.2 are offsets (relative addresses) from the base address of the **BUS_MONITOR**. All base addresses are provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

7.6.1 Bus Logging Configuration Registers

Name	Description	Address	Access
Bus logging configuration registers			
MEM_MONITOR_LOG_SETTING_REG	Configures bus access logging	0x0000	R/W
MEM_MONITOR_LOG_SETTING1_REG	Configures bus access logging	0x0004	R/W
MEM_MONITOR_LOG_CHECK_DATA_REG	Configures data to be monitored for bus access logging	0x0008	R/W
MEM_MONITOR_LOG_DATA_MASK_REG	Configures data mask for bus access logging	0x000C	R/W
MEM_MONITOR_LOG_MIN_REG	Configures the monitored lower address for bus access logging	0x0010	R/W
MEM_MONITOR_LOG_MAX_REG	Configures the monitored upper address for bus access logging	0x0014	R/W
MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG	Configures whether to update the monitored address space for HP CPU bus access logging	0x0018	WT
MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG	Configures whether to update the monitored address space for DMA bus access logging	0x001C	WT
MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the storage memory for recorded data	0x0020	R/W
MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the storage memory for recorded data	0x0024	R/W
MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x0028	RO
MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write with the starting address for the recorded data	0x002C	WT
MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Represents logging buffer overflow status register	0x0030	varies
Clock gating control register			
MEM_MONITOR_CLOCK_GATE_REG	Clock gating control register	0x0034	R/W
Version control register			

GoBack

Name	Description	Address	Access
MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

7.6.2 Other Registers

Name	Description	Address	Access
Monitor configuration registers (Below, *n*=0, 1)			
BUS_MONITOR_CORE_*n*_MONTR_ENA_REG	Configures whether to enable HP CPU*n* monitoring	0x0000+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MIN_REG	Configures lower boundary address of region 0 monitored on HP CPU*n* Data bus	0x0010+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MAX_REG	Configures upper boundary address of region 0 monitored on HP CPU*n* Data bus	0x0014+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MIN_REG	Configures lower boundary address of region 1 monitored on HP CPU*n* Data bus	0x0018+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MAX_REG	Configures upper boundary address of region 1 monitored on HP CPU*n* Data bus	0x001C+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_PIF_0_MIN_REG	Configures lower boundary address of region 0 monitored on HP CPU*n* Peripheral bus	0x0020+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_PIF_0_MAX_REG	Configures upper boundary address of region 0 monitored on HP CPU*n* Peripheral bus	0x0024+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_PIF_1_MIN_REG	Configures lower boundary address of region 1 monitored on HP CPU*n* Peripheral bus	0x0028+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_PIF_1_MAX_REG	Configures upper boundary address of region 1 monitored on HP CPU*n* Peripheral bus	0x002C+0x88×n	R/W
BUS_MONITOR_CORE_*n*_AREA_PC_REG	Represents the PC value when HP CPU*n* region monitor is triggered	0x0030+0x88×n	RO
BUS_MONITOR_CORE_*n*_AREA_SP_REG	Represents the SP value when HP CPU*n* region monitor is triggered	0x0034+0x88×n	RO

Name	Description	Address	Access
BUS_MONITOR_CORE_*n*_SP_MIN_REG	Configures HP CPU*n* SP monitoring lower boundary address	0x0038+0x88×n	R/W
BUS_MONITOR_CORE_*n*_SP_MAX_REG	Configures HP CPU*n* SP monitoring upper boundary address	0x003C+0x88×n	R/W
BUS_MONITOR_CORE_*n*_SP_PC_REG	Represents the HP CPU*n* PC value when HP CPU*n* SP monitor is triggered	0x0040+0x88×n	RO
Interrupt configuration register (Below, *n*=0, 1)			
BUS_MONITOR_CORE_*n*_INTR_RAW_REG	HP CPU*n* monitor raw interrupt status register	0x0004+0x88×n	RO
BUS_MONITOR_CORE_*n*_INTR_ENA_REG	HP CPU*n* monitor interrupt enable register	0x0008+0x88×n	R/W
BUS_MONITOR_CORE_*n*_INTR_CLR_REG	HP CPU*n* monitor interrupt clear register	0x000C+0x88×n	WT
PC and SP recording configuration register			
BUS_MONITOR_CORE_0_RCD_EN_REG	HP CPU (core0) PC and SP logging enable register	0x0044	R/W
BUS_MONITOR_CORE_1_RCD_EN_REG	HP CPU (core1) PC and SP logging enable register	0x00CC	R/W
PC and SP recording status register			
BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG	HP CPU (core0) PC logging register	0x0048	RO
BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG	HP CPU (core0) SP logging register	0x004C	RO
BUS_MONITOR_CORE_1_RCD_PDEBUGPC_REG	HP CPU (core1) PC logging register	0x00D0	RO
BUS_MONITOR_CORE_1_RCD_PDEBUGSP_REG	HP CPU (core1) SP logging register	0x00D4	RO
CPU status registers			
BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU0 enters exception	0x0060	RO
BUS_MONITOR_CORE_0_DEBUG_MODE_REG	HP CPU0 debug mode status register	0x0064	RO
BUS_MONITOR_CORE_1_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU1 enters exception	0x00E8	RO
BUS_MONITOR_CORE_1_DEBUG_MODE_REG	HP CPU1 debug mode status register	0x00EC	RO
Lockup recording status register (Below, *n*=0, 1)			
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_CAUSE_0_REG	HP CPU*n* TRACE second lockup cause logging register	0x0068+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_TVAL_0_REG	HP CPU*n* TRACE second lockup tval logging register	0x006C+0x88×n	RO

Name	Description	Address	Access
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_IADDR_0_REG	HP CPU*n* TRACE second lockup iaddr logging register	0x0070+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_PRIV_0_REG	HP CPU*n* TRACE second lockup priv logging register	0x0074+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_CAUSE_1_REG	HP CPU*n* TRACE first lockup cause logging register	0x0078+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_TVAL_1_REG	HP CPU*n* TRACE first lockup tval logging register	0x007C+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_IADDR_1_REG	HP CPU*n* TRACE first lockup iaddr logging register	0x0080+0x88×n	RO
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_PRIV_1_REG	HP CPU*n* TRACE first lockup priv logging register	0x0084+0x88×n	RO
Clock control Registers			
BUS_MONITOR_CLOCK_GATE_REG	Clock control register	0x0114	R/W
Version control register			
BUS_MONITOR_DATE_REG	Version control register	0x03FC	R/W

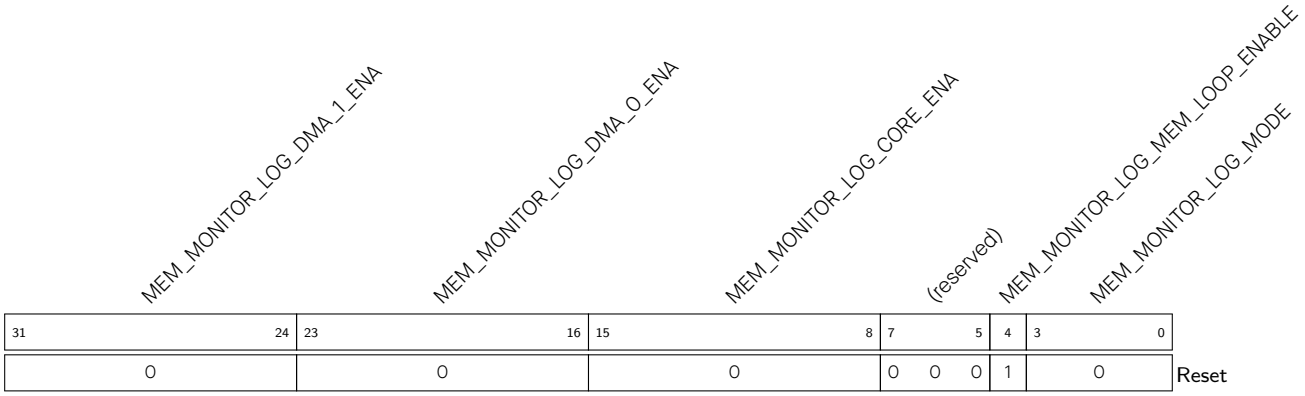
7.7 Registers

The addresses of **bus logging configuration registers** in Section 7.7.1 are relative to the **TCM_MEM_MONITOR** base address and the **PSRAM_MEM_MONITOR** base address. The addresses of other registers in Section 7.7.2 are relative to the **BUS_MONITOR** base address. All base addresses are provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

7.7.1 Bus Logging Configuration Registers

Register 7.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)



MEM_MONITOR_LOG_MODE Configures monitoring modes.

1: Configures write monitoring.
2: Configures word monitoring.
4: Configures halfword monitoring.
8: Configures byte monitoring.
Other values: Invalid
(R/W)

MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for recorded data.

0: Non-loop mode
1: Loop mode
(R/W)

MEM_MONITOR_LOG_CORE_ENA Configures whether to enable CPU bus access logging.

0: Disable
1: Enable
Other values: Invalid
(R/W)

Continued on the next page...

Register 7.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)

Continued from the previous page...

MEM_MONITOR_LOG_DMA_0_ENA Configures whether to enable DMA_0 bus access logging.

0: Disable

1: Enable

Other values: Invalid (R/W)

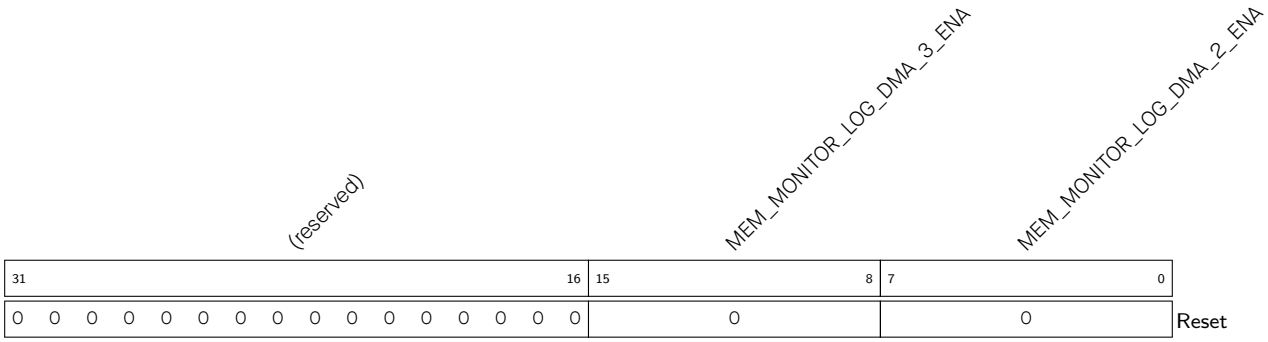
MEM_MONITOR_LOG_DMA_1_ENA Configures whether to enable DMA_1 bus access logging.

0: Disable

1: Enable

Other values: Invalid(R/W)

Register 7.2. MEM_MONITOR_LOG_SETTING1_REG (0x0004)



MEM_MONITOR_LOG_DMA_2_ENA Configures whether to enable DMA_2 bus access logging.

0: Disable

1: Enable

Other values: Invalid (R/W)

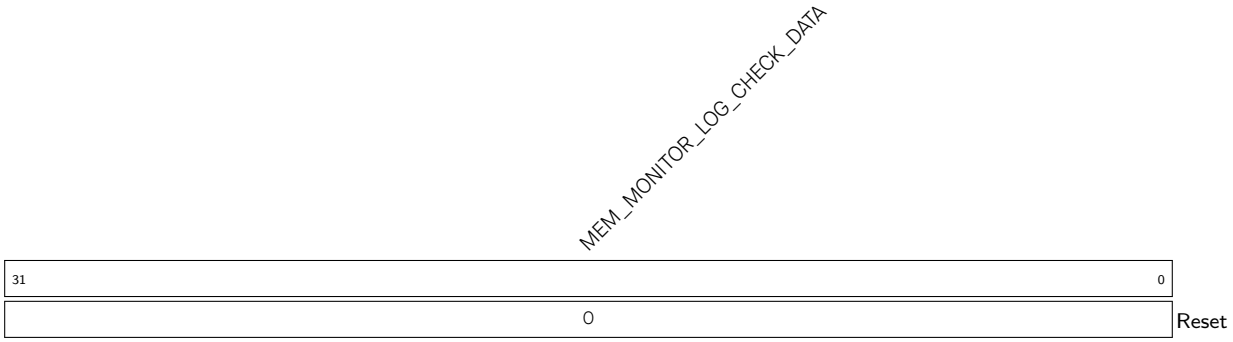
MEM_MONITOR_LOG_DMA_3_ENA Configures whether to enable DMA_3 bus access logging.

0: Disable

1: Enable

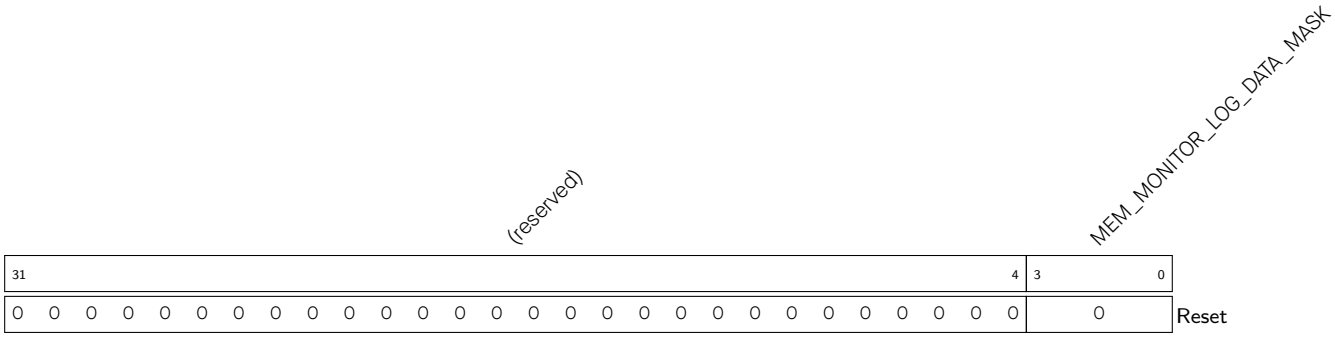
Other values: Invalid (R/W)

Register 7.3. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 7.4. MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

1: Configures to mask the least significant byte.

2: Configures to mask the second least significant byte.

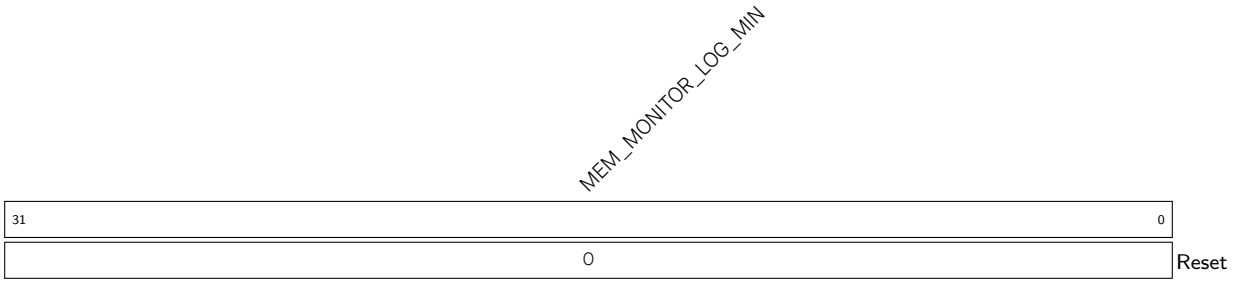
4: Configures to mask the second most significant byte.

8: Configures to mask the most significant byte.

Other values: Invalid

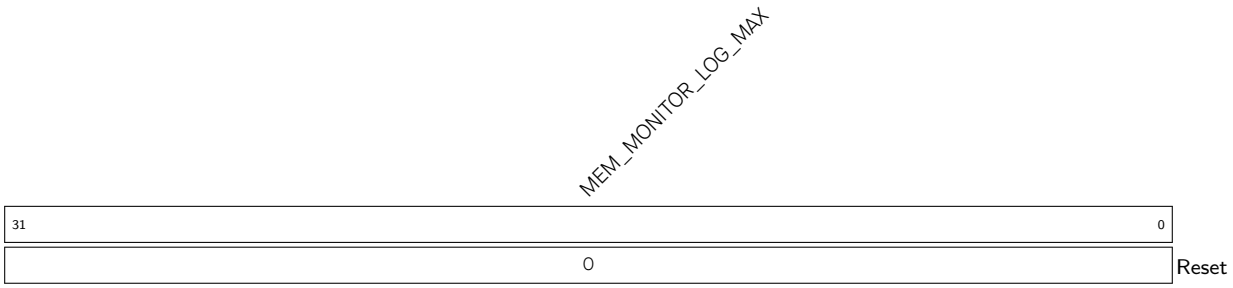
(R/W)

Register 7.5. MEM_MONITOR_LOG_MIN_REG (0x0010)



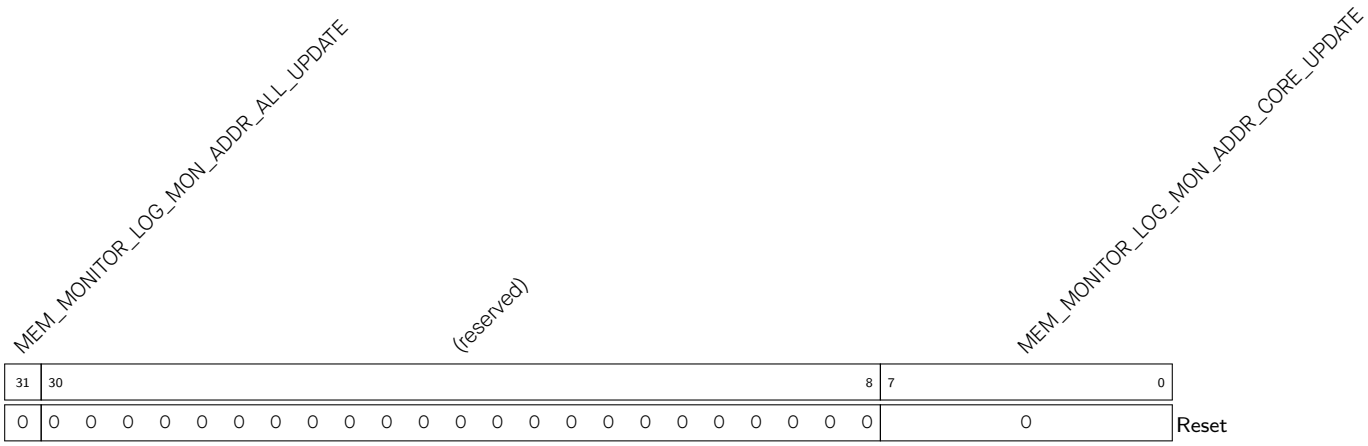
MEM_MONITOR_LOG_MIN Configures the lower bound address of the monitored address space. (R/W)

Register 7.6. MEM_MONITOR_LOG_MAX_REG (0x0014)



MEM_MONITOR_LOG_MAX Configures the upper bound address of the monitored address space. (R/W)

Register 7.7. MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG (0x0018)



MEM_MONITOR_LOG_MON_ADDR_CORE_UPDATE Configures whether to update the monitored address space of the HP CPU bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG as the monitored address space of the HP CPU bus.

0: Not update

1: Update

Other values: Invalid

(WT)

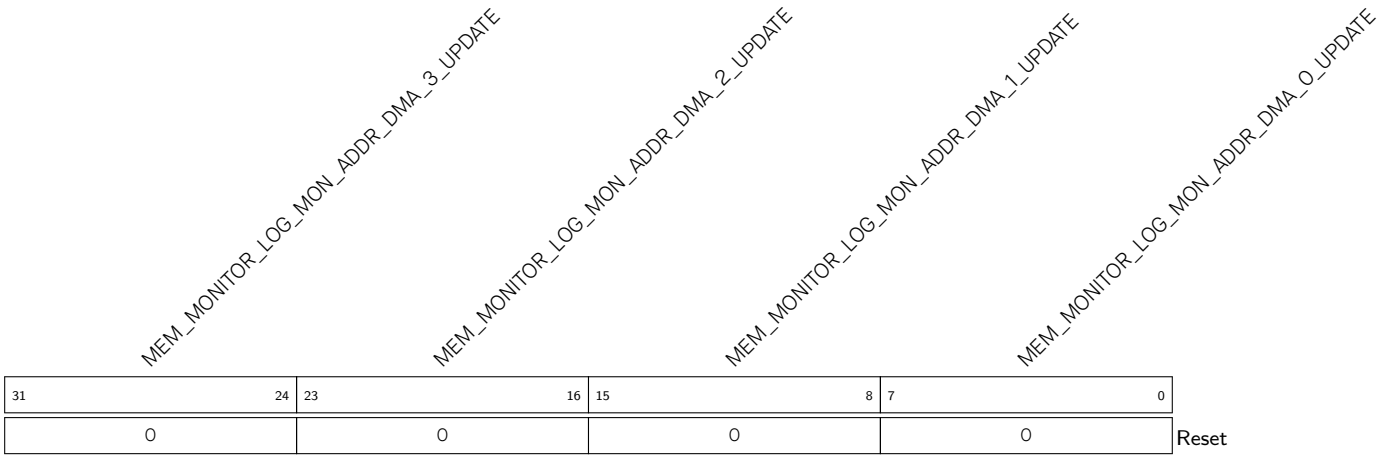
MEM_MONITOR_LOG_MON_ADDR_ALL_UPDATE Configures whether to update the monitored address space of all masters from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

(WT)

Register 7.8. MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG (0x001C)



MEM_MONITOR_LOG_MON_ADDR_DMA_0_UPDATE Configures whether to update the monitored address space of the DMA_0 bus as the address space of from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_1_UPDATE Configures whether to update the monitored address space of the DMA_1 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_2_UPDATE Configures whether to update the monitored address space of the DMA_2 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

0: Not update

1: Update

Other values: Invalid

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_3_UPDATE Configures whether to update the monitored address space of the DMA_3 bus as the address space from MEM_MONITOR_LOG_MIN_REG to MEM_MONITOR_LOG_MAX_REG.

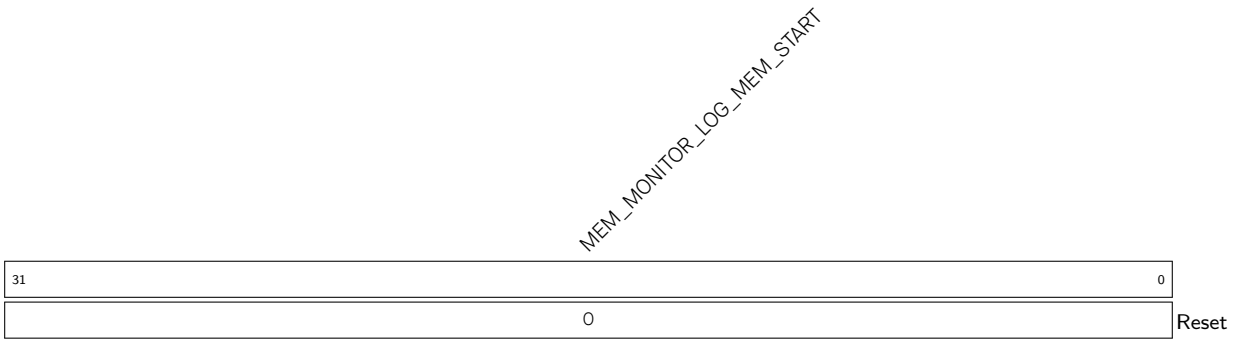
0: Not update

1: Update

Other values: Invalid

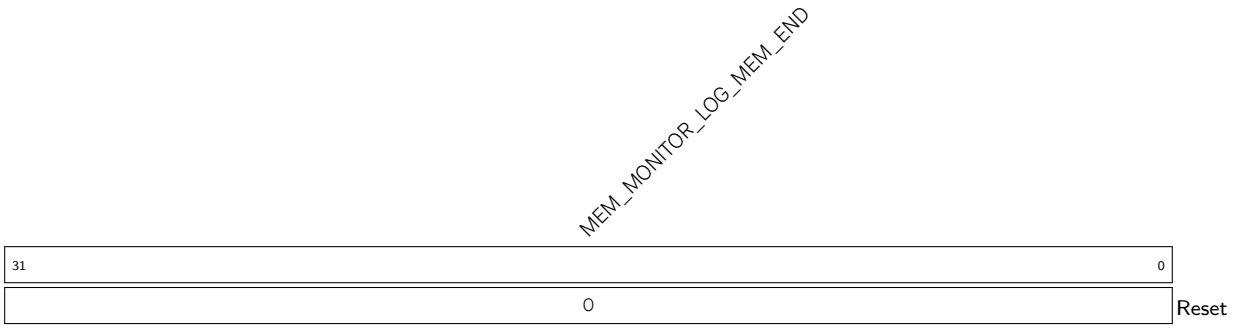
(WT)

Register 7.9. MEM_MONITOR_LOG_MEM_START_REG (0x0020)



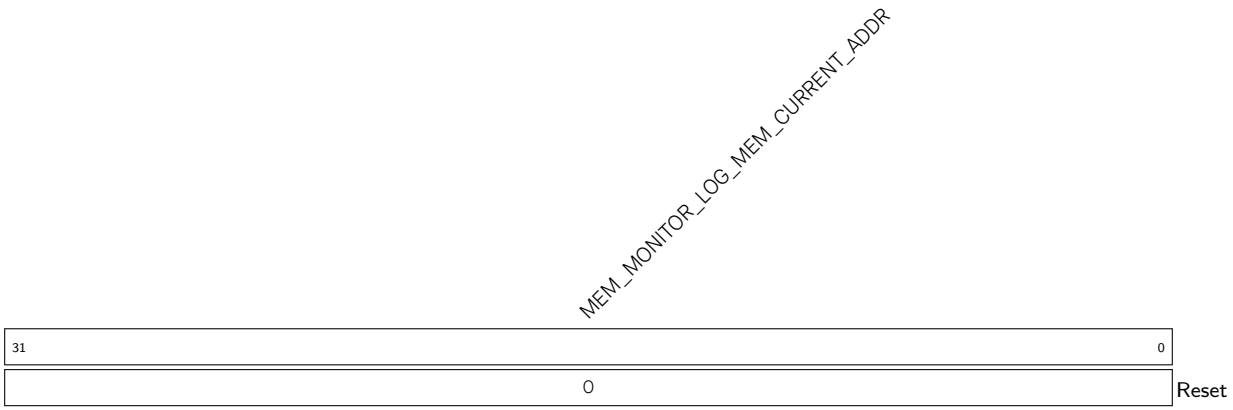
MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for recorded data. (R/W)

Register 7.10. MEM_MONITOR_LOG_MEM_END_REG (0x0024)



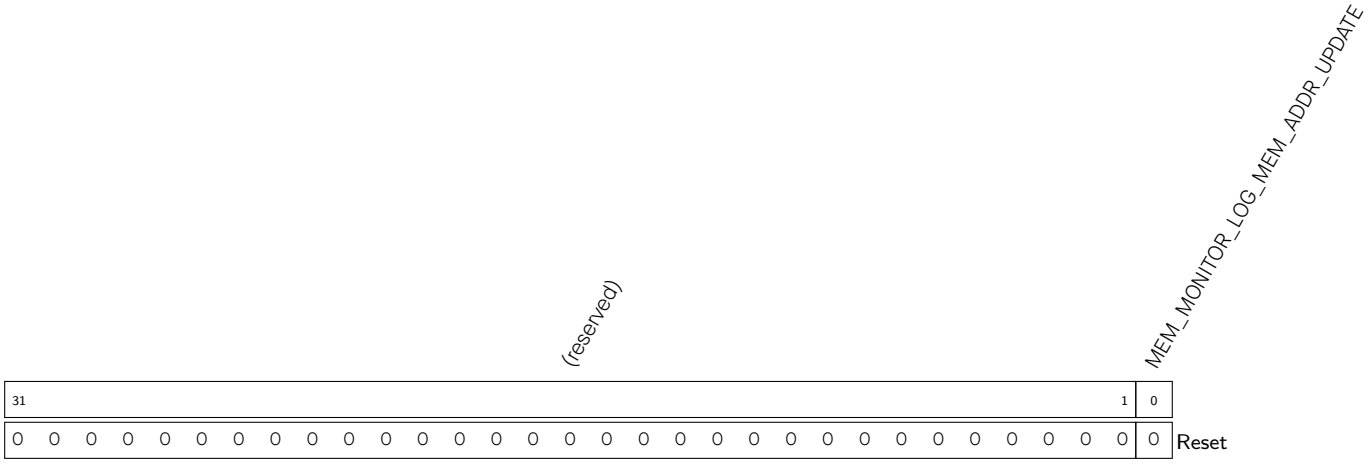
MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for recorded data. (R/W)

Register 7.11. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0028)



MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 7.12. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x002C)



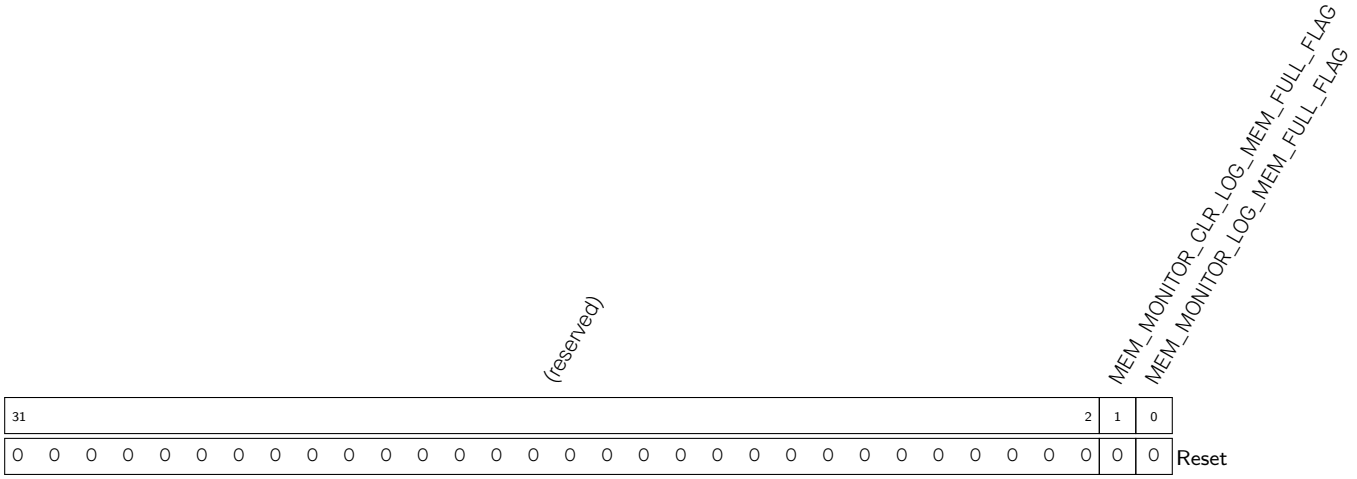
MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to MEM_MONITOR_LOG_MEM_START_REG.

0: Not update

1: Update

(WT)

Register 7.13. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0030)



MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space.

0: Not Overflow

1: Overflow

(RO)

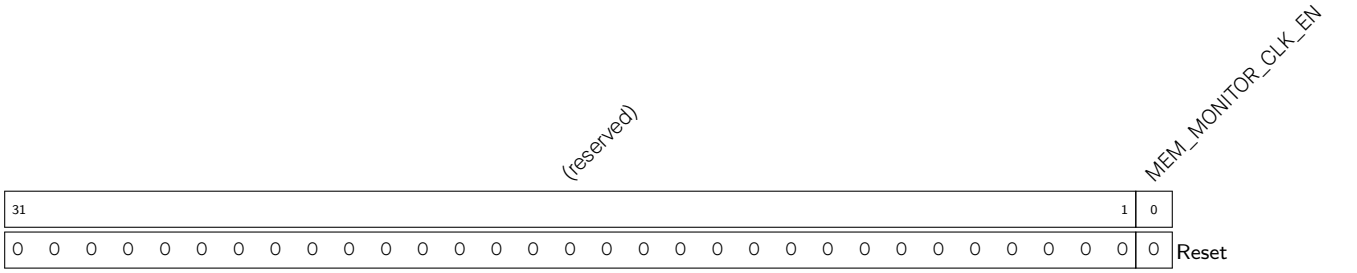
MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear

1: Clear

(WT)

Register 7.14. MEM_MONITOR_CLOCK_GATE_REG (0x0034)



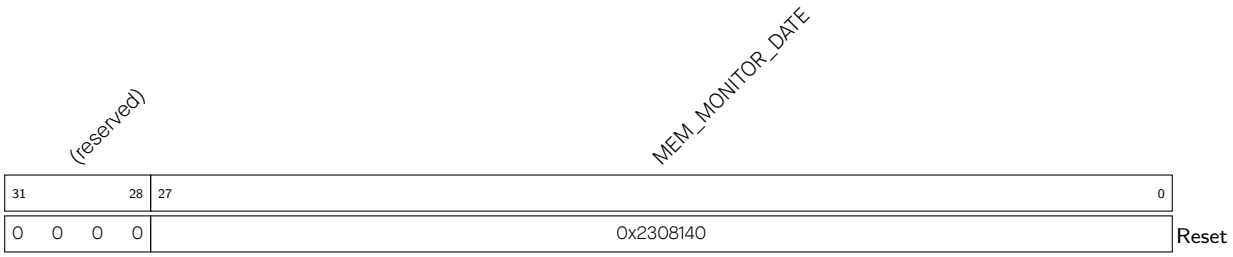
MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

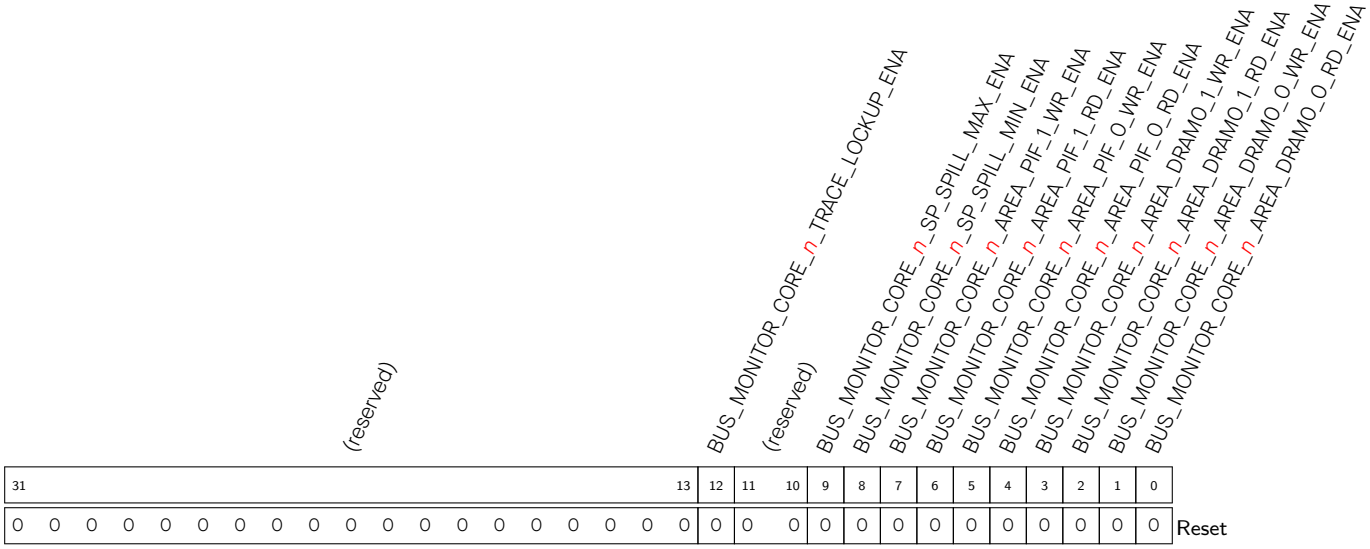
Register 7.15. MEM_MONITOR_DATE_REG (0x03FC)



MEM_MONITOR_DATE Version control register. (R/W)

7.7.2 Other Registers

Register 7.16. BUS_MONITOR_CORE_*n*_MONTR_ENA_REG (*n*=0,1) (0x0000+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_ENA Configures whether to monitor read operations in region 0 by the Data bus.

0: Not monitor

1: Monitor

(R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_ENA Configures whether to monitor write operations in region 0 by the Data bus.

0: Not monitor

1: Monitor

(R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_ENA Configures whether to monitor read operations in region 1 by the Data bus.

0: Not Monitor

1: Monitor

(R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_WR_ENA Configures whether to monitor write operations in region 1 by the Data bus.

0: Not Monitor

1: Monitor

(R/W)

BUS_MONITOR_CORE_*n*_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 7.16. BUS_MONITOR_CORE_*n*_MONTR_ENA_REG (0x0000+0x88×n)

Continued from the previous page...

BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in region 0 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in region 1 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

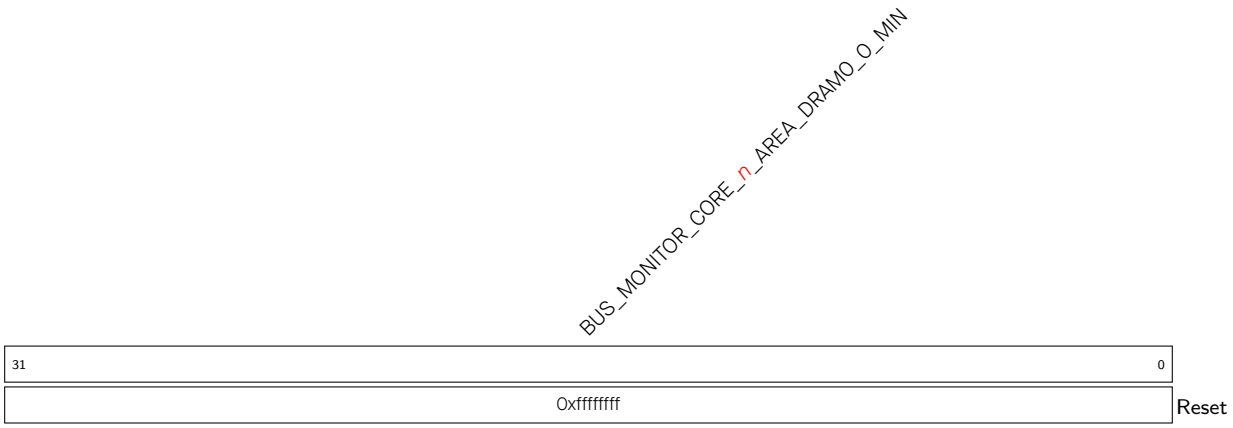
BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in region 1 by the Peripheral bus.
0: Not Monitor
1: Monitor
(R/W)

BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_ENA Configures whether to monitor SP falling below the lower bound address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_ENA Configures whether to monitor SP exceeding the upper bound address of SP monitored region.
0: Not Monitor
1: Monitor
(R/W)

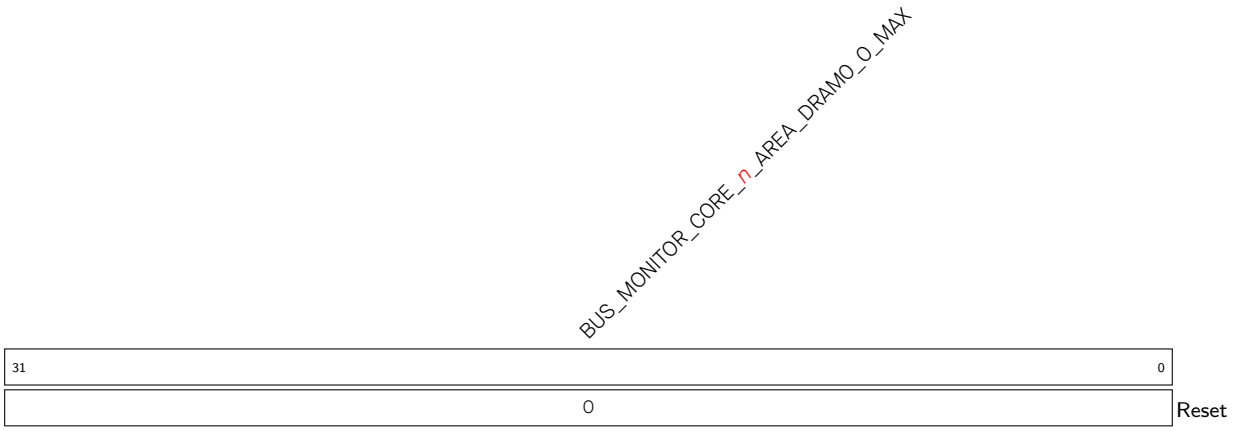
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_ENA Configures whether to monitor trace lockup conditions.
0: Not Monitor
1: Monitor
(R/W)

Register 7.17. BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MIN_REG (*n*=0,1) (0x0010+0x88×n)



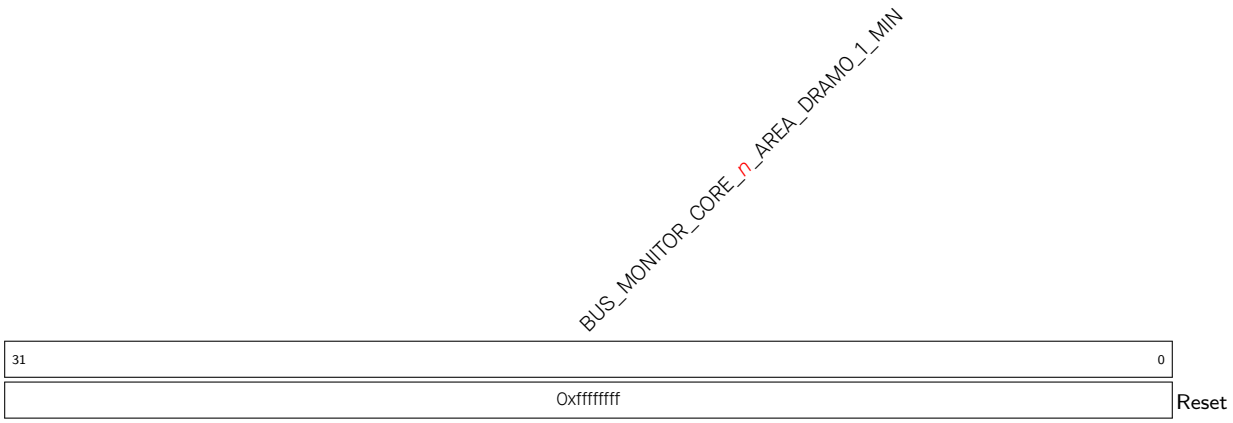
BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MIN Configures the lower bound address of Data bus region 0. (R/W)

Register 7.18. BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MAX_REG (*n*=0,1) (0x0014+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_MAX Configures the upper bound address of Data bus region 0. (R/W)

Register 7.19. BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MIN_REG (*n*=0,1) (0x0018+0x88×n)



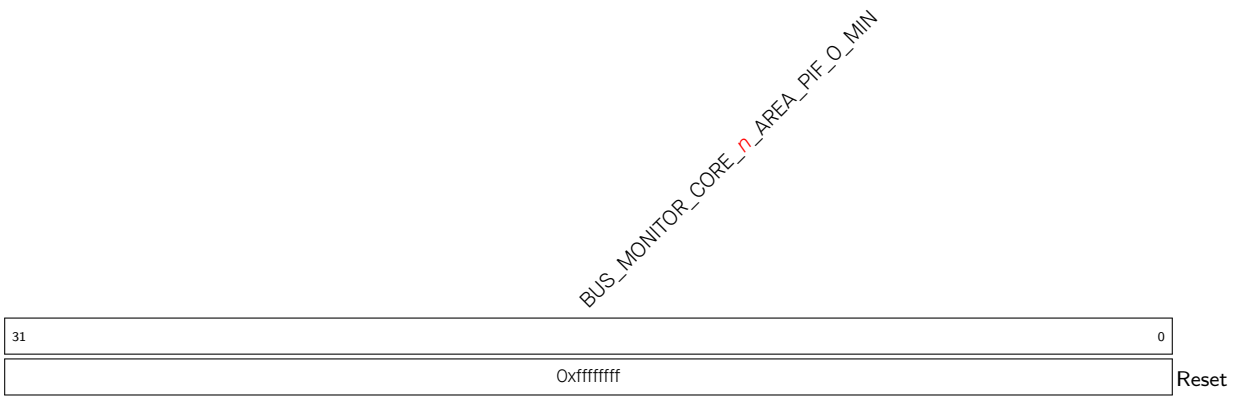
BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MIN Configures the lower bound address of Data bus region 1. (R/W)

Register 7.20. BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MAX_REG (*n*=0,1) (0x001C+0x88×n)



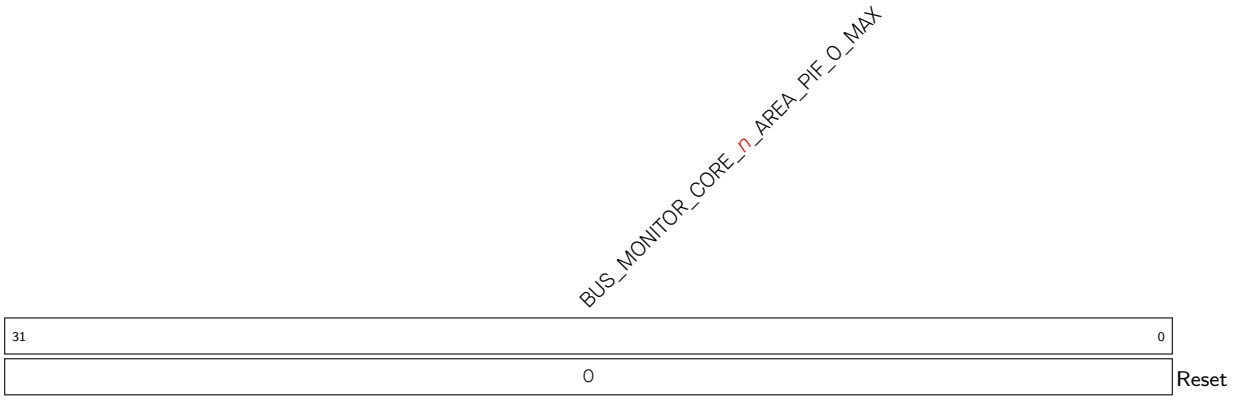
BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_MAX Configures the upper bound address of Data bus region 1. (R/W)

Register 7.21. BUS_MONITOR_CORE_*n*_AREA_PIF_0_MIN_REG (*n*=0,1) (0x0020+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_PIF_0_MIN Configures the lower bound address of Peripheral bus region 0. (R/W)

Register 7.22. BUS_MONITOR_CORE_*n*_AREA_PIF_0_MAX_REG (*n*=0,1) (0x0024+0x88×n)



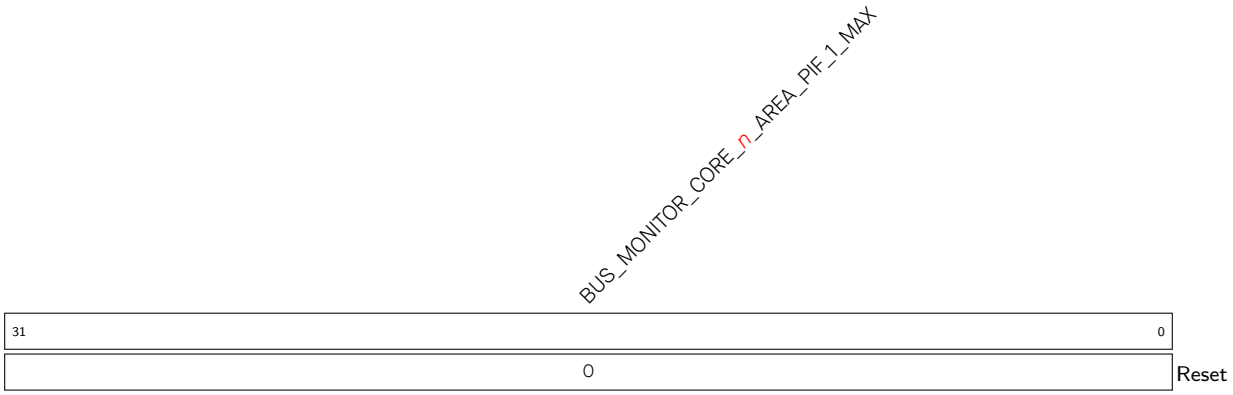
BUS_MONITOR_CORE_*n*_AREA_PIF_0_MAX Configures the upper bound address of Peripheral bus region 0. (R/W)

Register 7.23. BUS_MONITOR_CORE_*n*_AREA_PIF_1_MIN_REG (*n*=0,1) (0x0028+0x88×n)



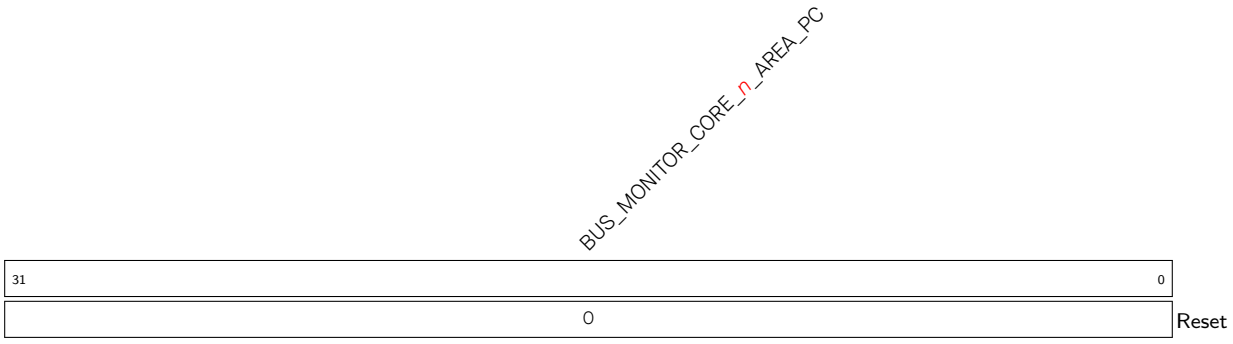
BUS_MONITOR_CORE_*n*_AREA_PIF_1_MIN Configures the lower bound address of Peripheral bus region 1. (R/W)

Register 7.24. BUS_MONITOR_CORE_*n*_AREA_PIF_1_MAX_REG (*n*=0,1) (0x002C+0x88×n)



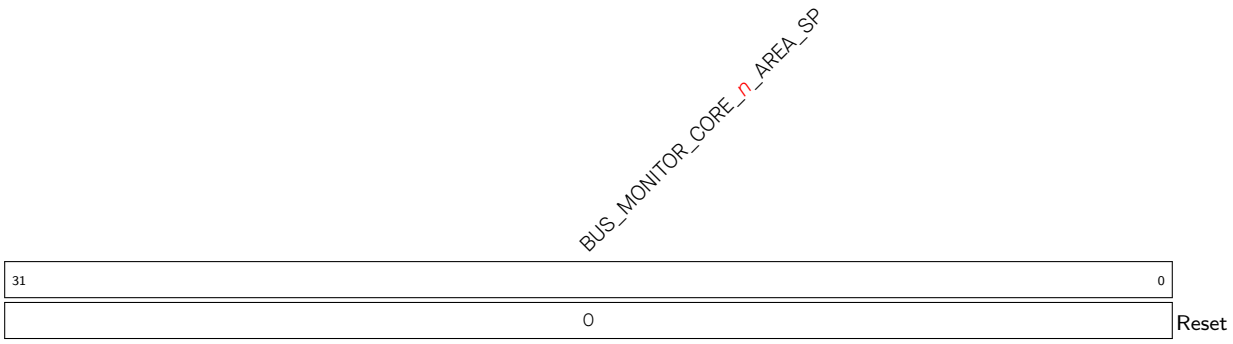
BUS_MONITOR_CORE_*n*_AREA_PIF_1_MAX Configures the upper bound address of Peripheral bus region 1. (R/W)

Register 7.25. BUS_MONITOR_CORE_*n*_AREA_PC_REG (*n*=0,1) (0x0030+0x88×n)



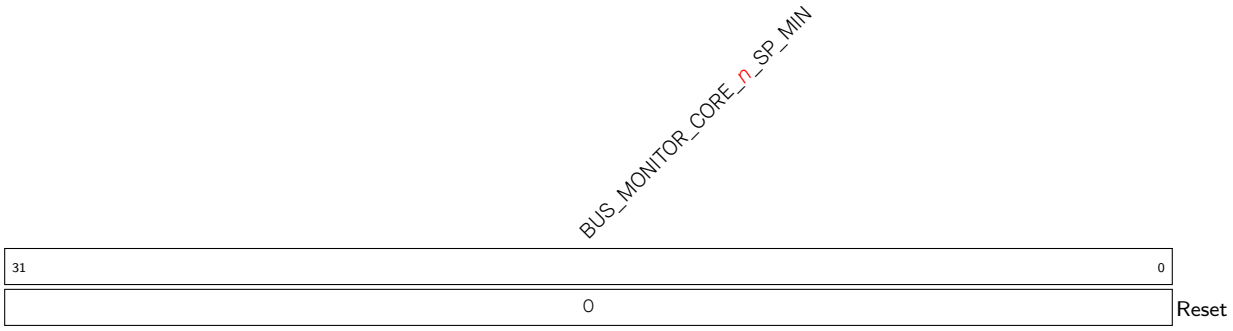
BUS_MONITOR_CORE_*n*_AREA_PC Represents the PC value when an interrupt is triggered during region monitoring. (RO)

Register 7.26. BUS_MONITOR_CORE_*n*_AREA_SP_REG (*n*=0,1) (0x0034+0x88×n)



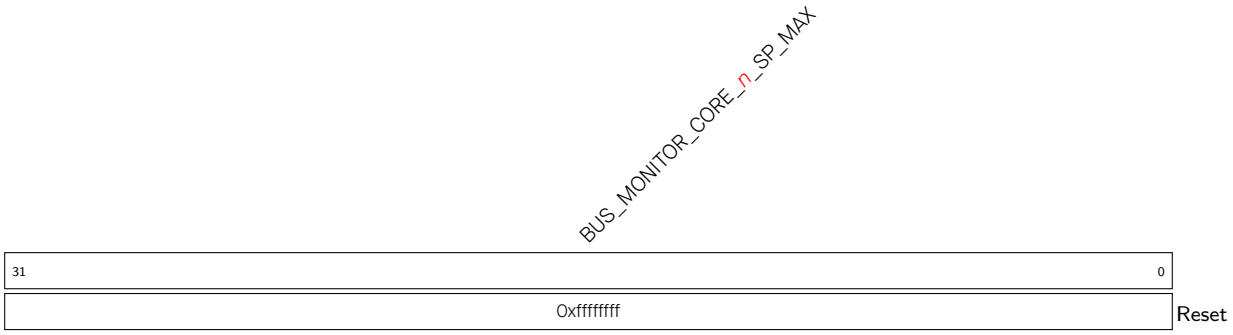
BUS_MONITOR_CORE_*n*_AREA_SP Represents the SP value when an interrupt is triggered during region monitoring. (RO)

Register 7.27. BUS_MONITOR_CORE_*n*_SP_MIN_REG (*n*=0,1) (0x0038+0x88×n)



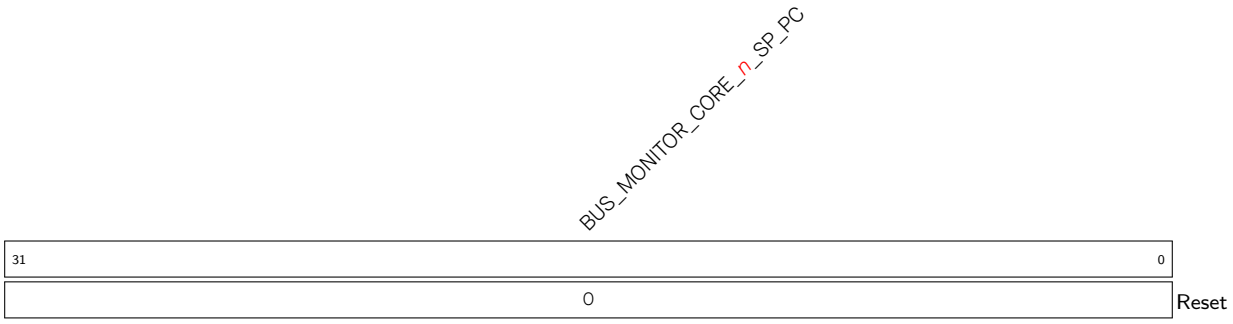
BUS_MONITOR_CORE_*n*_SP_MIN Configures the lower bound address of SP. (R/W)

Register 7.28. BUS_MONITOR_CORE_*n*_SP_MAX_REG (*n*=0,1) (0x003C+0x88×n)



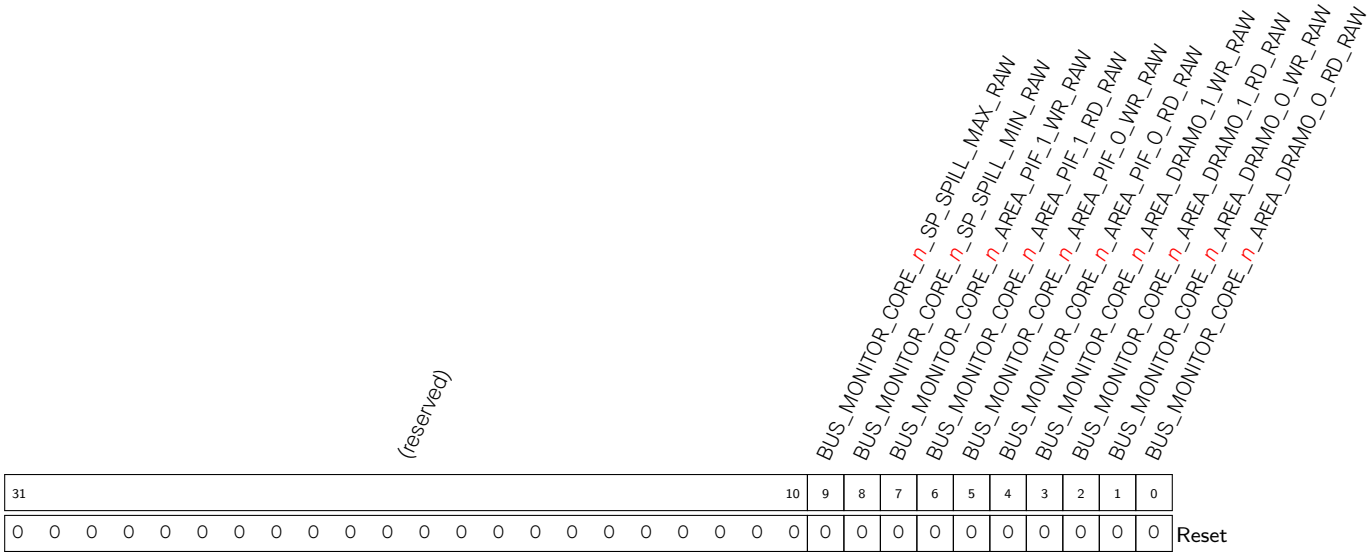
BUS_MONITOR_CORE_*n*_SP_MAX Configures the upper bound address of SP. (R/W)

Register 7.29. BUS_MONITOR_CORE_*n*_SP_PC_REG (*n*=0,1) (0x0040+0x88×n)



BUS_MONITOR_CORE_*n*_SP_PC Represents the HP CPU*n* PC value when SP monitoring is triggered. (RO)

Register 7.30. BUS_MONITOR_CORE_*n*_INTR_RAW_REG (*n*=0,1) (0x0004+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_WR_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_WR_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_INT. (RO)

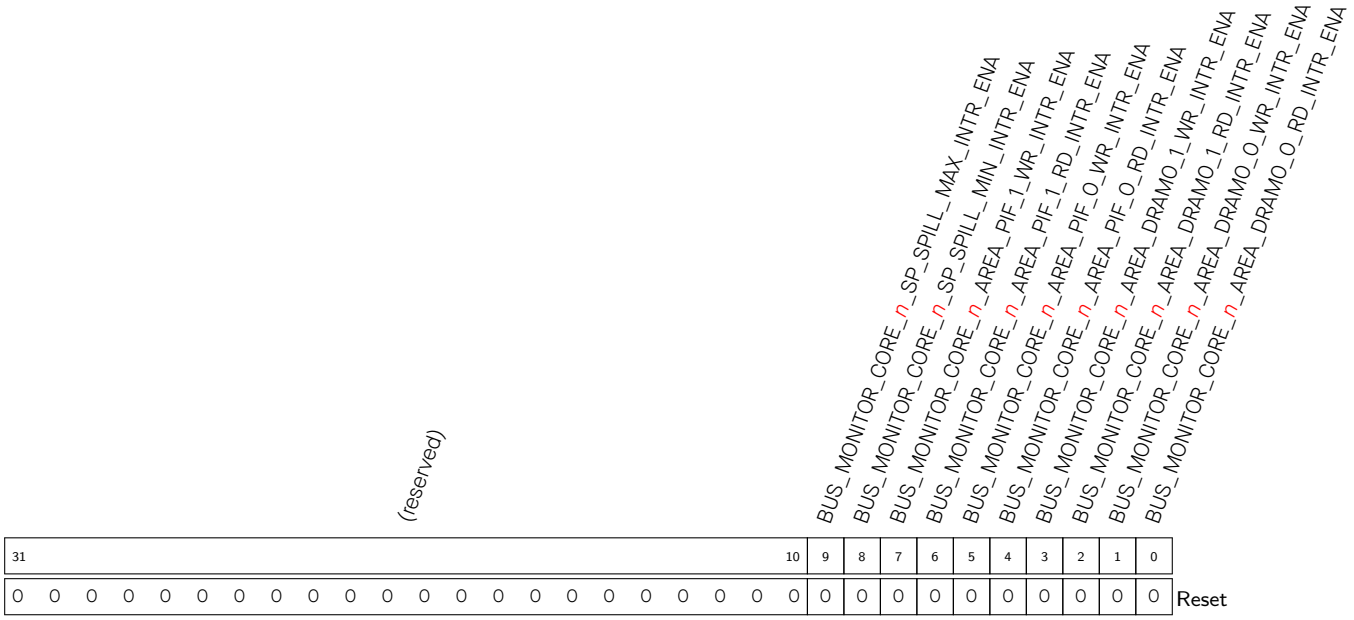
BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_INT. (RO)

BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_INT. (RO)

BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_INT. (RO)

BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_RAW The raw interrupt status of BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_INT. (RO)

Register 7.31. BUS_MONITOR_CORE_*n*_INTR_ENA_REG (*n*=0,1) (0x0008+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_RD_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_RD_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_WR_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_DRAMO_0_WR_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_RD_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_RD_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_WR_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_DRAMO_1_WR_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_INT. (R/W)

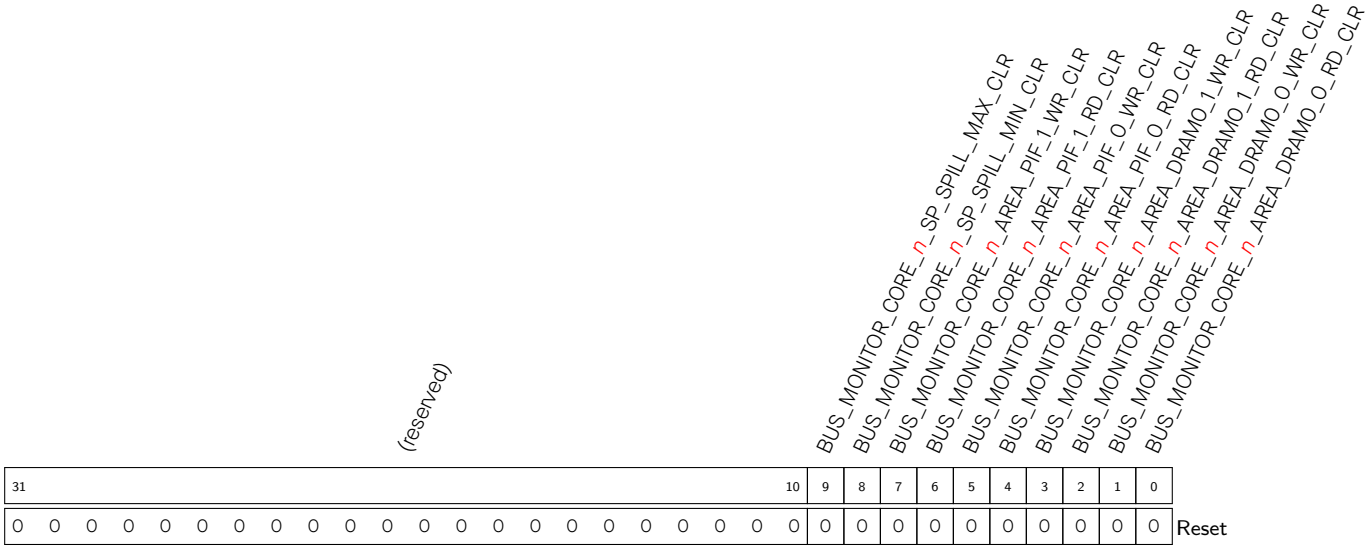
BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_INT. (R/W)

BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_INT. (R/W)

BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_INT. (R/W)

BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_INTR_ENA Write 1 to enable
BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_INT. (R/W)

Register 7.32. BUS_MONITOR_CORE_*n*_INTR_CLR_REG (*n*=0,1) (0x000C+0x88×n)



BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_RD_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_DRAM0_0_WR_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_RD_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_WR_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_DRAM0_1_WR_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_PIF_0_RD_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_PIF_0_WR_INT. (WT)

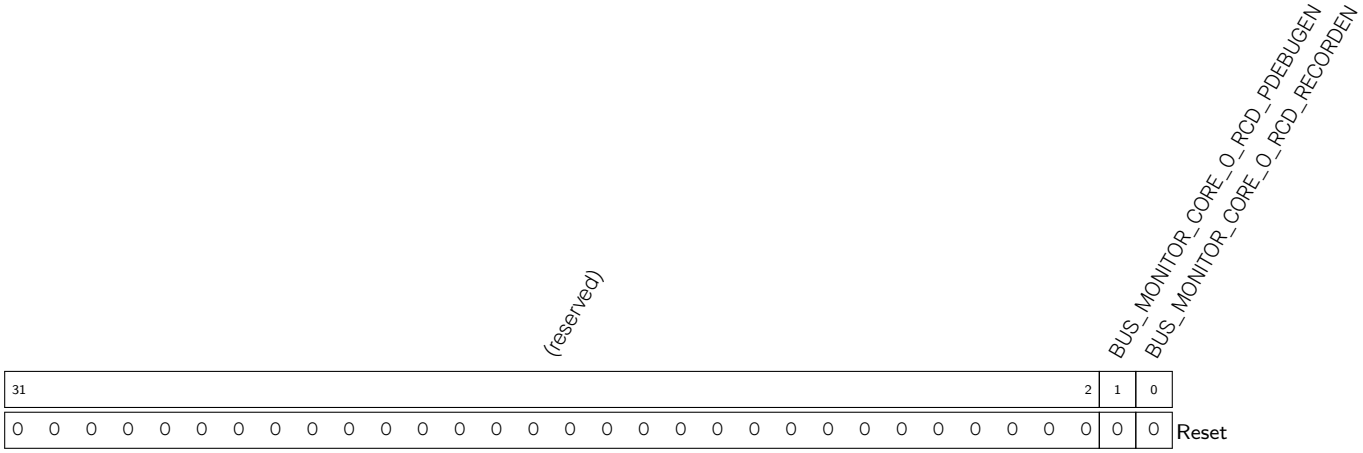
BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_PIF_1_RD_INT. (WT)

BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_AREA_PIF_1_WR_INT. (WT)

BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_SP_SPILL_MIN_INT. (WT)

BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_CLR Write 1 to clear BUS_MONITOR_CORE_*n*_SP_SPILL_MAX_INT. (WT)

Register 7.33. BUS_MONITOR_CORE_0_RCD_EN_REG (0x0044)



BUS_MONITOR_CORE_0_RCD_RECORDEN Configures whether to enable PC and SP logging.

0: Disable

1: BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG starts to record HP CPU0 PC in real time, and BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG starts to record HP CPU0 SP in real time

(R/W)

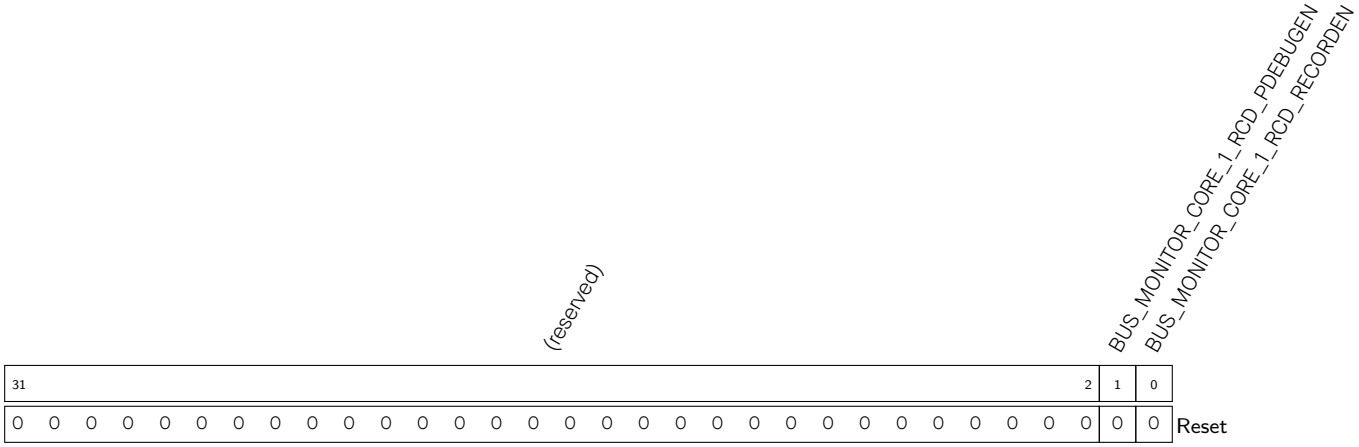
BUS_MONITOR_CORE_0_RCD_PDEBUGEN Configures whether to enable HP CPU0 debugging.

0: Disable

1: HP CPU0 outputs HP CPU0 PC

(R/W)

Register 7.34. BUS_MONITOR_CORE_1_RCD_EN_REG (0x00CC)



BUS_MONITOR_CORE_1_RCD_RECORDEN Configures whether to enable PC and SP logging.

0: Disable

1: BUS_MONITOR_CORE_1_RCD_PDEBUGPC_REG starts to record HP CPU1 PC in real time, and BUS_MONITOR_CORE_1_RCD_PDEBUGSP_REG starts to record HP CPU1 SP in real time

(R/W)

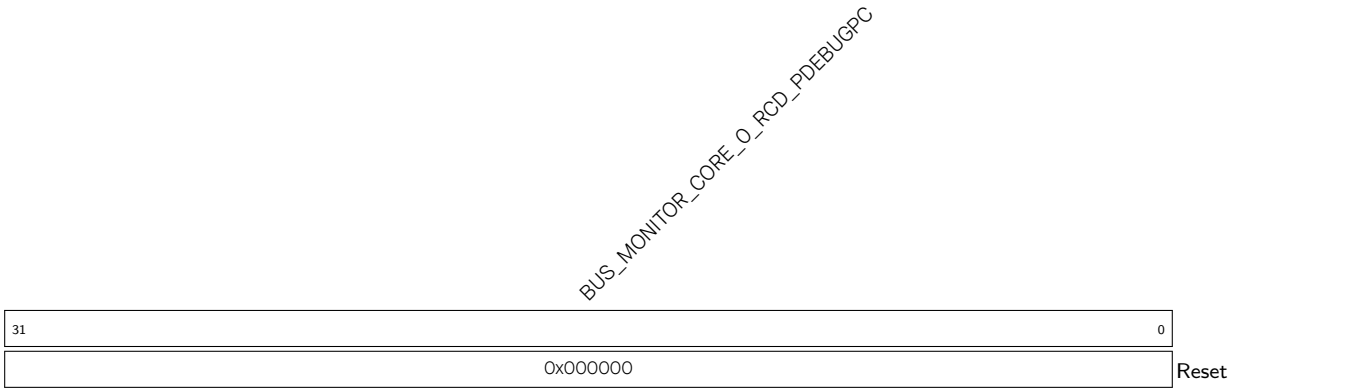
BUS_MONITOR_CORE_1_RCD_PDEBUGEN Configures whether to enable HP CPU1 debugging.

0: Disable

1: HP CPU1 outputs HP CPU1 PC

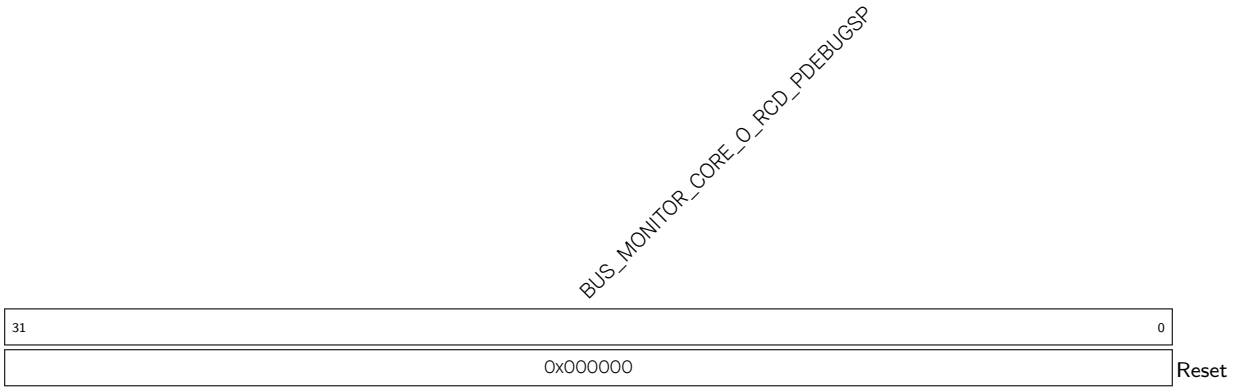
(R/W)

Register 7.35. BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG (0x0048)



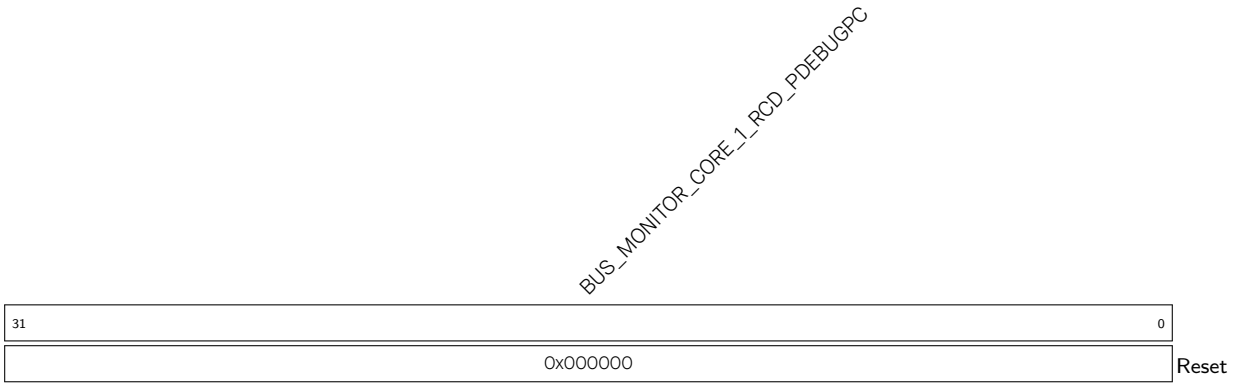
BUS_MONITOR_CORE_0_RCD_PDEBUGPC Represents the PC value of HP CPU0 at reset. (RO)

Register 7.36. BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG (0x004C)



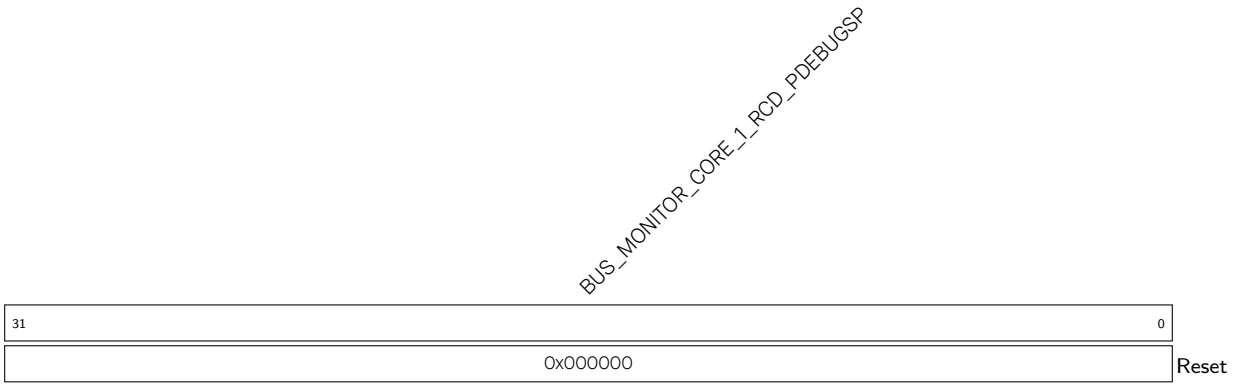
BUS_MONITOR_CORE_0_RCD_PDEBUGSP Represents HP CPU0 SP. (RO)

Register 7.37. BUS_MONITOR_CORE_1_RCD_PDEBUGPC_REG (0x00D0)



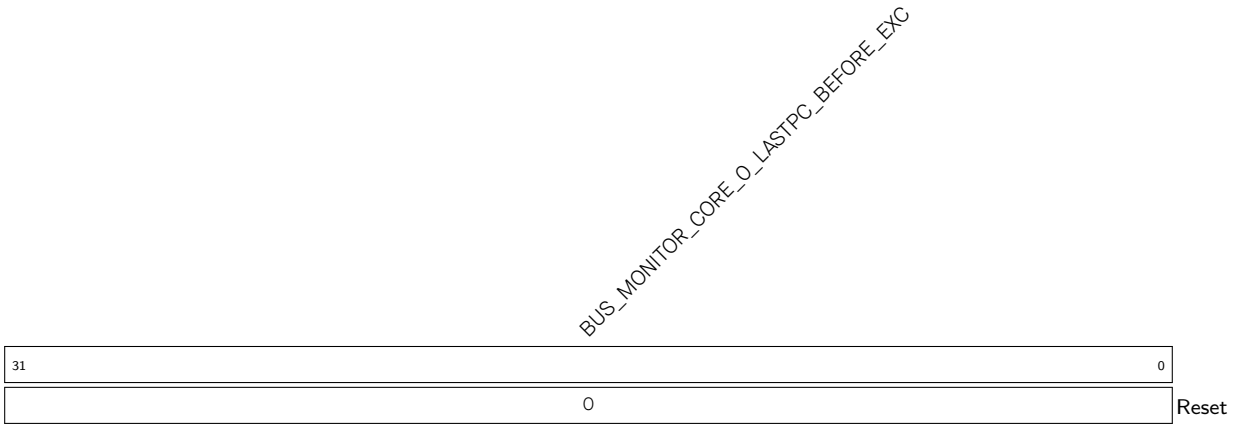
BUS_MONITOR_CORE_1_RCD_PDEBUGPC Represents the PC value at HP CPU1 reset. (RO)

Register 7.38. BUS_MONITOR_CORE_1_RCD_PDEBUGSP_REG (0x00D4)



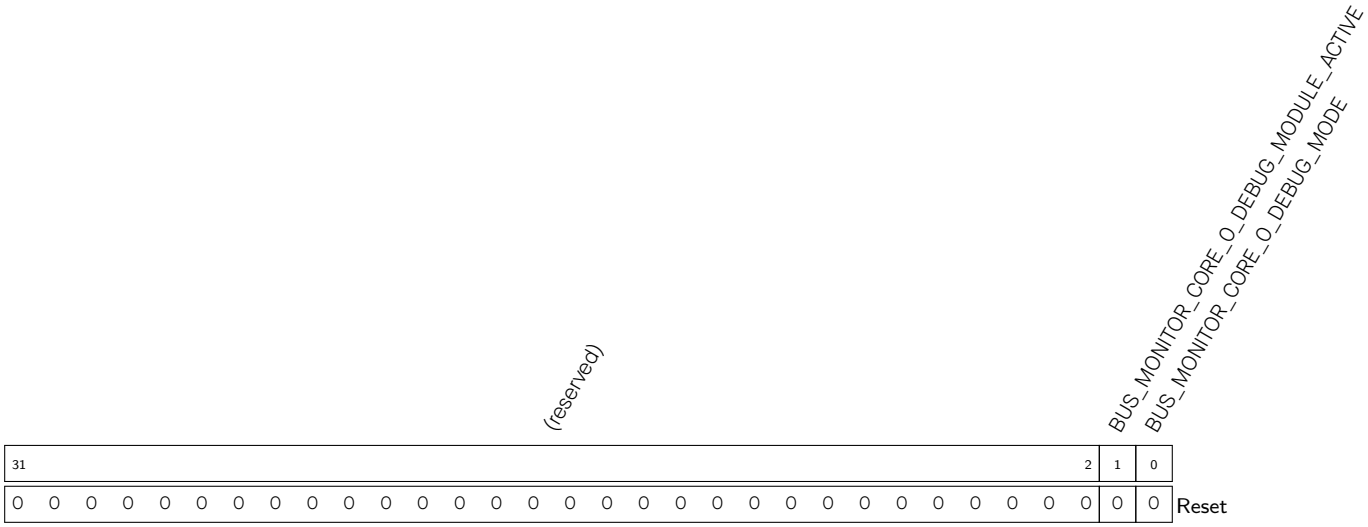
BUS_MONITOR_CORE_1_RCD_PDEBUGSP Represents HP CPU1 SP. (RO)

Register 7.39. BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0060)



BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXC Represents the HP CPU0 PC of the last command before the HP CPU0 enters exception. (RO)

Register 7.40. BUS_MONITOR_CORE_0_DEBUG_MODE_REG (0x0064)



BUS_MONITOR_CORE_0_DEBUG_MODE Represents whether RISC-V CPU0 (HP CPU0) is in debug mode.

0: Not in debug mode

1: In debug mode

(RO)

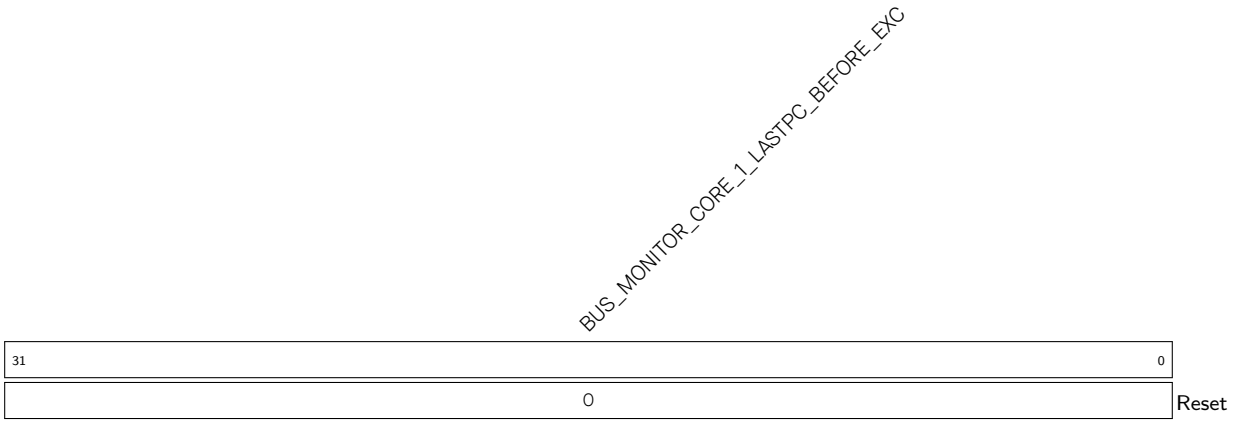
BUS_MONITOR_CORE_0_DEBUG_MODULE_ACTIVE Represents the status of the RISC-V CPU0 (HP CPU0) debug module.

1: Active status

Other: Inactive status

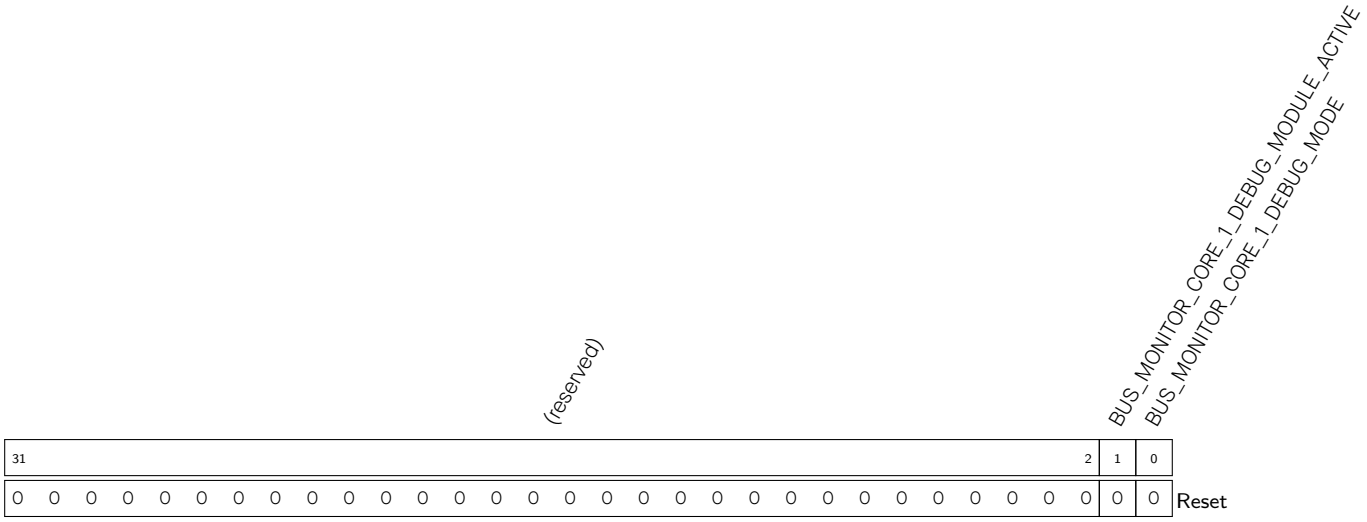
(RO)

Register 7.41. BUS_MONITOR_CORE_1_LASTPC_BEFORE_EXCEPTION_REG (0x00E8)



BUS_MONITOR_CORE_1_LASTPC_BEFORE_EXC Represents the HP CPU1 PC of the last command before the HP CPU1 enters exception. (RO)

Register 7.42. BUS_MONITOR_CORE_1_DEBUG_MODE_REG (0x00EC)



BUS_MONITOR_CORE_1_DEBUG_MODE Represents whether RISC-V CPU1 (HP CPU1) is in debug mode.

0: Not in debug mode

1: In debug mode

(RO)

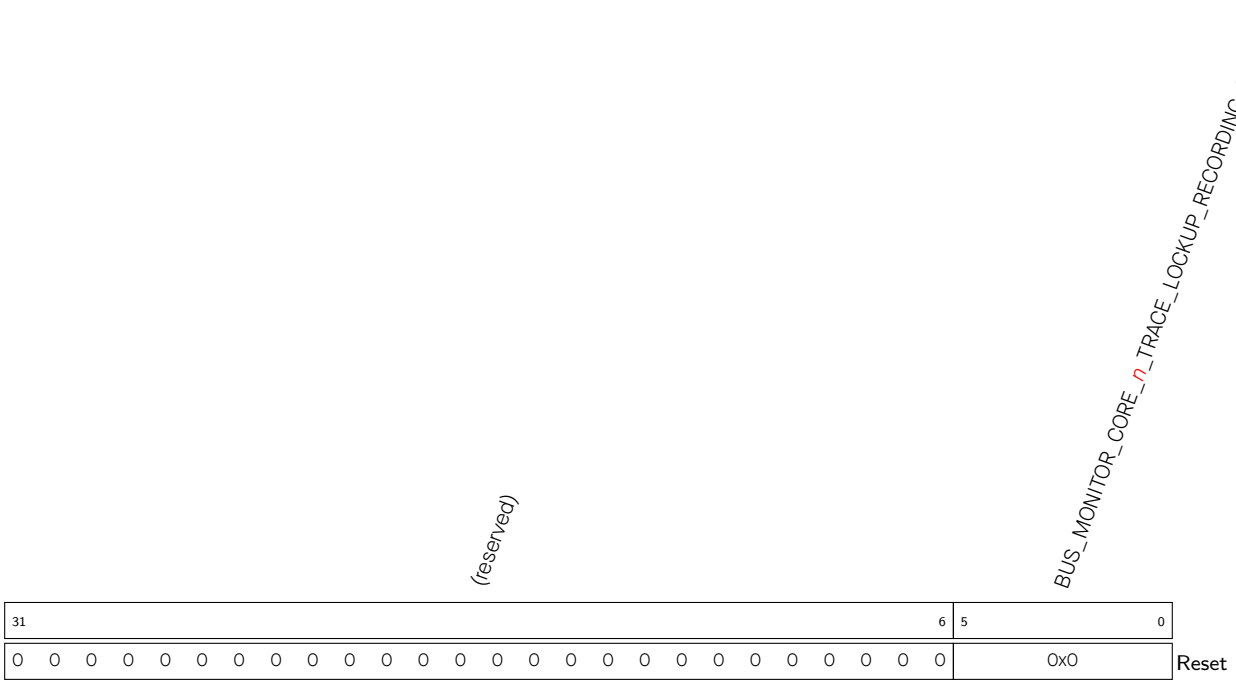
BUS_MONITOR_CORE_1_DEBUG_MODULE_ACTIVE Represents the status of the RISC-V CPU1 (HP CPU1) debug module.

1: Active status

Other: Inactive status

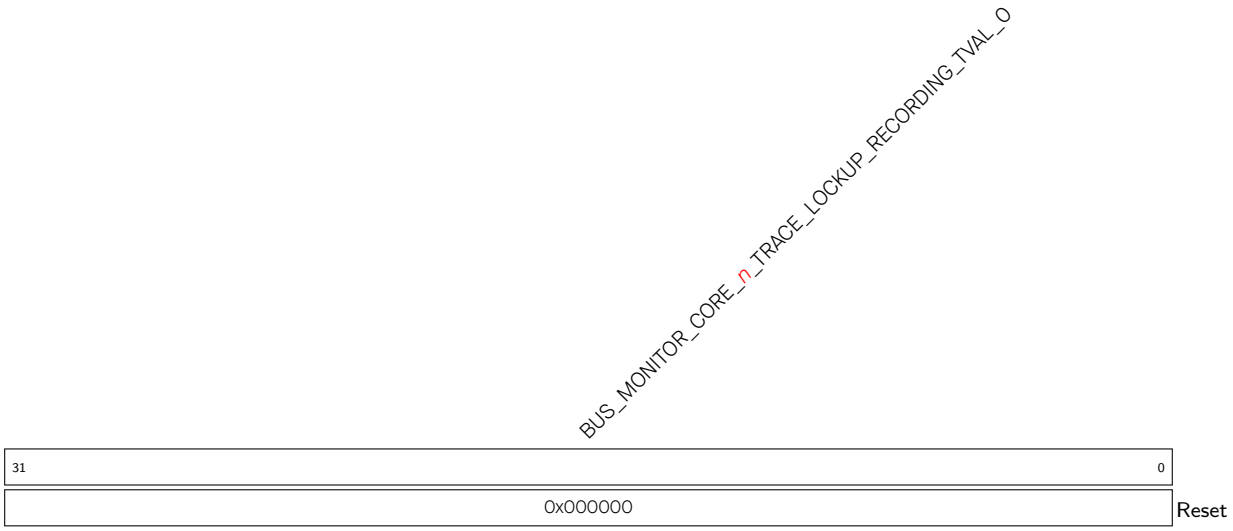
(RO)

Register 7.43. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_CAUSE_0_REG (*n*=0,1) (0x0068+0x88×n)



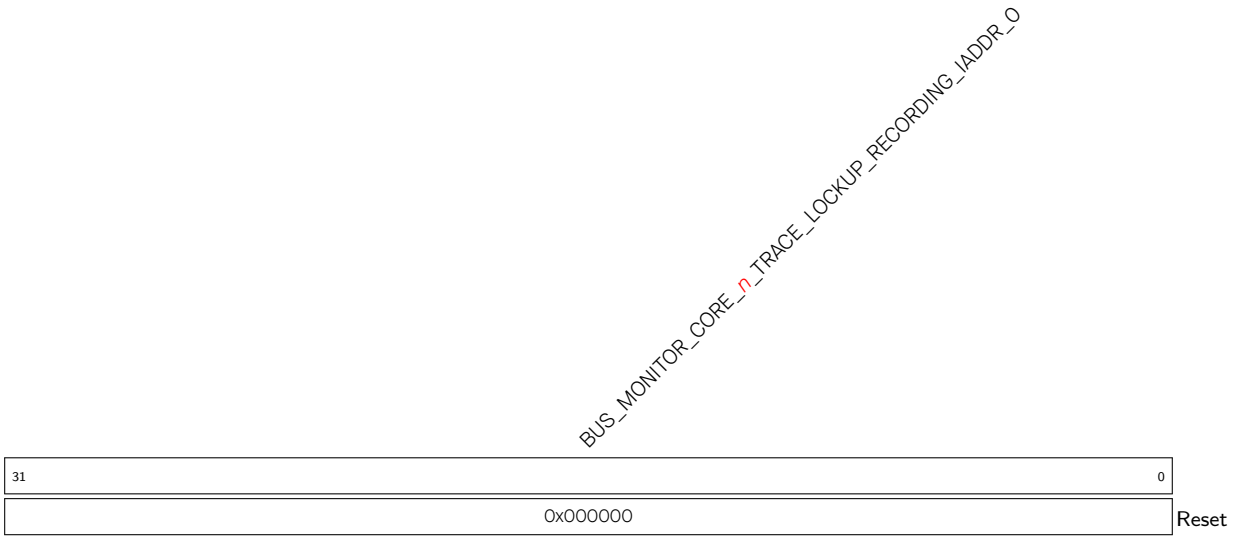
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_CAUSE_0 Represents the second lockup cause. (RO)

Register 7.44. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_TVAL_0_REG (*n*=0,1) (0x006C+0x88×n)



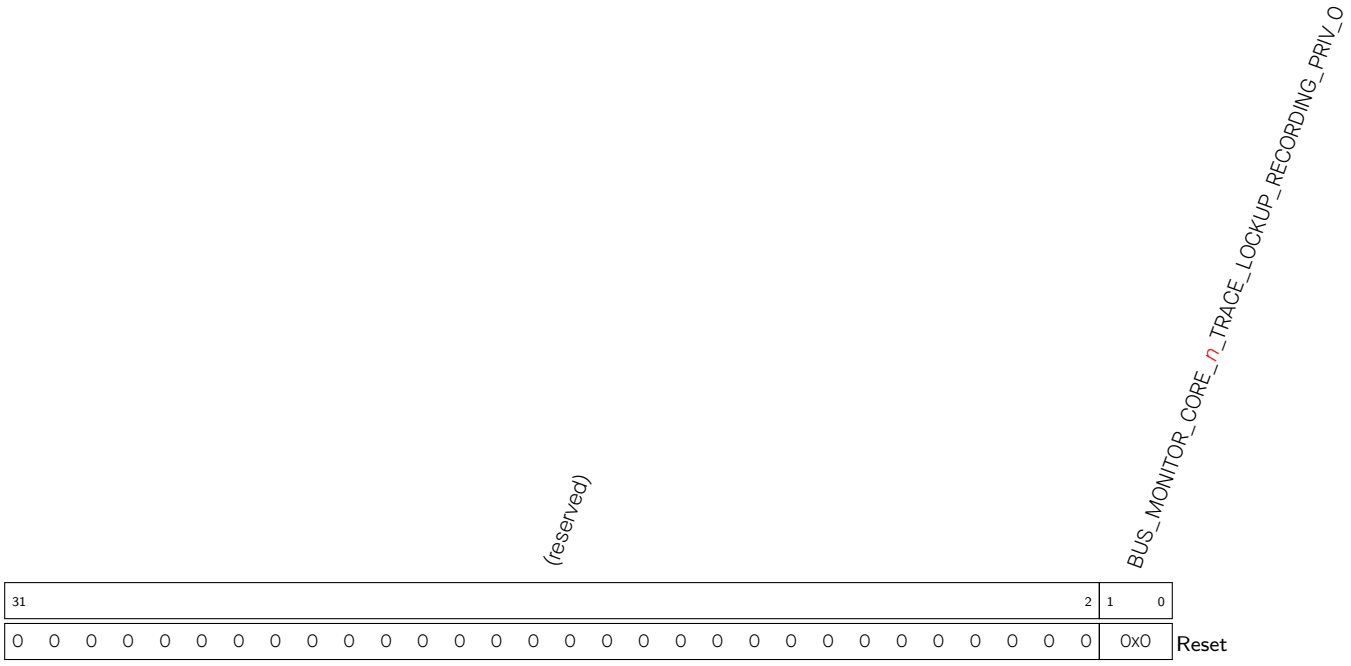
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_TVAL_0 Represents the second lockup trap value. (RO)

Register 7.45. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_IADDR_0_REG (*n*=0,1) (0x0070+0x88×n)



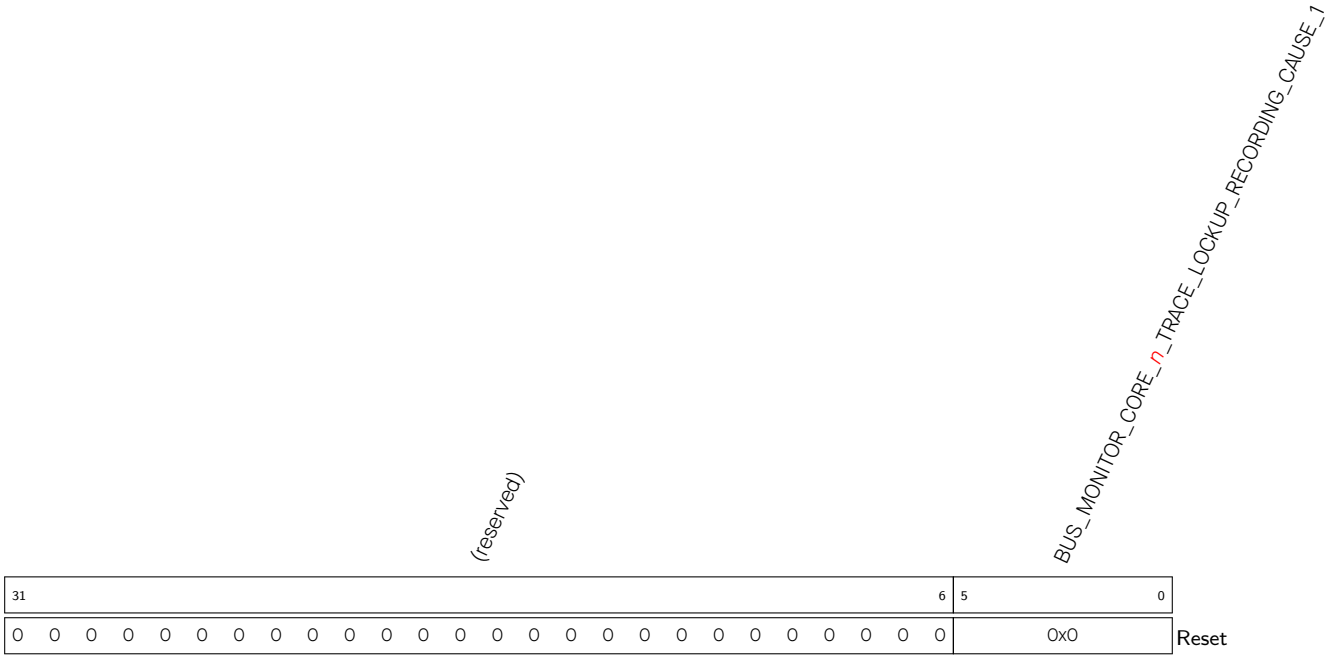
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_IADDR_0 Represents the program counter (PC) at the second lockup instruction. (RO)

Register 7.46. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_PRIV_0_REG (*n*=0,1) (0x0074+0x88×n)



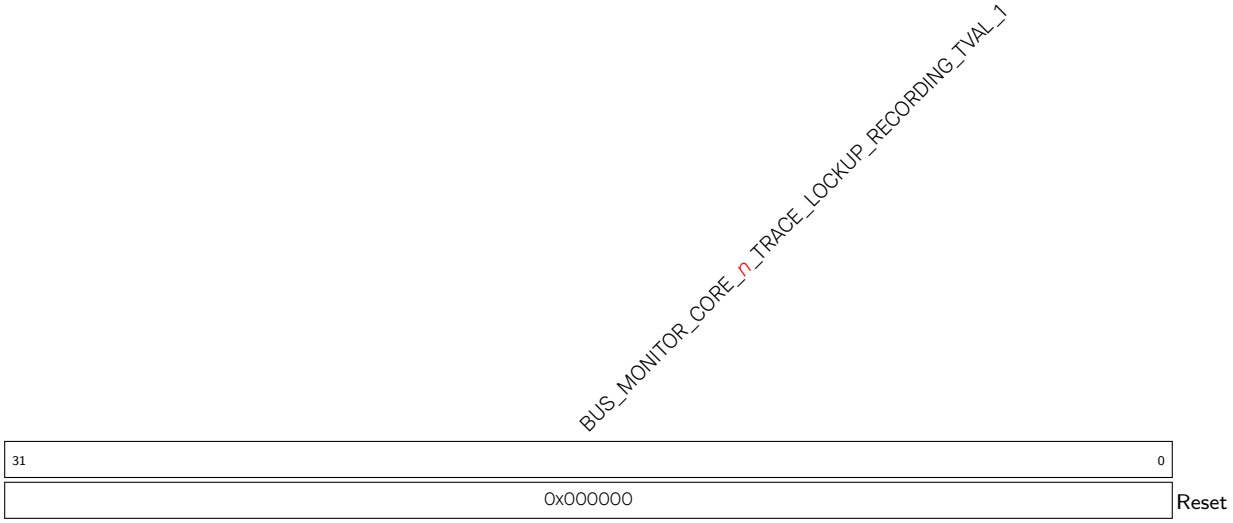
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_PRIV_0 Represents the second lockup privilege level. (RO)

Register 7.47. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_CAUSE_1_REG (*n*=0,1) (0x0078+0x88×n)



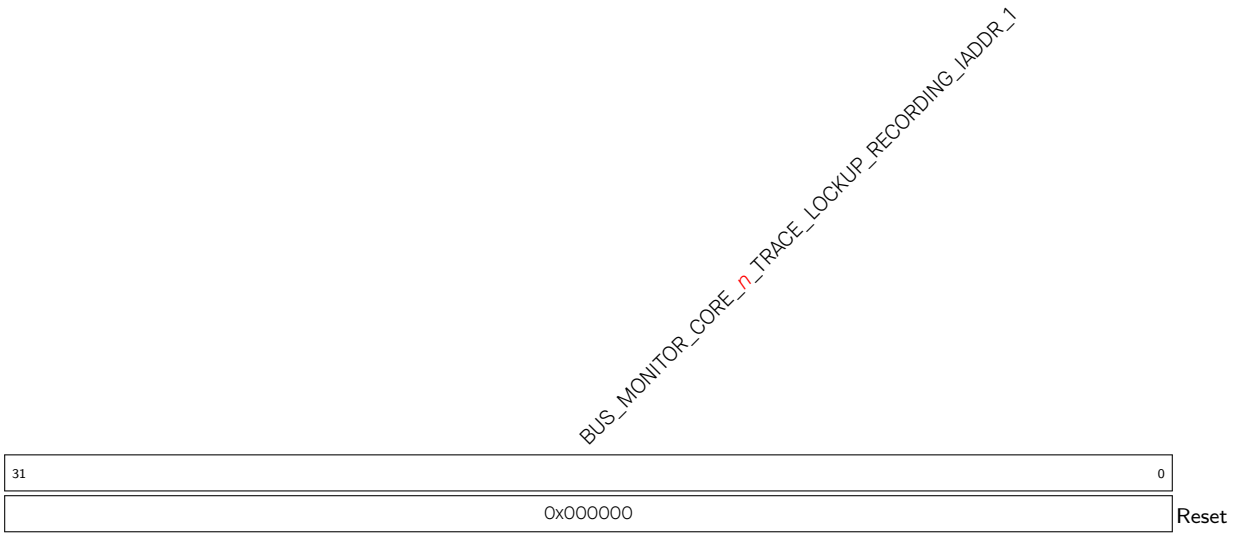
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_CAUSE_1 Represents the first lockup cause. (RO)

Register 7.48. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_TVAL_1_REG (*n*=0,1) (0x007C+0x88×n)



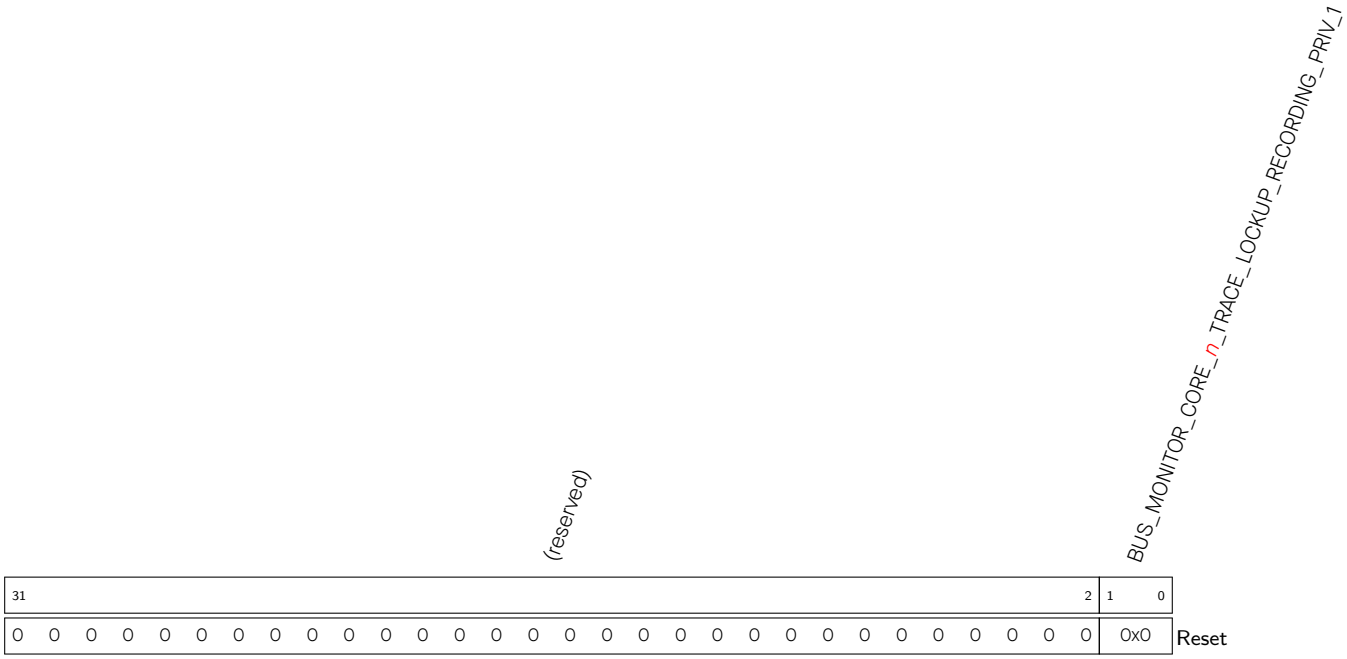
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_TVAL_1 Represents the first lockup trap value. (RO)

Register 7.49. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_IADDR_1_REG (*n*=0,1) (0x0080+0x88×n)



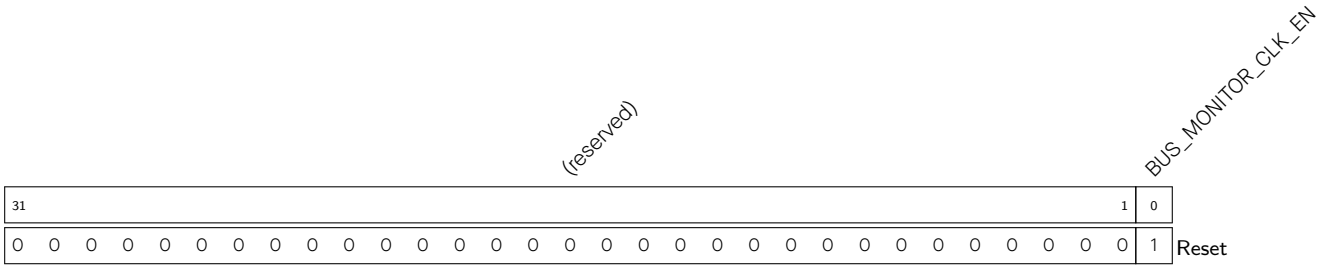
BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_IADDR_1 Represents the program counter (PC) at the first lockup instruction. (RO)

Register 7.50. BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_PRIV_1_REG (*n*=0,1) (0x0084+0x88×n)



BUS_MONITOR_CORE_*n*_TRACE_LOCKUP_RECORDING_PRIV_1 Represents the first lockup privilege level. (RO)

Register 7.51. BUS_MONITOR_CLOCK_GATE_REG (0x0114)



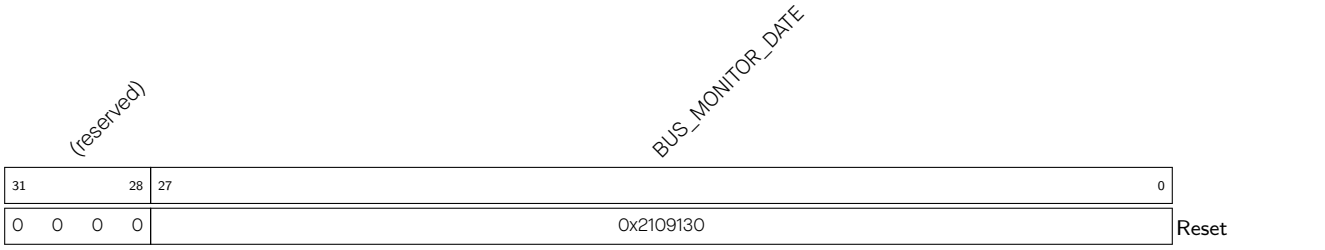
BUS_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

Register 7.52. BUS_MONITOR_DATE_REG (0x03FC)



BUS_MONITOR_DATE Version control register. (R/W)

Chapter 8

LP Mailbox

8.1 Overview

ESP32-S31 integrates an LP Mailbox module that provides an efficient inter-core communication mechanism between the LP CPU and HP CPU0/1. The LP Mailbox module comprises sixteen 32-bit message registers that the LP CPU and HP CPU0/1 can use to store and exchange messages. Inter-core communication between LP CPU and HP CPU0/1 is achieved through an interrupt mechanism implemented within the LP Mailbox module.

8.2 Features

ESP32-S31 LP Mailbox has the following features:

- Sixteen 32-bit message registers for inter-core communication
- LP CPU external interrupt signal: MB_LP_INTR
- HP CPU*x* (*x*: 0∼1) external interrupt signal: MB_HP_INTR

8.3 Functional Description

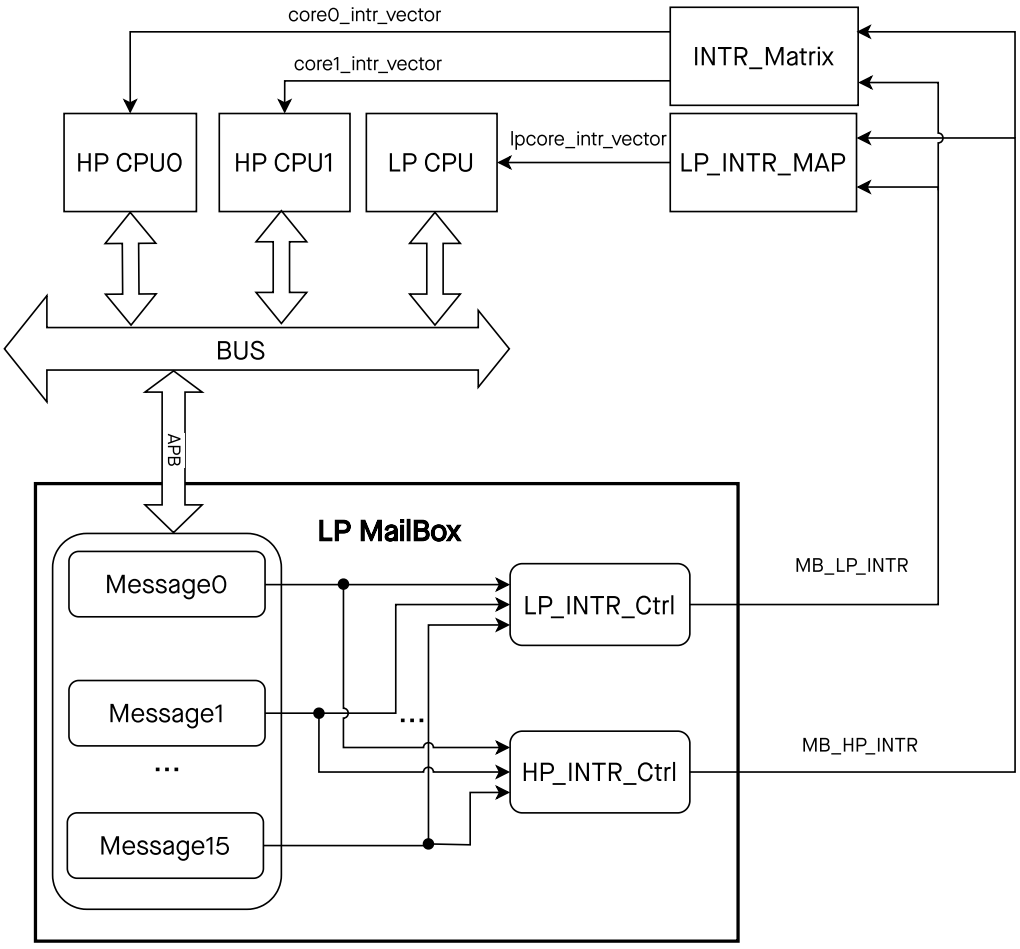


Figure 8.3-1. LP Mailbox Block Diagram

8.3.1 Message Registers

The LP Mailbox provides sixteen 32-bit message registers in total, denoted as MB_MESSAGE_0_REG~MB_MESSAGE_15_REG, for caching inter-core communication messages. Both LP CPU and HP CPU0/1 can read and write these message registers to pass or retrieve messages.

Reading and writing message registers is done through the APB bus, using LP_PERI_CLK, which is the only clock in LP Mailbox. For more information about the clock, please refer to Chapter *7 Reset and Clock [to be added later]*.

8.3.2 Interrupts

ESP32-S31 LP Mailbox can generate the following external interrupt signals:

- MB_LP_INTR
- MB_HP_INTR

MB_LP_INTR is directly mapped to LP CPU (see *Low-Power CPU*), and MB_HP_INTR can be mapped to HP CPU0/1 through the Interrupt Matrix. These interrupt signals are generated by internal interrupt sources of the

LP Mailbox module. Table 8.3-1 lists the internal interrupt sources, interrupt enable conditions, interrupt trigger conditions, interrupt clear conditions, and generated interrupt signals of the LP Mailbox. An internal interrupt source is generated only when the interrupt enable condition is met and the interrupt trigger condition is satisfied. When the interrupt clear condition is met, the internal interrupt source is cleared.

Table 8.3-1. LP Mailbox's Internal Interrupt Sources

Internal Interrupt Source	Interrupt Enable Condition	Interrupt Trigger Condition	Interrupt Clear Condition	Interrupt Signal
MB_HP_*i*_INT	MB_HP_*i*_INT_ENA is 1	Write to MB_MESSAGE_*i*_REG	Write 1 to MB_HP_*i*_INT_CLR	MB_HP_INTR
MB_LP_*i*_INT	MB_LP_*i*_INT_ENA is 1	Write to MB_MESSAGE_*i*_REG	Write 1 to MB_LP_*i*_INT_CLR	MB_LP_INTR

Note:

- *i* ranges from 0~15.
- The LP Mailbox does not identify which CPU is accessing it. As long as the trigger conditions in Table 8.3-1 are met, an interrupt is generated or cleared.
- MB_HP_INTR can also serve as an external interrupt source for LP CPU and be mapped to LP CPU by LP_INTR_MAP as illustrated in Figure 8.3-1. For more information about LP_INTR_MAP, please refer to Chapter 2 *Low-Power CPU*.
- MB_LP_INTR can also serve as an external interrupt source for HP CPU*x* and can be mapped to HP CPU*x* through the Interrupt Matrix.
- When the LP Mailbox is required to assign different interrupt sources for HP CPU0 and HP CPU1, it can map MB_HP_INTR and MB_LP_INTR independently to HP CPU0 and HP CPU1 using the Interrupt Matrix.

8.3.3 Inter-Core Communication

When communication is required between LP CPU and HP CPU*x*, a subset of or all message registers in the LP Mailbox can be utilized to share messages as needed. The synchronization of messages can be achieved through the corresponding interrupts. During communication, software must ensure that only the sender is allowed to write to the message registers to prevent data corruption.

- Example of initiating communication from HP CPU0 to LP CPU using MB_MESSAGE_*i*_REG:
  - Configure the LP interrupt mapping module (LP_INTR_MAP, see Figure 8.3-1) to map the MB_LP_*i*_INT interrupt to LP CPU.
  - Set MB_LP_*i*_INT_ENA to enable the MB_LP_*i*_INT interrupt.
  - HP CPU0 writes to MB_MESSAGE_*i*_REG, caching shared messages.
  - LP CPU detects the interrupt and enters the interrupt handler. In the interrupt handler, it checks the MB_LP_INT_ST_REG interrupt status register, finds the MB_LP_*i*_INT_ST status bit set, and determines that the MB_LP_*i*_INT interrupt has occurred.
  - LP CPU reads MB_MESSAGE_*i*_REG to retrieve shared message.

  - LP CPU writes 1 to the MB_LP_*i*_INT_CLR bit of the MB_LP_INT_CLR_REG register, which triggers the interrupt clear action.
  - LP CPU reads the MB_LP_INT_ST_REG interrupt status register until the MB_LP_*i*_ST status bit is cleared. Then LP CPU exits the interrupt handler.

- Example of initiating communication from LP CPU to HP CPU0:
  - Configure the Interrupt Matrix to map MB_HP_INTR to HP CPU0. See Chapter 9 *Interrupt Matrix [to be added later]*.
  - Set MB_HP_*i*_INT_ENA to enable the MB_HP_*i*_INT interrupt.
  - LP CPU writes to MB_MESSAGE_*i*_REG, caching shared messages.
  - HP CPU0 detects the interrupt and enters the interrupt handler. In the interrupt handler, it checks the MB_HP_INT_ST_REG interrupt status register, finds the MB_HP_*i*_INT_ST status bit set, and determines that the MB_HP_*i*_INT interrupt has occurred.
  - HP CPU0 reads MB_MESSAGE_*i*_REG to retrieve shared messages.
  - HP CPU0 writes 1 to the MB_HP_*i*_INT_CLR bit of the MB_HP_INT_CLR_REG register, which triggers the interrupt clear action.
  - HP CPU0 reads the MB_HP_INT_ST_REG interrupt status register until the MB_HP_*i*_ST status bit is cleared. Then HP CPU0 exits the interrupt handler.

8.4 Register Summary

The addresses in this section are relative to LP Mailbox base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Registers			
MB_MESSAGE_0_REG	Message register 0	0x0000	R/W
MB_MESSAGE_1_REG	Message register 1	0x0004	R/W
MB_MESSAGE_2_REG	Message register 2	0x0008	R/W
MB_MESSAGE_3_REG	Message register 3	0x000C	R/W
MB_MESSAGE_4_REG	Message register 4	0x0010	R/W
MB_MESSAGE_5_REG	Message register 5	0x0014	R/W
MB_MESSAGE_6_REG	Message register 6	0x0018	R/W
MB_MESSAGE_7_REG	Message register 7	0x001C	R/W
MB_MESSAGE_8_REG	Message register 8	0x0020	R/W
MB_MESSAGE_9_REG	Message register 9	0x0024	R/W
MB_MESSAGE_10_REG	Message register 10	0x0028	R/W
MB_MESSAGE_11_REG	Message register 11	0x002C	R/W
MB_MESSAGE_12_REG	Message register 12	0x0030	R/W
MB_MESSAGE_13_REG	Message register 13	0x0034	R/W
MB_MESSAGE_14_REG	Message register 14	0x0038	R/W

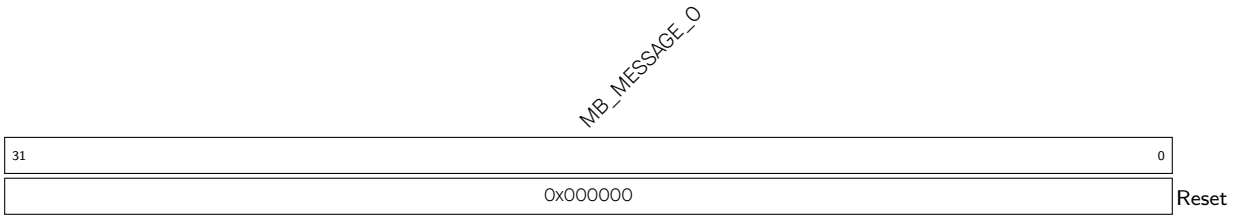
Name	Description	Address	Access
MB_MESSAGE_15_REG	Message register 15	0x003C	R/W
MB_REG_CLK_EN_REG	Clock gate register	0x0060	R/W
Interrupt Registers			
MB_LP_INT_RAW_REG	MB_LP_INTR raw bits	0x0040	RO
MB_LP_INT_ST_REG	MB_LP_INTR status bits	0x0044	RO
MB_LP_INT_ENA_REG	MB_LP_INTR enable bits	0x0048	R/W
MB_LP_INT_CLR_REG	MB_LP_INTR clear bits	0x004C	WO
MB_HP_INT_RAW_REG	MB_HP_INTR raw bits	0x0050	RO
MB_HP_INT_ST_REG	MB_HP_INTR status bits	0x0054	RO
MB_HP_INT_ENA_REG	MB_HP_INTR enable bits	0x0058	R/W
MB_HP_INT_CLR_REG	MB_HP_INTR clear bits	0x005C	WO

8.5 Registers

The addresses in this section are relative to LP Mailbox base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

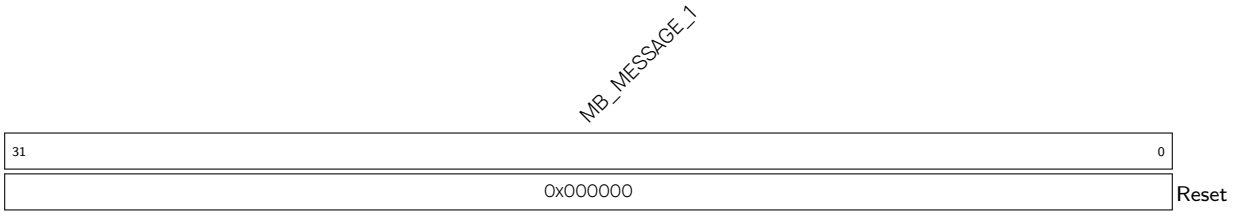
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 8.1. MB_MESSAGE_0_REG (0x0000)



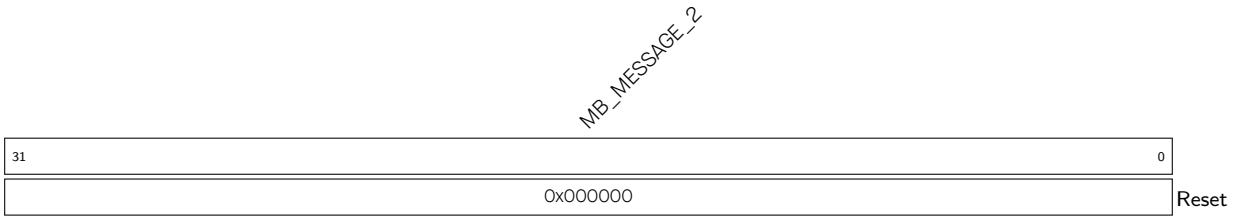
MB_MESSAGE_0 Message register 0. (R/W)

Register 8.2. MB_MESSAGE_1_REG (0x0004)



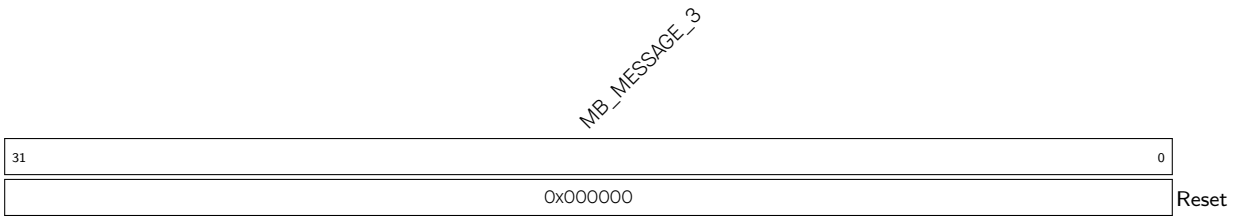
MB_MESSAGE_1 Message register 1. (R/W)

Register 8.3. MB_MESSAGE_2_REG (0x0008)



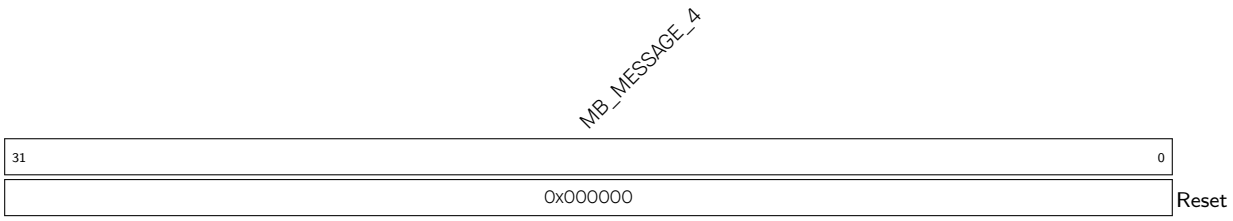
MB_MESSAGE_2 Message register 2. (R/W)

Register 8.4. MB_MESSAGE_3_REG (0x000C)



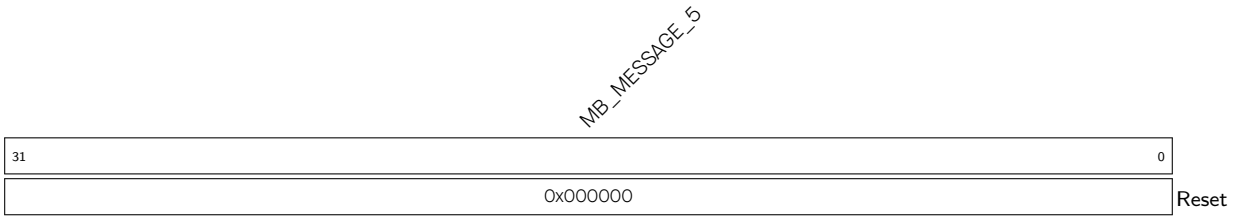
MB_MESSAGE_3 Message register 3. (R/W)

Register 8.5. MB_MESSAGE_4_REG (0x0010)



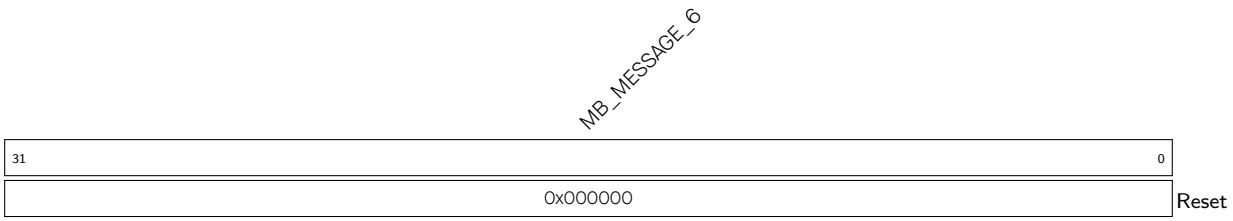
MB_MESSAGE_4 Message register 4. (R/W)

Register 8.6. MB_MESSAGE_5_REG (0x0014)



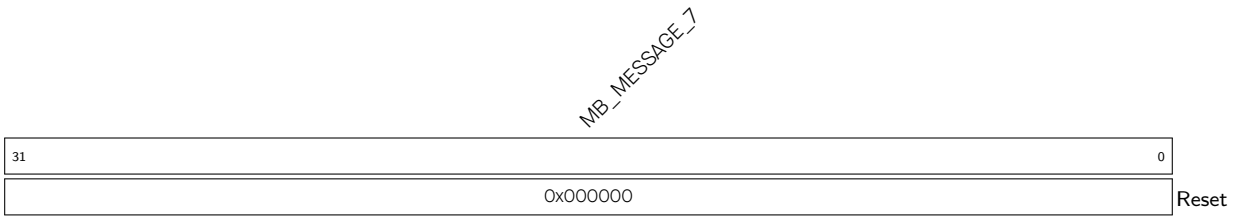
MB_MESSAGE_5 Message register 5. (R/W)

Register 8.7. MB_MESSAGE_6_REG (0x0018)



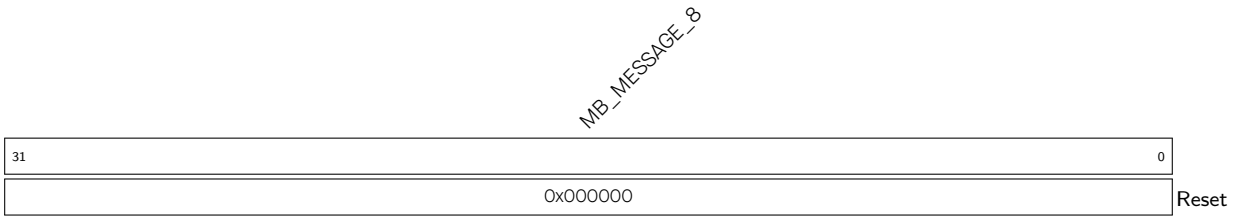
MB_MESSAGE_6 Message register 6. (R/W)

Register 8.8. MB_MESSAGE_7_REG (0x001C)



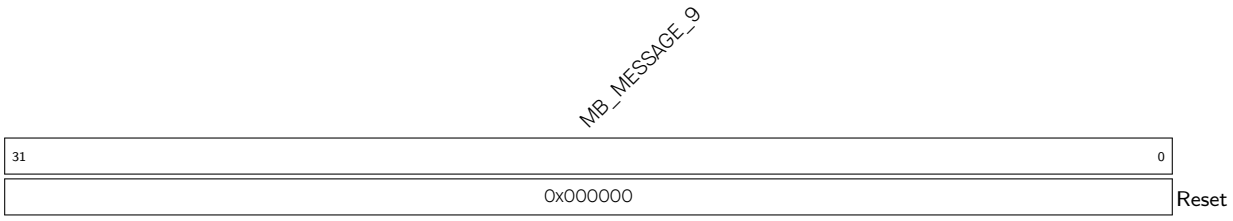
MB_MESSAGE_7 Message register 7. (R/W)

Register 8.9. MB_MESSAGE_8_REG (0x0020)



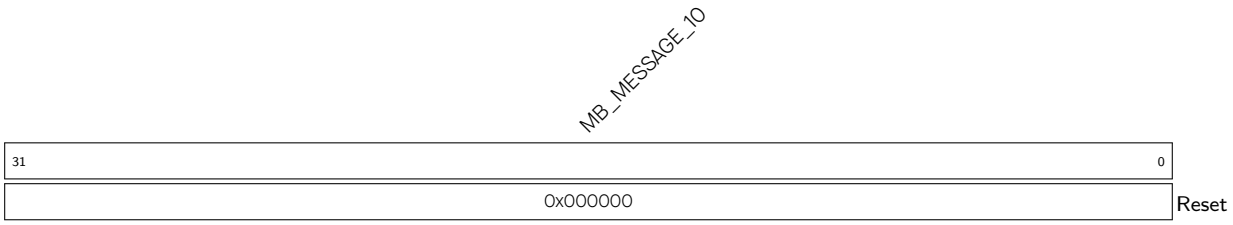
MB_MESSAGE_8 Message register 8. (R/W)

Register 8.10. MB_MESSAGE_9_REG (0x0024)



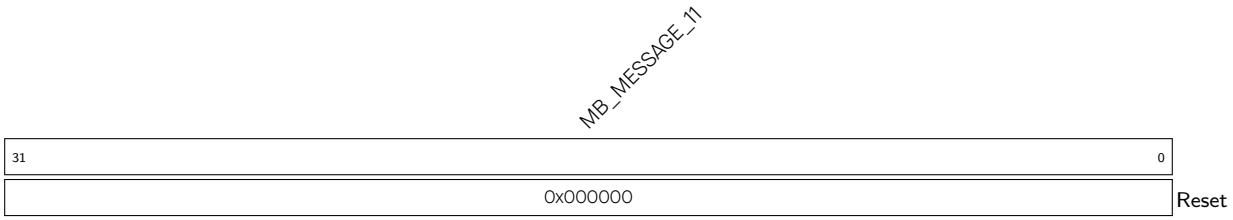
MB_MESSAGE_9 Message register 9. (R/W)

Register 8.11. MB_MESSAGE_10_REG (0x0028)



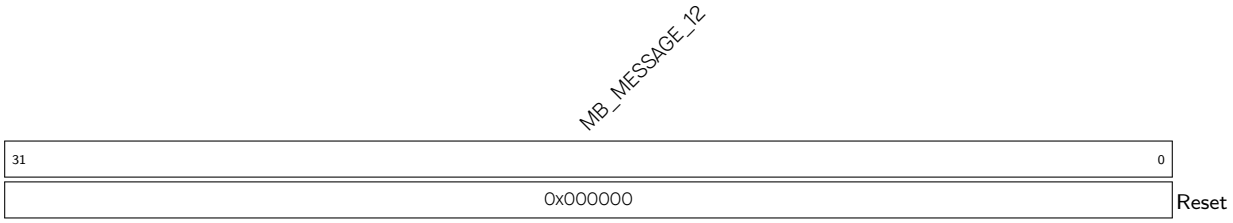
MB_MESSAGE_10 Message register 10. (R/W)

Register 8.12. MB_MESSAGE_11_REG (0x002C)



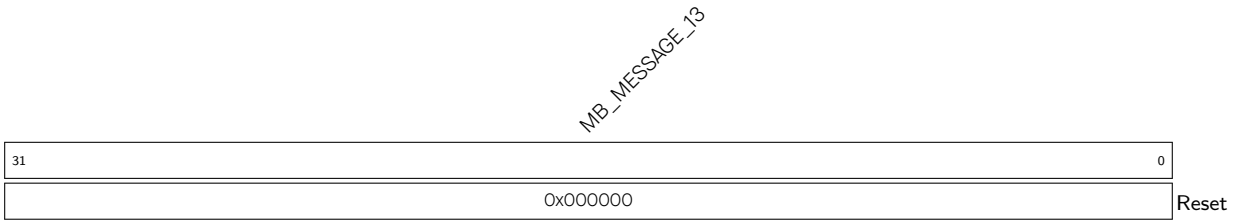
MB_MESSAGE_11 Message register 11. (R/W)

Register 8.13. MB_MESSAGE_12_REG (0x0030)



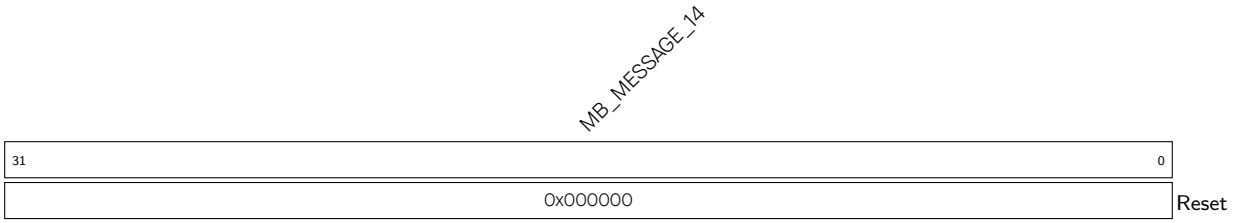
MB_MESSAGE_12 Message register 12. (R/W)

Register 8.14. MB_MESSAGE_13_REG (0x0034)



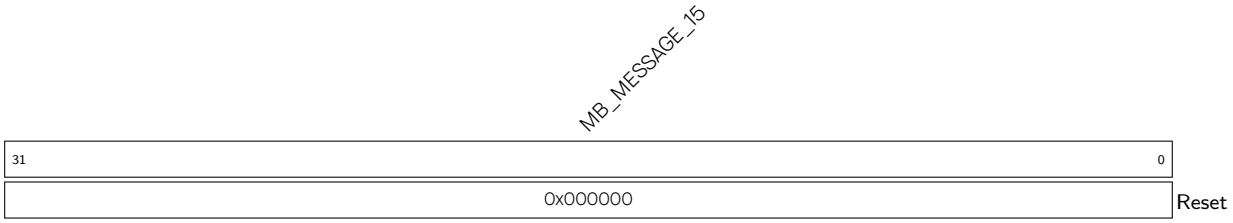
MB_MESSAGE_13 Message register 13. (R/W)

Register 8.15. MB_MESSAGE_14_REG (0x0038)



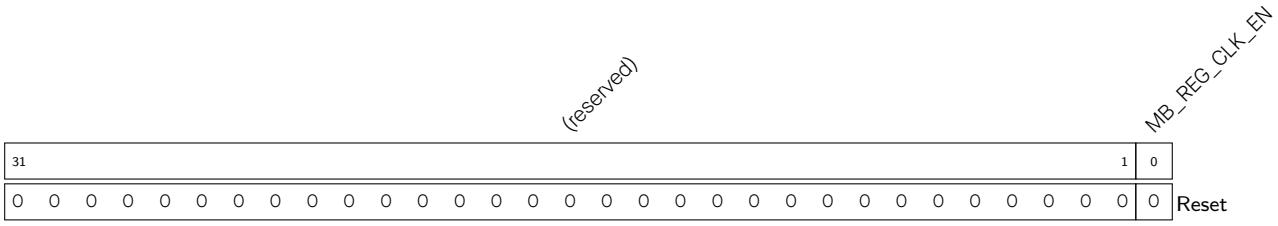
MB_MESSAGE_14 Message register 14. (R/W)

Register 8.16. MB_MESSAGE_15_REG (0x003C)



MB_MESSAGE_15 Message register 15. (R/W)

Register 8.17. MB_REG_CLK_EN_REG (0x0060)



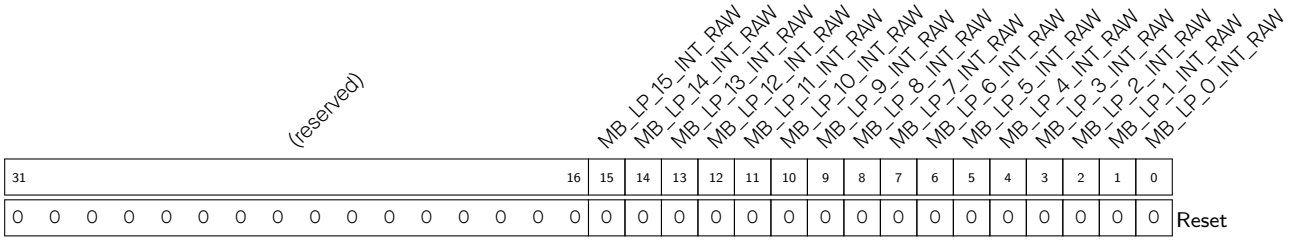
MB_REG_CLK_EN Configures register clock gating.

1: Register clock is always enabled for read and write operations.

0: Only enable clock for register read or write operations.

(R/W)

Register 8.18. MB_LP_INT_RAW_REG (0x0040)



MB_LP_0_INT_RAW The raw interrupt bit of the MB_LP_0_INT interrupt. (RO)

MB_LP_1_INT_RAW The raw interrupt bit of the MB_LP_1_INT interrupt. (RO)

MB_LP_2_INT_RAW The raw interrupt bit of the MB_LP_2_INT interrupt. (RO)

MB_LP_3_INT_RAW The raw interrupt bit of the MB_LP_3_INT interrupt. (RO)

MB_LP_4_INT_RAW The raw interrupt bit of the MB_LP_4_INT interrupt. (RO)

MB_LP_5_INT_RAW The raw interrupt bit of the MB_LP_5_INT interrupt. (RO)

MB_LP_6_INT_RAW The raw interrupt bit of the MB_LP_6_INT interrupt. (RO)

MB_LP_7_INT_RAW The raw interrupt bit of the MB_LP_7_INT interrupt. (RO)

MB_LP_8_INT_RAW The raw interrupt bit of the MB_LP_8_INT interrupt. (RO)

MB_LP_9_INT_RAW The raw interrupt bit of the MB_LP_9_INT interrupt. (RO)

MB_LP_10_INT_RAW The raw interrupt bit of the MB_LP_10_INT interrupt. (RO)

MB_LP_11_INT_RAW The raw interrupt bit of the MB_LP_11_INT interrupt. (RO)

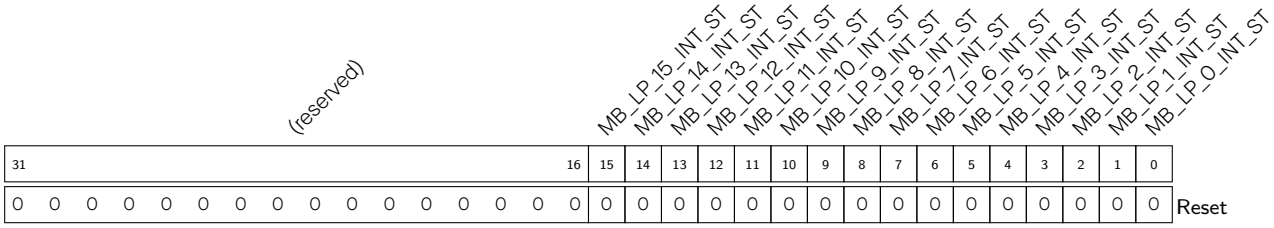
MB_LP_12_INT_RAW The raw interrupt bit of the MB_LP_12_INT interrupt. (RO)

MB_LP_13_INT_RAW The raw interrupt bit of the MB_LP_13_INT interrupt. (RO)

MB_LP_14_INT_RAW The raw interrupt bit of the MB_LP_14_INT interrupt. (RO)

MB_LP_15_INT_RAW The raw interrupt bit of the MB_LP_15_INT interrupt. (RO)

Register 8.19. MB_LP_INT_ST_REG (0x0044)



MB_LP_0_INT_ST The masked interrupt status bit of the MB_LP_0_INT interrupt. (RO)

MB_LP_1_INT_ST The masked interrupt status bit of the MB_LP_1_INT interrupt. (RO)

MB_LP_2_INT_ST The masked interrupt status bit of the MB_LP_2_INT interrupt. (RO)

MB_LP_3_INT_ST The masked interrupt status bit of the MB_LP_3_INT interrupt. (RO)

MB_LP_4_INT_ST The masked interrupt status bit of the MB_LP_4_INT interrupt. (RO)

MB_LP_5_INT_ST The masked interrupt status bit of the MB_LP_5_INT interrupt. (RO)

MB_LP_6_INT_ST The masked interrupt status bit of the MB_LP_6_INT interrupt. (RO)

MB_LP_7_INT_ST The masked interrupt status bit of the MB_LP_7_INT interrupt. (RO)

MB_LP_8_INT_ST The masked interrupt status bit of the MB_LP_8_INT interrupt. (RO)

MB_LP_9_INT_ST The masked interrupt status bit of the MB_LP_9_INT interrupt. (RO)

MB_LP_10_INT_ST The masked interrupt status bit of the MB_LP_10_INT interrupt. (RO)

MB_LP_11_INT_ST The masked interrupt status bit of the MB_LP_11_INT interrupt. (RO)

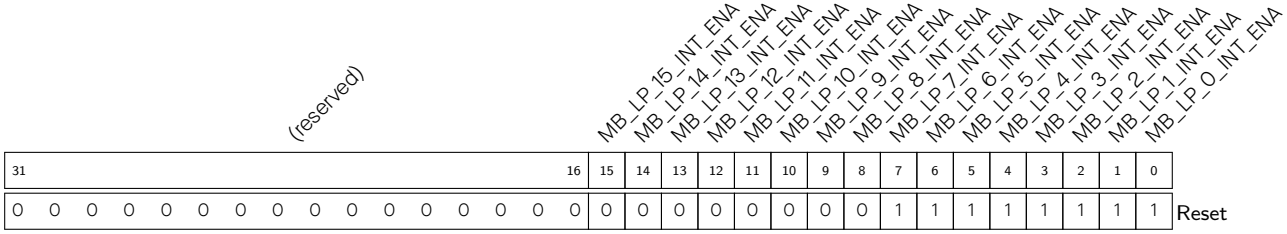
MB_LP_12_INT_ST The masked interrupt status bit of the MB_LP_12_INT interrupt. (RO)

MB_LP_13_INT_ST The masked interrupt status bit of the MB_LP_13_INT interrupt. (RO)

MB_LP_14_INT_ST The masked interrupt status bit of the MB_LP_14_INT interrupt. (RO)

MB_LP_15_INT_ST The masked interrupt status bit of the MB_LP_15_INT interrupt. (RO)

Register 8.20. MB_LP_INT_ENA_REG (0x0048)



MB_LP_0_INT_ENA Write 1 to enable the MB_LP_0_INT interrupt. (R/W)

MB_LP_1_INT_ENA Write 1 to enable the MB_LP_1_INT interrupt. (R/W)

MB_LP_2_INT_ENA Write 1 to enable the MB_LP_2_INT interrupt. (R/W)

MB_LP_3_INT_ENA Write 1 to enable the MB_LP_3_INT interrupt. (R/W)

MB_LP_4_INT_ENA Write 1 to enable the MB_LP_4_INT interrupt. (R/W)

MB_LP_5_INT_ENA Write 1 to enable the MB_LP_5_INT interrupt. (R/W)

MB_LP_6_INT_ENA Write 1 to enable the MB_LP_6_INT interrupt. (R/W)

MB_LP_7_INT_ENA Write 1 to enable the MB_LP_7_INT interrupt. (R/W)

MB_LP_8_INT_ENA Write 1 to enable the MB_LP_8_INT interrupt. (R/W)

MB_LP_9_INT_ENA Write 1 to enable the MB_LP_9_INT interrupt. (R/W)

MB_LP_10_INT_ENA Write 1 to enable the MB_LP_10_INT interrupt. (R/W)

MB_LP_11_INT_ENA Write 1 to enable the MB_LP_11_INT interrupt. (R/W)

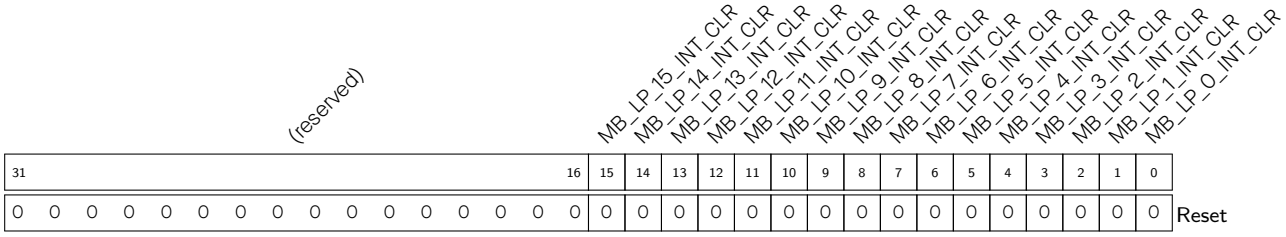
MB_LP_12_INT_ENA Write 1 to enable the MB_LP_12_INT interrupt. (R/W)

MB_LP_13_INT_ENA Write 1 to enable the MB_LP_13_INT interrupt. (R/W)

MB_LP_14_INT_ENA Write 1 to enable the MB_LP_14_INT interrupt. (R/W)

MB_LP_15_INT_ENA Write 1 to enable the MB_LP_15_INT interrupt. (R/W)

Register 8.21. MB_LP_INT_CLR_REG (0x004C)



MB_LP_0_INT_CLR Write 1 to clear the MB_LP_0_INT interrupt. (WO)

MB_LP_1_INT_CLR Write 1 to clear the MB_LP_1_INT interrupt. (WO)

MB_LP_2_INT_CLR Write 1 to clear the MB_LP_2_INT interrupt. (WO)

MB_LP_3_INT_CLR Write 1 to clear the MB_LP_3_INT interrupt. (WO)

MB_LP_4_INT_CLR Write 1 to clear the MB_LP_4_INT interrupt. (WO)

MB_LP_5_INT_CLR Write 1 to clear the MB_LP_5_INT interrupt. (WO)

MB_LP_6_INT_CLR Write 1 to clear the MB_LP_6_INT interrupt. (WO)

MB_LP_7_INT_CLR Write 1 to clear the MB_LP_7_INT interrupt. (WO)

MB_LP_8_INT_CLR Write 1 to clear the MB_LP_8_INT interrupt. (WO)

MB_LP_9_INT_CLR Write 1 to clear the MB_LP_9_INT interrupt. (WO)

MB_LP_10_INT_CLR Write 1 to clear the MB_LP_10_INT interrupt. (WO)

MB_LP_11_INT_CLR Write 1 to clear the MB_LP_11_INT interrupt. (WO)

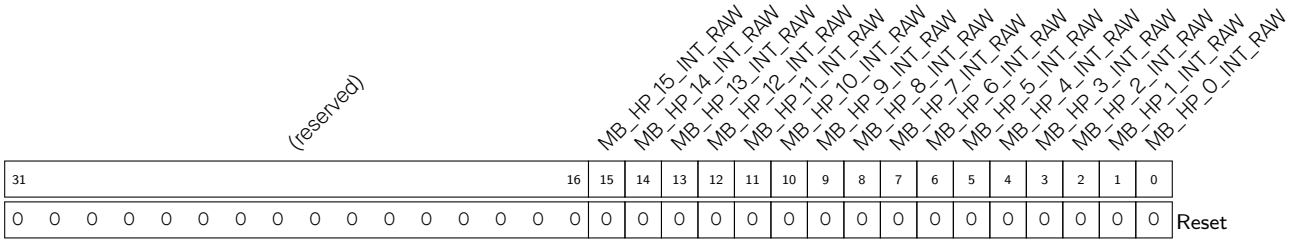
MB_LP_12_INT_CLR Write 1 to clear the MB_LP_12_INT interrupt. (WO)

MB_LP_13_INT_CLR Write 1 to clear the MB_LP_13_INT interrupt. (WO)

MB_LP_14_INT_CLR Write 1 to clear the MB_LP_14_INT interrupt. (WO)

MB_LP_15_INT_CLR Write 1 to clear the MB_LP_15_INT interrupt. (WO)

Register 8.22. MB_HP_INT_RAW_REG (0x0050)



MB_HP_0_INT_RAW The raw interrupt bit of the MB_HP_0_INT interrupt. (RO)

MB_HP_1_INT_RAW The raw interrupt bit of the MB_HP_1_INT interrupt. (RO)

MB_HP_2_INT_RAW The raw interrupt bit of the MB_HP_2_INT interrupt. (RO)

MB_HP_3_INT_RAW The raw interrupt bit of the MB_HP_3_INT interrupt. (RO)

MB_HP_4_INT_RAW The raw interrupt bit of the MB_HP_4_INT interrupt. (RO)

MB_HP_5_INT_RAW The raw interrupt bit of the MB_HP_5_INT interrupt. (RO)

MB_HP_6_INT_RAW The raw interrupt bit of the MB_HP_6_INT interrupt. (RO)

MB_HP_7_INT_RAW The raw interrupt bit of the MB_HP_7_INT interrupt. (RO)

MB_HP_8_INT_RAW The raw interrupt bit of the MB_HP_8_INT interrupt. (RO)

MB_HP_9_INT_RAW The raw interrupt bit of the MB_HP_9_INT interrupt. (RO)

MB_HP_10_INT_RAW The raw interrupt bit of the MB_HP_10_INT interrupt. (RO)

MB_HP_11_INT_RAW The raw interrupt bit of the MB_HP_11_INT interrupt. (RO)

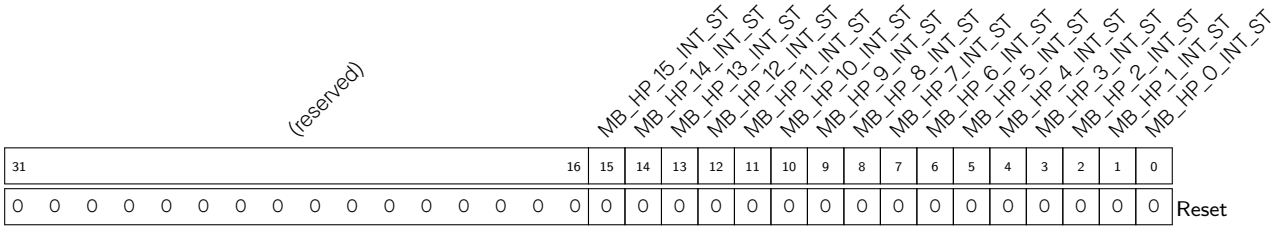
MB_HP_12_INT_RAW The raw interrupt bit of the MB_HP_12_INT interrupt. (RO)

MB_HP_13_INT_RAW The raw interrupt bit of the MB_HP_13_INT interrupt. (RO)

MB_HP_14_INT_RAW The raw interrupt bit of the MB_HP_14_INT interrupt. (RO)

MB_HP_15_INT_RAW The raw interrupt bit of the MB_HP_15_INT interrupt. (RO)

Register 8.23. MB_HP_INT_ST_REG (0x0054)



MB_HP_0_INT_ST The masked interrupt status bit of the MB_HP_0_INT interrupt. (RO)

MB_HP_1_INT_ST The masked interrupt status bit of the MB_HP_1_INT interrupt. (RO)

MB_HP_2_INT_ST The masked interrupt status bit of the MB_HP_2_INT interrupt. (RO)

MB_HP_3_INT_ST The masked interrupt status bit of the MB_HP_3_INT interrupt. (RO)

MB_HP_4_INT_ST The masked interrupt status bit of the MB_HP_4_INT interrupt. (RO)

MB_HP_5_INT_ST The masked interrupt status bit of the MB_HP_5_INT interrupt. (RO)

MB_HP_6_INT_ST The masked interrupt status bit of the MB_HP_6_INT interrupt. (RO)

MB_HP_7_INT_ST The masked interrupt status bit of the MB_HP_7_INT interrupt. (RO)

MB_HP_8_INT_ST The masked interrupt status bit of the MB_HP_8_INT interrupt. (RO)

MB_HP_9_INT_ST The masked interrupt status bit of the MB_HP_9_INT interrupt. (RO)

MB_HP_10_INT_ST The masked interrupt status bit of the MB_HP_10_INT interrupt. (RO)

MB_HP_11_INT_ST The masked interrupt status bit of the MB_HP_11_INT interrupt. (RO)

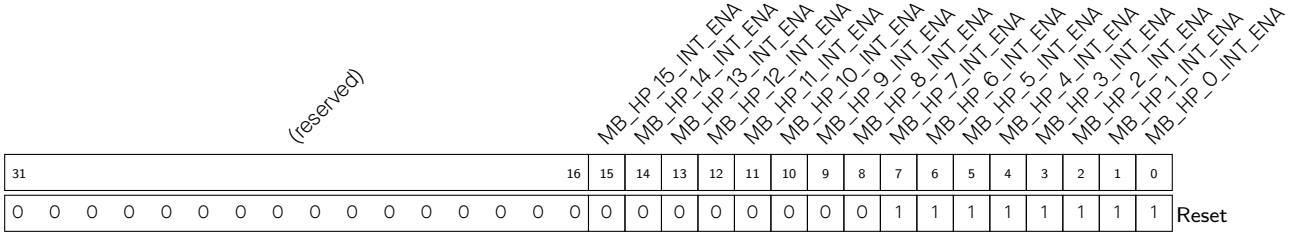
MB_HP_12_INT_ST The masked interrupt status bit of the MB_HP_12_INT interrupt. (RO)

MB_HP_13_INT_ST The masked interrupt status bit of the MB_HP_13_INT interrupt. (RO)

MB_HP_14_INT_ST The masked interrupt status bit of the MB_HP_14_INT interrupt. (RO)

MB_HP_15_INT_ST The masked interrupt status bit of the MB_HP_15_INT interrupt. (RO)

Register 8.24. MB_HP_INT_ENA_REG (0x0058)



MB_HP_0_INT_ENA Write 1 to enable the MB_HP_0_INT interrupt. (R/W)

MB_HP_1_INT_ENA Write 1 to enable the MB_HP_1_INT interrupt. (R/W)

MB_HP_2_INT_ENA Write 1 to enable the MB_HP_2_INT interrupt. (R/W)

MB_HP_3_INT_ENA Write 1 to enable the MB_HP_3_INT interrupt. (R/W)

MB_HP_4_INT_ENA Write 1 to enable the MB_HP_4_INT interrupt. (R/W)

MB_HP_5_INT_ENA Write 1 to enable the MB_HP_5_INT interrupt. (R/W)

MB_HP_6_INT_ENA Write 1 to enable the MB_HP_6_INT interrupt. (R/W)

MB_HP_7_INT_ENA Write 1 to enable the MB_HP_7_INT interrupt. (R/W)

MB_HP_8_INT_ENA Write 1 to enable the MB_HP_8_INT interrupt. (R/W)

MB_HP_9_INT_ENA Write 1 to enable the MB_HP_9_INT interrupt. (R/W)

MB_HP_10_INT_ENA Write 1 to enable the MB_HP_10_INT interrupt. (R/W)

MB_HP_11_INT_ENA Write 1 to enable the MB_HP_11_INT interrupt. (R/W)

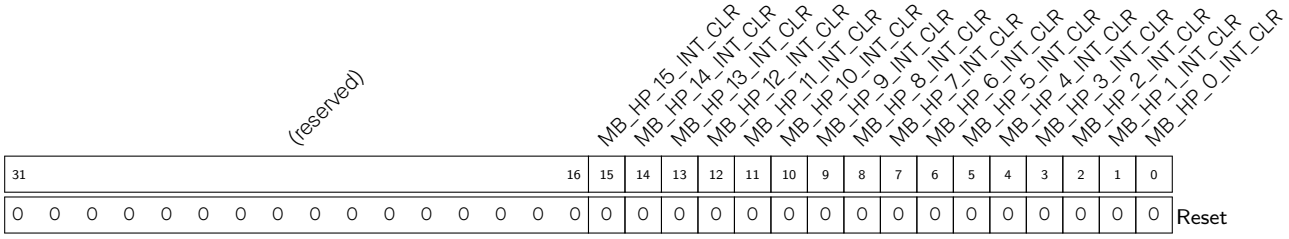
MB_HP_12_INT_ENA Write 1 to enable the MB_HP_12_INT interrupt. (R/W)

MB_HP_13_INT_ENA Write 1 to enable the MB_HP_13_INT interrupt. (R/W)

MB_HP_14_INT_ENA Write 1 to enable the MB_HP_14_INT interrupt. (R/W)

MB_HP_15_INT_ENA Write 1 to enable the MB_HP_15_INT interrupt. (R/W)

Register 8.25. MB_HP_INT_CLR_REG (0x005C)



MB_HP_0_INT_CLR Write 1 to clear the MB_HP_0_INT interrupt. (WO)

MB_HP_1_INT_CLR Write 1 to clear the MB_HP_1_INT interrupt. (WO)

MB_HP_2_INT_CLR Write 1 to clear the MB_HP_2_INT interrupt. (WO)

MB_HP_3_INT_CLR Write 1 to clear the MB_HP_3_INT interrupt. (WO)

MB_HP_4_INT_CLR Write 1 to clear the MB_HP_4_INT interrupt. (WO)

MB_HP_5_INT_CLR Write 1 to clear the MB_HP_5_INT interrupt. (WO)

MB_HP_6_INT_CLR Write 1 to clear the MB_HP_6_INT interrupt. (WO)

MB_HP_7_INT_CLR Write 1 to clear the MB_HP_7_INT interrupt. (WO)

MB_HP_8_INT_CLR Write 1 to clear the MB_HP_8_INT interrupt. (WO)

MB_HP_9_INT_CLR Write 1 to clear the MB_HP_9_INT interrupt. (WO)

MB_HP_10_INT_CLR Write 1 to clear the MB_HP_10_INT interrupt. (WO)

MB_HP_11_INT_CLR Write 1 to clear the MB_HP_11_INT interrupt. (WO)

MB_HP_12_INT_CLR Write 1 to clear the MB_HP_12_INT interrupt. (WO)

MB_HP_13_INT_CLR Write 1 to clear the MB_HP_13_INT interrupt. (WO)

MB_HP_14_INT_CLR Write 1 to clear the MB_HP_14_INT interrupt. (WO)

MB_HP_15_INT_CLR Write 1 to clear the MB_HP_15_INT interrupt. (WO)

Chapter 9

Power Supply Detector (PSDET)

9.1 Overview

ESP32-S31 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 protects the SoC from operating under low-voltage conditions by triggering a brown-out reset, while the voltage glitch detectors protect the SoC against voltage glitch attacks. The power supply detector operates in the always-on power domain, allowing it to monitor voltage at any time.

9.2 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 detectors can detect voltage glitches lasting 50 ns or longer and trigger a system reset.

9.3 Functional Description

9.3.1 Architecture

Figure 9.3-1 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. Among them, VDDPST_* refers to VDDPST2/3/4.

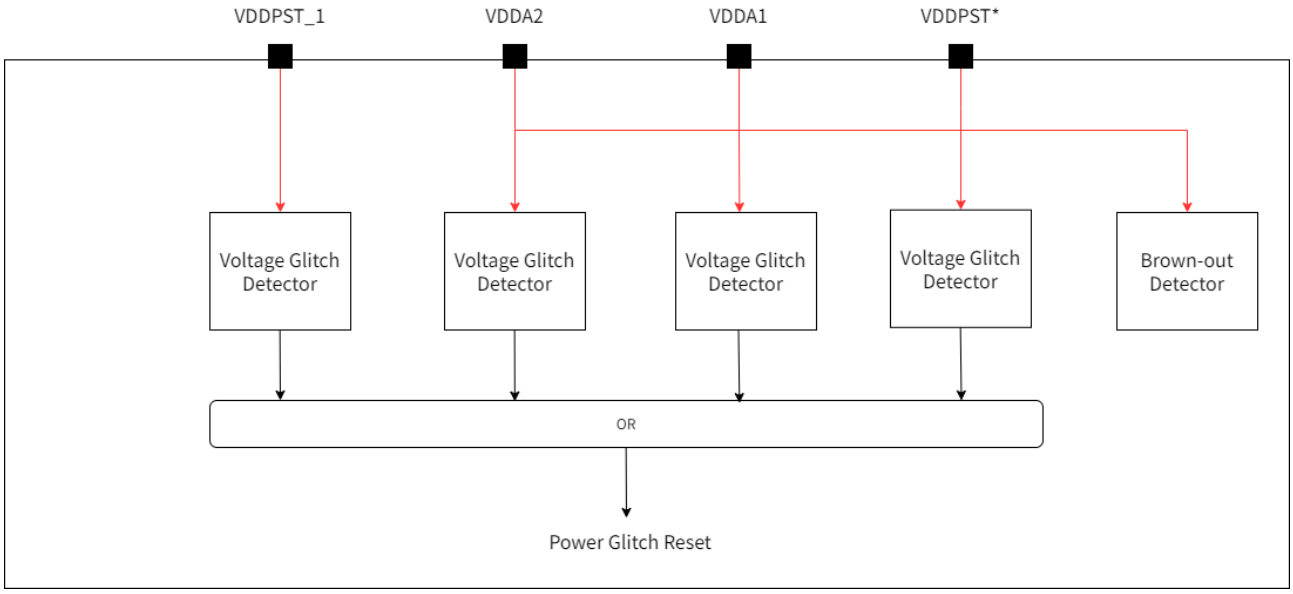


Figure 9.3-1. Architecture of Power Supply Detector

9.3.2 Brown-out Detector

The brown-out detector periodically checks pins VDDPST_1, VDDA2, and VDDPST_* in turn, 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 triggers reset according to 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.

See the figure below for details.

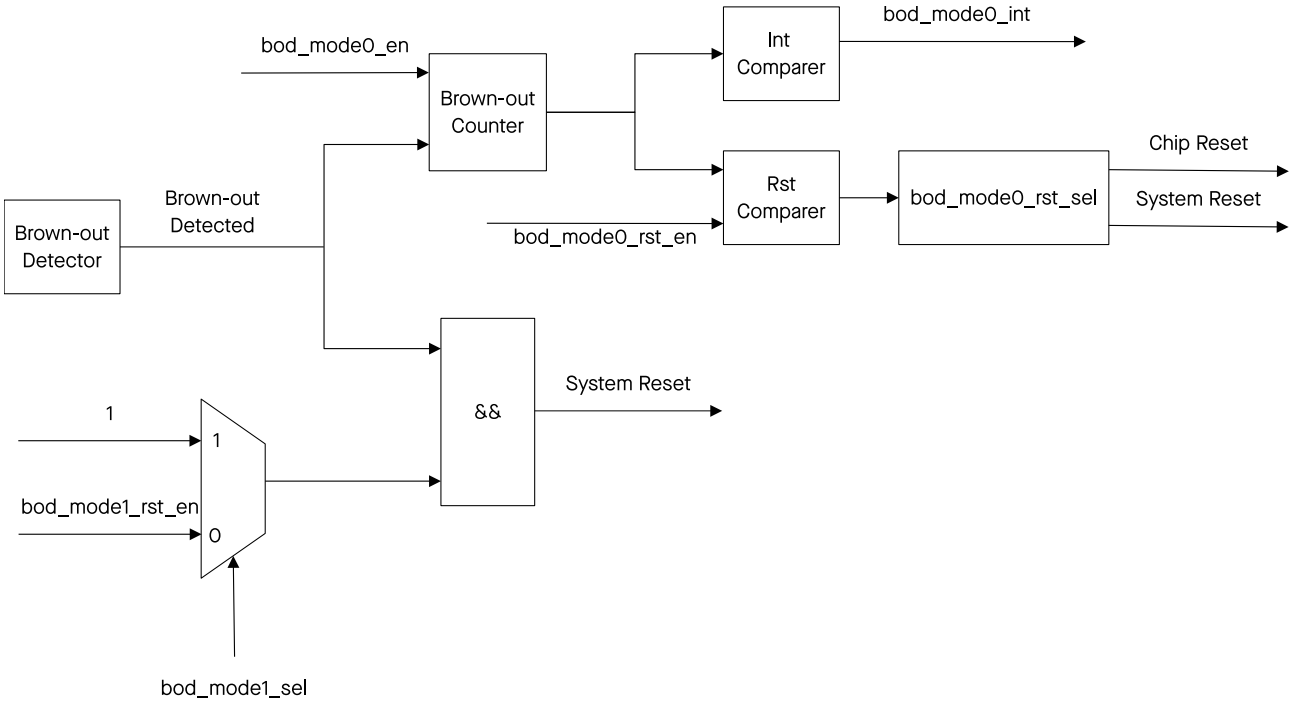


Figure 9.3-2. Brown-out Detector Handling

Registers for controlling the selection signals in the figure are described below.

- `bod_mode0_en`: LP_ANA_BOD_MODE0_INTR_ENA
- `bod_mode0_rst_en`: LP_ANA_BOD_MODE0_RESET_ENA
- `bod_mode0_rst_sel`: LP_ANA_BOD_MODE0_RESET_SEL configures the reset type:
  - 0: chip reset
  - 1: system reset

  For more information regarding chip reset and system reset, please refer to Chapter 7 *Reset and Clock [to be added later]*.
- `bod_mode1_sel`: The bit1 of LP_ANA_ANA_FIB_ENA
- `bod_mode1_rst_en`: LP_ANA_BOD_MODE1_RESET_ENA

9.3.3 Voltage Glitch Detectors

The structure of the voltage glitch detectors is shown in the figure below:

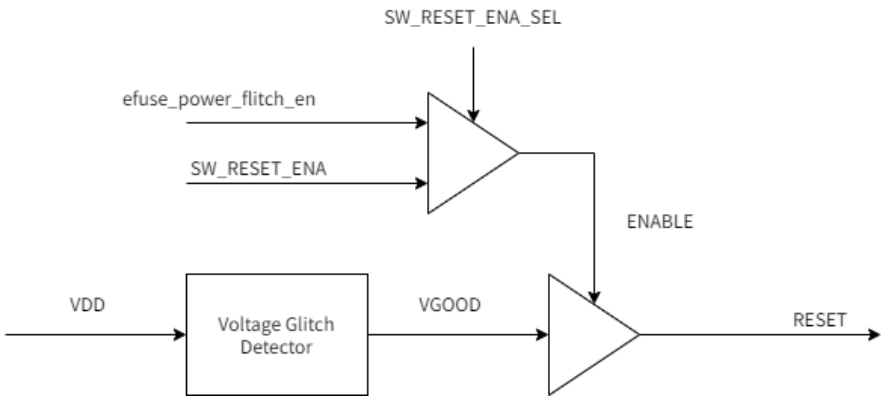


Figure 9.3-3. Voltage Glitch Detectors

The ESP32-S31 has four voltage glitch detectors, each of which can detect glitches lasting 50 ns or longer on the corresponding power supply. The corresponding power pins are VDDPST2/3/4, VDDA1, VDDA2, and VDDPST_1. VDDPST2 and VDDPST3 are connected and thus share one detector. All resets generated by the detectors are combined into one voltage glitch reset.

The voltage glitch detectors' enable control is managed by SW_RESET_ENA_SEL. When set to 1, it is forcibly enabled by hardware, and when set to 0, it is controlled by the software register SW_RESET_ENA. The configuration for detecting different power pins is shown in the table below:

Table 9.3-1. Configuration for Detecting Power Pins

VDD Pins	SW_RESET_ENA	SW_RESET_ENA_SEL
VDDPST2/3/4	LP_ANA_POWER_GLITCH_RESET_ENA[0]	LP_ANA_ANA_FIB_ENA[2]
VDDA1	LP_ANA_POWER_GLITCH_RESET_ENA[1]	LP_ANA_ANA_FIB_ENA[3]
VDDA2	LP_ANA_POWER_GLITCH_RESET_ENA[2]	LP_ANA_ANA_FIB_ENA[4]
VDDPST_1	LP_ANA_POWER_GLITCH_RESET_ENA[3]	LP_ANA_ANA_FIB_ENA[5]

9.4 Interrupts

ESP32-S31's power supply detector can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- LP_ANA_INTR (only for HP CPU)
- LP_ANA_LP_INTR (only for LP CPU)

The interrupt signals are generated by the power supply detector's internal interrupt sources which are listed in Table 9.4-1 with their trigger conditions.

Table 9.4-1. Power Supply Detector's Internal Interrupt Source

Internal Interrupt Source	Trigger Condition	Interrupt Signal
LP_ANA_BOD_MODE0_INT	Triggered when the brown-out detector detects that the voltage is below the threshold.	LP_ANA_INTR
LP_ANA_BOD_MODE0_LP_INT	Triggered when the brown-out detector detects that the voltage is below the threshold.	LP_ANA_LP_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 9.5 *Register Summary*.

9.5 Register Summary

The addresses in this section are relative to the Power Supply Detector base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

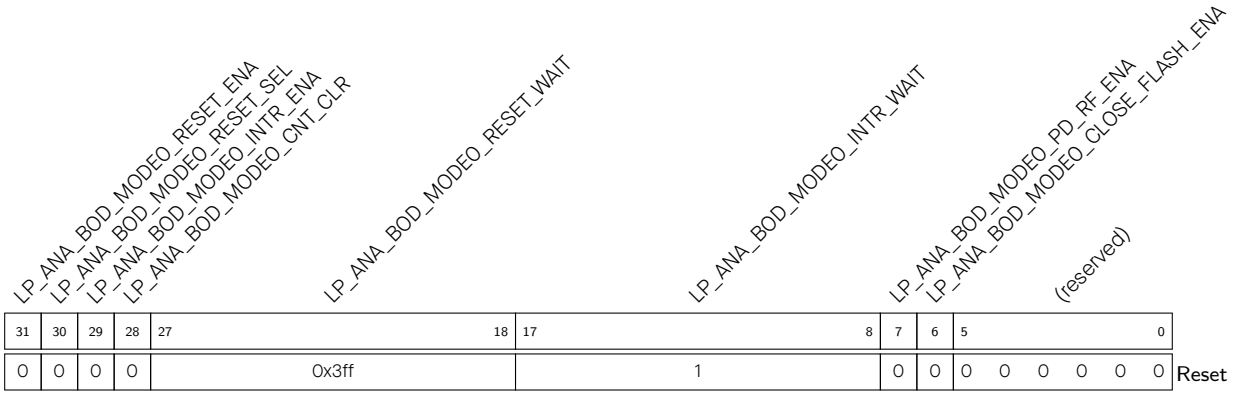
Name	Description	Address	Access
Configuration Registers			
LP_ANA_BOD_MODE0_CNTL_REG	Brown-out detector mode 0 configuration register	0x0000	R/W
LP_ANA_BOD_MODE1_CNTL_REG	Brown-out detector mode 1 configuration register	0x0004	R/W
LP_ANA_PG_GLITCH_CNTL_REG	Configures the reset enable of the voltage glitch detectors	0x0018	R/W
LP_ANA_FIB_ENABLE_REG	Configures the enable control of the voltage glitch detectors	0x001C	R/W
LP_ANA_INT_RAW_REG	LP_ANA_BOD_MODE0_INT raw interrupt	0x0020	R/WTC/SS
LP_ANA_INT_ST_REG	LP_ANA_BOD_MODE0_INT state interrupt	0x0024	RO
LP_ANA_INT_ENA_REG	LP_ANA_BOD_MODE0_INT enable register	0x0028	R/W
LP_ANA_INT_CLR_REG	LP_ANA_BOD_MODE0_INT clear register	0x002C	WT
LP_ANA_LP_INT_RAW_REG	LP_ANA_BOD_MODE0_LP_INT raw interrupt	0x0030	R/WTC/SS
LP_ANA_LP_INT_ST_REG	LP_ANA_BOD_MODE0_LP_INT state interrupt	0x0034	RO
LP_ANA_LP_INT_ENA_REG	LP_ANA_BOD_MODE0_LP_INT enable register	0x0038	R/W
LP_ANA_LP_INT_CLR_REG	LP_ANA_BOD_MODE0_LP_INT clear register	0x003C	WT
Version Control Registers			
LP_ANA_DATE_REG	Version control register	0x03FC	R/W

9.6 Registers

The addresses in this section are relative to the Power Supply Detector base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 9.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)



LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA Configures whether to enable the brown-out detector to trigger flash suspend.
0: Disable
1: Enable
(R/W)

LP_ANA_BOD_MODE0_PD_RF_ENA Configures whether to enable the brown-out detector to power down the RF module.
0: Disable
1: Enable
(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT Configures the time to generate an interrupt after the brown-out signal is valid. The unit is LP_FAST_CLK cycles. (R/W)

LP_ANA_BOD_MODE0_RESET_WAIT Configures the time to generate a reset after the brown-out signal is valid. The unit is LP_FAST_CLK cycles. (R/W)

LP_ANA_BOD_MODE0_CNT_CLR Configures whether to clear the count value of the brown-out detector.
0: Do not clear
1: Clear
(R/W)

LP_ANA_BOD_MODE0_INTR_ENA Enables the interrupts for the brown-out detector mode 0. LP_ANA_BOD_MODE0_INT_RAW and LP_ANA_BOD_MODE0_LP_INT_RAW are valid only when this field is set to 1.
0: Disable
1: Enable
(R/W)

LP_ANA_BOD_MODE0_RESET_SEL Configures the reset type when the brown-out detector is triggered.
0: Chip reset
1: System reset
(R/W)

Continued on the next page...

Register 9.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)

Continued from the previous page...

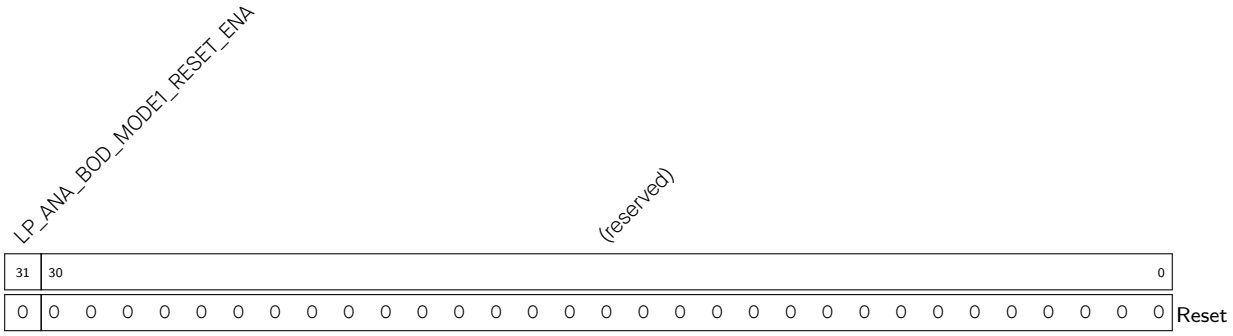
LP_ANA_BOD_MODE0_RESET_ENA Configures whether to enable reset for the brown-out detector.

0: Disable

1: Enable

(R/W)

Register 9.2. LP_ANA_BOD_MODE1_CNTL_REG (0x0004)



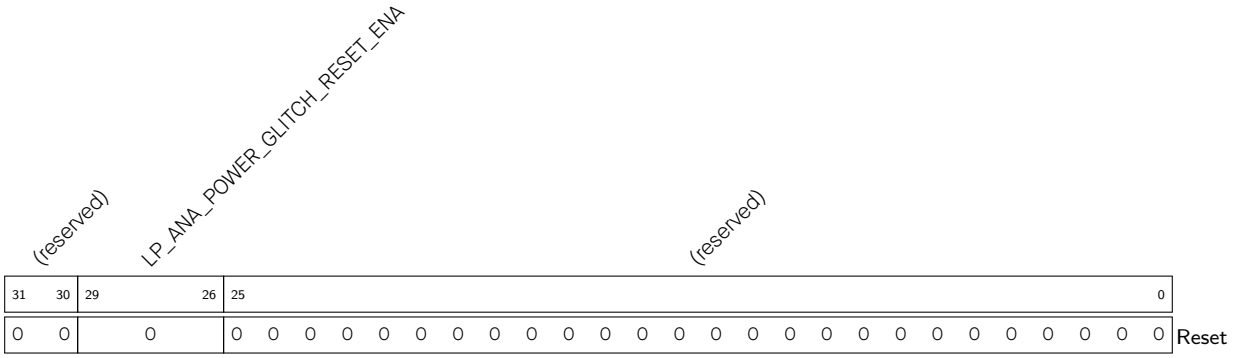
LP_ANA_BOD_MODE1_RESET_ENA Configures whether to enable brown-out detector mode 1.

0: Disable

1: Enable

(R/W)

Register 9.3. LP_ANA_PG_GLITCH_CNTL_REG (0x0018)



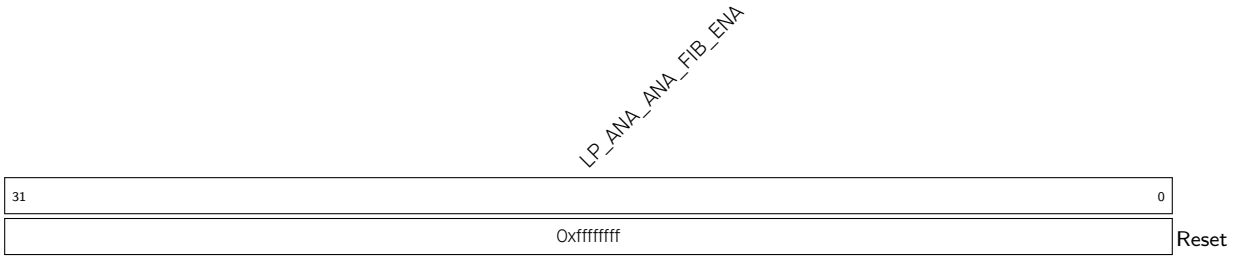
LP_ANA_POWER_GLITCH_RESET_ENA Configures the reset enable bitmap for voltage glitch.

0: Disable reset

1: Enable reset

(R/W)

Register 9.4. LP_ANA_FIB_ENABLE_REG (0x001C)



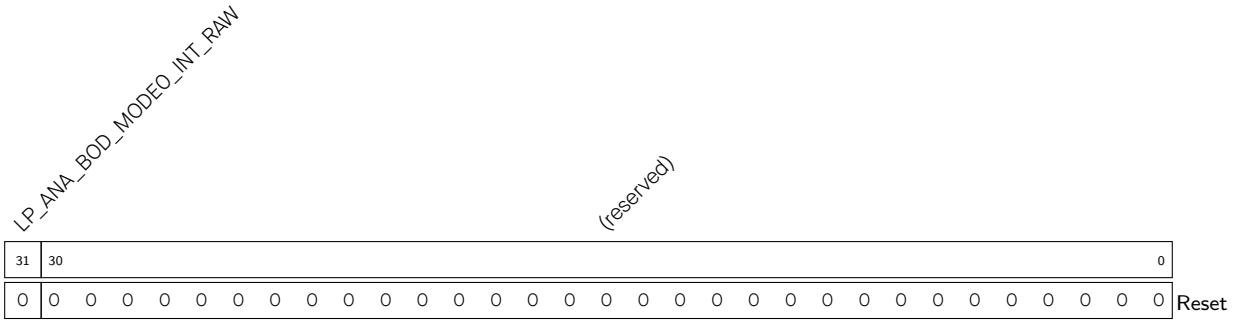
LP_ANA_ANA_FIB_ENA Configures the enable control mode for voltage glitch detectors and the brown-out detector. Bit2, bit3, bit4, bit5 correspond to VDDPST_2/3/4, VDDA1, VDDA2, and VDDPST_1, respectively; while bit1 corresponds to the brown-out detector.

0: bit2 ~bit5 are controlled by LP_ANA_POWER_GLITCH_RESET_ENA; bit1 is controlled by LP_ANA_BOD_MODE1_RESET_ENA

1: Forcibly enabled by hardware

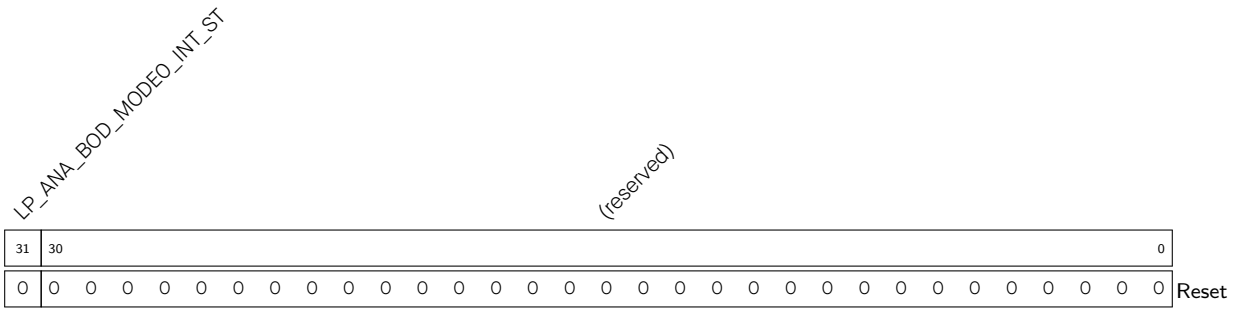
(R/W)

Register 9.5. LP_ANA_INT_RAW_REG (0x0020)



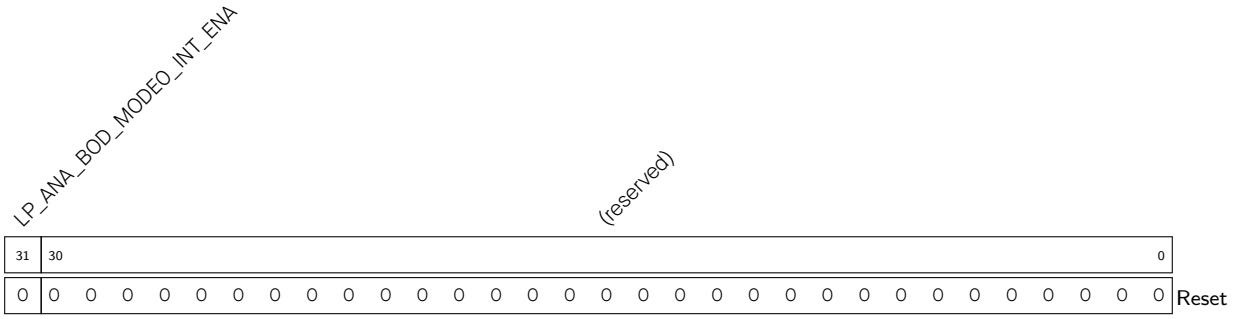
LP_ANA_BOD_MODE0_INT_RAW The raw interrupt status of LP_ANA_BOD_MODE0_INT. (R/WTC/SS)

Register 9.6. LP_ANA_INT_ST_REG (0x0024)



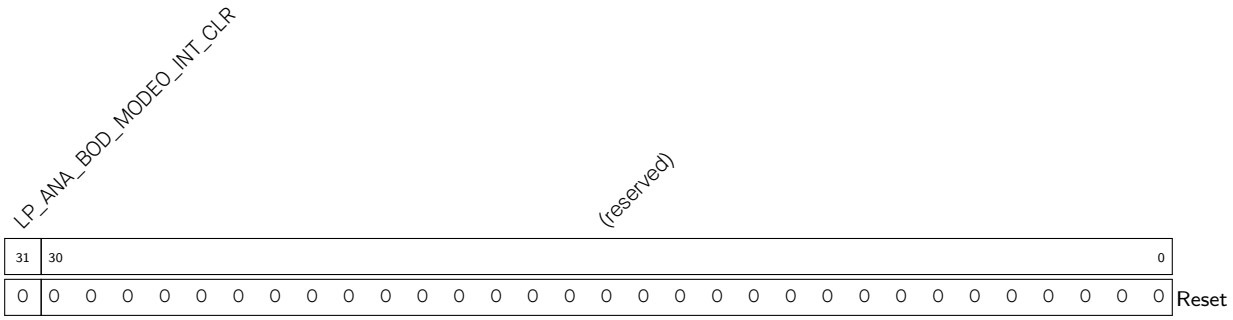
LP_ANA_BOD_MODE0_INT_ST The masked interrupt status of LP_ANA_BOD_MODE0_INT. (RO)

Register 9.7. LP_ANA_INT_ENA_REG (0x0028)



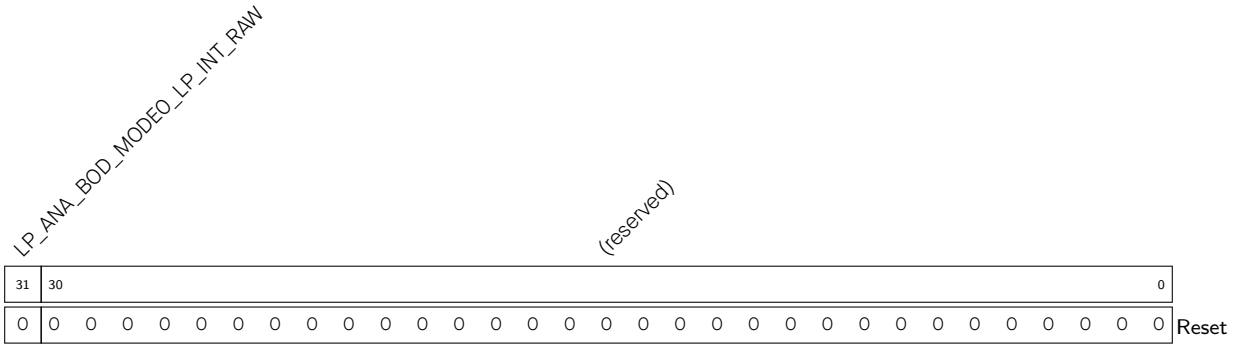
LP_ANA_BOD_MODE0_INT_ENA Write 1 to enable LP_ANA_BOD_MODE0_INT. (R/W)

Register 9.8. LP_ANA_INT_CLR_REG (0x002C)



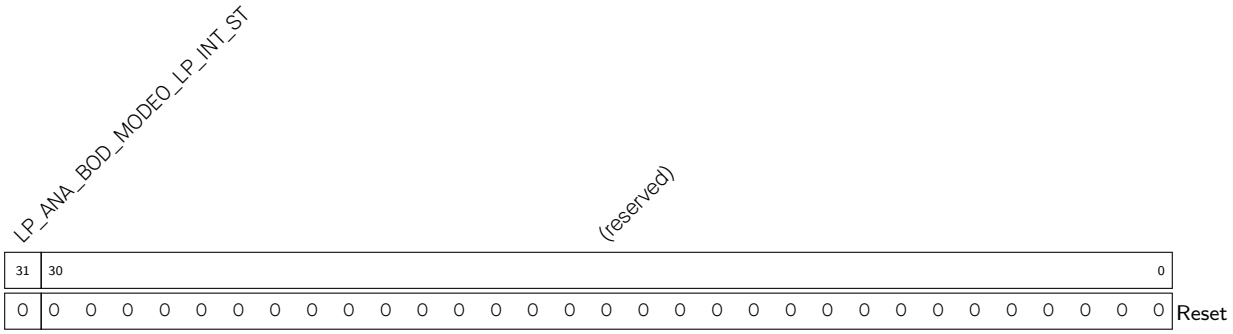
LP_ANA_BOD_MODE0_INT_CLR Write 1 to clear LP_ANA_BOD_MODE0_INT. (WT)

Register 9.9. LP_ANA_LP_INT_RAW_REG (0x0030)



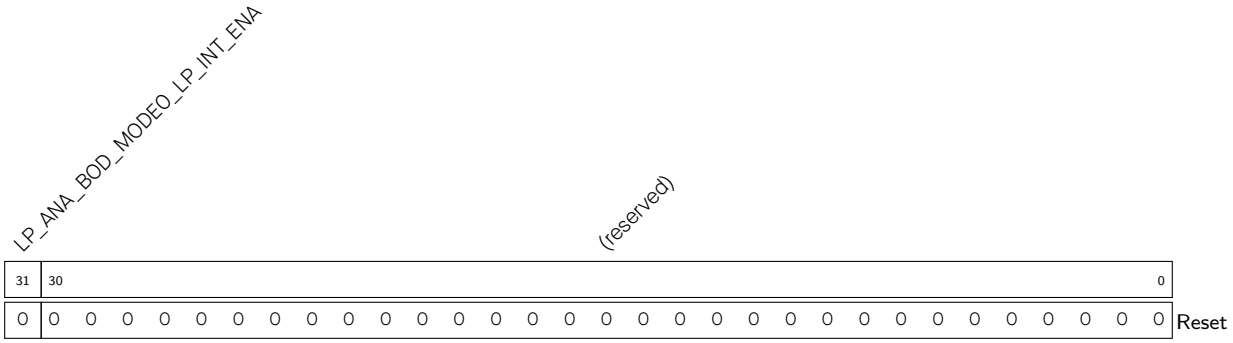
LP_ANA_BOD_MODE0_LP_INT_RAW The raw interrupt status of LP_ANA_BOD_MODE0_LP_INT. (R/WTC/SS)

Register 9.10. LP_ANA_LP_INT_ST_REG (0x0034)



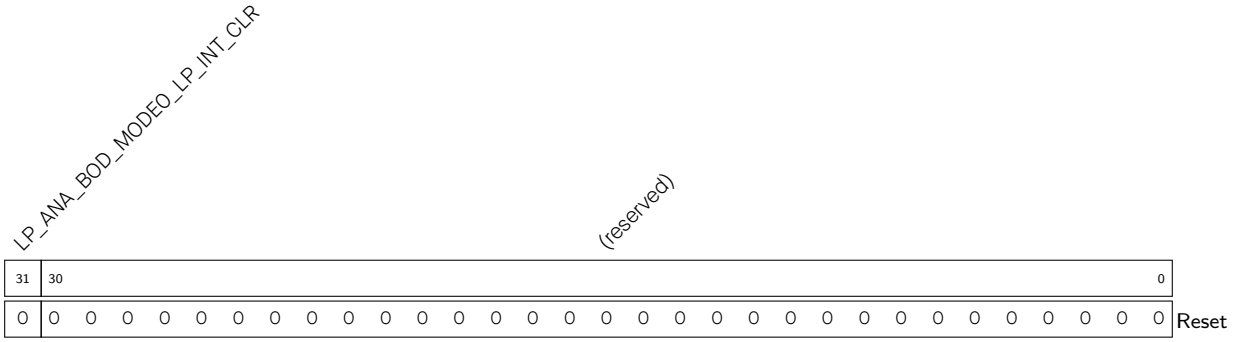
LP_ANA_BOD_MODE0_LP_INT_ST The masked interrupt status of LP_ANA_BOD_MODE0_LP_INT. (RO)

Register 9.11. LP_ANA_LP_INT_ENA_REG (0x0038)



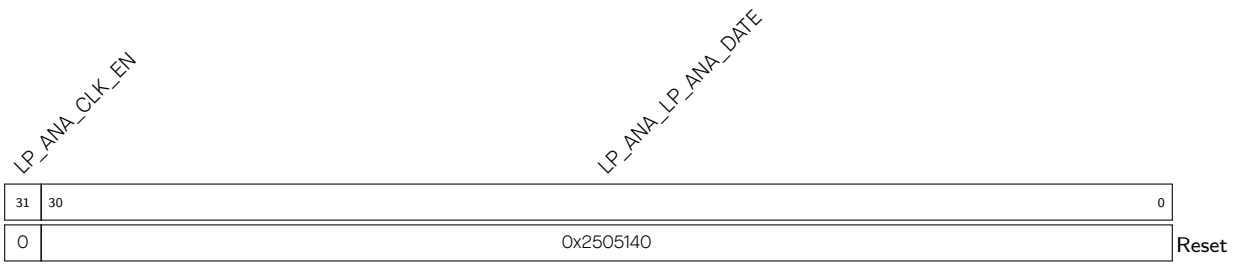
LP_ANA_BOD_MODE0_LP_INT_ENA Write 1 to enable LP_ANA_BOD_MODE0_LP_INT. (R/W)

Register 9.12. LP_ANA_LP_INT_CLR_REG (0x003C)



LP_ANA_BOD_MODE0_LP_INT_CLR Write 1 to clear LP_ANA_BOD_MODE0_LP_INT. (WT)

Register 9.13. LP_ANA_DATE_REG (0x03FC)



LP_ANA_LP_ANA_DATE Version control register. (R/W)

LP_ANA_CLK_EN Configures whether to force enable register clock.

0: Automatic clock gating

1: Force enable register clock

The configuration of this field does not affect the access of registers.

(R/W)

Part III

Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like AES and ECC. It also covers random number generation and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

Chapter 10

AES Accelerator (AES)

10.1 Introduction

ESP32-S31 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-S31 has two working modes, which are Typical AES and DMA-AES.

10.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-256 encryption and decryption
- DMA-AES working mode
  - AES-128/AES-256 encryption and decryption
  - Block cipher mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - GCM (Galois/Counter Mode)
  - Interrupt on completion of computation

10.3 Clock and Reset

The AES accelerator is activated by setting the HP_SYS_CLKRST_REG_CRYPTO_AES_CLK_EN bit and clearing the HP_SYS_CLKRST_REG_CRYPTO_AES_RST_EN bit in the HP_SYS_CLKRST_CRYPTO_CTRL0_REG register.

10.4 AES Working Modes

The AES accelerator integrated in ESP32-S31 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197.

  In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197;
  - Supports block cipher modes ECB, CBC, OFB, CTR, CFB8, and CFB128 under NIST SP 800-38A.
  - Supports GCM under NIST SP 800-38D.

  In this working mode, the plaintext and ciphertext are written and read via DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 10.4-1 below.

Table 10.4-1. 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 10.4-2 below.

Table 10.4-2. 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 10.5 and Section 10.6 below.

Notice: ESP32-S31's RSA Digital Signature Peripheral (RSA_DS) module also calls the AES accelerator when working. By default, AES_FREE (with value 1) would allow RSA Digital Signature Peripheral (RSA_DS) module to immediately reset any current AES operation and use AES for its operations. Users should set AES_FREE to 0 in order to prevent this from happening.

10.5 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 10.5-1 below.

Table 10.5-1. 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.

10.5.1 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.
- Alternatively, the AES-128 or AES-256 key can be set via Key Manager [to be added later].

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.

10.5.2 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 10.5-2.

Table 10.5-2. 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 10.5-3 and Table 10.5-4.

Table 10.5-3. 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 10.5-4. 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.

GoBack

10.5.3 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. If using the key from Key Manager [to be added later], write 1 to the AES_USE_HARDWARE_KEY.
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 is completed.
6. 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.

10.6 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 10.6-1 below.

Table 10.6-1. Block Cipher Mode

AES_BLOCK_MODE_REG[2:0]	Block Cipher Mode
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	GCM (Galois/Counter Mode)
7	reserved

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 10.6-2 below.

Table 10.6-2. 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.

10.6.1 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 10.6-3 below.

Table 10.6-3. 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**($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

10.6.2 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:
  - 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 10.6-4 below.

Table 10.6-4. 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

10.6.3 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.

10.6.4 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.

10.6.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 $\cdots$ Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 10.6-4, 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.

10.6.6 Block Operation Process

1. Select one of DMA channels to connect with AES, configure the DMA linked list, and then start DMA. For details, please refer to Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG or allow the usage of Key Manager [to be added later]'s deployed AES key with AES_USE_HARDWARE_KEY.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 10.6-1.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 10.6.4.
   - 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-AHB, GDMA-AXI)*.
6. Clear interrupt by writing 1 to the AES_INT_CLEAR_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.

10.6.7 GCM Operation Process

1. Configure DMA chain and start DMA. For details on DMA, please refer to Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG or allow the usage of Key Manager [to be added later]'s deployed AES Key with AES_USE_HARDWARE_KEY.
   - Write 6 to the AES_BLOCK_MODE_REG register.
   - Initialize the AES_BLOCK_NUM_REG register. Details about this register are described in Section 10.6.4 *Block Number*.
   - Initialize the AES_AAD_BLOCK_NUM_REG register. Details about this register are described in Section 10.7.4 *AAD Block Number*.
   - Initialize the AES_REMAINDER_BIT_NUM_REG register. Details about this register is described in Section 10.7.5 *Number of Effective Bits of Incomplete Blocks*.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2. For details on the working status of AES Accelerator, please refer to Table 10.6-2 *Working Status under DMA-AES Working mode*. At this step, no interrupt occurs.
5. Obtain the H value from the AES_H_MEM memory.
6. Generate J_0 and write it to the AES_J0_MEM memory.
7. Continue operating by writing 1 to the AES_CONTINUE_REG register.
8. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs. For details on the working status of AES Accelerator, please refer to Table 10.6-2 *Working Status under DMA-AES Working mode*.
9. Obtain T_0 by reading AES_T0_MEM.
10. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

11. Clear interrupt by writing 1 to the AES_INT_CLEAR_REG register, if any AES interrupt occurred during the computation.
12. Exit DMA by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG becomes 0. Note that, you can exit DMA earlier, but only after Step 8 is completed.

10.7 GCM Algorithm

ESP32-S31's AES accelerator fully supports GCM Algorithm. In reality, the AAD, C and P that are longer than $2^{32} - 1$ bits are seldom used. Therefore, we specify that the length of AAD, C and P should be no longer than $2^{32} - 1$ here. Figure 10.7-1 below demonstrates how GCM encryption is implemented in the AES Accelerator of ESP32-S31.

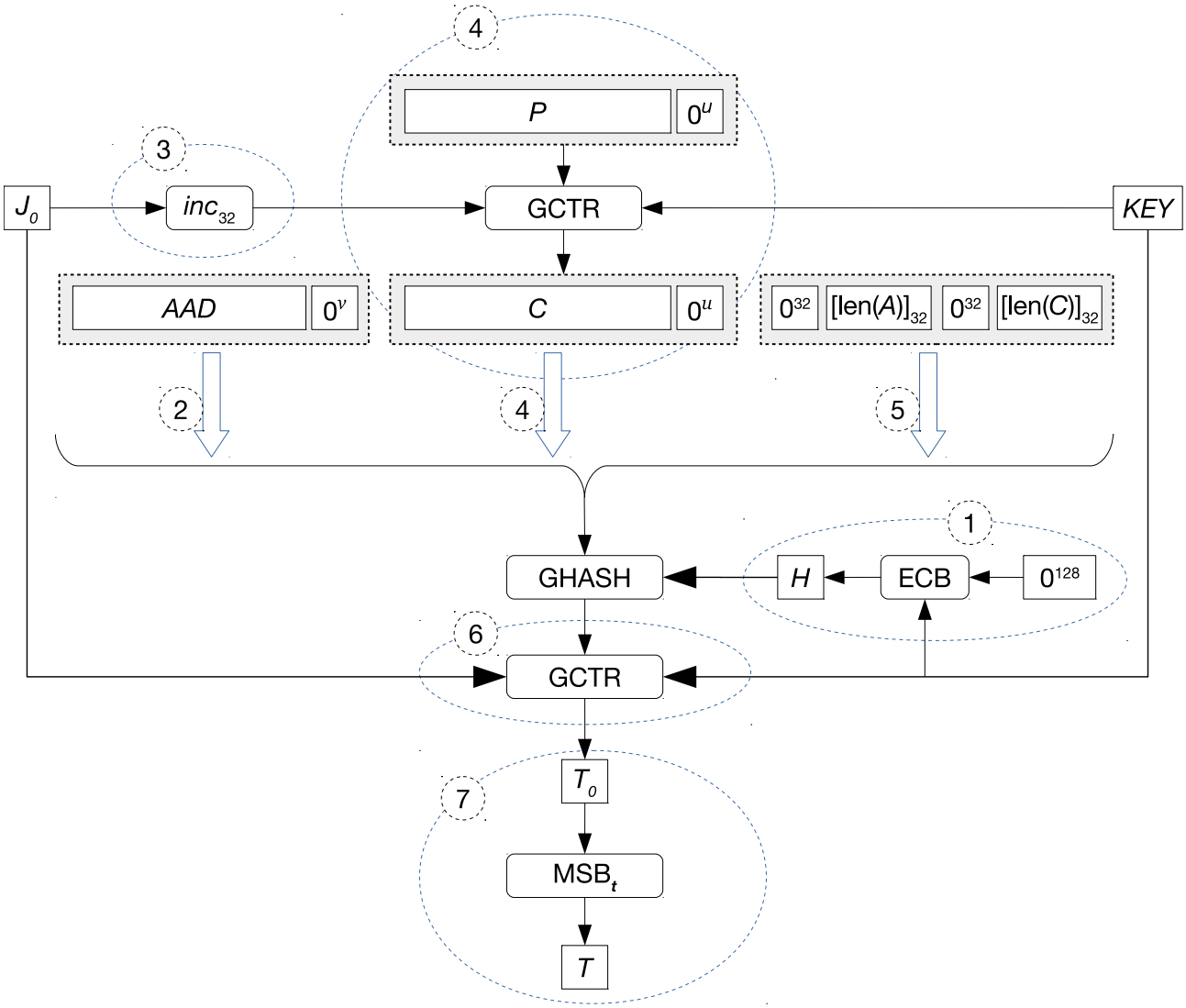


Figure 10.7-1. GCM Encryption Process

GCM encryption is implemented as follows:

1. Hardware executes the ECB Algorithm to obtain the Hash subkey H, which is needed in the Hash computation.
2. Hardware executes the GHASH Algorithm to perform Hash computation with the padded AAD.

3. Hardware gets ready for CTR encryption by obtaining the result of applying Standard Incrementing Function INC_{32} to J_0.
4. Hardware executes the GCTR Algorithm to encrypt the padded plaintext P, then executes the GHASH Algorithm to perform Hash computation on the padded ciphertext C.
5. Hardware executes the GHASH Algorithm to perform Hash computation on AAD Blocks, obtaining a 128-bit Hash result.
6. Hardware executes the GCTR Algorithm to encrypt J_0, obtaining T_0.
7. Software obtains the result T_0 from hardware, and executes MSB_t Algorithm to obtain the final authentication tag T.

The only difference between GCM decryption and GCM encryption lies in Step 4 in Figure 10.7-1. To be more specific, instead of executing GCTR Algorithm to encrypt the padded plaintext, the AES Accelerator executes the same algorithm to decrypt the padded ciphertext in GCM decryption. For details, please see NIST SP 800-38D.

10.7.1 Hash Subkey

During GCM operation, the Hash subkey H is a 128-bit value computed by hardware, which is demonstrated in Step 1 in Figure 10.7-1. Also you can find more information about Hash subkey at "Step 1. Let H = $\text{CIPH}_K(0^{128})$" in Chapter 7 GCM Specification of NIST SP 800-38D.

The Hash subkey H is stored in the AES_H_MEM memory. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 10.5-2.

10.7.2 J_0

J_0 is a 128-bit value computed by hardware, which is required during Step 3 and Step 6 in Figure 10.7-1. For details on the generation of J_0, please see Chapter 7 GCM Specification in NIST SP 800-38D.

J_0 is stored in the AES_J0_MEM memory. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 10.5-2.

10.7.3 Authentication Tag

Authentication Tag (Tag for short) is one of the key results of GCM computation, which is demonstrated in Step 7 of Figure 10.7-1. The value of the Tag is determined by its length t ($1 <= t <= 128$):

- When $t = 128$, the value of Tag equals to T_0, a 128-bit string that is stored in the AES_T0_MEM. Just like other endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 10.5-2.
- When $1 <= t < 128$, the value of Tag equals to the t most significant (i.e., left-most) bits of T_0. In this case, Tag is represented as $\text{MSB}_t(T_0)$, which returns the t most significant bits of T_0. For example, MSB_4(111011010) = 1110 and MSB_5(11010011010) = 11010. For details on the $\text{MSB}_t(\,)$ function, please refer to Chapter 6 Mathematical Components of GCM in NIST SP 800-38D.

10.7.4 AAD Block Number

Register AES_AAD_BLOCK_NUM_REG stores the Block Number of Additional Authenticated Data (AAD). The length of this register equals to (**TEXT-PADDING**(AAD) length)/128. AES Accelerator only uses this register when working in the DMA-AES mode.

10.7.5 Number of Effective Bits of Incomplete Blocks

Register AES_REMAINDER_BIT_NUM_REG stores the Remainder Bit Number, which indicates the number of effective bits of incomplete blocks in plaintext/ciphertext. The value stored in this register equals to length(P)/128 or length(C)/128. AES Accelerator only uses this register when working in the DMA-AES mode.

Register AES_REMAINDER_BIT_NUM_REG does not affect the results of plaintext or ciphertext, but does impact the value of T_0, therefore the Tag value too.

The GCM Algorithm can be viewed as the combination of GCTR operation and GHASH operation, among which, the GCTR performs the encryption and decryption, while the GHASH solves the Tag.

Note that the AES_REMAINDER_BIT_NUM_REG register is only effective for GCM encryption. To be more specific:

- For GCM encryption, the Hardware firstly computes C, then passes it in the form of **TEXT-PADDING**(C) as the input of GHASH operation. In this case, hardware determines how many trailing "0" should be added based on the content of AES_REMAINDER_BIT_NUM_REG.
- For GCM decryption, the padding is completed with the **TEXT-PADDING**(C) function. In this case, the AES_REMAINDER_BIT_NUM_REG register is not effective.

10.8 Pseudo-Round Anti-DPA Function

The pseudo-round function of ESP32-S31's AES can achieve higher anti-attack performance. When the pseudo-round function is enabled, AES will randomly add pseudo-rounds before and after the original operation rounds. In a pseudo-round, AES will use the pseudo-key, which is automatically generated by the hardware, to perform a round of fake operations. The operations in the pseudo-round do not affect the original AES operation results.

The configuration related to pseudo-round is as follows:

Mode of Pseudo-Round Function

Users can choose to enable pseudo-round function by configuring the AES_PSEUDO_EN register:

- 0: Not enable pseudo-round function;
- 1: Enable pseudo-round function.

Total number of Pseudo-Round

The total number of pseudo-rounds that will be randomly inserted into each AES operation round is controlled by:

- pseudo_base: Equal to the value of AES_PSEUDO_BASE, which configures the basic number of pseudo-rounds.

- pseudo_inc: Equal to the value of AES_PSEUDO_INC, which configures the random incremental number of pseudo-rounds.

The total number of pseudo-rounds will be randomly in the range $[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$.

Random Number Update Frequency

AES_PSEUDO_RNG_CNT configures the frequency of random key updates in the pseudo-round function. The higher the configured value, the higher the update frequency. This value is usually recommended to be set to maximum (7).

10.9 Interrupts

ESP32-S31's AES can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- AES_INTR

There is one internal interrupt source from AES that can generate the above interrupt signal(s). The interrupt source from AES are listed with its trigger condition and the resulted interrupt signal(s) in Table 10.9-1.

Table 10.9-1. AES's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
AES_DMA_DONE_INT	Completion of an AES DMA operation	AES_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

10.10 Memory Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Size (byte)	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W
AES_H_MEM	Memory H	16 bytes	0x0060	0x006F	RO
AES_J0_MEM	Memory J0	16 bytes	0x0070	0x007F	R/W
AES_T0_MEM	Memory T0	16 bytes	0x0080	0x008F	RO

10.11 Register Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

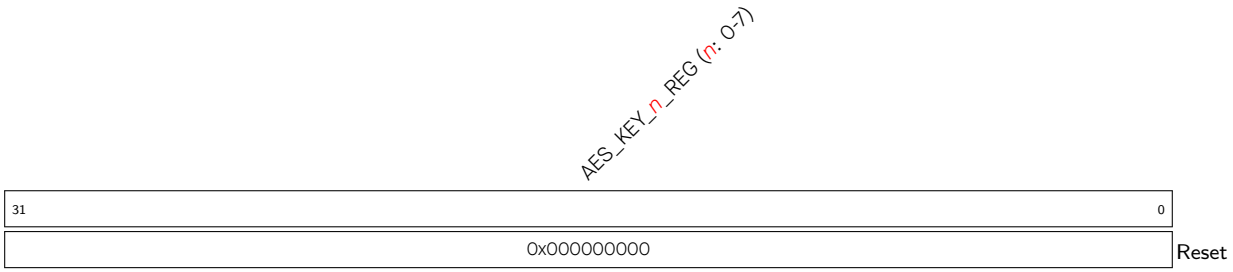
Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key data register 0	0x0000	R/W
AES_KEY_1_REG	AES key data register 1	0x0004	R/W
AES_KEY_2_REG	AES key data register 2	0x0008	R/W
AES_KEY_3_REG	AES key data register 3	0x000C	R/W
AES_KEY_4_REG	AES key data register 4	0x0010	R/W
AES_KEY_5_REG	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_TRIGGER_REG	Starts operation	0x0048	WT
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	Configures the block number	0x0098	R/W
AES_INC_SEL_REG	Configures the standard incrementing function	0x009C	R/W
AES_AAD_BLOCK_NUM_REG	Configures the AAD block number	0x00A0	R/W
AES_REMAINDER_BIT_NUM_REG	Configures the remainder bit number	0x00A4	R/W
AES_CONTINUE_REG	Continues operation	0x00A8	WT
AES_DMA_EXIT_REG	Exits operation	0x00B8	WT
AES_RX_RESET_REG	Resets RX FIFO	0x00C0	WT
AES_TX_RESET_REG	Resets TX FIFO	0x00C4	WT
AES_PSEUDO_REG	Configures the pseudo-round function	0x00D0	R/W
Status Registers			
AES_STATE_REG	Operation status register	0x004C	RO
AES_FREE_REG	Free state register	0x00D4	R/W
Interrupt Registers			
AES_INT_CLEAR_REG	DMA-AES interrupt clear register	0x00AC	WT
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W
Version Control register			
AES_DATE_REG	Version control register	0x00B4	R/W

10.12 Registers

The addresses in this section are relative to the AES accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

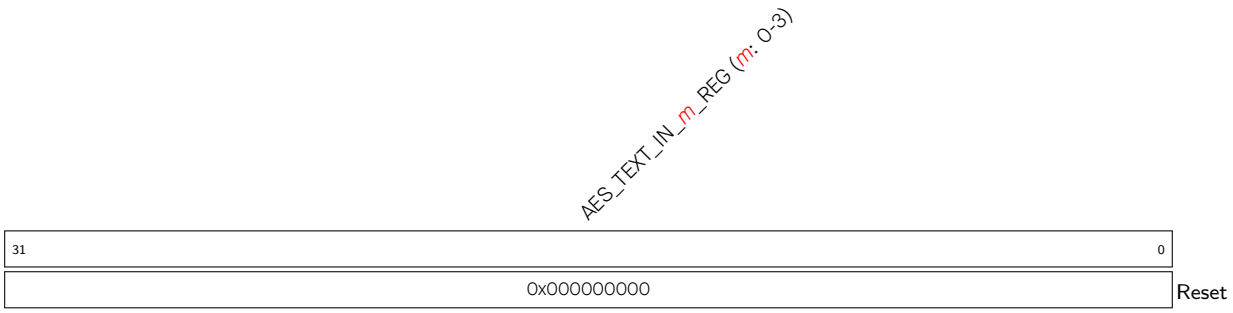
For how to program reserved fields, please refer to Section VII .

Register 10.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



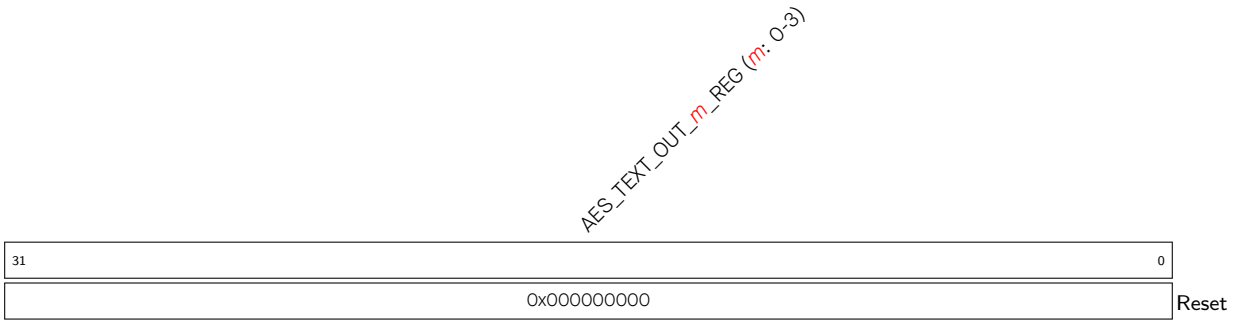
AES_KEY_*n*_REG (*n*: 0-7) Represents AES key data. (R/W)

Register 10.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



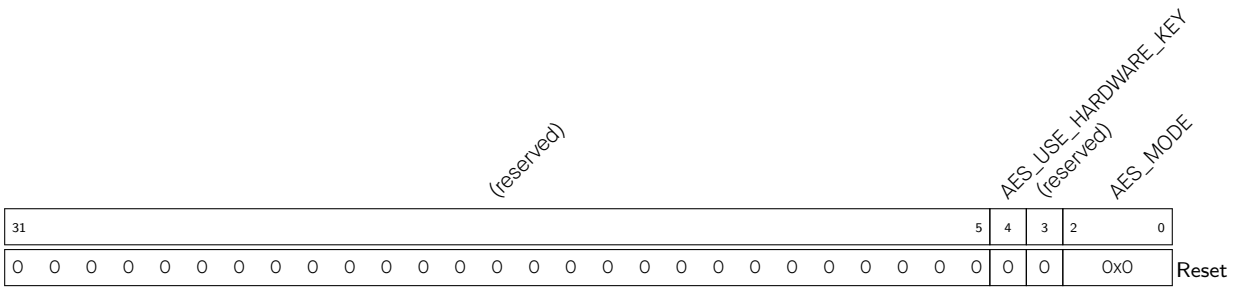
AES_TEXT_IN_*m*_REG (*m*: 0-3) Represents the source text data when the AES accelerator operates in the Typical AES working mode. (R/W)

Register 10.3. 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 10.4. 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)

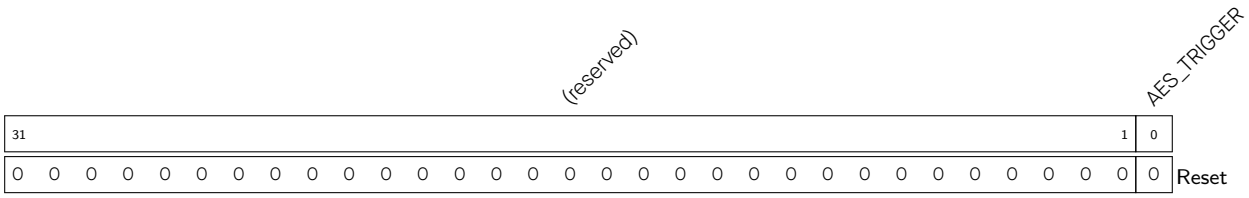
AES_USE_HARDWARE_KEY Configures whether the AES accelerator use software key or hardware key.

0: Software key.

1: Hardware key.

(R/W)

Register 10.5. AES_TRIGGER_REG (0x0048)



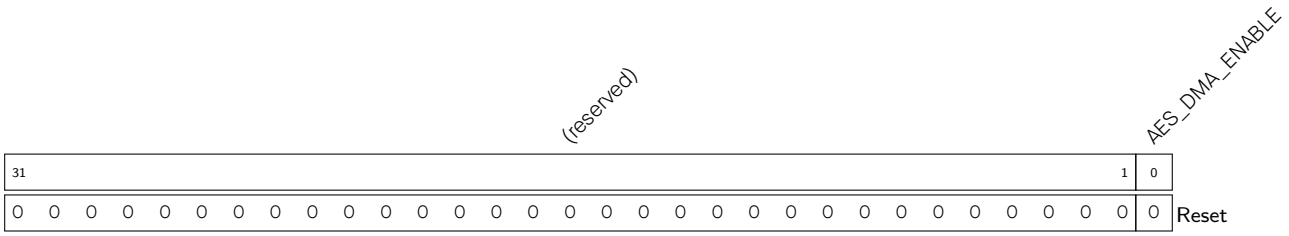
AES_TRIGGER Configures whether to start AES operation.

0: No effect

1: Start

(WT)

Register 10.6. AES_DMA_ENABLE_REG (0x0090)



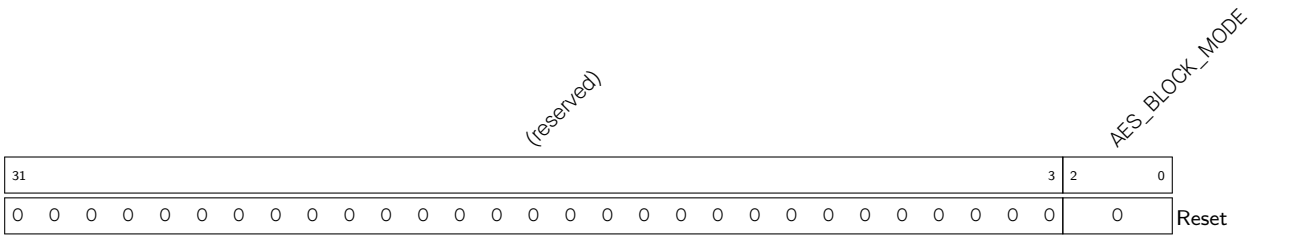
AES_DMA_ENABLE Configures the working mode of the AES accelerator.

0: Typical AES

1: DMA-AES

(R/W)

Register 10.7. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE Configures the block cipher mode of the AES accelerator operating under the DMA-AES working mode.

0: ECB (Electronic Code Block)

1: CBC (Cipher Block Chaining)

2: OFB (Output FeedBack)

3: CTR (Counter)

4: CFB8 (8-bit Cipher FeedBack)

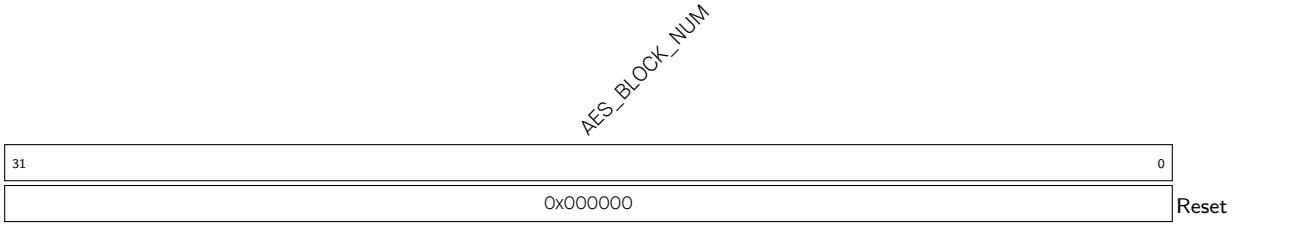
5: CFB128 (128-bit Cipher FeedBack)

6: GCM (Galois/Counter Mode)

7: Reserved

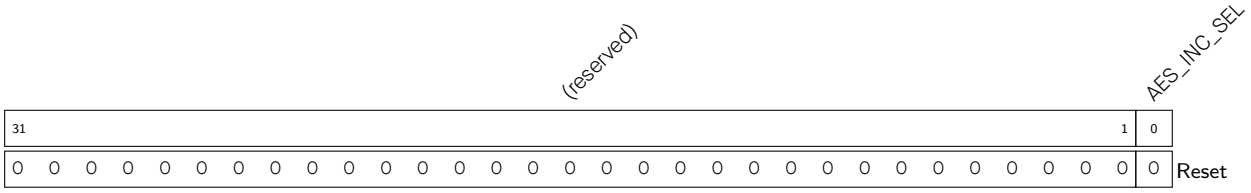
(R/W)

Register 10.8. 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 10.6.4. (R/W)

Register 10.9. AES_INC_SEL_REG (0x009C)



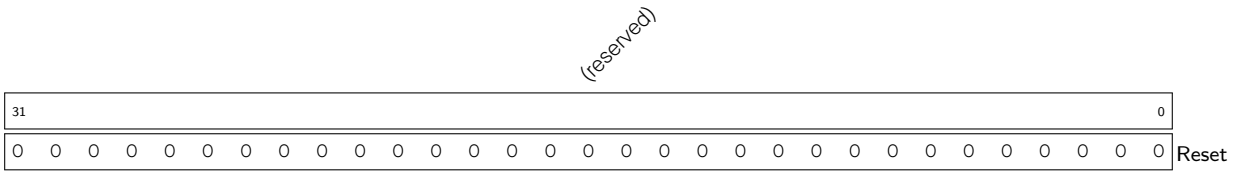
AES_INC_SEL Configures the Standard Incrementing Function for CTR block operation.

0: INC_{32}

1: INC_{128}

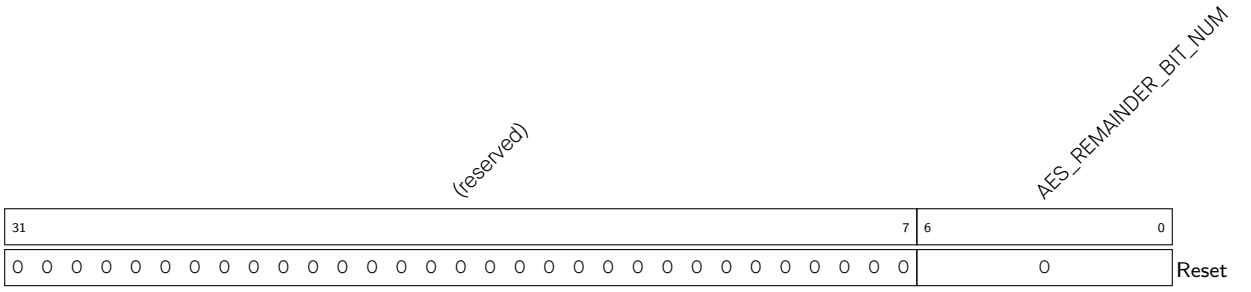
(R/W)

Register 10.10. AES_AAD_BLOCK_NUM_REG (0x00A0)



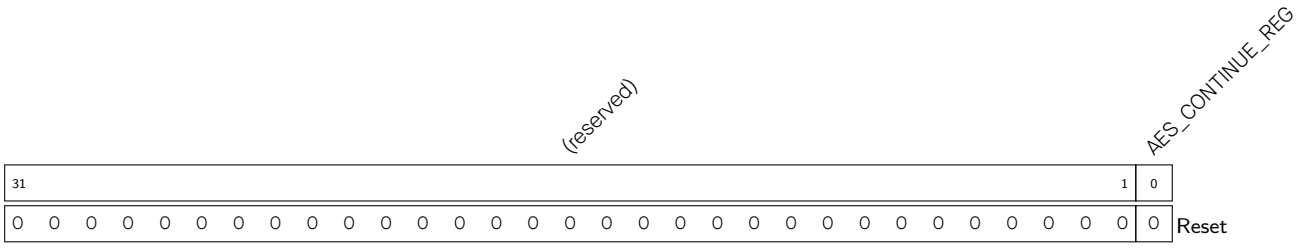
AES_AAD_BLOCK_NUM Represents the ADD Block Number for the GCM operation. For details, see Section 10.7.4. (R/W)

Register 10.11. AES_REMAINDER_BIT_NUM_REG (0x00A4)



AES_REMAINDER_BIT_NUM Represents the Remainder Bit Number for the GCM operation. For details, see Section 10.7.5. (R/W)

Register 10.12. AES_CONTINUE_REG (0x00A8)



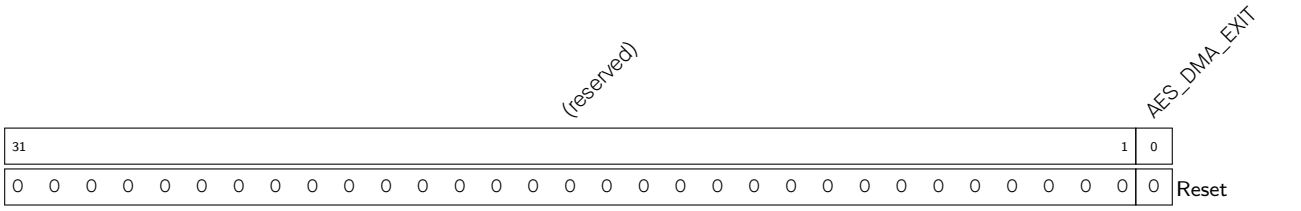
AES_CONTINUE_REG Configures whether to continue AES operation.

0: No effect

1: Continue

(WT)

Register 10.13. AES_DMA_EXIT_REG (0x00B8)



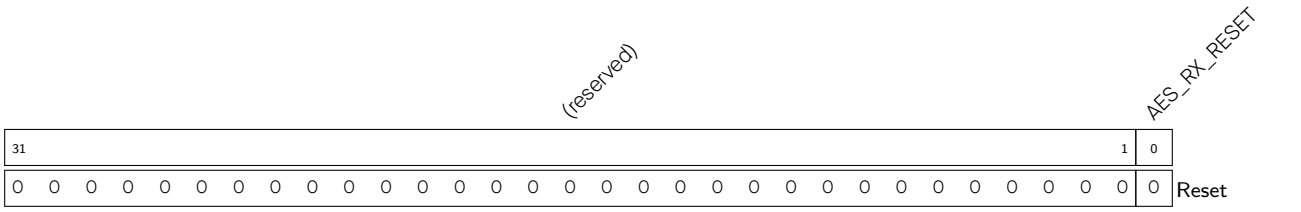
AES_DMA_EXIT Configures whether to exit AES operation.

0: No effect

1: Exit

Only valid for DMA-AES operation. (WT)

Register 10.14. AES_RX_RESET_REG (0x00C0)



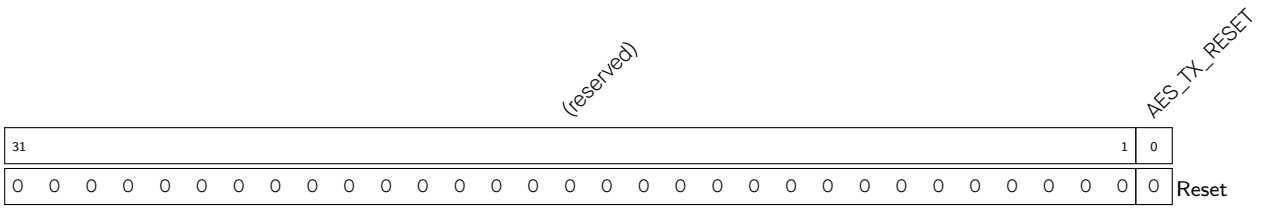
AES_RX_RESET Configures whether to reset RX_FIFO under DMA-AES working mode.

0: No effect

1: Reset

(WT)

Register 10.15. AES_TX_RESET_REG (0x00C4)



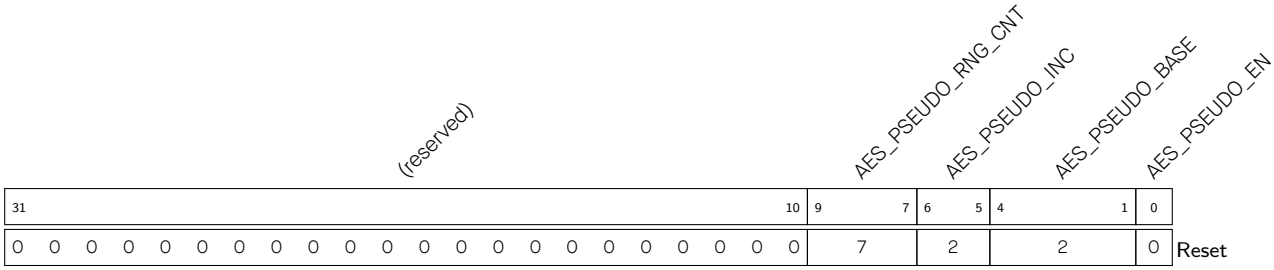
AES_TX_RESET Configures whether to reset TX_FIFO under DMA-AES working mode.

0: No effect

1: Reset

(WT)

Register 10.16. AES_PSEUDO_REG (0x00D0)



AES_PSEUDO_EN Configures whether to enable the pseudo-round function of AES.

0: Disable

1: Enable

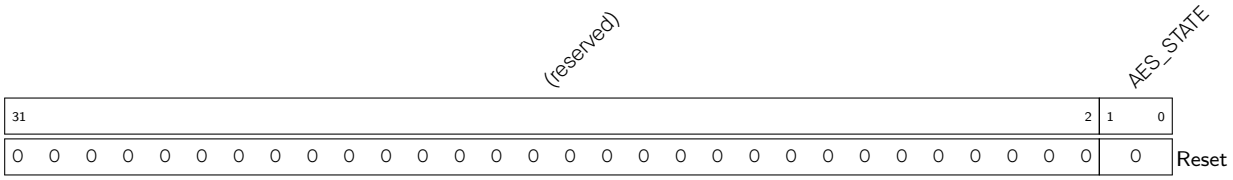
(R/W)

AES_PSEUDO_BASE Configures the basic number of pseudo-rounds. (R/W)

AES_PSEUDO_INC Configures the random incremental number of pseudo-rounds. (R/W)

AES_PSEUDO_RNG_CNT Configures the frequency of pseudo-key updates in the pseudo-round function. (R/W)

Register 10.17. 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

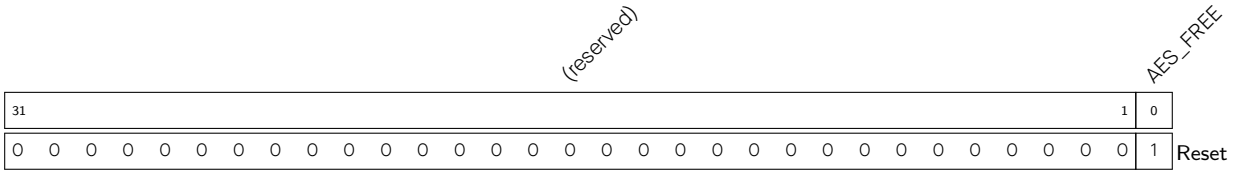
1: WORK

2: DONE

3: No effect

(RO)

Register 10.18. AES_FREE_REG (0x00D4)



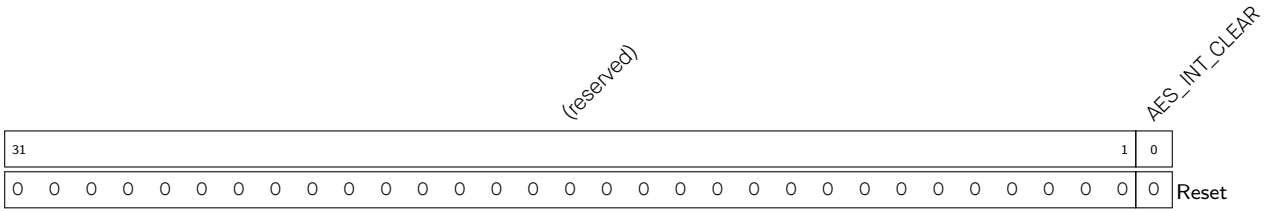
AES_FREE Configures whether AES can be set free (even if AES is in operation) and be occupied by another module that needs it.

0: AES cannot be set free

1: AES can be set free

(R/W)

Register 10.19. AES_INT_CLEAR_REG (0x00AC)



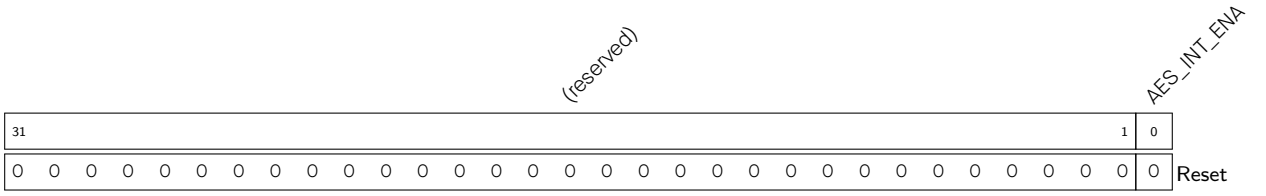
AES_INT_CLEAR Configures whether to clear the AES interrupt.

0: No effect

1: Clear

(WT)

Register 10.20. AES_INT_ENA_REG (0x00B0)



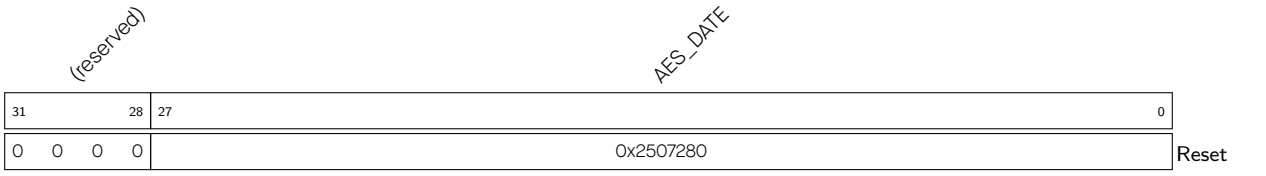
AES_INT_ENA Configures whether to enable the AES interrupt.

0: Disable

1: Enable

(R/W)

Register 10.21. AES_DATE_REG (0x00B4)



AES_DATE Version control register. (R/W)

Chapter 11

ECC Accelerator (ECC)

11.1 Overview

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves. ECC uses smaller keys compared to RSA cryptography while providing equivalent security.

ESP32-S31's ECC accelerator can complete various calculations based on different elliptic curves, thus accelerating the ECC algorithm and ECC-derived algorithms such as ECDSA.

11.2 Feature List

ESP32-S31's ECC accelerator has the following features:

- Four different elliptic curves, namely, P-192, P-256, and P-384 defined in FIPS 186-5 and SM2 defined in GM/T 0003.2-2012
- 11 working modes
- Enhanced anti-attack performance
- Interrupt upon completion of calculation and interrupt control

11.3 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.

11.3.1 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.

11.3.2 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:

$$X = x$$

$$Y = y$$

$$Z = 1$$

11.3.3 Memory Blocks

ECC's memory blocks store the input and output data of the ECC operation.

Table 11.3-1. ECC Accelerator Memory Blocks

Memory Block[1]	Size (byte)	Starting Address[2]	Ending Address [2]	Access
ECC_MULT_Mem_k	48	0x100	0x12F	R/W
ECC_MULT_Mem_Px	48	0x130	0x15F	R/W
ECC_MULT_Mem_Py	48	0x160	0x18F	R/W
ECC_MULT_Mem_Qx	48	0x190	0x1BF	R/W
ECC_MULT_Mem_Qy	48	0x1C0	0x1EF	R/W
ECC_MULT_Mem_Qz	48	0x1F0	0x21F	R/W

[1] The memory blocks store different types of data in different working modes. Refer to Section 11.4.2 for more details.

[2] Address offset related to the ECC accelerator base address is provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

11.3.4 Data and Data Block

ESP32-S31's ECC can operate on 192-bit, 256-bit, or 384-bit data, depending on the elliptic curve.

For example, the 256-bit data $D[255:0]$ can be divided into eight 32-bit data blocks $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with smaller indexes correspond to lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

11.3.5 Writing Data

Writing data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be specific, writing data to an ECC memory block means writing $D[n][31:0](n = 0, 1, \cdots, N-1)$ to the "starting address of this ECC memory block + $4 \times n$" sequentially, where N is the number of 32-bit data blocks required by the selected curve (6 for 192-bit, 8 for 256-bit, or 12 for 384-bit):

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[N-1]$ to "starting address + $4 \times (N-1)$"

Note:
When storing data, write only the data block of the required length. Do not zero-pad the most significant bits. For example, for 192-bit data, store only the 192 bits without appending 0.

11.3.6 Reading Data

Reading data means reading data from the starting address of an ECC memory block and using this data as the output from the ECC algorithm. To be specific, reading data from an ECC memory block means reading $D[n][31:0](n = 0, 1, \cdots, N-1)$ from the "starting address of this ECC memory block + $4 \times n$" successively, where N is the number of 32-bit data blocks required by the selected curve (6 for 192-bit, 8 for 256-bit, or 12 for 384-bit):

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[N-1]$ from "starting address + $4 \times (N-1)$"

Note:
When reading data, only the data block of the required length needs to be read. For example, to read 192-bit data, simply read the lower 192 bits (i.e., 6 data blocks).

11.3.7 Standard Calculation and Jacobian Calculation

ESP32-S31'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.

11.4 Function Description

11.4.1 Curve Mode

ESP32-S31's ECC supports acceleration based on four curve modes, each corresponding to an elliptic curve. By configuring the ECC_MULT_CURVE_MODE field, users can select the desired curve mode. For details, see Table 11.4-1 below. Support for the Chinese cryptographic standard (SM2) can be disabled by EFUSE_DIS_SM_CRYPT.

Table 11.4-1. ECC Accelerator Curve Mode Selection

ECC_MULT_CURVE_MODE	Elliptic Curves*
0	FIPS P-192
1	FIPS P-256
2	FIPS P-384
3	GM/T SM2

* See definition of FIPS P-192, P-256, and P-384 defined in FIPS 186-5 and SM2 defined in GM/T 0003.2-2012.

11.4.2 Working Modes

ESP32-S31's ECC accelerator supports 11 working modes, all of which are applicable across the four 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 11.4-2.

Table 11.4-2. 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.

11.4.2.1 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.
  - J_x, J_y, and J_z are stored in ECC_MULT_Mem_Qx, ECC_MULT_Mem_Qy, and ECC_MULT_Mem_Qz.

11.4.2.2 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.

11.4.2.3 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.

11.4.2.4 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.

11.4.2.5 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.

11.4.2.6 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.

11.4.2.7 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.

11.4.2.8 Mod Addition (Mod Add)

Mod Addition can be represented as:

$$R = A + B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    - ECC_MULT_CURVE_MODE to select the related curve.
    - ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Px.

11.4.2.9 Mod Subtraction (Mod Sub)

Mod Subtraction can be represented as:

$$R = A - B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    - ECC_MULT_CURVE_MODE to select the related curve.
    - ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Px.

11.4.2.10 Mod Multiplication (Mod Multi)

Mod Multiplication can be represented as:

$$R = A \cdot B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.

  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Py.

11.4.2.11 Mod Division (Mod Div)

Mod Division can be represented as:

$$R = A \cdot B^{-1} \bmod N$$

where,

- Input:
  - A and B are stored in ECC_MULT_Mem_Px and ECC_MULT_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_MULT_CURVE_MODE to select the related curve.
    * ECC_MULT_MOD_BASE to choose using mod base or order of the base point.
- Output: R is stored in ECC_MULT_Mem_Py.

11.4.3 Enhancing Anti-Attack Performance

When performing different types of point multiplication calculations, i.e., modes 0, 3, 4, and 7, ESP32-S31'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 performs a point multiplication calculation:

- The execution latency is constant for a given operation.
- Power consumption variations are minimized.

11.4.4 Clock

ESP32-S31's ECC uses two types of clocks:

- CRYPTO_ECC_CLK: Functional clock used by the ECC module
- CRYPTO_ECC_APB_CLK: Bus clock used to configure ECC registers

~~ECC's clock and reset are handled by the Power/Clock/Reset (PCR) module (see Chapter Reset and Clock [to be added later] for more information).~~

The conditions to enable the two clocks are as follows:

- CRYPTO_ECC_CLK: Jointly controlled by the clock gating registers HP_SYS_CLKRST_REG_CRYPTO_SYS_CLK_EN and HP_SYS_CLKRST_REG_CRYPTO_ECC_CLK_EN. When both registers are set to 1, CRYPTO_ECC_CLK is enabled. Its clock division is also controlled by HP_SYS_CLKRST_DPA_CTRLO_REG and the eFuse field EFUSE_SEC_DPA_LEVEL (see Section 7 *Reset and Clock [to be added later]* for more information).

- CRYPTO_ECC_APB_CLK: Jointly controlled by the clock gating registers HP_SYS_CLKRST_REG_CRYPTO_SEC_CLK_EN and HP_SYS_CLKRST_REG_CRYPTO_ECC_CLK_EN. When both registers are set to 1, CRYPTO_ECC_APB_CLK is enabled.

Before configuring or starting the ECC accelerator, make sure both CRYPTO_ECC_CLK and CRYPTO_ECC_APB_CLK are enabled.

Notice: ESP32-S31's ECDSA Digital Signature Peripheral (ECDSA_DS) module also calls the ECC accelerator when working. By default, when ECC_MULT_FREE is set to 1, this allows other modules (such as ECDSA Digital Signature Peripheral (ECDSA_DS)) to preempt the ECC accelerator resources, even if an ECC calculation is in progress. To ensure the current ECC operation is not interrupted, users should set ECC_MULT_FREE to 0.

11.4.5 Reset

ESP32-S31's ECC uses one reset signal, CRYPTO_ECC_RST, to reset ECC functional logic and configuration registers. Set HP_SYS_CLKRST_REG_CRYPTO_ECC_RST_EN to 0 to release CRYPTO_ECC_RST.

It is recommended to release CRYPTO_ECC_RST only after both clocks are enabled, and then configure registers and start ECC operations.

11.5 Interrupts

ESP32-S31's ECC accelerator can generate the ECC_INTR interrupt signal that will be sent to the Interrupt Matrix.

ECC_INTR has only one interrupt source to generate the ECC_INTR interrupt signal, i.e., ECC_MULT_CALC_DONE_INT, which is triggered on the completion of an ECC calculation.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The ECC_MULT_CALC_DONE_INT interrupt source is configured by the following registers (refer to Section 11.7 *Register Summary* for more information):

- ECC_MULT_CALC_DONE_INT_RAW: Stores the raw interrupt status of ECC_MULT_CALC_DONE_INT.
- ECC_MULT_CALC_DONE_INT_ST: Indicates the status of the ECC_MULT_CALC_DONE_INT interrupt. This bit is generated by enabling or disabling the ECC_MULT_CALC_DONE_INT_RAW bit via ECC_MULT_CALC_DONE_INT_ENA.
- ECC_MULT_CALC_DONE_INT_ENA: Enables or disables the ECC_MULT_CALC_DONE_INT interrupt.
- ECC_MULT_CALC_DONE_INT_CLR: Set this bit to clear the ECC_MULT_CALC_DONE_INT interrupt status. By setting this bit to 1, the ECC_MULT_CALC_DONE_INT_RAW and ECC_MULT_CALC_DONE_INT_ST bits will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

11.6 Programming Procedures

The programming procedure for configuring ECC is described below:

1. Configure the ECC clock and reset. Refer to Section 11.4.4 for detailed information.
2. Select the curve mode and working mode as described in Section 11.4.
3. Enable the ECC_MULT_CALC_DONE_INT interrupt as described in Section 11.5.
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 11.4.2.

11.7 Register Summary

The addresses in this section are relative to ECC accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

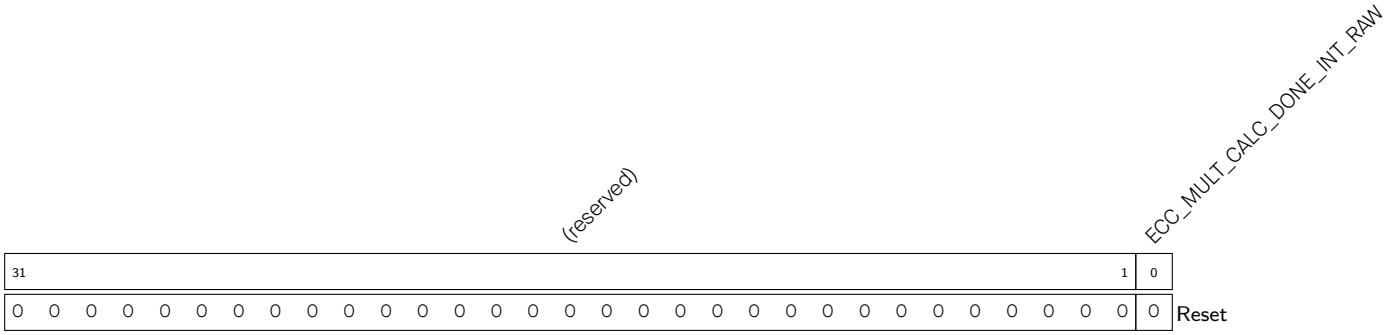
Name	Description	Address	Access
Interrupt Registers			
ECC_MULT_INT_RAW_REG	ECC raw interrupt status register	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	ECC masked interrupt status register	0x0010	RO
ECC_MULT_INT_ENA_REG	ECC interrupt enable register	0x0014	R/W
ECC_MULT_INT_CLR_REG	ECC interrupt clear register	0x0018	WT
Configuration Register			
ECC_MULT_CONF_REG	ECC configuration register	0x001C	varies
ECC_MULT_FREE_REG	ECC preemption configuration register	0x0020	R/W
Version Register			
ECC_MULT_DATE_REG	Version control register	0x00FC	R/W

11.8 Registers

The addresses in this section are relative to ECC accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

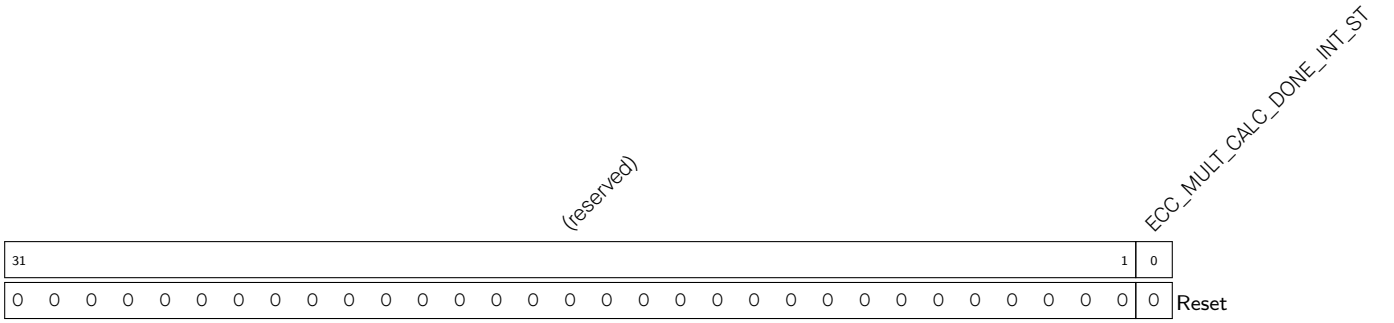
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 11.1. ECC_MULT_INT_RAW_REG (0x000C)



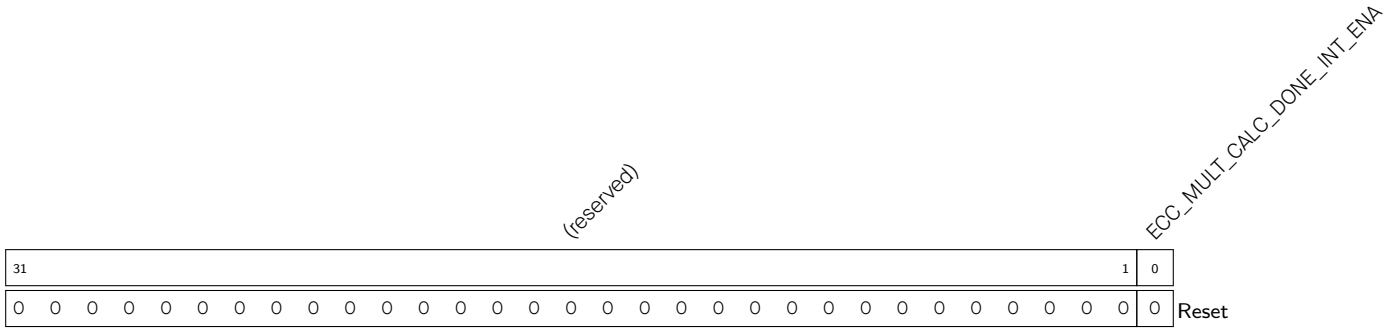
ECC_MULT_CALC_DONE_INT_RAW The raw interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (R/SS/WTC)

Register 11.2. ECC_MULT_INT_ST_REG (0x0010)



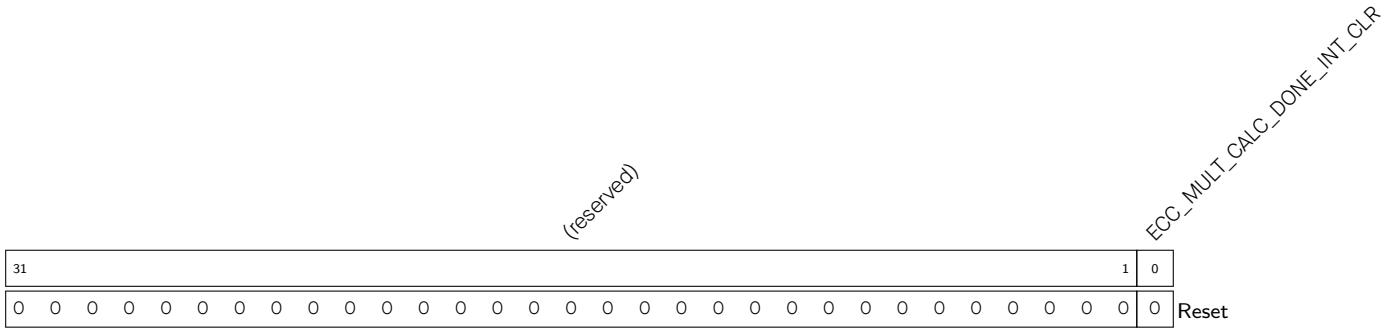
ECC_MULT_CALC_DONE_INT_ST The masked interrupt status of the ECC_MULT_CALC_DONE_INT interrupt. (RO)

Register 11.3. 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 11.4. 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 11.5. 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_CURVE_MODE	ECC_MULT_RESET	ECC_MULT_START	
31	30	29	28 … 10	9	8 … 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	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_CURVE_MODE Configures the curve mode of the ECC accelerator.

0: P-192

1: P-256

2: P-384

3: SM2

(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)

Continued on the next page...

Register 11.5. ECC_MULT_CONF_REG (0x001C)

Continued from the previous page...

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)

ECC_MULT_SECURITY_MODE Configures whether to enhance the anti-attack performance when calculating point multiplication. Only valid in working modes 0, 3, 4, and 7.

0: Disable
1: Enable
(R/W)

ECC_MULT_VERIFICATION_RESULT Represents the verification result of the ECC accelerator, valid only when the calculation is done.

0: Verification failed
1: Verification passed
(R/SS)

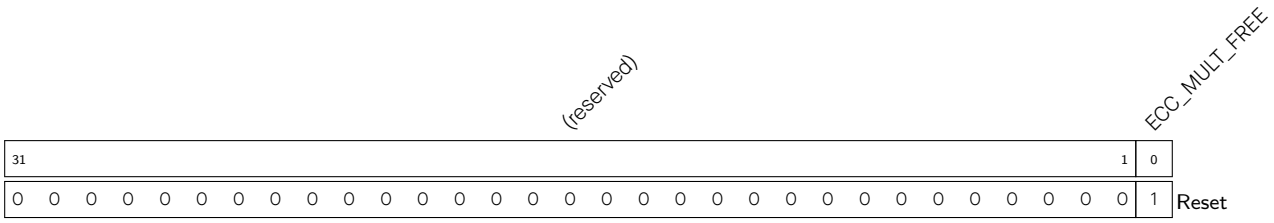
ECC_MULT_CLK_EN Configures whether to force on register clock gate.

0: No effect
1: Force on
(R/W)

ECC_MULT_MEM_CLOCK_GATE_FORCE_ON Configures whether to force the ECC memory clock gate on.

0: No effect
1: Force on
(R/W)

Register 11.6. ECC_MULT_FREE_REG (0x0020)



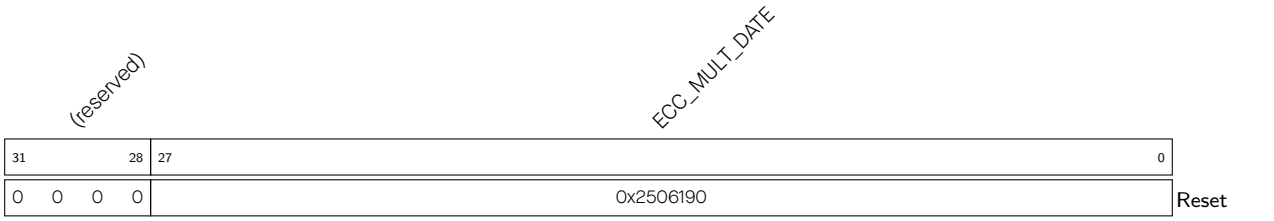
ECC_MULT_FREE Configures whether the ECC accelerator can be preempted by other modules. When set to 1, other modules that need ECC can interrupt the current operation and occupy the ECC resources, even if an ECC calculation is in progress.

0: ECC cannot be preempted

1: ECC can be preempted

(R/W)

Register 11.7. ECC_MULT_DATE_REG (0x00FC)



ECC_MULT_DATE Version control register. (R/W)

Chapter 12

HMAC Accelerator (HMAC)

12.1 Overview

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; that is, the key is not accessible from outside the HMAC accelerator.

12.2 Feature List

The HMAC module supports the following features:

- Standard HMAC-SHA-256 algorithm
- HMAC-SHA-256 calculation based on key in eFuse,
  - whose result cannot be accessed by software in downstream mode for high security
  - whose result can be accessed by software in upstream mode
- Generates required keys for the RSA Digital Signature Peripheral (RSA_DS) in downstream mode
- Re-enables soft-disabled JTAG in downstream mode

12.3 Functional Description

The HMAC module operates in two modes: upstream mode and downstream mode. In upstream mode, users provide the HMAC message and read back the calculation result.

In downstream mode, the HMAC module provides input to two possible other internal hardware modules. The downstream mode supports two primary use cases:

- **JTAG Re-enablement:** The HMAC module can re-enable JTAG after it has been temporarily disabled.
- **Key Derivation for RSA_DS:** The HMAC module can serve as the decryption key source for Digital Signature parameters stored in flash and used by the RSA_DS peripheral.

Furthermore, in downstream mode, all HMAC calculations are performed internally and automatically, ensuring the confidentiality of the key and any derived key material when the system is correctly configured.

Note:
After the reset signal is released, the HMAC module will check whether the RSA_DS_KEY exists in the eFuse. If the key exists, the HMAC module will enter downstream RSA_DS_KEY derivation mode and finish the key calculation

automatically. This process is automatically completed by hardware and does not require software participation. When the downstream operation after reset is completed, the HMAC will automatically return to the idle state.

12.3.1 Upstream Mode

To calculate the HMAC value in upstream mode, users should perform the following steps:

1. Initialize the HMAC module and enter upstream mode.
2. Write the correctly padded message to the HMAC, one block at a time.
3. Read back the result from HMAC.

For details of this process, please see Section 12.5.

Note:
Common use cases for the upstream mode are challenge-response protocols supporting HMAC-SHA-256. Assume the two entities in the challenge-response protocol are A and B respectively, and the data message they expect to exchange is M. The general authentication process of this protocol is as follows:

- A calculates a unique random number M.
- A sends M to B.
- B calculates the HMAC (through M and KEY) and sends the result to A.
- A calculates the HMAC (through M and KEY) internally.
- A compares the two results. If the results are the same, then the identity of B is authenticated.

12.3.2 Downstream Mode - JTAG Enable Feature

JTAG debugging can be disabled by eFuse in a way which allows later re-enabling using the HMAC or *Secure Debug Controller (SDC) [to be added later]* module. (For more details, please see Chapter 4 *eFuse Controller (eFuse) [to be added later]*.) By default, this function is controlled by the Secure Debug Controller module. However, setting EFUSE_RE_ENABLE_JTAG_SOURCE transfers control of JTAG re-enablement to 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.

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 12.5 for more details.
2. Write 1 to the HMAC_SOFT_JTAG_CTRL_REG register to enter JTAG re-enable mode.
3. Write the 256-bit HMAC value to register HMAC_WR_JTAG_REG. This value is obtained by performing a local HMAC calculation from the 32-byte 0x00 using SHA-256 and the key that has been written to the eFuse. It needs to be written 8 times, 32-bit each time in big-endian word order.

4. If the HMAC result calculated from the key in the eFuse matches the value that users wrote in step 3, then JTAG is re-enabled. Otherwise, JTAG remains disabled.
5. After writing 1 to HMAC_SET_INVALIDATE_JTAG_REG or resetting the chip, JTAG will be disabled. If users want to re-enable JTAG again, they need to repeat the above steps again.

12.3.3 Downstream Mode - RSA Digital Signature Peripheral and Key Derivation Feature

The RSA Digital Signature Peripheral (RSA_DS) module encrypts its parameters using the AES-CBC algorithm. The HMAC module is used as a Key Derivation Function (KDF) to derive the AES key to decrypt these parameters (parameter decryption key).

Before starting the RSA_DS module, users need to obtain the parameter decryption key for the RSA_DS module through HMAC calculation. For more information, please see Chapter 15 *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 digital signature algorithm mode and complete the HMAC calculation based on the chosen key.

12.4 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 12.4-1. 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, users 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 5 keys in eFuse, so the value of x is 0 ~ 4. EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_4 belong to the register EFUSE_RD_REPEAT_DATA2_REG) from 4 *eFuse Controller (eFuse) [to be added later]*. Take upstream mode as an example, if there is no EFUSE_KEY_PURPOSE_HMAC_UP in EFUSE_KEY_PURPOSE_0 ~ 4, it means there is no key in eFuse that can be used for the HMAC upstream mode. Users can burn a key to eFuse as follows:

1. Prepare a secret 256-bit HMAC key and burn the key to an empty eFuse block y. As there are 5 blocks for storing a key in eFuse and the numbers of those blocks range from 4 to 8, the value of y is 4 ~ 8. Hence, when talking about key0, it means eFuse block4. Then, program the purpose to EFUSE_KEY_PURPOSE_$(y-4)$. Take upstream mode as an example: after programming the key, the user should program EFUSE_KEY_PURPOSE_HMAC_UP (corresponding value is 8) to EFUSE_KEY_PURPOSE_$(y-4)$. Please see Chapter 4 *eFuse Controller (eFuse) [to be added later]* on how to program eFuse keys.
2. If needed, configure this eFuse key block to be read protected, so that users cannot read its value. A copy of this key should be kept by any party who needs to verify this device.

Please note that the key whose purpose is EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL can be used for both re-enabling JTAG or RSA_DS.

Table 12.4-1. HMAC Purposes and Configuration Value

Purpose(m)	Mode	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
RSA_DS KDF	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and RSA_DS KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Select eFuse Key Blocks and HMAC Purposes

The eFuse controller provides 5 key blocks, i.e., KEY0 ~ 4. To select a particular KEYn for an HMAC calculation, write the key number n to register HMAC_SET_PARA_KEY_REG.

Write a correct purpose to register HMAC_SET_PARA_PURPOSE_REG (see Section 12.5). Note that the purpose of the key has also been programmed to eFuse memory. Only when the configured HMAC purpose matches the defined purpose of KEYn, the HMAC module will execute the configured calculation. Otherwise, it will return a matching error and stop the current calculation.

For example, suppose a user selects KEY3 for HMAC calculation, and the value programmed to EFUSE_KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 12.4-1, KEY3 can be used to re-enable JTAG. If the value written to register HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC module will start the process to re-enable JTAG.

Select the Key from the Key Manager

Users can deploy the HMAC key in the key manager. To select this key from the key manager, write the key number 7 to register HMAC_SET_PARA_KEY_REG. When users choose the HMAC key from the key manager, the HMAC purpose will automatically be "HMAC Calculation", and any configuration to register HMAC_SET_PARA_PURPOSE_REG takes no effect.

12.5 HMAC Process (Detailed)

The process for users to call HMAC in ESP32-S31 is as follows:

12.5.1 Enable HMAC module

1. Set the peripheral clock bits for HMAC and SHA peripherals in register HP_SYS_CLKRST_REG_CRYPTO_HMAC_CLK_EN, and clear the corresponding peripheral reset bits in register HP_SYS_CLKRST_REG_CRYPTO_HMAC_RST_EN. For information on those registers, please see Chapter 7 *Reset and Clock [to be added later]*.
2. Write 1 to register HMAC_SET_START_REG.

12.5.2 Configure HMAC keys and key purposes

1. Write the key purpose m to register HMAC_SET_PARA_PURPOSE_REG. The possible key purpose values are shown in Table 12.4-1. For more information, please refer to Section 12.4.
2. Select KEYn in eFuse memory as the key by writing n (ranges from 0 to 4, and 7) to register HMAC_SET_PARA_KEY_REG. For more information, please refer to Section 12.4.
3. Write 1 to register HMAC_SET_PARA_FINISH_REG to complete the configuration.
4. Read register HMAC_QUERY_ERROR_REG. If its value is 1, it means the purpose of the selected block does not match the configured key purpose and the calculation will not proceed. If its value is 0, it means the purpose of the selected block matches the configured key purpose, and then the calculation can proceed.
5. When the value of HMAC_SET_PARA_PURPOSE_REG is not 8, it means the HMAC module is in downstream mode, proceed with 12.5.3. When the value is 8, it means the HMAC module is in upstream mode, proceed with 12.5.4.

12.5.3 Downstream mode process

1. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
2. To clear the result and make further usage of the dependent hardware (JTAG or RSA_DS) impossible, write 1 to either register HMAC_SET_INVALIDATE_JTAG_REG to clear the result generated by the JTAG key; or to register HMAC_SET_INVALIDATE_DS_REG to clear the result generated by RSA_DS_KEY. Afterwards, the HMAC process needs to be restarted to re-enable any of the dependent peripherals.

12.5.4 Upstream mode process

1. Write message in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Apply SHA padding to message as described in Section 12.6.1, and get message block: Block_n ($n >= 0$)
   (c) Perform one of the following operations according to the block number n:
   - If there is only one message block in total which has included all padding bits:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_ONE_BLOCK_REG, and finish message transmission.
   - If Block_n is the last padded block:
     i. Write the 512-bit Block_0 to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Message transmission finished automatically.

- If Block_n is the second last padded block:
  i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
  ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
  iii. Write 1 to register HMAC_SET_MESSAGE_PAD_REG, and continue message transmission (jump back to step(c)).
- If Block_n is neither the last nor the second last message block:
  i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
  ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
  iii. Write 1 to register HMAC_SET_MESSAGE_ING_REG, and continue message transmission (jump back to step(c)).

2. Read hash result in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Read hash result from register HMAC_RD_RESULT_*n*_REG (*n*: 0-7).
   (c) Write 1 to register HMAC_SET_RESULT_FINISH_REG to finish calculation. The result will be cleared at the same time.
   (d) Upstream mode operation is completed.

12.6 HMAC Algorithm Details

12.6.1 Padding Bits

The HMAC module uses SHA-256 as hash algorithm. If the input message is not a multiple of 512 bits, the user must apply a SHA-256 padding algorithm in software. The SHA-256 padding algorithm is the same as described in Section *Padding the Message* of FIPS PUB 180-4. In downstream mode, users do not need to input any message or apply padding. The HMAC module uses a default 32-byte pattern of 0x00 for re-enabling JTAG and a 32-byte pattern of 0xff for deriving the AES key for the RSA_DS module.

For the convenience of reading, here we will briefly describe the process of message padding. As shown in Figure 12.6-1, 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.

In upstream mode, if the length of the unpadded message is a multiple of 512 bits, users can configure hardware to apply SHA padding by writing 1 to HMAC_SET_MESSAGE_END_REG or do padding work themselves by writing 1 to HMAC_SET_MESSAGE_PAD_REG. If the length is not a multiple of 512 bits, SHA

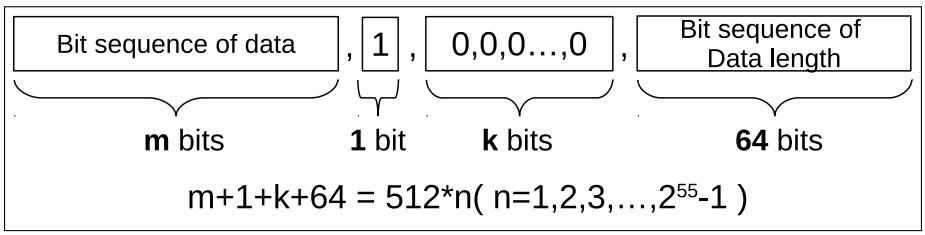


Figure 12.6-1. HMAC SHA-256 Padding Diagram

padding must be manually applied by the user. After users prepared the padding data, they should complete the subsequent configuration according to Section 12.5.

12.6.2 HMAC Algorithm Structure

The structure of the implemented algorithm in the HMAC module is shown in Figure 12.6-2. This is the standard HMAC algorithm as described in RFC 2104.

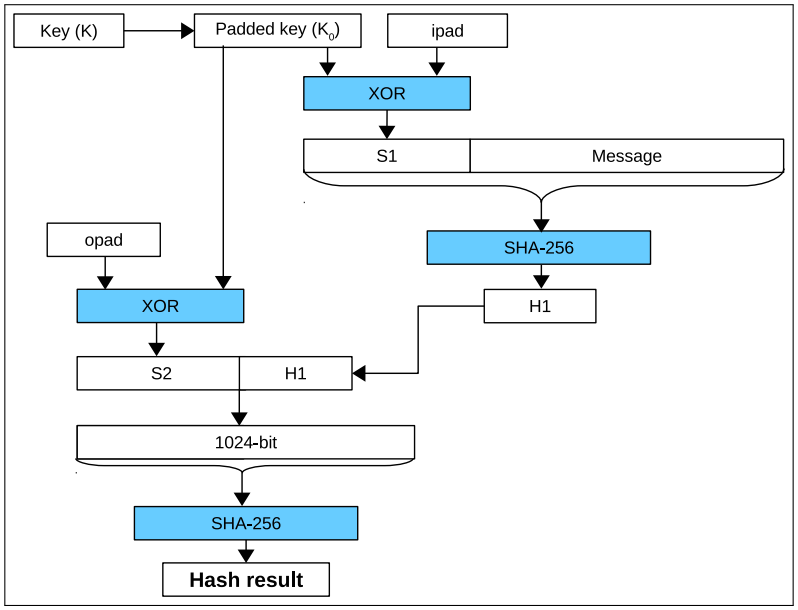


Figure 12.6-2. HMAC Structure Schematic Diagram

In Figure 12.6-2:

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 12.6.1 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).

12.6.3 Hardware Occupation

During the operation, the HMAC module of ESP32-S31 occupies the SHA accelerator. The occupancy rules are as follows:

- The SHA accelerator register field SHA_FREE indicates whether the accelerator is idle:
  - If this field is set to 1, HMAC immediately resets and occupies the SHA accelerator when the operation starts.
  - If this field is set to 0, HMAC waits until the current operation on the SHA accelerator completes before occupying it, or until the timeout configured by HMAC_TIMEOUT_LIMIT expires.
- HMAC retains ownership of the SHA accelerator throughout the operation and releases it only after the operation completes.
- If HMAC receives an external reset while occupying the SHA accelerator (configured through HP_SYS_CLKRST_REG_CRYPTO_HMAC_RST_EN), the reset is also propagated to the SHA accelerator. This mechanism clears any intermediate results temporarily stored inside the accelerator, preventing their disclosure through external reset attacks.

Note:

Hardware occupation is a mechanism to protect multiplexed modules and storage space. When a module is hardware occupied, the user will fail to:

- read or write data to the module's registers or memories.
- disable the module clock.
- reset the module.

After hardware occupation is finished, the occupied module will be automatically reset. In addition, when the user performs a software reset to the master module, all the occupied modules will be reset at the same time.

12.7 Register Summary

The addresses in this section are relative to HMAC Accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

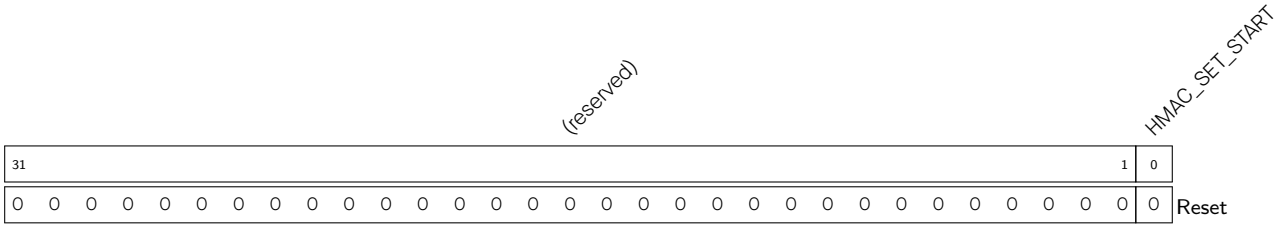
Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WS
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WS
HMAC_SET_MESSAGE_ONE_REG	HMAC message control register	0x0050	WS
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WS
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WS
HMAC_SET_RESULT_FINISH_REG	HMAC result reading finish register	0x005C	WS
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WS
HMAC_SET_INVALIDATE_DS_REG	Invalidate digital signature result register	0x0064	WS
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	WS
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_TIMEOUT_LIMIT_REG	HMAC timeout limit configuration register	0x0070	R/W
HMAC_WR_JTAG_REG	Re-enable JTAG register 1	0x00FC	WO
HMAC_SOFT_JTAG_CTRL_REG	Re-enable JTAG register 0	0x00F8	WS
HMAC Message Block			
HMAC_WR_MESSAGE_*n*_REG	Message register *n* (*n*: 0-15)	0x0080+4×*n*	WO
HMAC Upstream Result			
HMAC_RD_RESULT_*n*_REG	Hash result register *n* (*n*: 0-7)	0x00C0+4×*n*	RO
Version Register			
HMAC_DATE_REG	Version control register	0x01FC	R/W

12.8 Registers

The addresses in this section are relative to HMAC Accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 12.1. HMAC_SET_START_REG (0x0040)



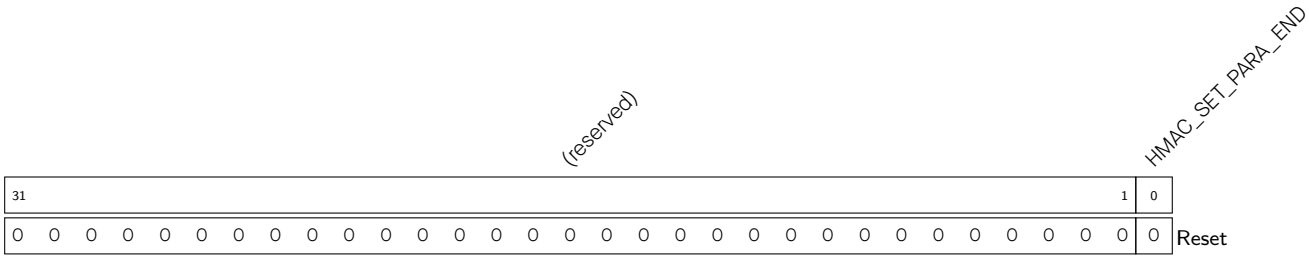
HMAC_SET_START Configures whether or not to enable HMAC.

0: Disable HMAC

1: Enable HMAC

(WS)

Register 12.2. HMAC_SET_PARA_FINISH_REG (0x004C)



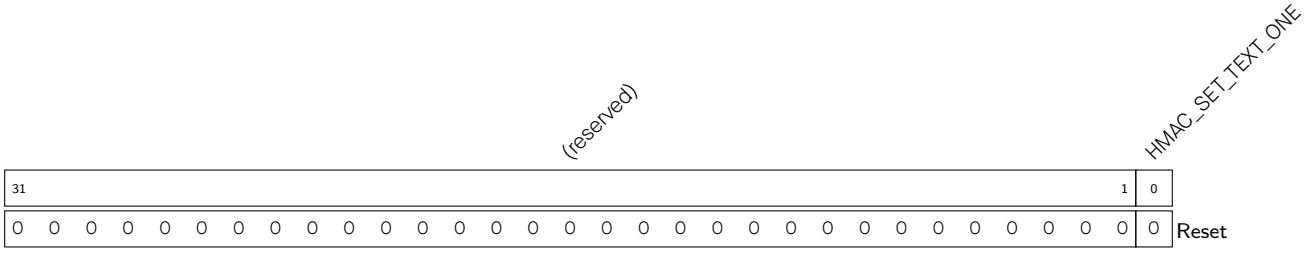
HMAC_SET_PARA_END Configures whether to finish HMAC configuration.

0: No effect

1: Finish configuration

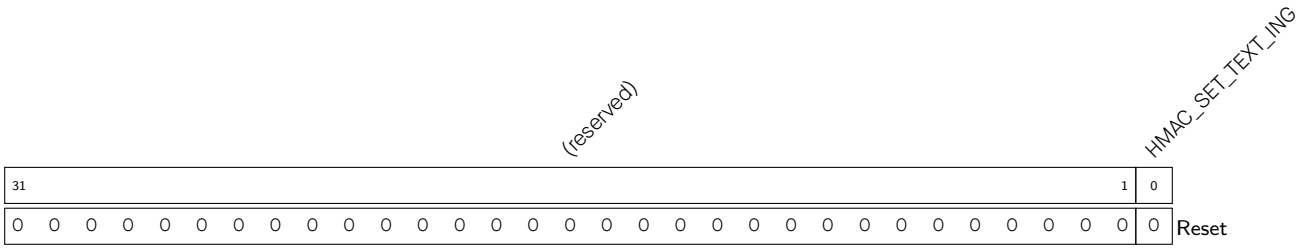
(WS)

Register 12.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



HMAC_SET_TEXT_ONE Calls SHA to calculate one message block. (WS)

Register 12.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



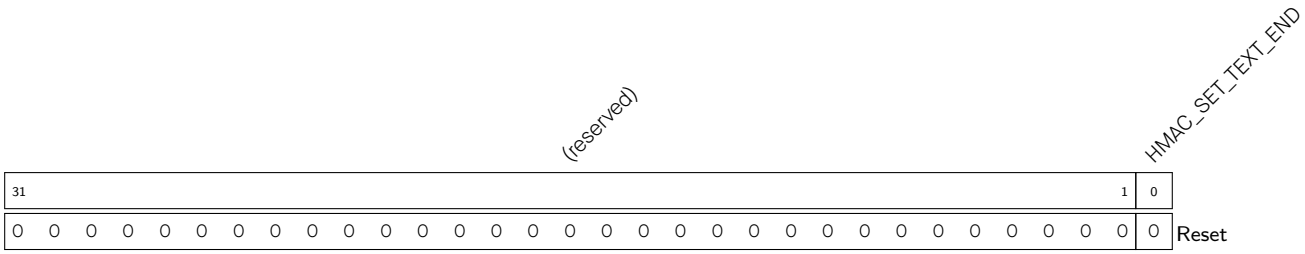
HMAC_SET_TEXT_ING Configures whether or not there are unprocessed message blocks.

0: No unprocessed message block

1: There are still some message blocks to be processed.

(WS)

Register 12.5. HMAC_SET_MESSAGE_END_REG (0x0058)



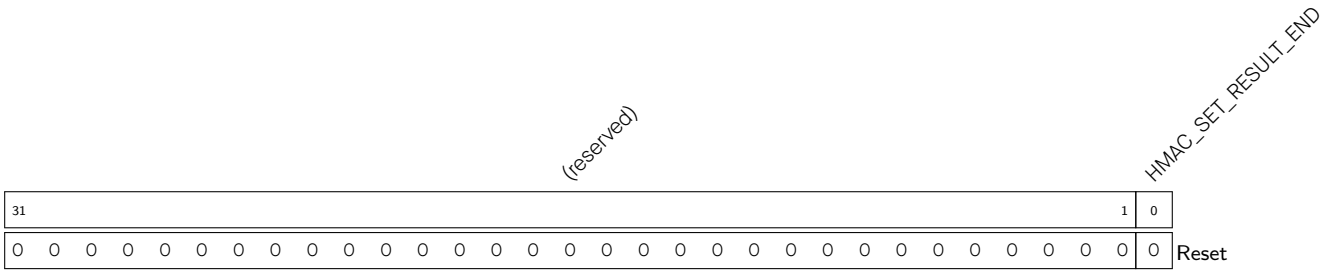
HMAC_SET_TEXT_END Configures whether to start hardware padding.

0: No effect

1: Start hardware padding

(WS)

Register 12.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



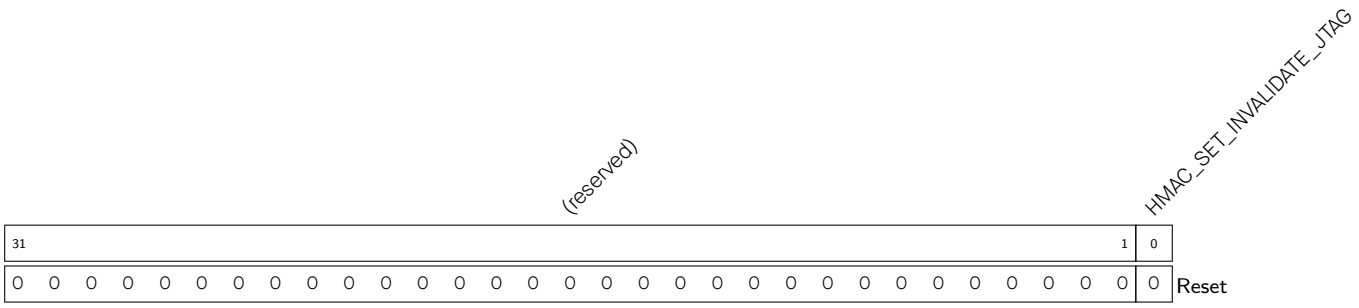
HMAC_SET_RESULT_END Configures whether to exit upstream mode and clear calculation results.

0: Not exit

1: Exit upstream mode and clear calculation results.

(WS)

Register 12.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



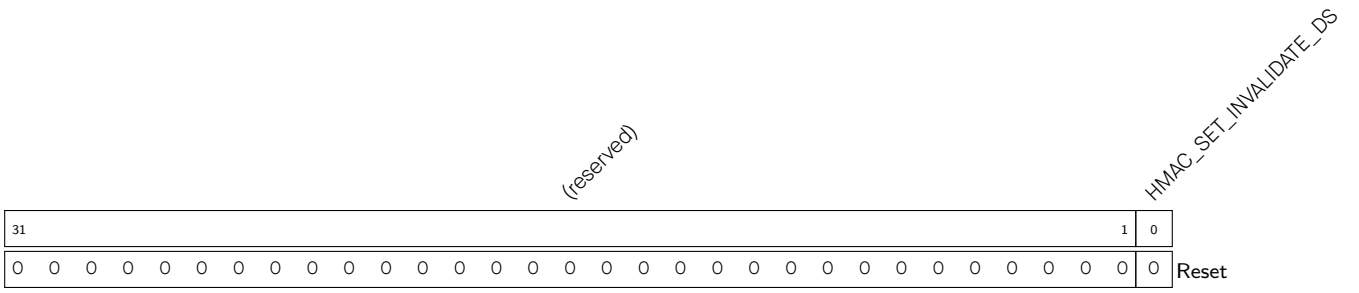
HMAC_SET_INVALIDATE_JTAG Configures whether or not to clear calculation results when re-enabling JTAG in downstream mode.

0: Not clear

1: Clear calculation results

(WS)

Register 12.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



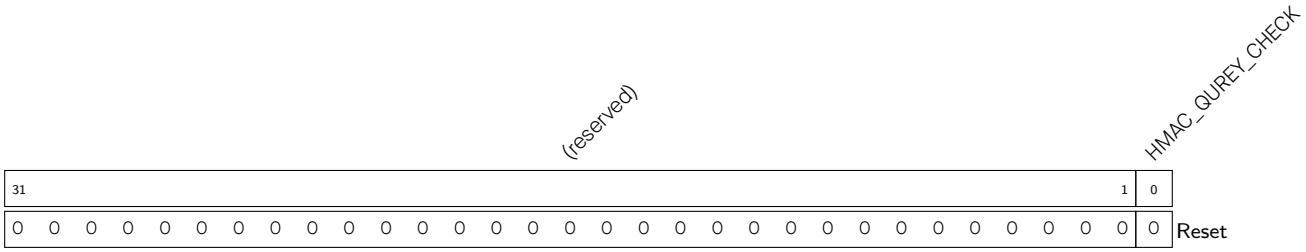
HMAC_SET_INVALIDATE_DS Configures whether or not to clear calculation results of the RSA_DS module in downstream mode.

0: Not clear

1: Clear calculation results

(WS)

Register 12.9. HMAC_QUERY_ERROR_REG (0x0068)



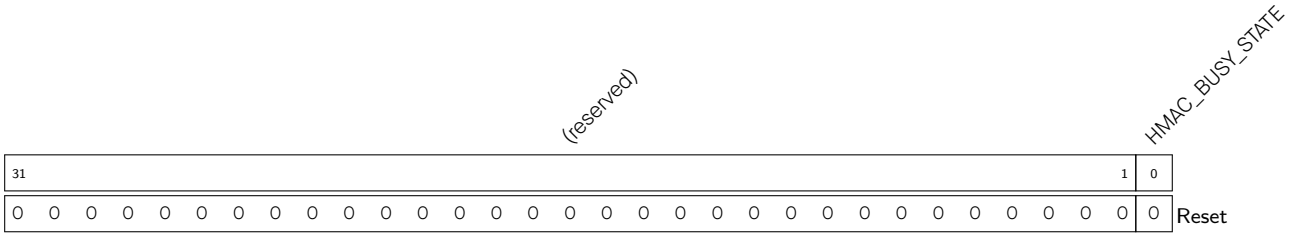
HMAC_QUREY_CHECK Represents whether or not an HMAC key matches the purpose.

0: Match

1: Error

(RO)

Register 12.10. HMAC_QUERY_BUSY_REG (0x006C)



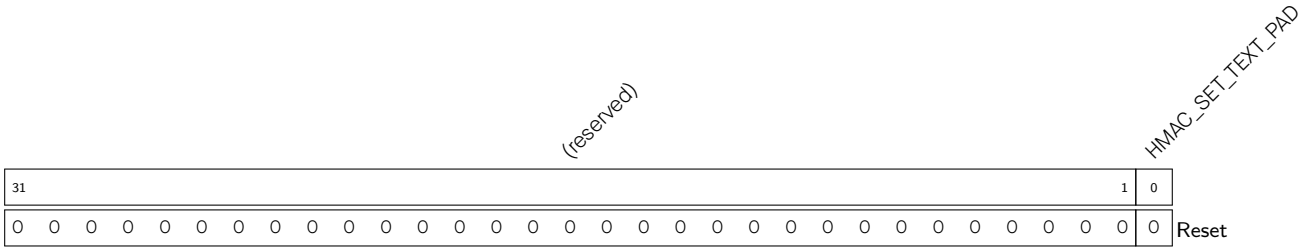
HMAC_BUSY_STATE Represents whether or not HMAC is in a busy state. Before configuring HMAC, please make sure HMAC is in an IDLE state.

0: Idle

1: HMAC is still working on the calculation

(RO)

Register 12.11. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



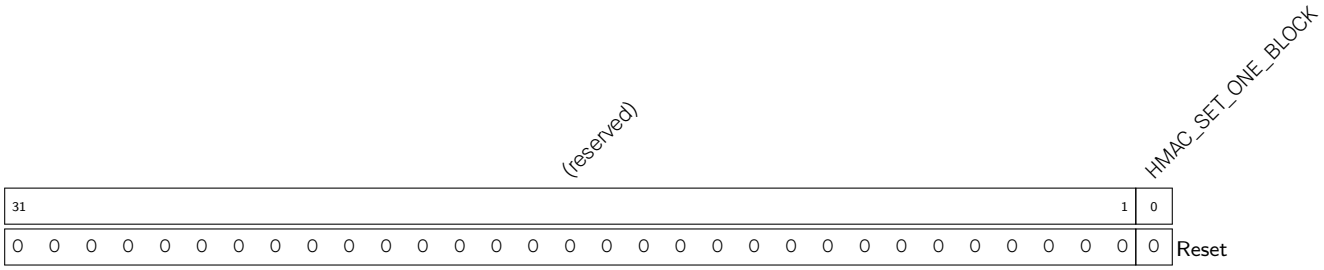
HMAC_SET_TEXT_PAD Configures whether or not the padding is applied by software.

0: Not applied by software

1: Applied by software

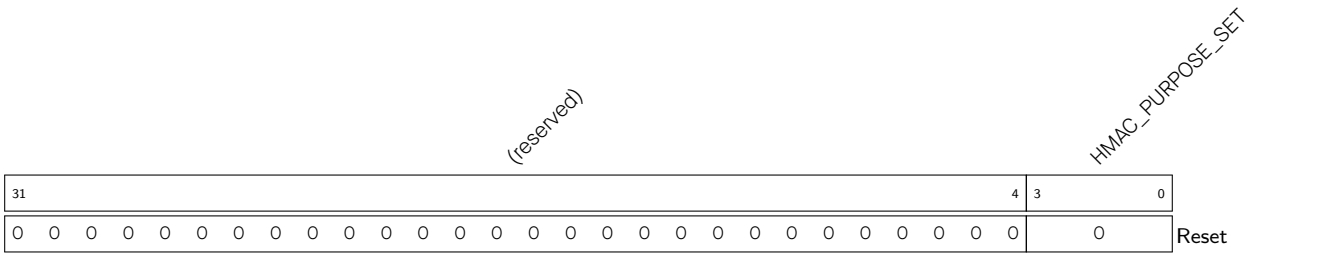
(WO)

Register 12.12. HMAC_ONE_BLOCK_REG (0x00F4)



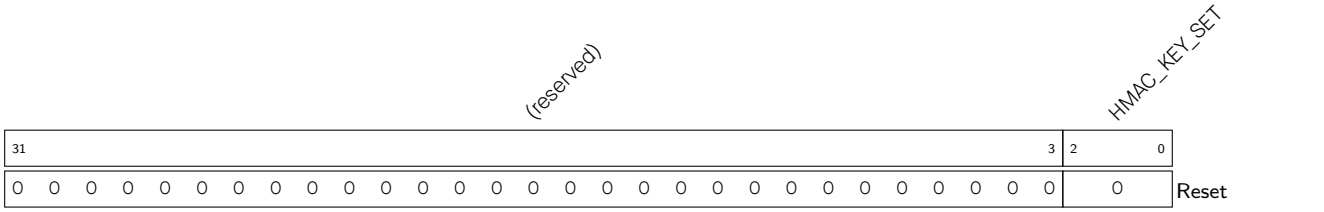
HMAC_SET_ONE_BLOCK Write 1 to indicate there is only one block which already contains padding bits and there is no need for padding. (WS)

Register 12.13. HMAC_SET_PARA_PURPOSE_REG (0x0044)



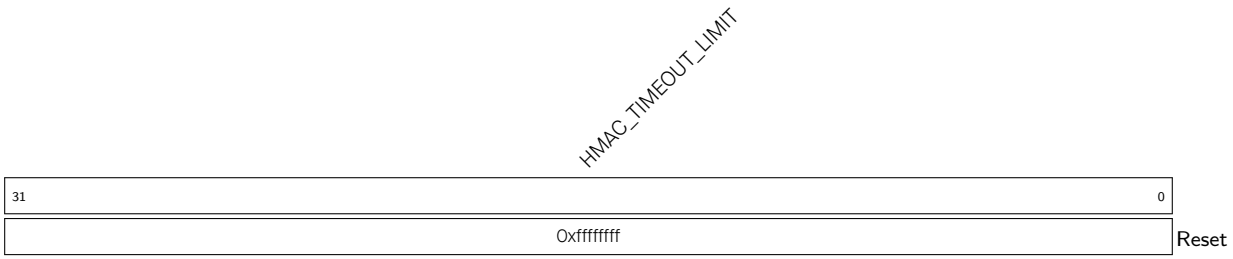
HMAC_PURPOSE_SET Configures the HMAC purpose, refer to the Table 12.4-1. (WO)

Register 12.14. HMAC_SET_PARA_KEY_REG (0x0048)



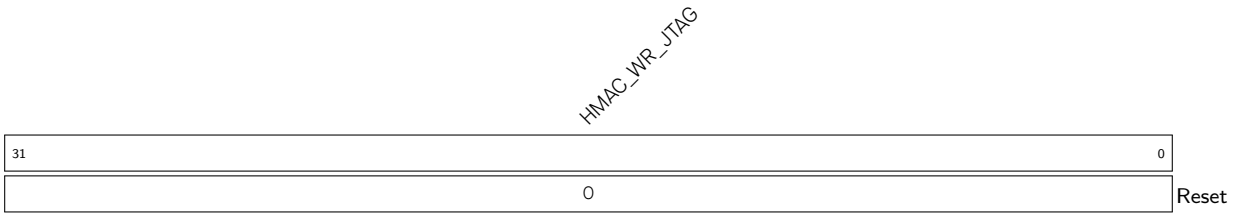
HMAC_KEY_SET Configures HMAC key. There are 5 eFuse keys with index 0 ~ 4 and one key from the key manager with index 7. Write the index of the selected key to this field. (WO)

Register 12.15. HMAC_TIMEOUT_LIMIT_REG (0x0070)



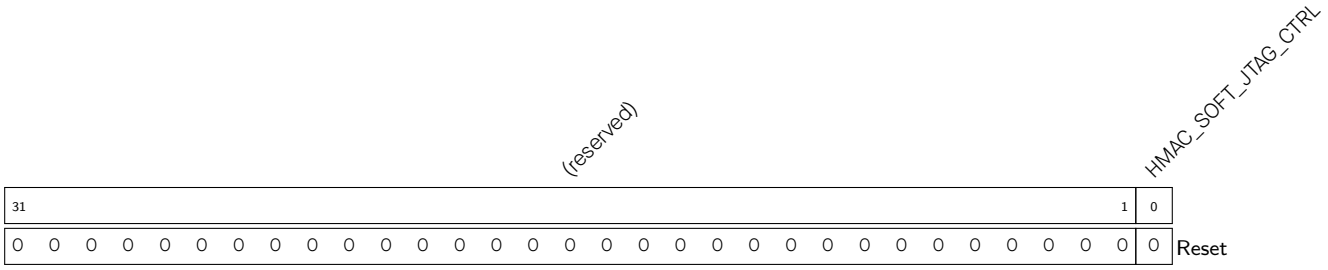
HMAC_TIMEOUT_LIMIT Configures the timeout limitation when waiting for SHA. (R/W)

Register 12.16. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG Writes the comparing input used for re-enabling JTAG. (WO)

Register 12.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



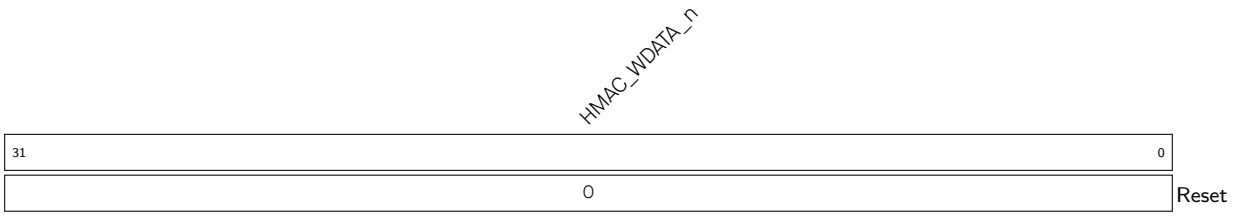
HMAC_SOFT_JTAG_CTRL Configures whether to enable JTAG authentication mode.

0: Disable

1: Enable

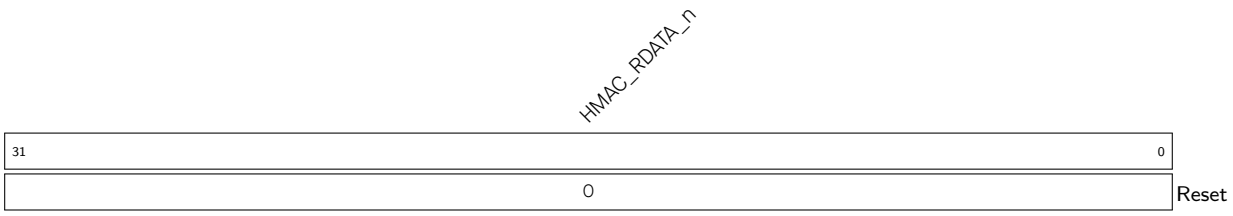
(WS)

Register 12.18. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4×*n*)



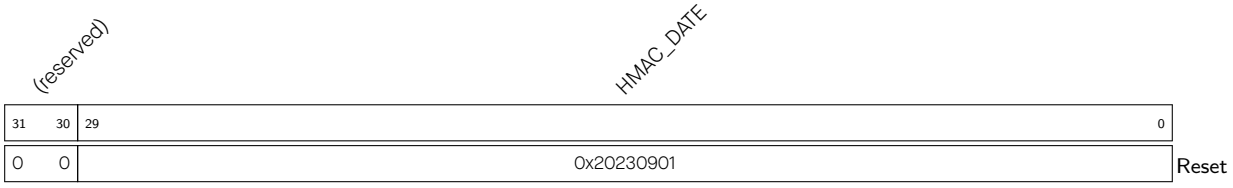
HMAC_WDATA_*n* Represents the *n*th 32-bit of message. (WO)

Register 12.19. 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 12.20. HMAC_DATE_REG (0x01FC)



HMAC_DATE HMAC date information/ HMAC version information. (R/W)

Chapter 13

RSA Accelerator (RSA)

13.1 Introduction

The RSA accelerator provides hardware support for high-precision computation used in various RSA asymmetric cipher algorithms, significantly improving their run time and reducing their software complexity compared with RSA algorithms implemented solely in software. The RSA accelerator also supports operands of different lengths, which provides more flexibility during the computation.

13.2 Features

The following functionality is supported:

- Large-number modular exponentiation (maximum operand length: 4,096 bits) with two additional acceleration options to further increase the calculation speed
- Large-number modular multiplication (maximum operand length: 4,096 bits)
- Large-number multiplication (maximum operand length: 2,048 bits)
- Operands of different lengths
- Interrupt on completion of computation

13.3 Functional Description

The RSA accelerator is activated in the HP_SYS_CLKRST_CRYPTO_CTRLO_REG register by setting the HP_SYS_CLKRST_REG_CRYPTO_RSA_CLK_EN bit to enable the clock and clearing the HP_SYS_CLKRST_REG_CRYPTO_RSA_RST_EN bit to release the RSA accelerator from reset.

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 field 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 disabled.

Notice:

ESP32-S31's RSA Digital Signature Peripheral (RSA_DS) module also calls the RSA accelerator when working. By default, if RSA_FREE_REG is 1, the RSA Digital Signature Peripheral (RSA_DS) can immediately abort any in-progress RSA operation and use the RSA accelerator to complete its operation. Set RSA_FREE_REG to 0 to prevent this.

13.3.1 Definitions and Representations

ESP32-S31's RSA accelerator supports operands of different lengths $N = 32 \times n$.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the values in $Z_{n-1}\cdots Z_0$, $X_{n-1}\cdots X_0$, $Y_{n-1}\cdots Y_0$, $M_{n-1}\cdots M_0$, $\overline{r}_{n-1}\cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional argument $\overline{r}$ can be calculated as $\overline{r} = R^2 \bmod M$.

Also, argument M' can be calculated using the formula below:

$$M' = -M^{-1} \bmod b$$

where, M^{-1} is the modular multiplicative inverse of M, and it can be calculated with the extended binary GCD algorithm.

13.3.2 Large-Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

Large-number modular exponentiation on the ESP32-S31 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 13.3.5.

3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.

4. Write 1 to the RSA_SET_START_MODEXP field of the RSA_SET_START_MODEXP_REG register to start computation.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

13.3.3 Large-Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. Therefore, similar to the large-number modular exponentiation, two additional arguments are required – $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports large-number modular multiplication with operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA field to enable or disable the interrupt function.
2. Configure relevant configuration registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM, and RSA_Z_MEM, respectively. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.

4. Write 1 to the RSA_SET_START_MODMULT field.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.

7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

13.3.4 Large-Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA accelerator only supports large-number multiplication with operand length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA field to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e., $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n-1\}$ to the address of the i words of the RSA_X_MEM memory block. Note that Y_i for $i \in \{0, 1, \ldots, n-1\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e., the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.
4. Write 1 to the RSA_SET_START_MULT register.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, \hat{n}-1\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

13.3.5 Options for Additional Acceleration

The ESP32-S31 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 both of these options to further accelerate the computation. Note that, even when neither of these two options is configured, using the hardware RSA accelerator is still much faster than implementing the RSA algorithm in software.

To be more specific, when neither of these two options are configured for additional acceleration, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. When either or both of these two options are configured for additional acceleration, the time required is also correlated with the 0 and 1 bit distribution in Y.

To better illustrate how these two options work, first assume Y is represented in binary as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e., the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2},\cdots,\widetilde{Y}_0$ is $t-m$.

When either of these two options is configured for additional acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for additional acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. Set α to a number smaller than N-1, which otherwise leads to the same result as if this option is not used for additional acceleration. The best acceleration performance can be achieved by setting α to t, in which case all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0 s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
  - Note that this option compromises the security because it ignores some bits, which essentially shortens the key length, thus should not be enabled for applications with high security requirement.
- CONSTANT_TIME Option (Configuring RSA_CONSTANT_TIME_REG to 0 for additional acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better is the acceleration performance.
  - Note that this option also compromises the security because its time cost correlates with the 0/1 distribution of the key, which can be used in a Side Channel Attack (SCA), thus should not be enabled for applications with high security requirement.

Below is an example to demonstrate the performance of the RSA accelerator under different combinations of SEARCH and CONSTANT_TIME configuration. In this example:

- We perform $Z = X^Y \bmod M$
- N = 3072
- Y = 65537
- X and M are taken at random

- When the SEARCH option is enabled, α, i.e., the position register RSA_SEARCH_POS_REG, is set to 16.

Table 13.3-1 below demonstrates the time cost in clock cycles under different combinations of SEARCH and CONSTANT_TIME configuration when performing $Z = X^Y \bmod M$ as described above.

Table 13.3-1. Acceleration Performance

SEARCH Option	CONSTANT_TIME Option	Time Cost (clock cycle)
No acceleration	No acceleration	174.7×10^6
Acceleration	No acceleration	1.023×10^6
No acceleration	Acceleration	0.546×10^6
Acceleration	Acceleration	0.540×10^6

As shown in Table 13.3-1:

- The time cost is biggest when none of these two options is configured for additional acceleration.
- The time cost is smallest when both of these two options are configured for additional acceleration.
- The time cost can be dramatically reduced when either or both option(s) are configured for additional acceleration.

13.4 Interrupts

ESP32-S31's RSA accelerator can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- RSA_INTR

There is one internal interrupt source from the RSA accelerator that can generate the above interrupt signal. The interrupt source from the RSA accelerator is listed with its trigger condition and the resulting interrupt signal in Table 13.4-1.

Table 13.4-1. RSA's Internal Interrupt Source

Internal Interrupt Source	Trigger Condition	Interrupt Signal
RSA_CALC_DONE_INT	Completion of an RSA calculation	RSA_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

13.5 Memory Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Table 13.5-1. RSA Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	512	0x0000	0x01FF	R/W
RSA_Z_MEM	Memory Z	512	0x0200	0x03FF	R/W
RSA_Y_MEM	Memory Y	512	0x0400	0x05FF	R/W
RSA_X_MEM	Memory X	512	0x0600	0x07FF	R/W

13.6 Register Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

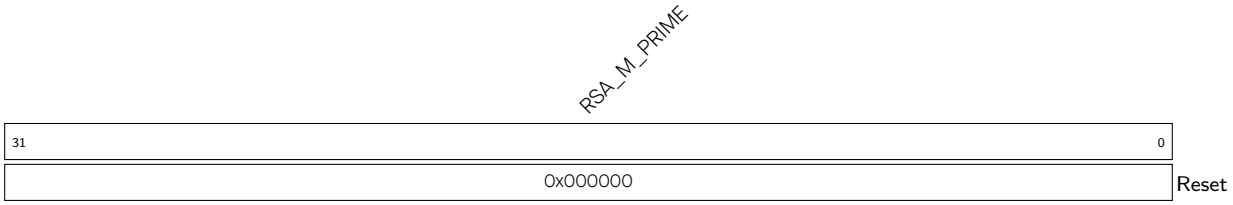
Name	Description	Address	Access
Control or Configuration Registers			
RSA_M_PRIME_REG	Represents M'	0x0800	R/W
RSA_MODE_REG	Configures RSA length	0x0804	R/W
RSA_SET_START_MODEXP_REG	Starts modular exponentiation	0x080C	WT
RSA_SET_START_MODMULT_REG	Starts modular multiplication	0x0810	WT
RSA_SET_START_MULT_REG	Starts multiplication	0x0814	WT
RSA_QUERY_IDLE_REG	Represents the RSA status	0x0818	RO
RSA_CONSTANT_TIME_REG	Configures the constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	Configures the search option	0x0824	R/W
RSA_SEARCH_POS_REG	Configures the search position	0x0828	R/W
RSA_CLK_FORCE_ON_REG	Configures the RSA memory clock force-on option	0x0834	R/W
RSA_FREE_REG	Configures the RSA release option	0x0838	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

13.7 Registers

The addresses in this section are relative to the RSA accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

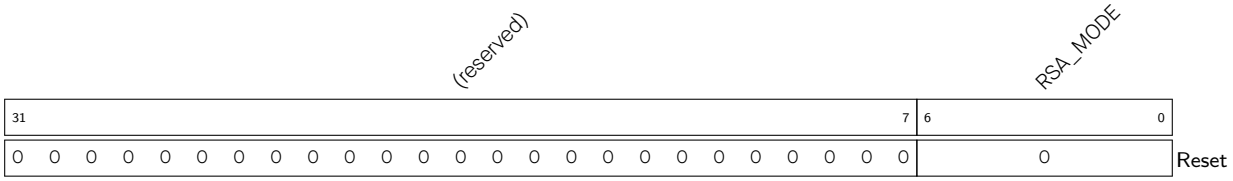
For how to program reserved fields, please refer to Section VII .

Register 13.1. RSA_M_PRIME_REG (0x0800)



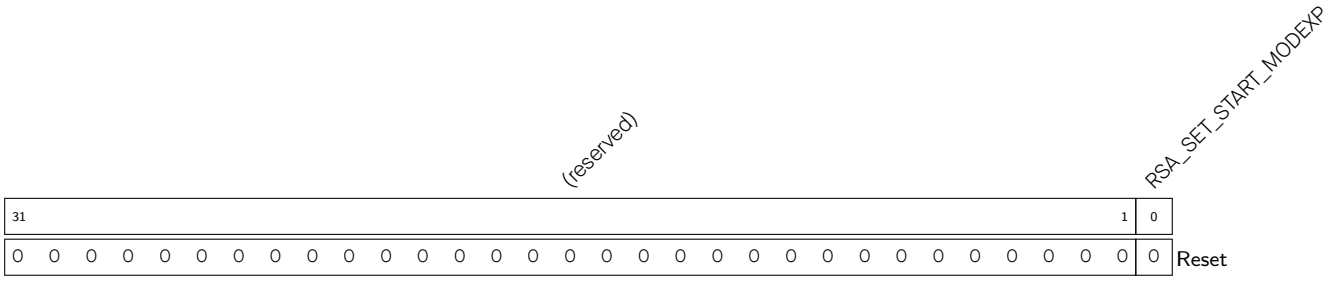
RSA_M_PRIME Represents M'. (R/W)

Register 13.2. RSA_MODE_REG (0x0804)



RSA_MODE Configures the RSA length. (R/W)

Register 13.3. RSA_SET_START_MODEXP_REG (0x080C)



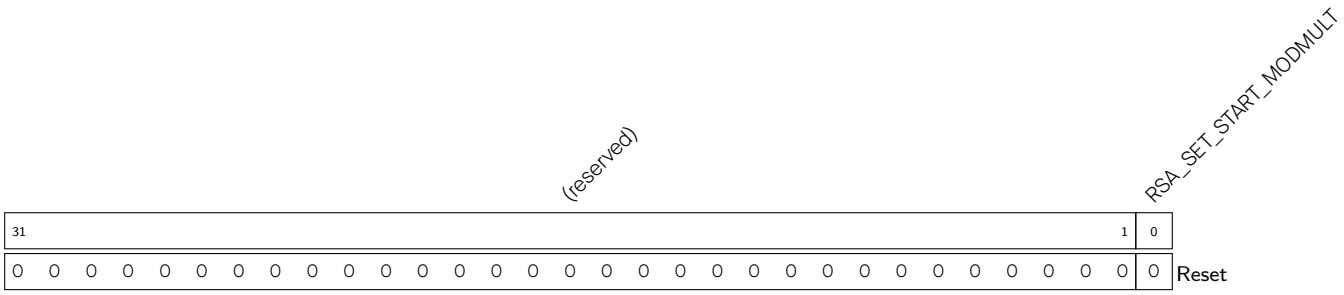
RSA_SET_START_MODEXP Configures whether or not to starts the modular exponentiation.

0: No effect

1: Start

(WT)

Register 13.4. RSA_SET_START_MODMULT_REG (0x0810)



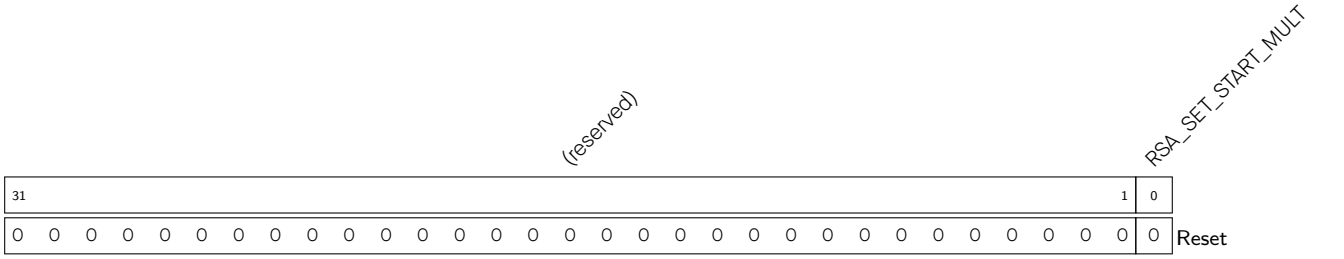
RSA_SET_START_MODMULT Configures whether or not to start the modular multiplication.

0: No effect

1: Start

(WT)

Register 13.5. RSA_SET_START_MULT_REG (0x0814)



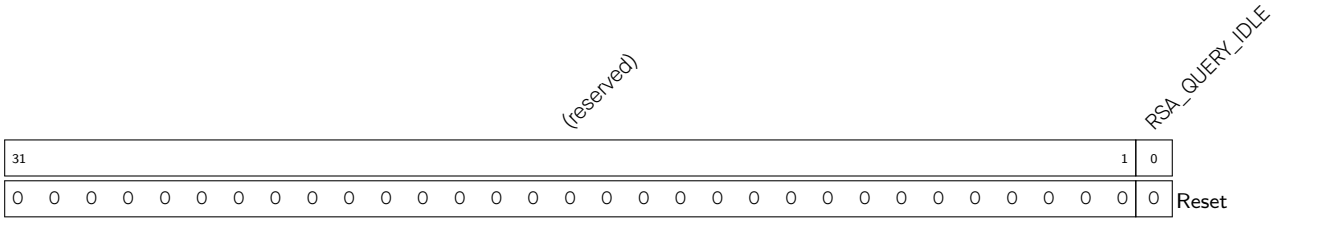
RSA_SET_START_MULT Configures whether or not to start the multiplication.

0: No effect

1: Start

(WT)

Register 13.6. RSA_QUERY_IDLE_REG (0x0818)



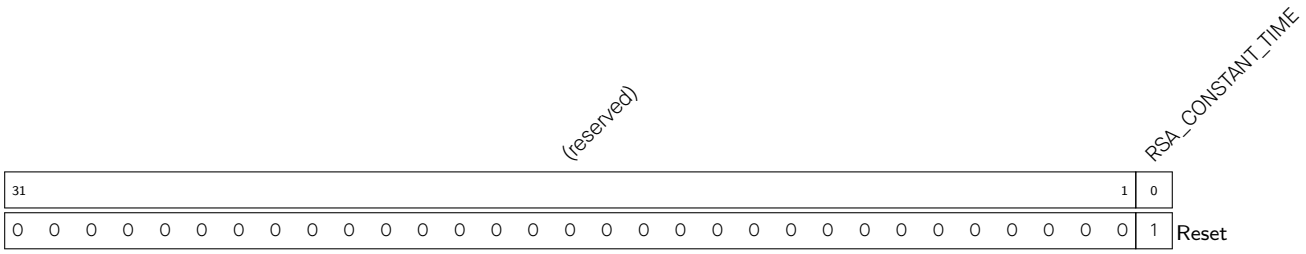
RSA_QUERY_IDLE Represents the RSA status.

0: Busy

1: Idle

(RO)

Register 13.7. RSA_CONSTANT_TIME_REG (0x0820)



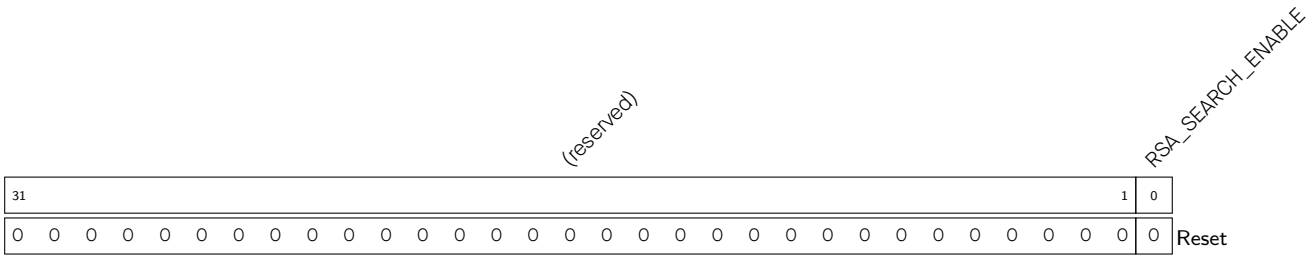
RSA_CONSTANT_TIME Configures the constant_time option.

0: Acceleration

1: No acceleration (default)

(R/W)

Register 13.8. RSA_SEARCH_ENABLE_REG (0x0824)



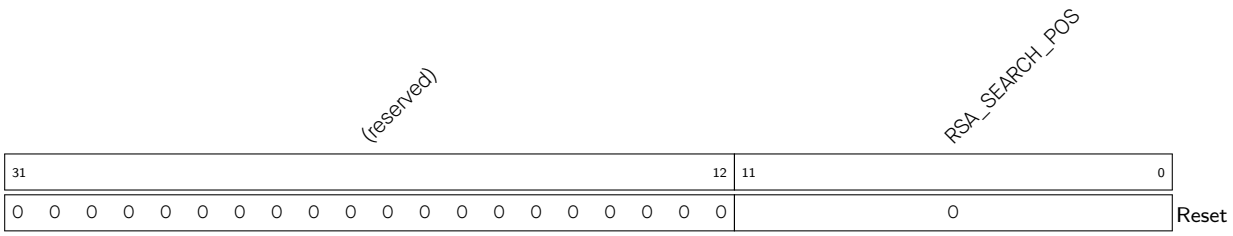
RSA_SEARCH_ENABLE Configures the search option.

0: No acceleration (default)

1: Acceleration

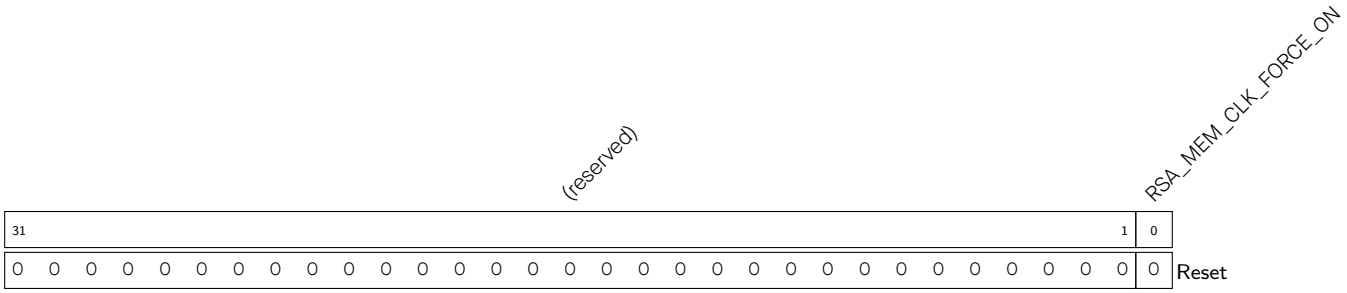
This option should be used together with RSA_SEARCH_POS_REG. (R/W)

Register 13.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS Configures the starting bit position 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 13.10. RSA_CLK_FORCE_ON_REG (0x0834)



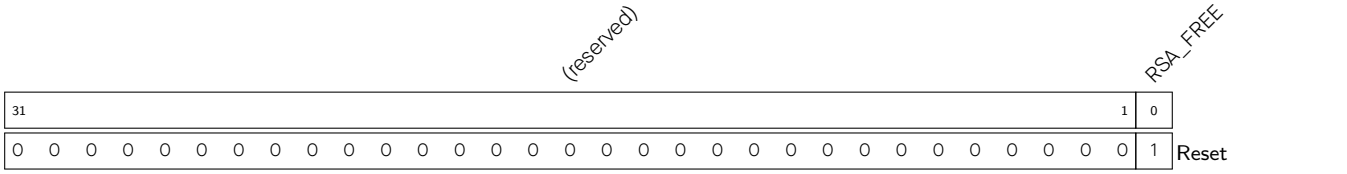
RSA_MEM_CLK_FORCE_ON Configures whether RSA memory clock is always forced on.

0: RSA mem is not force on

1: RSA mem is force on

(R/W)

Register 13.11. RSA_FREE_REG (0x0838)



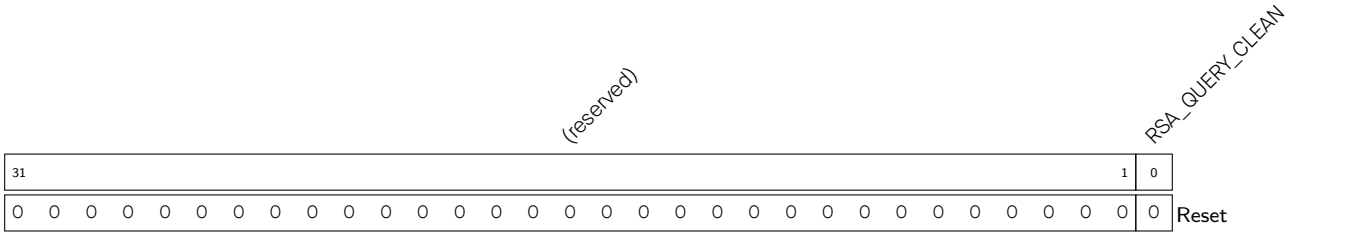
RSA_FREE Configures whether the RSA accelerator can be released to another module while it is operating.

0: Cannot be released

1: Can be released (default)

(R/W)

Register 13.12. RSA_QUERY_CLEAN_REG (0x0808)



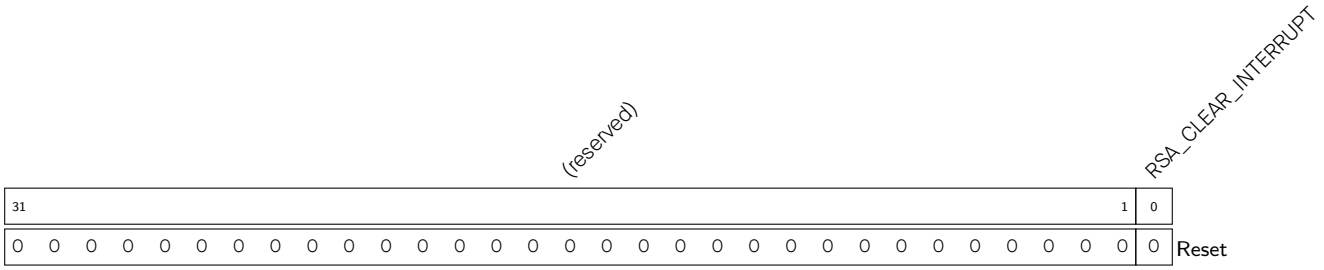
RSA_QUERY_CLEAN Represents whether or not the RSA memory completes initialization.

0: Not complete

1: Completed

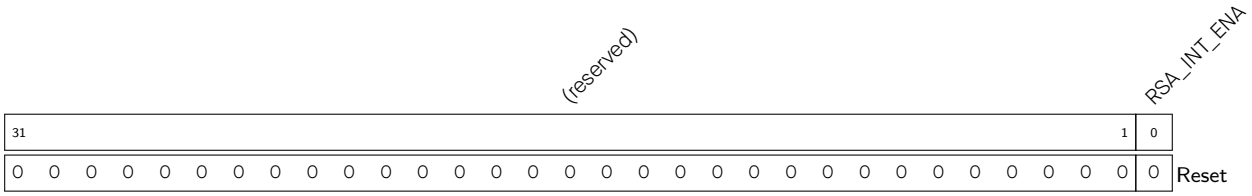
(RO)

Register 13.13. RSA_INT_CLR_REG (0x081C)



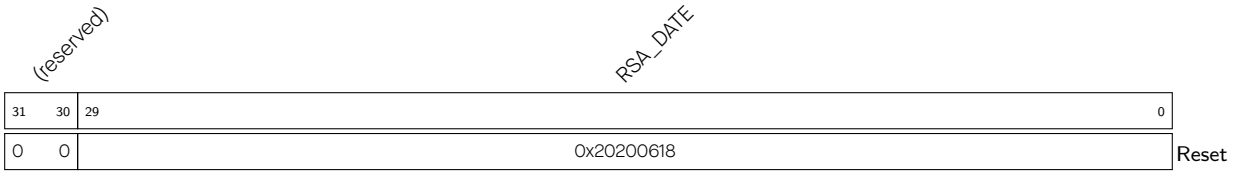
RSA_CLEAR_INTERRUPT Write 1 to clear the RSA interrupt. (WT)

Register 13.14. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA Write 1 to enable the RSA interrupt. (R/W)

Register 13.15. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

Chapter 14

SHA Accelerator (SHA)

14.1 Overview

ESP32-S31 integrates an SHA accelerator, which is a hardware device that speeds up the SHA algorithm significantly, compared to an SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-S31 has two working modes, which are Typical SHA and DMA-SHA.

14.2 Features

The SHA accelerator has the following features:

- The following hash algorithms introduced in FIPS PUB 180-4 Spec
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/256
  - SHA-512/*t*
- The following hash algorithms introduced in GM/T 0004-2012
  - SM3
- 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

14.3 Working Modes

The SHA accelerator integrated in ESP32-S31 has two working modes.

- Typical SHA Working Mode: all the data is written and read via CPU directly.

- DMA-SHA Working Mode: all the data is read via DMA. That is, users can configure the GDMA-AXI controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

The SHA accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN register field and clearing the HP_SYS_CLKRST_SHA_RST_EN register field.

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 14.3-1.

Table 14.3-1. 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 14.3-2. Support for the Chinese cryptographic standard (SM3) can be disabled by EFUSE_DIS_SM_CRYPT.

Table 14.3-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/*t*	7
SM3	14

Notice:

ESP32-S31's RSA Digital Signature Peripheral (RSA_DS) and HMAC Accelerator (HMAC) modules also call the SHA accelerator when working. By default, SHA_FREE (with value 1) would allow RSA Digital Signature Peripheral (RSA_DS) and HMAC Accelerator (HMAC) modules to immediately reset any current SHA operation and use SHA for its operations. Users should set SHA_FREE to 0 in order to prevent this from happening.

14.4 Function Description

The SHA accelerator generates the message digest via two steps: Preprocessing and Hash operation.

14.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

14.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 or 1024 bits, depending on the algorithm. Thus, all the messages should be padded to a multiple of 512 bits or 1024 bits before the hash operation.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

- **SHA-1**, **SHA-224**, **SHA-256**, and **SM-3**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k bits of zeros, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
  3. Last, append the 64-bit block of value equal to the number m expressed using a binary representation.
- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896 \ mod \ 1024$;
  3. Last, append the 128-bit block of value equal to the number m expressed using a binary representation.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

14.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit or 1024-bit blocks. During the task, all the message blocks should be written into the SHA_M_*n*_REG.

- For **SHA-1**, **SHA-224**, **SHA-256**, and **SM-3**:

  The message and its padding are parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, $M_0^{(i)}$ should be stored in SHA_M_0_REG, $M_1^{(i)}$ stored in SHA_M_1_REG, ..., and $M_{15}^{(i)}$ stored in SHA_M_15_REG.

- For **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**:
  The message and its padding are parsed into N 1024-bit blocks. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, the most significant 32 bits and the least significant 32 bits of $M_0^{(i)}$ should be stored in SHA_M_0_REG and SHA_M_1_REG, respectively, ..., the most significant 32 bits and the least significant 32 bits of $M_{15}^{(i)}$ should be stored in SHA_M_30_REG and SHA_M_31_REG, respectively.

Note:

For more information about "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

14.4.1.3 Setting the Initial Hash Value

Before hash task begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, and SM3 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t\_length = 7'h48$ and t_string is padded in the following format:

$8'h30 + 8'ht_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and t_string = $32'h38800000$.

   - If $10 <= t <= 99$, then $t\_length = 7'h50$ and t_string is padded in the following format:

$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and t_string = $32'h35368000$.

   - If $100 <= t < 512$, then $t\_length = 7'h58$ and t_string is padded in the following format:

$8'h30 + 8'ht_2$	$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and t_string = $32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, which is performing SHA-512 operation (with its initial hash value set to the result of 8-bitwise XOR operation of C and 0xa5) from the hexadecimal representation of the string "SHA-512/t".

14.4.2 Hash Operation

After the preprocessing, the ESP32-S31 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-S31 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.

14.4.2.1 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-S31 SHA also supports optional "interleaved" message digest calculation in Typical SHA mode, which means before SHA completes all blocks of the current message, users are given a chance to insert new computation of another message digest upon the completion of each individual block of the current message.

Specifically, users can read out the message digest from registers SHA_H_*n*_REG after completing part of a message digest calculation, and use the SHA accelerator for a different calculation. After the different calculation completes, users can restore the previous message digest to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused calculation.

Typical SHA Process (except for SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 14.3-2.
2. Process the current message block.
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator[1].
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;
   - If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/t)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register to 7 for SHA-512/t.
2. Calculate the initial hash value.
   (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 14.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
3. Process the current message block[1].
   - Write the message block in registers SHA_M_*n*_REG.
4. Start the SHA accelerator
   - Set the SHA_CONTINUE_REG register to 1. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
5. Check the progress of the calculation.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status[3].
6. Decide if you have more message blocks to process:
   - If yes, please go back to Step 3.
   - Otherwise, please continue.
7. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Note:

1. In this step, the software can also write the next message block, if any, to the SHA_M_*n*_REG registers, while the hardware is performing the SHA calculation, to reduce processing time.
2. 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-S31 SHA accelerator supports **"interleaving" calculation under the Typical SHA working mode**. The process for 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.

14.4.2.2 DMA-SHA Mode Process

ESP32-S31 SHA accelerator does not support "interleaving" message digest calculation at the level of individual message blocks when using DMA, which means you cannot insert new calculation before a complete DMA-SHA process (of one or more message blocks) completes. In this case, users who need interleaved operation are recommended to divide the message blocks and perform several DMA-SHA calculations, instead of trying to compute all the messages in one go.

Single DMA-SHA calculation supports up to 63 message blocks.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via GDMA-AXI. Therefore, users are required to configure the GDMA-AXI controller following the description in Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

DMA-SHA process (except SHA-512/t)

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 14.3-2.
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, first write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.

6. Obtain the message digest:
    - Read the message digest from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/*t*

1. Select a hash algorithm.
    - Select SHA-512/*t* algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
    (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 14.4.1.3.
    (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
    (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
4. Configure the number of message blocks.
    - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
5. Start the DMA-SHA calculation.
    - Write 1 to register SHA_DMA_CONTINUE_REG to start the accelerator.
6. Wait till the completion of the DMA-SHA calculation, which happens when:
    - The content of SHA_BUSY_REG register becomes 0, or
    - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
7. Obtain the message digest:
    - Read the message digest from registers SHA_H_*n*_REG.

14.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG(*n*: 0 ~ 15). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 14.4-4 below:

Table 14.4-4. The Storage and Length of Message Digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t*[2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG
SM3	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 14.4.1.2.

[2] The registers used for SHA-512/t algorithm depend on the value of t. x+1 indicates the number of 32-bit registers used to store t bits of message digest, so that $x = roundup(t/32)$-1. For example:

- When t = 8, then x = 0, indicating that the 8-bit long message digest is stored in the most significant 8 bits of register SHA_H_0_REG;
- When t = 32, then x = 0, indicating that the 32-bit long message digest is stored in register SHA_H_0_REG;
- When t = 132, then x = 4, indicating that the 132-bit long message digest is stored in registers SHA_H_0_REG, SHA_H_1_REG, SHA_H_2_REG, SHA_H_3_REG, and SHA_H_4_REG.

14.5 Interrupt

ESP32-S31's SHA accelerator can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- SHA_INTR

There are several internal interrupt sources from SHA that can generate the above interrupt signal(s). The interrupt sources from SHA are listed with their trigger conditions and the resulted interrupt signal(s) in Table 14.5-1.

Table 14.5-1. SHA's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SHA_CALC_DONE_INT	Completion of an message digest calculation in DMA-SHA mode	SHA_INTR

Note:

- For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.
- Different from the standard interrupt register group, the interrupt register group of SHA only contains the INT_ENA (SHA_INT_ENA_REG) and INT_CLR (SHA_INT_CLEAR_REG) fields, and does not support users to read the INT_RAW and INT_ST fields.

14.6 Register Summary

The addresses in this section are relative to SHA accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Control/Configuration Registers			
SHA_MODE_REG	Configures SHA algorithm	0x0000	R/W
SHA_T_STRING_REG	Initial Hash Value calculation factor t_string (for SHA-512/t)	0x0004	R/W
SHA_T_LENGTH_REG	Initial Hash Value calculation factor t_length (for SHA-512/t)	0x0008	R/W
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
SHA_DMA_CONTINUE_REG	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	Block number register (only effective in DMA-SHA mode)	0x000C	R/W
SHA_DMA_RX_RESET_REG	DMA RX Reset Signal	0x0030	WO
SHA_DMA_TX_RESET_REG	DMA TX Reset Signal	0x0034	WO
Status Registers			
SHA_BUSY_REG	Represents if SHA Accelerator is busy or not	0x0018	RO
SHA_FREE_REG	SHA free register	0x0038	R/W
Interrupt Registers			
SHA_INT_CLEAR_REG	DMA-SHA interrupt clear register	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA interrupt enable register	0x0028	R/W
Data Registers			
SHA_H_0_REG	Hash value	0x0040	R/W
SHA_H_1_REG	Hash value	0x0044	R/W
SHA_H_2_REG	Hash value	0x0048	R/W
SHA_H_3_REG	Hash value	0x004C	R/W
SHA_H_4_REG	Hash value	0x0050	R/W
SHA_H_5_REG	Hash value	0x0054	R/W
SHA_H_6_REG	Hash value	0x0058	R/W
SHA_H_7_REG	Hash value	0x005C	R/W
SHA_H_8_REG	Hash value	0x0060	R/W
SHA_H_9_REG	Hash value	0x0064	R/W
SHA_H_10_REG	Hash value	0x0068	R/W
SHA_H_11_REG	Hash value	0x006C	R/W
SHA_H_12_REG	Hash value	0x0070	R/W
SHA_H_13_REG	Hash value	0x0074	R/W
SHA_H_14_REG	Hash value	0x0078	R/W
SHA_H_15_REG	Hash value	0x007C	R/W
SHA_M_0_REG	Message	0x0080	R/W
SHA_M_1_REG	Message	0x0084	R/W
SHA_M_2_REG	Message	0x0088	R/W

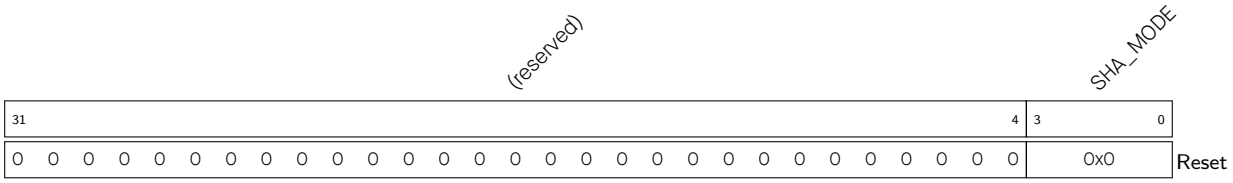
Name	Description	Address	Access
SHA_M_3_REG	Message	0x008C	R/W
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	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
SHA_M_16_REG	Message	0x00C0	R/W
SHA_M_17_REG	Message	0x00C4	R/W
SHA_M_18_REG	Message	0x00C8	R/W
SHA_M_19_REG	Message	0x00CC	R/W
SHA_M_20_REG	Message	0x00D0	R/W
SHA_M_21_REG	Message	0x00D4	R/W
SHA_M_22_REG	Message	0x00D8	R/W
SHA_M_23_REG	Message	0x00DC	R/W
SHA_M_24_REG	Message	0x00E0	R/W
SHA_M_25_REG	Message	0x00E4	R/W
SHA_M_26_REG	Message	0x00E8	R/W
SHA_M_27_REG	Message	0x00EC	R/W
SHA_M_28_REG	Message	0x00F0	R/W
SHA_M_29_REG	Message	0x00F4	R/W
SHA_M_30_REG	Message	0x00F8	R/W
SHA_M_31_REG	Message	0x00FC	R/W
Version Register			
SHA_DATE_REG	Version control register	0x002C	R/W

14.7 Registers

The addresses in this section are relative to SHA accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

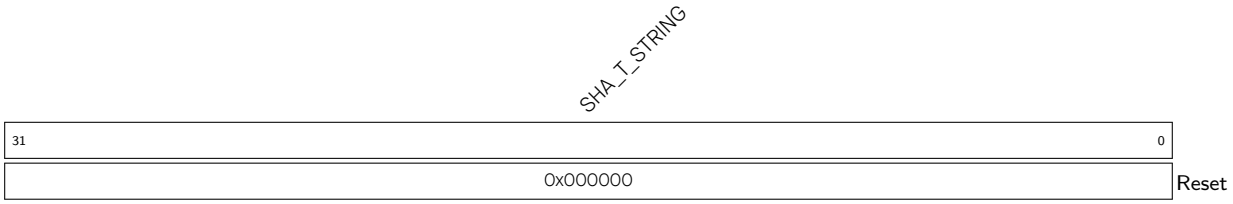
Register 14.1. SHA_MODE_REG (0x0000)



SHA_MODE Configures the SHA algorithm.

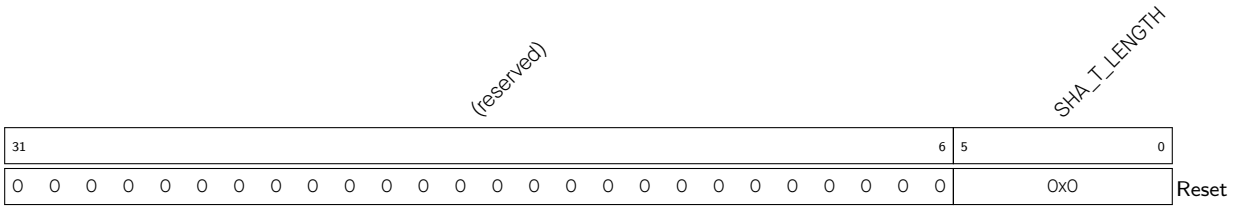
0: SHA-1
1: SHA-224
2: SHA-256
3: SHA-384
4: SHA-512
5: SHA-512/224
6: SHA-512/256
7: SHA-512/t
8-13: Reserved
14: SM3
15: Reserved
(R/W)

Register 14.2. SHA_T_STRING_REG (0x0004)



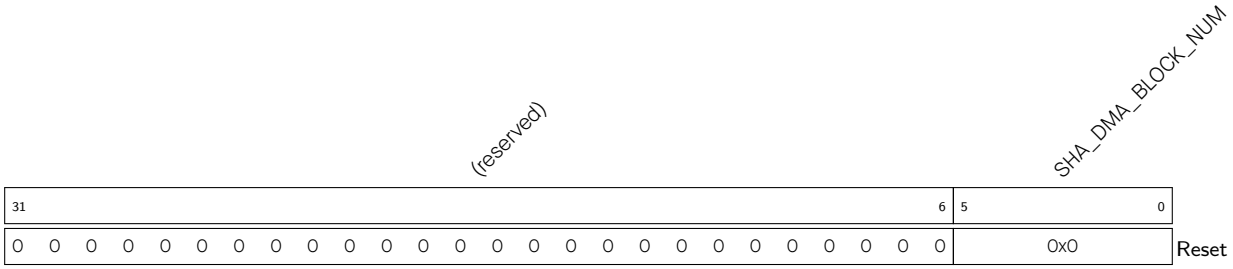
SHA_T_STRING Configures t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 14.3. SHA_T_LENGTH_REG (0x0008)



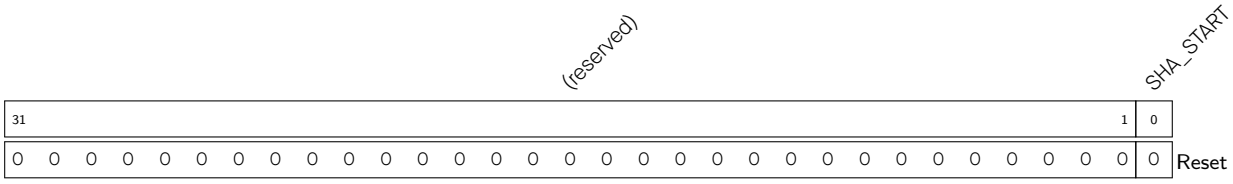
SHA_T_LENGTH Configures t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 14.4. SHA_DMA_BLOCK_NUM_REG (0x000C)



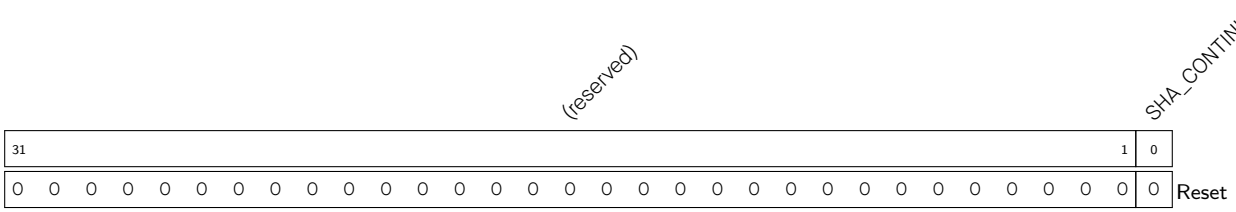
SHA_DMA_BLOCK_NUM Configures the DMA-SHA block number. (R/W)

Register 14.5. SHA_START_REG (0x0010)



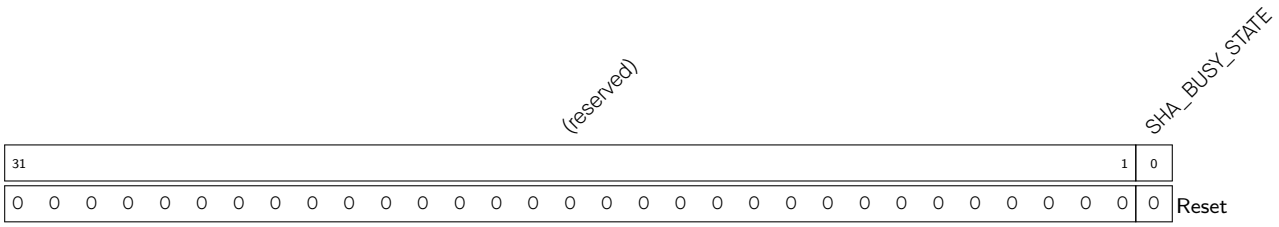
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 14.6. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 14.7. SHA_BUSY_REG (0x0018)



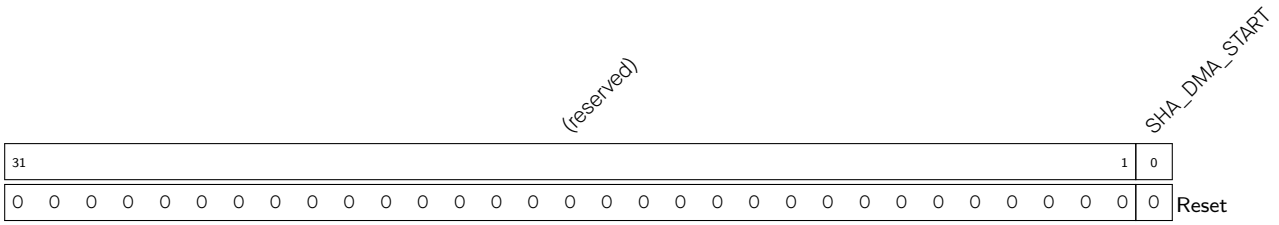
SHA_BUSY_STATE Represents the states of SHA accelerator.

0: idle

1: busy

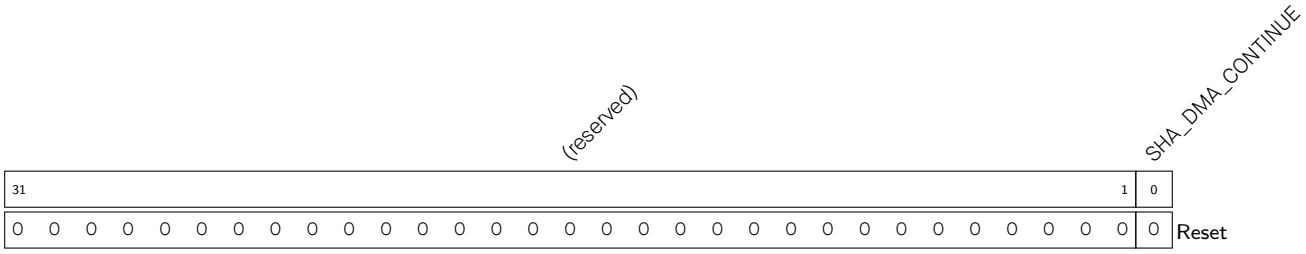
(RO)

Register 14.8. SHA_DMA_START_REG (0x001C)



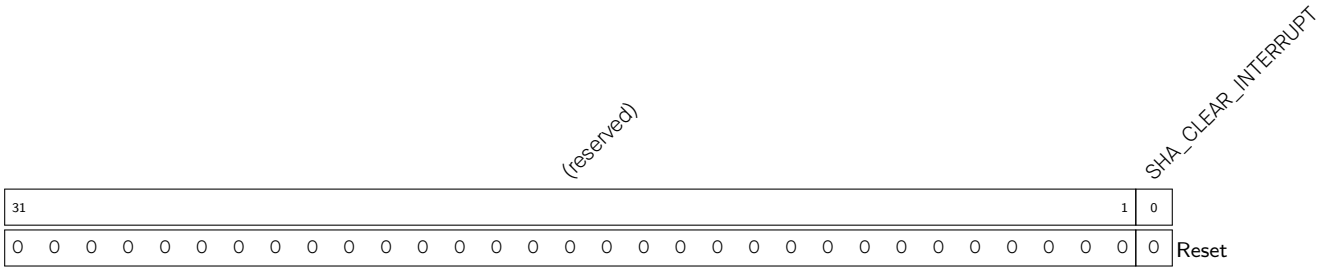
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 14.9. SHA_DMA_CONTINUE_REG (0x0020)



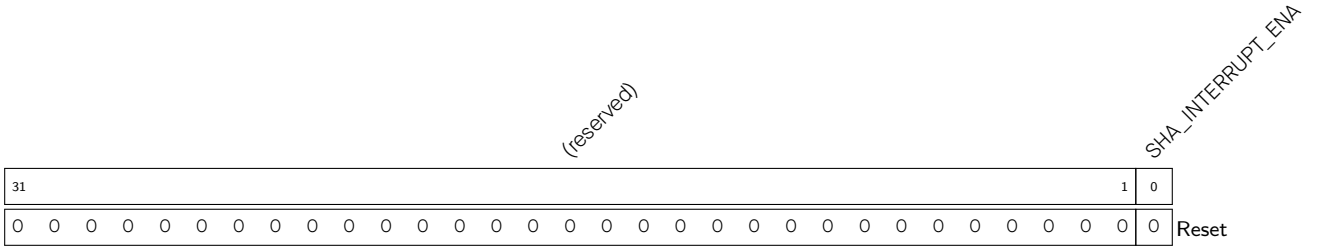
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 14.10. SHA_INT_CLEAR_REG (0x0024)



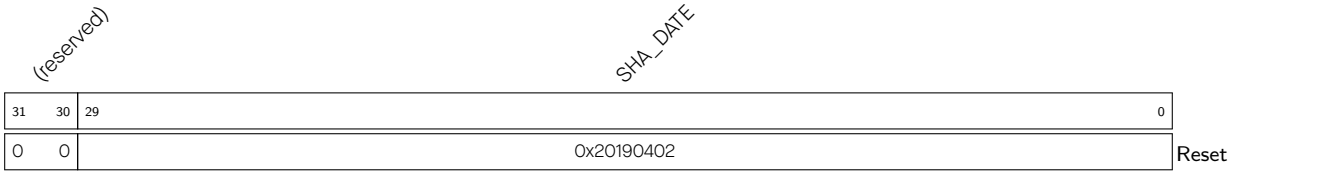
SHA_CLEAR_INTERRUPT Write 1 to clear DMA-SHA interrupt. (WO)

Register 14.11. SHA_INT_ENA_REG (0x0028)



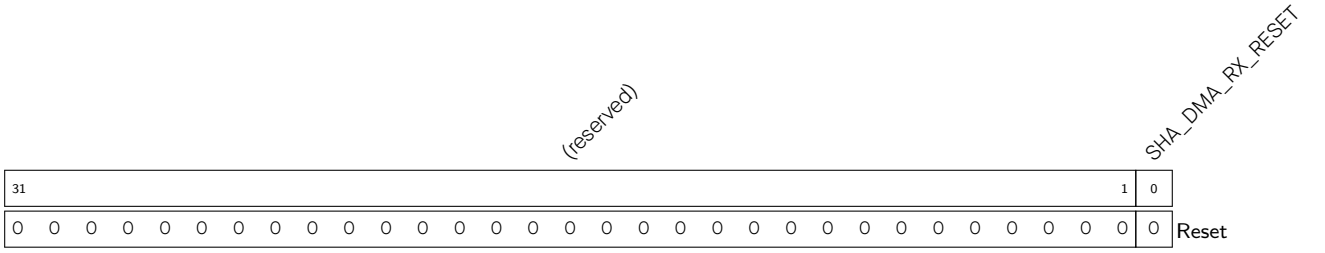
SHA_INTERRUPT_ENA Write 1 to enable DMA-SHA interrupt. (R/W)

Register 14.12. SHA_DATE_REG (0x002C)



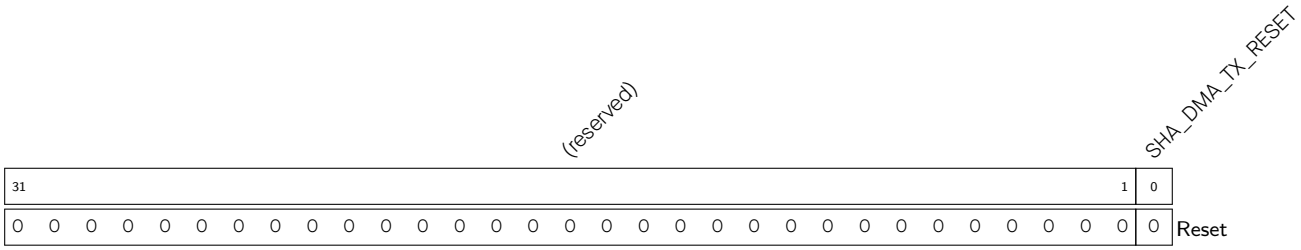
SHA_DATE Version control register. (R/W)

Register 14.13. SHA_DMA_RX_RESET_REG (0x0030)



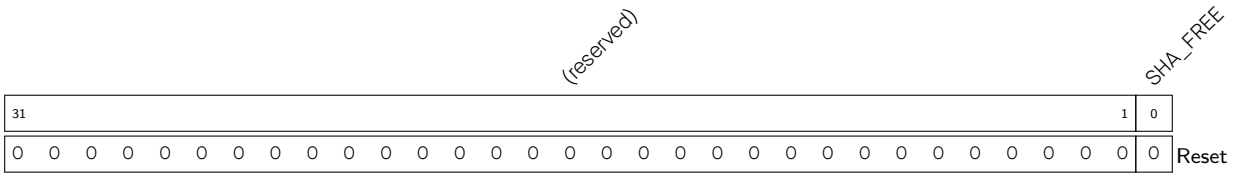
SHA_DMA_RX_RESET Write 1 to reset DMA-SHA RX fifo. (WO)

Register 14.14. SHA_DMA_TX_RESET_REG (0x0034)



SHA_DMA_TX_RESET Write 1 to reset DMA-SHA TX fifo. (WO)

Register 14.15. SHA_FREE_REG (0x0038)



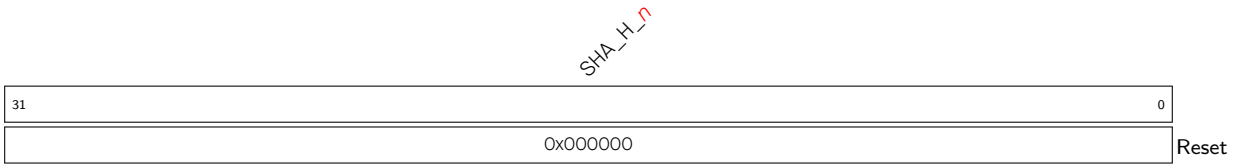
SHA_FREE Configures whether SHA can be set free (even if SHA is in operation) and be occupied by another module that needs it.

0: SHA cannot be set free

1: SHA can be set free.

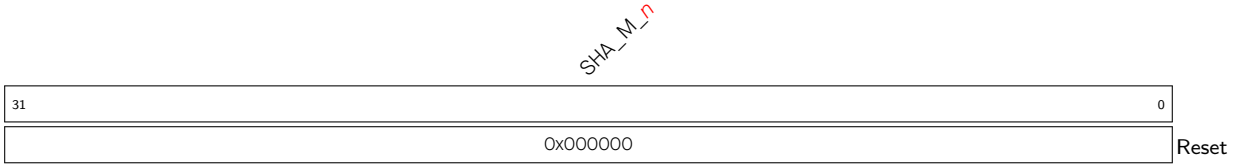
(R/W)

Register 14.16. SHA_H_*n*_REG (*n*: 0-15) (0x0040+4**n*)



SHA_H_*n* Represents the *n*th 32-bit piece of the Hash value. (R/W)

Register 14.17. SHA_M_*n*_REG (*n*: 0-31) (0x0080+4**n*)



SHA_M_*n* Represents the *n*th 32-bit piece of the message. (R/W)

Chapter 15

RSA Digital Signature Peripheral (RSA_DS)

15.1 Overview

The RSA Digital Signature Peripheral (RSA_DS) verifies the authenticity and integrity of a message using a cryptographic algorithm. This can be used to, for example, validate a device's identity to a server.

ESP32-S31 includes an RSA Digital Signature Peripheral (RSA_DS) providing hardware acceleration for RSA-based message signing operation. The HMAC peripheral is used as the key derivation function (KDF) to output RSA_DS_KEY using either a key stored in eFuse or deployed by the Key Manager as the input key. Subsequently, the RSA_DS peripheral uses RSA_DS_KEY to decrypt the pre-encrypted parameters and calculate the signature. The whole process happens in hardware so that neither the decryption key for the RSA parameters nor the input key for the HMAC key derivation function can be seen by users during signature calculation.

15.2 Features

The RSA_DS peripheral has the following features:

- Hardware RSA signature calculation as $Z = X^Y \bmod M$, with private-key operands up to 4096 bits
- Private key parameters are stored encrypted in flash, and decrypted inside the RSA_DS peripheral using RSA_DS_KEY; plaintext private key material is not visible to software
- SHA-256 digest over private key data and related parameters for tamper detection
- Hardware-provisioned RSA_DS_KEY (not readable by software), from HMAC-SHA256 using an eFuse-stored $HMAC\_KEY$ or from the Key Manager

15.3 Functional Description

15.3.1 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 a key (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 (RSA_DS_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 4 *eFuse Controller (eFuse) [to be added later]* and Chapter 12 *HMAC Accelerator (HMAC)*.

- Deploy from Key Manager module. The RSA_DS_KEY must be deployed by the Key Manager in a secured way.

The input message X will be sent directly to the RSA_DS peripheral by the software each time a signature is needed. After the RSA signature operation, the signature Z is read back by the software.

For better understanding, we define some symbols and functions here, which are only applicable to this chapter:

- **1**^{s}: A bit string consisting of s bits with the value of "1".
- **[**x**]**$_s$: A bit string of s bits, in which s is an integer multiple of 8 bits. If x is a number ($x < 2^s$), it is represented in little-endian byte order in the bit string. x may be a variable such as $[Y]_{4096}$ or a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]_t$ can be right-padded with $(s-t)$ number of zeros to reach s bits in length, and finally get $[x]_s$. For example, $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$.
- ||: A bit string concatenation operator for joining multiple-bit strings into a longer bit string.

15.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They have a particular RSA key length (up to 4096 bits). Two additional private key operands are needed: $\overline{r}$ and M'. These two operands are derived from Y and M.

Operands Y, M, $\overline{r}$, and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is input to the RSA_DS peripheral in this encrypted format, decrypted by the hardware, and then used for RSA signature calculation. A detailed description of how to generate C is provided in Section 15.3.3.

The RSA_DS peripheral supports RSA signature calculation $Z = X^Y \bmod M$, in which the length of operands should be $N = 32 \times x$ where $x \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ should be an arbitrary value in N, and all of them in a calculation must be of the same length, while the bit length of M' should always be 32. For more detailed information about RSA calculation, please refer to Section 13.3.2 *Large-Number Modular Exponentiation* in Chapter 13 *RSA Accelerator (RSA)*.

15.3.3 Software Prerequisites

If you want to use the RSA_DS peripheral, the software needs a series of preparations, as shown in Figure 15.3-1. 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.

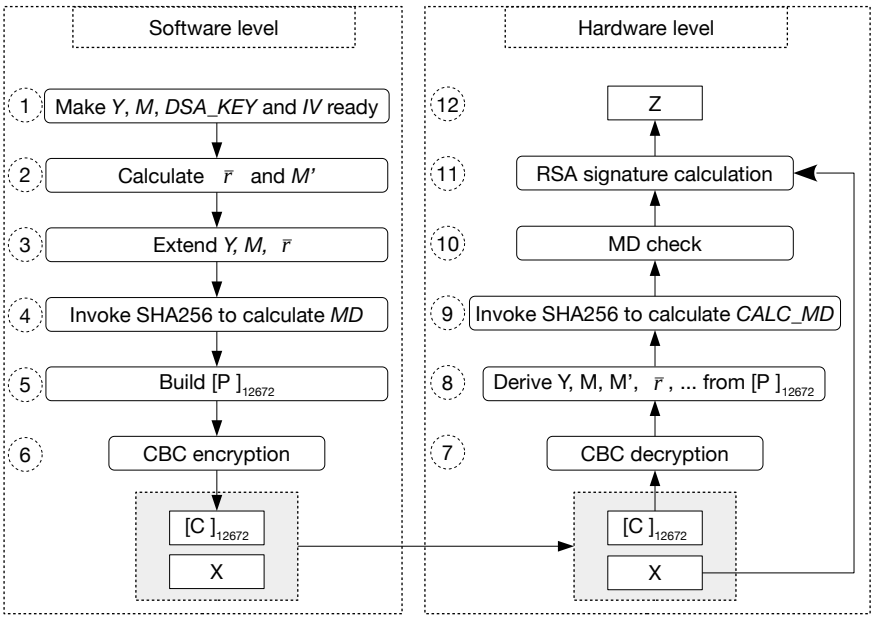


Figure 15.3-1. Software Preparations and Hardware Working Process

Note:

1. The software preparation (left side in Figure 15.3-1) is a one-time operation before any signature is calculated, while the hardware calculation (right side in Figure 15.3-1) repeats for every signature calculation.
2. Software preparation requires configuring the clock reset. For more information, please refer to Chapter 7 *Reset and Clock [to be added later]*.

Users need to follow the steps shown in the left part of Figure 15.3-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare operands Y and M whose lengths should meet the requirements in Section 15.3.2. Define $[L]_{32} = \frac{N}{32} - 1$ (i.e., for RSA 4096, $[L]_{32}$ == [0x80-1]$_{32}$). Prepare $[RSA\_DS\_KEY]_{256}$:
  - when calculated by the RSA_DS peripheral via the HMAC peripheral: RSA_DS_KEY = HMAC-SHA256 ($[HMAC\_KEY]_{256}$, 1^{256}).
  - when from Key Manager module: just use the deployed RSA_DS_KEY

  Generate a random $[IV]_{128}$ which should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 10 *AES Accelerator (AES)*.
- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$ in order to get $[Y]_{4096}$, $[M]_{4096}$, and $[\overline{r}]_{4096}$, respectively. This step is only required for Y, M, and $\overline{r}$ whose lengths are less than 4096 bits, since their largest length is 4096 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256: $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[Box]_{384}$), where $[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$) and $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a $[0x0808080808080808]_{64}$ string composed of 8 bytes (0x08). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.

- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[RSA\_DS\_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext with a length of 12672 bits. C can also be calculated as C = $[C]_{12672}$ = ($[\widehat{Y}]_{4096}||[\widehat{M}]_{4096}||[\widehat{\overline{r}}]_{4096}||[\widehat{Box}]_{384}$), where $[\widehat{Y}]_{4096}$, $[\widehat{M}]_{4096}$, $[\widehat{\overline{r}}]_{4096}$, $[\widehat{Box}]_{384}$ are the four sub-parameters of C, and correspond to the ciphertext of $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[Box]_{384}$ respectively.

15.3.4 RSA_DS Operation at the Hardware Level

The hardware operation is triggered each time an RSA 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 15.3-1**

   The decryption process is the inverse of Step 6 in Figure 15.3-1. The RSA_DS peripheral will call the AES accelerator to decrypt C in CBC block mode and get the resulting plaintext. The decryption process can be represented by P = AES-CBC-DEC (C, RSA_DS_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by the user. $[RSA\_DS\_KEY]_{256}$ is

   - when calculated by the RSA_DS peripheral via the HMAC peripheral: provided by the HMAC module, derived from $HMAC\_KEY$ stored in eFuse, which is not readable by users.
   - when from Key Manager module: RSA_DS_KEY deployed by Key Manager, which is not readable by users.

   With P, the RSA_DS peripheral can derive $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the inverse of Step 5.

2. **Check: Step 9 and 10 in Figure 15.3-1**

   The RSA_DS peripheral will perform two checks: MD check and padding check. The padding check is not shown in Figure 15.3-1, 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 the 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 15.3-1**

   The RSA_DS peripheral treats X (input by the user) and Y, M, $\overline{r}$ (decrypted in step 8) as big numbers. With M', all operands to perform X^Y mod M are in place. The operand length is defined by L only. The RSA_DS peripheral will calculate the signed result Z by calling RSA to perform $Z = X^Y$ mod M.

15.3.4.1 Hardware Occupation

During the operation, the following hardware modules will be occupied by ESP32-S31's RSA_DS:

- AES Accelerator

- RSA Accelerator
- SHA Accelerator

Each accelerator has a corresponding free register field to indicate whether RSA_DS can immediately occupy it. If the free register field is set to 1, RSA_DS will immediately reset the accelerator when it begins its operations. However, if the free register field is set to 0, RSA_DS will wait for the accelerator to finish its current operation, or up to a timeout limit set by RSA_DS_TIMEOUT_LIMIT.

Once RSA_DS has occupied the accelerators and begun its operation, they are released as follows:

- AES Accelerator will be released after CBC decryption.
- RSA Accelerator will be occupied during the whole RSA_DS operation.
- SHA Accelerator will be released after MD check.

If RSA_DS receives an external reset while it has ownership of one or more hardware accelerators, the reset is also propagated to those occupied accelerators. This ensures that any RSA_DS intermediate results temporarily stored within the accelerators are cleared, preventing leakage through an external reset attack.

Note:
Hardware occupation is a mechanism to protect multiplexed modules and storage space. When a module is hardware occupied, the user will fail to:

- read or write data to the module's registers or memories.
- disable the module clock.
- reset the module.

After hardware occupation is finished, the occupied module will be automatically reset. In addition, when the user performs a software reset to the master module, all the occupied modules will be reset at the same time.

15.3.5 RSA_DS Operation at the Software Level

The software steps below should be followed each time an RSA 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 15.3.4.

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 15.3.3.
   - To disable the PKCS#7 padding check, set RSA_DS_SKIP_PADDING_CHECK in RSA_DS_STATIC_REG to 1.
2. **Activate the RSA_DS peripheral**: Write 1 to RSA_DS_SET_START_REG.
3. **Select the source of RSA_DS_KEY**: Configure the RSA_DS_KEY_SOURCE field in RSA_DS_STATIC_REG, and check whether it is ready:
   - Write 1 to the RSA_DS_KEY_SOURCE field in RSA_DS_STATIC_REG to select the RSA_DS_KEY deployed by the Key Manager. Verify that RSA_DS_KEY has been successfully deployed by

checking the register field KEYMNG_KEY_DS_VLD. When its value is 1, it indicates that RSA_DS_KEY has been successfully deployed and is ready.

- Write 0 to the RSA_DS_KEY_SOURCE field in RSA_DS_STATIC_REG to select the RSA_DS_KEY from HMAC. Poll RSA_DS_QUERY_BUSY_REG until the software reads 0.

If the software does not read 0 in RSA_DS_QUERY_BUSY_REG after approximately 1 ms, it indicates a problem with HMAC initialization. In such a case, the software can read register RSA_DS_QUERY_KEY_WRONG_REG to get more information:

- If the software reads 0 in RSA_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 RSA_DS_QUERY_KEY_WRONG_REG, it indicates that HMAC was called, but the RSA_DS peripheral did not successfully get the RSA_DS_KEY value from the HMAC peripheral. This may indicate that the HMAC operation has been interrupted due to a software concurrency problem.

4. **Write IV to memory block RSA_DS_IV_MEM**: Write the content in the IV block to the memory block RSA_DS_IV_MEM, which is 16 bytes. For more information on the IV block, please refer to Chapter 10 *AES Accelerator (AES)*.
5. **Write X to memory block RSA_DS_X_MEM**: Write X_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, to memory block RSA_DS_X_MEM whose capacity is 128 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 RSA_DS_X_MEM block after the configured length of X (N bits, as described in Section 15.3.2), are ignored.
6. **Write C to corresponding memory blocks**: Write the four sub-parameters of C to corresponding memory blocks:
   - Write $\widehat{Y_i}$ ($i \in \{0, 1, \ldots, 127\}$) to RSA_DS_Y_MEM.
   - Write $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 127\}$) to RSA_DS_M_MEM.
   - Write $\widehat{r}_i$ ($i \in \{0, 1, \ldots, 127\}$) to RSA_DS_RB_MEM.
   - Write $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) to RSA_DS_BOX_MEM.

     The capacity of RSA_DS_Y_MEM, RSA_DS_M_MEM, and RSA_DS_RB_MEM is 128 words, whereas the capacity of RSA_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 RSA_DS_SET_CONTINUE_REG.
8. **Wait for the operation to be completed**: Poll register RSA_DS_QUERY_BUSY_REG until the software reads 0.
9. **Query check result**: Read register RSA_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 RSA_DS_Z_MEM. The memory block stores Z in little-endian byte order.
11. **Exit the operation**: Write 1 to RSA_DS_SET_FINISH_REG, and then poll RSA_DS_QUERY_BUSY_REG until the software reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

15.4 Memory Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_DS_Y_MEM	Memory block Y	512	0x0000	0x01FF	R/W
RSA_DS_M_MEM	Memory block M	512	0x0200	0x03FF	R/W
RSA_DS_RB_MEM	Memory block $\overline{r}$	512	0x0400	0x05FF	R/W
RSA_DS_BOX_MEM	Memory block Box	48	0x0600	0x062F	R/W
RSA_DS_IV_MEM	Memory block IV	16	0x0630	0x063F	R/W
RSA_DS_X_MEM	Memory block X	512	0x0800	0x09FF	R/W
RSA_DS_Z_MEM	Memory block Z	512	0x0A00	0x0BFF	R/W

15.5 Register Summary

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

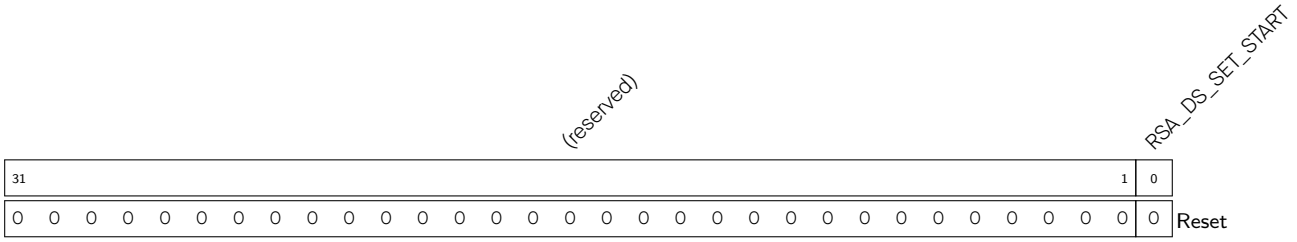
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Control/Status registers			
RSA_DS_SET_START_REG	Activates the RSA_DS peripheral	0x0E00	WT
RSA_DS_SET_CONTINUE_REG	Continues RSA_DS operation	0x0E04	WT
RSA_DS_SET_FINISH_REG	Ends RSA_DS operation	0x0E08	WT
RSA_DS_QUERY_BUSY_REG	Status of the RSA_DS peripheral	0x0E0C	RO
RSA_DS_QUERY_KEY_WRONG_REG	Checks the reason why RSA_DS_KEY is not ready	0x0E10	RO
RSA_DS_QUERY_CHECK_REG	Queries RSA_DS check result	0x0E14	RO
Configuration registers			
RSA_DS_STATIC_REG	Configures RSA_DS_KEY source and padding check bypass	0x0E18	R/W
RSA_DS_TIMEOUT_LIMIT_REG	Configures timeout limit for waiting sub-IP	0x0E24	R/W
version control register			
RSA_DS_DATE_REG	Version control register	0x0E20	R/W

15.6 Registers

The addresses in this section are relative to the RSA Digital Signature Peripheral (RSA_DS) base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Register 15.1. RSA_DS_SET_START_REG (0x0E00)



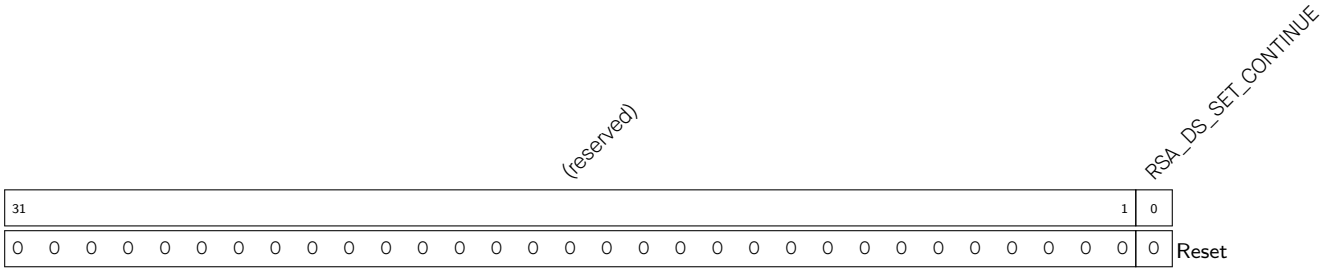
RSA_DS_SET_START Configures whether or not to activate the RSA_DS peripheral.

0: Invalid

1: Activate the RSA_DS peripheral

(WT)

Register 15.2. RSA_DS_SET_CONTINUE_REG (0x0E04)



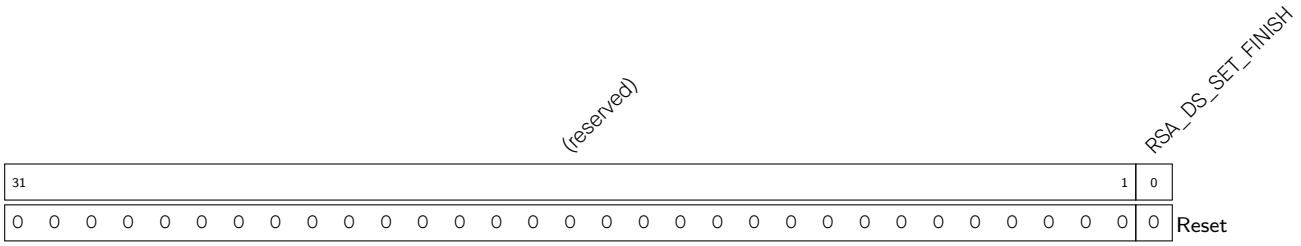
RSA_DS_SET_CONTINUE Configures whether or not to continue the RSA_DS operation.

0: No effect

1: Continue the RSA_DS operation

(WT)

Register 15.3. RSA_DS_SET_FINISH_REG (0x0E08)



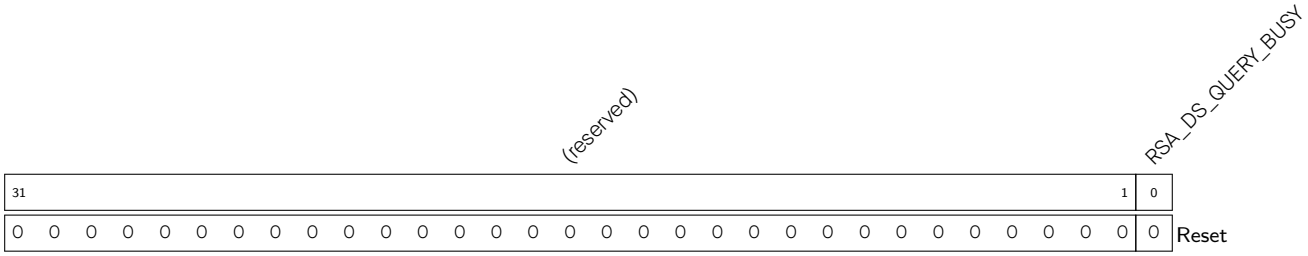
RSA_DS_SET_FINISH Configures whether or not to end the RSA_DS operation.

0: No effect

1: End the RSA_DS operation

(WT)

Register 15.4. RSA_DS_QUERY_BUSY_REG (0x0E0C)



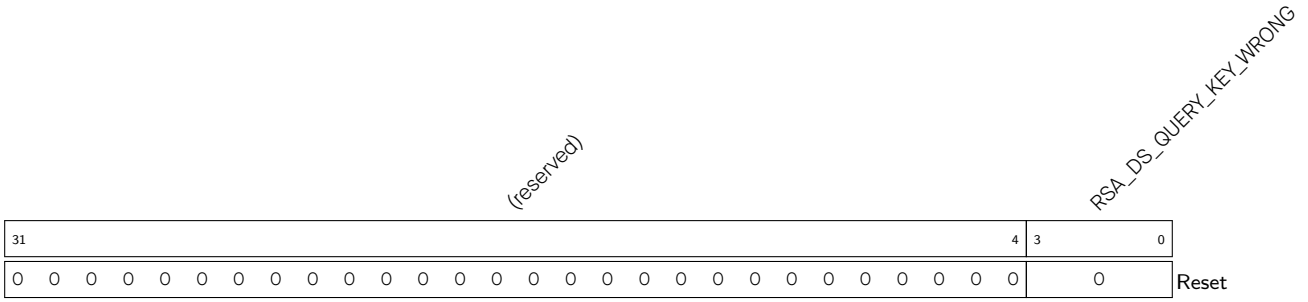
RSA_DS_QUERY_BUSY Represents whether or not the RSA_DS peripheral is idle.

0: The RSA_DS peripheral is idle

1: The RSA_DS peripheral is busy

(RO)

Register 15.5. RSA_DS_QUERY_KEY_WRONG_REG (0x0E10)



RSA_DS_QUERY_KEY_WRONG Represents the specific problem with HMAC initialization.

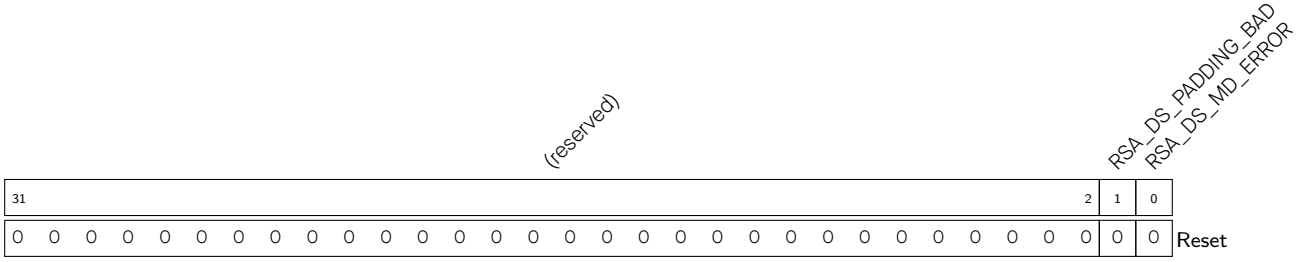
0: HMAC is not called

1-15: HMAC was activated, but the RSA_DS peripheral did not successfully receive the RSA_DS_KEY from the HMAC peripheral.

Larger than 15: Invalid

(RO)

Register 15.6. RSA_DS_QUERY_CHECK_REG (0x0E14)



RSA_DS_MD_ERROR Represents whether the MD check passes.

0: The MD check passes

1: The MD check fails

(RO)

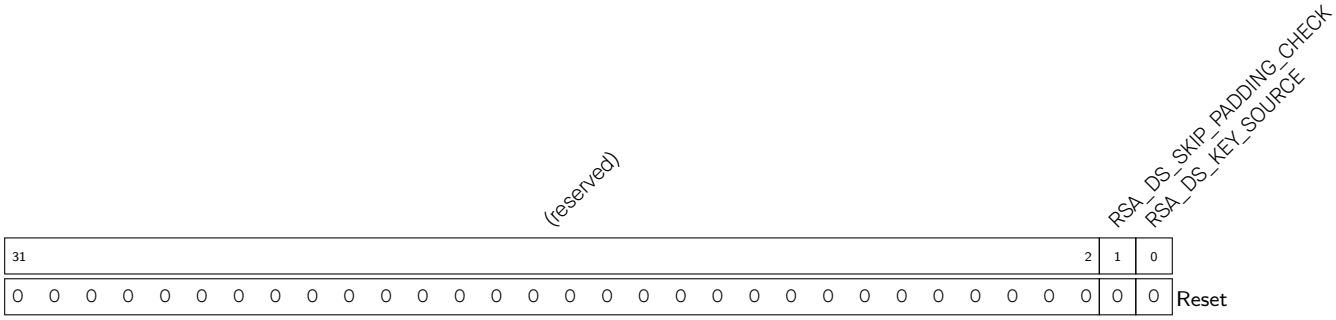
RSA_DS_PADDING_BAD Represents whether or not the padding check passes.

0: The padding check passes

1: The padding check fails

(RO)

Register 15.7. RSA_DS_STATIC_REG (0x0E18)



RSA_DS_KEY_SOURCE Represents the source of the RSA_DS_KEY.

0: HMAC

1: Key Manager

(R/W)

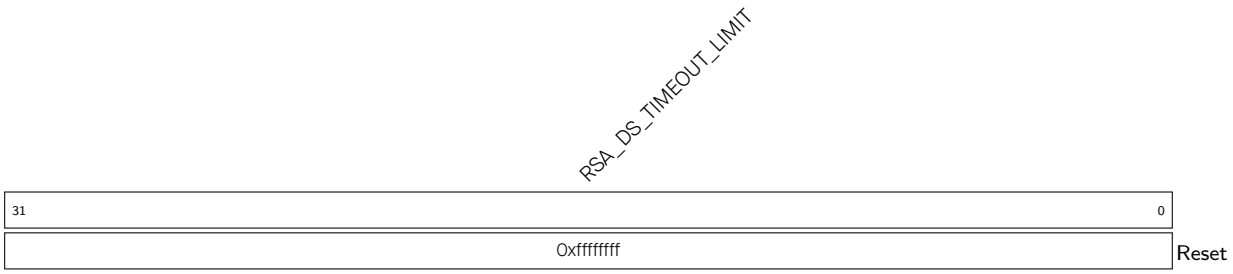
RSA_DS_SKIP_PADDING_CHECK Represents whether or not to skip padding check.

0: Padding check is enabled

1: Padding check is skipped

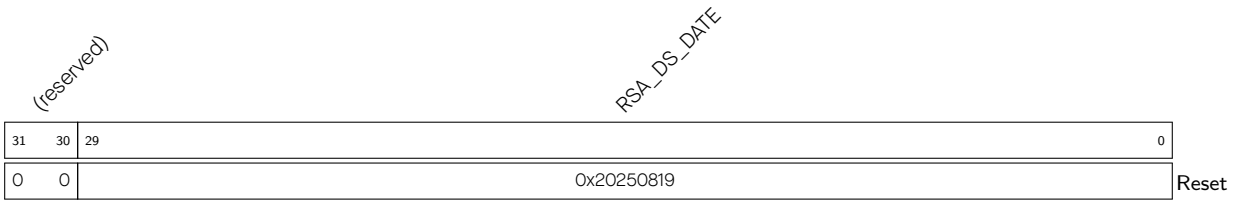
(R/W)

Register 15.8. RSA_DS_TIMEOUT_LIMIT_REG (0x0E24)



RSA_DS_TIMEOUT_LIMIT Configures timeout limit when waiting for AES, RSA and SHA accelerators. (R/W)

Register 15.9. RSA_DS_DATE_REG (0x0E20)



RSA_DS_DATE Version control register. (R/W)

Chapter 16

ECDSA Digital Signature Peripheral (ECDSA_DS)

16.1 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-S31's ECDSA Digital Signature Peripheral (ECDSA_DS) provides a secure and efficient environment for computing ECDSA signatures. It offers fast computations while ensuring the confidentiality of the signing process to prevent information leakage. This makes it a valuable tool for applications that require high-speed cryptographic operations with strong security guarantees. By using ECDSA_DS, users can be confident that their data is protected without sacrificing performance.

16.2 Features

ESP32-S31's ECDSA_DS module supports:

- Digital signature generation and verification
- Four elliptic curves, namely P-192, P-256, and P-384 as defined in FIPS 186-5, and SM2 as defined in GM/T 0003.2-2012
- Multiple message hash algorithms for ECDSA_DS operations, including SHA-224, SHA-256, SHA-384, SHA-512, SHA-512-224, and SHA-512-256 as defined in FIPS PUB 180-4 Spec, and SM3 as defined in GM/T 0004-2012
- State-dependent register access control in different operation states to ensure information security

16.3 ECDSA Basics

16.3.1 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.

16.3.2 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 be added later] to ensure that the private key is truly random and cannot be predicted or reproduced.
   - The private key is used for signature generation. The key must be kept secret and secure.
   - Once a private key is chosen, it should be burned into the eFuse in ESP32-S31. (Refer to Chapter 4 *eFuse Controller (eFuse) [to be added later]*).
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.

16.3.3 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.

16.3.4 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.

16.4 Functional Description

This section describes the details of ESP32-S31's ECDSA_DS module.

16.4.1 ECDSA_DS Working Modes

The ECDSA_DS module integrated in the ESP32-S31 has three working modes, which are Signature Generation, Signature Verification and Public Key Export modes.

Users can select the working mode for ECDSA_DS by configuring ECDSA_DS_WORK_MODE according to Table 16.4-1 below.

Table 16.4-1. ECDSA_DS Working Mode

ECDSA_DS_WORK_MODE	Working Mode
0	Signature Verification
1	Signature Generation
2	Public Key Export

Users can select the elliptic curves used by the ECDSA_DS module by configuring ECDSA_DS_ECC_CURVE according to Table 16.4-2.

Table 16.4-2. ECDSA_DS Elliptic Curve Selection

ECDSA_DS_ECC_CURVE	Elliptic Curve
0	P-192
1	P-256
2	P-384
3	SM2

Users can select the SHA algorithms for message hash by configuring ECDSA_DS_SHA_MODE according to Table 16.4-3.

Table 16.4-3. ECDSA_DS SHA Algorithm

ECDSA_DS_SHA_MODE	SHA Algorithm
1	SHA-224
2	SHA-256
3	SHA-384
4	SHA-512
5	SHA-512-224
6	SHA-512-256
14	SM3
Others	Invalid

Note:
Support for the Chinese cryptographic standard (SM2&SM3) can be disabled by EFUSE_DIS_SM_CRYPT.

Additionally, users can check the working states of ECDSA_DS by inquiring the ECDSA_DS_STATE_REG register and comparing the return value against the Table 16.4-4 below.

Table 16.4-4. ECDSA_DS Working State

ECDSA_DS_STATE_REG	State	Description
0	IDLE	Idle or completed operation. Corresponding to IDLE stage.
1	LOAD	Waiting for users to load information into ECDSA_DS. Corresponding to LOAD stage.
2	GAIN	Waiting for users to gain information from ECDSA_DS. Corresponding to GAIN stage.
3	BUSY	In the middle of a hardware operation. Corresponding to PREP, PROC, and POST stage.

16.4.2 Key Source

ESP32-S31's ECDSA_DS module can operate with a hardware-deployed key or a software-programmed key. The hardware-deployed key can be sourced from eFuse (refer to Chapter *4 eFuse Controller (eFuse) [to be added later]*) or Key Manager (refer to Chapter *21 Key Manager [to be added later]*). If a software-programmed key is used instead, the key should be separated into 32-bit blocks and written into ECDSA_DS_KEY_*n*_REG (*n* = 0-11). During configuration, ECDSA_DS_USE_HARDWARE_KEY should be set to 0 to enable ECDSA_DS to use the software-programmed key.

16.4.3 Data and Data Block

ESP32-S31's ECDSA_DS module can operate on data of 192, 256, 384, or 512 bits. For example, the data ($D[255:0]$) can be divided into 32-bit data blocks.

Take 256-bit long data as an example, $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with a smaller serial number correspond to the lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

16.4.3.1 Writing Data

Writing data means writing data to an ECDSA_DS memory block and using this data as the input to ECDSA_DS. To be specific, writing data to an ECDSA_DS memory block means writing $D[n][31:0]$ to the "starting address of this ECDSA_DS memory block + $4 \times n$".

For a 256-bit long data:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

Note:
When storing data, write only the data block of the required length. Do not append a 0 to the most significant bit. For example, for 192-bit data, store only the 192 bits without appending 0.

16.4.3.2 Reading Data

Reading data means reading data from the starting address of an ECDSA_DS memory block and using this data as the output from the ECDSA_DS. To be specific, reading data from an ECDSA_DS memory block means reading $D[n][31:0]$ from the "starting address of this ECDSA_DS memory block + $4 \times n$". For a 256-bit long data:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- ...
- read $D[7]$ from "starting address + 28"

Note:
When reading data, only the data block of the required length needs to be read. For example, to read 192-bit data, simply read the lower 192 bits (i.e., 6 data blocks).

16.4.3.3 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, refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

16.4.3.4 Parsing the Message

The message and its padding must be parsed into N 512-bit message blocks: $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$.

Note:
1. For more details about "parsing the message", refer to FIPS PUB 180-4 Spec > Section "Parsing the Message".
2. For more information on "message block", refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

16.4.4 Security Features

To ensure the security of the operation process, the ECDSA_DS module implements a variety of security functions.

16.4.4.1 High Anti-Attack Performance

ESP32-S31's ECDSA_DS module leverages ECC's enhanced anti-attack performance (refer to Section 11.4.3 *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-S31's ECDSA_DS module strong anti-attack performance.

16.4.4.2 State-Dependent Register Access Control

ESP32-S31'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_DS_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, refer to Table 16.4-4.

For detailed information on the state-dependent access control of each ECDSA_DS register, refer to Section Register Summary.

16.4.4.3 Hardware Occupation

During operation, ECDSA_DS occupies the following hardware modules:

- SHA Accelerator
- ECC Accelerator

Each accelerator has a corresponding free field (SHA_FREE and ECC_MULT_FREE) that indicates whether ECDSA_DS can occupy it immediately. If the free field is set to 1, ECDSA_DS resets the accelerator immediately when it starts operation. If the free field is set to 0, ECDSA_DS waits for the accelerator to finish its current operation, or until the timeout limit set by ECDSA_DS_TIMEOUT_LIMIT is reached.

Once ECDSA_DS has occupied the accelerators and started operation, they are released as follows: the SHA accelerator is released when ECDSA_DS_SHA_RELEASE_INT is triggered, while the ECC accelerator remains occupied for the entire ECDSA_DS operation.

If ECDSA_DS receives an external reset while it owns one or more hardware accelerators, the reset is also propagated to those accelerators. This ensures that any intermediate results temporarily stored in the accelerators are cleared, preventing leakage through an external reset attack.

Note:
Hardware occupation protects multiplexed modules and storage space. When a module is hardware occupied, users cannot:

- read or write the module's registers or memories.
- disable the module clock.

- reset the module.

When hardware occupation ends, the occupied module is automatically reset. In addition, if the user performs a software reset on the master module, all occupied modules are reset at the same time.

Notice: ESP32-S31's Secure Debug Controller (SDC) [to be added later] module also uses ECDSA_DS during its operations. By default, ECDSA_DS_FREE (value 1) allows Secure Debug Controller (SDC) [to be added later] to immediately reset any ongoing ECDSA_DS operation and occupy ECDSA_DS. To prevent this behavior, set ECDSA_DS_FREE to 0.

16.5 Programming Procedures

16.5.1 ECDSA_DS Process

The overall ECDSA_DS process consists of the following six stages.

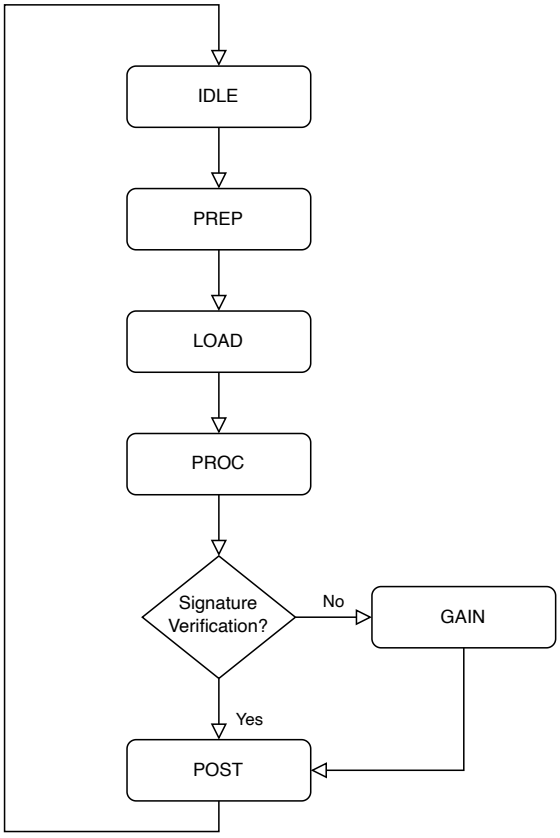


Figure 16.5-1. ECDSA_DS Process

The detailed programming procedures of each stage are described in the following sections.

16.5.1.1 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. Select one of the following sources:

i. eFuse: Write the key value in KEY*n* (*n* = 0-5), and set the corresponding EFUSE_KEY_PURPOSE_*n* as ECDSA_KEY. Refer to Chapter *4 eFuse Controller (eFuse) [to be added later]* for more detailed configuration steps.

ii. Key Manager: Use one of the Key Manager modes to deploy a key for ECDSA. Refer to Chapter *21 Key Manager [to be added later]* for more detailed configuration steps.

iii. ECDSA Software Registers: Write the key value in ECDSA_DS_KEY_*n*_REG (*n* = 0-11) during LOAD stage.

2. Configure ECDSA_DS_CONF_REG, including the following fields:

(a) ECDSA_DS_WORK_MODE: Select the working mode of ECDSA_DS.

(b) ECDSA_DS_ECC_CURVE: Select the elliptic curve of ECDSA_DS.

(c) ECDSA_DS_SOFTWARE_SET_Z: Configure whether to use direct input z.

(d) ECDSA_DS_DETERMINISTIC_K: Configure whether to use deterministic k.

(e) ECDSA_DS_USE_HARDWARE_KEY: Configure whether the ECDSA key is sourced from a software-programmed register or a hardware source (eFuse or Key Manager).

3. Configure the register field ECDSA_DS_START to enter the PREP stage.

Note:
For more details about Deterministic ECDSA, refer to RFC6979.

16.5.1.2 PREP Stage

In the PREP stage, ECDSA_DS_STATE_REG is BUSY, and ECDSA_DS performs preparation.

Wait for the PREP stage to end by polling ECDSA_DS_BUSY until it is not BUSY. Then ECDSA_DS will automatically enter the LOAD stage.

16.5.1.3 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_DS_MEM_Z.
   - ECDSA_DS SHA interface: generate z from the message. For details, refer to Section 16.5.1.7.
2. According to the selected ECDSA_DS_WORK_MODE, configure as follows:
   - Signature Verification:
     (a) write signature (r, s) to ECDSA_DS_MEM_R and ECDSA_DS_MEM_S.
     (b) write public key (Qx, Qy) to ECDSA_DS_MEM_Qx and ECDSA_DS_MEM_Qy.
   - Signature Generation:
     (a) no other configuration is required.

- Public Key Export:

  (a) no other configuration is required.

3. If using a key from ECDSA software registers, configure ECDSA_DS_KEY_*n*_REG (*n* = 0-11), in big-endian format. A 192-bit key uses *n* = 0-5, a 256-bit key uses *n* = 0-7, and a 384-bit key uses *n* = 0-11. Given $Key[255:0]$, $Key[255:224]$ is written into *n* = 7, and $Key[31:0]$ is written into *n* = 0.
4. Write 1 to ECDSA_DS_LOAD_DONE, indicating that the configuration is done. Then ECDSA_DS will automatically enter the PROC stage.

16.5.1.4 PROC Stage

In the PROC stage, ECDSA_DS_STATE_REG is BUSY, and ECDSA_DS performs operation based on the selected working mode.

Wait for the PROC stage to end by polling ECDSA_DS_STATE_REG until the field ECDSA_DS_BUSY indicates not BUSY. Then ECDSA_DS will automatically enter either the GAIN stage or the POST stage depending on the selected working mode:

- Signature generation: GAIN stage
- Signature verification: POST stage
- Public key export: GAIN stage

16.5.1.5 GAIN Stage

When the Signature Generation mode or the Public Key Export mode is selected, ECDSA_DS enters the GAIN stage after PROC stage:

1. Read data from the ECDSA_DS memory:
   - read signature (r, s) from ECDSA_DS_MEM_R and ECDSA_DS_MEM_S only in the Signature Generation mode.
   - read public key (Qx, Qy) from ECDSA_DS_MEM_Qx and ECDSA_DS_MEM_Qy both in the Signature Generation mode and the Public Key Export mode.
2. Write 1 to ECDSA_DS_GET_DONE, indicating that the GAIN stage is done. Then ECDSA_DS will automatically enter the POST stage.

16.5.1.6 POST Stage

In the POST stage, ECDSA_DS_STATE_REG is BUSY, and ECDSA_DS performs wrap-up work of the operation.

Wait for the POST stage to end by polling ECDSA_DS_BUSY until it is not BUSY. Then ECDSA_DS will automatically return to the IDLE stage.

16.5.1.7 ECDSA_DS SHA Interface

ESP32-S31's ECDSA_DS module can automatically execute a hash operation and generate 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 16.4.3.3.
2. Parse the message and its padding into message blocks. See details in Section 16.4.3.4.
3. Process the current message block.
   - Write the current message block into ECDSA_DS_MEM_M.
4. Start the ECDSA_DS SHA interface[1].
   - If this is the first time to execute this step, write 1 to ECDSA_DS_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_DS_SHA_CONTINUE to continue the operation.
5. Check the progress of the current message block processing by polling.
   - Poll register ECDSA_DS_SHA_BUSY until it's IDLE, indicating that the interface has completed the operation for the current message block.
6. Process the next message block:
   - If there is a message block to process, go back to Step 3.
   - If there are no more message blocks, exit.

Note:

1. In this step, the software can also write the next message block to be processed, if any, in register ECDSA_DS_MEM_M, while the interface starts SHA operation, to save time.
2. You are resuming the ECDSA_DS SHA interface with the previously paused operation.

16.5.2 Clock

ECDSA_DS uses two types of clocks:

- clk_ecdsa: function clock for ECDSA_DS to operate
- clk_apb_ecdsa: bus clock used to configure ECDSA_DS registers

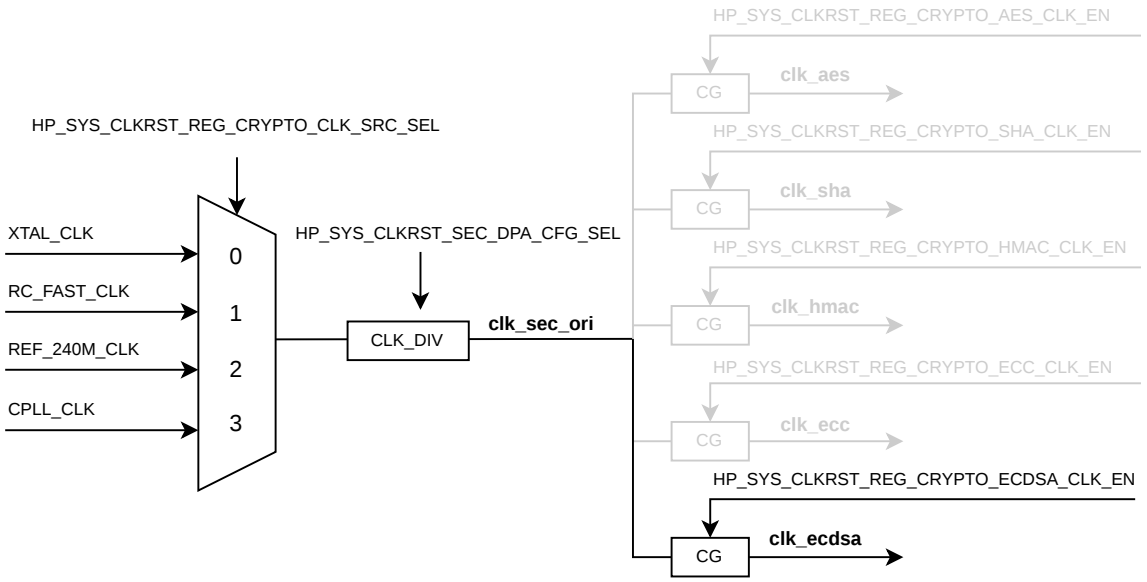


Figure 16.5-2. clk_ecdsa Diagram

As shown in Figure 16.5-2 *clk_ecdsa Diagram*, the secure function source clock clk_sec_ori can be sourced from:

- XTAL_CLK
- RC_FAST_CLK
- REF_240M_CLK
- CPLL_CLK

ECDSA_DS's function clock clk_ecdsa is configured by two clock gating registers, HP_SYS_CLKRST_REG_CRYPTO_SYS_CLK_EN and HP_SYS_CLKRST_REG_CRYPTO_ECDSA_CLK_EN. When both registers are set to 1, the clock gates are enabled, and clk_ecdsa is activated.

The clock division is configured by the clock register HP_SYS_CLKRST_DPA_CTRL0_REG and the eFuse field EFUSE_SEC_DPA_LEVEL (refer to 7 *Reset and Clock [to be added later]* for more details).

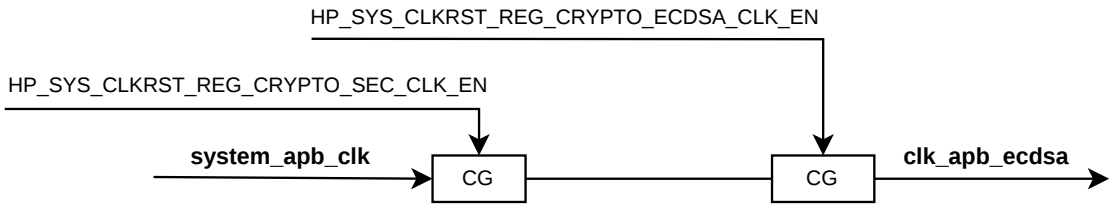


Figure 16.5-3. clk_apb_ecdsa Diagram

As shown in Figure 16.5-3 *clk_apb_ecdsa Diagram*, ECDSA_DS's configuration clock clk_apb_ecdsa can be sourced from system_apb_clk.

ECDSA_DS's configuration clock clk_apb_ecdsa is controlled by two clock gating registers, HP_SYS_CLKRST_REG_CRYPTO_SEC_CLK_EN and HP_SYS_CLKRST_REG_CRYPTO_ECDSA_CLK_EN. When

both registers are set to 1, the clock gates are enabled, and clk_apb_ecdsa is activated.

For more information about clocks, see Section *7 Reset and Clock [to be added later]*.

16.5.3 Reset

The ECDSA_DS module supports one reset domain. Write 1 and then 0 to HP_SYS_CLKRST_REG_CRYPTO_ECDSA_RST_EN to reset the entire ECDSA_DS module.

16.6 Interrupts

ESP32-S31's ECDSA_DS module can generate the interrupt signal ECDSA_DS_INTR and send it to Interrupt Matrix.

The ECDSA_DS module has four interrupt sources that can generate the ECDSA_DS_INTR interrupt signal as follows. Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*.

- ECDSA_DS_PREP_DONE_INT: triggered on the completion of the PREP stage. This interrupt source is configured by the following registers:
  - ECDSA_DS_PREP_DONE_INT_RAW: stores the raw interrupt status of ECDSA_DS_PREP_DONE_INT.
  - ECDSA_DS_PREP_DONE_INT_ST: indicates the status of the ECDSA_DS_PREP_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_DS_PREP_DONE_INT_RAW field.
  - ECDSA_DS_PREP_DONE_INT_ENA: enables/disables the ECDSA_DS_PREP_DONE_INT interrupt.
  - ECDSA_DS_PREP_DONE_INT_CLR: set this bit to clear the ECDSA_DS_PREP_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_DS_PREP_DONE_INT_RAW and ECDSA_DS_PREP_DONE_INT_ST will be cleared.
- ECDSA_DS_PROC_DONE_INT: triggered on the completion of the PROC stage. This interrupt source is configured by the following registers:
  - ECDSA_DS_PROC_DONE_INT_RAW: stores the raw interrupt status of ECDSA_DS_PROC_DONE_INT.
  - ECDSA_DS_PROC_DONE_INT_ST: indicates the status of the ECDSA_DS_PROC_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_DS_PROC_DONE_INT_RAW field.
  - ECDSA_DS_PROC_DONE_INT_ENA: enables/disables the ECDSA_DS_PROC_DONE_INT interrupt.
  - ECDSA_DS_PROC_DONE_INT_CLR: set this bit to clear the ECDSA_DS_PROC_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_DS_PROC_DONE_INT_RAW and ECDSA_DS_PROC_DONE_INT_ST will be cleared.
- ECDSA_DS_POST_DONE_INT: triggered on the completion of the POST stage. This interrupt source is configured by the following registers:
  - ECDSA_DS_POST_DONE_INT_RAW: stores the raw interrupt status of ECDSA_DS_POST_DONE_INT.
  - ECDSA_DS_POST_DONE_INT_ST: indicates the status of the ECDSA_DS_POST_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_DS_POST_DONE_INT_RAW field.

  - ECDSA_DS_POST_DONE_INT_ENA: enables/disables the ECDSA_DS_POST_DONE_INT interrupt.
  - ECDSA_DS_POST_DONE_INT_CLR: set this bit to clear the ECDSA_DS_POST_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_DS_POST_DONE_INT_RAW and ECDSA_DS_POST_DONE_INT_ST will be cleared.

- ECDSA_DS_SHA_RELEASE_INT: triggered when SHA is released. This interrupt source is configured by the following registers:
  - ECDSA_DS_SHA_RELEASE_INT_RAW: stores the raw interrupt status of ECDSA_DS_SHA_RELEASE_INT.
  - ECDSA_DS_SHA_RELEASE_INT_ST: indicates the status of the ECDSA_DS_SHA_RELEASE_INT interrupt. This field is generated by enabling/disabling the ECDSA_DS_SHA_RELEASE_INT_RAW field via ECDSA_DS_SHA_RELEASE_INT_ENA.
  - ECDSA_DS_SHA_RELEASE_INT_ENA: enables/disables the ECDSA_DS_SHA_RELEASE_INT interrupt.
  - ECDSA_DS_SHA_RELEASE_INT_CLR: set this bit to clear the ECDSA_DS_SHA_RELEASE_INT interrupt status. By setting this bit to 1, fields ECDSA_DS_SHA_RELEASE_INT_RAW and ECDSA_DS_SHA_RELEASE_INT_ST will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

16.7 Memory Blocks

ECDSA_DS's memory blocks store input data and output data of the operation.

Table 16.7-1. ECDSA_DS Memory Blocks

Memory	Size (byte)	Starting Address*	Ending Address *	Access
ECDSA_DS_MEM_M	128	0x280	0x2FF	R/W
ECDSA_DS_MEM_R	48	0x3E0	0x40F	R/W
ECDSA_DS_MEM_S	48	0x410	0x43F	R/W
ECDSA_DS_MEM_Z	48	0x440	0x46F	R/W
ECDSA_DS_MEM_Qx	48	0x470	0x49F	R/W
ECDSA_DS_MEM_Qy	48	0x4A0	0x4CF	R/W

* Address offset related to ECDSA_DS base address is provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

16.8 Register Summary

The addresses in this section are relative to ECDSA_DS base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

To enhance security, ECDSA_DS registers have different read and write permissions in different operating states. The following is the abbreviation and corresponding relationship of each state:

- PI: IDLE state, corresponding to IDLE stage.
- PL: LOAD state, corresponding to LOAD stage.
- PG: GAIN state, corresponding to GAIN stage.
- PB: BUSY state, corresponding to PREP, PROC, and POST stage.

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

Name	Description	Address	Access			
			PI	PL	PG	PB
Data Memory	See Table 16.7-1.					
Configuration Registers						
ECDSA_DS_CONF_REG	ECDSA_DS configuration register	0x0004	R/W	N/A		
ECDSA_DS_START_REG	ECDSA_DS start register	0x001C	WT			N/A
ECDSA_DS_TIMEOUT_LIMIT_REG	ECDSA_DS timeout limit configuration register	0x0028	R/W	RO		
ECDSA_DS_FREE_REG	ECDSA_DS free register	0x002C	R/W			
Clock and Reset Register						
ECDSA_DS_CLK_REG	ECDSA_DS clock gate register	0x0008	R/W	N/A		
Interrupt Registers						
ECDSA_DS_INT_RAW_REG	ECDSA_DS interrupt raw register	0x000C	RO/WTC/SS			
ECDSA_DS_INT_ST_REG	ECDSA_DS interrupt status register	0x0010	RO			
ECDSA_DS_INT_ENA_REG	ECDSA_DS interrupt enable register	0x0014	R/W			
ECDSA_DS_INT_CLR_REG	ECDSA_DS interrupt clear register	0x0018	WT			
Status Registers						
ECDSA_DS_STATE_REG	ECDSA_DS state register	0x0020	RO			
Result Register						
ECDSA_DS_RESULT_REG	ECDSA_DS result register	0x0024	RO/SS			N/A
Key Registers						
ECDSA_DS_KEY_0_REG	ECDSA_DS key data register 0	0x0040	RO	RW	RO	
ECDSA_DS_KEY_1_REG	ECDSA_DS key data register 1	0x0044	RO	RW	RO	
ECDSA_DS_KEY_2_REG	ECDSA_DS key data register 2	0x0048	RO	RW	RO	
ECDSA_DS_KEY_3_REG	ECDSA_DS key data register 3	0x004C	RO	RW	RO	
ECDSA_DS_KEY_4_REG	ECDSA_DS key data register 4	0x0050	RO	RW	RO	
ECDSA_DS_KEY_5_REG	ECDSA_DS key data register 5	0x0054	RO	RW	RO	
ECDSA_DS_KEY_6_REG	ECDSA_DS key data register 6	0x0058	RO	RW	RO	
ECDSA_DS_KEY_7_REG	ECDSA_DS key data register 7	0x005C	RO	RW	RO	
ECDSA_DS_KEY_8_REG	ECDSA_DS key data register 8	0x0060	RO	RW	RO	
ECDSA_DS_KEY_9_REG	ECDSA_DS key data register 9	0x0064	RO	RW	RO	
ECDSA_DS_KEY_10_REG	ECDSA_DS key data register 10	0x0068	RO	RW	RO	
ECDSA_DS_KEY_11_REG	ECDSA_DS key data register 11	0x006C	RO	RW	RO	
SHA Registers						

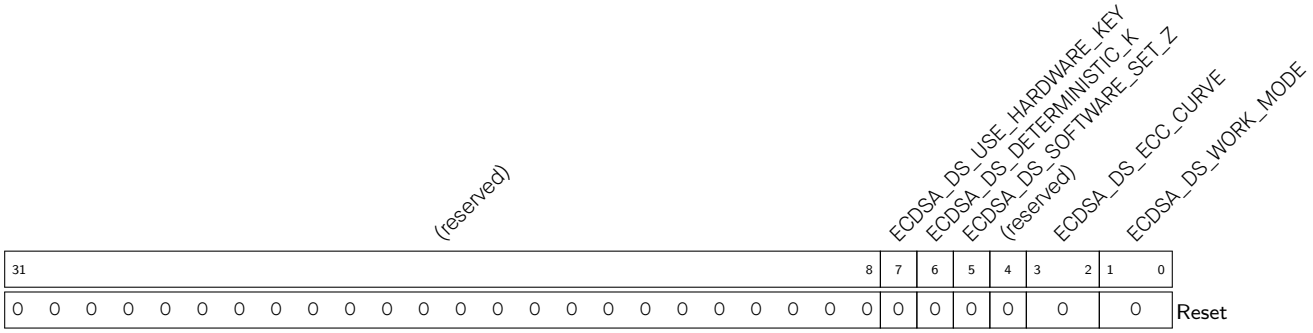
Name	Description	Address	Access			
			PI	PL	PG	PB
ECDSA_DS_SHA_MODE_REG	ECDSA_DS SHA-control register (hash algorithm)	0x0200	N/A	R/W	N/A	
ECDSA_DS_SHA_START_REG	ECDSA_DS SHA-control register (operation)	0x0210	N/A	WT	N/A	
ECDSA_DS_SHA_CONTINUE_REG	ECDSA_DS SHA-control register (operation)	0x0214	N/A	WT	N/A	
ECDSA_DS_SHA_BUSY_REG	ECDSA_DS SHA-control status register	0x0218	N/A	RO	N/A	
Version Register						
ECDSA_DS_DATE_REG	Version control register	0x00FC	R/W	N/A		

16.9 Registers

The addresses in this section are relative to ECDSA_DS base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 16.1. ECDSA_DS_CONF_REG (0x0004)



ECDSA_DS_WORK_MODE Configures the working mode of ECDSA_DS.

0: Signature Verification Mode

1: Signature Generation Mode

2: Public Key Export Mode

3: Invalid

(R/W)

ECDSA_DS_ECC_CURVE Configures the elliptic curve used.

0: P-192

1: P-256

2: P-384

3: SM2

(R/W)

ECDSA_DS_SOFTWARE_SET_Z Configures how the parameter z is set.

0: Generated from SHA result

1: Written by software

(R/W)

ECDSA_DS_DETERMINISTIC_K Configures how the parameter k is set.

0: Automatically generated by TRNG

1: Generated by the deterministic derivation algorithm

(R/W)

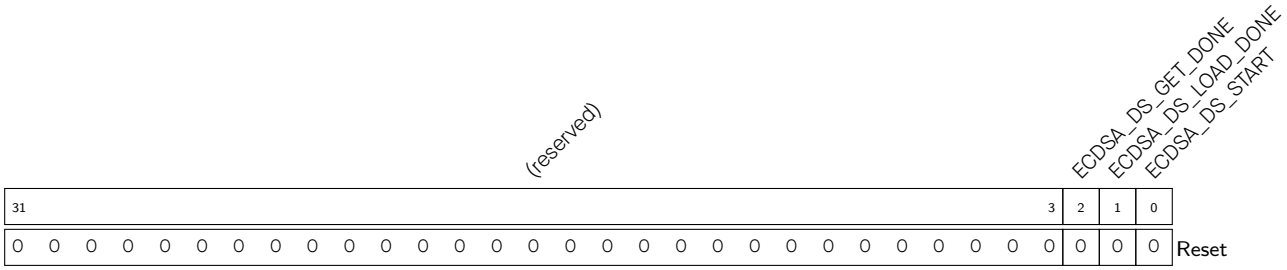
ECDSA_DS_USE_HARDWARE_KEY Configures whether to use software key or hardware key.

0: Software key

1: Hardware key

(R/W)

Register 16.2. ECDSA_DS_START_REG (0x001C)



ECDSA_DS_START Configures whether to start the ECDSA_DS operation. This bit will be self-cleared after configuration.

0: No effect

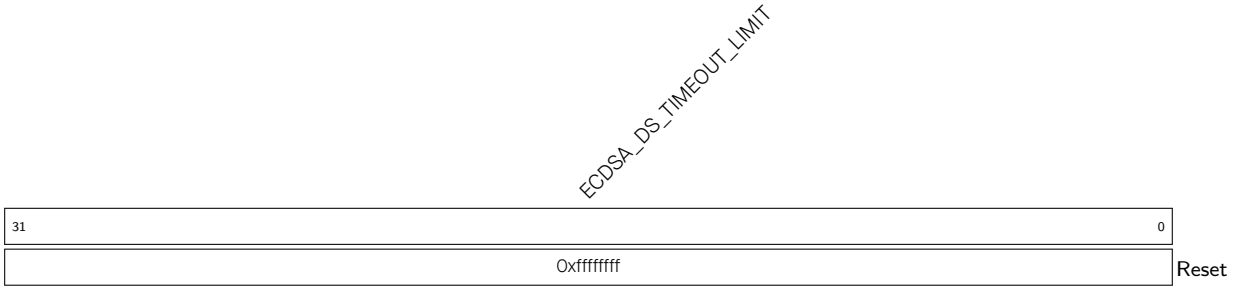
1: Start the ECDSA_DS operation

(WT)

ECDSA_DS_LOAD_DONE Write 1 to generate a signal indicating the ECDSA_DS's LOAD operation is done. This bit will be self-cleared after configuration. (WT)

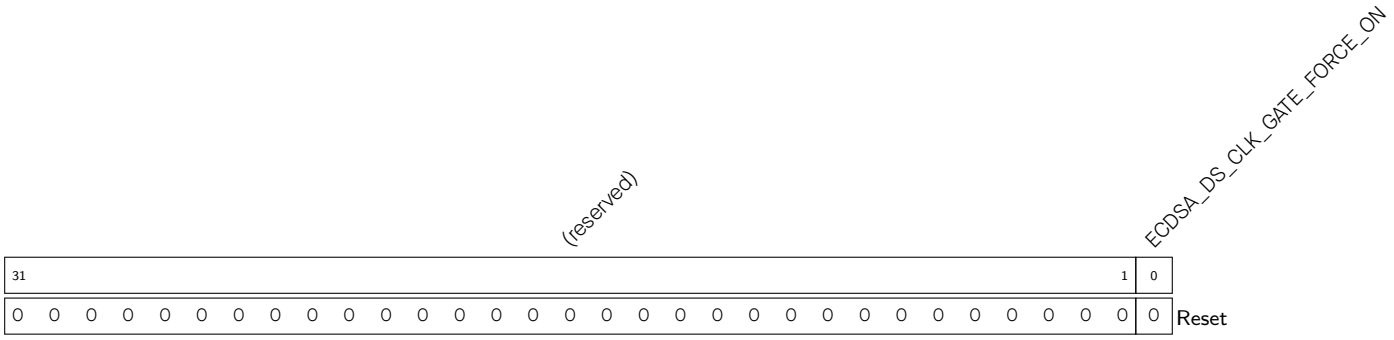
ECDSA_DS_GET_DONE Write 1 to generate a signal indicating the ECDSA_DS's GAIN operation is done. This bit will be self-cleared after configuration. (WT)

Register 16.3. ECDSA_DS_TIMEOUT_LIMIT_REG (0x0028)



ECDSA_DS_TIMEOUT_LIMIT Configures the timeout limit for waiting until shared cryptographic accelerators complete before reusing them for subsequent operations. (R/W)

Register 16.4. ECDSA_DS_CLK_REG (0x0008)



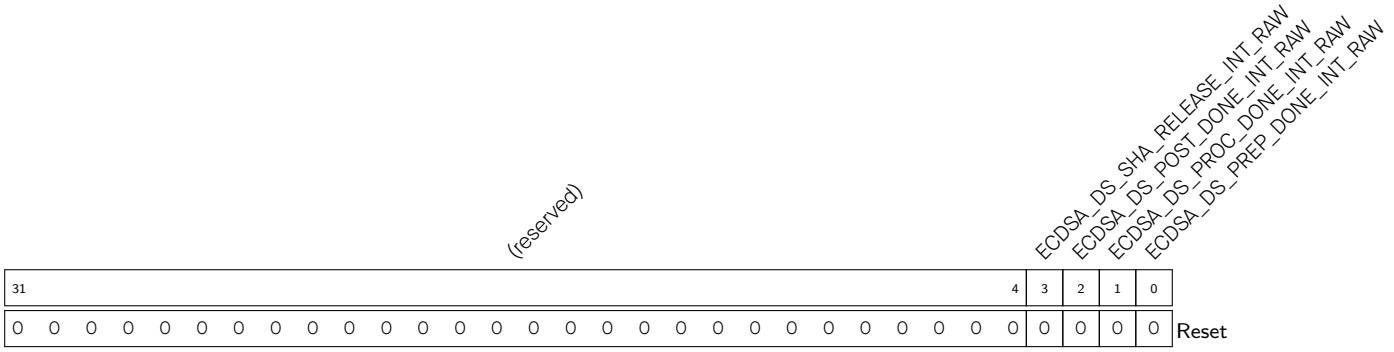
ECDSA_DS_CLK_GATE_FORCE_ON Configures whether to force on the ECDSA_DS memory clock gate.

0: No effect

1: Force on

(R/W)

Register 16.5. ECDSA_DS_INT_RAW_REG (0x000C)



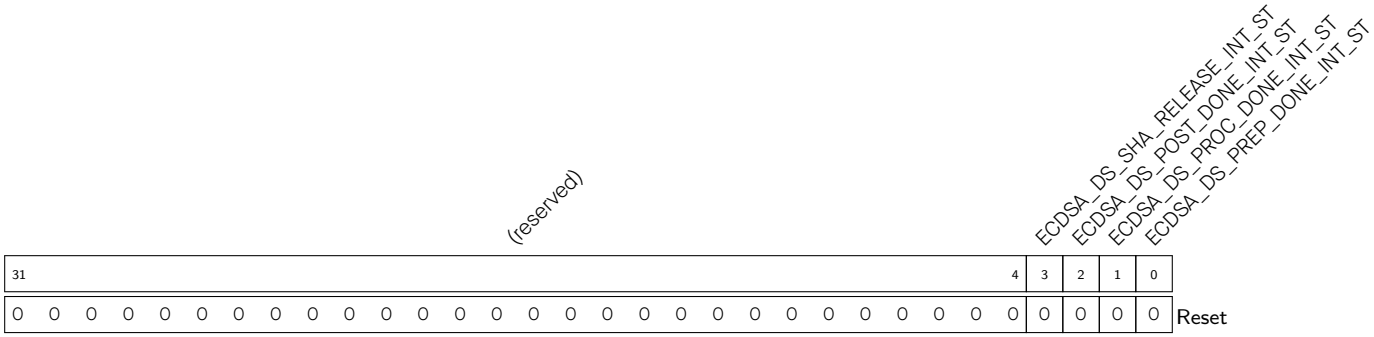
ECDSA_DS_PREP_DONE_INT_RAW The raw interrupt status of the ECDSA_DS_PREP_DONE_INT interrupt. (R/SS/WTC)

ECDSA_DS_PROC_DONE_INT_RAW The raw interrupt status of the ECDSA_DS_PROC_DONE_INT interrupt. (R/SS/WTC)

ECDSA_DS_POST_DONE_INT_RAW The raw interrupt status of the ECDSA_DS_POST_DONE_INT interrupt. (R/SS/WTC)

ECDSA_DS_SHA_RELEASE_INT_RAW The raw interrupt status of the ECDSA_DS_SHA_RELEASE_INT interrupt. (R/SS/WTC)

Register 16.6. ECDSA_DS_INT_ST_REG (0x0010)



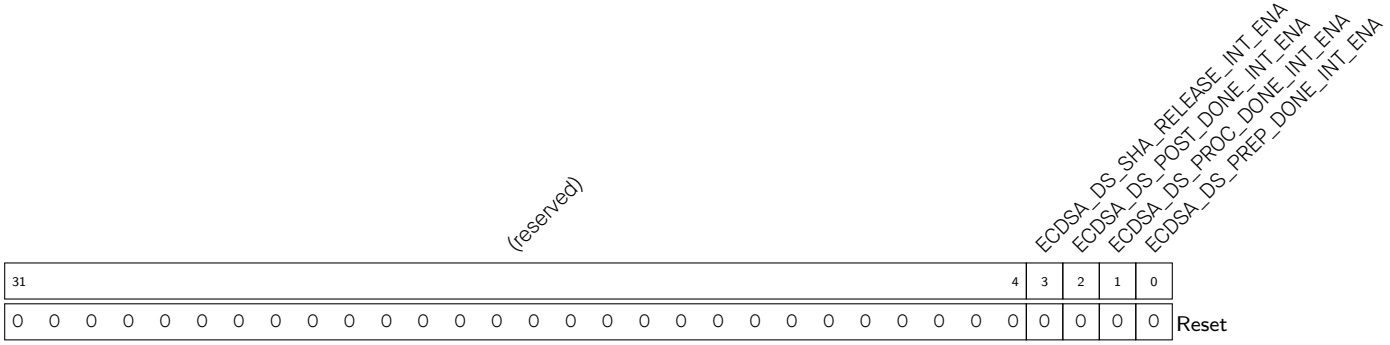
ECDSA_DS_PREP_DONE_INT_ST The masked interrupt status of the ECDSA_DS_PREP_DONE_INT interrupt. (RO)

ECDSA_DS_PROC_DONE_INT_ST The masked interrupt status of the ECDSA_DS_PROC_DONE_INT interrupt. (RO)

ECDSA_DS_POST_DONE_INT_ST The masked interrupt status of the ECDSA_DS_POST_DONE_INT interrupt. (RO)

ECDSA_DS_SHA_RELEASE_INT_ST The masked interrupt status of the ECDSA_DS_SHA_RELEASE_INT interrupt. (RO)

Register 16.7. ECDSA_DS_INT_ENA_REG (0x0014)



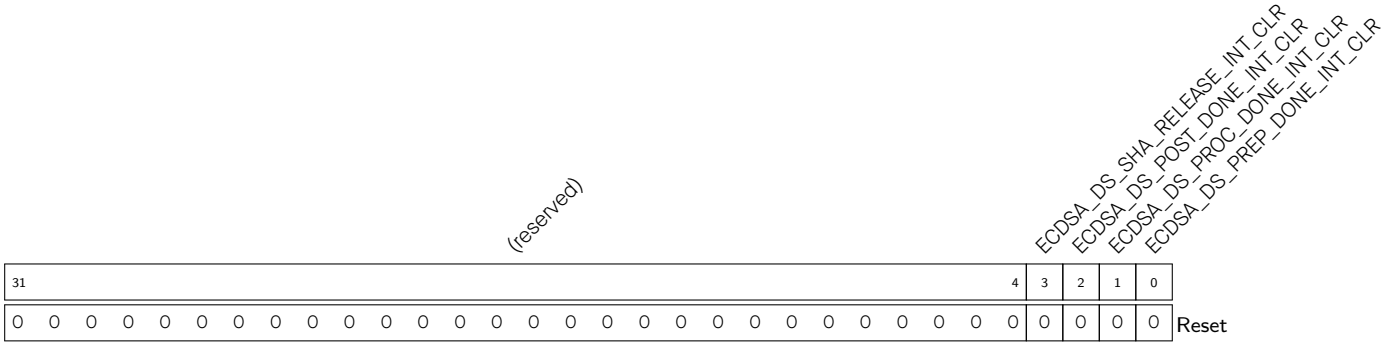
ECDSA_DS_PREP_DONE_INT_ENA Write 1 to enable the ECDSA_DS_PREP_DONE_INT interrupt. (R/W)

ECDSA_DS_PROC_DONE_INT_ENA Write 1 to enable the ECDSA_DS_PROC_DONE_INT interrupt. (R/W)

ECDSA_DS_POST_DONE_INT_ENA Write 1 to enable the ECDSA_DS_POST_DONE_INT interrupt. (R/W)

ECDSA_DS_SHA_RELEASE_INT_ENA Write 1 to enable the ECDSA_DS_SHA_RELEASE_INT interrupt. (R/W)

Register 16.8. ECDSA_DS_INT_CLR_REG (0x0018)



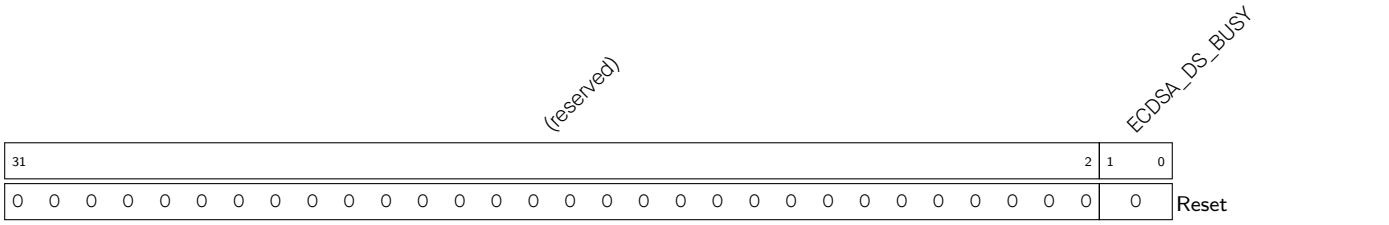
ECDSA_DS_PREP_DONE_INT_CLR Write 1 to clear the ECDSA_DS_PREP_DONE_INT interrupt. (WT)

ECDSA_DS_PROC_DONE_INT_CLR Write 1 to clear the ECDSA_DS_PROC_DONE_INT interrupt. (WT)

ECDSA_DS_POST_DONE_INT_CLR Write 1 to clear the ECDSA_DS_POST_DONE_INT interrupt. (WT)

ECDSA_DS_SHA_RELEASE_INT_CLR Write 1 to clear the ECDSA_DS_SHA_RELEASE_INT interrupt. (WT)

Register 16.9. ECDSA_DS_STATE_REG (0x0020)



ECDSA_DS_BUSY Represents the working state of the ECDSA_DS module.

0: IDLE

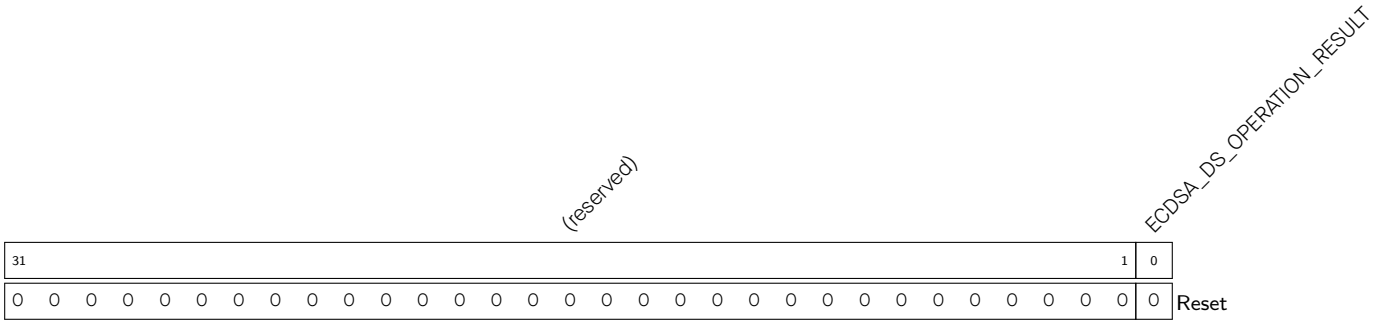
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 16.10. ECDSA_DS_RESULT_REG (0x0024)



ECDSA_DS_OPERATION_RESULT Indicates if the ECDSA_DS operation is successful.

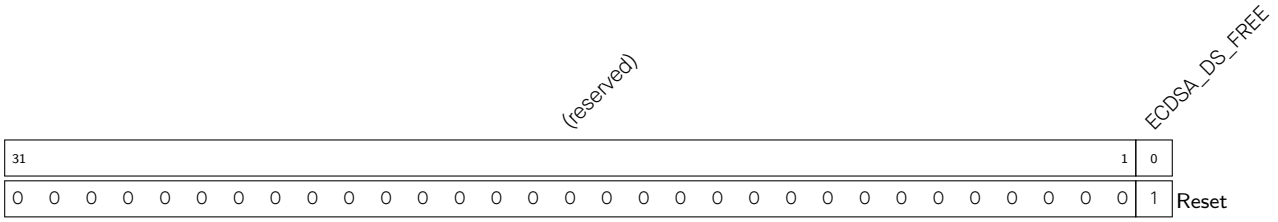
0: Not successful

1: Successful

Only valid when the ECDSA_DS operation is done.

(RO/SS)

Register 16.11. ECDSA_DS_FREE_REG (0x002C)



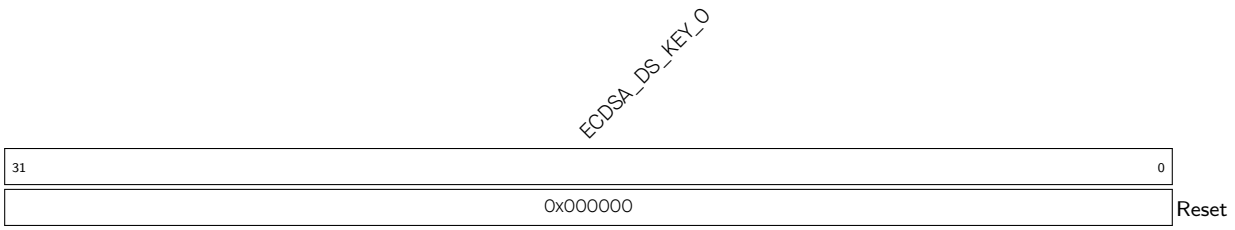
ECDSA_DS_FREE Configures whether ECDSA_DS can be released to another module while it is operating.

0: Cannot be released

1: Can be released

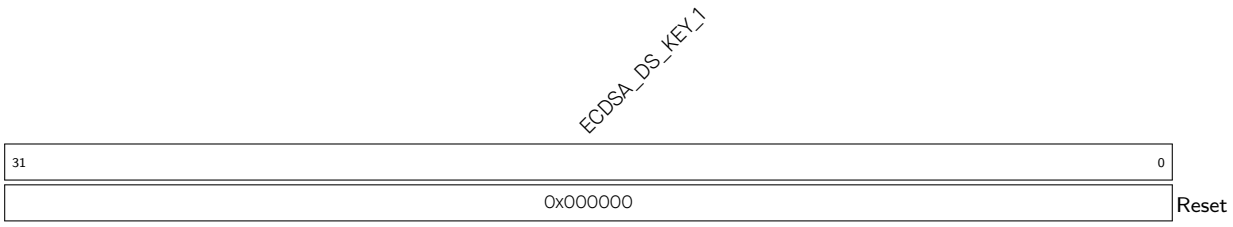
(R/W)

Register 16.12. ECDSA_DS_KEY_0_REG (0x0040)



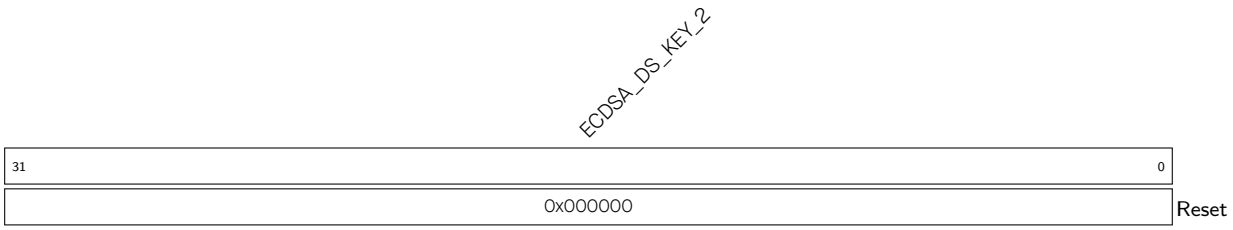
ECDSA_DS_KEY_0 Stores key_0 that is a part of the key material. (R/W)

Register 16.13. ECDSA_DS_KEY_1_REG (0x0044)



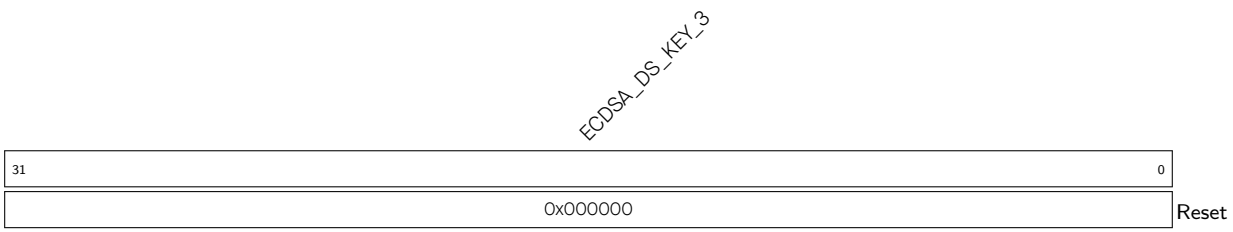
ECDSA_DS_KEY_1 Stores key_1 that is a part of the key material. (R/W)

Register 16.14. ECDSA_DS_KEY_2_REG (0x0048)



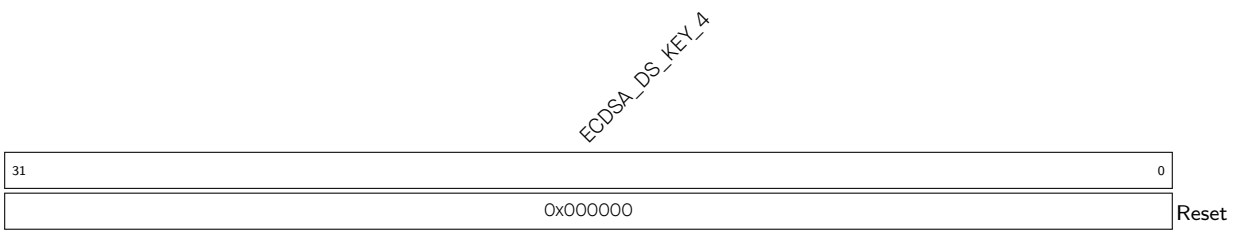
ECDSA_DS_KEY_2 Stores key_2 that is a part of the key material. (R/W)

Register 16.15. ECDSA_DS_KEY_3_REG (0x004C)



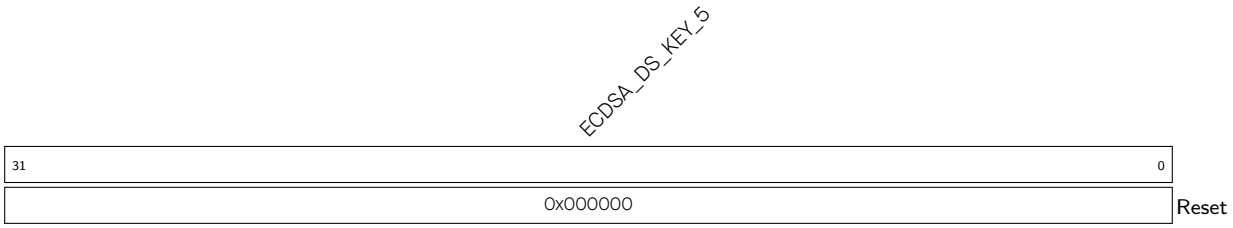
ECDSA_DS_KEY_3 Stores key_3 that is a part of the key material. (R/W)

Register 16.16. ECDSA_DS_KEY_4_REG (0x0050)



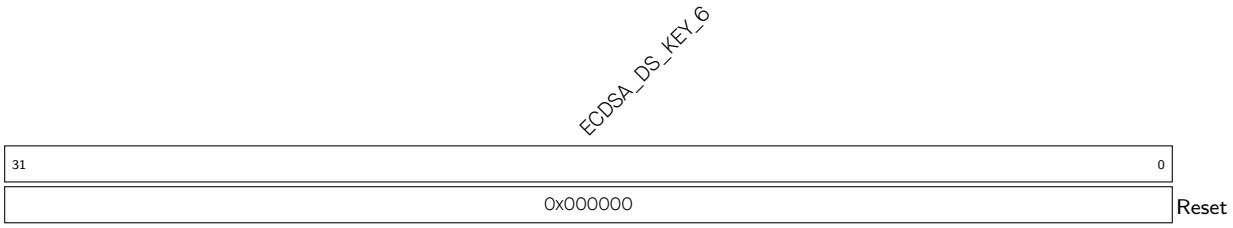
ECDSA_DS_KEY_4 Stores key_4 that is a part of the key material. (R/W)

Register 16.17. ECDSA_DS_KEY_5_REG (0x0054)



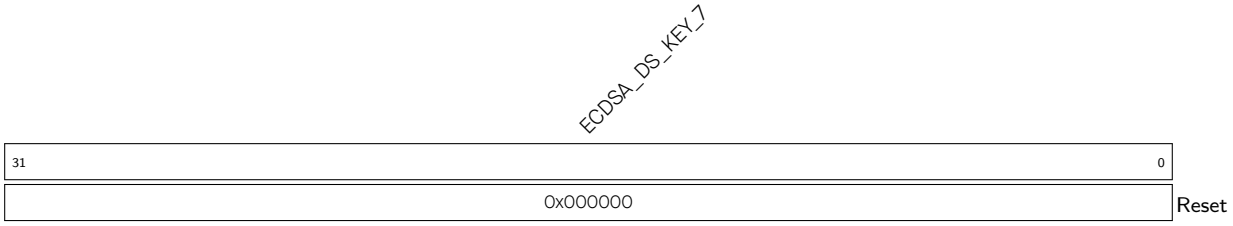
ECDSA_DS_KEY_5 Stores key_5 that is a part of the key material. (R/W)

Register 16.18. ECDSA_DS_KEY_6_REG (0x0058)



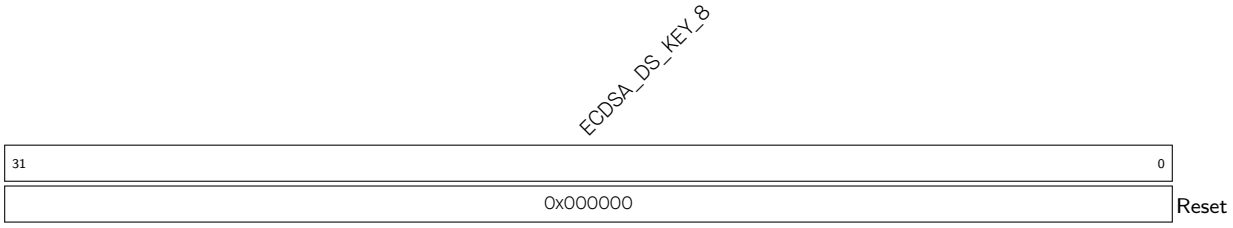
ECDSA_DS_KEY_6 Stores key_6 that is a part of the key material. (R/W)

Register 16.19. ECDSA_DS_KEY_7_REG (0x005C)



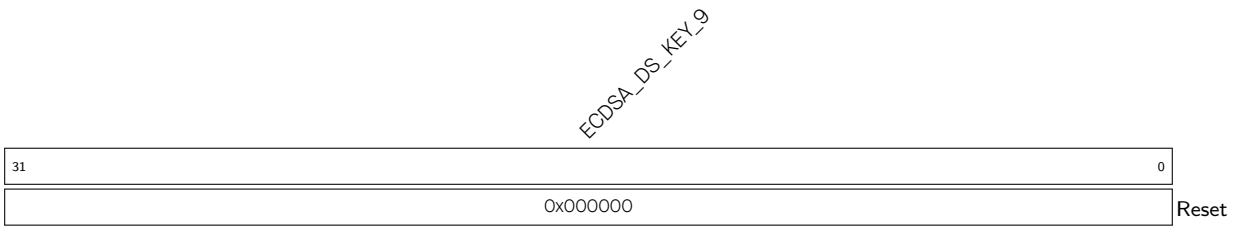
ECDSA_DS_KEY_7 Stores key_7 that is a part of the key material. (R/W)

Register 16.20. ECDSA_DS_KEY_8_REG (0x0060)



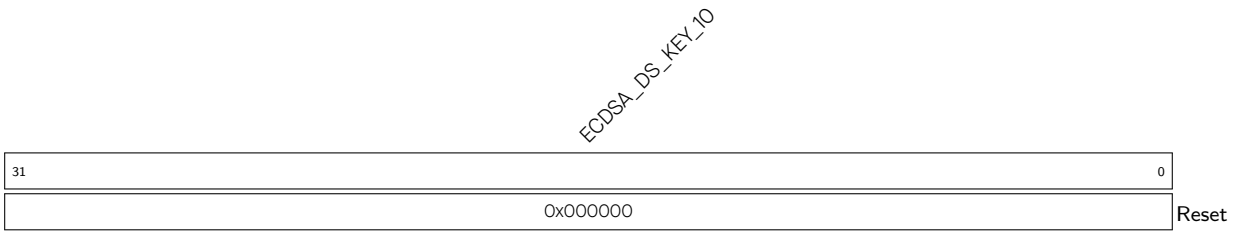
ECDSA_DS_KEY_8 Stores key_8 that is a part of the key material. (R/W)

Register 16.21. ECDSA_DS_KEY_9_REG (0x0064)



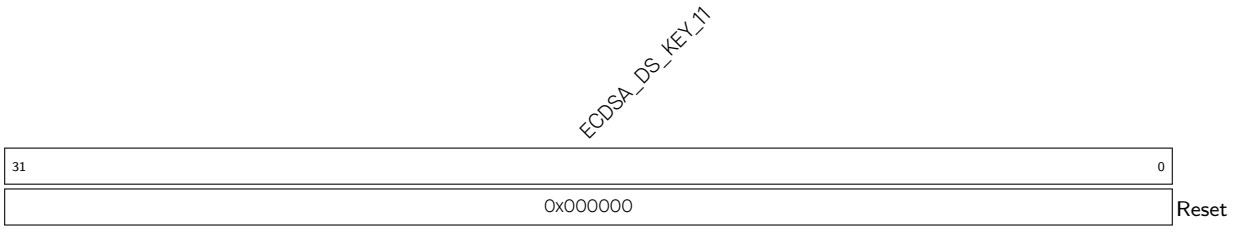
ECDSA_DS_KEY_9 Stores key_9 that is a part of the key material. (R/W)

Register 16.22. ECDSA_DS_KEY_10_REG (0x0068)



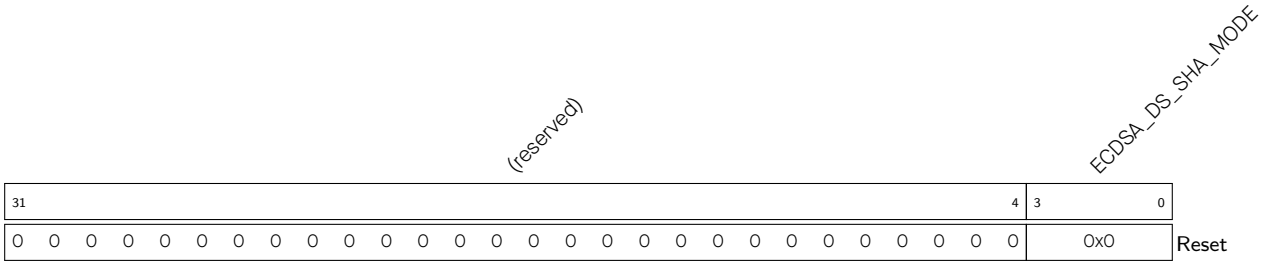
ECDSA_DS_KEY_10 Stores key_10 that is a part of the key material. (R/W)

Register 16.23. ECDSA_DS_KEY_11_REG (0x006C)



ECDSA_DS_KEY_11 Stores key_11 that is a part of the key material. (R/W)

Register 16.24. ECDSA_DS_SHA_MODE_REG (0x0200)



ECDSA_DS_SHA_MODE Configures SHA algorithms for message hash.

1: SHA-224

2: SHA-256

3: SHA-384

4: SHA-512

5: SHA-512-224

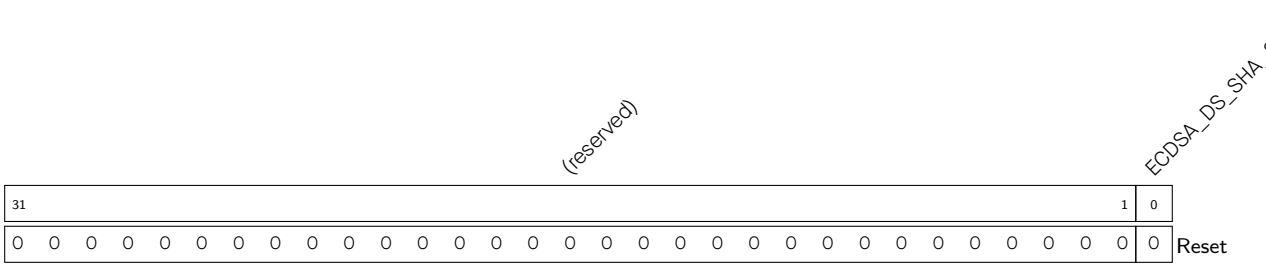
6: SHA-512-256

14: SM3

Others: invalid

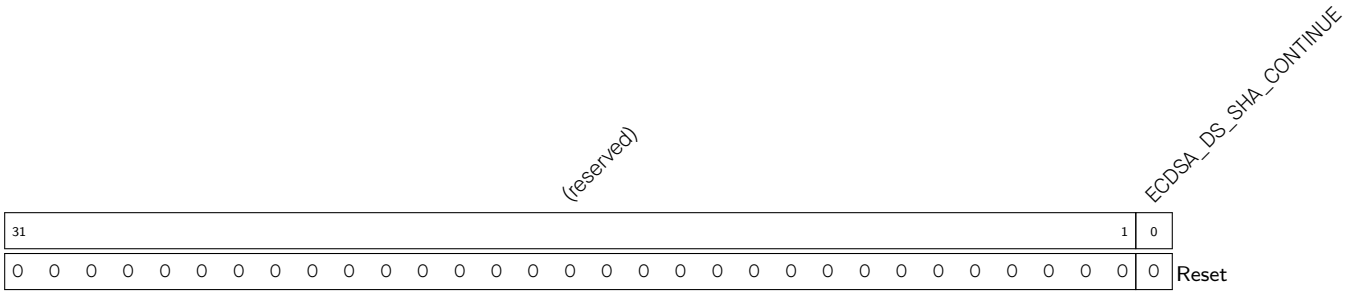
(R/W)

Register 16.25. ECDSA_DS_SHA_START_REG (0x0210)



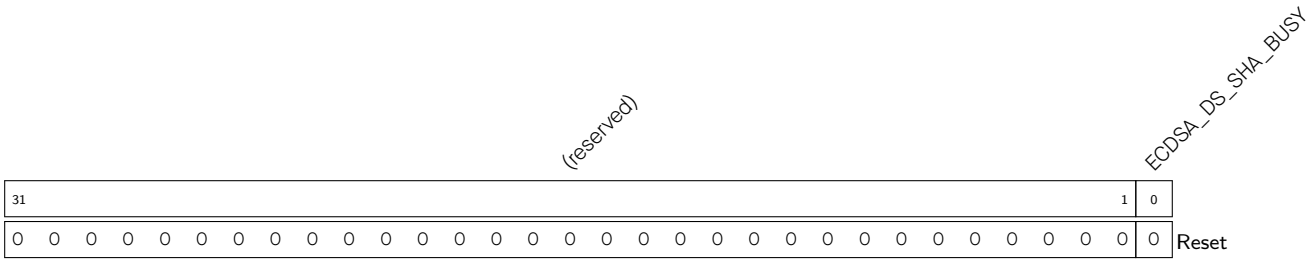
ECDSA_DS_SHA_START Write 1 to start the first SHA operation in the ECDSA_DS process. This bit will be self-cleared after configuration. (WT)

Register 16.26. ECDSA_DS_SHA_CONTINUE_REG (0x0214)



ECDSA_DS_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 16.27. ECDSA_DS_SHA_BUSY_REG (0x0218)



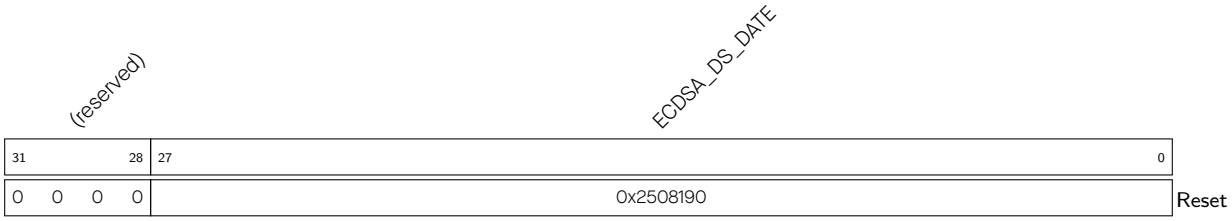
ECDSA_DS_SHA_BUSY Represents the working state of the SHA accelerator in the ECDSA_DS process.

0: IDLE

1: BUSY

(RO)

Register 16.28. ECDSA_DS_DATE_REG (0x00FC)



ECDSA_DS_DATE The ECDSA_DS version control register. (R/W)

Part IV

Image and Voice Processing

Focusing on multimedia processing, this part classifies components related to image and voice processing, including codecs, pixel-processing accelerator, audio sample rate converter, and camera controllers.

Chapter 17

JPEG Codec

ESP32-S31's JPEG codec complies with the JPEG baseline standard and supports both image compression (encoding) and decompression (decoding). By reducing the bandwidth required for image transmission and the storage space required for high-resolution images, the codec makes it practical to work with high-resolution images.

The encoding process consists of the following steps: discrete cosine transform (DCT), zigzag scanning, quantization, and Huffman coding.

The decoding process consists of the following steps: Huffman decoding, inverse quantization, inverse zigzag scanning, and inverse discrete cosine transform (IDCT).

17.1 Terminology

MCU	Minimum coded unit. The basic unit of codec processing
baseline	A particular sequential DCT-based encoding and decoding process specified in the standard, which is required for all DCT-based decoding processes
image	A set of two-dimensional arrays of integer data
component	One of the two-dimensional arrays that comprise an image
data unit	An 8 × 8 block of pixels of one component
marker	A two-byte code in which the first byte is hexadecimal FF (0xFF) and the second byte is a value between 1 and hexadecimal FE (0xFE). It is served to identify the various structural parts of the compressed data formats
restart marker	Used to isolate coded data segments and for application-specific uses, such as decoder re-synchronization and error recovery
restart interval	The integer number of MCUs processed between two restart markers
scan	A single pass through the data for one or more of the components in an image

Figure 17.1-1 *JPEG Bitstream Format* shows the composition of a JPEG bitstream. In this figure, dashed boxes indicate the structure of the corresponding bitstream segments, and orange blocks indicate marker codes.

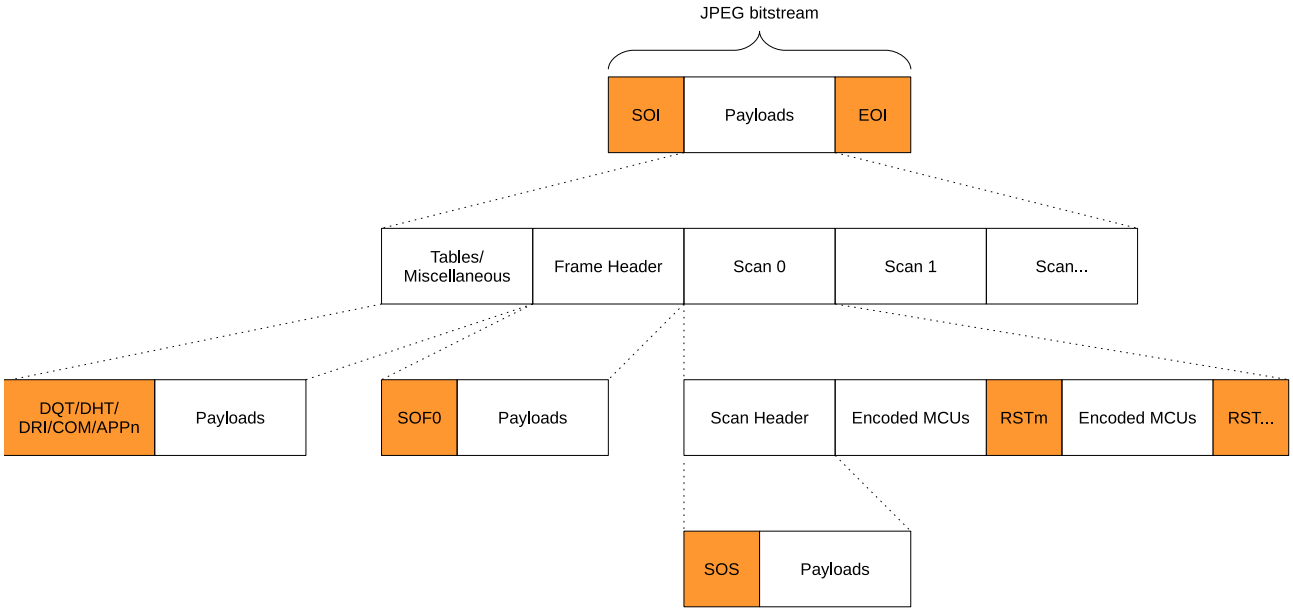


Figure 17.1-1. JPEG Bitstream Format

The marker code assignments are shown in Table 17.1-2 *Marker Code Assignments*.

Table 17.1-2. Marker Code Assignments

Code Assignment	Marker	Description
0xFFC0	SOF0	Start of frame (Baseline DCT)
0xFFC4	DHT	Define Huffman table(s)
0xFFDm	RSTm	Restart with modulo 8 count “m”
0xFFD8	SOI	Start of image
0xFFD9	EOI	End of image
0xFFDA	SOS	Start of scan
0xFFDB	DQT	Define quantization table(s)
0xFFDD	DRI	Define restart interval
0xFFEn	APPn	Reserved for application segments, “n” is 0 to F
0xFFFE	COM	Comment

17.2 Overview

ESP32-S31's JPEG codec is based on the JPEG baseline standard. The standard specifies that encoding is completed through one or more scans, and each scan can process one or more minimum coded units (MCUs). An MCU consists of a sequence of data units; the sequence depends on the sampling factors of each image component. MCUs of different image formats may contain different data unit sequences, as shown in Table 17.2-1 *Number of Data Units in the MCU for Different Image Formats*.

Table 17.2-1. Number of Data Units in the MCU for Different Image Formats

Image Format	MCU[1]		
	Luminance Component Y	Chrominance Component U	Chrominance Component V

YUV444	1 × 1	1 × 1	1 × 1
YUV422	2 × 1	1 × 1	1 × 1
YUV420	2 × 2	1 × 1	1 × 1
GRAY	1 × 1	None	None

[1] The MCU of each image format can be represented as: number of data units in the horizontal direction × number of data units in the vertical direction, for each component.

At any given time, the JPEG codec can be configured as either an encoder or a decoder, but not both simultaneously:

- When used as an encoder, it receives the image to be compressed from 2D DMA and sends the encoded bitstream to 2D DMA.
- When used as a decoder, it receives the coded bitstream from 2D DMA and sends the decoded image to 2D DMA.

The JPEG codec workflow is shown in Figure 17.2-1 *JPEG Codec System Connection Block Diagram*.

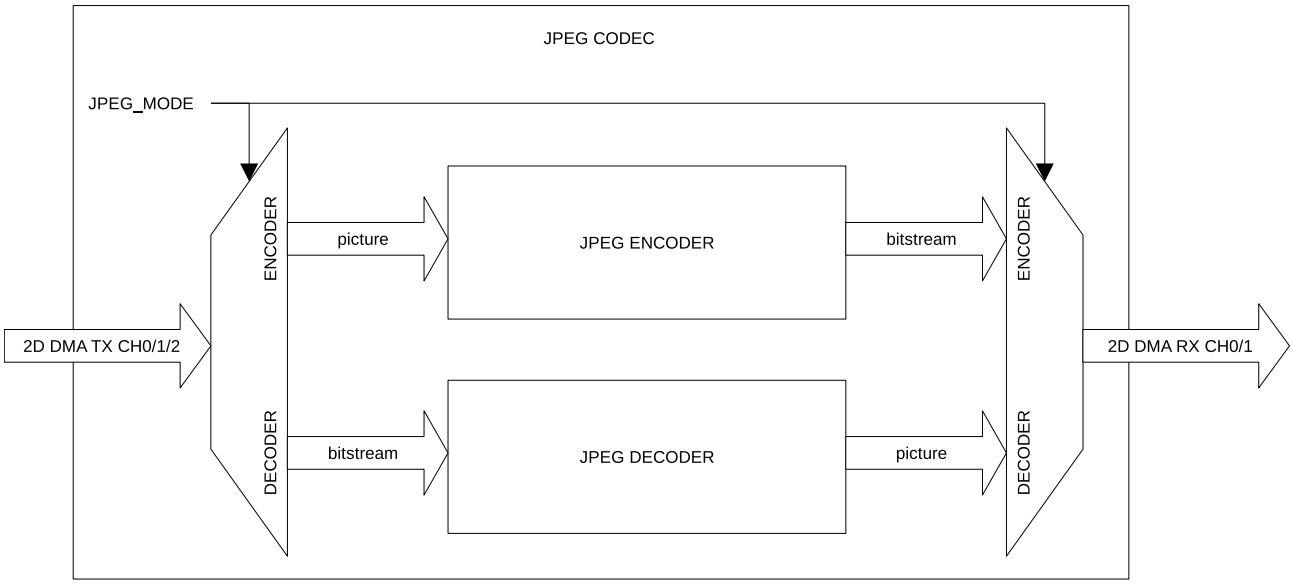


Figure 17.2-1. JPEG Codec System Connection Block Diagram

The function clock of the JPEG codec is SYS_CLK.

17.3 Feature List

When used as an encoder, the JPEG codec has the following features:

- Pause during encoding
- Discrete cosine transform (DCT) algorithm
- Canonical Huffman coding
- Supported input image formats:
  - YUV444, YUV422, YUV420, and GRAY

  - RGB888 and RGB565 (after conversion to YUV444, YUV422, or YUV420)
- Four configurable quantization coefficient tables with 8-bit or 12-bit precision
- Performance:
  - Still image compression: up to 4K resolution
  - Dynamic image compression: up to 720P@30 fps (excluding header encoding time)
- Automatic zero-byte stuffing
- Automatic EOI marker insertion
- Timeout detection

When used as a decoder, the JPEG codec has the following features:

- Inverse discrete cosine transform (IDCT) algorithm
- Huffman decoding
- Compressed bitstream decoding for YUV444, YUV422, YUV420, and GRAY image formats
- RST marker parsing
- Four configurable quantization coefficient tables with 8-bit or 12-bit precision
- Two DC and two AC Huffman tables
- Image decoding at any resolution. The output resolution is rounded up according to the input image format and may differ from the original resolution in the bitstream:
  - YUV444, GRAY: both horizontal and vertical resolutions are rounded up to multiples of 8. For example, a 150 × 150 image has an output resolution of 152 × 152
  - YUV422: horizontal resolution is rounded up to a multiple of 16, and vertical resolution to a multiple of 8. For example, a 150 × 150 image has an output resolution of 160 × 152
  - YUV420: both horizontal and vertical resolutions are rounded up to multiples of 16. For example, a 150 × 150 image has an output resolution of 160 × 160
- Timeout detection
- Performance:
  - Still image decoding: up to 4K resolution
  - Dynamic image decoding: up to 720P@30 fps (excluding header parsing time)

17.4 Architectural Overview

When the JPEG codec is configured as an encoder, its architecture is shown in Figure 17.4-1 *JPEG Encoder Architecture*.

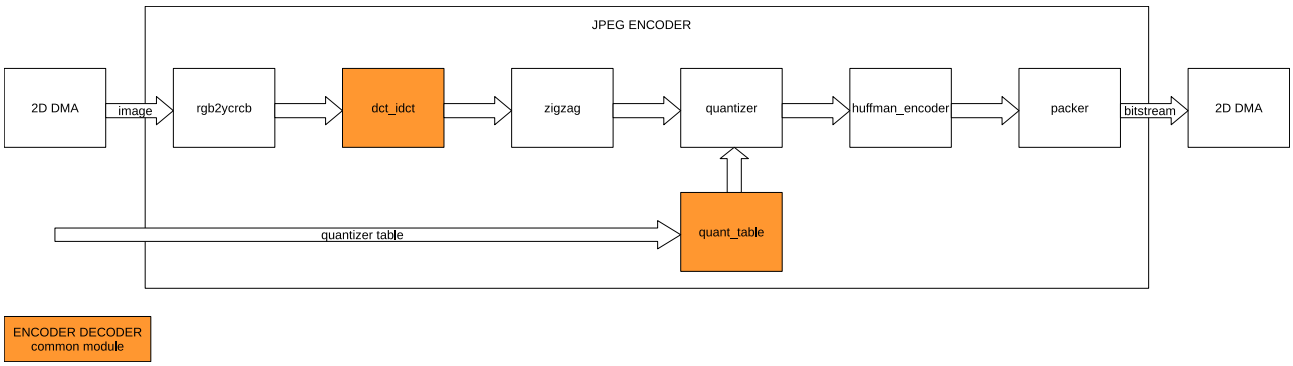


Figure 17.4-1. JPEG Encoder Architecture

- **rgb2ycrcb** implements color space conversion and converts the original image to a compressible image format supported by the codec
- **dct_idct** performs two-dimensional DCT on data units
- **zigzag** performs zigzag scanning and sorting on DCT-transformed data units
- **quant_table** stores the quantization coefficient table delivered by the software
- **quantizer** quantizes zigzag-sorted data based on the quantization coefficient table stored in **quant_table**
- **huffman_encoder** performs Huffman encoding on quantized data units
- **packer** packs Huffman-encoded data in groups of 32 bits and outputs the encoded bitstream

When the JPEG codec is configured as a decoder, its architecture is shown in Figure 17.4-2 *JPEG Decoder Architecture*.

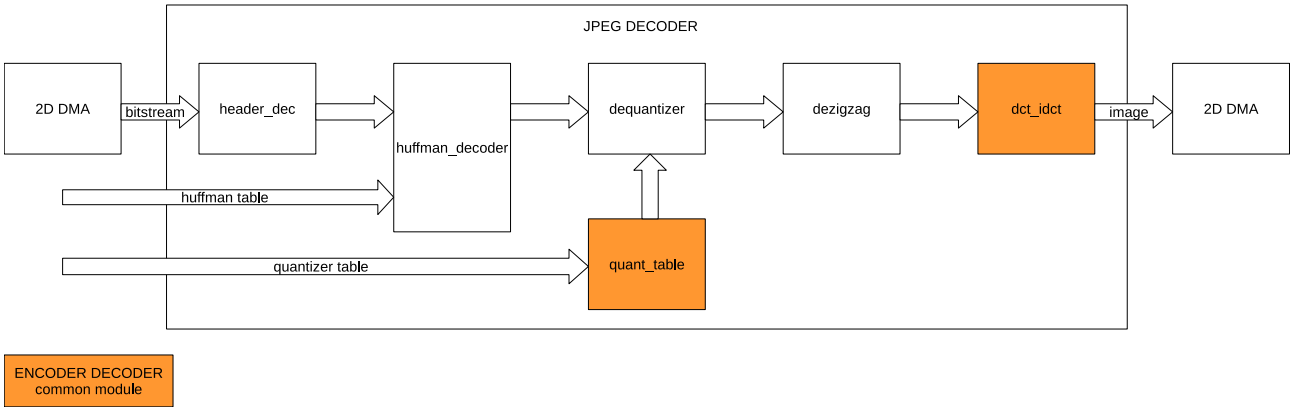


Figure 17.4-2. JPEG Decoder Architecture

- **header_dec** extracts the bitstream, parses SOS and EOI markers, and controls the decoding process
- **huffman_decoder** stores the Huffman table delivered by the software, and performs Huffman decoding on the bitstream
- **quant_table** stores the quantization coefficient table delivered by the software
- **dequantizer** dequantizes Huffman-decoded data based on the quantization coefficient table stored in **quant_table**
- **dezigzag** performs inverse zigzag scanning and sorting on dequantized data, and restores data units

- **dct_idct** performs two-dimensional inverse DCT on dezigzag-sorted data units, and outputs decoded data units

Since the JPEG codec cannot be used as an encoder and decoder at the same time, the encoder and decoder share the **quant_table** and **dct_idct** modules to save hardware resources.

17.5 Functional Description

The JPEG codec operates as either an encoder or a decoder.

17.5.1 JPEG Encoder

17.5.1.1 Pause

The JPEG encoder can be paused when higher-priority tasks are executed. This feature is controlled by the JPEG_PAUSE_EN bit.

- When JPEG_PAUSE_EN is set to 1, the JPEG encoder stops outputting the bitstream to the 2D DMA RX channel.
- When JPEG_PAUSE_EN is cleared to 0, the JPEG encoder resumes and outputs the bitstream normally.

17.5.1.2 Color Space Conversion

According to the JPEG standard, the image formats supported by the JPEG encoder are YUV444, YUV422, YUV420, and GRAY. With integrated color space conversion, ESP32-S31's JPEG encoder converts RGB888 and RGB565 formats to these encoder-supported formats before encoding, so it supports a wider set of input formats.

The color space conversion formulas for converting RGB to YUV are:

$$Y = (0.299 \times R) + (0.587 \times G) + (0.114 \times B)$$
$$U = (-0.1687 \times R) + (-0.3313 \times G) + (0.5 \times B) + 128$$
$$V = (0.5 \times R) + (-0.4187 \times G) + (-0.0813 \times B) + 128$$

The image format before conversion is represented by its format and pixel order. Table 17.5-1 *Supported Image Formats for Color Space Conversion* lists the image formats supported for color space conversion and their representations:

Table 17.5-1. Supported Image Formats for Color Space Conversion

JPEG_ COLOR_SPACE	Format[1]	JPEG_ PIXEL_REV	Pixel Order
0	RGB888	0	RGB
		1	BGR
1	YUV422[2]	0	YUYV
		1	UYVY
2	RGB565	0	RGB
		1	BGR
3	GRAY[2]	N/A	YYY
4	YUV444	0	YUV
		1	VUY
5	YUV420[2]	N/A	YYU/YYV[3]

[1] Configure JPEG_SAMPLE_SEL to convert the image format to another format supported by the encoder:

- 0: YUV444
- 1: YUV422
- 2: YUV420

[2] Image format conversion has the following limits:

- YUV422: can remain unchanged or be downsampled to YUV420. Setting JPEG_SAMPLE_SEL to 0 has no effect.
- YUV420 or GRAY: is not converted, regardless of the value of JPEG_SAMPLE_SEL.

[3] Odd-numbered image rows are YYU, even-numbered image rows are YYV.

17.5.1.3 Configurable Quantization Coefficient Table

ESP32-S31's JPEG encoder hardware supports four 8 × 8 quantization coefficient tables, each of which can be configured with 64 coefficient values via software.

The precision of each quantization table is configured via JPEG_QNR_PRECISION:

- 0: lower 8 bits are valid
- 1: lower 12 bits are valid

The configuration mode for quantization table coefficients is selected via JPEG_QNR_FIFO_EN:

- 0: non-FIFO mode; each coefficient is written to a specific address
- 1: FIFO mode; all coefficients are written to the same address

Table 17.5-2. Position Number of Coefficients in a Quantization Coefficient Table (Example)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

...							
56	57	58	59	60	61	62	63

The configuration method differs between modes. Note that *n* (*n* = 0, 1, 2, 3) is the index of each quantization coefficient table.

- Non-FIFO mode: all quantization coefficients in quantization coefficient table *n* must be written by position to a specific address, i.e., (JPEG codec base address + *n* * 0x100 + *quantization coefficient position number* * 0x4), where *quantization coefficient position number* can be 0, 1, 2, ..., 63 (see Table 17.5-2 *Position Number of Coefficients in a Quantization Coefficient Table (Example)*). In this mode, since each coefficient is written to a specific address, the 64 quantization coefficients do not need to be configured in sequence.
- FIFO mode: all 64 quantization coefficients in quantization coefficient table *n* are written to the same address, i.e., JPEG_T*n*QNR_REG. In this mode, since all coefficients are written to the same address, the 64 quantization coefficients must be configured in sequence. The hardware then automatically stores the 64 quantization coefficients in the order shown in Table 17.5-2 *Position Number of Coefficients in a Quantization Coefficient Table (Example)*.

If fewer than four quantization tables are used, the unused tables do not need to be configured.

To use quantization coefficient table *n* for the luminance or chrominance component, set the corresponding register field to *n*:

- Luminance component: JPEG_LQNR_TBL_SEL
- Chrominance component: JPEG_CQNR_TBL_SEL

Tables 17.5-3 *Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component* and 17.5-4 *Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component* show the 8-bit precision quantization coefficient tables recommended by the JPEG baseline standard for the luminance and chrominance components, respectively.

Table 17.5-3. Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 17.5-4. Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99

47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

See ITU T.81 Recommendation for more information. Adjusting the quantization coefficient table changes the compression ratio (the ratio of uncompressed image size to compressed image size) and encoded image quality. The larger the quantization coefficient, the higher the compression ratio and the lower the image quality.

17.5.1.4 Stuffed Zero Byte

The JPEG baseline standard requires that no marker appear within a Huffman-coded segment. Each 0xFF byte generated by Huffman coding must be followed by a stuffed 0x00 byte.

The JPEG encoder implements this feature and can automatically stuff a 0x00 byte after each 0xFF byte in the Huffman-coded segment of the bitstream.

Whether to add a 0x00 byte after each 0xFF byte is configured via JPEG_FF_CHECK_EN:

- 0: do not add
- 1: add (default)

17.5.1.5 EOI Marker

The end-of-image (EOI) marker marks the end of a compressed image bitstream. The EOI marker code is 0xFFD9.

The JPEG encoder can automatically add the EOI marker at the end of the bitstream.

Whether to add the EOI marker is configured via JPEG_TAILER_EN:

- 0: do not add
- 1: add

17.5.1.6 Timeout Detection

The JPEG codec supports timeout detection when used as an encoder. The hardware triggers the timeout interrupt JPEG_ENCODE_TIMEOUT_INT when:

- the encoder cannot write the encoded bitstream to 2D DMA for more than $(2^{JPEG\_ENCODE\_TIMEOUT\_THRES} - 1)$ clock cycles

17.5.2 JPEG Decoder

The information in the bitstream must first be parsed by software, and then sent to the hardware through the APB configuration bus. Note that the hardware does not support parsing any marker segments other than SOS, RST, and EOI marker segments.

17.5.2.1 Multiple Components

The frame header contains the number of components, the ID of each component, and the horizontal and vertical sampling factors of each component.

The JPEG decoder software must parse this information and write it to the corresponding registers JPEG_COMPONENT_NUM, JPEG_C*n*_ID, JPEG_C*n*_X_FACTOR, and JPEG_C*n*_Y_FACTOR. Here, *n* can be 0, 1, or 2, indicating that the hardware supports up to three components (for example, Y, U, and V in YUV). If fewer than three components are used, the registers corresponding to the unused components do not need to be configured.

The hardware checks whether the value of each component register JPEG_C*n*_ID matches the ID in the scan header information. The sampling factor indicates the number of data units of that component contained in one MCU in the horizontal and vertical directions. Based on the values of JPEG_COMPONENT_NUM, JPEG_C*n*_X_FACTOR, and JPEG_C*n*_Y_FACTOR, the hardware determines the proportional relationship of each component, that is, the format of the image to be decoded. See Table 17.5-5 *Sampling Factor and Image Format*.

Table 17.5-5. Sampling Factor and Image Format

JPEG_COMPONENT_NUM	JPEG_C*n*_X/Y_FACTOR	Format of the Image to be Decoded
0	-	invalid
1	C0_X_FACTOR is 1 C0_Y_FACTOR is 1	GRAY
2	-	invalid
3	C0_X_FACTOR is 1 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV444
	C0_X_FACTOR is 2 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV422
	C0_X_FACTOR is 2 C0_Y_FACTOR is 2 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV420

Regardless of the format of the image to be decoded, the decoder supports only one scan that contains all components. After decoding, the image is sent to 2D DMA. For the layout of each decoded image format, see Chapter 4 *2D-DMA Controller (2D-DMA)*.

17.5.2.2 Parsing RST Marker

The bitstream may have a restart (RST) marker in the scan segment. The RST marker code is 0xFFD*x* (*x* = 0, 1, ..., 7).

The JPEG decoder hardware can parse RST markers and check whether the actual number of MCUs between two RST markers matches the restart interval defined in the DRI segment of the Tables/Miscellaneous segment.

First, software parses the DRI segment and writes the restart interval to JPEG_RESTART_INTERVAL. Then, after every JPEG_RESTART_INTERVAL MCUs are decoded, the hardware checks whether the subsequent bitstream contains an RST marker.

If JPEG_RESTART_INTERVAL is 0, the bitstream contains no RST markers and the hardware does not check for them.

17.5.2.3 Configurable Quantization Coefficient Table

ESP32-S31's JPEG decoder also supports four configurable quantization coefficient tables. Since the JPEG codec cannot be used as an encoder and decoder at the same time, the decoder can share the same four configurable coefficient tables with the encoder to save hardware resources.

The decoder quantization coefficient table configuration is similar to that of the encoder (see Section 17.5.1.3 *Configurable Quantization Coefficient Table*), with the following differences:

- The software needs to parse the DQT segment in the Tables/Miscellaneous segment to obtain the quantization coefficient tables of the decoder. The recommended quantization coefficient tables are the same as the encoder shown in Table 17.5-3 *Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component* and Table 17.5-4 *Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component*.
- The precision of quantization coefficient table *n* is configured via bit[4] of JPEG_T*n*_DQT_INFO:
  - 0: lower 8 bits are valid
  - 1: higher 12 bits are valid

  The precision is also obtained from the DQT segment in the Tables/Miscellaneous segment.
- Each component can select a different quantization coefficient table. The lower 4 bits of JPEG_T*n*_DQT_INFO configure the quantization coefficient table ID, and JPEG_C*n*_DQT_TBL_SEL configures the quantization coefficient table selected for each component. The ID of each quantization coefficient table is also obtained from the DQT segment in the Tables/Miscellaneous segment.

17.5.2.4 Configurable Huffman Table

ESP32-S31 supports up to two DC Huffman tables and two AC Huffman tables, which can be configured as needed within the hardware limits.

Each Huffman table contains three types of information, which can be obtained from the DHT segment in the Tables/Miscellaneous segment:

- the number of codewords with lengths from 1 to 16 bits
- the symbols corresponding to decoded codewords

  - DC table: up to 16 symbols
  - AC table: up to 256 symbols
- the minimum codewords with lengths from 1 to 16 bits

DC and AC Huffman tables are configured in FIFO or non-FIFO mode via JPEG_DHT_FIFO_EN:

- 0: non-FIFO mode
- 1: FIFO mode

In different modes, the Huffman table configuration differs. The following describes DC table *n* (*n* = 0, 1):

- Non-FIFO mode: write each type of information for DC*n* to its type-specific address.
  - Number of codewords with lengths from 1 to 16 bits: JPEG codec base address + 0x500 + *n* * 0x80 + (*codeword length* - 1) * 0x4, where *codeword length* = 1, 2, ..., 16.
  - Symbols corresponding to decoded codewords: JPEG codec base address + 0x600 + *n* * 0x40 + *symbol number* * 0x4, where *symbol number* = 0, 1, ..., 15.
  - Minimum codewords with lengths from 1 to 16 bits: JPEG codec base address + 0xF00 + *n* * 0x80 + (*codeword length* - 1) * 0x4, where *codeword length* = 1, 2, ..., 16.

  In this mode, information of each type does not need to be written in sequence.
- FIFO mode: write all information of the same type for DC*n* to the same address.
  - Number of codewords with lengths from 1 to 16 bits: JPEG_DHT_TOTLEN_DC*n*_REG.
  - All symbols corresponding to decoded codewords (DC table: up to 16 symbols): JPEG_DHT_VAL_DC*n*_REG.
  - Minimum codewords with lengths from 1 to 16 bits: JPEG_DHT_CODEMIN_DC*n*_REG.

  In this mode, information of each type must be written in sequence. The hardware then stores it in order.

The AC*n* (*n* = 0, 1) table configuration is similar to that of DC*n* (*n* = 0, 1):

- Non-FIFO mode: write each type of information for AC*n* to its type-specific address.
  - Number of codewords with lengths from 1 to 16 bits: JPEG codec base address + 0x540 + *n* * 0x80 + (*codeword length* - 1) * 0x4, where *codeword length* = 1, 2, ..., 16.
  - Symbols corresponding to decoded codewords: JPEG codec base address + 0x680 + *n* * 0x400 + *symbol number* * 0x4, where *symbol number* = 0, 1, ..., 255.
  - Minimum codewords with lengths from 1 to 16 bits: JPEG codec base address + 0xF40 + *n* * 0x80 + (*codeword length* - 1) * 0x4, where *codeword length* = 1, 2, ..., 16.

  In this mode, information of each type does not need to be written in sequence.
- FIFO mode: write all information of the same type for AC*n* to the same address.
  - Number of codewords with lengths from 1 to 16 bits: JPEG_DHT_TOTLEN_AC*n*_REG.
  - All symbols corresponding to decoded codewords (AC table: up to 256 symbols): JPEG_DHT_VAL_AC*n*_REG.
  - Minimum codewords with lengths from 1 to 16 bits: JPEG_DHT_CODEMIN_AC*n*_REG.

In this mode, information of each type must be written in sequence. The hardware then stores it in order.

Minimum codewords with lengths from 1 to 16 bits must be left-aligned in the 16-bit configuration value. If no codeword exists for a given length, set the configuration value to 0xFFFF. For example, if the minimum codeword is 0x7C, the 16-bit configuration value is 0x7C00.

Recommended DC and AC Huffman tables for the JPEG baseline standard are in ITU T.81 Recommendation.

The software obtains the IDs of the DC*n* and AC*n* Huffman tables by parsing the DHT segment in the Tables/Miscellaneous segment, and writes the IDs to the corresponding fields JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

The hardware automatically uses the DC*n* and AC*n* Huffman tables whose IDs match JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

17.5.2.5 Timeout Detection

The JPEG codec supports timeout detection when used as a decoder. The hardware triggers the timeout interrupt JPEG_DECODE_TIMEOUT_INT when:

- the decoder stops reading the bitstream for more than $(2^{JPEG\_DECODE\_TIMEOUT\_THRES} - 1)$ clock cycles

After the timeout interrupt is triggered, the decoder is reset according to the configuration of JPEG_DECODE_TIMEOUT_TASK_SEL:

- 0: the hardware takes no action; software must set JPEG_SOFT_RST or JPEG_FSM_RST to perform a reset.
- 1: the hardware automatically resets the decoder, exits the timeout state, and returns to the initial state.

17.6 Interrupts

ESP32-S31's JPEG Codec can generate the JPEG_INTR interrupt signal that will be sent to the Interrupt Matrix.

Interrupt signal JPEG_INTR can be generated by the following internal interrupt sources:

- JPEG_DONE_INT: Triggered when a frame of image encoding or decoding ends.
- JPEG_RLE_PARALLEL_ERR_INT: Triggered when a run-length encoding error occurs.
- JPEG_CID_ERR_INT: Triggered when the decoded component ID is different from the component ID configured by the software.
- JPEG_C_DHT_DC_ID_ERR_INT: Triggered when the decoded DC Huffman table ID of each component is not the software configured DC0 Huffman table ID or DC1 Huffman table ID.
- JPEG_C_DHT_AC_ID_ERR_INT: Triggered when the decoded AC Huffman table ID of each component is not the software configured AC0 Huffman table ID or AC1 Huffman table ID.
- JPEG_C_DQT_ID_ERR_INT: Triggered when the decoded quantization table ID of each component is different from the software configured quantization table ID.

- JPEG_RST_UXP_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is 0 but the RST marker is parsed by the decoder.
- JPEG_RST_CHECK_NONE_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is non-0 but the RST marker cannot be parsed by the decoder.
- JPEG_RST_CHECK_POS_ERR_INT: Triggered when the MCU number between two parsed RST markers is not equal to the JPEG_RESTART_INTERVAL configured by the software.
- JPEG_OUT_EOF_INT: Triggered when the EOF marker is read from the TX channel of 2D DMA.
- JPEG_SR_COLOR_MODE_ERR_INT: This interrupt is invalid.
- JPEG_DCT_DONE_INT: Triggered when the DCT or IDCT calculation of one data unit is completed.
- JPEG_BS_LAST_BLOCK_EOF_INT: Triggered when the encoding of the last data unit is completed.
- JPEG_SCAN_CHECK_NONE_ERR_INT: Triggered when an image frame has multiple scans to be decoded and the SOS marker is not parsed within (JPEG_SOS_CHECK_BYTE_NUM + 1) bytes in any scan header information.
- JPEG_SCAN_CHECK_POS_ERR_INT: Triggered when the position of the scan header information parsed by the decoder is wrong.
- JPEG_UXP_DET_INT: Triggered when the marker parsed by the decoder is not supported by the hardware.
- JPEG_EN_FRAME_EOF_ERR_INT: Triggered when the number of data units to be encoded read from 2D DMA is less than the number of data units calculated based on the image resolution configured by the software.
- JPEG_EN_FRAME_EOF_LACK_INT: Triggered when a frame of image to be encoded is completely read from 2D DMA but the EOF marker is not read.
- JPEG_DE_FRAME_EOF_ERR_INT: Triggered when the number of data units obtained after decoding a frame of image is different from the number of data units calculated based on the image resolution configured by the software.
- JPEG_DE_FRAME_EOF_LACK_INT: Triggered when the bitstream of an image is completely read from 2D DMA but neither the EOF marker nor the EOI marker is read.
- JPEG_SOS_UNMATCH_ERR_INT: Triggered when either of the following occurs in the scan header information parsed by the decoder:
  - The number of components is 0
  - The header length does not match the actual header length
- JPEG_MARKER_ERR_FST_SCAN_INT: Triggered when there is an error in the first scan header information parsed by the decoder. The errors include those that trigger:
  - JPEG_CID_ERR_INT
  - JPEG_C_DHT_DC_ID_ERR_INT
  - JPEG_C_DHT_AC_ID_ERR_INT
  - JPEG_C_DQT_ID_ERR_INT

  - JPEG_SOS_UNMATCH_ERR_INT

- JPEG_MARKER_ERR_OTHER_SCAN_INT: Triggered when there is an error in a non-first scan header information parsed by the decoder. The error types are the same as those that trigger JPEG_MARKER_ERR_FST_SCAN_INT.
- JPEG_UNDET_INT: Triggered when the bitstream of an image is completely read from 2D DMA but the SOS marker is not read.
- JPEG_DECODE_TIMEOUT_INT: Triggered when the decoder times out.
- JPEG_ENCODE_TIMEOUT_INT: Triggered when the encoder times out.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 17.8 *Register Summary*.

17.7 Programming Procedures

17.7.1 JPEG Encoder

When the JPEG codec is working as an encoder, the configuration process is:

1. Activate the JPEG codec:
   - Set HP_SYS_CLKRST_JPEG_SYS_CLK_EN to 1 to enable the JPEG codec clock.
   - Set HP_SYS_CLKRST_RST_EN_JPEG to 0 to release the JPEG codec system-level reset.
2. Configure the JPEG codec:
   - Set JPEG_SOFT_RST to 1 to reset the JPEG codec, and then set JPEG_SOFT_RST to 0 to release the JPEG codec reset.
   - Set JPEG_MODE to 0 to configure the JPEG codec to be used as an encoder.
   - Configure the following fields to select the original image format:
     - Format: JPEG_COLOR_SPACE
     - Pixel order: JPEG_PIXEL_REV

     See Table 17.5-1 *Supported Image Formats for Color Space Conversion*.
   - Configure JPEG_TAILER_EN to select whether to add the EOI marker (0xFFD9) at the end of the bitstream:
     - 0: disable
     - 1: enable

- Configure JPEG_SAMPLE_SEL to select the image format to be compressed:
  - 0: YUV444
  - 1: YUV422
  - 2: YUV420

  Note that this field is valid only when the original image format is RGB. See Table 17.5-1 *Supported Image Formats for Color Space Conversion*.
- Configure JPEG_HA as the original image width and JPEG_VA as the original image height.
- Configure the quantization coefficient table by following the instructions in Section 17.5.1.3 *Configurable Quantization Coefficient Table*.
- Configure timeout detection by following the instructions in Section 17.5.1.6 *Timeout Detection*.

3. Configure 2D DMA (for configuration details and related concepts, see Chapter 4 *2D-DMA Controller (2D-DMA)*.)

   (a) Configure the linked list of 2D DMA TX channel.
   - Set 2DEN to 1 to enable 2D function.
   - Configure the Buffer address pointer as the starting address where the original image is stored.
   - Configure pbyte to select the number of bytes of one pixel of the original image.
   - Configure HA as the width of the original image and VA as the height of the original image.
   - Configure hb as the number of pixels in the horizontal direction of an image block and vb as the number of pixels in the vertical direction of an image block.
   - Configure mod as 1, which means continuous reading of image blocks of hb × vb size.
   - Configure (X, Y) as the coordinate of the first pixel in the starting image block. According to the original image format and the image format to be compressed, hb and vb must meet the restrictions in Table 17.7-1 *JPEG Encoder hb and vb Configuration*:

Table 17.7-1. JPEG Encoder hb and vb Configuration

Format of Original Image	Format of the Image to be Compressed	hb	vb
RGB	YUV444	Multiples of 8	Multiples of 8
RGB	YUV422	Multiples of 16	Multiples of 8
RGB	YUV420	Multiples of 16	Multiples of 16
YUV422	YUV422	Multiples of 16	Multiples of 8
GRAY	GRAY	Multiples of 8	Multiples of 8

   (b) Configure the linked list of the 2D DMA RX channel:
   - Set 2DEN to 0 to enable 1D function.
   - Configure the Buffer address pointer as the starting address where the encoded bitstream will be stored.
   - Configure size as the number of bytes in the space to store the encoded bitstream.

- Configure mod as 0. mod is invalid when 2DEN is 0.

(c) Connect 2D DMA channels to JPEG codec:

- Configure DMA2D_OUT_PERI_SEL_CH*n* to 0 to connect the TX channel*n* to the JPEG codec.
- Configure DMA2D_IN_PERI_SEL_CH*n* to 0 to connect the RX channel*n* to the JPEG codec.

(d) Reset 2D DMA channels:

- 2D DMA TX channel*n*: Set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.
- 2D DMA RX channel*n*: Set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*.
- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*.

(f) Select the 2D DMA output image block size by configuring DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* according to the format of original image and the format of the image to be compressed. See details in Table 17.7-2 *DMA2D_OUT_MACRO_BLOCK_SIZE_CHn Configuration*:

- 0: 8 × 8
- 1: 16 × 8
- 2: 16 × 16

Table 17.7-2. DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configuration

Format of Original Image	Format of the Image to be Compressed	DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*
RGB	YUV444	0
RGB	YUV422	1
RGB	YUV420	2
YUV422	YUV422	1
GRAY	GRAY	0

(g) Enable the reorder function to improve bandwidth utilization by setting DMA2D_OUT_REORDER_EN_CH0 to 1. This function is available only on 2D DMA TX channel 0, not on other TX channels.

(h) Configure 2D DMA channel linked list address:

- 2D DMA TX channel: DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX channel: DMA2D_INLINK_ADDR_CH*n*

4. Enable interrupt sources:

- DMA2D_OUT_INT_ENA_CH*n*_REG

- DMA2D_IN_INT_ENA_CH*n*_REG
- JPEG_INT_ENA_REG

5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0.
   - 2D DMA RX channel: set DMA2D_INLINK_START_CH*n* to 1 and then to 0.
6. Start the JPEG codec encoding by setting the JPEG_JPEG_START field.
7. Wait until DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the encoded bitstream has been completely written to memory.

17.7.2 JPEG Decoder

When the JPEG codec is working as a decoder, the configuration process is:

1. Activate the JPEG codec:
   - Set HP_SYS_CLKRST_JPEG_SYS_CLK_EN to 1 to enable the JPEG codec clock.
   - Set HP_SYS_CLKRST_RST_EN_JPEG to 0 to release the JPEG codec system-level reset.
2. Configure the JPEG codec:
   - Set JPEG_SOFT_RST to 1 to reset the JPEG codec, and then set JPEG_SOFT_RST to 0 to release the JPEG codec reset.
   - Set JPEG_MODE to 1 to configure the JPEG codec to be used as a decoder.
   - Enable the RST marker check following the instructions in Section 17.5.2.2 *Parsing RST Marker*, if required.
   - Configure the quantization coefficient tables by following the instructions in Section 17.5.2.3 *Configurable Quantization Coefficient Table*.
   - Configure the Huffman tables by following the instructions in Section 17.5.2.4 *Configurable Huffman Table*.
   - Configure timeout detection by following the instructions in Section 17.5.2.5 *Timeout Detection*.
   - Configure the width and height of the decoded image:
     - Width: write the decoded image width to JPEG_HA.
     - Height: write the height of the decoded image to JPEG_VA.
     - According to different decoded image formats, JPEG_HA and JPEG_VA must meet the requirements of Table 17.7-3 *JPEG Decoder JPEG_HA and JPEG_VA Configuration*.

Table 17.7-3. JPEG Decoder JPEG_HA and JPEG_VA Configuration

Format of Decoded Image	JPEG_HA	JPEG_VA
YUV444	Multiples of 8	Multiples of 8
YUV422	Multiples of 16	Multiples of 8
YUV420	Multiples of 16	Multiples of 16

GRAY	Multiples of 8	Multiples of 8

3. Configure 2D DMA (for configuration details and related concepts, see Chapter 4 *2D-DMA Controller (2D-DMA)*.)
   (a) Configure the linked list of the 2D DMA TX channel.
      - Set 2DEN to 0 to enable the 1D function.
      - Configure the Buffer address pointer as the starting address where the bitstream to be decoded is stored.
      - Configure size as the number of bytes in the space to store the bitstream to be decoded.
      - Configure mod as 0. mod is invalid when 2DEN is 0.
   (b) Configure the linked list of 2D DMA RX channel.
      - Set 2DEN to 1 to enable 2D function.
      - Configure the Buffer address pointer as the starting address where the decoded image will be stored.
      - Configure pbyte as the number of bytes per pixel after decoding and 2D DMA color space conversion.
      - Configure HA as the output decoded image width, VA as the output decoded image height, hb as the number of pixels per image block in the horizontal direction, and vb as the number of pixels per image block in the vertical direction.
      - Configure mod as 1, which means continuous writing of image blocks of hb × vb size.
      - Configure (X, Y) as the coordinate of the first pixel in the starting image block. According to the decoded image format, hb and vb must meet the restrictions in Table 17.7-4 *JPEG Decoder hb and vb Configuration*:

Table 17.7-4. JPEG Decoder hb and vb Configuration

Format of Decoded Image	hb	vb
YUV444	Multiples of 8	Multiples of 8
YUV422	Multiples of 16	Multiples of 8
YUV420	Multiples of 16	Multiples of 16
GRAY	Multiples of 8	Multiples of 8

   (c) Connect 2D DMA channels to JPEG codec:
      - Connect the TX channel*n* to the JPEG codec by configuring DMA2D_OUT_PERI_SEL_CH*n* to 0;
      - Connect the RX channel*n* to the JPEG codec by configuring DMA2D_IN_PERI_SEL_CH*n* to 0.
   (d) Reset 2D DMA channels:
      - 2D DMA TX channel*n*: set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.

- 2D DMA RX channel*n*: set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*.
- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*.

(f) Select 2D DMA receive image block size by configuring DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* according to the format of the decoded image. See details in Table 17.7-5 *DMA2D_IN_MACRO_BLOCK_SIZE_CHn Configuration*:

- 0: 8 × 8
- 1: 16 × 8
- 2: 16 × 16

Table 17.7-5. DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration

Format of Decoded Image	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*
YUV444	0
YUV422	1
YUV420	2
GRAY	0

(g) Enable the reorder function to improve bandwidth utilization by setting DMA2D_IN_REORDER_EN_CH0 to 1. Note that this function is only available for 2D DMA RX channel 0, not for any other RX channels.

(h) Select whether to scramble the pixel order of the decoded image before 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1
- 5: BYTE0-1-2

Configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 to 5 swaps the pixel order, for example, from YUV444 output by the JPEG decoder to VUY444. This function is available only on 2D DMA RX channel 0, not on other RX channels.

(i) Configure DMA2D_IN_COLOR_INPUT_SEL_CH0, DMA2D_IN_COLOR_3B_PROC_EN_CH0, and DMA2D_IN_COLOR_OUTPUT_SEL_CH0 to select the color conversion method. See Table 17.7-6 *2D DMA Color Space Conversion Configuration*:

Table 17.7-6. 2D DMA Color Space Conversion Configuration

Color Space Conversion Method	INPUT_SEL_CH0	3B_PROC_EN_CH0	OUTPUT_SEL_CH0
YUV420 → RGB	0	1	1
YUV422 → RGB	0	1	1
YUV444 → RGB	2	1	1
YUV444 (only reverse the pixel order)	2	0	1
No color space conversion	7	N/A	N/A

This function is available only on 2D DMA RX channel 0, not on other RX channels.

(j) Configure the parameters for the color space conversion formula, including DMA2D_IN_COLOR_PARAM_H0_CH0, DMA2D_IN_COLOR_PARAM_H1_CH0, DMA2D_IN_COLOR_PARAM_M0_CH0, DMA2D_IN_COLOR_PARAM_M1_CH0, DMA2D_IN_COLOR_PARAM_L0_CH0, and DMA2D_IN_COLOR_PARAM_L1_CH0. This function is available only on 2D DMA RX channel 0, not on other RX channels.

(k) Select whether to reverse the pixel order of the image after 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_POST_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1
- 5: BYTE0-1-2

Setting DMA2D_IN_SCRAMBLE_SEL_POST_CH0 to 5 reverses the pixel order, for example, from RGB to BGR after 2D DMA color space conversion. This function is available only on 2D DMA RX channel 0, not on other RX channels.

(l) Configure 2D DMA channel linked list addresses:

- 2D DMA TX channel: DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX channel: DMA2D_INLINK_ADDR_CH*n*

4. Enable interrupt sources:
   - DMA2D_OUT_INT_ENA_CH*n*_REG
   - DMA2D_IN_INT_ENA_CH*n*_REG
   - JPEG_INT_ENA_REG
5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel*n*: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0.
   - 2D DMA RX channel*n*: set DMA2D_INLINK_START_CH*n* to 1 and then to 0.
6. Start JPEG codec decoding by setting the JPEG_JPEG_START field.

7. Wait until DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the decoded image has been completely written to memory.

17.7.3 Reset

Depending on the scenario, users can **reset** all or part of the JPEG codec modules during encoding or decoding:

- Reset all modules (including all RAM, FIFO, and state machines): set JPEG_SOFT_RST to 1. Register configuration is not reset.
- Reset all RAM and FIFO (for example, when changing the quantization coefficient table or Huffman table): set JPEG_FIFO_RST to 1.
- Reset all state machines (for example, when terminating the current encoding or decoding process): set JPEG_FSM_RST to 1.

17.8 Register Summary

The addresses in this section are relative to JPEG Codec base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Control and configuration registers			
JPEG_CONFIG_REG	System level control and configuration register	0x0000	varies
JPEG_DQT_INFO_REG	Quantization coefficient table information register	0x0004	R/W
JPEG_PIC_SIZE_REG	Image size register	0x0008	R/W
JPEG_TOQNR_REG	Quantization coefficient table0 register	0x0010	HRO
JPEG_T1QNR_REG	Quantization coefficient table1 register	0x0014	HRO
JPEG_T2QNR_REG	Quantization coefficient table2 register	0x0018	HRO
JPEG_T3QNR_REG	Quantization coefficient table3 register	0x001C	HRO
JPEG_DECODE_CONF_REG	Decoder configuration register	0x0020	varies
JPEG_C0_REG	Component 0 register	0x0024	R/W
JPEG_C1_REG	Component 1 register	0x0028	R/W
JPEG_C2_REG	Component 2 register	0x002C	R/W
JPEG_DHT_INFO_REG	Huffman table information register	0x0034	R/W
Interrupt registers			
JPEG_INT_RAW_REG	Interrupt raw status register	0x0038	R/WTC/SS
JPEG_INT_ENA_REG	Interrupt enable register	0x003C	R/W
JPEG_INT_ST_REG	Interrupt masked status register	0x0040	RO
JPEG_INT_CLR_REG	Interrupt clear register	0x0044	WT
Trace and Debug registers			
JPEG_STATUS0_REG	Trace and debug status register 0	0x0048	RO
JPEG_STATUS2_REG	Trace and debug status register 2	0x004C	RO
JPEG_STATUS3_REG	Trace and debug status register 3	0x0050	RO
JPEG_STATUS4_REG	Trace and debug status register 4	0x0054	RO
JPEG_DHT_TOTLEN_DC0_REG	DC0 Huffman table codeword length register	0x0058	HRO
JPEG_DHT_VAL_DC0_REG	DC0 Huffman table symbol register	0x005C	HRO
JPEG_DHT_TOTLEN_AC0_REG	AC0 Huffman table codeword length register	0x0060	HRO
JPEG_DHT_VAL_AC0_REG	AC0 Huffman table symbol register	0x0064	HRO
JPEG_DHT_TOTLEN_DC1_REG	DC1 Huffman table codeword length register	0x0068	HRO
JPEG_DHT_VAL_DC1_REG	DC1 Huffman table symbol register	0x006C	HRO
JPEG_DHT_TOTLEN_AC1_REG	AC1 Huffman table codeword length register	0x0070	HRO
JPEG_DHT_VAL_AC1_REG	AC1 Huffman table symbol register	0x0074	HRO
JPEG_DHT_CODEMIN_DC0_REG	DC0 Huffman table minimum codeword register	0x0078	HRO
JPEG_DHT_CODEMIN_AC0_REG	AC0 Huffman table minimum codeword register	0x007C	HRO

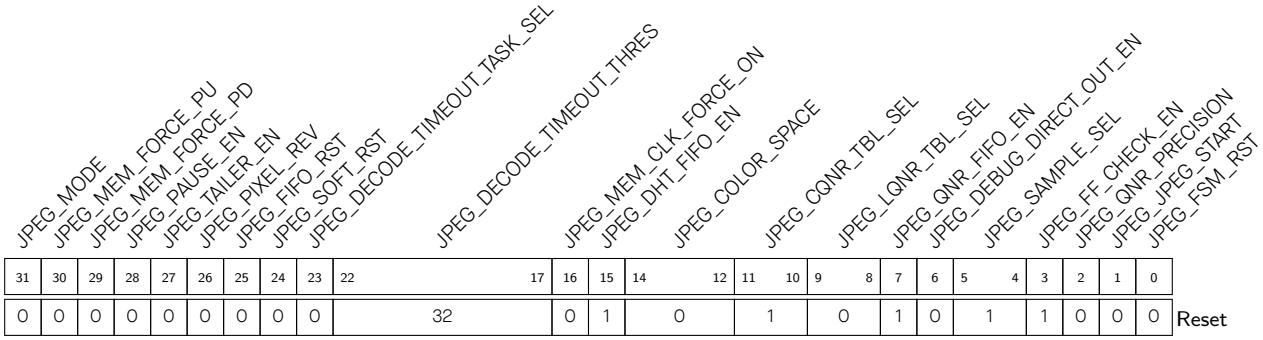
Name	Description	Address	Access
JPEG_DHT_CODEMIN_DC1_REG	DC1 Huffman table minimum codeword register	0x0080	HRO
JPEG_DHT_CODEMIN_AC1_REG	AC1 Huffman table minimum codeword register	0x0084	HRO
JPEG_DECODER_STATUS0_REG	Decoder trace and debug status register 0	0x0088	RO
JPEG_DECODER_STATUS1_REG	Decoder trace and debug status register 1	0x008C	RO
JPEG_DECODER_STATUS2_REG	Decoder trace and debug status register 2	0x0090	RO
JPEG_DECODER_STATUS3_REG	Decoder trace and debug status register 3	0x0094	RO
JPEG_DECODER_STATUS4_REG	Decoder trace and debug status register 4	0x0098	RO
JPEG_DECODER_STATUS5_REG	Decoder trace and debug status register 5	0x009C	varies
JPEG_STATUS5_REG	Trace and debug status register 5	0x00A0	RO
JPEG_ECO_LOW_REG	Redundant register (low)	0x00A4	R/W
JPEG_ECO_HIGH_REG	Redundant register (high)	0x00A8	R/W
JPEG_CONFIG1_REG	JPEG Codec configuration register 1	0x00AC	R/W
JPEG_SYS_REG	System configuration register	0x00F8	R/W
JPEG_VERSION_REG	Version control register	0x00FC	R/W

17.9 Registers

The addresses in this section are relative to the JPEG Codec base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 17.1. JPEG_CONFIG_REG (0x0000)



JPEG_FSM_RST Configures whether to reset the state machine of JPEG Codec.

0: Invalid. No effect

1: Reset the state machine

(WT)

JPEG_JPEG_START Configures whether to start the JPEG Codec.

0: Invalid. No effect

1: Start the JPEG Codec

(WT)

JPEG_QNR_PRECISION Configures the quantization coefficient table precision for the encoder.

0: 8-bit precision

1: 12-bit precision

(R/W)

JPEG_FF_CHECK_EN Configures whether to add "0x00" after "0xFF".

0: Do not add

1: Add

(R/W)

JPEG_SAMPLE_SEL Configures the format of the image to be compressed.

0: YUV444

1: YUV422

2: YUV420

3: Invalid. No effect

(R/W)

JPEG_DEBUG_DIRECT_OUT_EN Configures whether to enable debug mode for encoder.

0: Disable debug mode, the encoder will work in normal mode

1: Enable debug mode, the input image will be directly output by the encoder

(R/W)

JPEG_QNR_FIFO_EN Configures whether to enable FIFO mode when configuring quantization coefficient tables.

0: Disable. Use non-FIFO mode

1: Enable

(R/W)

Continued on the next page...

Register 17.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_LQNR_TBL_SEL Configures the luminance quantization table ID for the encoder. (R/W)

JPEG_CQNR_TBL_SEL Configures the chrominance quantization table ID for the encoder. (R/W)

JPEG_COLOR_SPACE Configures the original image's color space.

0: RGB888
1: YUV422
2: RGB565
3: GRAY
4: YUV444
5: YUV420
6: Invalid. No effect
7: Invalid. No effect
(R/W)

JPEG_DHT_FIFO_EN Configures whether to enable FIFO mode when configuring Huffman tables.

0: Disable. Use non-FIFO mode
1: Enable
Note that to read Huffman tables only non-FIFO mode is supported.
(R/W)

JPEG_MEM_CLK_FORCE_ON Configures whether to force on memory's clock gate.

0: Do not force on
1: Force on
(R/W)

JPEG_DECODE_TIMEOUT_THRES Configures decode timeout period to trigger JPEG_DECODE_TIMEOUT_INT_RAW. The clock cycles of timeout period = $2^{\text{JPEG\_DECODE\_TIMEOUT\_THRES}} - 1$. (R/W)

JPEG_DECODE_TIMEOUT_TASK_SEL Configures the reset mode upon the decoder timeout.

0: Software uses reset to abort the decoding process
1: Hardware automatically aborts decoding process
(R/W)

JPEG_SOFT_RST Configures whether to apply soft reset to the JPEG Codec.

0: Release the soft reset
1: Apply soft reset to the JPEG Codec (the register configuration value will not be reset)
(R/W)

JPEG_FIFO_RST Configures whether to apply FIFO reset to the JPEG Codec.

0: Release the FIFO reset
1: Apply FIFO reset to the JPEG Codec
(R/W)

Continued on the next page...

Register 17.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_PIXEL_REV Configures whether to reverse the original image's pixel order.

0: Do not reverse

1: Reverse

(R/W)

JPEG_TAILER_EN Configures whether to add EOI marker "0xFFD9" at the end of bitstream.

0: Do not add

1: Add

(R/W)

JPEG_PAUSE_EN Configures whether to pause the JPEG Codec.

0: Do not pause

1: Pause

(R/W)

JPEG_MEM_FORCE_PD Configures whether to force JPEG memory to power down.

0: No operation

1: Force power down

(R/W)

JPEG_MEM_FORCE_PU Configures whether to force JPEG memory to power up.

0: No operation

1: Force power up

(R/W)

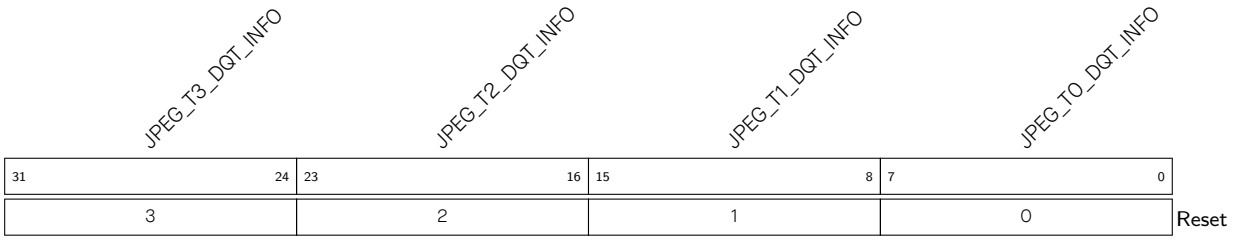
JPEG_MODE Configures whether the JPEG Codec works as an encoder or a decoder.

0: Encoder

1: Decoder

(R/W)

Register 17.2. JPEG_DQT_INFO_REG (0x0004)



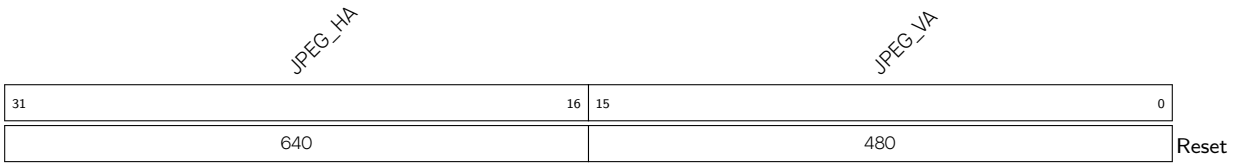
JPEG_T0_DQT_INFO Configures quantization coefficient table 0's precision and ID for decoder.
Bit[4]: Configures precision
0: Lower 8 bits are valid
1: Higher 12 bits are valid
Bit[3:0]: Configures ID
(R/W)

JPEG_T1_DQT_INFO Configures quantization coefficient table 1's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T2_DQT_INFO Configures quantization coefficient table 2's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T3_DQT_INFO Configures quantization coefficient table 3's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

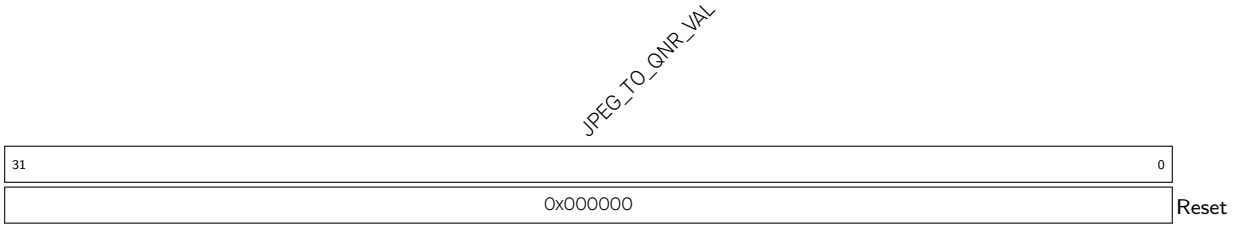
Register 17.3. JPEG_PIC_SIZE_REG (0x0008)



JPEG_VA Configures the image's height.
When JPEG Codec works as an encoder, the maximum configurable bits is 14.
When JPEG Codec works as a decoder, the maximum configurable bits is 16.
(R/W)

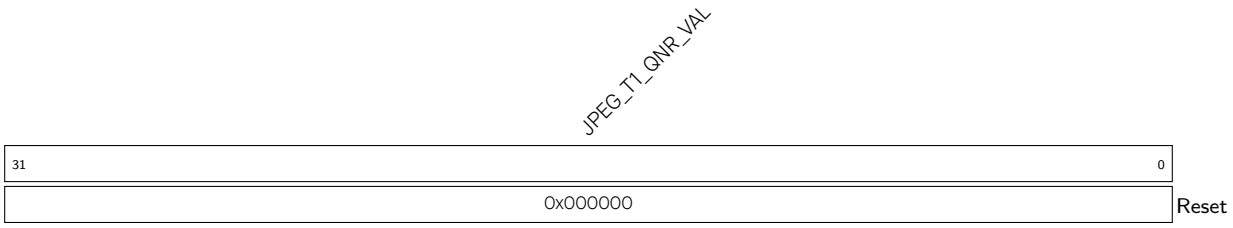
JPEG_HA Configures the image's width.
When JPEG Codec works as an encoder, the maximum configurable bits is 14.
When JPEG Codec works as a decoder, the maximum configurable bits is 16.
(R/W)

Register 17.4. JPEG_TOQNR_REG (0x0010)



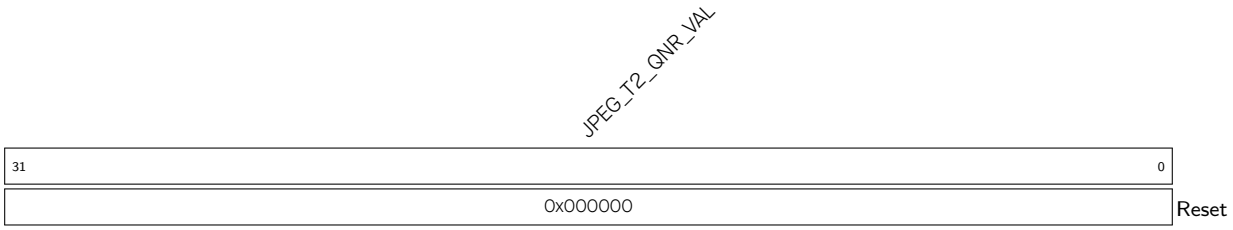
JPEG_TO_QNR_VAL Configures the quantization coefficients of the quantization table 0 in FIFO mode. (HRO)

Register 17.5. JPEG_T1QNR_REG (0x0014)



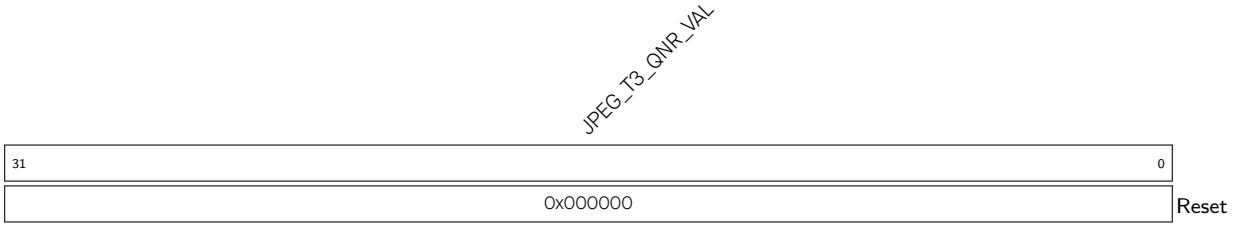
JPEG_T1_QNR_VAL Configures the quantization coefficients of the quantization table 1 in FIFO mode. (HRO)

Register 17.6. JPEG_T2QNR_REG (0x0018)



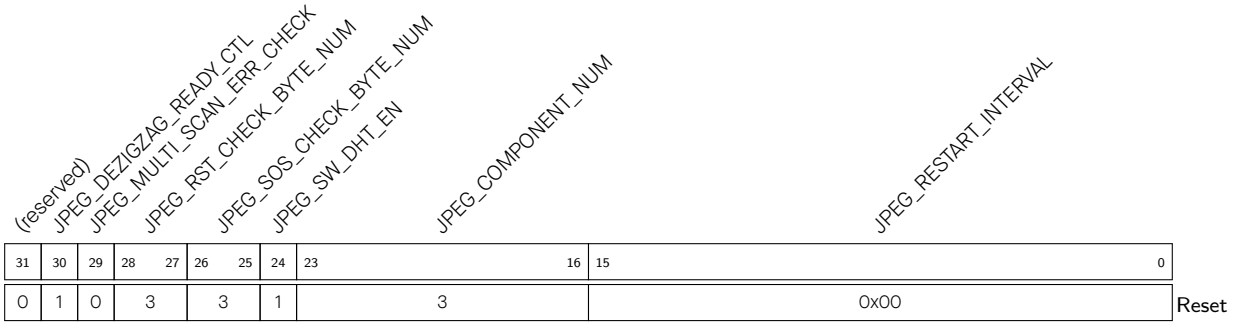
JPEG_T2_QNR_VAL Configures the quantization coefficients of the quantization table 2 in FIFO mode. (HRO)

Register 17.7. JPEG_T3QNR_REG (0x001C)



JPEG_T3_QNR_VAL Configures the quantization coefficients of the quantization table 3 in FIFO mode. (HRO)

Register 17.8. JPEG_DECODE_CONF_REG (0x0020)



JPEG_RESTART_INTERVAL Configures the restart interval in DRI segment when decoding. (R/W)

JPEG_COMPONENT_NUM Configures the number of components in the image when decoding. (R/W)

JPEG_SW_DHT_EN Represents the Huffman table configured by the software for the decoder. The value is always 1. (RO)

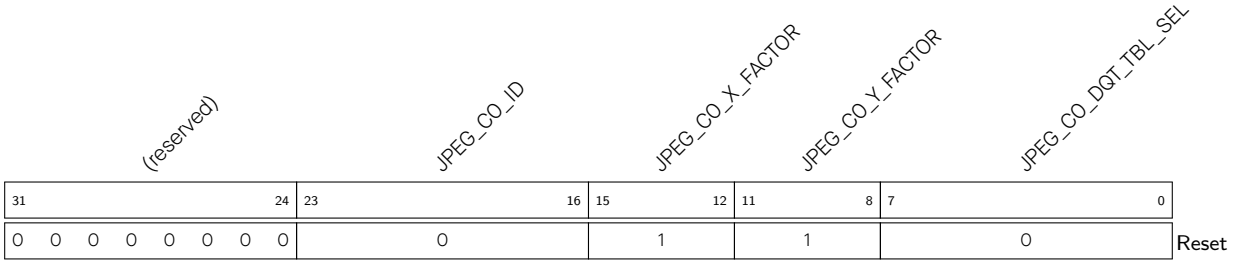
JPEG_SOS_CHECK_BYTE_NUM Configures the byte number to check the next SOS marker in the multi-scan image after one scan is decoded down. The actual check number is JPEG_SOS_CHECK_BYTE_NUM + 1. (R/W)

JPEG_RST_CHECK_BYTE_NUM Configures the byte number to check the next RST marker after one RST interval is decoded down. The actual check number is JPEG_RST_CHECK_BYTE_NUM + 1. (R/W)

JPEG_MULTI_SCAN_ERR_CHECK Reserved for decoder and should not be configured. (R/W)

JPEG_DEZIGZAG_READY_CTL Reserved for decoder and should not be configured. (R/W)

Register 17.9. JPEG_C0_REG (0x0024)



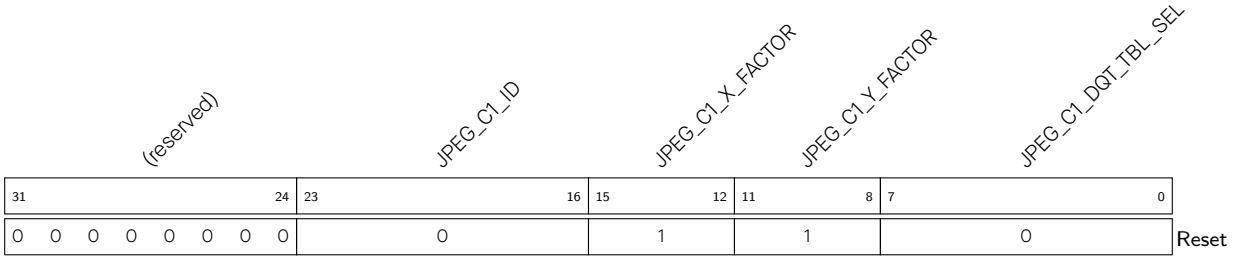
JPEG_C0_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component 0 in decoder mode. (R/W)

JPEG_C0_Y_FACTOR Configures the vertical sampling factor of component 0 in decoder mode. (R/W)

JPEG_C0_X_FACTOR Configures the horizontal sampling factor of component 0 in decoder mode. (R/W)

JPEG_C0_ID Configures the ID of component 0 in decoder mode. (R/W)

Register 17.10. JPEG_C1_REG (0x0028)



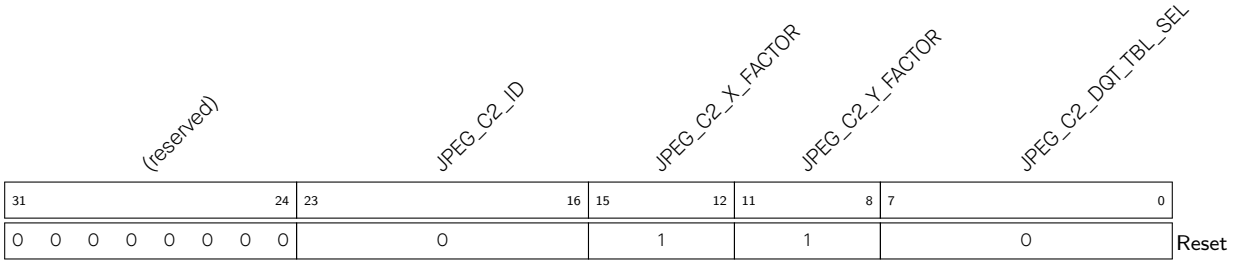
JPEG_C1_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component 1 in decoder mode. (R/W)

JPEG_C1_Y_FACTOR Configures the vertical sampling factor of component 1 in decoder mode. (R/W)

JPEG_C1_X_FACTOR Configures the horizontal sampling factor of component 1 in decoder mode. (R/W)

JPEG_C1_ID Configures the ID of component 1 in decoder mode. (R/W)

Register 17.11. JPEG_C2_REG (0x002C)



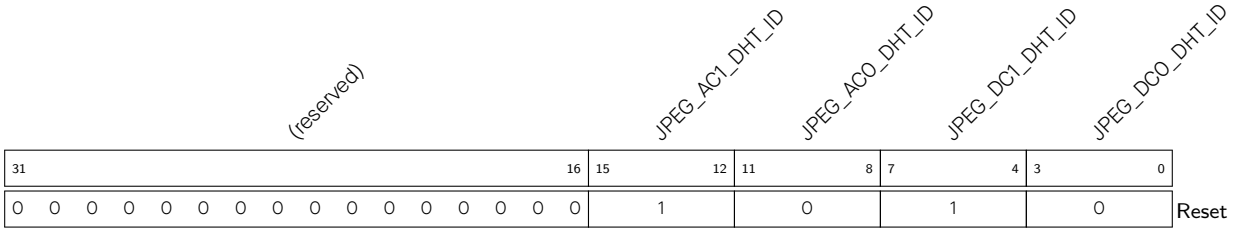
JPEG_C2_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component 2 in decoder mode. (R/W)

JPEG_C2_Y_FACTOR Configures the vertical sampling factor of component 2 in decoder mode. (R/W)

JPEG_C2_X_FACTOR Configures the horizontal sampling factor of component 2 in decoder mode. (R/W)

JPEG_C2_ID Configures the ID of component 2 in decoder mode. (R/W)

Register 17.12. JPEG_DHT_INFO_REG (0x0034)



JPEG_DC0_DHT_ID Configures the ID of DC0 Huffman table in decoder mode. (R/W)

JPEG_DC1_DHT_ID Configures the ID of DC1 Huffman table in decoder mode. (R/W)

JPEG_AC0_DHT_ID Configures the ID of AC0 Huffman table in decoder mode. (R/W)

JPEG_AC1_DHT_ID Configures the ID of AC1 Huffman table in decoder mode. (R/W)

Register 17.13. JPEG_INT_RAW_REG (0x0038)



JPEG_DONE_INT_RAW The raw interrupt status of JPEG_DONE_INT. (R/WTC/SS)

JPEG_RLE_PARALLEL_ERR_INT_RAW The raw interrupt status of JPEG_RLE_PARALLEL_ERR_INT. (R/WTC/SS)

JPEG_CID_ERR_INT_RAW The raw interrupt status of JPEG_CID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_DC_ID_ERR_INT_RAW The raw interrupt status of JPEG_C_DHT_DC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_AC_ID_ERR_INT_RAW The raw interrupt status of JPEG_C_DHT_AC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DQT_ID_ERR_INT_RAW The raw interrupt status of JPEG_C_DQT_ID_ERR_INT. (R/WTC/SS)

JPEG_RST_UXP_ERR_INT_RAW The raw interrupt status of JPEG_RST_UXP_ERR_INT. (R/WTC/SS)

JPEG_RST_CHECK_NONE_ERR_INT_RAW The raw interrupt status of JPEG_RST_CHECK_NONE_ERR_INT. (R/WTC/SS)

JPEG_RST_CHECK_POS_ERR_INT_RAW The raw interrupt status of JPEG_RST_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_OUT_EOF_INT_RAW The raw interrupt status of JPEG_OUT_EOF_INT. (R/WTC/SS)

JPEG_SR_COLOR_MODE_ERR_INT_RAW The raw interrupt status of JPEG_SR_COLOR_MODE_ERR_INT. This interrupt is invalid. (R/WTC/SS)

JPEG_DCT_DONE_INT_RAW The raw interrupt status of JPEG_DCT_DONE_INT. (R/WTC/SS)

JPEG_BS_LAST_BLOCK_EOF_INT_RAW The raw interrupt status of JPEG_BS_LAST_BLOCK_EOF_INT. (R/WTC/SS)

JPEG_SCAN_CHECK_NONE_ERR_INT_RAW The raw interrupt status of JPEG_SCAN_CHECK_NONE_ERR_INT. (R/WTC/SS)

Continued on the next page...

Register 17.13. JPEG_INT_RAW_REG (0x0038)

Continued from the previous page...

JPEG_SCAN_CHECK_POS_ERR_INT_RAW The raw interrupt status of JPEG_SCAN_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_UXP_DET_INT_RAW The raw interrupt status of JPEG_UXP_DET_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_ERR_INT_RAW The raw interrupt status of JPEG_EN_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_LACK_INT_RAW The raw interrupt status of JPEG_EN_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_ERR_INT_RAW The raw interrupt status of JPEG_DE_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_LACK_INT_RAW The raw interrupt status of JPEG_DE_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_SOS_UNMATCH_ERR_INT_RAW The raw interrupt status of JPEG_SOS_UNMATCH_ERR_INT. (R/WTC/SS)

JPEG_MARKER_ERR_FST_SCAN_INT_RAW The raw interrupt status of JPEG_MARKER_ERR_FST_SCAN_INT. (R/WTC/SS)

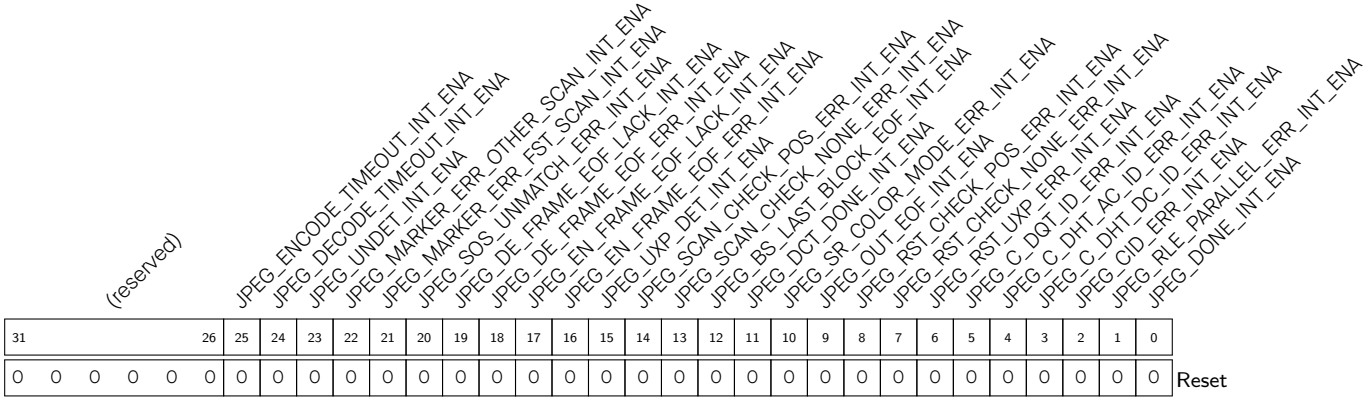
JPEG_MARKER_ERR_OTHER_SCAN_INT_RAW The raw interrupt status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/WTC/SS)

JPEG_UNDET_INT_RAW The raw interrupt status of JPEG_UNDET_INT. (R/WTC/SS)

JPEG_DECODE_TIMEOUT_INT_RAW The raw interrupt status of JPEG_DECODE_TIMEOUT_INT. (R/WTC/SS)

JPEG_ENCODE_TIMEOUT_INT_RAW The raw interrupt status of JPEG_ENCODE_TIMEOUT_INT. (R/WTC/SS)

Register 17.14. JPEG_INT_ENA_REG (0x003C)



JPEG_DONE_INT_ENA Write 1 to enable JPEG_DONE_INT. (R/W)

JPEG_RLE_PARALLEL_ERR_INT_ENA Write 1 to enable JPEG_RLE_PARALLEL_ERR_INT. (R/W)

JPEG_CID_ERR_INT_ENA Write 1 to enable JPEG_CID_ERR_INT. (R/W)

JPEG_C_DHT_DC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_DC_ID_ERR_INT. (R/W)

JPEG_C_DHT_AC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_AC_ID_ERR_INT. (R/W)

JPEG_C_DQT_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DQT_ID_ERR_INT. (R/W)

JPEG_RST_UXP_ERR_INT_ENA Write 1 to enable JPEG_RST_UXP_ERR_INT. (R/W)

JPEG_RST_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_NONE_ERR_INT. (R/W)

JPEG_RST_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_POS_ERR_INT. (R/W)

JPEG_OUT_EOF_INT_ENA Write 1 to enable JPEG_OUT_EOF_INT. (R/W)

JPEG_SR_COLOR_MODE_ERR_INT_ENA Write 1 to enable JPEG_SR_COLOR_MODE_ERR_INT. (R/W)

JPEG_DCT_DONE_INT_ENA Write 1 to enable JPEG_DCT_DONE_INT. (R/W)

JPEG_BS_LAST_BLOCK_EOF_INT_ENA Write 1 to enable JPEG_BS_LAST_BLOCK_EOF_INT. (R/W)

JPEG_SCAN_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_NONE_ERR_INT. (R/W)

JPEG_SCAN_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_POS_ERR_INT. (R/W)

JPEG_UXP_DET_INT_ENA Write 1 to enable JPEG_UXP_DET_INT. (R/W)

Continued on the next page...

Register 17.14. JPEG_INT_ENA_REG (0x003C)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_ERR_INT. (R/W)

JPEG_EN_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_LACK_INT. (R/W)

JPEG_DE_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_ERR_INT. (R/W)

JPEG_DE_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_LACK_INT. (R/W)

JPEG_SOS_UNMATCH_ERR_INT_ENA Write 1 to enable JPEG_SOS_UNMATCH_ERR_INT. (R/W)

JPEG_MARKER_ERR_FST_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_FST_SCAN_INT. (R/W)

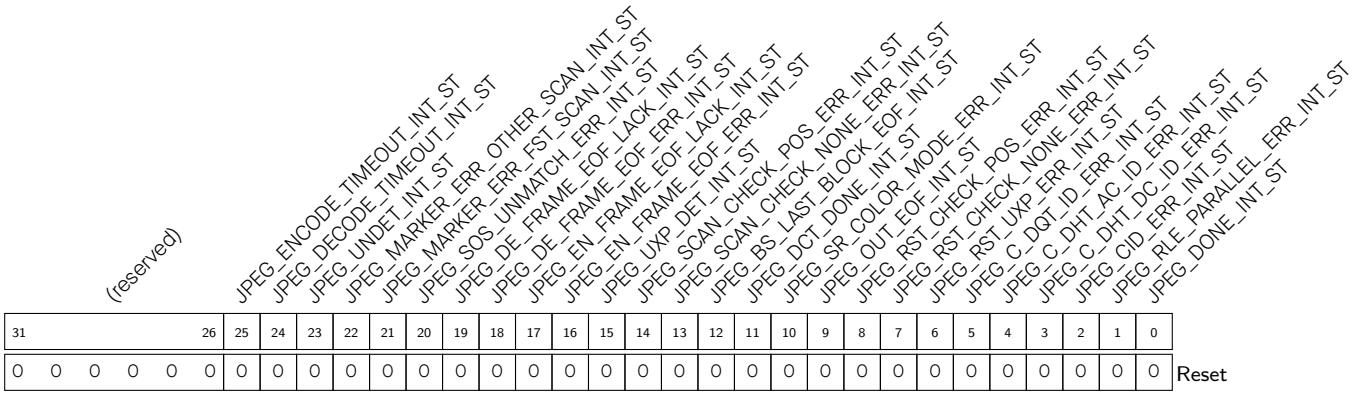
JPEG_MARKER_ERR_OTHER_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/W)

JPEG_UNDET_INT_ENA Write 1 to enable JPEG_UNDET_INT. (R/W)

JPEG_DECODE_TIMEOUT_INT_ENA Write 1 to enable JPEG_DECODE_TIMEOUT_INT. (R/W)

JPEG_ENCODE_TIMEOUT_INT_ENA Write 1 to enable JPEG_ENCODE_TIMEOUT_INT. (R/W)

Register 17.15. JPEG_INT_ST_REG (0x0040)



JPEG_DONE_INT_ST The masked interrupt status of JPEG_DONE_INT. (RO)

JPEG_RLE_PARALLEL_ERR_INT_ST The masked interrupt status of JPEG_RLE_PARALLEL_ERR_INT. (RO)

JPEG_CID_ERR_INT_ST The masked interrupt status of JPEG_CID_ERR_INT. (RO)

JPEG_C_DHT_DC_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DHT_DC_ID_ERR_INT. (RO)

JPEG_C_DHT_AC_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DHT_AC_ID_ERR_INT. (RO)

JPEG_C_DQT_ID_ERR_INT_ST The masked interrupt status of JPEG_C_DQT_ID_ERR_INT. (RO)

JPEG_RST_UXP_ERR_INT_ST The masked interrupt status of JPEG_RST_UXP_ERR_INT. (RO)

JPEG_RST_CHECK_NONE_ERR_INT_ST The masked interrupt status of JPEG_RST_CHECK_NONE_ERR_INT. (RO)

JPEG_RST_CHECK_POS_ERR_INT_ST The masked interrupt status of JPEG_RST_CHECK_POS_ERR_INT. (RO)

JPEG_OUT_EOF_INT_ST The masked interrupt status of JPEG_OUT_EOF_INT. (RO)

JPEG_SR_COLOR_MODE_ERR_INT_ST The masked interrupt status of JPEG_SR_COLOR_MODE_ERR_INT. (RO)

JPEG_DCT_DONE_INT_ST The masked interrupt status of JPEG_DCT_DONE_INT. (RO)

JPEG_BS_LAST_BLOCK_EOF_INT_ST The masked interrupt status of JPEG_BS_LAST_BLOCK_EOF_INT. (RO)

JPEG_SCAN_CHECK_NONE_ERR_INT_ST The masked interrupt status of JPEG_SCAN_CHECK_NONE_ERR_INT. (RO)

JPEG_SCAN_CHECK_POS_ERR_INT_ST The masked interrupt status of JPEG_SCAN_CHECK_POS_ERR_INT. (RO)

JPEG_UXP_DET_INT_ST The masked interrupt status of JPEG_UXP_DET_INT. (RO)

Continued on the next page...

Register 17.15. JPEG_INT_ST_REG (0x0040)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_ERR_INT. (RO)

JPEG_EN_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_LACK_INT. (RO)

JPEG_DE_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_ERR_INT. (RO)

JPEG_DE_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_LACK_INT. (RO)

JPEG_SOS_UNMATCH_ERR_INT_ST The masked interrupt status of JPEG_SOS_UNMATCH_ERR_INT. (RO)

JPEG_MARKER_ERR_FST_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_FST_SCAN_INT. (RO)

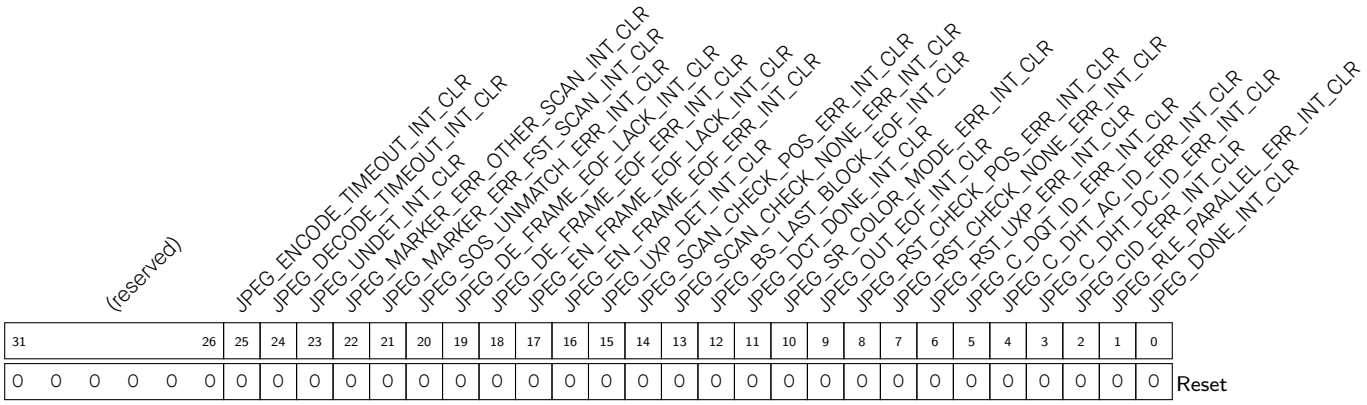
JPEG_MARKER_ERR_OTHER_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (RO)

JPEG_UNDET_INT_ST The masked interrupt status of JPEG_UNDET_INT. (RO)

JPEG_DECODE_TIMEOUT_INT_ST The masked interrupt status of JPEG_DECODE_TIMEOUT_INT. (RO)

JPEG_ENCODE_TIMEOUT_INT_ST The masked interrupt status of JPEG_ENCODE_TIMEOUT_INT. (RO)

Register 17.16. JPEG_INT_CLR_REG (0x0044)



JPEG_DONE_INT_CLR Write 1 to clear JPEG_DONE_INT. (WT)

JPEG_RLE_PARALLEL_ERR_INT_CLR Write 1 to clear JPEG_RLE_PARALLEL_ERR_INT. (WT)

JPEG_CID_ERR_INT_CLR Write 1 to clear JPEG_CID_ERR_INT. (WT)

JPEG_C_DHT_DC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_DC_ID_ERR_INT. (WT)

JPEG_C_DHT_AC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_AC_ID_ERR_INT. (WT)

JPEG_C_DQT_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DQT_ID_ERR_INT. (WT)

JPEG_RST_UXP_ERR_INT_CLR Write 1 to clear JPEG_RST_UXP_ERR_INT. (WT)

JPEG_RST_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_NONE_ERR_INT. (WT)

JPEG_RST_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_POS_ERR_INT. (WT)

JPEG_OUT_EOF_INT_CLR Write 1 to clear JPEG_OUT_EOF_INT. (WT)

JPEG_SR_COLOR_MODE_ERR_INT_CLR Write 1 to clear JPEG_SR_COLOR_MODE_ERR_INT. (WT)

JPEG_DCT_DONE_INT_CLR Write 1 to clear JPEG_DCT_DONE_INT. (WT)

JPEG_BS_LAST_BLOCK_EOF_INT_CLR Write 1 to clear JPEG_BS_LAST_BLOCK_EOF_INT. (WT)

JPEG_SCAN_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_NONE_ERR_INT. (WT)

JPEG_SCAN_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_POS_ERR_INT. (WT)

JPEG_UXP_DET_INT_CLR Write 1 to clear JPEG_UXP_DET_INT. (WT)

JPEG_EN_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_ERR_INT. (WT)

Continued on the next page...

Register 17.16. JPEG_INT_CLR_REG (0x0044)

Continued from the previous page...

JPEG_EN_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_LACK_INT. (WT)

JPEG_DE_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_ERR_INT. (WT)

JPEG_DE_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_LACK_INT. (WT)

JPEG_SOS_UNMATCH_ERR_INT_CLR Write 1 to clear JPEG_SOS_UNMATCH_ERR_INT. (WT)

JPEG_MARKER_ERR_FST_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_FST_SCAN_INT. (WT)

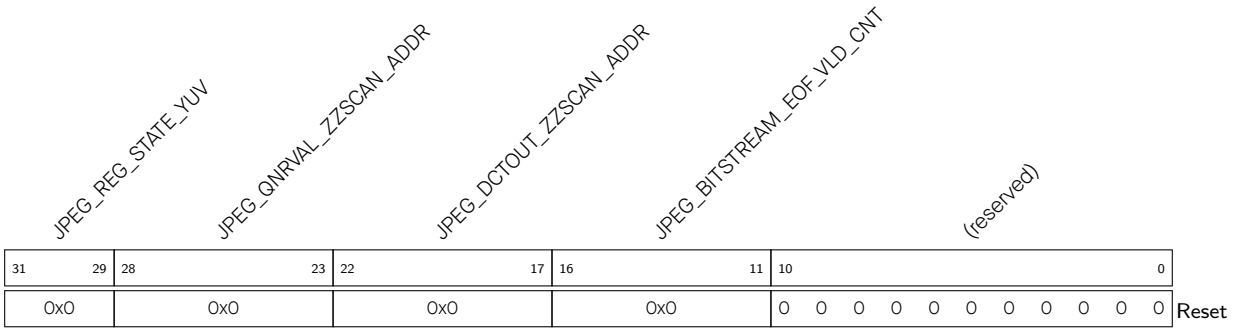
JPEG_MARKER_ERR_OTHER_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_OTHER_SCAN_INT. (WT)

JPEG_UNDET_INT_CLR Write 1 to clear JPEG_UNDET_INT. (WT)

JPEG_DECODE_TIMEOUT_INT_CLR Write 1 to clear JPEG_DECODE_TIMEOUT_INT. (WT)

JPEG_ENCODE_TIMEOUT_INT_CLR Write 1 to clear JPEG_ENCODE_TIMEOUT_INT. (WT)

Register 17.17. JPEG_STATUS0_REG (0x0048)



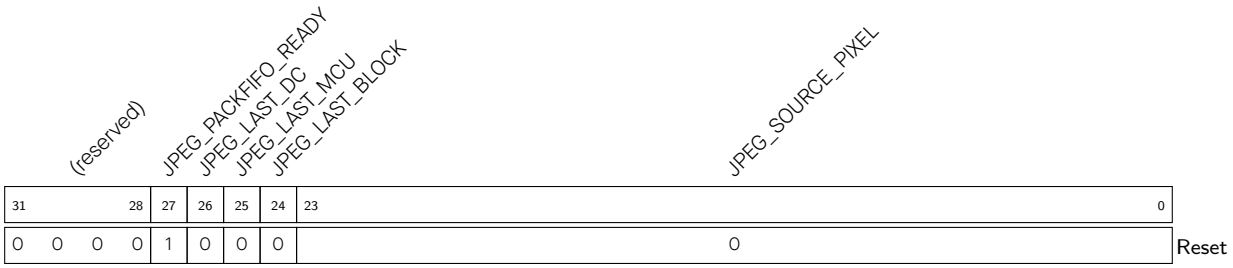
JPEG_BITSTREAM_EOF_VLD_CNT Represents the valid bit count for the last bitstream. (RO)

JPEG_DCTOUT_ZZSCAN_ADDR Represents the zigzag read address from dctout_ram. (RO)

JPEG_QNRVAL_ZZSCAN_ADDR Represents the zigzag read address from qnrval_ram. (RO)

JPEG_REG_STATE_YUV Represents the state of the JPEG Codec state machine. (RO)

Register 17.18. JPEG_STATUS2_REG (0x004C)



JPEG_SOURCE_PIXEL Represents source pixels fetched from DMA. (RO)

JPEG_LAST_BLOCK Represents whether the encoding process is at the last block of the picture.

0: Not the last block

1: Last block

(RO)

JPEG_LAST_MCU Represents whether the encoding process is at the last MCU of the picture.

0: Not the last MCU

1: Last MCU

(RO)

JPEG_LAST_DC Represents whether the encoding process is at the header of the last block of the picture. (RO)

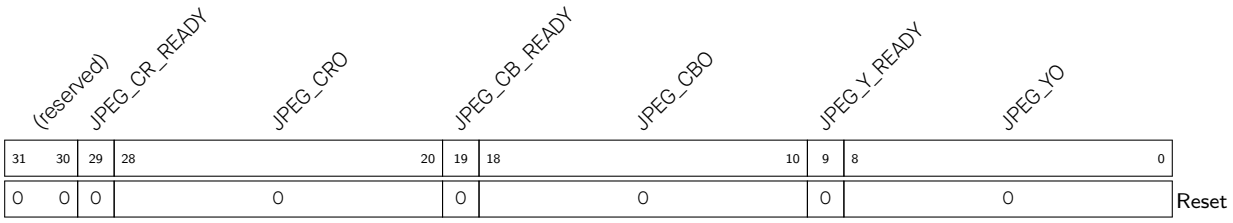
JPEG_PACKFIFO_READY Represents the JPEG pack_fifo ready signal.

0: Not ready

1: Ready

(RO)

Register 17.19. JPEG_STATUS3_REG (0x0050)



JPEG_YO Represents the Y component transferred from the RGB input. (RO)

JPEG_Y_READY Represents the Y component valid signal.

0: Invalid

1: Valid

(RO)

JPEG_CBO Represents the Cb component transferred from the RGB input. (RO)

JPEG_CB_READY Represents the Cb component valid signal.

0: Invalid

1: Valid

(RO)

JPEG_CRO Represents the Cr component transferred from the RGB input. (RO)

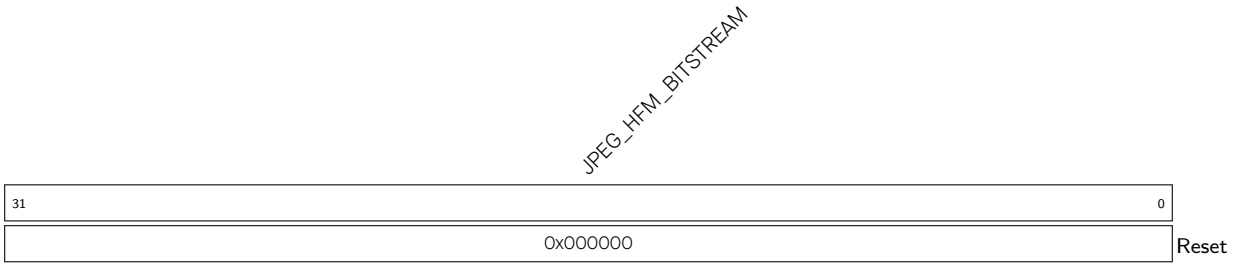
JPEG_CR_READY Represents the Cr component valid signal.

0: Invalid

1: Valid

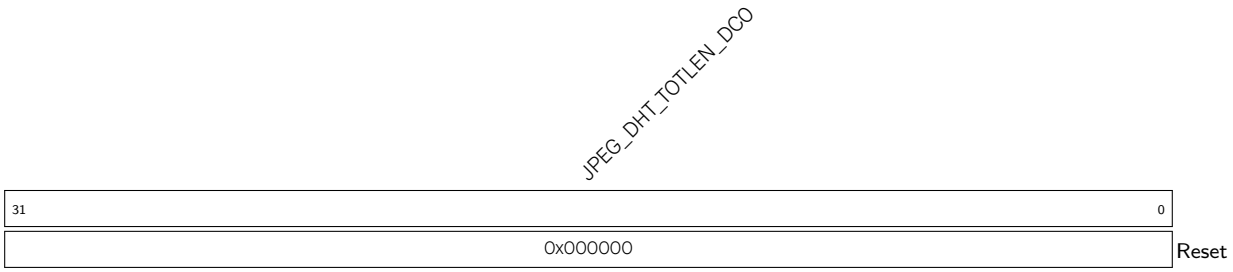
(RO)

Register 17.20. JPEG_STATUS4_REG (0x0054)



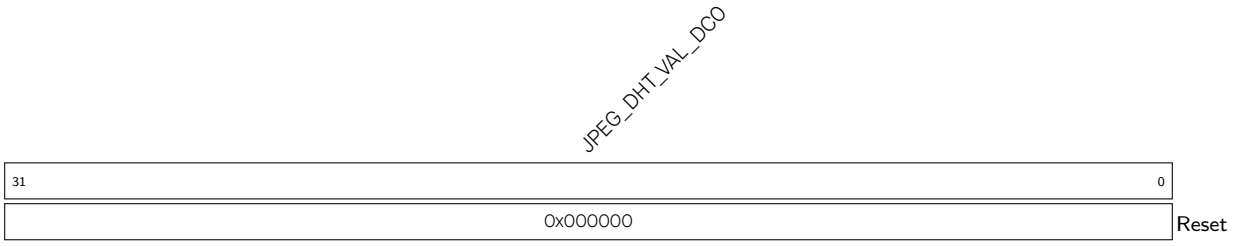
JPEG_HFM_BITSTREAM Represents the Huffman bitstream during the encoding process. (RO)

Register 17.21. JPEG_DHT_TOTLEN_DC0_REG (0x0058)



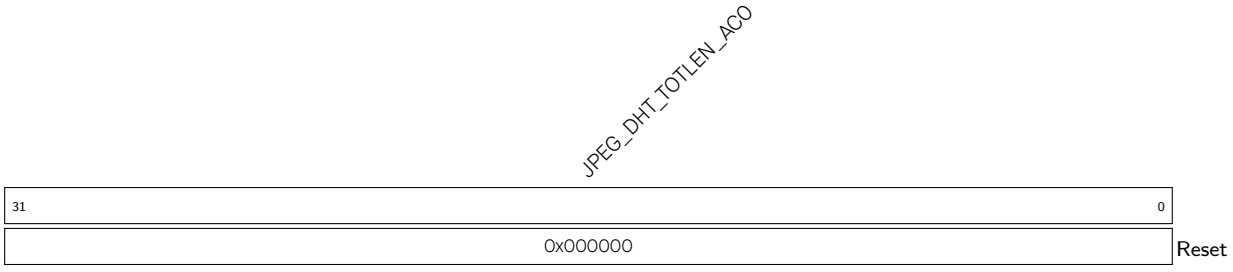
JPEG_DHT_TOTLEN_DC0 Configures the number of codeword with a codeword length of 1 to 16 bits for DC0 Huffman table in FIFO mode. (HRO)

Register 17.22. JPEG_DHT_VAl_DC0_REG (0x005C)



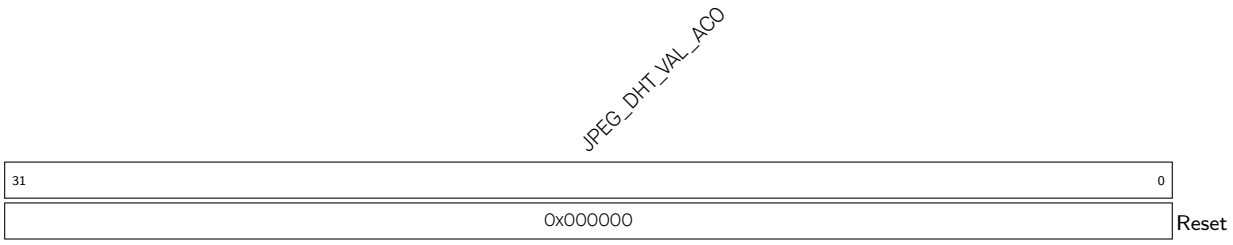
JPEG_DHT_VAL_DC0 Configures the symbols corresponding to the decoded codeword for DC0 Huffman table in FIFO mode. (HRO)

Register 17.23. JPEG_DHT_TOTLEN_AC0_REG (0x0060)



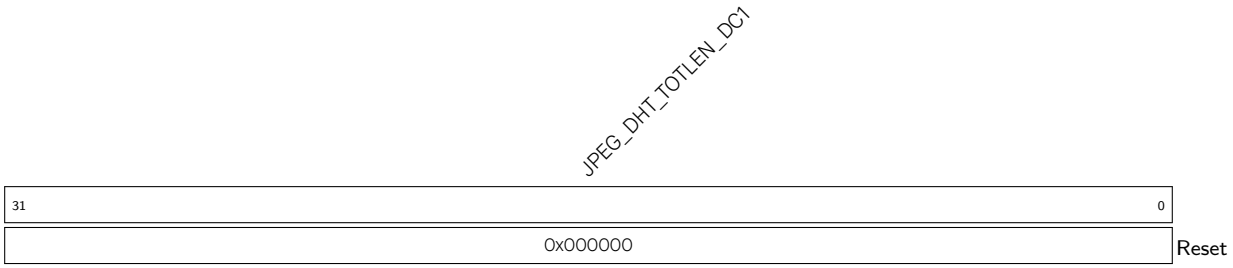
JPEG_DHT_TOTLEN_AC0 Configures the number of codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. (HRO)

Register 17.24. JPEG_DHT_VAl_AC0_REG (0x0064)



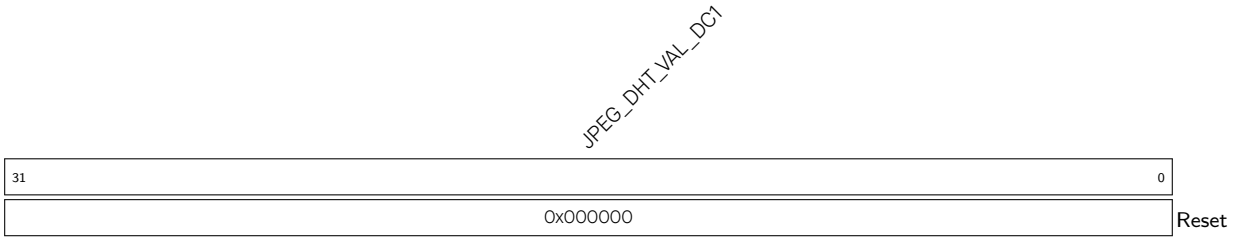
JPEG_DHT_VAL_AC0 Configures the symbols corresponding to the decoded codeword for AC0 Huffman table in FIFO mode. (HRO)

Register 17.25. JPEG_DHT_TOTLEN_DC1_REG (0x0068)



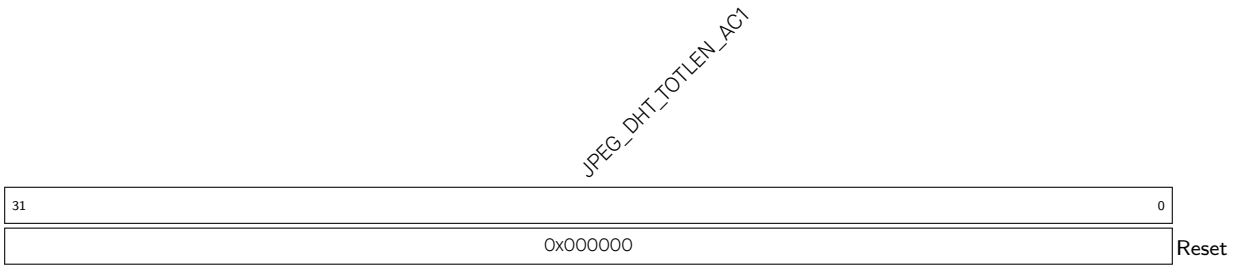
JPEG_DHT_TOTLEN_DC1 Configures the number of codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. (HRO)

Register 17.26. JPEG_DHT_VAl_DC1_REG (0x006C)



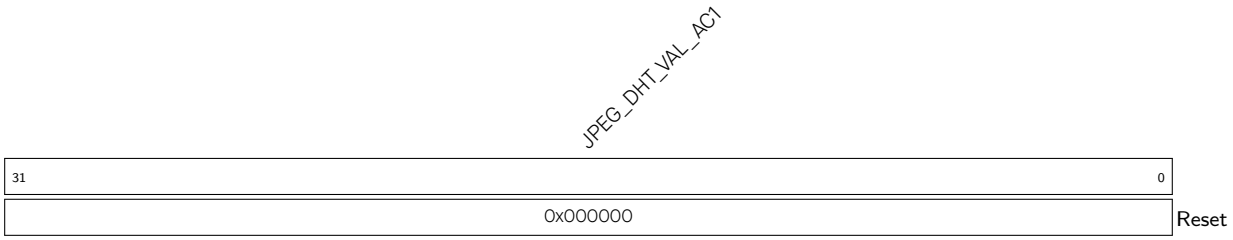
JPEG_DHT_VAL_DC1 Configures the symbols corresponding to the decoded codeword for DC1 Huffman table in FIFO mode. (HRO)

Register 17.27. JPEG_DHT_TOTLEN_AC1_REG (0x0070)



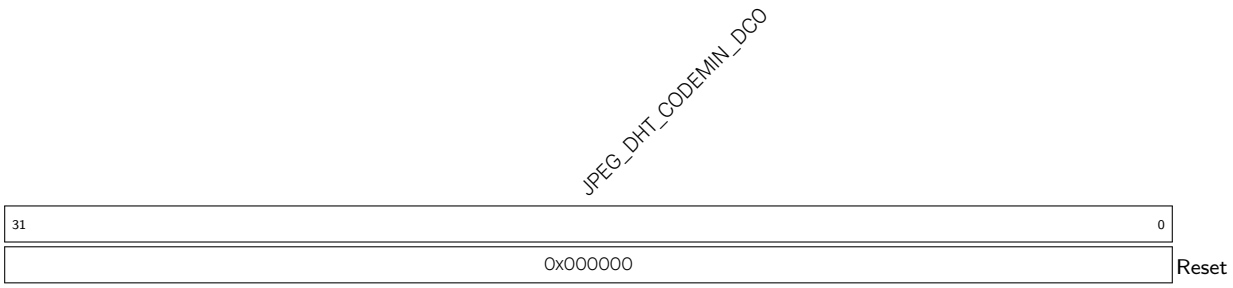
JPEG_DHT_TOTLEN_AC1 Configures the number of codeword with a codeword length of 1 to 16 bits for AC1 Huffman table in FIFO mode. (HRO)

Register 17.28. JPEG_DHT_VAl_AC1_REG (0x0074)



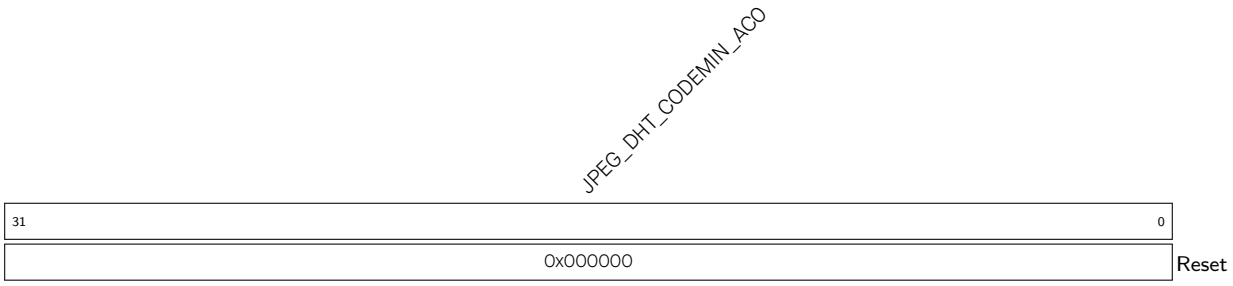
JPEG_DHT_VAL_AC1 Configures the symbols corresponding to the decoded codeword for AC1 Huffman table in FIFO mode. (HRO)

Register 17.29. JPEG_DHT_CODEMIN_DC0_REG (0x0078)



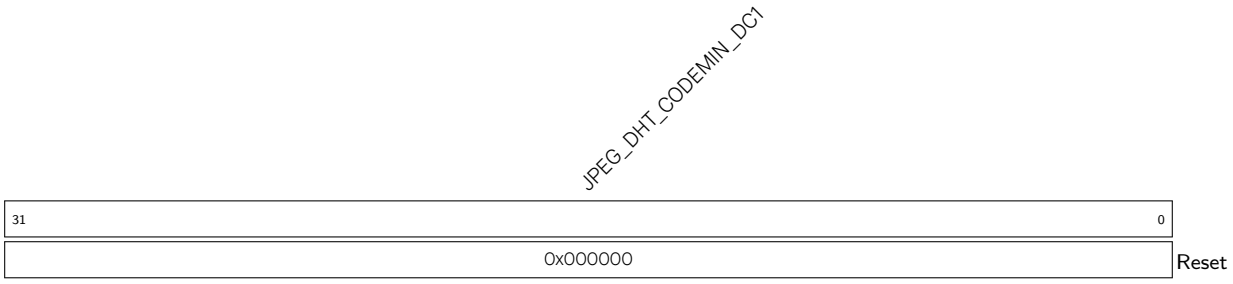
JPEG_DHT_CODEMIN_DC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 17.30. JPEG_DHT_CODEMIN_AC0_REG (0x007C)



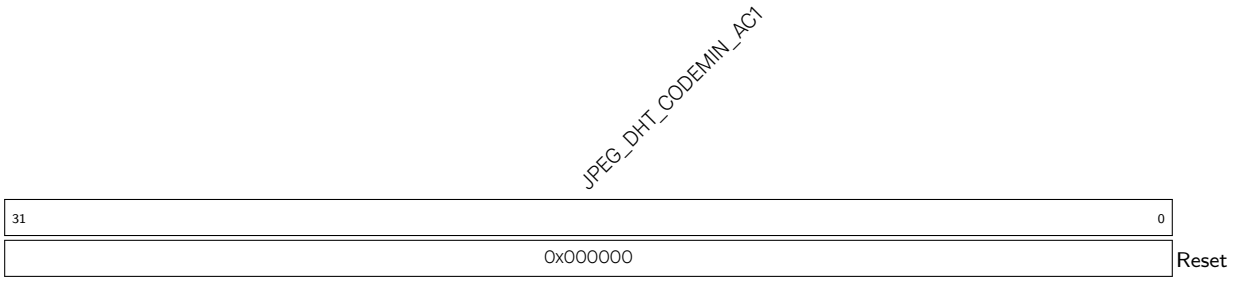
JPEG_DHT_CODEMIN_AC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 17.31. JPEG_DHT_CODEMIN_DC1_REG (0x0080)



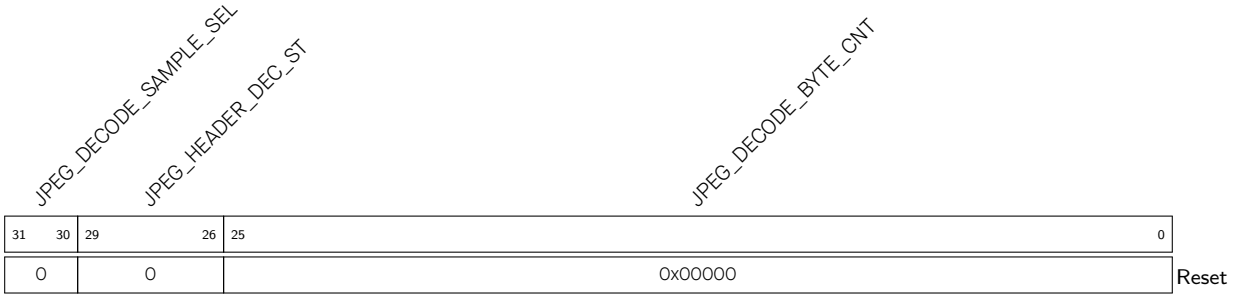
JPEG_DHT_CODEMIN_DC1 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 17.32. JPEG_DHT_CODEMIN_AC1_REG (0x0084)



JPEG_DHT_CODEMIN_AC1 Configures the minimum codeword with a codeword length of 1 to 16 bits for AC1 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 17.33. JPEG_DECODER_STATUS0_REG (0x0088)

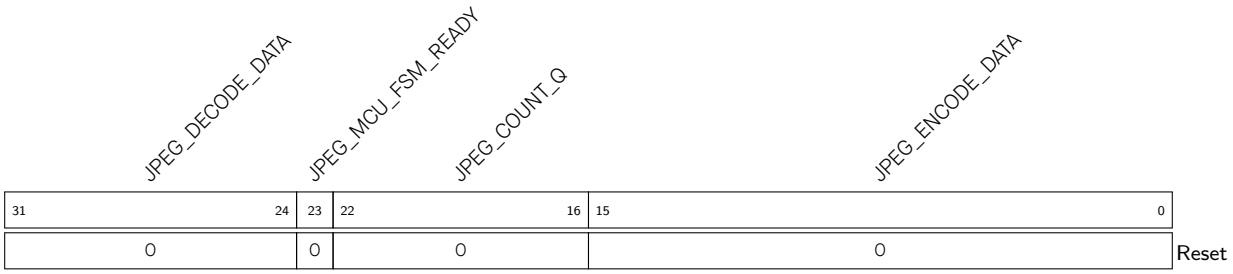


JPEG_DECODE_BYTE_CNT Reserved. (RO)

JPEG_HEADER_DEC_ST Reserved. (RO)

JPEG_DECODE_SAMPLE_SEL Reserved. (RO)

Register 17.34. JPEG_DECODER_STATUS1_REG (0x008C)



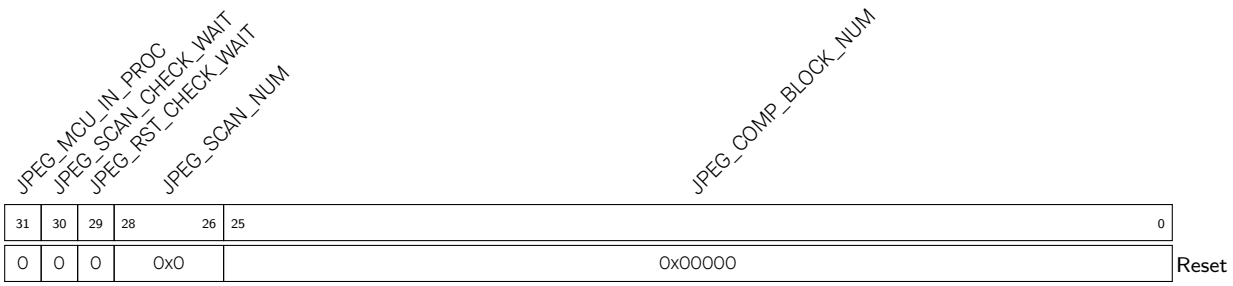
JPEG_ENCODE_DATA Reserved. (RO)

JPEG_COUNT_Q Reserved. (RO)

JPEG_MCU_FSM_READY Reserved. (RO)

JPEG_DECODE_DATA Reserved. (RO)

Register 17.35. JPEG_DECODER_STATUS2_REG (0x0090)



JPEG_COMP_BLOCK_NUM Reserved. (RO)

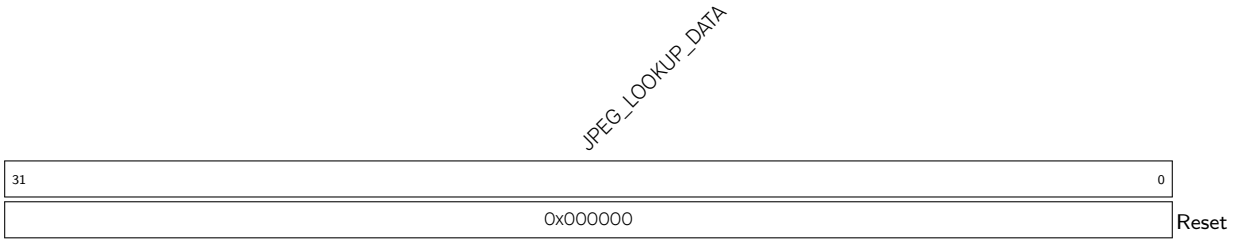
JPEG_SCAN_NUM Reserved. (RO)

JPEG_RST_CHECK_WAIT Reserved. (RO)

JPEG_SCAN_CHECK_WAIT Reserved. (RO)

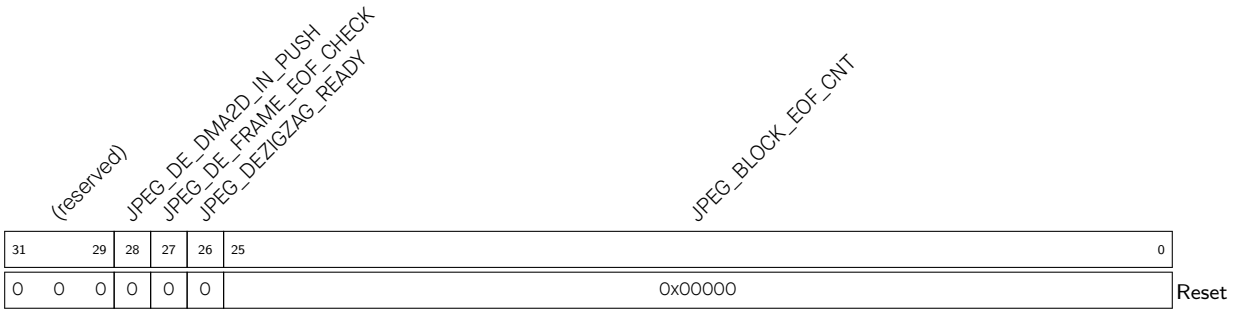
JPEG_MCU_IN_PROC Reserved. (RO)

Register 17.36. JPEG_DECODER_STATUS3_REG (0x0094)



JPEG_LOOKUP_DATA Reserved. (RO)

Register 17.37. JPEG_DECODER_STATUS4_REG (0x0098)



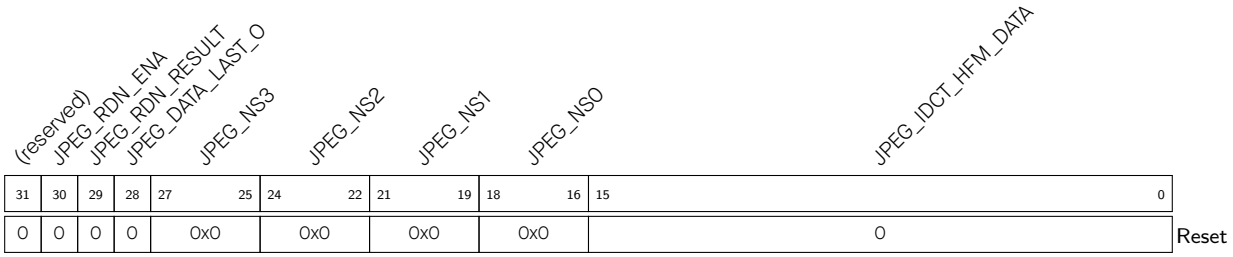
JPEG_BLOCK_EOF_CNT Reserved. (RO)

JPEG_DEZIGZAG_READY Reserved. (RO)

JPEG_DE_FRAME_EOF_CHECK Reserved. (RO)

JPEG_DE_DMA2D_IN_PUSH Reserved. (RO)

Register 17.38. JPEG_DECODER_STATUS5_REG (0x009C)



JPEG_IDCT_HFM_DATA Reserved. (RO)

JPEG_NS0 Reserved. (RO)

JPEG_NS1 Reserved. (RO)

JPEG_NS2 Reserved. (RO)

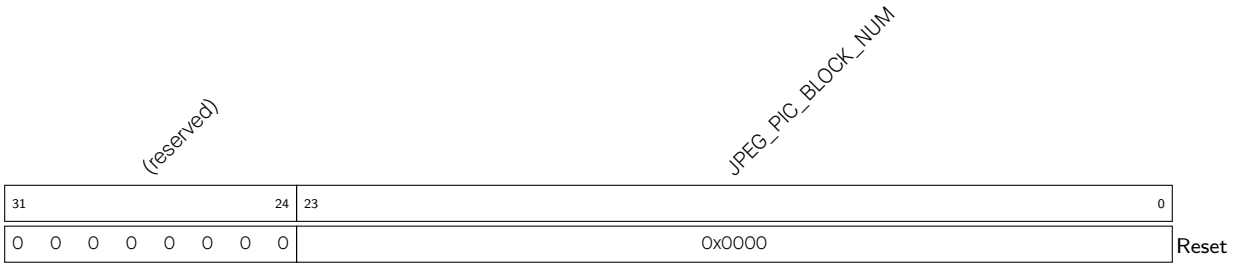
JPEG_NS3 Reserved. (RO)

JPEG_DATA_LAST_O Reserved. (RO)

JPEG_RDN_RESULT Reserved. (RO)

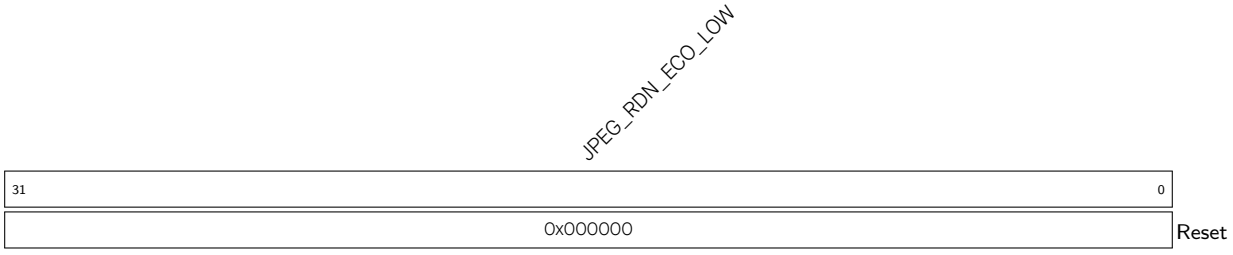
JPEG_RDN_ENA Reserved. (R/W)

Register 17.39. JPEG_STATUS5_REG (0x00A0)



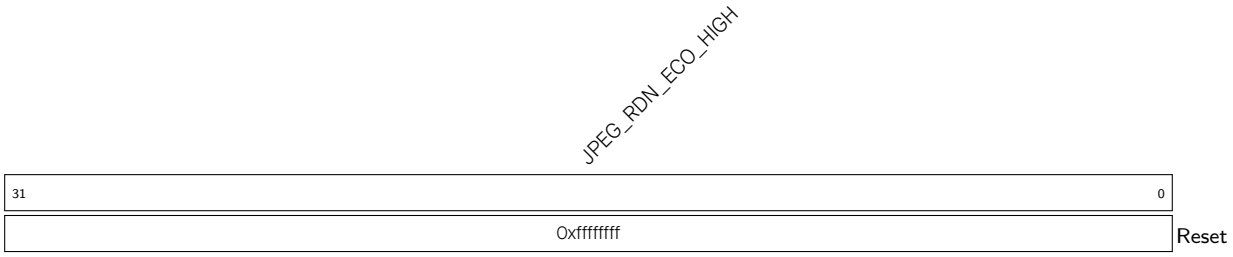
JPEG_PIC_BLOCK_NUM Reserved. (RO)

Register 17.40. JPEG_ECO_LOW_REG (0x00A4)



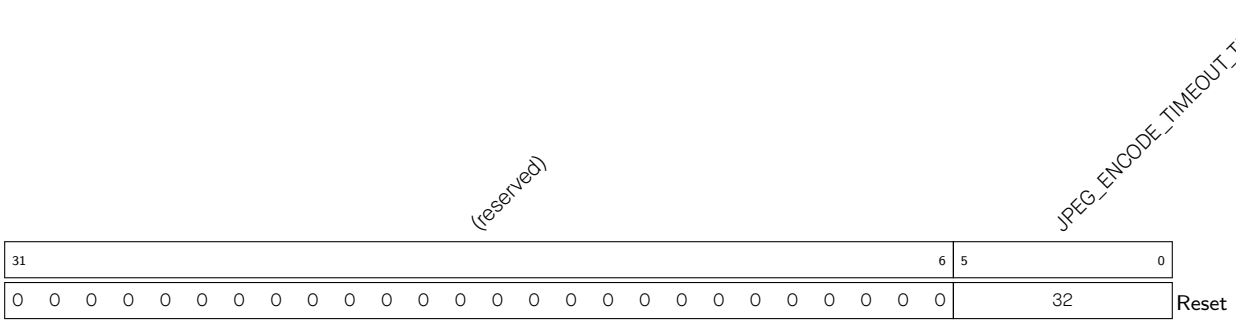
JPEG_RDN_ECO_LOW Reserved. (R/W)

Register 17.41. JPEG_ECO_HIGH_REG (0x00A8)



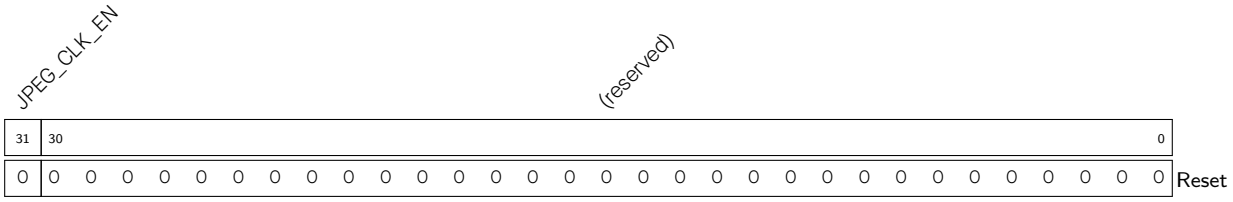
JPEG_RDN_ECO_HIGH Reserved. (R/W)

Register 17.42. JPEG_CONFIG1_REG (0x00AC)



JPEG_ENCODE_TIMEOUT_THRES Configures encode timeout period to trigger JPEG_ENCODE_TIMEOUT_INT_RAW. The clock cycles of timeout period = $2^{\text{JPEG\_ENCODE\_TIMEOUT\_THRES}} - 1$. (R/W)

Register 17.43. JPEG_SYS_REG (0x00F8)



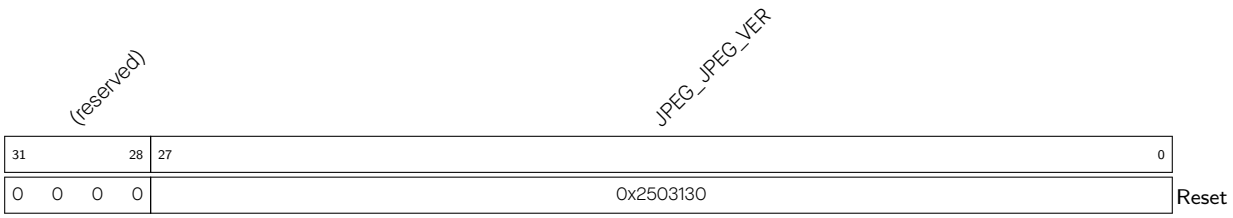
JPEG_CLK_EN Configures whether to force open register clock gate.

0: Open the clock gate only when the application writes registers

1: Force open the clock gate for register

(R/W)

Register 17.44. JPEG_VERSION_REG (0x00FC)



JPEG_JPEG_VER Version control register. (R/W)

Chapter 18

Pixel-Processing Accelerator (PPA)

18.1 Overview

ESP32-S31 includes a pixel-processing accelerator (PPA) designed for 2D image display. This module realizes hardware-level acceleration of image algorithms and implements functions such as image rotation, scaling, mirroring, and blending.

PPA consists of three functional modules: scaling-rotation-mirroring (SRM) , image blending (BLEND), and image filling (FILL). The SRM module implements image rotation, scaling, and XY-axis mirroring functions. The BLEND module achieves the blending of two layers of the same size based on the Alpha channel, i.e., transparency. The FILL module generates a complete image frame according to the configured resolution and pixel data, and writes it to system memory.

PPA processes images in units of pixel block. These pixel blocks are acquired from memory through 2D-DMA. The connection between PPA and 2D-DMA is shown in Figure 18.1-1.

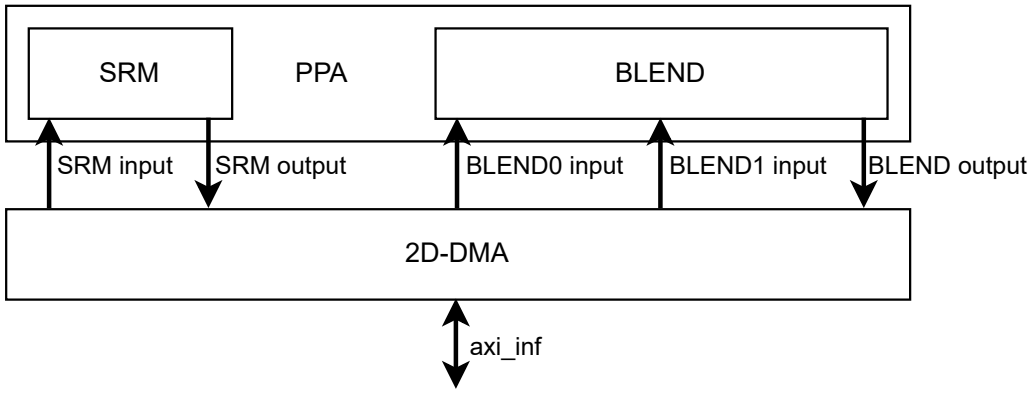


Figure 18.1-1. PPA 2D-DMA Connection

18.2 Terminology

To better illustrate the functions of the PPA module, the following terms are used in this chapter:

image	A complete image stored in the system memory
image block	A portion cropped from an image at a certain size, with the maximum size equivalent to the entire image. i.e., hb, vb size under 2D-DMA 2D-MODO.
pixel block	A portion cropped from an image block. SRM reads and writes data in pixel blocks.

18.3 Features

- Image rotation, scaling, and mirroring by SRM:
  - Input formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Output formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Counterclockwise rotation angles: 0°, 90°, 180°, 270°
  - Horizontal and vertical scaling with scaling factors of 8-bit integer part and 4-bit fractional part
  - Horizontal and vertical mirroring
- Blending two layers of the same size and filling images with specific pixels by BLEND:
  - Foreground input formats: ARGB8888, RGB888, RGB565, L4, L8, A4, A8
  - Background input formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY, L4, L8
  - Output formats: ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY
  - Layer blending based on the Alpha channel. If layers lack an Alpha channel, it can be provided through register configuration.
  - Special color filtering by setting color-key ranges of foreground and background layers
  - FILL module

18.4 Architectural Overview

As shown in figure 18.4-1, there are two independent functional modules in PPA: SRM and BLEND.

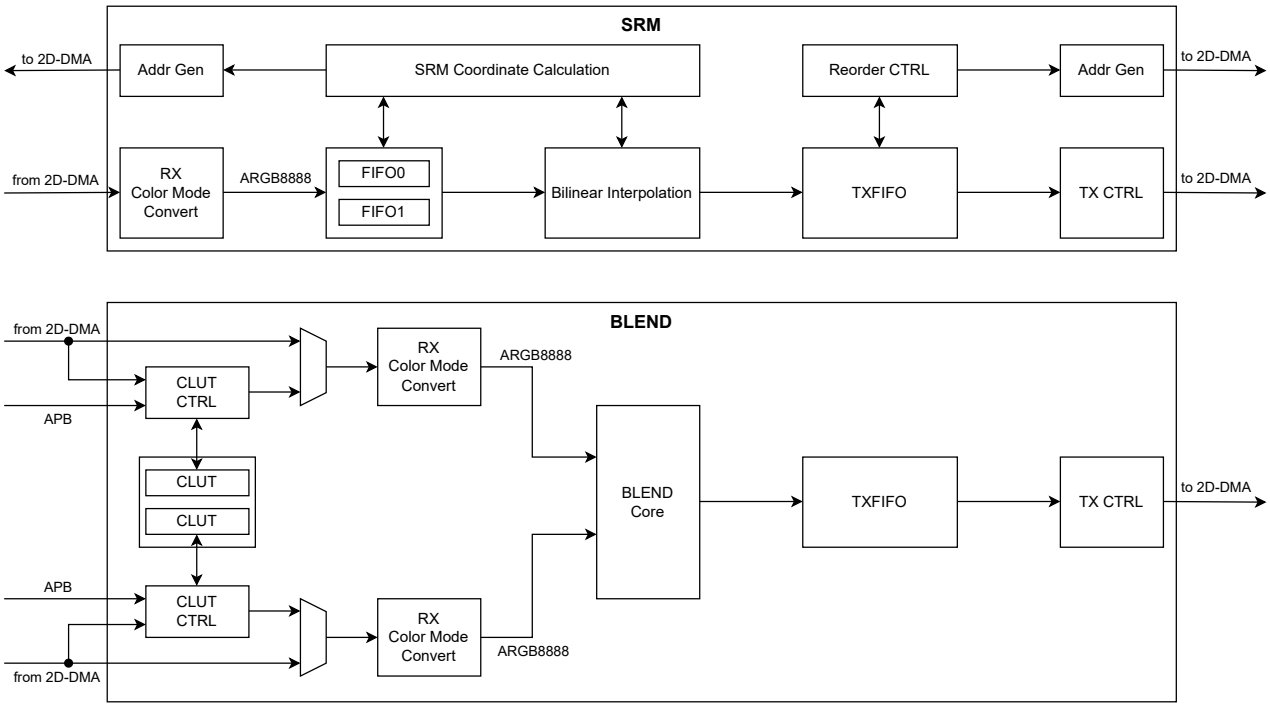


Figure 18.4-1. PPA Architecture

For SRM, SRM Coordinate Calculation scans coordinates of the output pixel blocks and calculates coordinates of the corresponding input pixel blocks. Addr Gen calculates the address according to the input pixel block coordinates and sends them to 2D-DMA. After that, the PPA receives the pixel block data from 2D-DMA, converts it into ARGB8888 format according to the register configuration, and stores it into the RX FIFO (a ping-pong FIFO composed of two SRAMs, each of which can store the whole pixel block). When a complete pixel block is cached, Bilinear Interpolation reads the input pixels from RX FIFO, interpolates them to get the output pixels, and outputs them to TX FIFO. Reorder CTRL then calculates the output pixel block coordinates and address according to the register configuration and sends them to 2D-DMA. At the same time, Reorder CTRL takes out the corresponding pixels from TX FIFO according to the output pixel block coordinates and sends them to 2D-DMA, which saves them to the corresponding address.

For BLEND, PPA also receives data from 2D-DMA and converts it to ARGB8888 format based on the input data type. Yet the converted pixels will be processed directly by the BLEND Core algorithm module, enter the TX FIFO, and ultimately output to the system memory through 2D-DMA. Unlike SRM, BLEND does not need to calculate the address itself as it is completely controlled by 2D-DMA through a linked list.

18.5 Functional Description

18.5.1 PPA Color Space

PPA supports both true color and pseudo color. True color refers to a color representation in which each pixel in a color image consists of three primary color components: R (red), G (green), and B (blue). Each of these components directly determines the intensity of the corresponding primary color on the display device. The true color formats supported by PPA include ARGB8888, RGB888, and RGB565.

Pseudo color refers to a color representation in which the color of each pixel in a color image is not directly determined by the numerical values of the primary color components. Instead, the pixel value serves as an entry into a Color Look-Up Table (CLUT) to retrieve the corresponding A, R, G, and B values. PPA supports CLUT entry addresses with 8-bit or 4-bit width.

In addition, SRM and BLEND background support the YUV422, YUV420, and GRAY data formats. The storage order of supported color formats in memory is shown in Table 18.5-1. The input YUV422 storage order can be set using the SRM or BLEND YUV422_BYTE_ORDER register; the output, however, is always stored with YUV422_BYTE_ORDER = 1.

Table 18.5-1. Pixel Storage Order

Color Format	Start Addr + 3	Start Addr + 2	Start Addr + 1	Start Addr + 0
ARGB8888	A0[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
RGB888	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]
	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
RGB565	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
L8	L3[7:0]	L2[7:0]	L1[7:0]	L0[7:0]
L4	L7[3:0]L6[3:0]	L5[3:0]L4[3:0]	L3[3:0]L2[3:0]	L1[3:0]L0[3:0]
A8	A3[7:0]	A2[7:0]	A1[7:0]	A0[7:0]
A4	A7[3:0]A6[3:0]	A5[3:0]A4[3:0]	A3[3:0]A2[3:0]	A1[3:0]A0[3:0]

Continued on the next page...

Table 18.5-1 – ...Continued from the previous page

Color Format	Start Addr + 3	Start Addr + 2	Start Addr + 1	Start Addr + 0
YUV422 YUV422_BYTE_ORDER=0	**Y1**[7:0]	**V1**[7:0]	**Y0**[7:0]	**U0**[7:0]
YUV422 YUV422_BYTE_ORDER=1	**Y1**[7:0]	**U1**[7:0]	**Y0**[7:0]	**V0**[7:0]
YUV422 YUV422_BYTE_ORDER=2	**V1**[7:0]	**Y1**[7:0]	**U0**[7:0]	**Y0**[7:0]
YUV422 YUV422_BYTE_ORDER=3	**U1**[7:0]	**Y1**[7:0]	**V0**[7:0]	**Y0**[7:0]
YUV420 Odd rows such as 1, 3, 5...	**U2**[7:0]	**Y1**[7:0]	**Y0**[7:0]	**U0**[7:0]
	Y4[7:0]	**U4**[7:0]	**Y3**[7:0]	**Y2**[7:0]
	Y7[7:0]	**Y6**[7:0]	**U6**[7:0]	**Y5**[7:0]
YUV420 Even rows such as 2, 4, 6...	**V2**[7:0]	**Y1**[7:0]	**Y0**[7:0]	**V0**[7:0]
	Y4[7:0]	**V4**[7:0]	**Y3**[7:0]	**Y2**[7:0]
	Y7[7:0]	**Y6**[7:0]	**V6**[7:0]	**Y5**[7:0]
GRAY	**P3**[7:0]	**P2**[7:0]	**P1**[7:0]	**P0**[7:0]

18.5.1.1 SRM Color Space

SRM supports ARGB8888, RGB888, RGB565, and YUV422, YUV420, and GRAY formats for both input and output. The input format is configured by PPA_SRM_RX_CM, while the output format is configured by PPA_SRM_TX_CM. The SRM input pixel processing steps are shown in Figure 18.5-1.

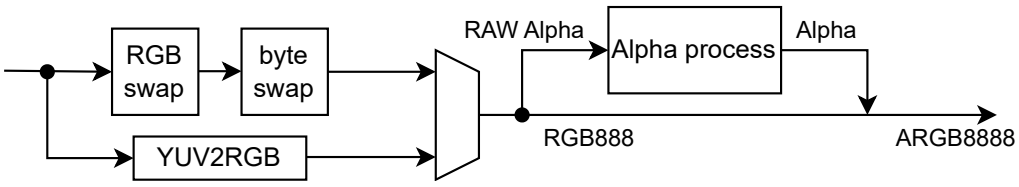


Figure 18.5-1. PPA SRM Input Pixel Processing

SRM supports RGB and byte swapping for ARGB8888, RGB888, and RGB565 input formats. The specific configurations are as follows:

- Set PPA_SRM_RX_RGB_SWAP_EN to 1 to reverse RGB to BGR and ARGB to BGRA.
- Set PPA_SRM_RX_BYTE_SWAP_EN to 1 to reverse bytes 0, 1 and 2, 3 respectively, i.e., {0, 1, 2, 3} will be reverted to {1, 0, 3, 2}. RGB888 would not be affected by this configuration as it only has three bytes per pixel.

SRM uses ARGB8888 when processing images internally. When the input lacks an Alpha channel, the default Alpha channel value for the image is set to 255. The Alpha channel can be further configured using the PPA_SRM_FIX_ALPHA_REG register, as shown in Figure 18.5-2. For specific configuration effects, please refer to Table 18.5-2.

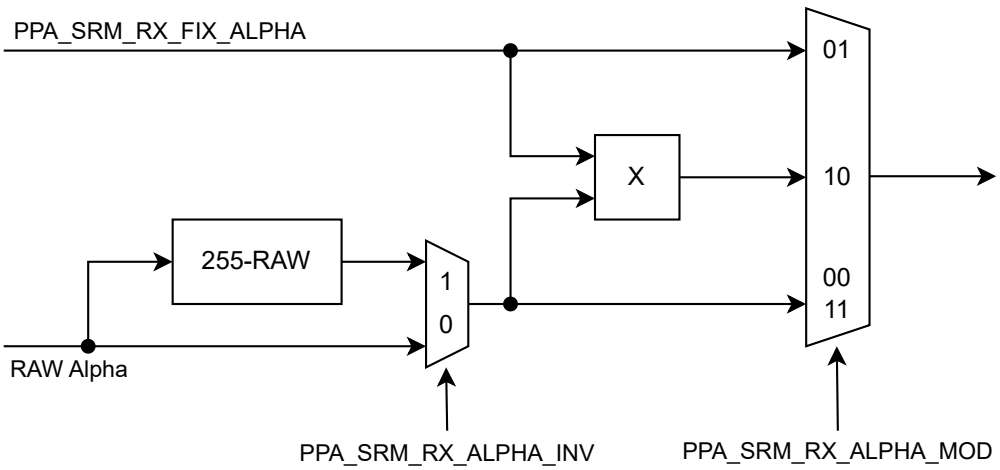


Figure 18.5-2. PPA SRM Alpha Configuration Logic

Table 18.5-2. PPA SRM Alpha Configuration

PPA_SRM_RX_ALPHA_INV	PPA_SRM_RX_ALPHA_MOD	Output
0	00/11	Original Alpha value
	01	The configuration value of PPA_SRM_RX_FIX_ALPHA
	10	The high 8 bits of the value obtained by multiplying the original Alpha value with PPA_SRM_RX_FIX_ALPHA
1	00/11	The value obtained by subtracting the original Alpha value from 255
	01	The configuration value of PPA_SRM_RX_FIX_ALPHA
	10	The high 8 bits of the value obtained by multiplying the result of subtracting the original Alpha value from 255 by PPA_SRM_RX_FIX_ALPHA

Note that when the input or output format is YUV, further configuration of YUV to RGB color conversion is required.

When the input format is YUV422 or YUV420, the input YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_RX_RANGE field, and the protocol used for YUV to RGB conversion needs to be configured as BT601 or BT709 via the PPA_YUV2RGB_PROTOCOL field. The conversion formulas are as follows:

- YUV_{full} to YUV_{limit}
  - $Y_{limit} = \frac{220}{256} Y_{full} + 16$
  - $U_{limit} = \frac{225}{256} U_{full} + 16$
  - $V_{limit} = \frac{225}{256} V_{full} + 16$
- YUV_{limit} to RGB in BT601
  - $R = \frac{298}{256} Y_{limit} + \frac{409}{256} V_{limit} - \frac{56906}{256}$
  - $G = \frac{298}{256} Y_{limit} - \frac{100}{256} U_{limit} - \frac{208}{256} V_{limit} + \frac{34707}{256}$

  - $B = \frac{298}{256}Y_{limit} + \frac{516}{256}U_{limit} - \frac{70836}{256}$

- YUV_{limit} to RGB in BT709
  - $R = \frac{298}{256}Y_{limit} + \frac{459}{256}V_{limit} - \frac{63367}{256}$
  - $G = \frac{298}{256}Y_{limit} - \frac{55}{256}U_{limit} - \frac{136}{256}V_{limit} + \frac{19681}{256}$
  - $B = \frac{298}{256}Y_{limit} + \frac{541}{256}U_{limit} - \frac{73918}{256}$

When the output format is YUV422 or YUV420, the output YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_TX_RANGE field, and the protocol used for RGB to YUV conversion needs to be configured as BT601 or BT709 via the PPA_RGB2YUV_PROTOCOL field. The conversion formulas are as follows:

- YUV_{limit} to YUV_{full}
  - $Y_{full} = \frac{298}{256}Y_{limit} + \frac{4768}{256}$
  - $U_{full} = \frac{291}{256}U_{limit} + \frac{4550}{256}$
  - $V_{full} = \frac{291}{256}V_{limit} + \frac{4550}{256}$
- RGB to YUV_{limit} in BT601
  - $Y_{limit} = \frac{4096}{256} + \frac{66}{256}R + \frac{129}{256}G + \frac{25}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{38}{256}R - \frac{74}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{94}{256}G - \frac{18}{256}B$
- RGB to YUV_{limit} in BT709
  - $Y_{limit} = \frac{4096}{256} + \frac{47}{256}R + \frac{157}{256}G + \frac{16}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{26}{256}R - \frac{86}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{102}{256}G - \frac{10}{256}B$

When the input or output data format is GRAY, additional register configuration is required.

When the input format is configured as GRAY, the SRM replicates the GRAY pixel value to all three RGB channels, thereby converting the input from GRAY to RGB.

When the output format is configured as GRAY, the RGB data is converted to GRAY via PPA_RGB2GRAY_REG. The conversion formula is as follows:

$$GRAY = \frac{R*PPA\_RGB2GRAY\_R + G*PPA\_RGB2GRAY\_G + B*PPA\_RGB2GRAY\_B}{256}$$

18.5.1.2 BLEND Color Space

BLEND supports ARGB8888, RGB888, RGB565, L4, L8, A4, and A8 foreground input formats, and ARGB8888, RGB888, RGB565, YUV422, YUV420, GRAY, L4, L8 background input formats. The input color format of the foreground and background layers can be configured respectively via the PPA_BLEND0/1_RX_CM field. Among the above input formats, A4 and A8 only support the foreground layer, and when the input format is L4 or A4, the size of the image block, hb, and the offset in the image, x, must be an even number. For the output format, BLEND supports ARGB8888, RGB888, RGB565, YUV422, YUV420, and GRAY, which can be configured via the PPA_BLEND_TX_CM field.

BLEND's RGB conversion, byte conversion, and alpha channel configuration are consistent with SRM input. Additionally, when the foreground layer data format is A4 or A8, the input data will be Alpha channel values. In this case, if PPA_BLEND_BYPASS is set as 1, the foreground layer's input Alpha value replaces the background layer's Alpha channel and output. If set as 0, it will be a normal BLEND operation, and the foreground layer's RGB values will be specified by PPA_BLEND_RGB_REG.

When the input data format is L4 or L8, it requires to first initialize the CLUT for BLEND. BLEND has two CLUTs with a depth of 256 and a width of 32. There are two initialization modes: FIFO and MEM.

- Write 0 to PPA_APB_FIFO_MASK to enter the FIFO mode.
  In the FIFO mode, access to CLUT is achieved by reading/writing PPA_RDWR_WORD_BLENDx_CLUT, where 0 corresponds to the background channel CLUT and 1 corresponds to the foreground channel CLUT.
  Each time this register is read/written, the corresponding read/write address is automatically incremented by 1. The read/write address can be reset by writing 1 and then 0 to PPA_BLENDx_CLUT_MEM_RST. The read address can be reset by writing 1 then 0 to PPA_BLENDx_CLUT_MEM_RDADDR_RST.
- Write 1 to PPA_APB_FIFO_MASK to enter the MEM mode.
  In the MEM mode, CLUT can be directly accessed through addresses. The CLUT addresses in the PPA's bus address range are as shown in Table 18.5-3.

Table 18.5-3. BLEND CLUT Address

addr[11:10]	addr[9:2]	addr[1:0]
01: Select BLEND0 CLUT 10: Select BLEND1 CLUT	CLUT actual address, 0 ~ 255	00

18.5.2 PPA 2D-DMA Linked List Configuration

PPA exchanges data with the system memory via 2D-DMA. For both SRM and BLEND, it is necessary to configure the 2D-DMA linked list information for the required image block operations in one go. The 2D-DMA linked list is shown in Figure 18.5-3. For detailed information, please refer to *4 2D-DMA Controller (2D-DMA)*.

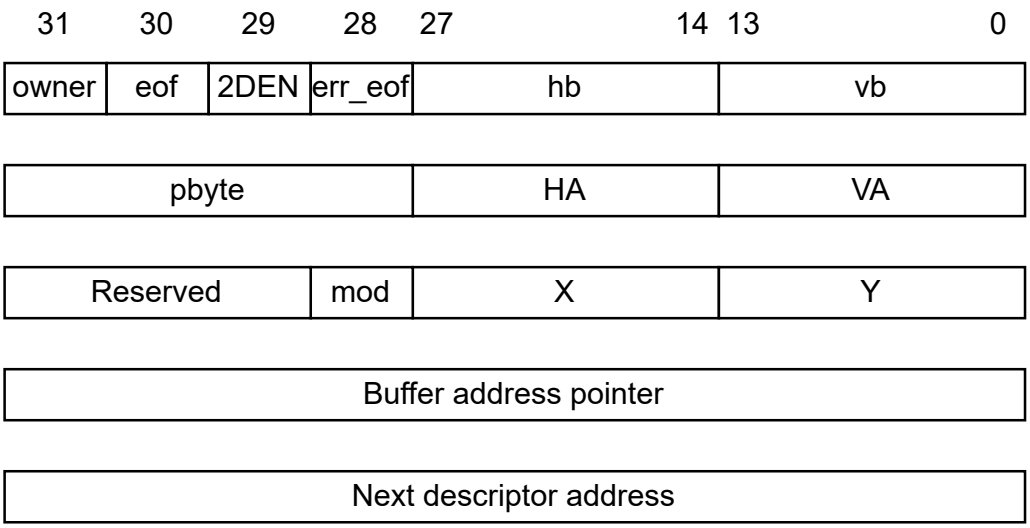


Figure 18.5-3. 2D-DMA Linked List

The corresponding linked list parameters of PPA are shown in Table 18.5-4.

Table 18.5-4. PPA Linked List Configuration

Field	Configuration
owner	1
eof	1
2DEN	1
err_eof	0
hb	Horizontal size of the image block. Must be even if the input or output data format is YUV422 or YUV420
vb	Vertical size of the image block. Must be even if the input or output data format is YUV420
pbyte	ARGB8888: 5 (4 byte/pixel) RGB888: 4 (3 byte/pixel) RGB565/YUV422: 3 (2 byte/pixel) YUV420: 2 (1.5 byte/pixel) GRAY/A8/L8: 1 (1 byte/pixel) A4/L4: 0 (0.5 byte/pixel)
HA	Horizontal size of the entire image. Must be even if the input or output data format is YUV422 or YUV420
VA	Vertical size of the entire image. Must be even if the input or output data format is YUV420
mod	0

Continued on the next page...

Table 18.5-4 – ...Continued from the previous page

Field	Configuration
X	Horizontal offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV422 or YUV420
Y	Vertical offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV420
Buffer address pointer	Starting address of the image
Next descriptor address	Address of next descriptor

18.5.2.1 SRM Special Configuration

Once the 2D-DMA linked lists for input and output image blocks are configured, SRM needs to select the required pixel blocks from the input image block memory space and process them in units of pixel blocks. The processed pixel blocks are then stored in the corresponding memory space of the output image block. Throughout this process, the starting address of each pixel block needs to be controlled by SRM. To realize this functionality, write 1 to the DMA2D_IN/OUT_DSCR_PORT_EN_CHx field of the corresponding channel of the 2D-DMA to enable the descriptor interface (also called DSCR-PORT in 2D-DMA), allowing PPA to control the starting address of the next pixel block in real-time.

For the data input, SRM must receive a complete pixel block each time. When the pixel block selection reaches the boundary of the image block, 2D-DMA needs to automatically pad the pixel block beyond the boundary. This can be achieved by configuring the DMA2D_OUT_DSCR_PORT_BLK_V/H_CHx field of the corresponding channel of 2D-DMA to the pixel block size. The required values depend on the image format, as shown in Table 18.5-5 below.

Table 18.5-5. Automatic Padding Block Size Configuration

Pixel Format	DMA2D_OUT_DSCR_PORT_BLK_H_CHx	DMA2D_OUT_DSCR_PORT_BLK_V_CHx
ARGB8888 RGB888 RGB565 GRAY	34	34
YUV422	36	34
YUV420	36	36

18.5.3 Scaling - Rotation - Mirroring (SRM)

18.5.3.1 Basic Functionality

SRM supports scaling, rotation, and mirroring. It first scales the image based on the origin point, then rotates it around the center point (X_{middle}, Y_{middle}), and finally mirrors it horizontally and vertically based on the center

point (X_{target}, Y_{target}), as shown in Figure 18.5-4. Scaling, rotation, and mirroring parameters are all configured through registers:

- PPA_SRM_SCAL_X_INT: Integer part of the horizontal scaling factor
- PPA_SRM_SCAL_X_FRAG: Fractional part of the horizontal scaling factor
- PPA_SRM_SCAL_Y_INT: Integer part of the vertical scaling factor
- PPA_SRM_SCAL_Y_FRAG: Fractional part of the vertical scaling factor
- PPA_SRM_ROTATE_ANGLE: Counterclockwise rotation angle
- PPA_SRM_MIRROR_X: Enable horizontal mirroring
- PPA_SRM_MIRROR_Y: Enable vertical mirroring

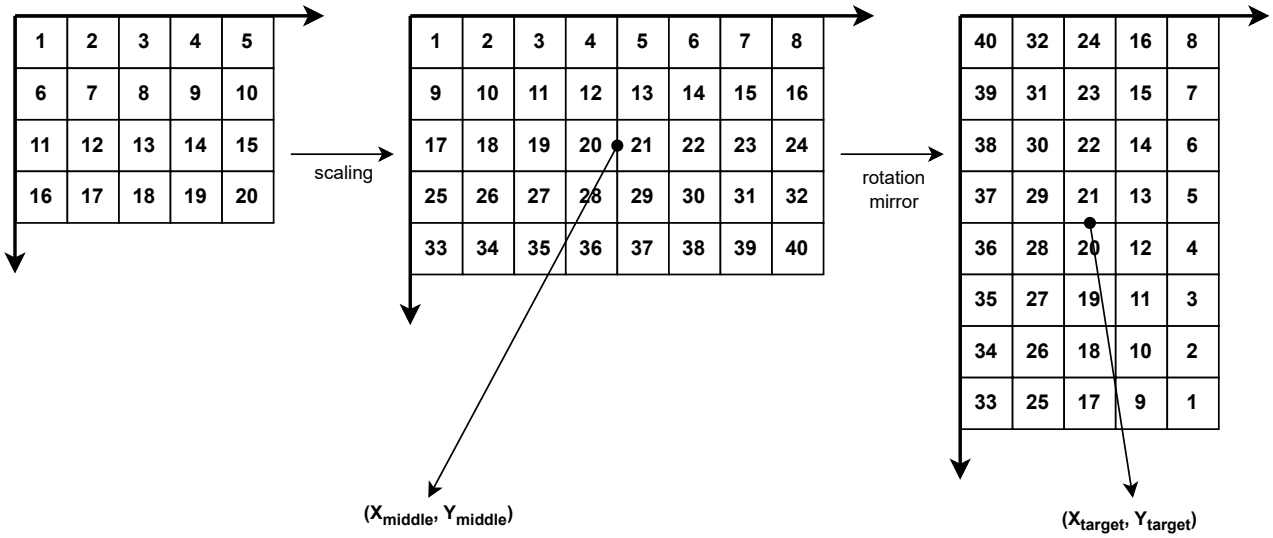


Figure 18.5-4. SRM Procedure

18.5.3.2 Pixel Block Reorder

When the output pixel block is small, initiating a 2D-DMA transfer for each pixel block would result in low bus bandwidth utilization. To improve bus bandwidth utilization, SRM can cache and concatenate output pixel blocks into a larger pixel block before outputting them all at once. This feature is enabled by default and can be disabled by setting PPA_SRM_MACRO_BK_RO_BYPASS to 1.

Once pixel block reorder is enabled, under certain conditions, the output pixel blocks will be automatically rearranged and concatenated, as specified in Table 18.5-6.

- For the output pixel block in ARGB8888/RGB888 format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 16, pixel block reorder will be automatically enabled.
- For the output pixel block in RGB565/YUV422/GRAY format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 32, pixel block reorder will be automatically enabled.
- For the output pixel block in YUV420 format, if its vertical height is less than or equal to 32, and horizontal width is less than or equal to 32, pixel block reorder will be automatically enabled.

Table 18.5-6. Number of Pixel Block Reorders in SRM Output

ARGB8888/RGB888		RGB565/YUV422/GRAY		YUV420	
Horizontal Size	Reorder Number	Horizontal Size	Reorder Number	Horizontal Size (Even)	Reorder Number
1	32	1	64	2	32
2	16	2	32	4	16
3	10	3	20	6	10
4	8	4	16	8	8
5	6	5	12	10	6
6	5	6	10	12	5
7	4	7	9	14	4
8	4	8	8	16	4
9	3	9	7	18	3
10	3	10	6	20-32	2
11	2	11	5		
12	2	12	5		
13	2	13	4		
14	2	14	4		
15	2	15	4		
16	2	16	4		
		17	3		
		18	3		
		19-32	2		

18.5.4 Layer Blending (BLEND)

18.5.4.1 Basic Functionality

BLEND combines two image blocks of the same size using the Alpha channel and then outputs the result. The blending formula is as follows, where Ab is the Alpha channel of the background layer, Af is the Alpha channel of the foreground layer, Cb corresponds to the R, G, and B components of the background layer, and Cf corresponds to the R, G, B components of the foreground layer:

- $Aout = Ab + Af - Ab * Af$
- $Cout = (Cb * Ab * (1 - Af) + Cf * Af)/(Ab + Af - Ab * Af)$

Additionally, BLEND supports color-keying based on pixel color. By configuring color ranges via PPA_CK_FG/BG_HIGH/LOW_REG, BLEND allows for image segmentation into four distinct regions, each subjected to different operations:

- When the foreground pixel falls within the color-key range and the background pixel falls outside the color-key range, output the background pixel.
- When the background pixel falls within the color-key range and the foreground pixel falls outside the color-key range, the output depends on the value of PPA_COLORKEY_FG_BG_REVERSE. If 0, output the background pixel; if 1, output the foreground pixel.

- When both the background and foreground pixels fall within the color-key range, the output is set to the color defined by PPA_COLORKEY_DEFAULT_R/G/B.
- When neither the background nor foreground pixels fall within the color-key range, output follows the normal Alpha Blending process.

When PPA_BLEND_BYPASS is set to 1, BLEND directly outputs the background layer data. If the foreground input data type is A4/A8 at this time, the foreground input data will replace the background input data's Alpha channel and then be output.

18.5.4.2 Filled Image Output

BLEND also supports filled image output. When this feature is enabled, only the output side of BLEND works, continuously outputting the filled image. The size and data of the filled image can be configured via PPA_BLEND_TX_SIZE_REG and PPA_BLEND_TX_FIX_PIXEL.

18.6 Interrupts

ESP32-S31's PPA can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- PPA_INTR

The following interrupt sources can generate PPA_INTR interrupt signal:

- PPA_SRM_PARAM_CFG_ERR_INT: Triggered when invalid parameters are configured in the SRM mode. You can acquire the specific error type through PPA_SRM_PARAM_ERR_ST_REG, as shown in Table 18.6-1.
- PPA_BLEND_PARAM_CFG_ERR_INT: Triggered when invalid parameters are configured in the BLEND mode. You can acquire the specific error type through PPA_BLEND_ST_REG, as shown in Table 18.6-2.
- PPA_SRM_EOF_INT: Triggered when SRM workflow reaches EOF.
- PPA_BLEND_EOF_INT: Triggered when BLEND workflow reaches EOF.

Table 18.6-1. SRM Parameter Error Types

Error Type	Details
PPA_TX_DSCR_VB_ERR_ST	The sum of the vertical size of the output image block from SRM and the vertical offset Y specified in the output descriptor exceeds the total vertical size of the image configured in the output descriptor (VA)
PPA_TX_DSCR_HB_ERR_ST	The sum of the horizontal size of the output image block from SRM and the horizontal offset X specified in the output descriptor exceeds the total horizontal size of the image configured in the output descriptor (HA)
PPA_Y_RX_SCAL_EQUAL_0_ERR_ST	The vertical scaling factor for the input image block in SRM is set to 0

Continued on the next page...

Table 18.6-1 – ...Continued from the previous page

Error Type	Details
PPA_RX_DSCR_VB_ERR_ST	The sum of the vertical size VB and the vertical offset Y in the SRM input descriptor exceeds the overall vertical size VA in the input descriptor
PPA_YDST_LEN_TOO_SAMLL_ERR_ST	The vertical size of the image block after SRM scaling is 0. For example, if the vertical size of the image block is 14 and the vertical scaling factor is 1/16 without rotation, the scaled vertical size of the image block becomes 0
PPA_YDST_LEN_TOO_LARGE_ERR_ST	The vertical size of the image block after SRM scaling exceeds 8191
PPA_X_RX_SCAL_EQUAL_0_ERR_ST	The horizontal scaling factor for the input image block in SRM is 0
PPA_RX_DSCR_HB_ERR_ST	The sum of the horizontal size HB and the horizontal offset X in the SRM input descriptor exceeds the overall horizontal size HA in the input descriptor
PPA_XDST_LEN_TOO_SAMLL_ERR_ST	The horizontal size of the image block after SRM scaling is 0
PPA_XDST_LEN_TOO_LARGE_ERR_ST	The horizontal size of the image block after SRM scaling exceeds 8191
PPA_X_YUV_RX_SCALE_ERR_ST	The input descriptor parameters HA/hb/X happen to be odd numbers when the SRM input format is YUV422 or YUV420
PPA_Y_YUV_RX_SCALE_ERR_ST	The input descriptor parameters VA/vb/Y happen to be odd numbers when the SRM input format is YUV420
PPA_X_YUV_TX_SCALE_ERR_ST	The output descriptor parameters HA/hb/X happen to be odd numbers when the SRM output format is YUV422 or YUV420
PPA_Y_YUV_TX_SCALE_ERR_ST	The output descriptor parameters VA/vb/Y happen to be odd numbers when the SRM output format is YUV420

Table 18.6-2. BLEND Parameter Error Types

Error Type	Details
PPA_BLEND_SIZE_DIFF_ST	The BLEND foreground and background sizes do not match
PPA_BLEND_YUV_X_SCALE_ERR_ST	The BLEND background image format is YUV422 or YUV420, the horizontal size is an odd number
PPA_BLEND_YUV_Y_SCALE_ERR_ST	The BLEND background image format is YUV420, the vertical size is an odd number

18.7 Programming Procedures

18.7.1 PPA Clock Reset Configuration

1. Enable the clock of the PPA and 2D-DMA modules:
   - Write 1 to HP_SYS_CLKRST_REG_PPA_SYS_CLK_EN to enable the PPA clock
   - Write 1 to HP_SYS_CLKRST_REG_DMA2D_SYS_CLK_EN to enable the 2D-DMA clock

2. Reset the PPA and 2D-DMA modules:
   - Write 1 and then 0 to HP_SYS_CLKRST_REG_PPA_RST_EN reset PPA
   - Write 1 and then 0 to HP_SYS_CLKRST_REG_DMA2D_RST_EN reset 2D-DMA

18.7.2 SRM Configuration

1. Refer to Section 18.7.1 to enable and reset PPA and 2D-DMA.
2. Configure the inlink and outlink of 2D-DMA. For detailed parameters, please refer to Table 18.5-4.
3. Select the input and output channels of 2D-DMA:
   - Write 0x1 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to the PPA SRM input
   - Write 0x1 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to the PPA SRM output
4. Enable descriptor interface for 2D-DMA channels used by PPA:
   - Write 1 to DMA2D_OUT_DSCR_PORT_EN_CHx to enable descriptor interface
   - Write 1 to DMA2D_IN_DSCR_PORT_EN_CHx to enable descriptor interface
5. Configure the descriptor interface completion parameters of 2D-DMA:
   - Configure the horizontal completion parameters via DMA2D_OUT_DSCR_PORT_BLK_H_CHx. See Table 18.5-5 for details
   - Configure the vertical completion parameters via DMA2D_OUT_DSCR_PORT_BLK_V_CHx. See Table 18.5-5 for details
6. Enable the input and output channels of 2D-DMA:
   - Write 1 to DMA2D_OUTLINK_START_CHx to enable the output channel of 2D-DMA
   - Write 1 to DMA2D_INLINK_START_CHx to enable the input channel of 2D-DMA
7. Configure SRM parameters:
   - Configure the size of image blocks to be processed via PPA_SRM_BK_SIZE_SEL
   - Configure SRM input data format via PPA_SRM_RX_CM
   - Configure SRM output data format via PPA_SRM_TX_CM
   - Configure the integer part of the horizontal scaling factor via PPA_SRM_SCAL_X_INT
   - Configure the fractional part of the horizontal scaling factor via PPA_SRM_SCAL_X_FRAG
   - Configure the integer part of the vertical scaling factor via PPA_SRM_SCAL_Y_INT
   - Configure the fractional part of the vertical scaling factor via PPA_SRM_SCAL_Y_FRAG
   - Configure the counterclockwise rotation angle via PPA_SRM_ROTATE_ANGLE
   - Enable horizontal mirroring via PPA_SRM_MIRROR_X
   - Enable vertical mirroring via PPA_SRM_MIRROR_Y
8. Write 1 to PPA_SCAL_ROTATE_START to start the SRM workflow.

18.7.3 BLEND CLUT Configuration

1. In the FIFO mode:
   - Write 0 to PPA_APB_FIFO_MASK to enter the FIFO mode
   - Write data to PPA_RDWR_WORD_BLEND0_CLUT to initialize the background layer's CLUT
   - Write data to PPA_RDWR_WORD_BLEND1_CLUT to initialize the foreground layer's CLUT
2. In the MEM mode:
   - Write 1 to PPA_APB_FIFO_MASK to enter the MEM mode
   - Initialize the CLUT by writing data directly to the corresponding address, referring to Table 18.5-3

18.7.4 BLEND Configuration

1. Refer to Section 18.7.1 to enable and reset PPA and 2D-DMA.
2. Configure the input and output link lists of 2D-DMA. For detailed parameters, please refer to Table 18.5-4.
3. Select the input and output channels of 2D-DMA:
   - Write 0x2 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to PPA BLEND for background layer input
   - Write 0x3 to DMA2D_OUT_PERI_SEL_CHx to allocate one 2D-DMA output channel to PPA BLEND for foreground layer input
   - Write 0x2 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to PPA BLEND for output
4. Enable the input and output channels for 2D-DMA. The x in the registers refers to the two channels allocated to BLEND foreground and background respectively:
   - Write 1 to DMA2D_OUTLINK_START_CHx to enable the output channel of 2D-DMA
   - Write 1 to DMA2D_INLINK_START_CHx to enable the input channel of 2D-DMA
5. Configure BLEND parameter:
   - Configure the data type for the background layer input of BLEND via PPA_BLEND0_RX_CM. If the format is L4/L8, initialize the CLUT first as described in 18.7.3
   - Configure the data type for the foreground layer input of BLEND via PPA_BLEND1_RX_CM. If the format is L4/L8, initialize the CLUT first as described in 18.7.3
   - Configure the output data format of BLEND via PPA_BLEND_TX_CM
   - Configure the upper and lower limits of the color-key for the background layer via PPA_CK_BG_HIGH_REG and PPA_CK_BG_LOW_REG
   - Configure the upper and lower limits of the color-key for the foreground layer via PPA_CK_FG_HIGH_REG and PPA_CK_FG_LOW_REG
   - Configure the default value and behavior for the color-key via PPA_CK_DEFAULT_REG
   - Write 1 to PPA_BLEND_EN to enable BLEND

6. Write 1 to PPA_BLEND_TRANS_MODE_UPDATE to start BLEND workflow

18.7.5 BLEND Image Filling Configuration

1. Refer to Section 18.7.1 to enable and reset PPA and 2D-DMA.

2. Configure the input link lists of 2D-DMA (output link lists of PPA BLEND). For detailed parameters, please refer to Table 18.5-4.

3. Write 0x2 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to PPA BLEND output.

4. Write 1 to DMA2D_INLINK_START_CHx to enable the input channel of 2D-DMA.

5. Configure BLEND parameters:

   - Configure the output data format of BLEND via PPA_BLEND_TX_CM
   - Write 1 to PPA_BLEND_FIX_PIXEL_FILL_EN to enable image filling
   - Configure the pixel value for image filling via PPA_BLEND_TX_FIX_PIXEL
   - Configure the filled image size via PPA_BLEND_HB and PPA_BLEND_VB

6. Write 1 to PPA_BLEND_TRANS_MODE_UPDATE to start BLEND workflow.

18.7.6 Error Handling

- Refer to Section 18.7.1 to reset the whole PPA, and configure again.
- Or write 1 and then 0 to reset SRM and BLEND via PPA_SCAL_ROTATE_RST, PPA_BLEND_RST, and configure again.

Note:

- The 2D-DMA configurations mentioned in this chapter involve only key steps. For detailed procedures, please refer to *4 2D-DMA Controller (2D-DMA)*.
- In this chapter, DMA2D_OUT corresponds to the PPA input direction, and DMA2D_IN corresponds to the PPA output direction.

18.8 Register Summary

The addresses in this section are relative to the Pixel-Processing Accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Registers			
PPA_BLEND0_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND background layer	0x0000	R/W
PPA_BLEND1_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND foreground layer	0x0004	R/W
PPA_CLUT_CONF_REG	Configures BLEND CLUT	0x000C	R/W
PPA_SRM_COLOR_MODE_REG	Configures SRM color space	0x0020	R/W
PPA_BLEND_COLOR_MODE_REG	Configures BLEND color space	0x0024	R/W
PPA_SRM_BYTE_ORDER_REG	Configures SRM byte order	0x0028	R/W
PPA_BLEND_BYTE_ORDER_REG	Configures BLEND byte order	0x002C	R/W
PPA_BLEND_TRANS_MODE_REG	Configures the BLEND mode	0x0034	varies
PPA_SRM_FIX_ALPHA_REG	Configures SRM Alpha channel	0x0038	R/W
PPA_BLEND_TX_SIZE_REG	Configures BLEND image filling size	0x003C	R/W
PPA_BLEND_FIX_ALPHA_REG	Configures BLEND Alpha override	0x0040	R/W
PPA_BLEND_RGB_REG	Configures RGB color	0x0048	R/W
PPA_BLEND_FIX_PIXEL_REG	Configures BLEND image filling pixel	0x004C	R/W
PPA_CK_FG_LOW_REG	Configures the foreground color-key lower threshold of BLEND	0x0050	R/W
PPA_CK_FG_HIGH_REG	Configures the foreground color-key higher threshold of BLEND	0x0054	R/W
PPA_CK_BG_LOW_REG	Configures the background color-key lower threshold of BLEND	0x0058	R/W
PPA_CK_BG_HIGH_REG	Configures the background color-key higher threshold of BLEND	0x005C	R/W
PPA_CK_DEFAULT_REG	Configures the default color-key value of BLEND	0x0060	R/W
PPA_SRM_SCAL_ROTATE_REG	Configures the SRM mode	0x0064	varies
PPA_REG_CONF_REG	Enables register clock	0x006C	R/W
PPA_RGB2GRAY_REG	Configures the coefficient in RGB to GRAY conversion	0x0098	R/W
Interrupt Registers			
PPA_INT_RAW_REG	Raw status interrupt	0x0010	R/WTC/SS
PPA_INT_ST_REG	Masked interrupt	0x0014	RO
PPA_INT_ENA_REG	Interrupt enable bits	0x0018	R/W
PPA_INT_CLR_REG	Interrupt clear bits	0x001C	WT

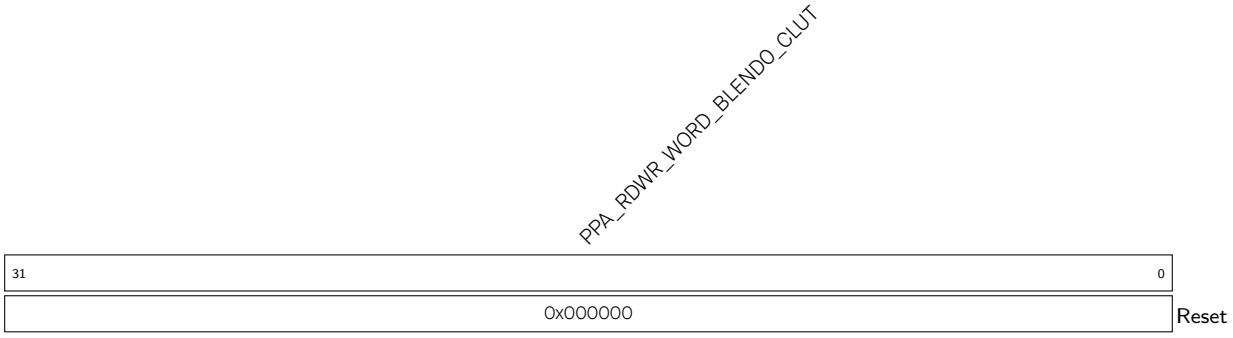
Name	Description	Address	Access
Status Registers			
PPA_CLUT_CNT_REG	BLEND CLUT write counter register	0x0070	RO
PPA_BLEND_ST_REG	BLEND status register	0x0074	RO
PPA_SRM_PARAM_ERR_ST_REG	SRM configuration error status register	0x0078	RO
Debug Register			
PPA_DEBUG_CTRL0_REG	Debug register	0x0090	R/W
PPA_DEBUG_CTRL1_REG	Debug register	0x0094	R/W
Version Register			
PPA_DATE_REG	PPA Version register	0x0100	R/W

18.9 Registers

The addresses in this section are relative to the Pixel-Processing Accelerator base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

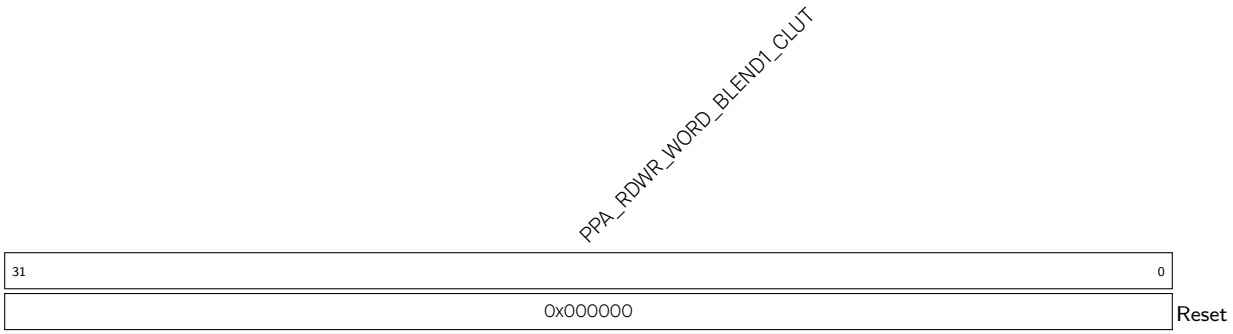
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 18.1. PPA_BLEND0_CLUT_DATA_REG (0x0000)



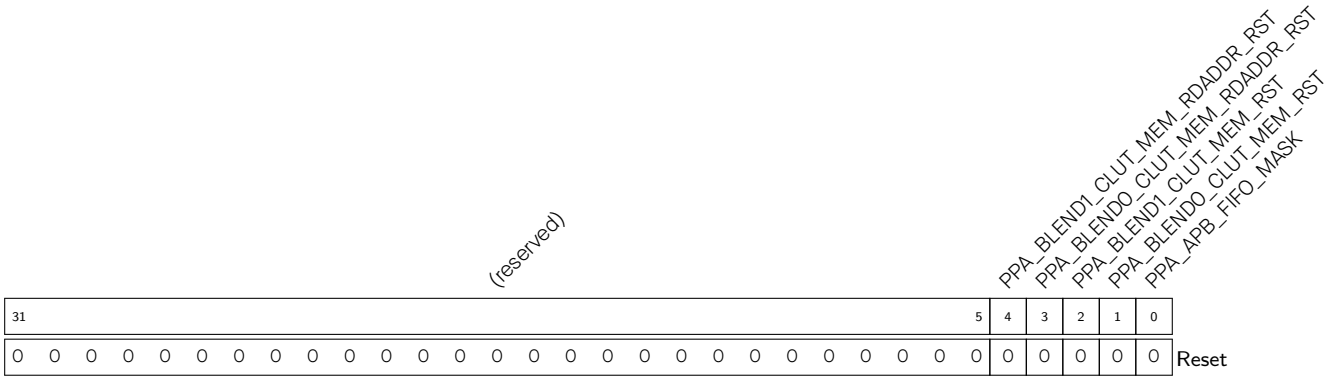
PPA_RDWR_WORD_BLEND0_CLUT Configures BLEND background layer CLUT access in FIFO mode. (R/W)

Register 18.2. PPA_BLEND1_CLUT_DATA_REG (0x0004)



PPA_RDWR_WORD_BLEND1_CLUT Configures BLEND foreground layer CLUT access in FIFO mode. (R/W)

Register 18.3. PPA_CLUT_CONF_REG (0x000C)



PPA_APB_FIFO_MASK Configures the access mode of BLEND CLUT

1'b0: Access CLUT in FIFO mode. For details, please refer to 18.5.1.2.

1'b1: Access CLUT in MEM mode. For details, please refer to 18.5.1.2.

(R/W)

PPA_BLEND0_CLUT_MEM_RST Configures whether to reset BLEND0 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND1_CLUT_MEM_RST Configures whether to reset BLEND1 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND0_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND0 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

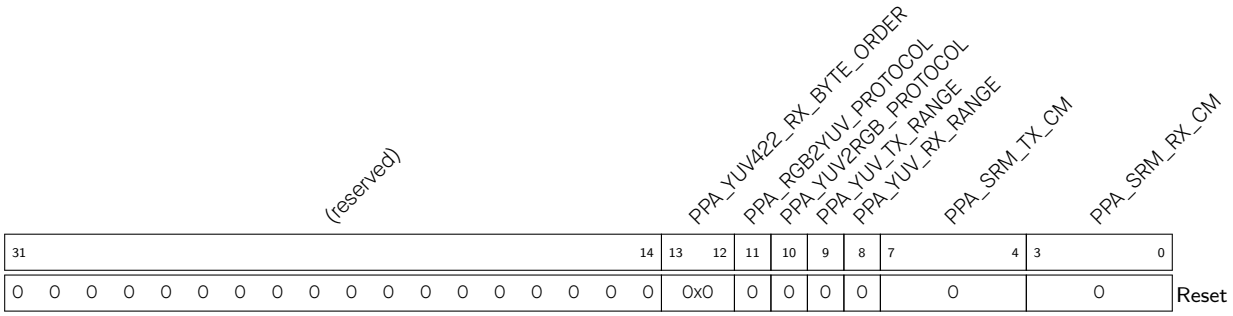
PPA_BLEND1_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND1 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

Register 18.4. PPA_SRM_COLOR_MODE_REG (0x0020)



PPA_SRM_RX_CM Configures the input image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

9: YUV422

12: GRAY

Others: Reserved

(R/W)

PPA_SRM_TX_CM Configures the output image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

9: YUV422

12: GRAY

Others: Reserved

(R/W)

PPA_YUV_RX_RANGE Configures the YUV range when the SRM input format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV_TX_RANGE Configures the YUV range when the SRM output format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV2RGB_PROTOCOL Configures the protocol used for the SRM input direction YUV to RGB.

0: BT601

1: BT709

(R/W)

PPA_RGB2YUV_PROTOCOL Configures the protocol used for the SRM output direction RGB to YUV.

0: BT601

1: BT709

(R/W)

Continued on the next page...

Register 18.4. PPA_SRM_COLOR_MODE_REG (0x0020)

Continued from the previous page...

PPA_YUV422_RX_BYTE_ORDER Configures the byte order when the SRM input format is YUV422.

0: YVYU

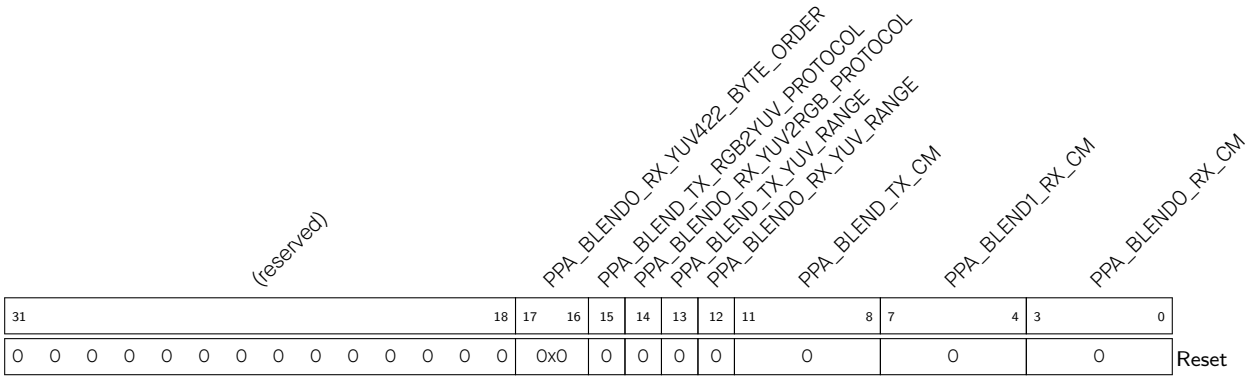
1: YUYV

2: VYUY

3: UYVY

(R/W)

Register 18.5. PPA_BLEND_COLOR_MODE_REG (0x0024)



PPA_BLEND0_RX_CM Configures the input image color format for BLEND background layer.

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
8: YUV420
9: YUV422
12: GRAY
Others: Reserved
(R/W)

PPA_BLEND1_RX_CM Configures the input image color format for BLEND foreground layer.

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
6: A8
7: A4
Others: Reserved
(R/W)

PPA_BLEND_TX_CM Configures the output image color format of BLEND.

0: ARGB8888
1: RGB888
2: RGB565
8: YUV420
9: YUV422
12: GRAY
Others: Reserved
(R/W)

Continued on the next page...

Register 18.5. PPA_BLEND_COLOR_MODE_REG (0x0024)

Continued from the previous page...

PPA_BLEND0_RX_YUV_RANGE Configures the YUV range when the BLEND background input is YUV.
0: Limited range
1: Full range
(R/W)

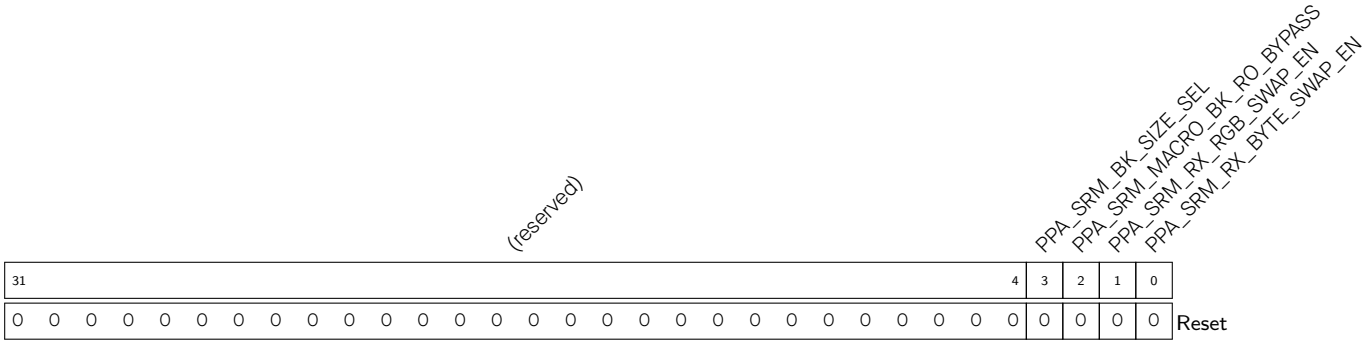
PPA_BLEND_TX_YUV_RANGE Configures the YUV range when the BLEND output is YUV.
0: Limited range
1: Full range
(R/W)

PPA_BLEND0_RX_YUV2RGB_PROTOCOL Configures the YUV to RGB conversion protocol for the BLEND background input.
0: BT601
1: BT709
(R/W)

PPA_BLEND_TX_RGB2YUV_PROTOCOL Configures the RGB to YUV conversion protocol for the BLEND output.
0: BT601
1: BT709
(R/W)

PPA_BLEND0_RX_YUV422_BYTE_ORDER Configures the byte order when the BLEND background input is YUV422.
0: YVYU
1: YUYV
2: VYUY
3: UYVY
(R/W)

Register 18.6. PPA_SRM_BYTE_ORDER_REG (0x0028)



PPA_SRM_RX_BYTE_SWAP_EN Configures whether to swap the SRM valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_SRM_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the SRM input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

PPA_SRM_MACRO_BK_RO_BYPASS Configures whether to enable the pixel block order function

0: Enable and automatically calculate if the current block needs to be reordered

1: Disable

(R/W)

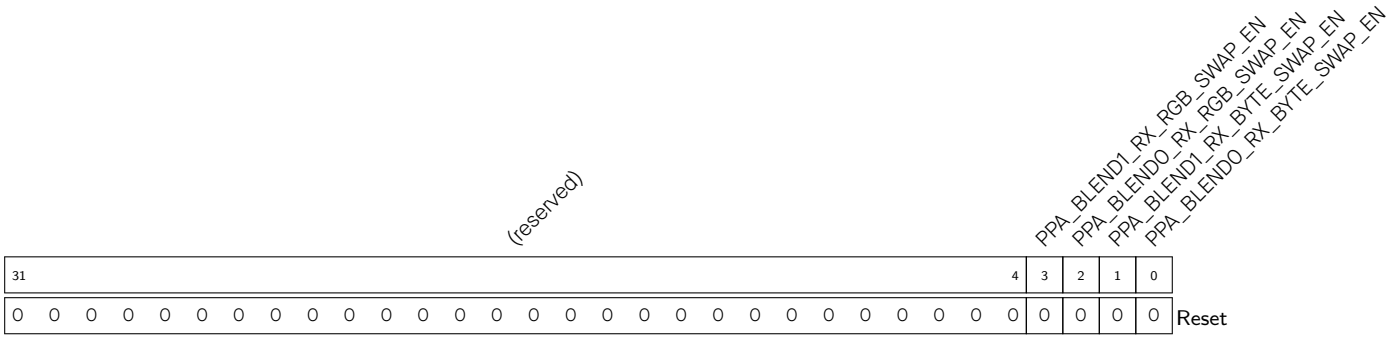
PPA_SRM_BK_SIZE_SEL Configures the size of SRM pixel blocks.

0: 32x32

1: 16x16

(R/W)

Register 18.7. PPA_BLEND_BYTE_ORDER_REG (0x002C)



PPA_BLEND0_RX_BYTE_SWAP_EN Configures whether to swap the BLEND background layer valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_BLEND1_RX_BYTE_SWAP_EN Configures whether to swap the BLEND foreground layer valid data input in byte.

0: Not swap

1: Swap

(R/W)

PPA_BLEND0_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the BLEND background layer input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

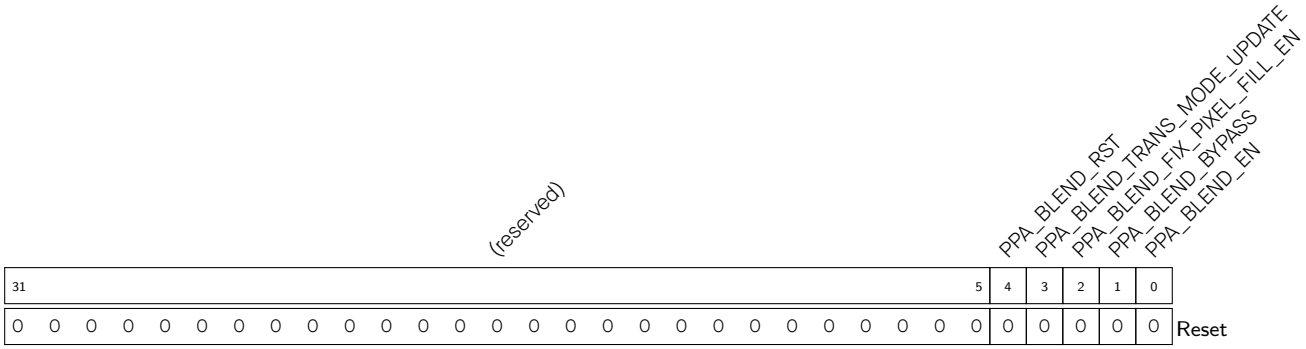
PPA_BLEND1_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the BLEND foreground layer input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

Register 18.8. PPA_BLEND_TRANS_MODE_REG (0x0034)



PPA_BLEND_EN Configures whether to enable BLEND.

0: Disable

1: Enable

(R/W)

PPA_BLEND_BYPASS Configures whether to bypass BLEND and directly output the background layer.

0: Not bypass

1: Bypass

(R/W)

PPA_BLEND_FIX_PIXEL_FILL_EN Configures whether to enable filled image output.

0: Disable

1: Enable, where only BLEND TX works and the output pixel and size are configured by PPA_BLEND_TX_SIZE_REG and PPA_BLEND_TX_FIX_PIXEL.

(R/W)

PPA_BLEND_TRANS_MODE_UPDATE Write 1 to update the transfer mode, active PPA_BLEND_FIX_PIXEL_FILL_EN, and start BLEND workflow. (WT)

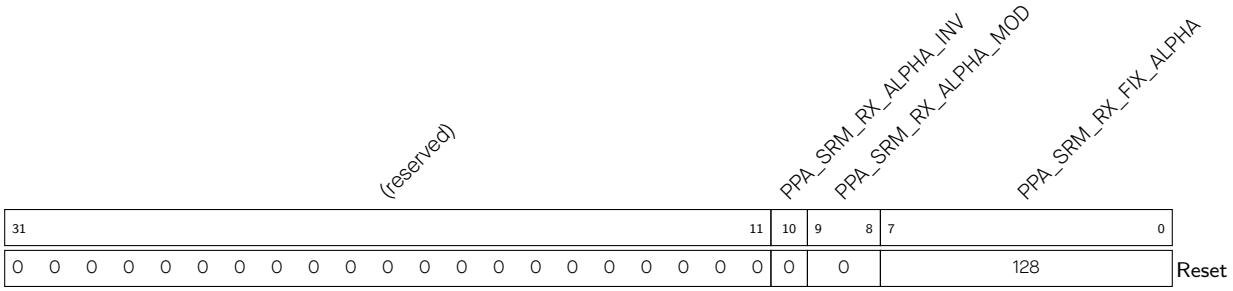
PPA_BLEND_RST Configures whether to reset BLEND.

0: Release reset

1: Reset

(R/W)

Register 18.9. PPA_SRM_FIX_ALPHA_REG (0x0038)



PPA_SRM_RX_FIX_ALPHA Configures the Alpha channel value of SRM. (R/W)

PPA_SRM_RX_ALPHA_MOD Configures the Alpha mode of SRM input data.

1: Use the value of PPA_SRM_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_SRM_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_SRM_RX_ALPHA_INV Configures whether to invert the original Alpha value of SRM.

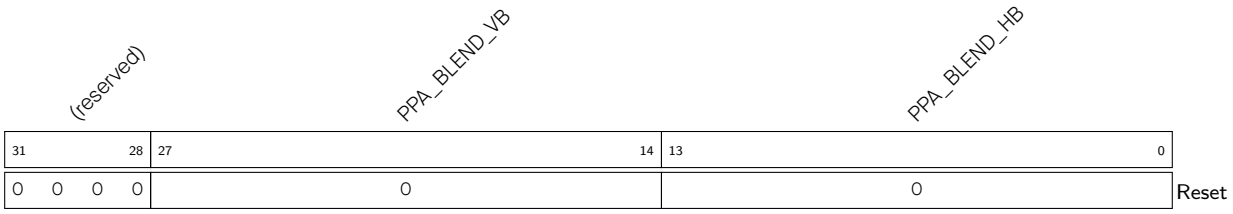
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

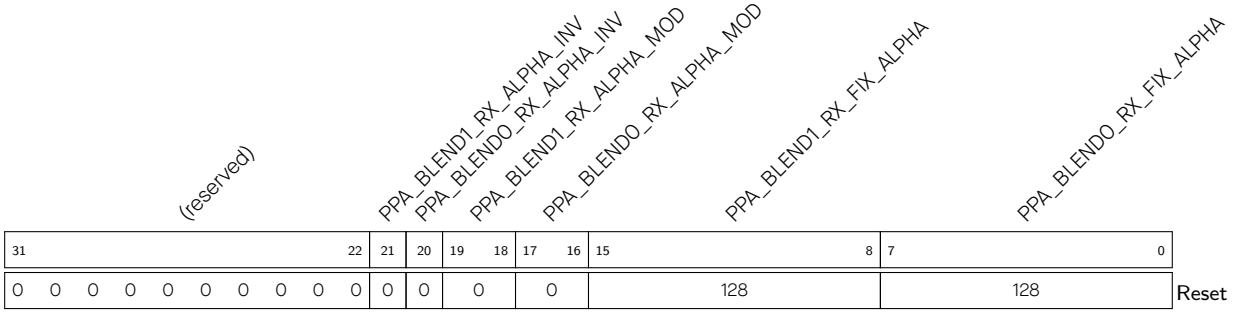
Register 18.10. PPA_BLEND_TX_SIZE_REG (0x003C)



PPA_BLEND_HB Configures the horizontal width of image block that would be filled in the image filling mode. Measurement unit: pixel. (R/W)

PPA_BLEND_VB Configures the vertical width of image block that would be filled in the image filling mode. Measurement unit: pixel. (R/W)

Register 18.11. PPA_BLEND_FIX_ALPHA_REG (0x0040)



PPA_BLEND0_RX_FIX_ALPHA Configures the Alpha channel value of the BLEND background layer. (R/W)

PPA_BLEND1_RX_FIX_ALPHA Configures the Alpha channel value of the BLEND foreground layer. (R/W)

PPA_BLEND0_RX_ALPHA_MOD Configures the Alpha mode of BLEND background layer data.

1: Use the value of PPA_BLEND0_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND0_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND1_RX_ALPHA_MOD Configures the Alpha mode of BLEND foreground layer data.

1: Use the value of PPA_BLEND1_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND1_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND0_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND background layer data.

0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

PPA_BLEND1_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND foreground layer data.

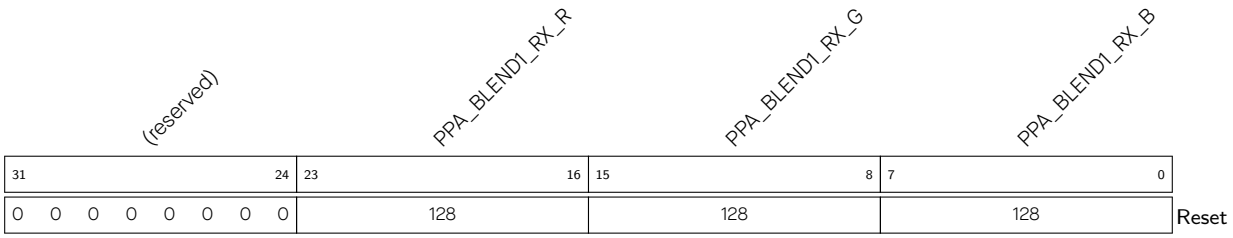
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

Register 18.12. PPA_BLEND_RGB_REG (0x0048)

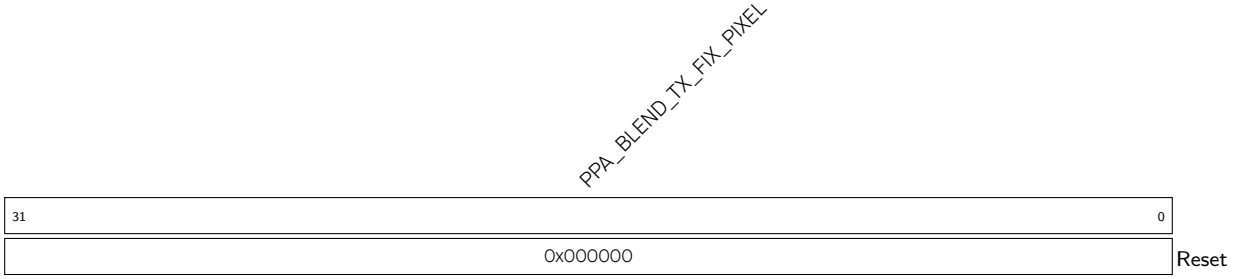


PPA_BLEND1_RX_B Configures the B channel value when the BLEND foreground layer is in A4/A8. (R/W)

PPA_BLEND1_RX_G Configures the G channel value when the BLEND foreground layer is in A4/A8. (R/W)

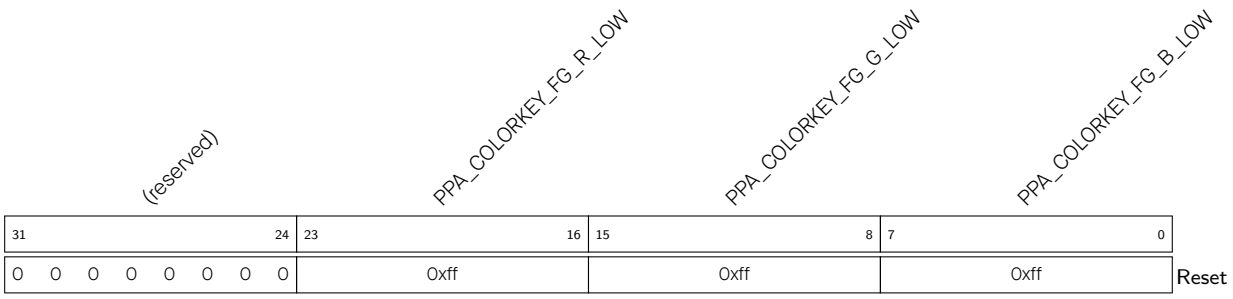
PPA_BLEND1_RX_R Configures the R channel value when the BLEND foreground layer is in A4/A8. (R/W)

Register 18.13. PPA_BLEND_FIX_PIXEL_REG (0x004C)



PPA_BLEND_TX_FIX_PIXEL Configures the pixel in the BLEND image filling mode.
[31:24]: A, [23:16]:R, [15:8]: G, [7:0]: B.
(R/W)

Register 18.14. PPA_CK_FG_LOW_REG (0x0050)

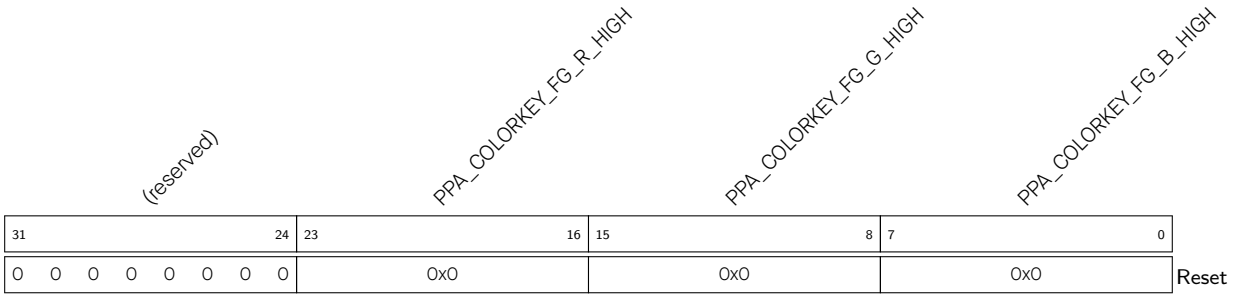


PPA_COLORKEY_FG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_G_LOW Configures the color-key lower threshold of the G channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_R_LOW Configures the color-key lower threshold of the R channel value of the BLEND foreground layer. (R/W)

Register 18.15. PPA_CK_FG_HIGH_REG (0x0054)

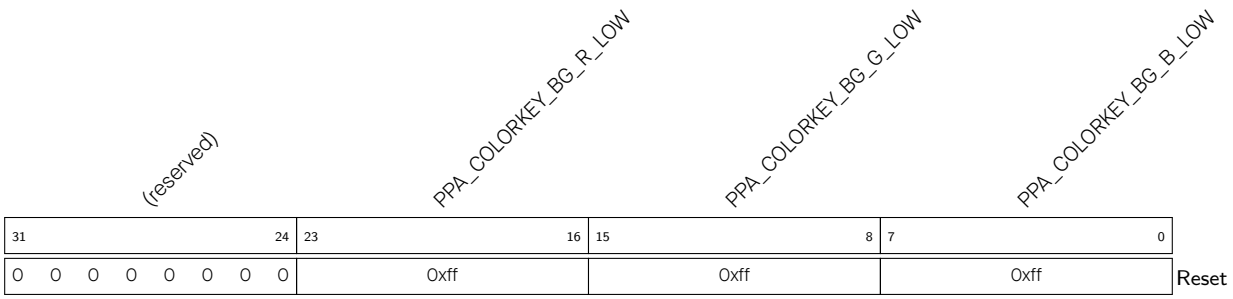


PPA_COLORKEY_FG_B_HIGH Configures the color-key higher threshold of the B channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_G_HIGH Configures the color-key higher threshold of the G channel value of the BLEND foreground layer. (R/W)

PPA_COLORKEY_FG_R_HIGH Configures the color-key higher threshold of the R channel value of the BLEND foreground layer. (R/W)

Register 18.16. PPA_CK_BG_LOW_REG (0x0058)

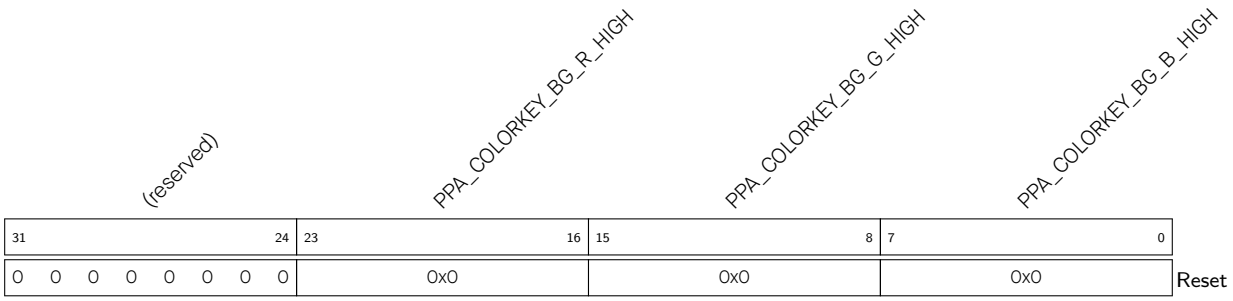


PPA_COLORKEY_BG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_G_LOW Configures the color-key lower threshold of the G channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_R_LOW Configures the color-key lower threshold of the R channel value of the BLEND background layer. (R/W)

Register 18.17. PPA_CK_BG_HIGH_REG (0x005C)



PPA_COLORKEY_BG_B_HIGH Configures the color-key higher threshold of the B channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_G_HIGH Configures the color-key higher threshold of the G channel value of the BLEND background layer. (R/W)

PPA_COLORKEY_BG_R_HIGH Configures the color-key higher threshold of the R channel value of the BLEND background layer. (R/W)

Register 18.18. PPA_CK_DEFAULT_REG (0x0060)



PPA_COLORKEY_DEFAULT_B Configures the default B channel value of BLEND color-key. (R/W)

PPA_COLORKEY_DEFAULT_G Configures the default G channel value of BLEND color-key. (R/W)

PPA_COLORKEY_DEFAULT_R Configures the default R channel value of BLEND color-key. (R/W)

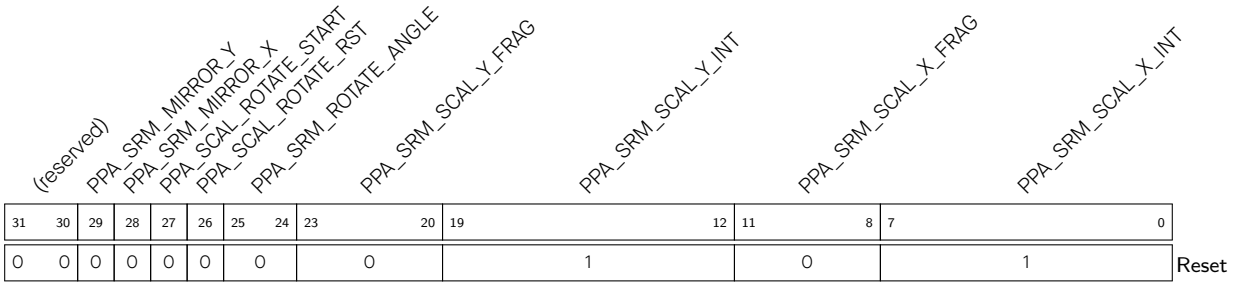
PPA_COLORKEY_FG_BG_REVERSE Configures the workflow when when a pixel is within the background color-key range but not within the foreground color-key range.

0: Output background pixel

1: Output foreground pixel

(R/W)

Register 18.19. PPA_SRM_SCAL_ROTATE_REG (0x0064)



PPA_SRM_SCAL_X_INT Configures the integer part of the horizontal scaling factor for SRM. (R/W)

PPA_SRM_SCAL_X_FRAG Configures the fragment part of the horizontal scaling factor for SRM. (R/W)

PPA_SRM_SCAL_Y_INT Configures the integer part of the vertical scaling factor for SRM. (R/W)

PPA_SRM_SCAL_Y_FRAG Configures the fragment part of the vertical scaling factor for SRM. (R/W)

PPA_SRM_ROTATE_ANGLE Configures the counterclockwise rotation angle for the SRM.

0: 0 degree
1: 90 degree
2: 180 degree
3: 270 degree
(R/W)

PPA_SCAL_ROTATE_RST Configures whether to reset SRM.

0: Release reset
1: Reset
(R/W)

PPA_SCAL_ROTATE_START Write 1 to enable SRM. (WT)

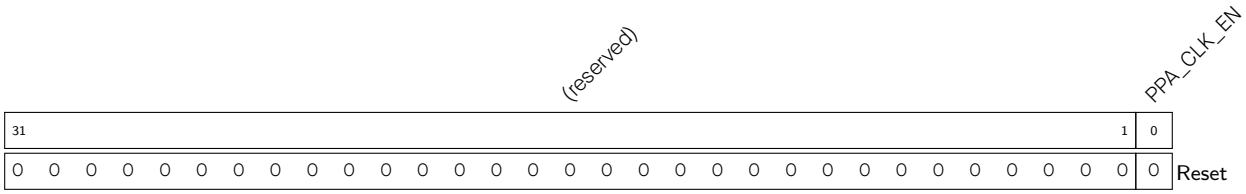
PPA_SRM_MIRROR_X Configures whether to enable the horizontal mirroring for SRM.

0: Disable
1: Enable
(R/W)

PPA_SRM_MIRROR_Y Configures whether to enable the vertical mirroring for SRM.

0: Disable
1: Enable
(R/W)

Register 18.20. PPA_REG_CONF_REG (0x006C)



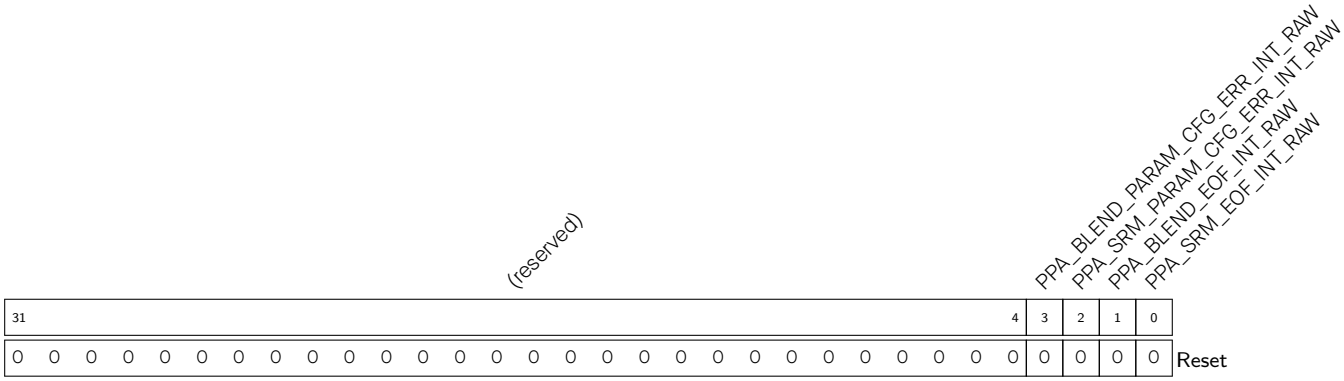
PPA_CLK_EN Configures whether to keep the PPA register clock always on.

0: Clock only turns on when there's a register access

1: Clock always on

(R/W)

Register 18.21. PPA_INT_RAW_REG (0x0010)



PPA_SRM_EOF_INT_RAW The raw interrupt status of PPA_SRM_EOF_INT.

(R/WTC/SS)

PPA_BLEND_EOF_INT_RAW The raw interrupt status of PPA_BLEND_EOF_INT.

(R/WTC/SS)

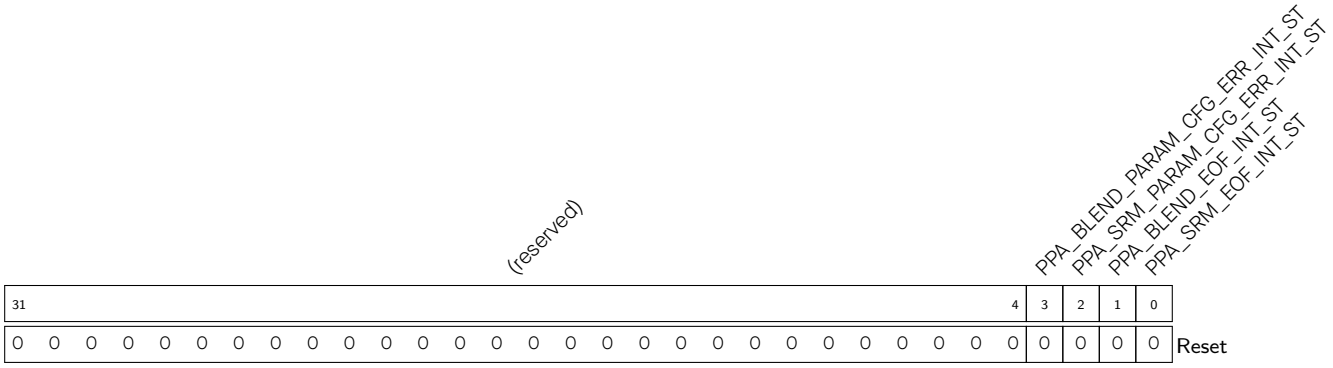
PPA_SRM_PARAM_CFG_ERR_INT_RAW The raw interrupt status of PPA_SRM_PARAM_CFG_ERR_INT. Check PPA_SRM_PARAM_ERR_ST_REG to get the specific error.

(R/WTC/SS)

PPA_BLEND_PARAM_CFG_ERR_INT_RAW The raw interrupt status of PPA_BLEND_PARAM_CFG_ERR_INT. Check PPA_BLEND_ST_REG to get the specific error.

(R/WTC/SS)

Register 18.22. PPA_INT_ST_REG (0x0014)



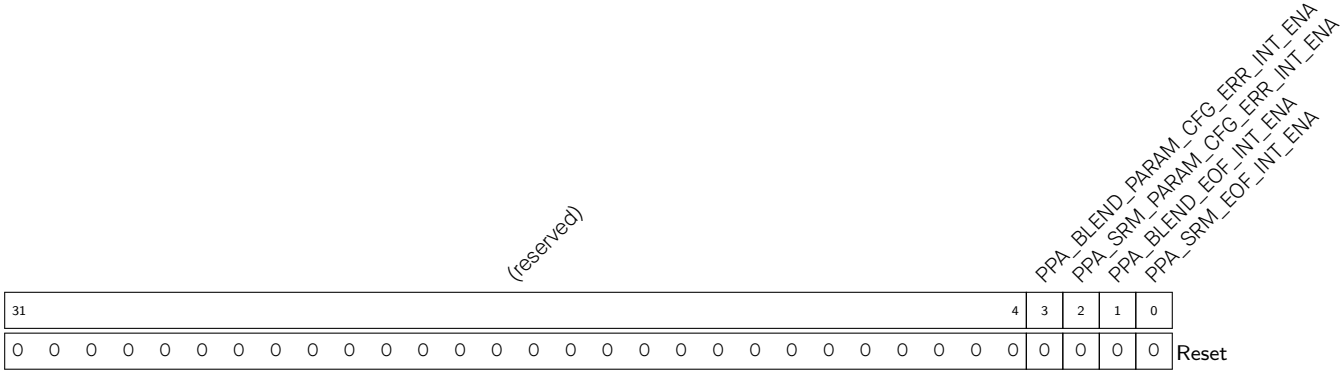
PPA_SRM_EOF_INT_ST The masked interrupt status of PPA_SRM_EOF_INT. (RO)

PPA_BLEND_EOF_INT_ST The masked interrupt status of PPA_BLEND_EOF_INT. (RO)

PPA_SRM_PARAM_CFG_ERR_INT_ST The masked interrupt status of PPA_SRM_PARAM_CFG_ERR_INT. (RO)

PPA_BLEND_PARAM_CFG_ERR_INT_ST The masked interrupt status of PPA_BLEND_PARAM_CFG_ERR_INT. (RO)

Register 18.23. PPA_INT_ENA_REG (0x0018)



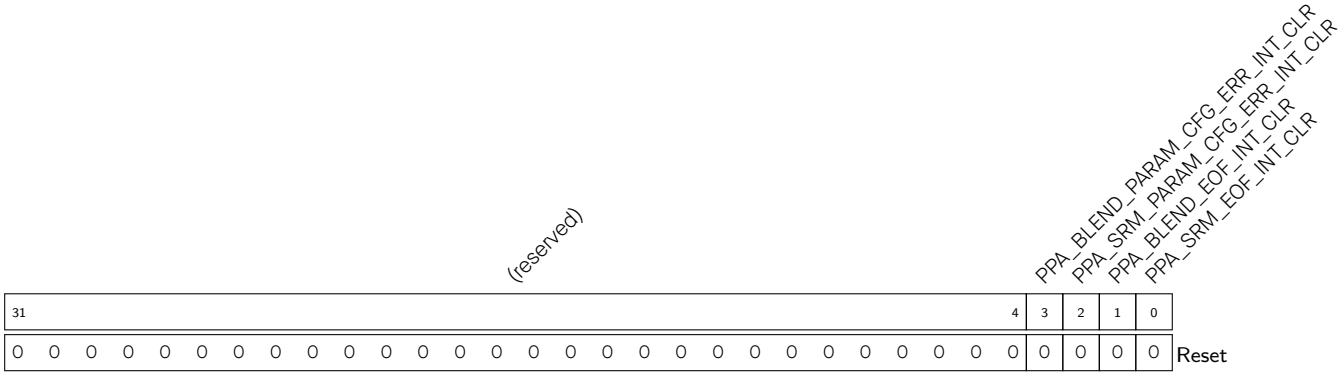
PPA_SRM_EOF_INT_ENA Write 1 to enable PPA_SRM_EOF_INT. (R/W)

PPA_BLEND_EOF_INT_ENA Write 1 to enable PPA_BLEND_EOF_INT. (R/W)

PPA_SRM_PARAM_CFG_ERR_INT_ENA Write 1 to enable PPA_SRM_PARAM_CFG_ERR_INT. (R/W)

PPA_BLEND_PARAM_CFG_ERR_INT_ENA Write 1 to enable PPA_BLEND_PARAM_CFG_ERR_INT. (R/W)

Register 18.24. PPA_INT_CLR_REG (0x001C)



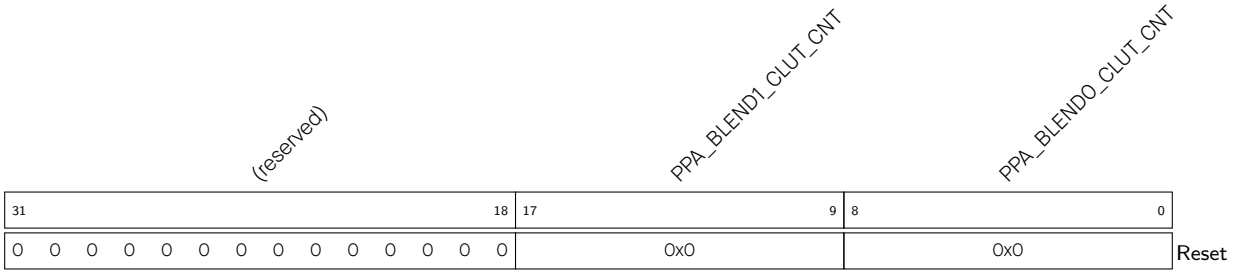
PPA_SRM_EOF_INT_CLR Write 1 to clear PPA_SRM_EOF_INT. (WT)

PPA_BLEND_EOF_INT_CLR Write 1 to clear PPA_BLEND_EOF_INT. (WT)

PPA_SRM_PARAM_CFG_ERR_INT_CLR Write 1 to clear PPA_SRM_PARAM_CFG_ERR_INT. (WT)

PPA_BLEND_PARAM_CFG_ERR_INT_CLR □ 1 □□ PPA_BLEND_PARAM_CFG_ERR_INT□ (WT)

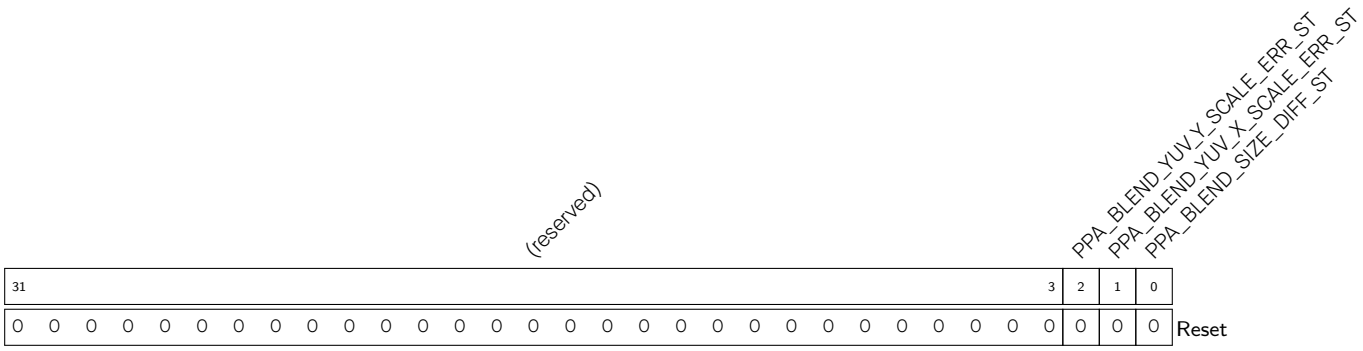
Register 18.25. PPA_CLUT_CNT_REG (0x0070)



PPA_BLEND0_CLUT_CNT Represents the current write address of the BLEND background layer CLUT in FIFO mode. (RO)

PPA_BLEND1_CLUT_CNT Represents the current write address of the BLEND foreground layer CLUT in FIFO mode. (RO)

Register 18.26. PPA_BLEND_ST_REG (0x0074)



PPA_BLEND_SIZE_DIFF_ST Represents whether the size of the BLEND background and foreground images are consistent.

0: Consistent

1: Different

(RO)

PPA_BLEND_YUV_X_SCALE_ERR_ST Represents the image width (X) is odd for YUV422 or YUV420 formats. (RO)

PPA_BLEND_YUV_Y_SCALE_ERR_ST Represents the image height (Y) is odd for the YUV420 format. (RO)

Register 18.27. PPA_SRM_PARAM_ERR_ST_REG (0x0078)



PPA_TX_DSCR_VB_ERR_ST Represents whether the vertical size of the SRM output image block plus the vertical offset Y configured in the output list exceeds the vertical size of the entire image configured in the output list, VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_TX_DSCR_HB_ERR_ST Represents whether the horizontal size of the SRM output image block plus the horizontal offset X configured in the output list exceeds the horizontal size of the entire image configured in the output list, HA.

0: Not exceeded

1: Exceeded

(RO)

PPA_Y_RX_SCAL_EQUAL_0_ERR_ST Represents whether the vertical direction scaling factor of the SRM input image block is 0.

0: Not 0

1: 0

(RO)

PPA_RX_DSCR_VB_ERR_ST Represents whether the sum of the vertical direction image block size VB configured in the SRM input list and the vertical offset Y exceeds the entire image's vertical size VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_YDST_LEN_TOO_SAMLL_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is 0.

0: Not 0

1: 0

(RO)

Continued on the next page...

Register 18.27. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_YDST_LEN_TOO_LARGE_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_RX_SCAL_EQUAL_0_ERR_ST Represents whether the horizontal direction scaling factor of the SRM input image block is 0.
0: Not 0
1: 0
(RO)

PPA_RX_DSCR_HB_ERR_ST Represents whether the sum of the horizontal direction image block size HB configured in the SRM input list and the horizontal offset X exceeds the entire image's horizontal size HA.
0: Not exceeded
1: Exceeded
(RO)

PPA_XDST_LEN_TOO_SAMLL_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is 0.
0: Not 0
1: 0
(RO)

PPA_XDST_LEN_TOO_LARGE_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_YUV_RX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM input list are odd when the input format is YUV420/YUV422.
0: Even
1: Odd
(RO)

PPA_Y_YUV_RX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM input list are odd when the input format is YUV420.
0: Even
1: Odd
(RO)

Continued on the next page...

Register 18.27. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_X_YUV_TX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM output list are odd when the output format is YUV420/YUV422.

0: Even

1: Odd

(RO)

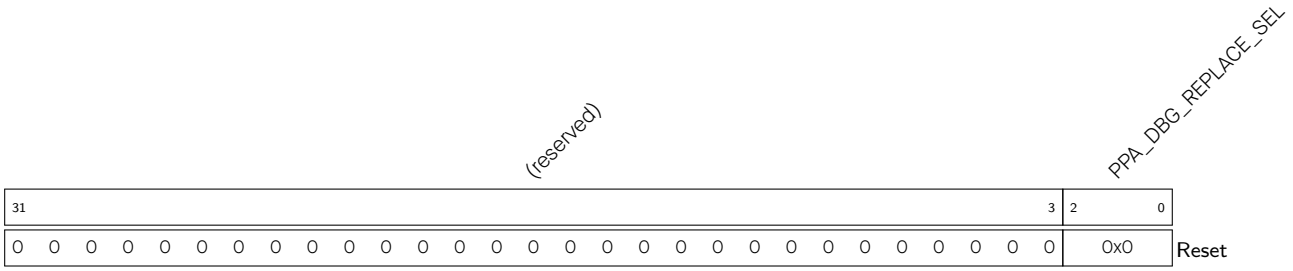
PPA_Y_YUV_TX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM output list are odd when the output format is YUV420.

0: Even

1: Odd

(RO)

Register 18.28. PPA_DEBUG_CTRL0_REG (0x0090)



PPA_DBG_REPLACE_SEL Configures where the replacement pixel is inserted.

0: No replacement

1: SRM RX input

2: SRM interpolation output

3: SRM TX output

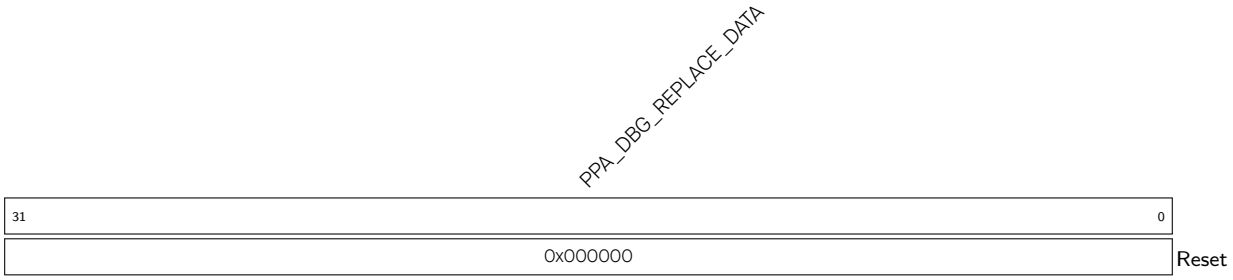
4: BLEND foreground input

5: BLEND background input

6: BLEND output

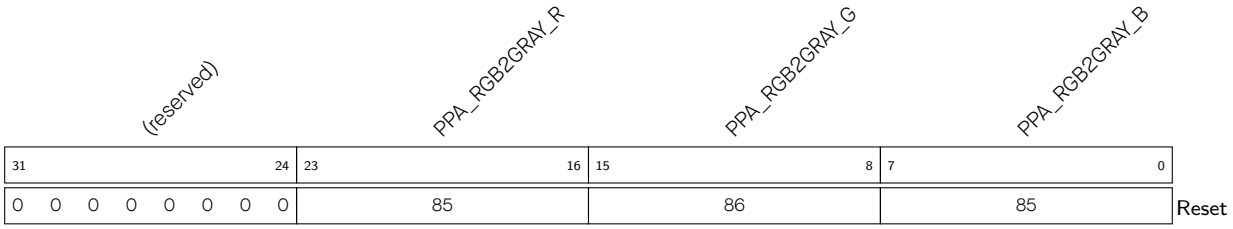
(R/W)

Register 18.29. PPA_DEBUG_CTRL1_REG (0x0094)



PPA_DBG_REPLACE_DATA Configures the replacement pixel value. (R/W)

Register 18.30. PPA_RGB2GRAY_REG (0x0098)

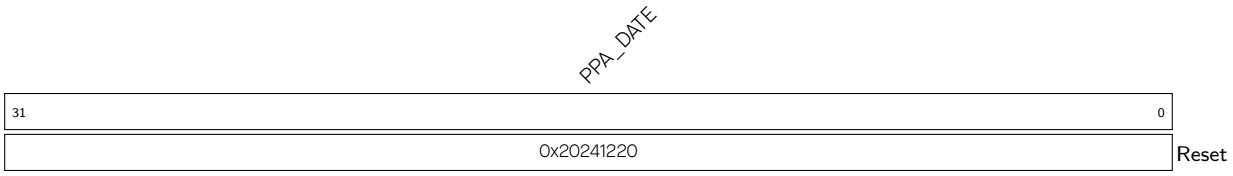


PPA_RGB2GRAY_B Configures the coefficient for B in RGB to GRAY conversion. (R/W)

PPA_RGB2GRAY_G Configures the coefficient for G in RGB to GRAY conversion. (R/W)

PPA_RGB2GRAY_R Configures the coefficient for R in RGB to GRAY conversion. (R/W)

Register 18.31. PPA_DATE_REG (0x0100)



PPA_DATE Version control register. (R/W)

Chapter 19

Audio Sample Rate Converter

19.1 Overview

Audio Sample Rate Converter (ASRC) is an accelerator-type audio processing module used for high-quality conversion of audio signals between different sample rates. It enables collaboration between devices with varying sample rate standards. This module typically interacts with on-chip or off-chip memory through a DMA interface to achieve efficient transmission and processing of audio data.

19.2 Terminology

To better explain the functionality of the ASRC module, the following terms are used in this chapter:

Audio Sample Rate Converter (ASRC)	A module that converts audio signals between different sample rates
Integer Resampler	Performs sample rate conversion by fixed integer multiples
Fractional Sample Rate Converter (FRC)	Performs sample rate conversion by non-integer ratios
Interpolation	The process of inserting new samples between existing ones to increase the sample rate
Decimation	The process of discarding samples at fixed ratio to reduce the sample rate
Upsampling	The process of increasing the sample rate, typically involving interpolation and filtering
Downsampling	The process of decreasing the sample rate, typically involving decimation and filtering
Filtering	The process of attenuating unwanted frequency components such as aliasing or interpolation noise
Sample	The basic unit of audio data, typically 2 bytes in size
End of Frame (EOF)	A status flag or interrupt indicating the completion of processing a frame of audio data

19.3 Features

The ASRC module in ESP32-S31 has the following features:

- Supports conversion between 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, and 48 kHz
- Input and output signal width is 16 bits, using Q1.15 format

- Sample rate conversion uses two integer resamplers and one fractional sample rate converter (FRC) in a cascade, with the cascade configurable via software:
  - The integer resampler's conversion factor can be set to 2 or $\frac{1}{2}$
  - The FRC supports conversion factors configurable between 0 and 2 (excluding 0 and 2)
- Supports multiple channel modes, including:
  - Mono receive (Rx) / Mono transmit (Tx)
  - Stereo receive (Rx) / Stereo transmit (Tx), processed in parallel
  - Mono receive / Stereo transmit
  - Stereo receive / Mono transmit
- Supports input data of any length, and generates EOF (End of Frame) flags based on output length
- When two channels are used independently, each channel supports the following modes:
  - Mono input, mono output
  - Mono input, stereo output (both channels use the same data)
  - Stereo input, mono output (choose one channel for processing)
- When both channels are used together, each channel can be configured as stereo input or output

19.4 Architectural Overview

The system architecture of the ASRC module in ESP32-S31 is shown in Figure 19.4-1.

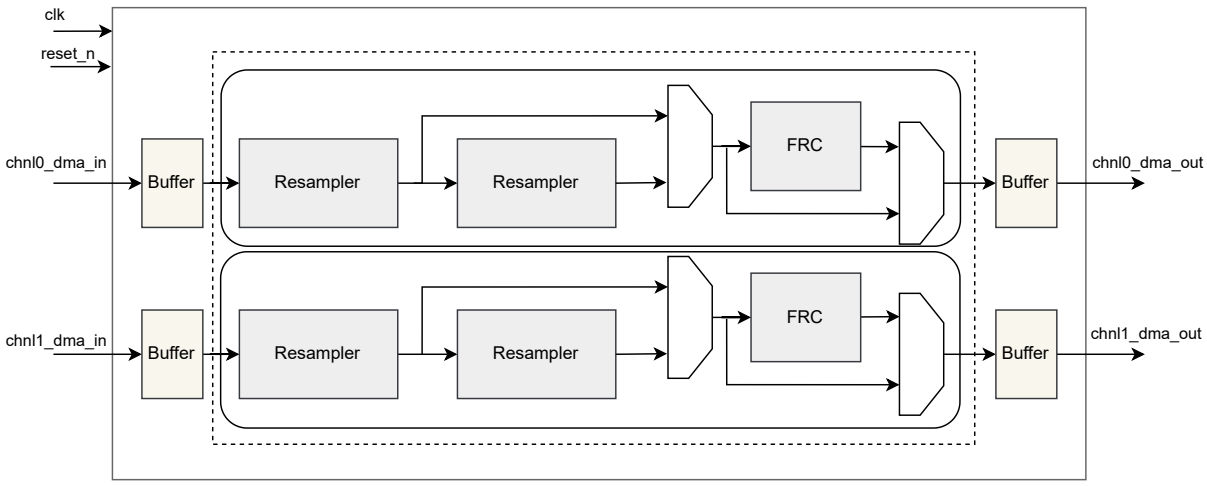


Figure 19.4-1. ASRC Module Structure

- Data In Path: ch0_dma_in and ch1_dma_in receive input from the two TX channels of the DMA module.
- Data Out Path: ch0_dma_out and ch1_dma_out output results to the two RX channels of the DMA module.
- Clock structure: The module has only one clock clk.
- DMA Transfer Process:

  - After configuring mode parameters and starting the corresponding channels, all processing units of the ASRC enter a ready state.
  - Once the TX and RX DMA channel descriptor lists are configured, the first stage conversion unit reads input data from DMA, processes it, and writes results back to DMA.
  - If TX input data or RX space is insufficient, the conversion process automatically pauses until resources become available.

- Sample Rate Conversion Structure:
  - The module contains two conversion channels, CHNL0 and CHNL1, which can simultaneously process two audio channels.
  - Each channel structure includes two integer resamplers (Resampler) and one FRC.
  - The Resampler supports configuration for either $2\times$ upsampling or $\frac{1}{2}\times$ downsampling.
  - The FRC performs non-integer sample rate conversions.
  - The cascade configuration of the two Resamplers and the FRC can be flexibly controlled via registers.

- Channel Operation Modes:
  - Independent mode: Each channel can be configured separately with its own DMA interface.
  - Cooperative mode: Both channels share the same configuration, use CHNL0's DMA interface, and only CHNL0 generates interrupts.

19.5 Functional Description

19.5.1 Clock

As shown in Figure 19.4-1, the clock signal clk is used for register configuration, data processing, and transmission.

For more information about the clock, please refer to Section 7 *Reset and Clock [to be added later]*.

For detailed configuration procedures of the clock, please see Section 19.7 *Configuration Procedure*.

19.5.2 Reset

ASRC supports full reset and partial reset:

- Full reset: Resets the entire ASRC module, including registers and the two data processing channels CHNL0 and CHNL1.
- Partial reset: Resets specific components in the module. See Section 19.7.1 *Usage Restrictions and Notes* for details.

19.5.3 Resampler

To improve filter processing parallelism and reduce data latency, the ASRC module adopts an architecture consisting of two multiply-accumulate units and three adders. The module supports $2\times$ upsampling and $\frac{1}{2}\times$

downsampling, with flexible mode configuration via control registers.

By cascading two identical resamplers, $4\times$ upsampling or $\frac{1}{4}\times$ downsampling can be achieved. However, mixed configurations (e.g., "$2\times$ upsampling + $\frac{1}{2}\times$ downsampling" or "$\frac{1}{2}\times$ downsampling + $2\times$ upsampling") are prohibited, as they may lead to undefined behavior or degraded performance.

19.5.4 Fractional Sample Rate Converter

The fractional sample rate converter supports arbitrary sample rate conversions, with its conversion factor expressed as $\frac{L}{M} \in (0, 2)$, where L is the interpolation factor and M is the decimation factor. To ensure conversion accuracy and performance, both L and M must be positive integers and coprime; if they are not coprime, it will affect the conversion quality.

Before the module operates, the following register configurations are required:

- Write the interpolation factor L to register ASRC_CHNL*n*_FRAC_L
- Write the decimation factor M to register ASRC_CHNL*n*_FRAC_M
- Write the value $\left\lfloor \frac{2^{20}+L}{2L} \right\rfloor$ to register ASRC_CHNL*n*_FRAC_RECIPL, which is used to approximate $1/L$

Note: The *n* in the text refers to the channel number (0 or 1). This notation will not be repeated in the following content.

19.5.5 Input/Output Data Length

The ASRC module is used to implement sample rate conversion functions, supporting three modes: increasing, decreasing, and maintaining the sample rate. The case where the sample rate remains unchanged should be achieved by configuring the selector to bypass mode. Sample rate changes affect output data length:

- When the sample rate increases, the output data length increases.
- When the sample rate decreases, the output data length decreases.
- When the sample rate remains unchanged, the output data length remains unchanged.

Let F_i be the input sample rate, F_o the output sample rate, and N_i the input data length. The output data length N_o is:

$$N_o = \lfloor xN_i \rfloor, \quad \text{where } x = \frac{F_o}{F_i}, \quad x \in \mathbb{Q}^+$$

For example, if $F_i = 8\,\text{kHz}$, $F_o = 44.1\,\text{kHz}$, and $N_i = 128$ (unit: samples), then:

$$N_o = \left\lfloor \frac{F_o}{F_i} \cdot N_i \right\rfloor = \left\lfloor \frac{44.1}{8} \cdot 128 \right\rfloor = 705 \text{ (samples)}$$

Note:

- In mono mode, the input data length N_i should be an integer multiple of 1 sample (i.e., 2 bytes).
- In stereo mode, N_i should be an integer multiple of 2 samples (i.e., 4 bytes), and the data length of both channels must be equal.
- All input/output data lengths in this document are in units of samples.

19.5.5.1 No Residual Data

Each of the three cascaded ASRC submodules produces output data based on its input length and the configured sample rate conversion ratio. Supported conversion ratios include rational numbers $\frac{L}{M}$, 2, or $\frac{1}{2}$.

For a single-stage structure, if the conversion ratio $\frac{L}{M} < 1$ or equals $\frac{1}{2}$, the input length must be no less than $\lceil \frac{M}{L} \rceil$ or 2 samples to avoid residual data during processing.

For a cascade structure, the input data length N_i should be no less than $4 \times \lceil \frac{M}{L} \rceil$ to ensure no data remains in any submodule.

19.5.5.2 Residual Data Handling

When the input frequency is greater than the output frequency, special consideration must be given to the issue of input data residue. The following example illustrates this for two-times downsampling:

In two-times downsampling, the system generates 1 output sample for every 2 input samples received. If the number of input data is odd (e.g., 1), it will be impossible to produce an output, resulting in the last sample remaining in the system. This residue may affect the organization and parsing of data packets by upper-level applications.

If the condition of no data residue for each module's input cannot be met, the hardware output data compensation function can be enabled by configuring the register ASRC_CHNL*n*_OUT_LEN_COMP to 1. When hardware detects that the output data length has reached the configured register ASRC_CHNL*n*_OUT_LEN (in bytes) and continues to produce new data, the system will add 2 bytes (mono) or 4 bytes (stereo) to the length in the next transmission.

When the compensation function is enabled, increase the DMA reception buffer by at least 2 or 4 bytes to prevent overflow.

After enabling compensation, notify the ASRC controller that all input data has been sent by either of the following methods:

- Enable the DMA EOF feature
- Set the ASRC reception data length via ASRC_CHNL*n*_DIN_CFG_REG

Inappropriate configuration of this may lead to compensation errors and affect data transmission stability.

19.5.6 DMA Interface

The ASRC module uses a DMA interface for input/output.

Once the output reaches the specified length, an EOF signal will be generated and sent to DMA. The output length is configured by the software according to the formula described in 19.5.5 *Input/Output Data Length*. After generating EOF, the system waits for input.

When producing EOF by selecting the software configuration output length, once the output count reaches the specified value, the EOF will be written to DMA.

19.6 Interrupts

Each ASRC channel can generate a level interrupt when the output data length count reaches the set value.

If the output/input frequency ratio of the ASRC module is x, and the data length received at EOF is N_i, then the interrupt will be triggered when the total output data length is $\lfloor xN_i \rfloor$.

The audio sample rate converter of ESP32-S31 can generate the following interrupt signals that will be sent to the interrupt matrix:

- ASRC_CHNL0_INT
- ASRC_CHNL1_INT

These interrupt signals are generated by the internal interrupt sources of ASRC. Table 19.6-1 lists the internal interrupt sources of ASRC, trigger conditions, and the generated interrupt signals.

Table 19.6-1. Internal Interrupt Sources of ASRC

Internal Interrupt Source	Trigger Condition	Interrupt Signal
ASRC_CHNL0_INT	Output counter reaches the preset value of CHNL0	ASRC_CHNL0_INTR
ASRC_CHNL1_INT	Output counter reaches the preset value of CHNL1	ASRC_CHNL1_INTR

Note:
For explanations of terms such as interrupts, interrupt sources, and interrupt signals, as well as the relationships between them, please refer to section 9 *Interrupt Matrix [to be added later]* > 9.1 *Interrupt Terminology*.

Each interrupt source has a set of general configuration registers, detailed in section *Interrupt Configuration Registers*. For specific register names, please refer to 19.9 *Register Summary*.

19.7 Configuration Procedure

The configuration procedure of the ASRC module is as follows:

1. Configure the following registers to set parameters related to sample rate conversion:
   - Cascade mode: ASRC_CHNL*n*_CTRL_CFG_REG
   - Channel mode: ASRC_CHNL*n*_CTRL_CFG_REG
   - Sample rate conversion factor and channel mode selection: ASRC_CHNL*n*_CTRL_CFG_REG, ASRC_CHNL*n*_FRAC_COEFF_REG, ASRC_CHNL*n*_FRAC_RECIPL_REG
   - Output data length EOF generation mode: ASRC_CHNL*n*_DOUT_CFG_REG
2. Start the module by setting ASRC_CHNL*n*_START_REG. After startup, the module will enter a data-waiting state.

3. Optionally clear the module's internal cache using ASRC_CHNL*n*_RESET_REG. Clearing the cache will discard the module's internal history. The recommendation is:
   - If a frame of data represents a complete audio signal, the cache can be cleared.
   - If a complete audio signal consists of multiple frames of data, the cache should not be cleared between frames.
4. After enabling compensation, notify the ASRC controller that all input data has been sent by either of the following methods:
   - Enable the DMA EOF feature
   - Set the ASRC reception data length via ASRC_CHNL*n*_DIN_CFG_REG

   For details, refer to section 19.5.5 *Input/Output Data Length*.
5. Monitor output data status using the following registers:
   - ASRC_CHNL*n*_OUT_CNT: Total number of bytes written to DMA
   - ASRC_CHNL*n*_OUT_INC: Number of bytes written to DMA since the last EOF

19.7.1 Usage Restrictions and Notes

1. Before using the ASRC module, configure related registers, including mode selection, interpolation coefficients, decimation coefficients, and channel modes.
2. There is no strict requirement on the startup order of DMA channels and ASRC channels.
3. Do not modify configuration registers during operation to avoid inconsistent states or abnormal data.
4. The data counter unit involved in ASRC is **bytes**, with each sample occupying 2 bytes.
5. When the two channels work independently, configure each channel separately. Supported modes:
   - Input mono, output mono
   - Input mono, output stereo (both channels have the same data)
   - Input stereo, output mono (select one channel for processing)

   When the two channels work cooperatively, the configuration of both channels must remain consistent, supporting:
   - Input stereo, output stereo
6. The configuration must be based on the actual data length of the receiving register ASRC_CHNL*n*_IN_LEN and the sending register ASRC_CHNL*n*_OUT_LEN. The configuration rules are as follows:
   - When channels work independently:
     - If processing mono data, configure only the data length of 1 channel for the corresponding channel (unit: bytes).
     - If processing stereo data, configure the total data length of 2 channels on that channel (unit: bytes).
   - When channels work together:

  - Each channel processes only one channel of data; configure the data length of that channel (unit: bytes).

7. In stereo input or stereo output mode, only the CHNL0 channel can be used, and it will only produce a CHNL0 interrupt signal.
8. If new data needs to be processed, it is recommended to execute the module reset procedure or clear the internal state in the following order after the last transmission is complete:
   - Clear ASRC_CHNL*n*_START
   - Set ASRC_CHNL*n*_INFIFO_RESET
   - Set ASRC_CHNL*n*_RESET
   - Set ASRC_CHNL*n*_OUTFIFO_RESET

   Then proceed with configuration and start.
9. The selection criteria for fractional-order filter structures are as follows:
   - If the input sample rate $<$ output sample rate, use a post-filter structure.
   - If the input sample rate $>$ output sample rate, use a pre-filter structure.
   - If the input sample rate is equal to the output sample rate, then all filters bypass.
10. Pay attention to input and output data length ratios and limits. For specific rules, see 19.5.5 *Input/Output Data Length*.

19.8 Application Example

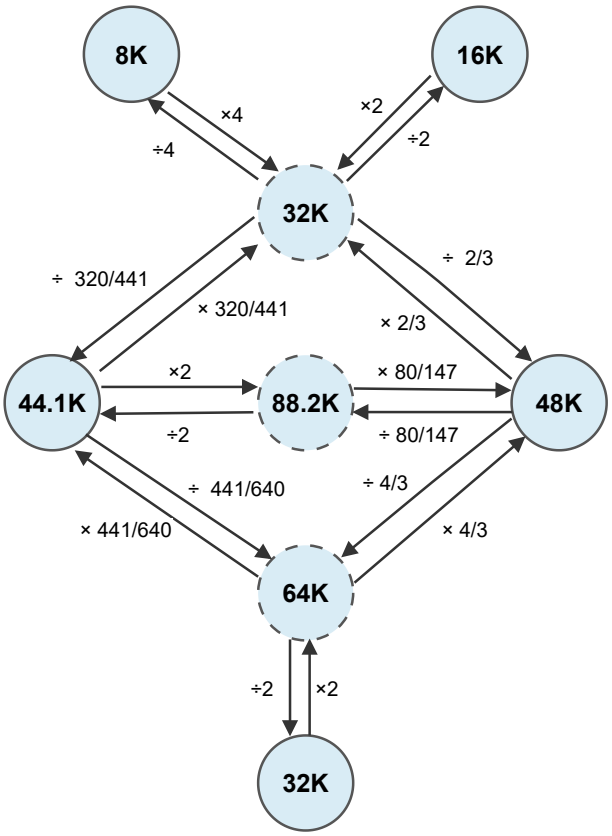


Figure 19.8-1. Sample Rate Conversion Relationship

To achieve high-quality sample rate conversion, it is recommended to follow the illustrated configuration methods. The processing flows for common sample rate conversions are as follows:

- 8 kHz/ 16 kHz to 44.1 kHz/ 48 kHz
  The input signal is first interpolated to 32 kHz, and then converted to the target sample rate via the FRC.
- 32 kHz to 44.1 kHz/ 48 kHz
  The input signal is first interpolated to 64 kHz, and then converted to the target sample rate via the FRC.
- 44.1 kHz/ 48 kHz to 8 kHz/ 16 kHz
  The input signal is first converted to 32 kHz via the FRC, and then downsampled to 8 kHz or 16 kHz.
- 44.1 kHz/ 48 kHz to 32 kHz
  The input signal is first converted to 64 kHz via the FRC, and then downsampled to 32 kHz.
- Converting between 44.1 kHz and 48 kHz
  A two-stage conversion via an intermediate rate of 88.2 kHz is required:
  - 44.1 kHz to 48 kHz: First upsample to 88.2 kHz, then convert to 48 kHz.
  - 48 kHz to 44.1 kHz: First upsample to 88.2 kHz, then convert to 44.1 kHz.

19.9 Register Summary

The addresses in this section are address offsets relative to ASRC base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Please check Section *Access Types for Registers* for the meanings of the abbreviations in the "Access" column.

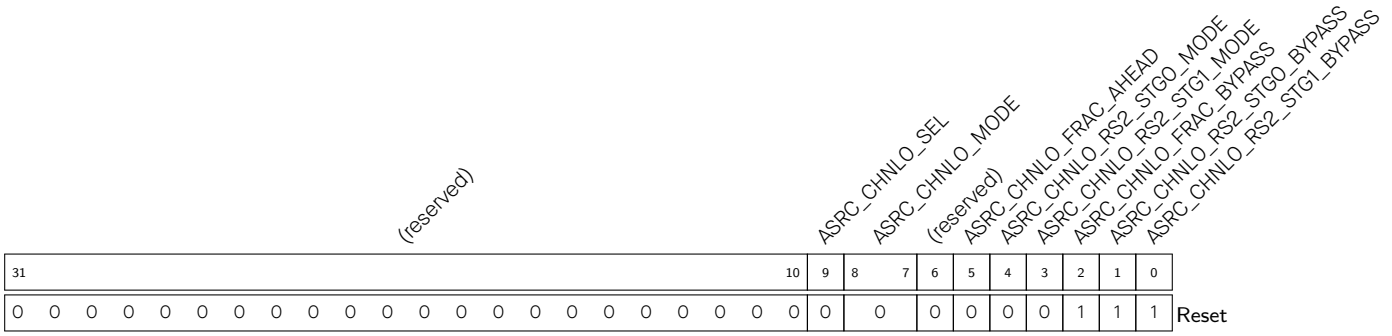
Name	Description	Address	Access
Control and Configuration Registers			
ASRC_CHNL0_CTRL_CFG_REG	CHNL0 control register	0x0000	R/W
ASRC_CHNL0_FRAC_COEFF_REG	CHNL0 FRC configuration register	0x0004	R/W
ASRC_CHNL0_FRAC_RECIPL_REG	CHNL0 FRC configuration register	0x0008	R/W
ASRC_CHNL0_RESET_REG	CHNL0 reset register	0x000C	WT
ASRC_CHNL0_START_REG	CHNL0 start register	0x0010	R/W
ASRC_CHNL0_DIN_CFG_REG	CHNL0 input control and configuration register	0x0014	varies
ASRC_CHNL0_DOUT_CFG_REG	CHNL0 output control and configuration register	0x0018	varies
ASRC_CHNL0_FIFO_CTRL_REG	CHNL0 FIFO reset control register	0x001C	WT
ASRC_CHNL1_CTRL_CFG_REG	CHNL1 control register	0x003C	R/W
ASRC_CHNL1_FRAC_COEFF_REG	CHNL1 FRC configuration register	0x0040	R/W
ASRC_CHNL1_FRAC_RECIPL_REG	CHNL1 FRC configuration register	0x0044	R/W
ASRC_CHNL1_RESET_REG	CHNL1 reset register	0x0048	WT
ASRC_CHNL1_START_REG	CHNL1 start register	0x004C	R/W
ASRC_CHNL1_DIN_CFG_REG	CHNL1 input control and configuration register	0x0050	varies
ASRC_CHNL1_DOUT_CFG_REG	CHNL1 output control and configuration register	0x0054	varies
ASRC_CHNL1_FIFO_CTRL_REG	CHNL1 FIFO reset control register	0x0058	WT
Interrupt Registers			
ASRC_CHNL0_INT_RAW_REG	CHNL0 raw interrupt status	0x0020	R/WTC/SS
ASRC_CHNL0_INT_ST_REG	CHNL0 masked interrupt status	0x0024	RO
ASRC_CHNL0_INT_ENA_REG	CHNL0 interrupt enable bits	0x0028	R/W
ASRC_CHNL0_INT_CLR_REG	CHNL0 interrupt clear bits	0x002C	WT
ASRC_CHNL1_INT_RAW_REG	CHNL1 raw interrupt status	0x005C	R/WTC/SS
ASRC_CHNL1_INT_ST_REG	CHNL1 masked interrupt status	0x0060	RO
ASRC_CHNL1_INT_ENA_REG	CHNL1 interrupt enable bits	0x0064	R/W
ASRC_CHNL1_INT_CLR_REG	CHNL1 interrupt clear bits	0x0068	WT
Status Registers			
ASRC_CHNL0_OUT_CNT_REG	CHNL0 status register	0x0030	RO
ASRC_CHNL1_OUT_CNT_REG	CHNL1 status register	0x006C	RO
Debug Registers			
ASRC_CHNL0_TRACE1_REG	CHNL0 debug register 1	0x0038	RO
ASRC_CHNL1_TRACE1_REG	CHNL1 debug register 1	0x0074	RO
Configuration Registers			
ASRC_SYS_REG	Clock control and configuration register	0x00F8	R/W
Version Registers			
ASRC_DATE_REG	Version control register	0x00FC	R/W

19.10 Registers

The addresses in this section are address offsets relative to ASRC base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 19.1. ASRC_CHNL0_CTRL_CFG_REG (0x0000)



ASRC_CHNL0_RS2_STG1_BYPASS Set this bit to bypass stage 1 resampler in channel 0. (R/W)

ASRC_CHNL0_RS2_STG0_BYPASS Set this bit to bypass stage 0 resampler in channel 0. (R/W)

ASRC_CHNL0_FRAC_BYPASS Set this bit to bypass fractional resampler in channel 0. (R/W)

ASRC_CHNL0_RS2_STG1_MODE Write this bit to configure stage 1 resampler mode in channel 0:

0: interpolation by factor of 2

1: decimation by factor of 2

(R/W)

ASRC_CHNL0_RS2_STG0_MODE Write this bit to configure stage 0 resampler mode in channel 0:

0: interpolation by factor of 2

1: decimation by factor of 2

(R/W)

ASRC_CHNL0_FRAC_AHEAD Set this bit to move fractional resampler to the first stage in channel 0, it should be set to 1 when input frequency is higher than the output frequency. (R/W)

ASRC_CHNL0_MODE Write this field to configure the channel mode:

0: in and out both mono

1: in and out both stereo

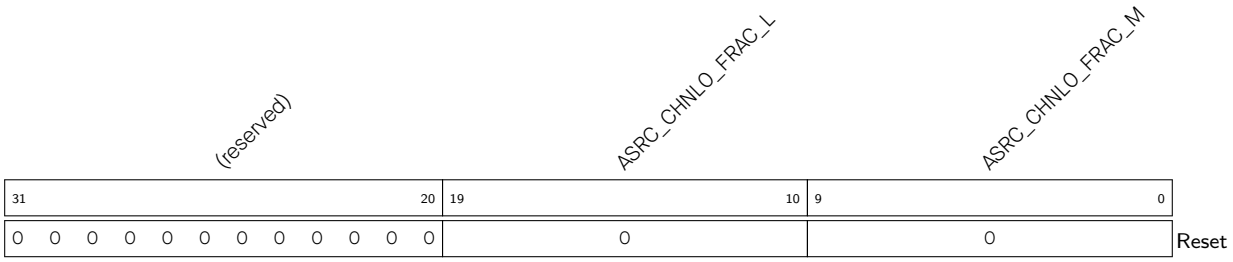
2: in mono, out stereo

3: in stereo, out mono

(R/W)

ASRC_CHNL0_SEL Write this bit to select which 16 bit data will be processed. (R/W)

Register 19.2. ASRC_CHNL0_FRAC_COEFF_REG (0x0004)



ASRC_CHNL0_FRAC_M Write the bits to specify the denominator of fractional resampler factor in channel 0. This value must be coprime with ASRC_CHNL0_FRAC_L. (R/W)

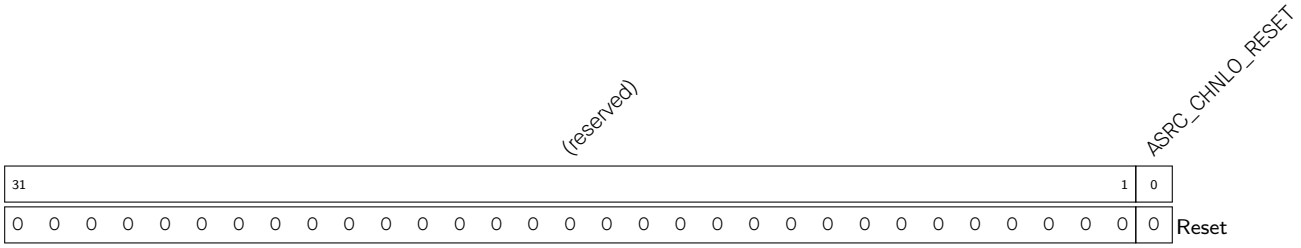
ASRC_CHNL0_FRAC_L Write the bits to specify the numerator of fractional resampler factor in channel 0. This value must be coprime with ASRC_CHNL0_FRAC_M. (R/W)

Register 19.3. ASRC_CHNL0_FRAC_RECIPL_REG (0x0008)



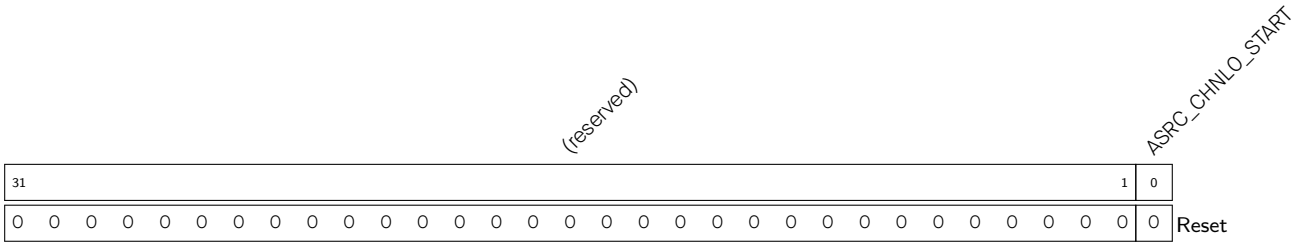
ASRC_CHNL0_FRAC_RECIPL Write the bits to set the value in channel 0, calculated as $\left\lfloor \frac{2^{19}+L}{2L} \right\rfloor$. (R/W)

Register 19.4. ASRC_CHNL0_RESET_REG (0x000C)



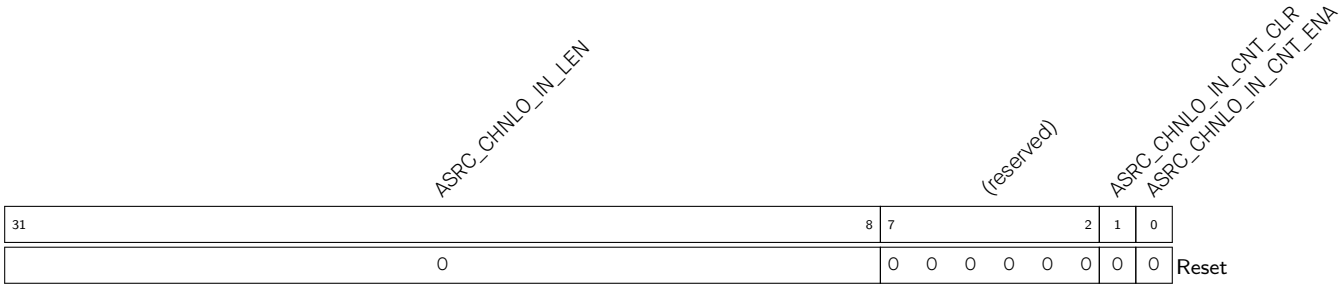
ASRC_CHNL0_RESET Set this bit to reset channel 0. (WT)

Register 19.5. ASRC_CHNLO_START_REG (0x0010)



ASRC_CHNLO_START Set this bit to start channel 0. (R/W)

Register 19.6. ASRC_CHNLO_DIN_CFG_REG (0x0014)



ASRC_CHNLO_IN_CNT_ENA Set this bit to enable input data byte counter. (R/W)

ASRC_CHNLO_IN_CNT_CLR Set this bit to clear input data byte counter. (WT)

ASRC_CHNLO_IN_LEN Write the bits to specify the number of data bytes to be received from the DMA. (R/W)

Register 19.7. ASRC_CHNLO_DOUT_CFG_REG (0x0018)



ASRC_CHNLO_OUT_EOF_GEN_MODE Write the bits to specify which EOF will be written to DMA:

0: Counter EOF

1: DMA input EOF

2: Both counter EOF and DMA input EOF

3: None

(R/W)

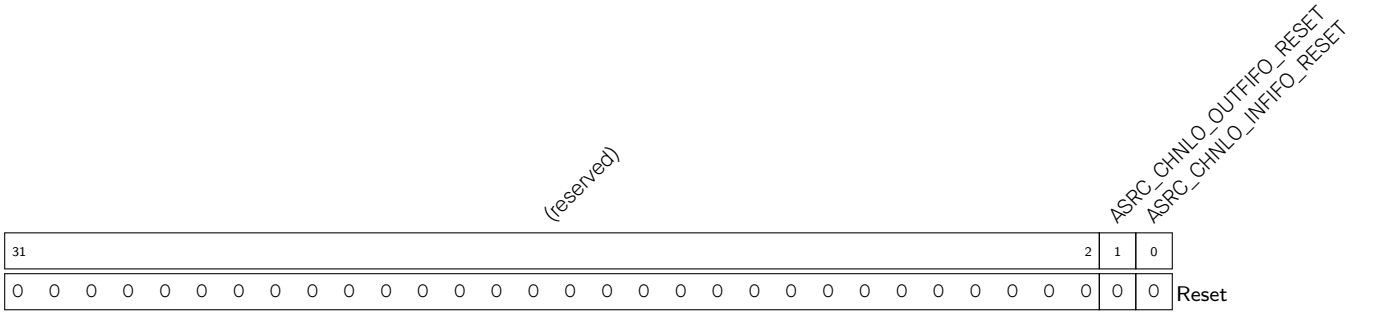
ASRC_CHNLO_OUT_CNT_ENA Set this bit to enable output data byte counter. (R/W)

ASRC_CHNLO_OUT_CNT_CLR Set this bit to clear output data byte counter. (WT)

ASRC_CHNLO_OUT_LEN_COMP Set this bit to enable compensation for output data byte counter when using fractional resampler with decimation by 2 if ASRC_CHNLO_OUT_CNT ≥ ASRC_CHNLO_OUT_LEN. (R/W)

ASRC_CHNLO_OUT_LEN Write the bits to specify the number of data bytes to transfer to the DMA; counter EOF flag will be set when the counter reaches its end. (R/W)

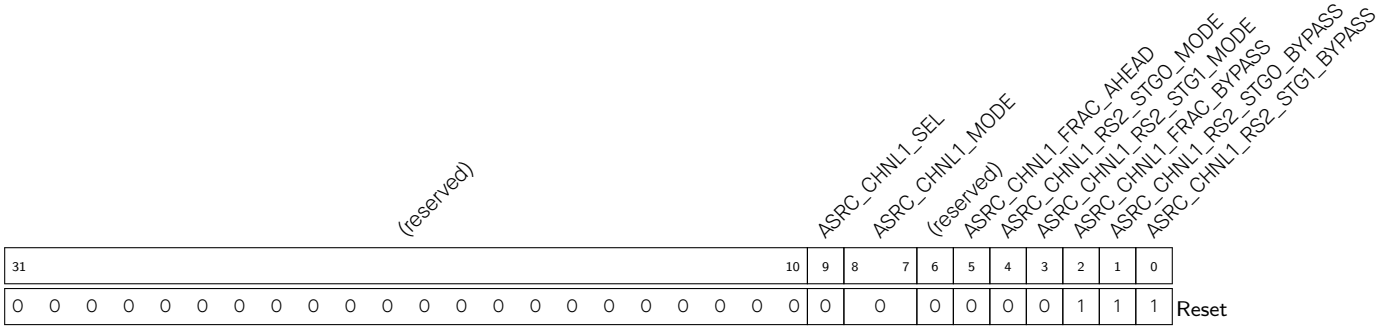
Register 19.8. ASRC_CHNLO_FIFO_CTRL_REG (0x001C)



ASRC_CHNLO_INFIFO_RESET Set this bit to reset infifo. (WT)

ASRC_CHNLO_OUTFIFO_RESET Set this bit to reset outfifo. (WT)

Register 19.9. ASRC_CHNL1_CTRL_CFG_REG (0x003C)



ASRC_CHNL1_RS2_STG1_BYPASS Set this bit to bypass stage 1 resampler in channel 1. (R/W)

ASRC_CHNL1_RS2_STG0_BYPASS Set this bit to bypass stage 0 resampler in channel 1. (R/W)

ASRC_CHNL1_FRAC_BYPASS Set this bit to bypass fractional resampler in channel 1. (R/W)

ASRC_CHNL1_RS2_STG1_MODE Write this bit to configure stage 1 resampler mode in channel 1:

0: interpolation by factor of 2

1: decimation by factor of 2

(R/W)

ASRC_CHNL1_RS2_STG0_MODE Write this bit to configure stage 0 resampler mode in channel 1:

0: interpolation by factor of 2

1: decimation by factor of 2

(R/W)

ASRC_CHNL1_FRAC_AHEAD Set this bit to move fractional resampler to the first stage in channel 1, it should be 1 when input frequency is higher than the output frequency. (R/W)

ASRC_CHNL1_MODE Write this field to configure the channel mode:

0: in and out both mono

1: in and out both stereo

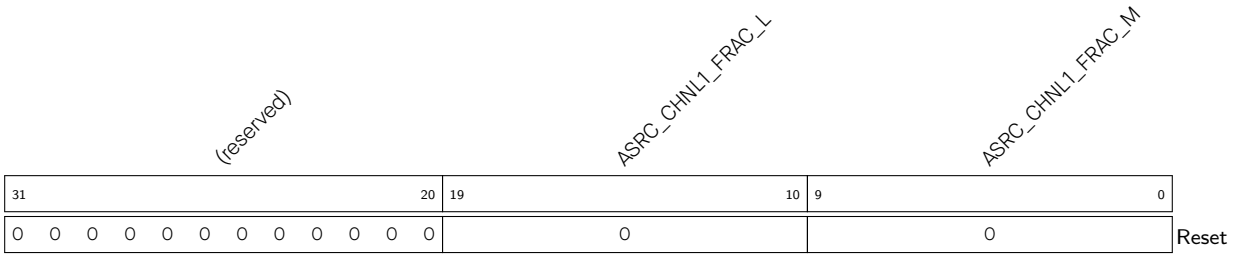
2: in mono, out stereo

3: in stereo, out mono

(R/W)

ASRC_CHNL1_SEL Write this bit to select which 16 bit data will be processed. (R/W)

Register 19.10. ASRC_CHNL1_FRAC_COEFF_REG (0x0040)



ASRC_CHNL1_FRAC_M Write the bits to specify the denominator of fractional resampler factor in channel 1. This value must be coprime with ASRC_CHNL1_FRAC_L. (R/W)

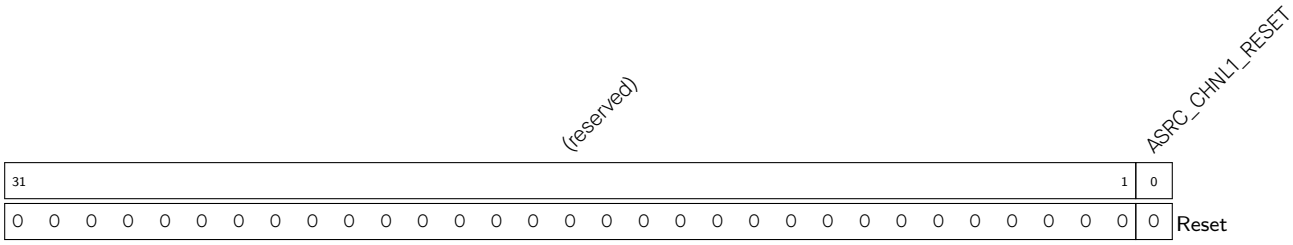
ASRC_CHNL1_FRAC_L Write the bits to specify the numerator of fractional resampler factor in channel 1. This value must be coprime with ASRC_CHNL1_FRAC_M. (R/W)

Register 19.11. ASRC_CHNL1_FRAC_RECIPL_REG (0x0044)



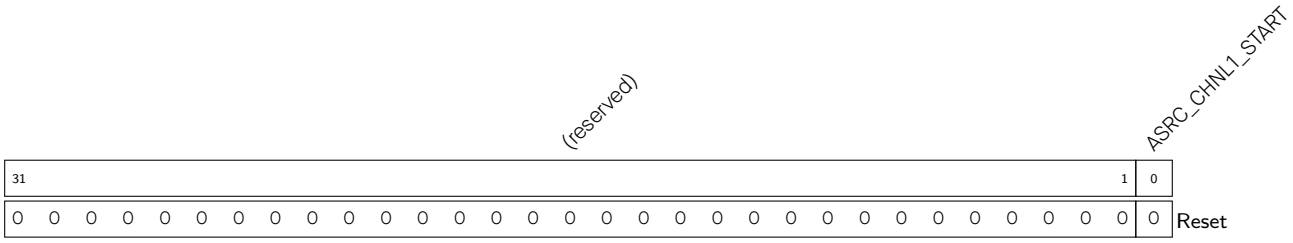
ASRC_CHNL1_FRAC_RECIPL Write the bits to set the value in channel 1, calculated as $\left\lfloor \frac{2^{19}+L}{2L} \right\rfloor$. (R/W)

Register 19.12. ASRC_CHNL1_RESET_REG (0x0048)



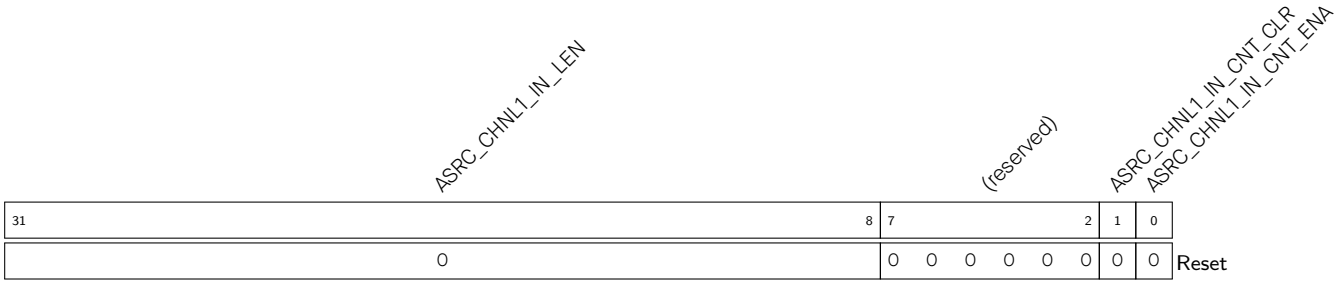
ASRC_CHNL1_RESET Set this bit to reset channel 1. (WT)

Register 19.13. ASRC_CHNL1_START_REG (0x004C)



ASRC_CHNL1_START Set this bit to start channel 1. (R/W)

Register 19.14. ASRC_CHNL1_DIN_CFG_REG (0x0050)

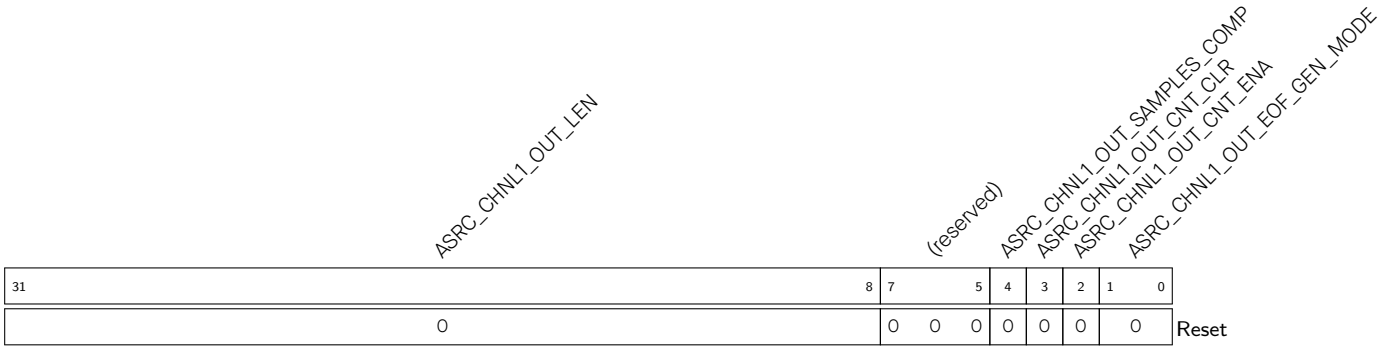


ASRC_CHNL1_IN_CNT_ENA Set this bit to enable input data byte counter. (R/W)

ASRC_CHNL1_IN_CNT_CLR Set this bit to clear input data byte counter. (WT)

ASRC_CHNL1_IN_LEN Write the bits to specify the number of data bytes to be received from the DMA. (R/W)

Register 19.15. ASRC_CHNL1_DOUT_CFG_REG (0x0054)



ASRC_CHNL1_OUT_EOF_GEN_MODE Write the bits to specify which EOF will be written to DMA:

0: Counter EOF

1: DMA input EOF

2: Both counter EOF and DMA input EOF

3: None

(R/W)

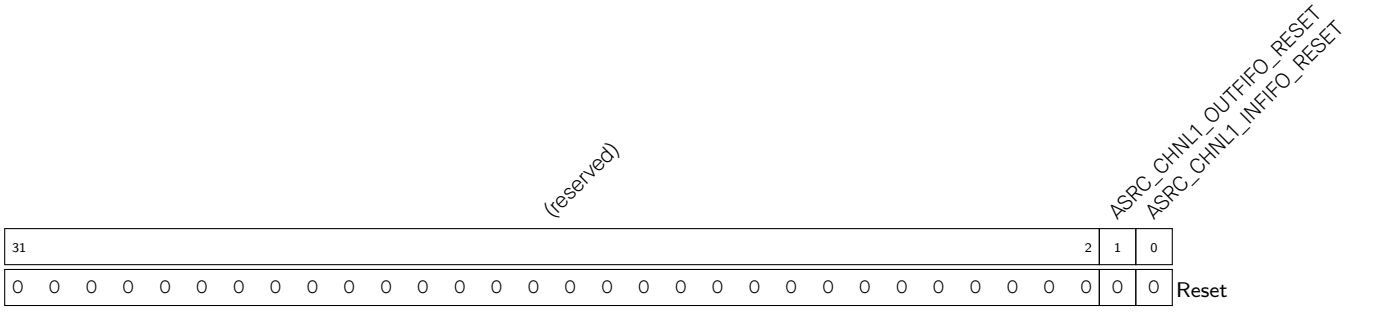
ASRC_CHNL1_OUT_CNT_ENA Set this bit to enable output data byte counter. (R/W)

ASRC_CHNL1_OUT_CNT_CLR Set this bit to clear output data byte counter. (WT)

ASRC_CHNL1_OUT_SAMPLES_COMP Set this bit to enable compensation for output byte counter when using fractional resampler with decimation by 2 if ASRC_CHNL1_OUT_CNT ≥ ASRC_CHNL1_OUT_LEN. (R/W)

ASRC_CHNL1_OUT_LEN Write the bits to specify the number of data bytes to transfer to the DMA; counter EOF flag will be set when the counter reaches its end. (R/W)

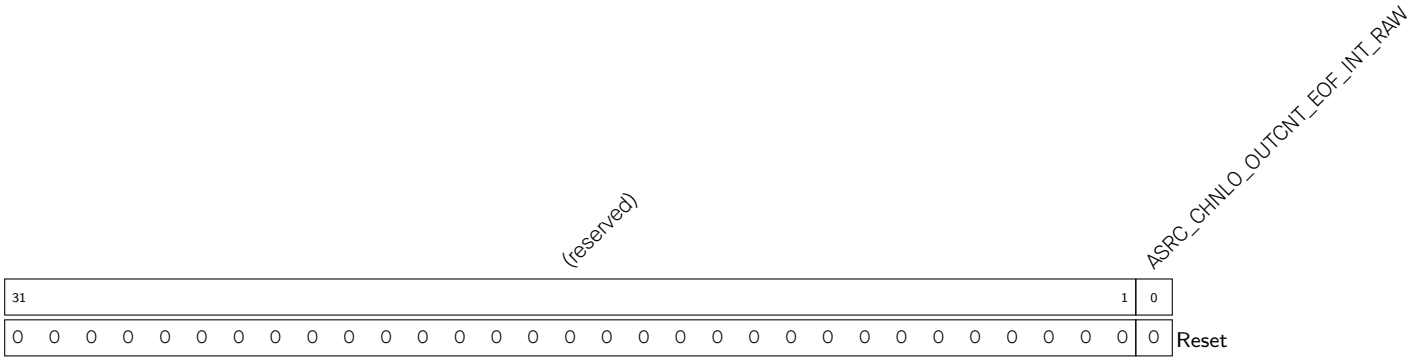
Register 19.16. ASRC_CHNL1_FIFO_CTRL_REG (0x0058)



ASRC_CHNL1_INFIFO_RESET Set this bit to reset infifo. (WT)

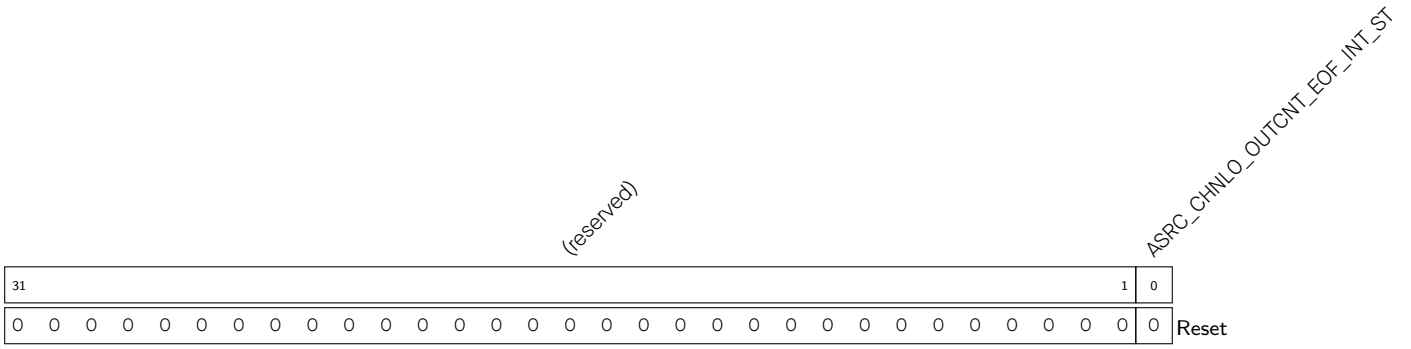
ASRC_CHNL1_OUTFIFO_RESET Set this bit to reset outfifo. (WT)

Register 19.17. ASRC_CHNLO_INT_RAW_REG (0x0020)



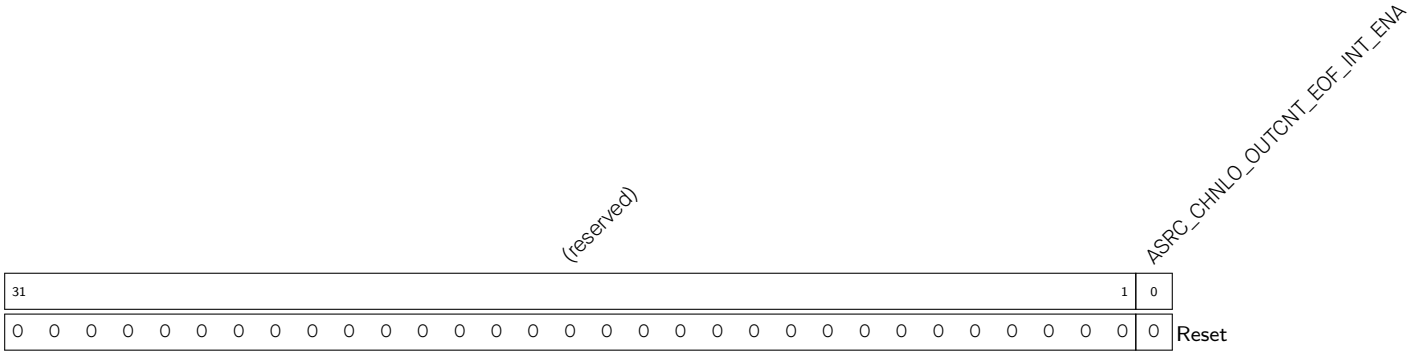
ASRC_CHNLO_OUTCNT_EOF_INT_RAW The raw interrupt status of ASRC_CHNLO_INT. (R/WTC/SS)

Register 19.18. ASRC_CHNLO_INT_ST_REG (0x0024)



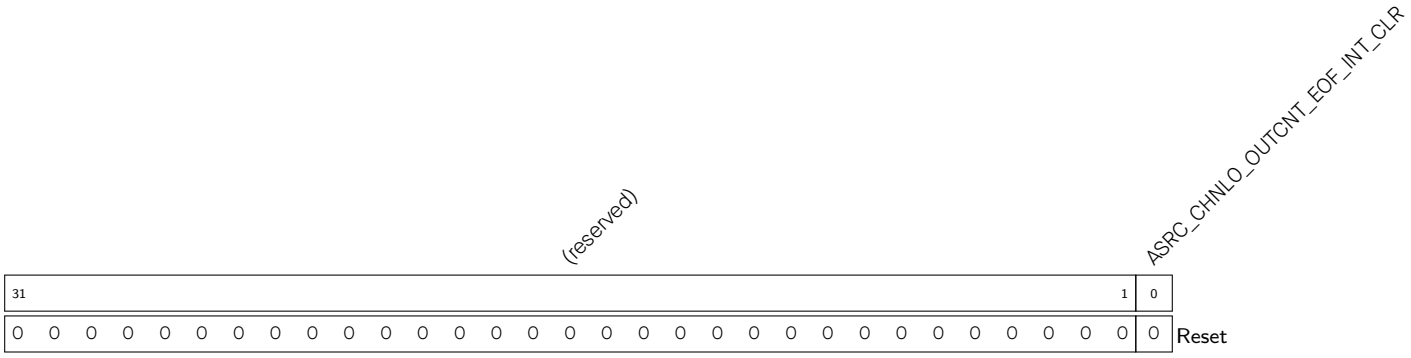
ASRC_CHNLO_OUTCNT_EOF_INT_ST The masked interrupt status of ASRC_CHNLO_INT. (RO)

Register 19.19. ASRC_CHNLO_INT_ENA_REG (0x0028)



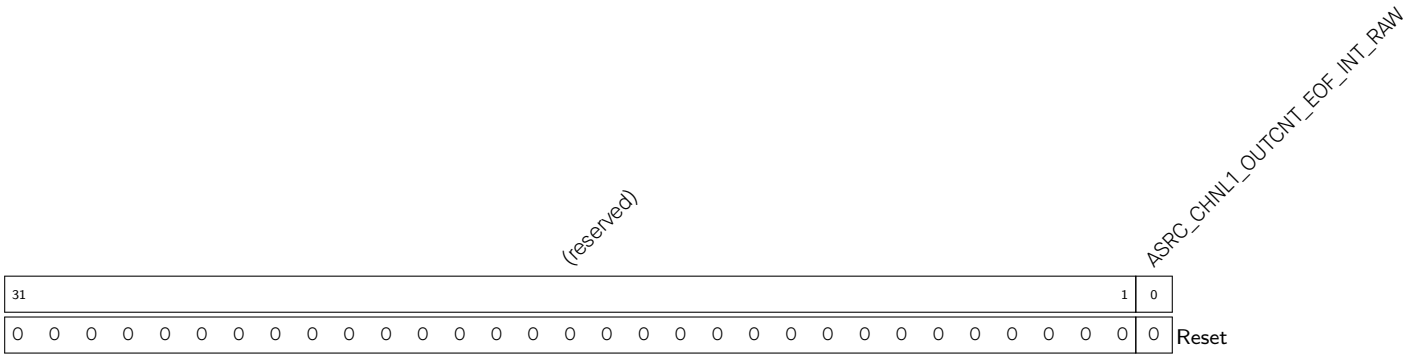
ASRC_CHNLO_OUTCNT_EOF_INT_ENA Write 1 to enable ASRC_CHNLO_INT. (R/W)

Register 19.20. ASRC_CHNL0_INT_CLR_REG (0x002C)



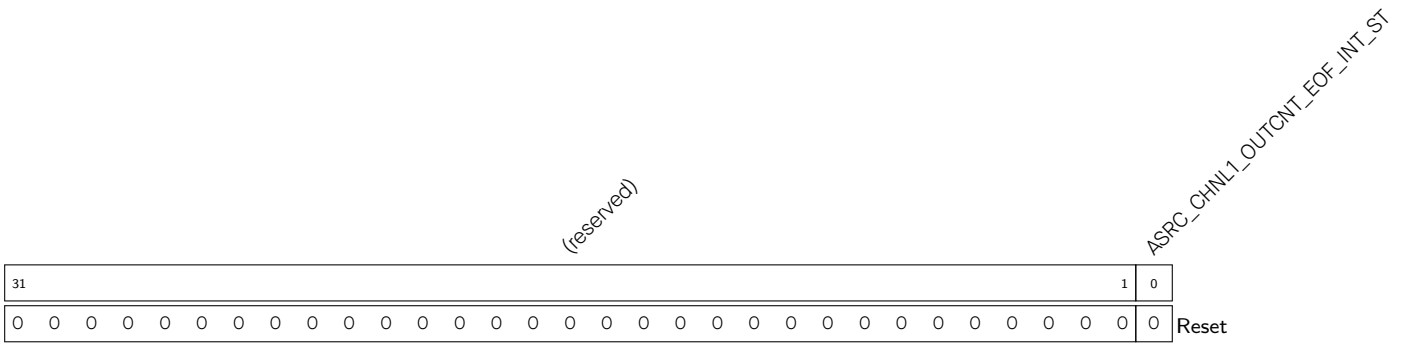
ASRC_CHNL0_OUTCNT_EOF_INT_CLR Write 1 to clear ASRC_CHNL0_INT. (WT)

Register 19.21. ASRC_CHNL1_INT_RAW_REG (0x005C)



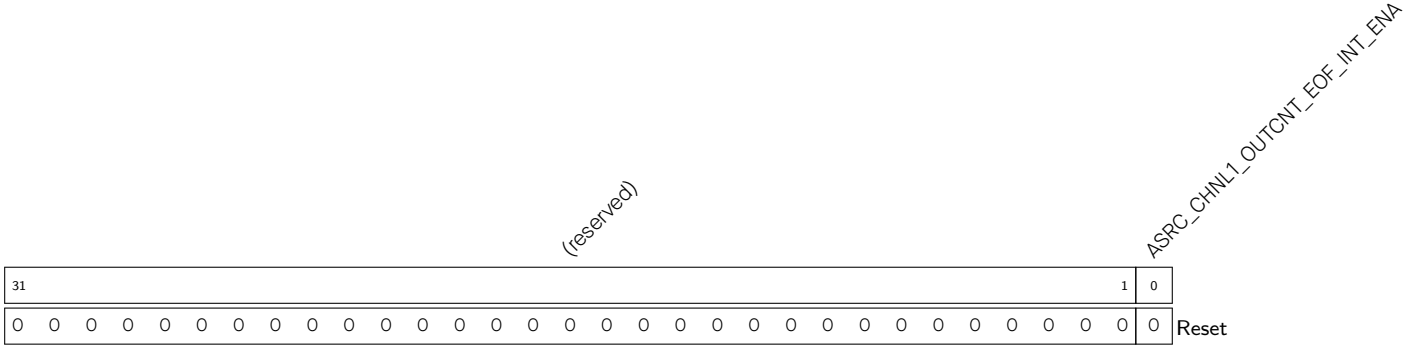
ASRC_CHNL1_OUTCNT_EOF_INT_RAW The raw interrupt status of ASRC_CHNL1_INT. (R/WTC/SS)

Register 19.22. ASRC_CHNL1_INT_ST_REG (0x0060)



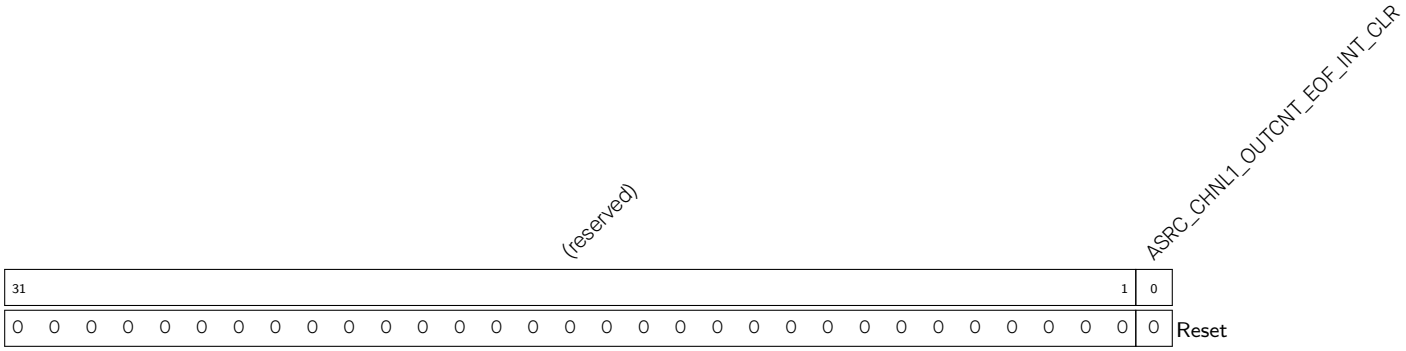
ASRC_CHNL1_OUTCNT_EOF_INT_ST The masked interrupt status of ASRC_CHNL1_INT. (RO)

Register 19.23. ASRC_CHNL1_INT_ENA_REG (0x0064)



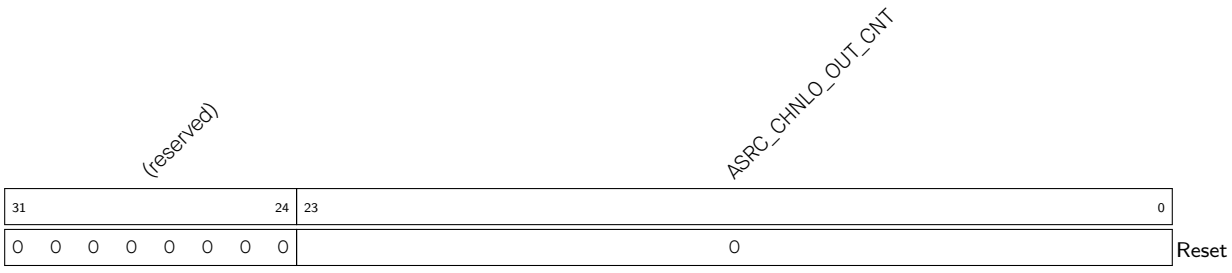
ASRC_CHNL1_OUTCNT_EOF_INT_ENA Write 1 to enable ASRC_CHNL1_INT. (R/W)

Register 19.24. ASRC_CHNL1_INT_CLR_REG (0x0068)



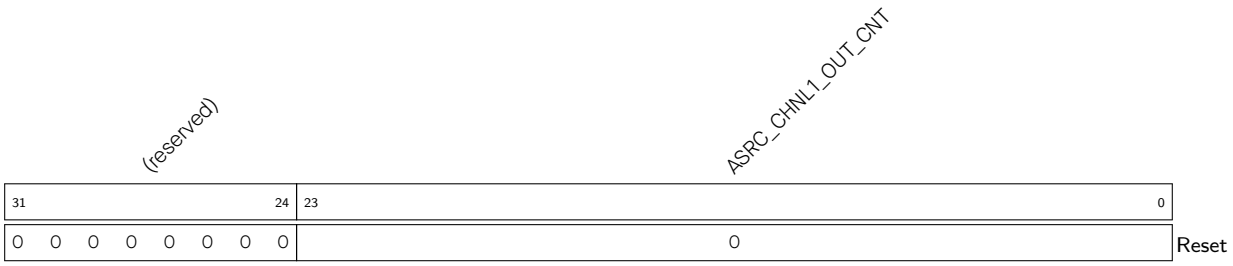
ASRC_CHNL1_OUTCNT_EOF_INT_CLR Write 1 to clear ASRC_CHNL1_INT. (WT)

Register 19.25. ASRC_CHNLO_OUT_CNT_REG (0x0030)



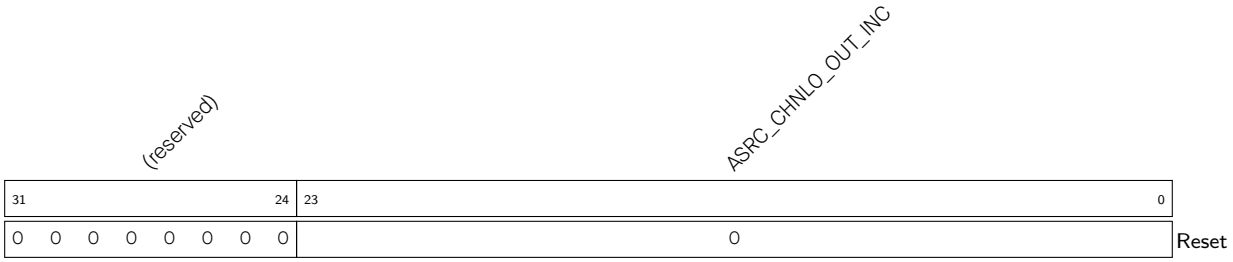
ASRC_CHNLO_OUT_CNT Represents the cumulative number of data bytes sent to the DMA. (RO)

Register 19.26. ASRC_CHNL1_OUT_CNT_REG (0x006C)



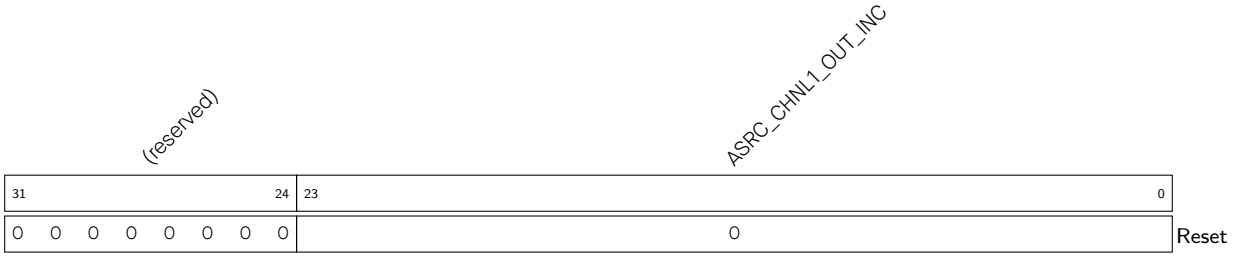
ASRC_CHNL1_OUT_CNT Represents the cumulative number of data bytes sent to the DMA. (RO)

Register 19.27. ASRC_CHNL0_TRACE1_REG (0x0038)



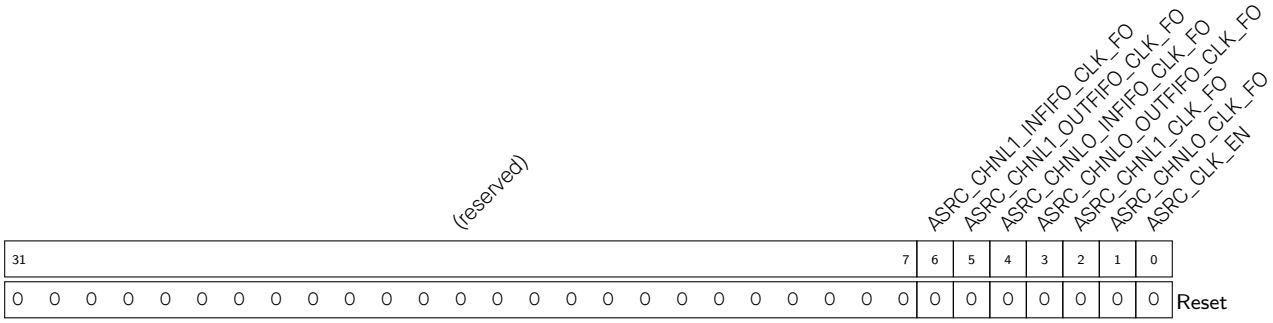
ASRC_CHNL0_OUT_INC Represents the number of data bytes sent to the DMA since the last EOF. (RO)

Register 19.28. ASRC_CHNL1_TRACE1_REG (0x0074)



ASRC_CHNL1_OUT_INC Represents the number of data bytes sent to the DMA since the last EOF. (RO)

Register 19.29. ASRC_SYS_REG (0x00F8)



ASRC_CLK_EN Configures whether to enable the ASRC module clock.

0: Disable

1: Enable

(R/W)

ASRC_CHNL0_CLK_FO Set this bit to make channel 0 clock free run. (R/W)

ASRC_CHNL1_CLK_FO Set this bit to make channel 1 clock free run. (R/W)

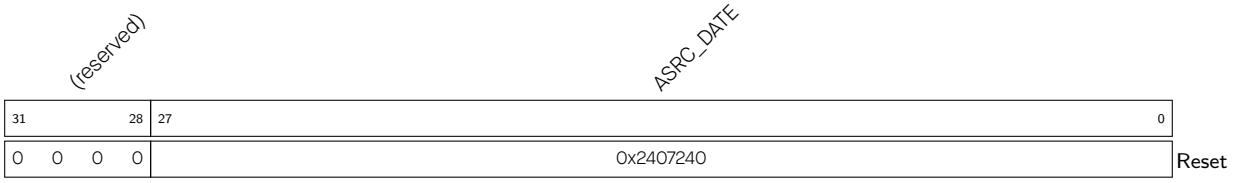
ASRC_CHNL0_OUTFIFO_CLK_FO Set this bit to make channel 0 outfifo clock free run. (R/W)

ASRC_CHNL0_INFIFO_CLK_FO Set this bit to make channel 0 infifo clock free run. (R/W)

ASRC_CHNL1_OUTFIFO_CLK_FO Set this bit to make channel 1 outfifo clock free run. (R/W)

ASRC_CHNL1_INFIFO_CLK_FO Set this bit to make channel 1 infifo clock free run. (R/W)

Register 19.30. ASRC_DATE_REG (0x00FC)



ASRC_DATE Version control register. (R/W)

Chapter 20

LCD and Camera Controller (LCD_CAM)

20.1 Overview

The LCD and Camera controller (LCD_CAM) on the ESP32-S31 consists of an independent LCD control module and a camera control module. It is a versatile component that can be used to interface with both LCDs and cameras.

The LCD module provides an LCD interface that can connect to RGB, Motorola 6800 (MOTO6800), and Intel 8080 (I8080) compatible LCD devices. The LCD module is responsible for sending video data to the display.

The Camera module provides one parallel camera interface that can connect to external digital video port (DVP) cameras in 8/16-bit modes. It manages communication with the camera to capture images or video frames.

The LCD_CAM is particularly useful for projects that require both displaying information on screens and working with cameras for video processing.

20.2 Features

LCD_CAM has the following features:

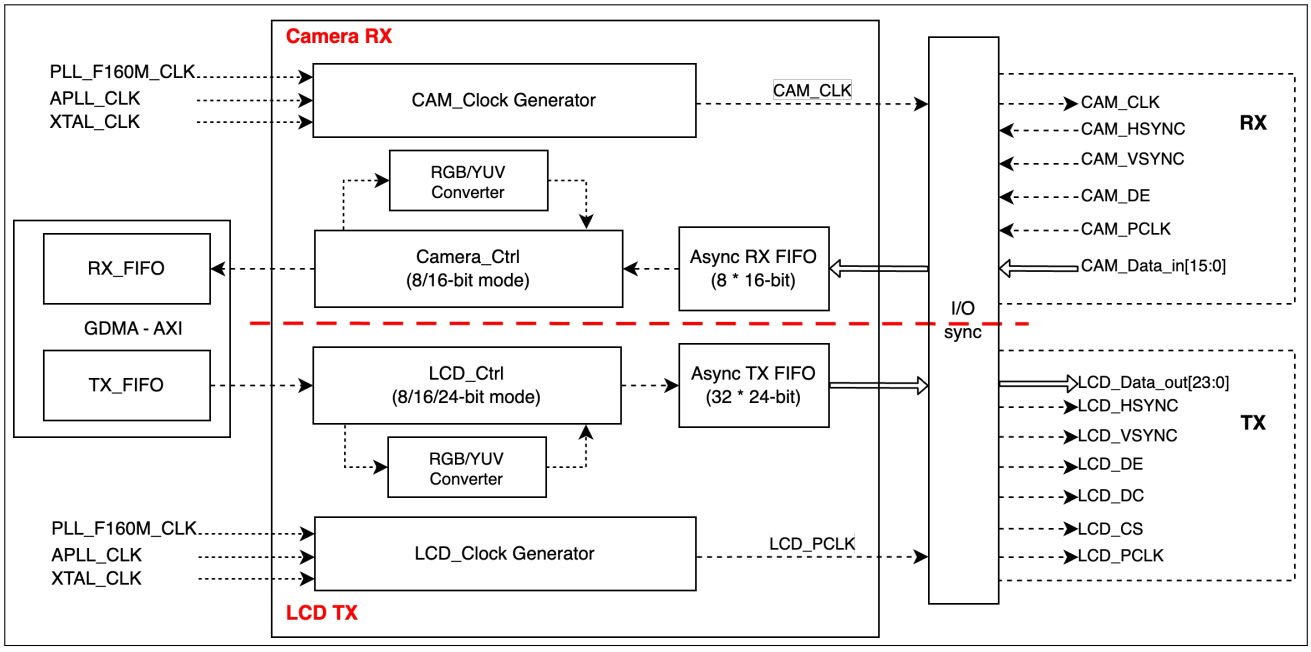
- Supports the following operation modes:
  - LCD master TX mode
  - Camera slave RX mode
  - Camera master RX mode
- Supports simultaneous connection to an external LCD and a camera
- When interfacing with an external LCD, the following is supported:
  - 8/16/24-bit parallel output modes
  - RGB, MOTO6800, and I8080 LCD formats
  - LCD data retrieved from internal memory or external memory via GDMA-AXI
- When interfacing with an external camera (i.e., DVP image sensor), the following is supported:
  - 8/16-bit parallel input modes
  - Camera data stored in internal or external memory via GDMA-AXI
- Supports interrupts

20.3 Functional Description

20.3.1 Block Diagram

Figure 20.3-1 shows the structure of LCD_CAM which includes:

- One TX control unit (LCD_Ctrl)
- One RX control unit (Camera_Ctrl)
- One asynchronous TX FIFO (Async TX FIFO) for communicating with external devices, e.g., LCDs
- One asynchronous RX FIFO (Async RX FIFO) for communicating with external devices, e.g., cameras
- Two clock generators (LCD_Clock Generator and CAM_Clock Generator) for generating clocks for each module
- Two format converters (RGB/YUV Converter) for converting video data into various formats



- →: control signal =>: data flow

Figure 20.3-1. LCD_CAM Block Diagram

20.3.2 Signal Description

Table 20.3-1 shows the input and output signals required for the operation of the LCD and Camera controller.

The definition of the three operation modes is as follows:

- **Camera Slave RX Mode:** In this mode, the Camera module is the receiver of control signals.
- **Camera Master RX Mode:** In this mode, the Camera module sends clock signals to the slave, which will transmit data to the master upon receiving the signals.
- **LCD Master TX Mode:** In this mode, the LCD module initiates communication with the slave and sends control signals to the slave.

Table 20.3-1. LCD_CAM Input and Output Signal Descriptions

Operation Mode	Signal[1]	Direction	Function
Camera Slave RX Mode	CAM_PCLK	Input	Camera pixel clock signal
	CAM_VSYNC	Input	Vertical synchronization signal (VSYNC)[3]
	CAM_HSYNC	Input	Horizontal synchronization signal (HSYNC)[3]
	CAM_DE	Input	Horizontal enable signal (DE)[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
Camera Master RX Mode	CAM_PCLK	Input	Camera pixel clock input signal
	CAM_CLK	Output	Camera master clock output signal
	CAM_VSYNC	Input	Vertical synchronization signal[3]
	CAM_HSYNC	Input	Horizontal synchronization signal[3]
	CAM_DE	Input	Horizontal enable signal[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
LCD Master TX Mode	LCD_PCLK	Output	LCD pixel clock signal
	LCD_HSYNC	Output	Horizontal synchronization signal in RGB format
	LCD_VSYNC	Output	Vertical synchronization signal in RGB format
	LCD_DE	Output	Horizontal enable signal in RGB format
	LCD_DC	Output	Data and command (DC) signal in I8080 format
	LCD_CS	Output	Chip select (CS) signal in I8080/MOTO6800 format
	LCD_Data_out[*M*:0][2]	Output	LCD parallel output data bus, 8/16/24-bit modes supported

[1] All the signals must be mapped to the chip's pins via GPIO matrix. For more information, see Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.

[2] For the input signals with 8 or 16-bit width, *N* = 7 or 15 respectively. For the output signals with 8, 16, or 24-bit width, *M* = 7, 15, or 23 respectively.

[3] If LCD_CAM_CAM_VH_DE_MODE_EN is set, i.e., VSYNC + HSYNC mode is selected, then VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines. If LCD_CAM_CAM_VH_DE_MODE_EN is cleared, i.e., DE mode is selected, then VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the Camera module is not available.

20.3.3 LCD_CAM Module Clocks

20.3.3.1 LCD Clock

The clocks used in the LCD module are generated from clock sources by the LCD_Clock Generator, see Figure 20.3-2. The clocks include:

- Master clock: LCD_CLK, divided from the clock sources
- Pixel clock: LCD_PCLK, divided from LCD_CLK

HP_SYS_CLKRST_LCD_CLK_EN is used to enable LCD_CLK_SRC (the clock source), and HP_SYS_CLKRST_LCD_CLK_SRC_SEL is used to select LCD_CLK_SRC from one of the following clock sources:

- 0: XTAL_CLK
- 1: PLL_F160M_CLK
- 2: APLL_CLK
- 3: Disable LCD clock source

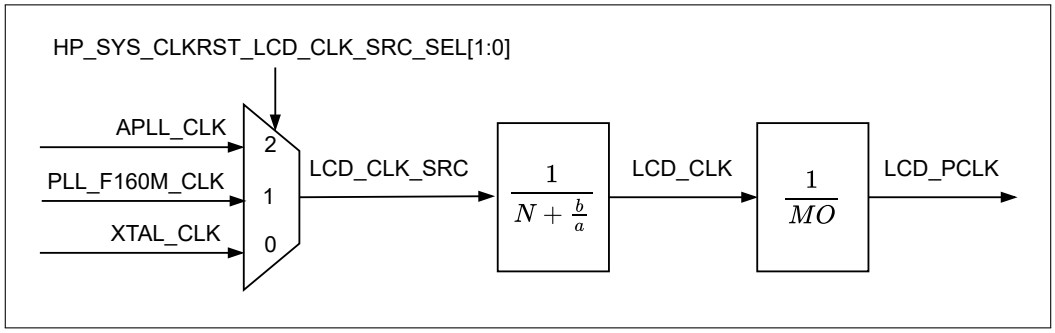


Figure 20.3-2. LCD Clock

The following formula shows the relation between the frequency of LCD_CLK ($f_{\text{LCD\_CLK}}$) and LCD_CLK_SRC ($f_{\text{LCD\_CLK\_SRC}}$):

$$f_{\text{LCD\_CLK}} = \frac{f_{\text{LCD\_CLK\_SRC}}}{N + \frac{b}{a}}$$

The values of *N*, *a*, and *b* are related to HP_SYS_CLKRST_LCD_CLK_DIV_NUM, HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR, and HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR. Specifically,

- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM = 0, *N* = 1. The values of *a* and *b* do not take effect. The divider is always 1.
- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM >=1, *N* = HP_SYS_CLKRST_LCD_CLK_DIV_NUM + 1.
  - For integer divider, please clear HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR.
  - For fractional divider, *b* = HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR, *a* = HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR. The value of HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR should be smaller than the value of HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR.

The following formula shows the relation between the frequencies of LCD_PCLK ($f_{\text{LCD\_PCLK}}$) and LCD_CLK ($f_{\text{LCD\_CLK}}$):

$$f_{\text{LCD\_PCLK}} = \frac{f_{\text{LCD\_CLK}}}{\text{MO}}$$

MO is determined by LCD_CAM_LCD_CLK_EQU_SYSCLK and LCD_CAM_LCD_CLKCNT_N, specifically:

- When LCD_CAM_LCD_CLK_EQU_SYSCLK = 1, MO = 1.

- When LCD_CAM_LCD_CLK_EQU_SYSCLK = 0, MO = LCD_CAM_LCD_CLKCNT_N + 1.

Notes:

- LCD_CAM_LCD_CLKCNT_N must not be configured as 0.
- Using fractional divider may introduce clock jitters. In case LCD_CLK and LCD_PCLK can not be generated from PLL_F160M_CLK by integer divider, APLL_CLK can be used as the clock source. For more information, please refer to Chapter 7 *Reset and Clock [to be added later]*.

20.3.3.2 Camera Clock

The clocks used in the Camera module are generated from clock sources by the CAM_Clock Generator, see Figure 20.3-3. The clocks include:

- Master clock: CAM_CLK, master clock output from the Camera module, divided from the clock sources.
- Pixel clock: CAM_PCLK, clock input from camera slave.

HP_SYS_CLKRST_CAM_CLK_EN is used to enable the CAM_CLK_SRC (the clock source), and HP_SYS_CLKRST_CAM_CLK_SRC_SEL is used to select CAM_CLK_SRC from one of the following clock sources:

- 0: XTAL_CLK
- 1: PLL_F160M_CLK
- 2: APLL_CLK
- 3: Disable camera clock source

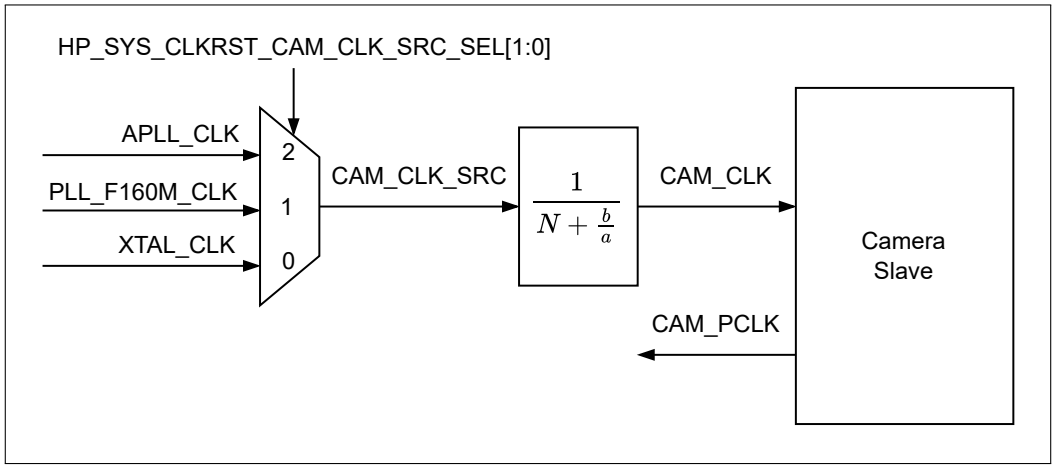


Figure 20.3-3. Camera Clock

The following formula shows the relation between the frequencies of CAM_CLK (f_{CAM_CLK}) and the divider's clock source ($f_{CAM\_CLK\_SRC}$):

$$f_{CAM\_CLK} = \frac{f_{CAM\_CLK\_SRC}}{N + \frac{b}{a}}$$

N is an integer value between 2 and 256. The value of *N* corresponds to the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUM as follows:

- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 0, *N* = 256.

- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 1, *N* = 2.
- When HP_SYS_CLKRST_CAM_CLK_DIV_NUM = other value, *N* = HP_SYS_CLKRST_CAM_CLK_DIV_NUM.

b corresponds to the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR, and *a* to the value of HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.

- For integer divider, please clear HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.
- For fractional divider, the value of HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR should be smaller than the value of HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR.

20.3.4 LCD_CAM Reset

LCD_CAM module has the following reset registers that can reset different parts of the module:

- HP_SYS_CLKRST_LCDCAM_RST_EN: Setting this register resets the whole LCD_CAM module.
- LCD_CAM_LCD_RESET: Setting this register resets the whole LCD module, including the LCD control unit (LCD_Ctrl), the RGB/YUV Converter, and Async TX FIFO. Configuring this register will not affect the Camera module.
- LCD_CAM_CAM_RESET: Setting this register resets the whole Camera module, including the Camera control unit (Camera_Ctrl), the RGB/YUV Converter, and Async RX FIFO. Configuring this register will not affect the LCD module.
- LCD_CAM_LCD_AFIFO_RESET: Setting this register resets the Async TX FIFO. For example, setting this reset register every time the LCD finishes sending a frame can prevent transmission errors of the previous frame from affecting the transmission of the next frame.
- LCD_CAM_CAM_AFIFO_RESET: Setting this register resets the Async RX FIFO. For example, setting this reset register every time the Camera finishes receiving a frame can prevent transmission errors of the previous frame from affecting the receiving of the next frame.

Notes:

- The reset register HP_SYS_CLKRST_LCDCAM_RST_EN is not self-clearing, meaning writing 1 enables the reset, and writing 0 releases the reset.
- All the above reset registers, except for HP_SYS_CLKRST_LCDCAM_RST_EN, are self-clearing, meaning that after writing 1, the hardware will generate a reset pulse and automatically clear the register.
- The module clocks LCD_CLK and CAM_CLK must be configured first before the module and FIFO are reset.
- To reset Async TX FIFO, configure LCD_CAM_LCD_RESET first, then configure LCD_CAM_LCD_AFIFO_RESET.
- To reset Async RX FIFO, configure LCD_CAM_CAM_RESET first, then configure LCD_CAM_CAM_AFIFO_RESET.

20.3.5 LCD_CAM Data Format Control

20.3.5.1 LCD Data Format Control

When the data is transmitted by the LCD module, it undergoes three stages of processing:

1. Stage One: Data preprocessing
   The LCD module processes the input data from GDMA-AXI and sends it to the YUV-RGB converter.
2. Stage Two: Data color space conversion
   The YUV-RGB converter processes the data. If the YUV-RGB converter is disabled, the data remains unchanged at this stage.
3. Stage Three: Data post-processing
   The LCD module gets data from the YUV-RGB converter for post-processing and then transmits the data to GPIO.

Note:
The output data from the previous stage is the input data for the subsequent stage.

The detailed configuration of each stage is described below.

Stage One: Data Preprocessing

At this stage, configure the following registers to adjust the bit width, bit order, and byte order of the data from GDMA-AXI.

- LCD_CAM_LCD_BYTE_MODE
  - 0: The data width of the LCD input from GDMA-AXI is 8 bits.
  - 1: The data width of the LCD input from GDMA-AXI is 16 bits.
  - 2: The data width of the LCD input from GDMA-AXI is 24 bits.
- LCD_CAM_LCD_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the input data bit order.
- LCD_CAM_LCD_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16/24-bit modes.

For the detailed configuration, see Table 20.3-2.

Table 20.3-2. LCD Data Format Control – Stage One

GDMA-AXI Data Order[1]	LCD_CAM_LCD_BYTE_MODE	LCD_CAM_LCD_BIT_ORDER	LCD_CAM_LCD_BYTE_ORDER[2]	Output Data Order[3,4]
B0,B1,B2,B3,B4,B5	0	0	0	{B0}{B1}{B2}{B3}{B4}{B5}
		1	0	{B0'}{B1'}{B2'}{B3'}{B4'}{B5'}
	1	0	0	{B1,B0}{B3,B2}{B5,B4}
			1	{B0,B1}{B2,B3}{B4,B5}
		1	0	{B1',B0'}{B3',B2'}{B5',B4'}
			1	{B0',B1'}{B2',B3'}{B4',B5'}
	2	0	0	{B2,B1,B0}{B5,B4,B3}
			1	{B0,B1,B2}{B3,B4,B5}
		1	0	{B2',B1',B0'}{B5',B4',B3'}
			1	{B0',B1',B2'}{B3',B4',B5'}

[1] B0 ~ B5 represent the bytes of the data from GDMA-AXI, from low address to high address.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] In output data, the bits in {} are in big-endian, and are in parallel with each other, while the data of {} is in serial with the data of other {}. Data is sent out from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is sent out first.

[4] In output data, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3,4,5).

Stage Two: Data Color Space Conversion

At this stage, the data undergoes color space conversion. For detailed configurations, please refer to Section 20.3.6.

> **Note:**
> Users can skip Stage Two by configuring LCD_CAM_LCD_CONV_ENABLE as 0, in which case the output of Stage One will directly become the input of Stage Three.

Stage Three: Data Post-processing

At this stage, configure the following registers to adjust the bit and byte order of the output data.

- LCD_CAM_LCD_WIRE_MODE
  - 0: The bit width of the data transmitted to GPIO is 8 bits.
  - 1: The bit width of the data transmitted to GPIO is 16 bits.
  - 2: The bit width of the data transmitted to GPIO is 24 bits.
- LCD_CAM_LCD_DOUT_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the output data bit order of every byte.
- LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE
  - 0: Disable LCD output data byte reordering.
  - 1: Enable LCD output data byte reordering by configuring LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE. For the reordering results see Table 20.3-3.

Table 20.3-3. LCD Output Data Reordering

Stage Three Input Data Byte Order[1]	LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE[2]	Stage Three Output Data Byte Order[3]
B0,B1,B2,B3,B4,B5	0	B1,B0,B3,B2,B5,B4
	1	B0,B2,B1,B3,B5,B4
	2	B1,B0,B2,B4,B3,B5
	3	B1,B2,B0,B4,B5,B3
	4	B2,B0,B1,B5,B3,B4
	5	B2,B1,B0,B5,B4,B3

[1] B0 ~ B5 represent the bytes of the input data in stage three, i.e., the input data is arranged in a little-endian byte sequence. For example, when the valid data width of the input data is 16 bits, the original input data sequence B1,B0B3,B2B5,B4 is converted to B0,B1,B2,B3,B4,B5.

[2] Only the values listed in the table are valid. Other values may cause unexpected data errors.

[3] The byte order of data that is ultimately transmitted to GPIO is determined by the "Stage Three Output Data Byte Order" and the bit width. For example, if the "Stage Three Output Data Byte Order" is B1,B0,B2,B4,B3,B5 and the transmitted data width is 16 bits, then the transmitted data byte order would be {B0,B1}{B4,B2}{B5,B3}. In this sequence, the data within each {} is in big-endian parallel format, and the data in each {} is in serial with the data in other {}. Data is transmitted from left to right. In this particular example, the data in {B0,B1} are in parallel with each other, while the data in {B0,B1} is in serial with the data in {B4,B2} and {B5,B3}. {B0,B1} is transmitted first.

20.3.5.2 Camera Data Format Control

When the Camera module receives data, configure the following registers to adjust the bit/byte order of the data sent to GDMA-AXI.

- LCD_CAM_CAM_2BYTE_EN
  - 0: The data width of the Camera input is 8 bits.
  - 1: The data width of the Camera input is 16 bits.
- LCD_CAM_CAM_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert data bit order.
    - Invert CAM_DATA_in[7:0] to CAM_DATA_in[0:7] in 8-bit mode.
    - Invert CAM_DATA_in[15:0] to CAM_DATA_in[0:15] in 16-bit mode.
- LCD_CAM_CAM_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16-bit mode.

For the detailed configuration, see Table 20.3-4.

Table 20.3-4. Camera Data Format Control

RX Data Order[1]	LCD_CAM_CAM_2BYTE_EN	LCD_CAM_CAM_BIT_ORDER[2]	LCD_CAM_CAM_BYTE_ORDER[2]	GDMA-AXI Data Order[3,4]
{B0}{B1}{B2}{B3}	0	0	0	B0,B1,B2,B3
		1	0	B0',B1',B2',B3'
{B1,B0}{B3,B2}	1	0	0	B0,B1,B2,B3
			1	B1,B0,B3,B2
		1	0	B0',B1',B2',B3'
			1	B1',B0',B3',B2'

[1] In RX data, the bits in {} are in big-endian, and are in parallel with each other, while the data in {} is in serial with the data in other {}. Data is received from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is received first.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] B0 ~ B3 represent the bytes of the data to GDMA-AXI, from low address to high address.

[4] In the data to GDMA-AXI, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3).

Note:
If only one byte is received each time, CAM_Data_in[7:0] is valid data. For such case, users must connect CAM_Data_in[7:0] with the master.

20.3.6 YUV-RGB Data Format Conversion

LCD_CAM is capable of converting data formats between YUV and RGB. The LCD module and Camera module each have a data format converter. The converters support format conversion:

- under BT601 and BT709 standards
- from RGB565 to YUV422/420/411 (full/limited range)
- from YUV422/420/411 (full/limited range) to RGB565
- between YUV422/420/411 (full/limited range) formats

The converter in the LCD module also supports the following features:

- The conversion from YUV422/420/411 (full/limited range) to RGB888 (full/limited range).
- The conversion from RGB565 to YUV444 (full/limited range).
- The conversion between RGB888 (full/limited range) and RGB565.
- The conversion from YUV422/420/411 (full/limited range) to YUV444 (full/limited range).
- When YUV420 upsampling is disabled, users can enable LCD_CAM_LCD_TRANS_BUFFER_ENA to use YUV420 upsampling SRAM as a data buffer to improve robustness.

20.3.6.1 YUV Formats

In LCD_CAM module, assume that there are 8 pixels to be transmitted, corresponding to YUV data [Y_i, U_i, V_i] (i = 1 ~ 8). Then:

- In YUV422 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	V_1	Y_3	U_3	Y_4	V_3	Y_5	U_5	Y_6	V_5	Y_7	U_7	Y_8	V_7

- In YUV420 mode, the LCD sends (or the camera receives) the data as follows.
  - Odd lines

U_1	Y_1	Y_2	U_3	Y_3	Y_4

  - Even lines

V_1	Y_1	Y_2	V_3	Y_3	Y_4

- In YUV411 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	Y_3	V_3	Y_4	Y_5	U_5	Y_6	Y_7	V_7	Y_8

20.3.6.2 Format Conversion Configuration

The configuration process for the format conversion in the Camera module is almost identical to that in the LCD module. The following is an example of format conversion in the LCD module.

1. Enable YUV-RGB format converter by setting LCD_CAM_LCD_CONV_ENABLE.
2. Configure the valid bit width of the input data by configuring LCD_CAM_LCD_CONV_MODE_8BITS_ON:
   - 0: The valid bit width is 16 bits.
   - 1: The valid bit width is 8 bits.
3. Configure the valid bit width of the output data:
   - For the format conversion in the Camera module, the valid bit width of the output data is equal to that of the input data.
   - For the format conversion in the LCD module, when LCD_CAM_LCD_WIRE_MODE is:
     - 2: The valid bit width of the format converter's output data is 24 bits.
     - Other value: The valid bit width of the format converter's output data is equal to that of the input data.
4. Select the standard by configuring LCD_CAM_LCD_CONV_PROTOCOL_MODE:
   - 0: BT601 standard
   - 1: BT709 standard
5. Configure the conversion mode:

- In 24-bit output mode:

Table 20.3-5. Conversion Mode Control in 24-bit Mode

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV444	1	-	3	-
RGB565 → RGB888	0	-	-	1
YUV422 → RGB888	0	0	3	-
YUV420 → RGB888	0	1	3	-
YUV411 → RGB888	0	2	3	-
YUV422 → YUV444	1	0	0/1/2	-
YUV420 → YUV444	1	1	0/1/2	-
YUV411 → YUV444	1	2	0/1/2	-

[1] The value of LCD_CAM_LCD_CONV_TRANS_MODE
[2] The value of LCD_CAM_LCD_CONV_YUV_MODE
[3] The value of LCD_CAM_LCD_CONV_YUV2YUV_MODE
[4] The value of LCD_CAM_LCD_CONV_RGB2RGB_MODE

- In other output modes:

Table 20.3-6. Conversion Mode Control in Other Modes

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV422	1	0	3	-
RGB565 → YUV420	1	1	3	-
RGB565 → YUV411	1	2	3	-
RGB888 → RGB565	0	-	-	0
YUV422 → RGB565	0	0	3	-
YUV420 → RGB565	0	1	3	-
YUV411 → RGB565	0	2	3	-
YUV422 → YUV420	1	0	1	-
YUV422 → YUV411	1	0	2	-
YUV420 → YUV422	1	1	0	-
YUV420 → YUV420	1	1	1	-
YUV411 → YUV422	1	2	0	-
YUV411 → YUV420	1	2	1	-

[1] The value of LCD_CAM_LCD_CONV_TRANS_MODE
[2] The value of LCD_CAM_LCD_CONV_YUV_MODE
[3] The value of LCD_CAM_LCD_CONV_YUV2YUV_MODE
[4] The value of LCD_CAM_LCD_CONV_RGB2RGB_MODE

6. Configure the color range for the input data by configuring LCD_CAM_LCD_CONV_DATA_IN_MODE:
   - 0: limited color range[1]
   - 1: full color range[2]
7. Configure the color range for the output data by configuring LCD_CAM_LCD_CONV_DATA_OUT_MODE:

- 0: limited color range[1]
- 1: full color range[2]

Note:

1. If the limited color range is selected,
   - the color range of RGB is: 16 ~ 235.
   - the color range of YUV is:
     - Y: 16 ~ 235.
     - U-V: 16 ~ 240.
2. If the full color range is selected, the color range of RGB or YUV is 0 ~ 255.

20.3.7 LCD_CAM Timing

20.3.7.1 LCD Timing (RGB Format)

Figure 20.3-4 shows the LCD frame structure.

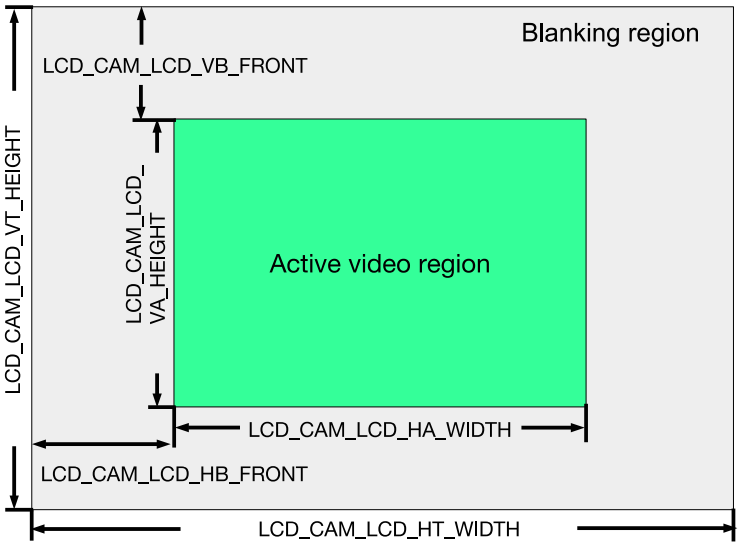


Figure 20.3-4. LCD Frame Structure

As depicted in the figure, the frame structure can be set up with the following registers:

- LCD_CAM_LCD_VT_HEIGHT
- LCD_CAM_LCD_VA_HEIGHT
- LCD_CAM_LCD_HB_FRONT
- LCD_CAM_LCD_HT_WIDTH
- LCD_CAM_LCD_HA_WIDTH
- LCD_CAM_LCD_VB_FRONT

Figure 20.3-5 shows the timing of an LCD full frame.

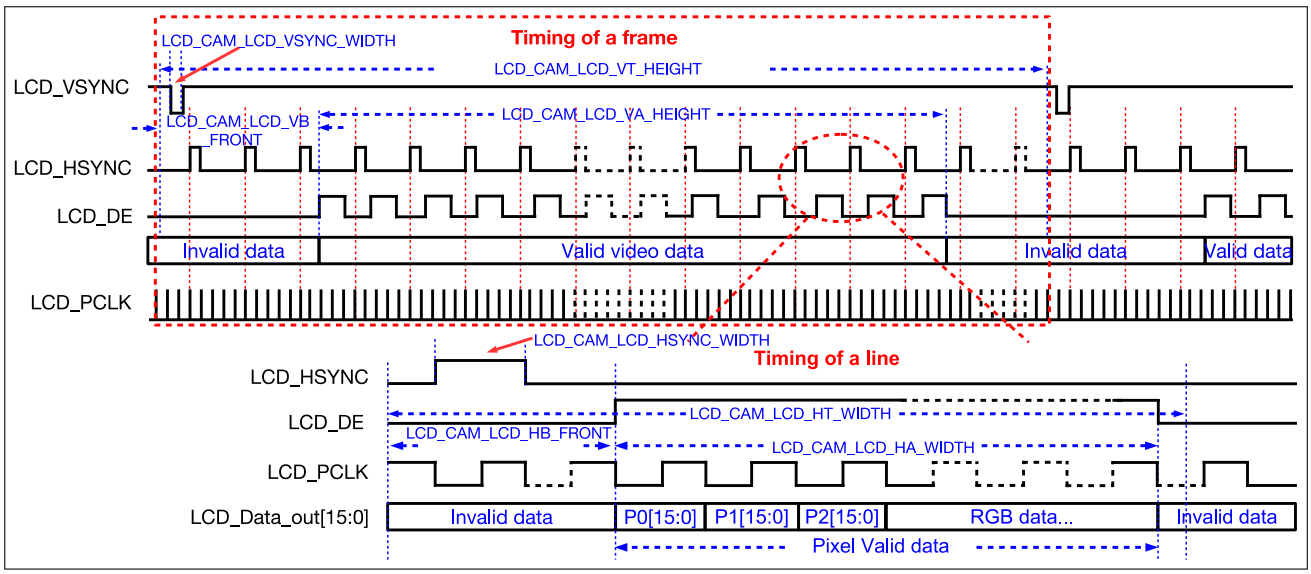


Figure 20.3-5. LCD Timing (RGB Format)

Note:

When configuring the parameters shown in the figures above, please note that the parameter is equal to the register value + 1. For example, if the expected VSYNC width is 1, then please configure LCD_CAM_LCD_VSYNC_WIDTH to 0. For more information, see the register description.

20.3.7.2 LCD Timing (I8080/MOTO6800 Format)

Figure 20.3-6 shows the LCD timing sequence in I8080/MOTO6800 format.

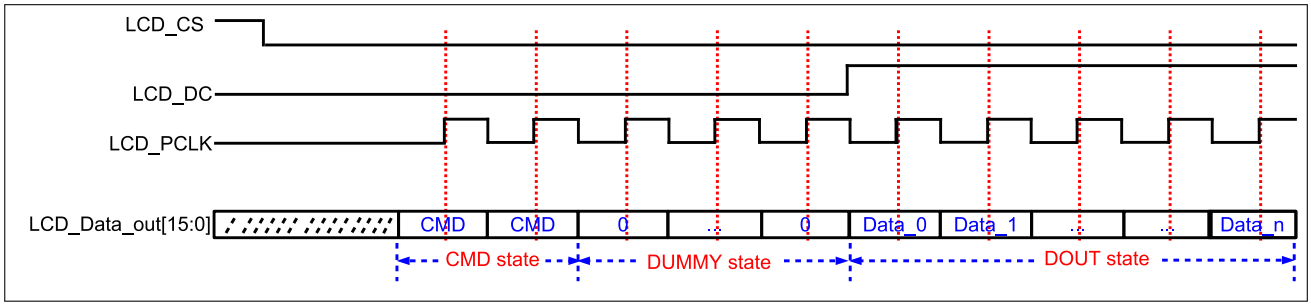


Figure 20.3-6. LCD Timing (I8080/MOTO6800 Format)

20.4 Interrupts

ESP32-S31's LCD_CAM can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- LCD_CAM_INTR

The following internal interrupt sources from LCD_CAM can generate LCD_CAM_INTR:

- LCD_CAM_CAM_HS_INT: Triggered when the Camera module receives lines more than or equal to the value of LCD_CAM_CAM_LINE_INT_NUM + 1.

- LCD_CAM_CAM_VSYNC_INT: Triggered when the Camera module receives a whole frame.
- LCD_CAM_LCD_TRANS_DONE_INT: Triggered when the LCD module finishes the transmission.
- LCD_CAM_LCD_VSYNC_INT: Triggered when the LCD module transmits a whole frame.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 20.6 *Register Summary*.

20.5 Software Configuration Process

Note:
Updating the configurations of the LCD module or the Camera module requires setting the LCD_CAM_LCD_UPDATE and LCD_CAM_CAM_UPDATE respectively to synchronize registers from the APB clock domain to the LCD/camera clock domain. Check the configuration examples below for more details.

20.5.1 Configure LCD (RGB Format) as TX Mode

Follow the steps below to configure LCD (RGB format) as TX mode via software:

1. Configure clock according to Section 20.3.3.
2. Configure signal pins according to Table 20.3-1 and Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.
3. Enable corresponding interrupts, see Section 20.4.
4. Enable RGB format by setting LCD_CAM_LCD_RGB_MODE_EN.
5. Configure frame structure according to Section 20.3.7.1.
6. Set LCD_CAM_LCD_UPDATE so that the configurations take effect.
7. Reset TX control unit (LCD_Ctrl) and Async TX FIFO as described in Section 20.3.4.
8. Configure GDMA-AXI outlink, see Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
9. Start transmitting data:
   - Wait till LCD slave gets ready.
   - Set LCD_CAM_LCD_START to start transmitting data.
10. Wait for the interrupt signals set in Step 3.
11. In frame intervals during data transmission, check if LCD_CAM_LCD_NEXT_FRAME_EN is set:
    - If yes, LCD continues transmitting next frame.
    - If not, LCD stops transmitting data.
12. Clear LCD_CAM_LCD_START if data transmission is done.

20.5.2 Configure LCD (I8080/MOTO6800 Format) as TX Mode

Follow the steps below to configure LCD (I8080/MOTO6800 format) as TX mode via software:

1. Configure clock according to Section 20.3.3.
2. Configure signal pins according to Table 20.3-1 and Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.
3. Enable corresponding interrupts, see Section 20.4.
4. Clear LCD_CAM_LCD_RGB_MODE_EN to disable RGB format.
5. Configure CMD phase with LCD_CAM_LCD_CMD, LCD_CAM_LCD_CMD_2_CYCLE_EN, LCD_CAM_LCD_FIRST_CMD_VALUE, and LCD_CAM_LCD_LATTER_CMD_VALUE.
6. Configure DUMMY phase with LCD_CAM_LCD_DUMMY and LCD_CAM_LCD_DUMMY_CYCLELEN.
7. Configure DOUT phase with LCD_CAM_LCD_DOUT, depending on the output mode:
   - In a fixed-length output[a], configure data length in LCD_CAM_LCD_DOUT_CYCLELEN.
   - In a continuous output[b], set LCD_CAM_LCD_ALWAYS_OUT_EN. Users do not need to configure LCD_CAM_LCD_DOUT_CYCLELEN.

   Note:

   (a) In a fixed-length output, the LCD module stops sending data once the data length reaches the value set in LCD_CAM_LCD_DOUT_CYCLELEN.

   (b) In a continuous output, LCD module keeps sending data till:
   - LCD_CAM_LCD_START is cleared;
   - or LCD_CAM_LCD_RESET is set;
   - or all the data in GDMA-AXI is sent out.
8. Configure the DC signal mode, including the default value of the DC signal and the values at each phase, see the description of LCD_CAM_LCD_MISC_REG.
9. Set LCD_CAM_LCD_UPDATE so that the configurations take effect.
10. Reset TX control unit (LCD_Ctrl) and Async TX FIFO as described in Section 20.3.4.
11. Configure GDMA-AXI outlink, see Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.
12. Start transmitting data:
    - Wait till LCD slave gets ready.
    - Set LCD_CAM_LCD_START to start transmitting data.
13. Wait for the interrupt signals set in Step 3.
14. Clear LCD_CAM_LCD_START if data transmission is done.

Notes:

No matter in which format, RGB or I8080/MOTO6800, the rules below must be followed when accessing internal and external memory via GDMA-AXI:

- If LCD data bus is configured to 8-bit parallel output mode, then
  - The pixel clock frequency must be less than 80 MHz.
  - If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 60 MHz.
- If LCD data bus is configured to 16-bit parallel output mode, then
  - The pixel clock frequency must be less than 40 MHz.
  - If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

20.5.3 Configure Camera as RX Mode

Follow the steps below to configure the Camera as RX mode via software:

1. Configure clock according to Section 20.3.3. Note that in slave mode, the module clock frequency should be more than two times as fast as the PCLK frequency of the image sensor.
2. Configure signal pins according to Table 20.3-1 and Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.
3. Set or clear LCD_CAM_CAM_VH_DE_MODE_EN according to the control signal HSYNC.
4. Set needed RX data format according to Section 20.3.5.2.
5. Set LCD_CAM_CAM_UPDATE so that the configurations take effect.
6. Reset RX control unit (Camera_Ctrl) and Async RX FIFO as described in Section 20.3.4.
7. Enable corresponding interrupts, see Section 20.4.
8. Configure GDMA-AXI inlink, and set the length of RX data in LCD_CAM_CAM_REC_DATA_BYTELEN.
9. Start receiving data:
   - In master mode, when the slave is ready, set LCD_CAM_CAM_START to start receiving data.
   - In slave mode, set LCD_CAM_CAM_START. Receiving data starts after the master provides clock and control signals.
10. Receive data and store the data to GDMA-AXI. Then corresponding interrupts set in Step 7 will be generated.

Notes:

- No matter in which operation mode, camera master RX mode or camera slave RX mode, the rules below must be followed:
  - If 8-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 80 MHz.
    * If YUV-RGB format is being used at the same time, the pixel clock frequency must be less than 60 MHz.
  - If 16-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 40 MHz.

* If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

Note:

If both LCD and camera are connected externally at the same time, it is necessary to ensure that the maximum data throughput on the interface is less than the total data bandwidth of GDMA-AXI when accessing internal/external storage. The default frequency of AXI_CLK is 80 MHz in this scenario. For more information about AXI_CLK, see Chapter *7 Reset and Clock [to be added later]*.

20.6 Register Summary

The addresses in this section are relative to LCD_CAM base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

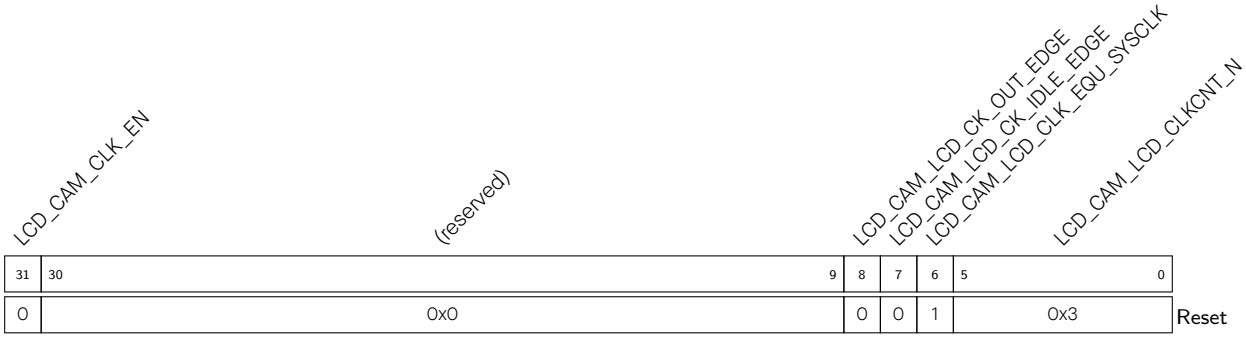
Name	Description	Address	Access
LCD Configuration Registers			
LCD_CAM_LCD_CLOCK_REG	LCD clock configuration register	0x0000	R/W
LCD_CAM_LCD_RGB_YUV_REG	LCD data format conversion configuration register	0x0010	R/W
LCD_CAM_LCD_USER_REG	LCD user configuration register	0x0014	varies
LCD_CAM_LCD_MISC_REG	LCD MISC configuration register	0x0018	varies
LCD_CAM_LCD_RGB_VERTICAL_REG	LCD vertical parameter signal configuration register	0x001C	R/W
LCD_CAM_LCD_RGB_HORIZONTAL_REG	LCD horizontal parameter signal configuration register	0x0020	R/W
LCD_CAM_LCD_RGB_BLANK_REG	LCD blank parameter signal configuration register	0x0024	R/W
LCD_CAM_LCD_RGB_CTRL_REG	LCD timing parameter signal configuration register	0x0028	R/W
LCD_CAM_LCD_FIRST_CMD_VAL_REG	LCD CMD value configuration register	0x002C	R/W
LCD_CAM_LCD_LATTER_CMD_VAL_REG	LCD CMD value configuration register	0x0030	R/W
LCD_CAM_LCD_DLY_MODE_CFG1_REG	LCD data/signal delay mode configuration register	0x0034	R/W
LCD_CAM_LCD_DLY_MODE_CFG2_REG	LCD data delay mode configuration register	0x0038	R/W
LCD_CAM_LCD_TRANS_BUFF_CFG_REG	LCD transfer buffer configuration register	0x003C	R/W
Camera Configuration Registers			
LCD_CAM_CAM_CTRL_REG	Camera clock configuration register	0x0004	R/W
LCD_CAM_CAM_CTRL1_REG	Camera control register	0x0008	varies
LCD_CAM_CAM_RGB_YUV_REG	Camera data format conversion configuration register	0x000C	R/W
Interrupt Registers			
LCD_CAM_LC_DMA_INT_ENA_REG	LCD_CAM GDMA-AXI interrupt enable register	0x0064	R/W
LCD_CAM_LC_DMA_INT_RAW_REG	LCD_CAM GDMA-AXI raw interrupt register	0x0068	RO
LCD_CAM_LC_DMA_INT_ST_REG	LCD_CAM GDMA-AXI masked interrupt status register	0x006C	RO
LCD_CAM_LC_DMA_INT_CLR_REG	LCD_CAM GDMA-AXI interrupt clear register	0x0070	WO
Version Register			
LCD_CAM_LC_DATE_REG	Version control register	0x00FC	R/W

20.7 Registers

The addresses in this section are relative to LCD_CAM base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 20.1. LCD_CAM_LCD_CLOCK_REG (0x0000)



LCD_CAM_LCD_CLKCNT_N Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLK_EQU_SYSCLK.

When LCD_CAM_LCD_CLK_EQU_SYSCLK = 0,

$f_{LCD\_PCLK} = f_{LCD\_CLK}/(LCD\_CAM\_LCD\_CLKCNT\_N + 1)$

Note: this field must not be configured to 0.

(R/W)

LCD_CAM_LCD_CLK_EQU_SYSCLK Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLKCNT_N.

0: $f_{LCD\_PCLK} = f_{LCD\_CLK}/(LCD\_CAM\_LCD\_CLKCNT\_N + 1)$

1: $f_{LCD\_PCLK} = f_{LCD\_CLK}$

(R/W)

LCD_CAM_LCD_CK_IDLE_EDGE Configures the level of LCD_PCLK in idle state.

0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_LCD_CK_OUT_EDGE Configures the level of LCD_PCLK in the first half clock cycle.

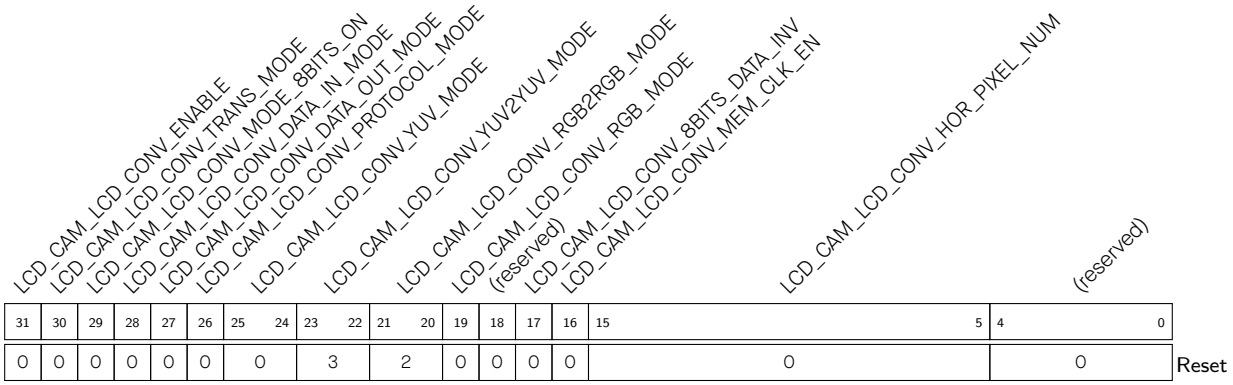
0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_CLK_EN Configures whether to force enable the clock for all configuration registers. Clock gate is not used. (R/W)

Register 20.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)



LCD_CAM_LCD_CONV_HOR_PIXEL_NUM Configures the number of horizontal pixels in YUV420 mode. (R/W)

LCD_CAM_LCD_CONV_MEM_CLK_EN Configures whether to enable the color conversion MEM CLOCK GATE.

0: Enable CLOCK GATE

1: Disable CLOCK GATE

(R/W)

LCD_CAM_LCD_CONV_8BITS_DATA_INV Configures whether to swap every two 8-bit GDMA-AXI input data.

0: Do not swap.

1: Swap.

(R/W)

LCD_CAM_LCD_CONV_RGB_MODE Configures the output RGB data format.

0: Data is converted to RGB565

1: Data is converted to RGB888

(R/W)

LCD_CAM_LCD_CONV_RGB2RGB_MODE Configures data format conversion in RGB-to-RGB mode.

0: Data is converted to RGB565 format.

1: Data is converted to RGB888 format.

2/3: Data format conversion is disabled.

(R/W)

LCD_CAM_LCD_CONV_YUV2YUV_MODE Configures data format conversion in YUV-to-YUV mode.

0: Data is converted to YUV422 format.

1: Data is converted to YUV420 format.

2: Data is converted to YUV411 format.

3: Data format conversion is disabled.

Only valid when LCD_CAM_LCD_CONV_TRANS_MODE = 1.

(R/W)

Continued on the next page...

Register 20.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)

Continued from the previous page...

LCD_CAM_LCD_CONV_YUV_MODE Configures the YUV format of input data in YUV-to-YUV mode and YUV-to-RGB mode.

0: Input data is in YUV422 format.

1: Input data is in YUV420 format.

2: Input data is in YUV411 format.

(R/W)

LCD_CAM_LCD_CONV_PROTOCOL_MODE Configures the data format conversion standard.

0: BT601.

1: BT709.

(R/W)

LCD_CAM_LCD_CONV_DATA_OUT_MODE Configures the color range for output data.

0: Limited color range.

1: Full color range.

(R/W)

LCD_CAM_LCD_CONV_DATA_IN_MODE Configures the color range for input data.

0: Limited color range.

1: Full color range.

(R/W)

LCD_CAM_LCD_CONV_MODE_8BITS_ON Configures the valid bit width of the YUV-RGB converter's input data.

0: 16 bits.

1: 8 bits.

(R/W)

LCD_CAM_LCD_CONV_TRANS_MODE Configures the RGB or YUV format for data conversion.

0: Converted to RGB format.

1: Converted to YUV format.

(R/W)

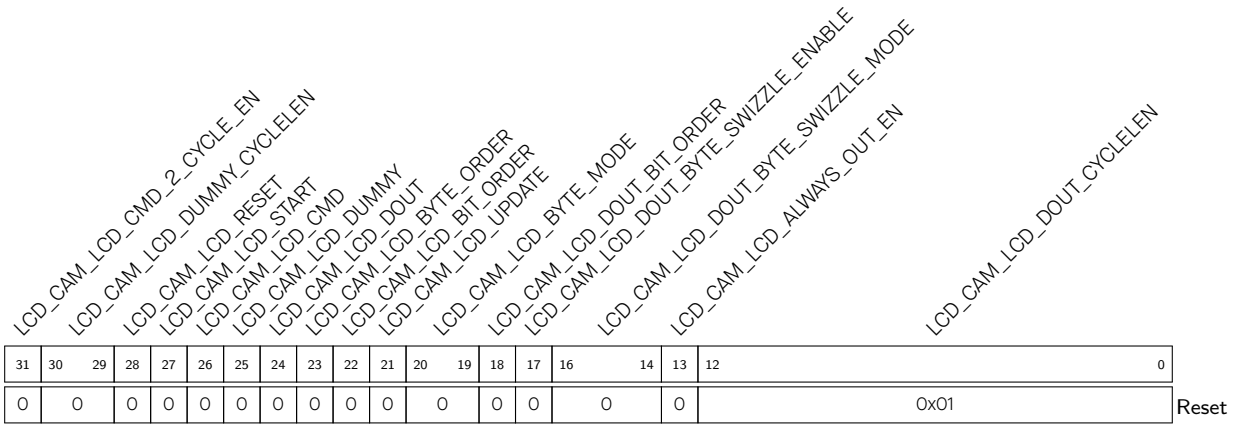
LCD_CAM_LCD_CONV_ENABLE Configures whether to enable YUV-RGB converter.

0: Bypass the converter.

1: Enable the converter.

(R/W)

Register 20.3. LCD_CAM_LCD_USER_REG (0x0014)



LCD_CAM_LCD_DOUT_CYCLELEN Configures the period of the LCD module's data output phase (DOUT). The actual period duration = the value of this field + 1. (R/W)

LCD_CAM_LCD_ALWAYS_OUT_EN Enables continuous output of the LCD module. In this mode, the LCD module continuously outputs data during the DOUT phase, until LCD_CAM_LCD_START is cleared or LCD_CAM_LCD_RESET is set. (R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE Configures LCD output data byte reordering. For details please refer to Table 20.3-3. (R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE Configures whether to enable LCD output data byte reordering.

0: Disable

1: Enable

(R/W)

LCD_CAM_LCD_DOUT_BIT_ORDER Configures whether to invert the LCD output data bit order.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_out[n-1:0] is inverted to LCD_DATA_out[0:n-1].

(R/W)

LCD_CAM_LCD_BYTE_MODE Configures the data bit width from GDMA-AXI.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_UPDATE Configures whether to update the LCD register configurations.

0: Do not update.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_LCD_BIT_ORDER Configures whether to invert the bit order of the LCD input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_in[n-1:0] is inverted to LCD_DATA_in[0:n-1].

(R/W)

Continued on the next page...

Register 20.3. LCD_CAM_LCD_USER_REG (0x0014)

Continued from the previous page...

LCD_CAM_LCD_BYTE_ORDER Configures whether to invert the byte order of the LCD input data.

0: Do not invert.

1: Invert data byte order, only valid in 16/24-bit mode.

(R/W)

LCD_CAM_LCD_DOUT Configures the LCD DOUT phase.

0: Disable.

1: The LCD module sends data in RGB/I8080 format.

(R/W)

LCD_CAM_LCD_DUMMY Configures the LCD DUMMY phase.

0: Disable.

1: Enable the DUMMY phase when LCD initiates.

(R/W)

LCD_CAM_LCD_CMD Configures the LCD CMD phase.

0: Disable.

1: The LCD module sends commands.

(R/W)

LCD_CAM_LCD_START When set to 1, the LCD module starts transmitting data. (R/W)

LCD_CAM_LCD_RESET When set to 1, the LCD module is reset. (WO)

LCD_CAM_LCD_DUMMY_CYCLELEN Configures DUMMY cycles. DUMMY cycles = this value + 1. (R/W)

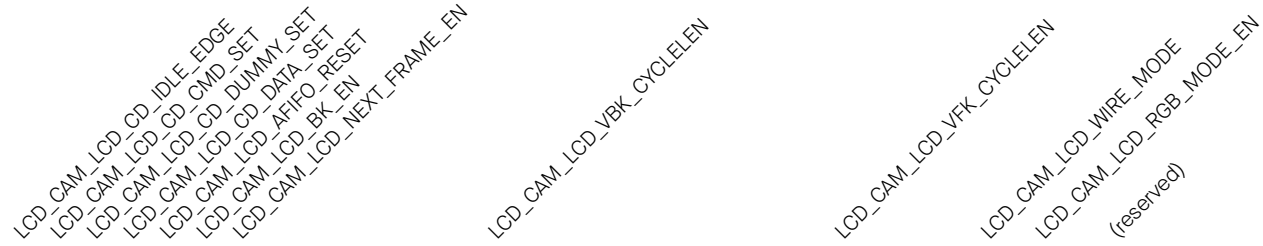
LCD_CAM_LCD_CMD_2_CYCLE_EN Configures the number of cycles of the Command phase.

0: One cycle.

1: Two cycles.

(R/W)

Register 20.4. LCD_CAM_LCD_MISC_REG (0x0018)



LCD_CAM_LCD_RGB_MODE_EN Configures whether to enable RGB mode.

0: Disable RGB mode.

1: Enable RGB mode.

(R/W)

LCD_CAM_LCD_WIRE_MODE Configures the bit width of LCD output data to GPIO.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_VFK_CYCLELEN Configures the clock cycles of the setup time in LCD non-RGB mode. Setup clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_VBK_CYCLELEN Configures the clock cycles of the hold time in LCD non-RGB mode. Hold clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_NEXT_FRAME_EN Configures whether to send the next frame.

0: The LCD module stops when the current frame is sent out.

1: The LCD module continues sending the next frame when the current frame is sent out.

(R/W)

LCD_CAM_LCD_BK_EN Configures whether to enable blanking region when LCD sends data.

0: No blanking region.

1: Enable blanking region.

(R/W)

LCD_CAM_LCD_AFIFO_RESET When set to 1, Async TX FIFO is reset. (WO)

LCD_CAM_LCD_CD_DATA_SET Configures LCD_DC value in DOUT phase for LCD I8080 mode.

0: LCD_DC = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_DC = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_DUMMY_SET Configures LCD_DC value in DUMMY phase for LCD I8080 mode.

0: LCD_DC = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_DC = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

Continued on the next page...

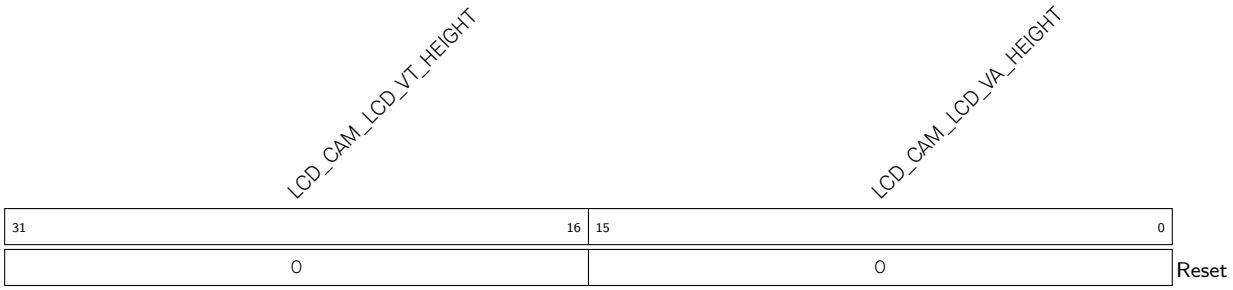
Register 20.4. LCD_CAM_LCD_MISC_REG (0x0018)

Continued from the previous page...

LCD_CAM_LCD_CD_CMD_SET Configures LCD_DC value in CMD phase for LCD I8080 mode.
0: LCD_DC = LCD_CAM_LCD_CD_IDLE_EDGE
1: LCD_DC = !LCD_CAM_LCD_CD_IDLE_EDGE
(R/W)

LCD_CAM_LCD_CD_IDLE_EDGE The default value of LCD_DC. (R/W)

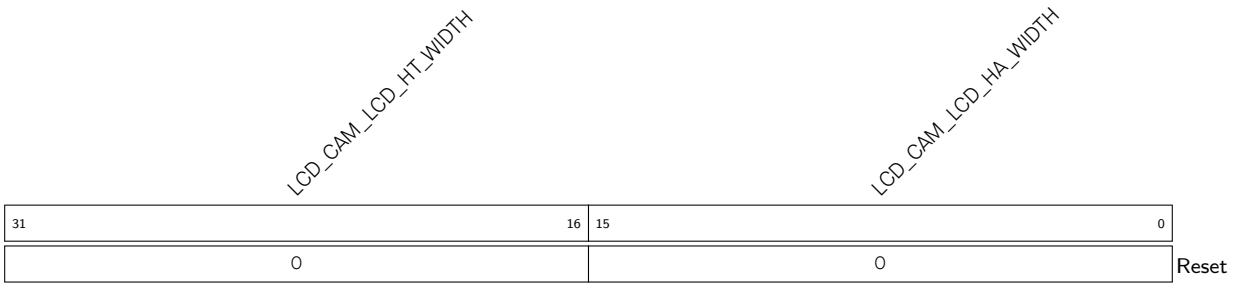
Register 20.5. LCD_CAM_LCD_RGB_VERTICAL_REG (0x001C)



LCD_CAM_LCD_VA_HEIGHT Configures the vertical active height of a frame. (R/W)

LCD_CAM_LCD_VT_HEIGHT Configures the vertical total height of a frame. (R/W)

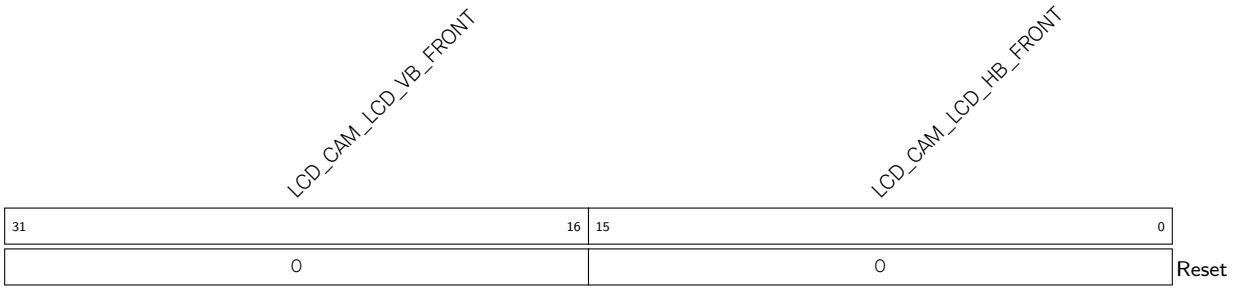
Register 20.6. LCD_CAM_LCD_RGB_HORIZONTAL_REG (0x0020)



LCD_CAM_LCD_HA_WIDTH Configures the horizontal active width of a frame. (R/W)

LCD_CAM_LCD_HT_WIDTH Configures the horizontal total width of a frame. (R/W)

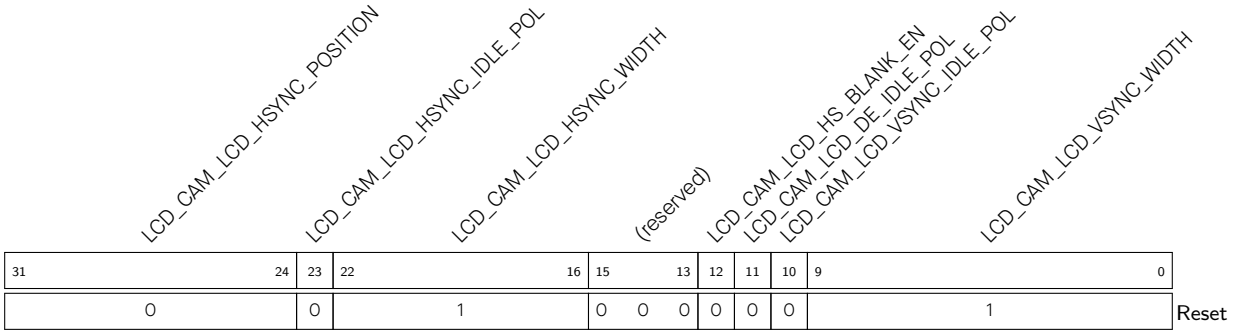
Register 20.7. LCD_CAM_LCD_RGB_BLANK_REG (0x0024)



LCD_CAM_LCD_VB_FRONT Configures the value of (VSYNC_WIDTH + vertical back porch). (R/W)

LCD_CAM_LCD_HB_FRONT Configures the value of (HSYNC_POSITION + HSYNC_WIDTH + horizontal back porch). (R/W)

Register 20.8. LCD_CAM_LCD_RGB_CTRL_REG (0x0028)



LCD_CAM_LCD_VSYNC_WIDTH Configures the width of LCD_VSYNC active pulse in a line. Expected width = this value + 1. (R/W)

LCD_CAM_LCD_VSYNC_IDLE_POL Configures the idle value of LCD_VSYNC.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_DE_IDLE_POL Configures the idle value of LCD_DE.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_HS_BLANK_EN Configures the LCD_HSYNC output in RGB mode.

0: LCD_HSYNC is output only in active video lines.

1: LCD_HSYNC can be output in vertical blanking intervals.

(R/W)

LCD_CAM_LCD_HSYNC_WIDTH Configures the width of LCD_HSYNC active pulse. Expected width = this value +1. (R/W)

LCD_CAM_LCD_HSYNC_IDLE_POL Configures the idle value of LCD_HSYNC.

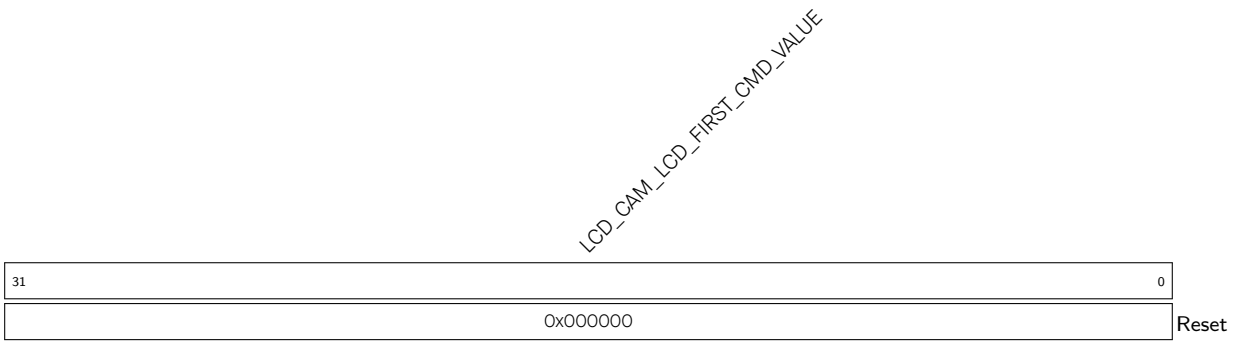
0: The idle value is low.

1: The idle value is high.

(R/W)

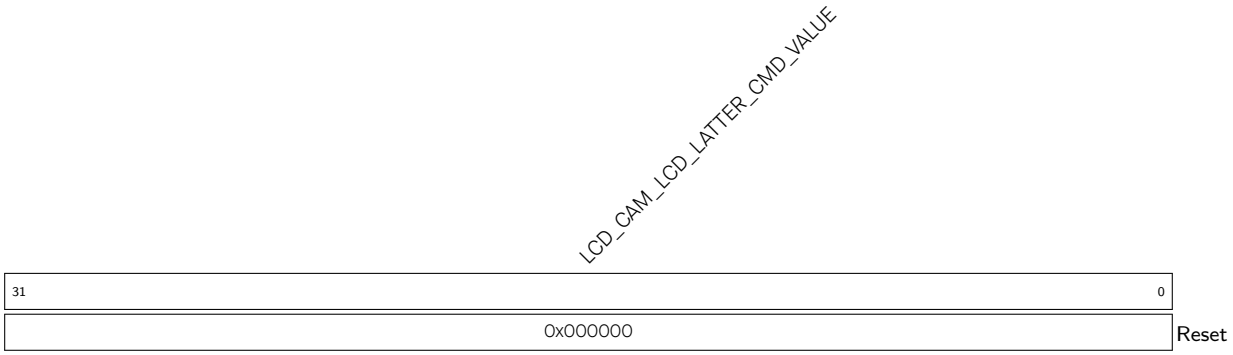
LCD_CAM_LCD_HSYNC_POSITION Configures the position of LCD_HSYNC active pulse. Expected position = this value + 1. Unit is a pixel. (R/W)

Register 20.9. LCD_CAM_LCD_FIRST_CMD_VAL_REG (0x002C)



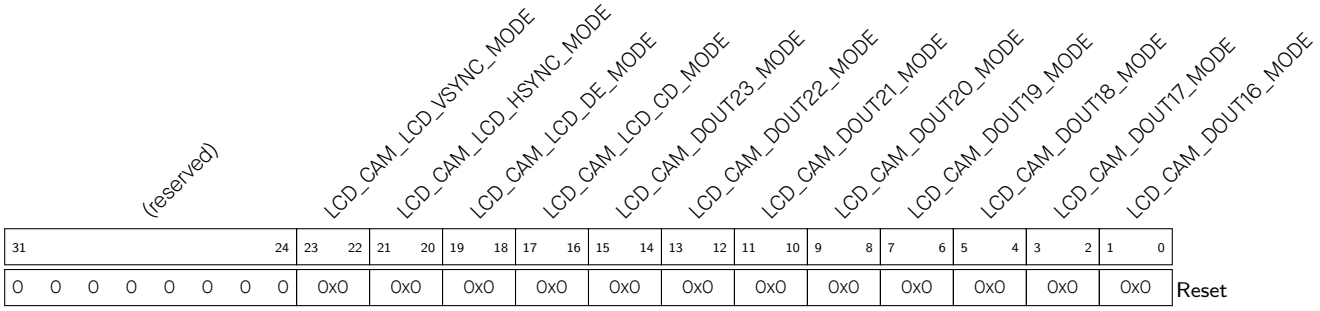
LCD_CAM_LCD_FIRST_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the first cycle of the CMD phase. (R/W)

Register 20.10. LCD_CAM_LCD_LATTER_CMD_VAL_REG (0x0030)



LCD_CAM_LCD_LATTER_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the second cycle of the CMD phase. (R/W)

Register 20.11. LCD_CAM_LCD_DLY_MODE_CFG1_REG (0x0034)



LCD_CAM_DOUT*n*_MODE (*n*: 16 - 23) Configures the delay of the output data bit (*n*).

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_CD_MODE Configures the delay of LCD_DC output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_DE_MODE Configures the delay of LCD_DE output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_HSYNC_MODE Configures the delay of LCD_HSYNC output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_VSYNC_MODE Configures the delay of LCD_VSYNC output signal.

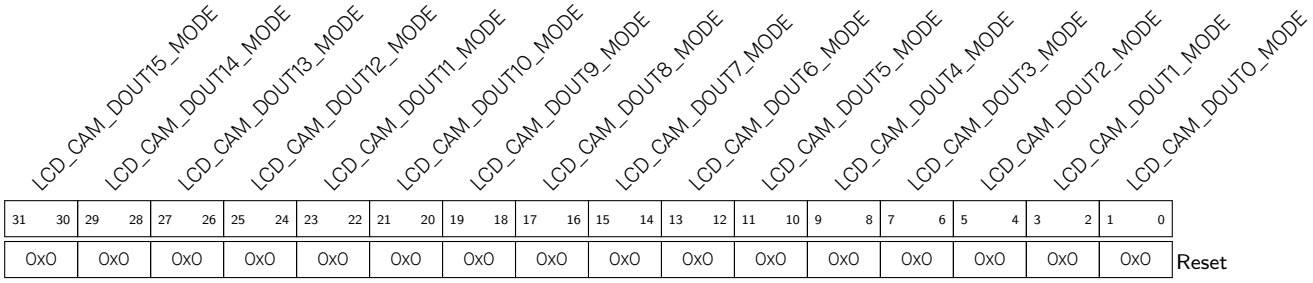
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 20.12. LCD_CAM_LCD_DLY_MODE_CFG2_REG (0x0038)



LCD_CAM_DOUT*n*_MODE (*n*: 0 - 15) Configures the delay of the output data bit *n*.

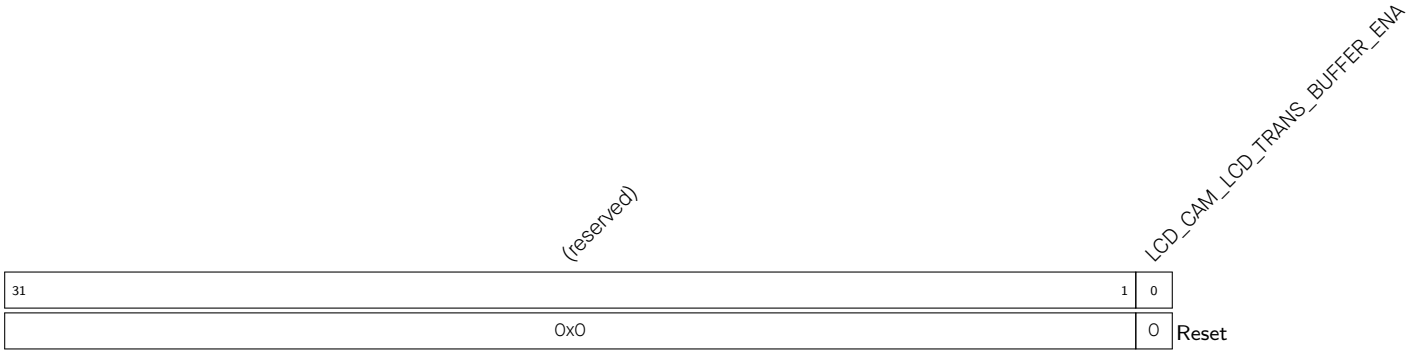
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 20.13. LCD_CAM_LCD_TRANS_BUFF_CFG_REG (0x003C)



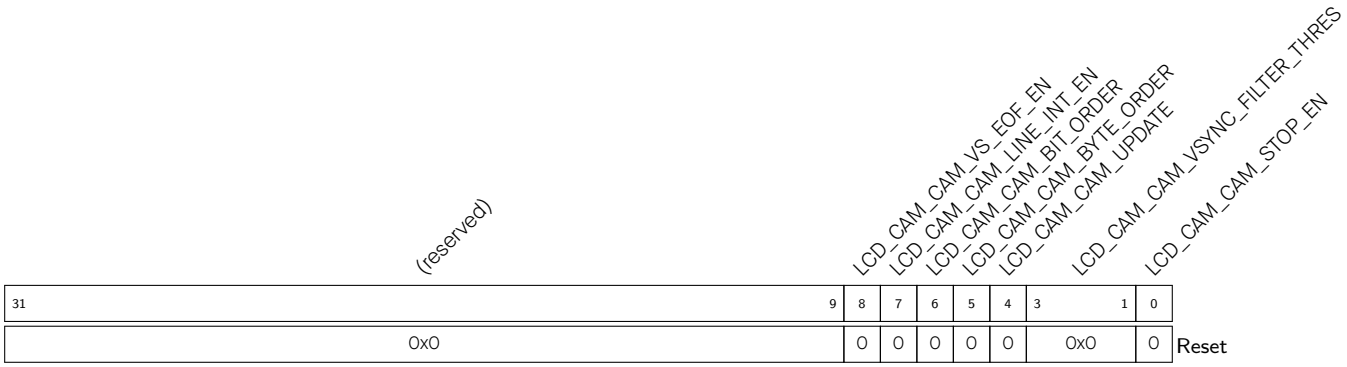
LCD_CAM_LCD_TRANS_BUFFER_ENA Configures whether to use SRAM as a data buffer.

0: Disable

1: Enable (YUV420 upsampling color conversion must be disabled)

(R/W)

Register 20.14. LCD_CAM_CAM_CTRL_REG (0x0004)



LCD_CAM_CAM_STOP_EN Configures whether to stop the Camera module.

0: Do not stop.

1: The Camera module stops when GDMA-AXI RX FIFO is full.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_THRES Configures the filter threshold value for CAM_VSYNC signal. (R/W)

LCD_CAM_CAM_UPDATE Configures whether to update the Camera register configurations.

0: No effect.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_CAM_BYTE_ORDER Configures whether to invert the byte order of the Camera's input data.

0: Do not invert.

1: Invert data byte order, only valid in 16-bit mode.

(R/W)

LCD_CAM_CAM_BIT_ORDER Configures whether to invert the bit order of the Camera's input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, CAM_DATA_in[n-1:0] is inverted to CAM_DATA_in[0:n-1]. n = 8 or 16.

(R/W)

LCD_CAM_CAM_LINE_INT_EN Configures whether the RAW bit of LCD_CAM_CAM_HS_INT can generate.

0: Disable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW cannot generate.

1: Enable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW can generate.

(R/W)

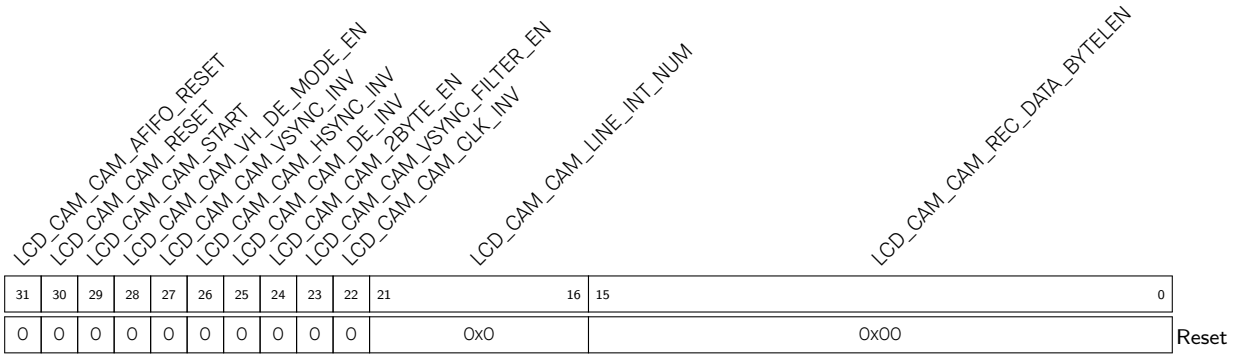
LCD_CAM_CAM_VS_EOF_EN Configures whether to enable CAM_VSYNC to generate in_suc_eof.

0: in_suc_eof is controlled by LCD_CAM_CAM_REC_DATA_BYTELEN.

1: in_suc_eof is controlled by CAM_VSYNC.

(R/W)

Register 20.15. LCD_CAM_CAM_CTRL1_REG (0x0008)



LCD_CAM_CAM_REC_DATA_BYTELEN Configures the data byte length received by the Camera module. When the length of received data reaches this value + 1, GDMA-AXI in_suc_eof_int is triggered. (R/W)

LCD_CAM_CAM_LINE_INT_NUM Configures the number of video lines. When the number of video lines reaches this value + 1, LCD_CAM_CAM_HS_INT is triggered. (R/W)

LCD_CAM_CAM_CLK_INV Configures whether to invert the input signal CAM_PCLK.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_EN Configures whether to enable CAM_VSYNC filter function.

0: Bypass.

1: Enable.

(R/W)

LCD_CAM_CAM_2BYTE_EN Configures the width of input data.

0: 8 bits.

1: 16 bits.

(R/W)

LCD_CAM_CAM_DE_INV Configures whether to invert the input signal CAM_DE.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_HSYNC_INV Configures whether to invert the input signal CAM_HSYNC.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_INV Configures whether to invert the input signal CAM_VSYNC.

0: Do not invert.

1: Invert.

(R/W)

Continued on the next page...

Register 20.15. LCD_CAM_CAM_CTRL1_REG (0x0008)

Continued from the previous page...

LCD_CAM_CAM_VH_DE_MODE_EN Configures the input control signals.

0: VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the camera module is not available.

1: VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines.

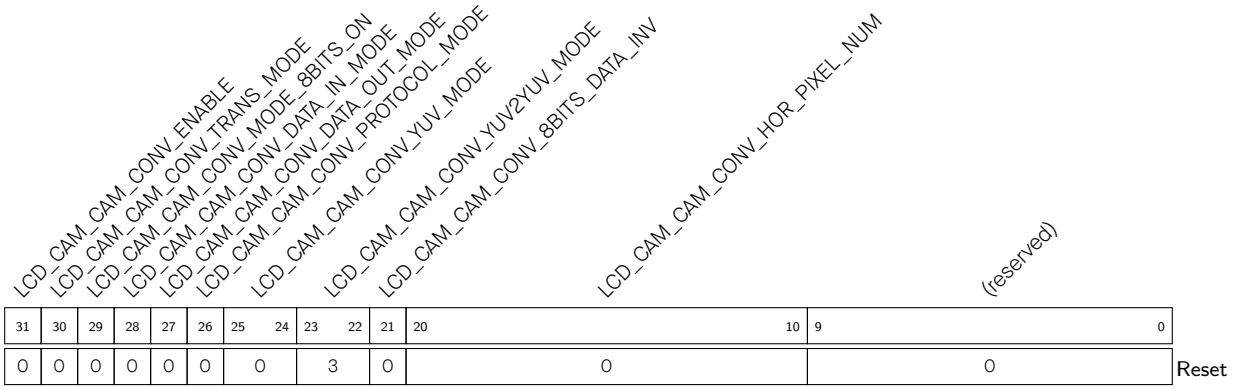
(R/W)

LCD_CAM_CAM_START Camera module start signal. (R/W)

LCD_CAM_CAM_RESET When set to 1, the Camera module is reset. (WO)

LCD_CAM_CAM_AFIFO_RESET When set to 1, the Camera Async RX FIFO is reset. (WO)

Register 20.16. LCD_CAM_CAM_RGB_YUV_REG (0x000C)



LCD_CAM_CAM_CONV_HOR_PIXEL_NUM Configures the number of horizontal pixels of the image in RGB-to-YUV420 mode.

LCD_CAM_CAM_CONV_8BITS_DATA_INV Configures whether to swap every two bytes of input data.

0: Do not swap.

1: Swap.

(R/W)

LCD_CAM_CAM_CONV_YUV2YUV_MODE Configures the data format conversion in YUV-to-YUV mode.

0: Data is converted to YUV422 format.

1: Data is converted to YUV420 format.

2: Data is converted to YUV411 format.

3: Disabled.

Valid only when LCD_CAM_CAM_CONV_TRANS_MODE = 1.

(R/W)

LCD_CAM_CAM_CONV_YUV_MODE Configures the YUV format of data in RGB-to-YUV mode, YUV-to-YUV mode, or YUV-to-RGB mode. In RGB-to-YUV mode, this field configures the converted YUV format; in YUV-to-YUV mode or YUV-to-RGB mode, this field configures the Camera input YUV format.

0: YUV422 format.

1: YUV420 format (Camera input does not support this mode when color space conversion is enabled).

2: YUV411 format.

(R/W)

LCD_CAM_CAM_CONV_PROTOCOL_MODE Configures the data format conversion standard.

0: BT601.

1: BT709.

(R/W)

LCD_CAM_CAM_CONV_DATA_OUT_MODE Configures the color range for the YUV-RGB converter's output data.

0: Limited color range.

1: Full color range.

(R/W)

Continued on the next page...

Register 20.16. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

Continued from the previous page...

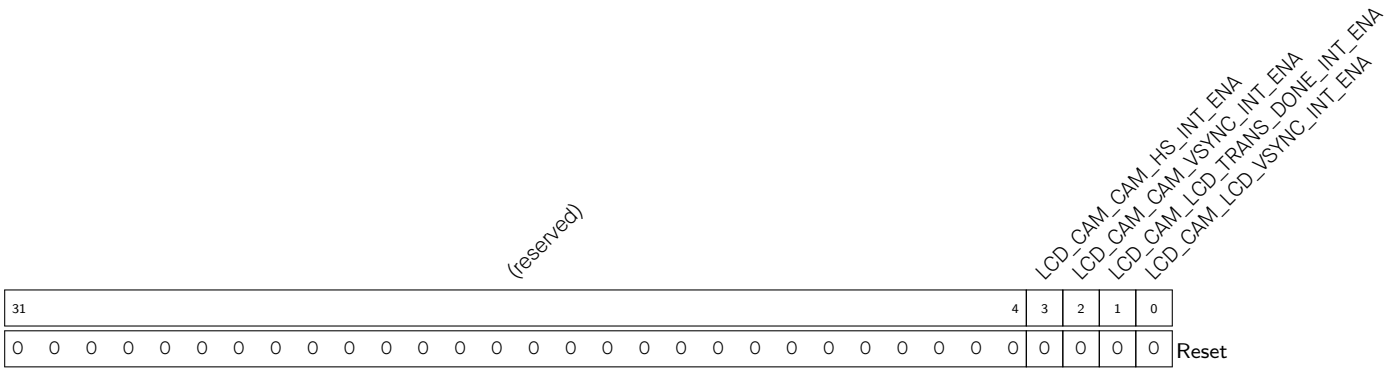
LCD_CAM_CAM_CONV_DATA_IN_MODE Configures the color range for the YUV-RGB converter's input data.
0: Limited color range.
1: Full color range.
(R/W)

LCD_CAM_CAM_CONV_MODE_8BITS_ON Configures the bit width of the YUV-RGB converter's input data.
0: 16-bit mode.
1: 8-bit mode.
(R/W)

LCD_CAM_CAM_CONV_TRANS_MODE Configures the data conversion format.
0: Data is converted to RGB format.
1: Data is converted to YUV format.
(R/W)

LCD_CAM_CAM_CONV_ENABLE Configures whether to enable YUV-RGB converter.
0: Bypass the format converter
1: Enable the format converter
(R/W)

Register 20.17. LCD_CAM_LC_DMA_INT_ENA_REG (0x0064)



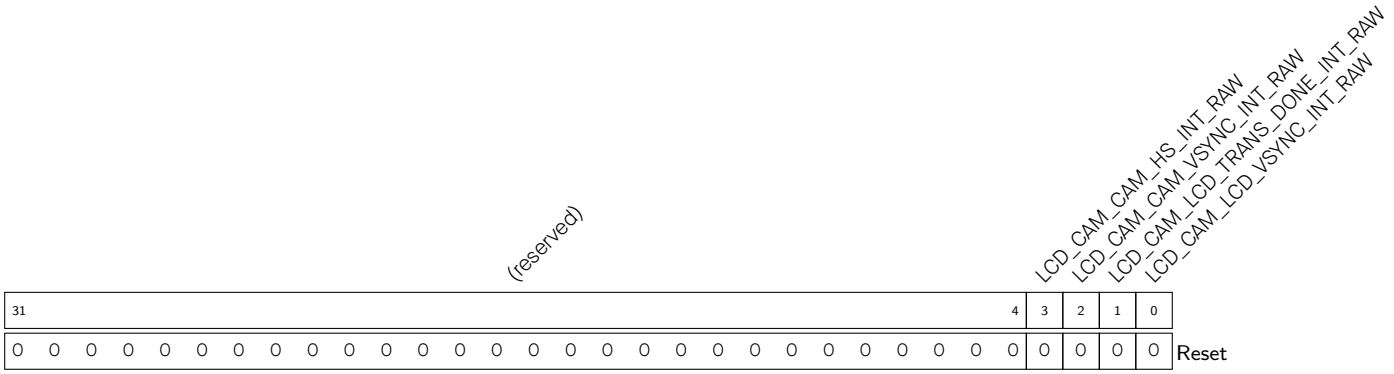
LCD_CAM_LCD_VSYNC_INT_ENA Write 1 to enable LCD_CAM_LCD_VSYNC_INT. (R/W)

LCD_CAM_LCD_TRANS_DONE_INT_ENA Write 1 to enable LCD_CAM_LCD_TRANS_DONE_INT. (R/W)

LCD_CAM_CAM_VSYNC_INT_ENA Write 1 to enable LCD_CAM_CAM_VSYNC_INT. (R/W)

LCD_CAM_CAM_HS_INT_ENA Write 1 to enable LCD_CAM_CAM_HS_INT. (R/W)

Register 20.18. LCD_CAM_LC_DMA_INT_RAW_REG (0x0068)



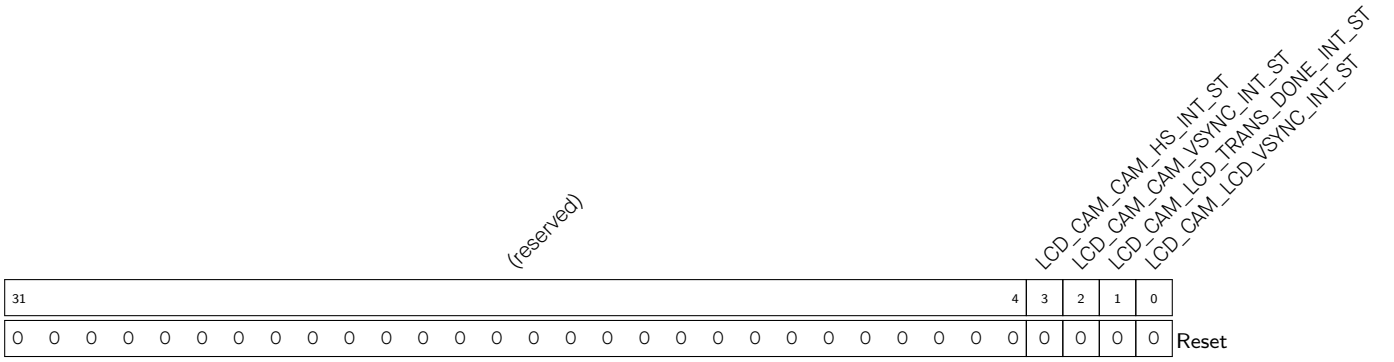
LCD_CAM_LCD_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_RAW The raw interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_RAW The raw interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 20.19. LCD_CAM_LC_DMA_INT_ST_REG (0x006C)



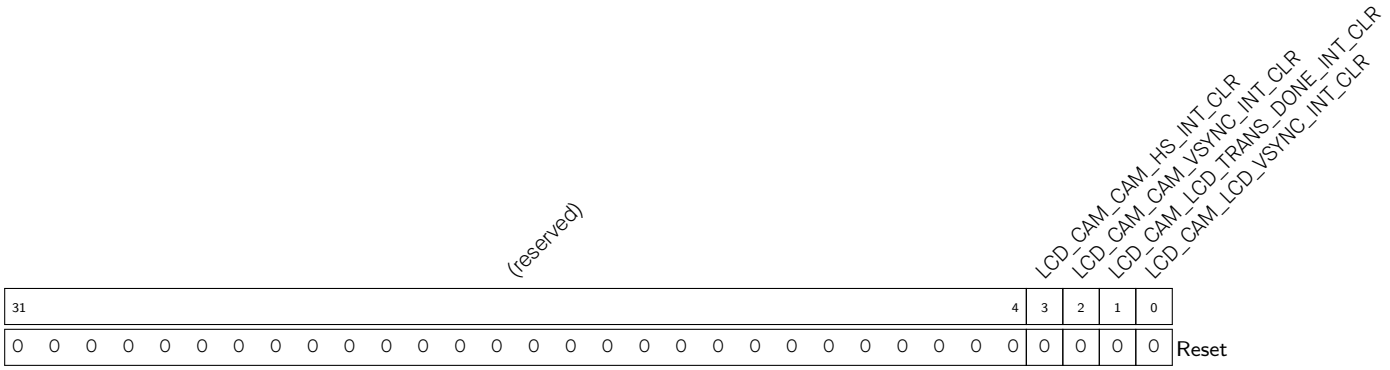
LCD_CAM_LCD_VSYNC_INT_ST The masked interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_ST The masked interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_ST The masked interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_ST The masked interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 20.20. LCD_CAM_LC_DMA_INT_CLR_REG (0x0070)



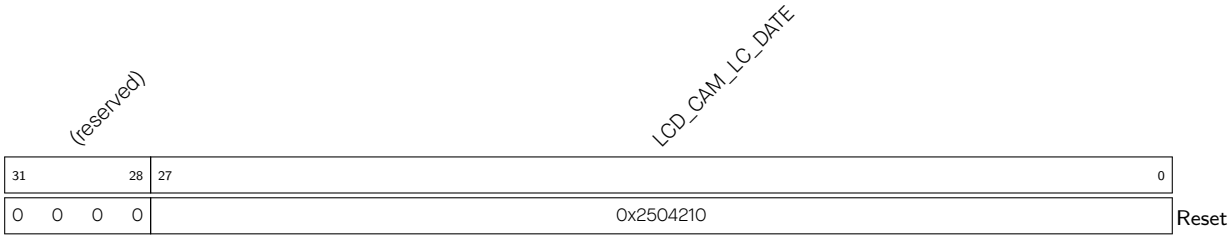
LCD_CAM_LCD_VSYNC_INT_CLR Write 1 to clear LCD_CAM_LCD_VSYNC_INT. (WO)

LCD_CAM_LCD_TRANS_DONE_INT_CLR Write 1 to clear LCD_CAM_LCD_TRANS_DONE_INT. (WO)

LCD_CAM_CAM_VSYNC_INT_CLR Write 1 to clear LCD_CAM_CAM_VSYNC_INT. (WO)

LCD_CAM_CAM_HS_INT_CLR Write 1 to clear LCD_CAM_CAM_HS_INT. (WO)

Register 20.21. LCD_CAM_LC_DATE_REG (0x00FC)



LCD_CAM_LC_DATE Version control register. (R/W)

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 21

UART Controller (UART)

21.1 Overview

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to the data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit(s) and parity bit(s).

A UART data frame usually begins with one start bit, followed by data bits, one parity bit (optional), and one or more stop bits. UART controllers on ESP32-S31 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as GDMA for high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

ESP32-S31 has five UART controllers, including four regular UARTs and one low-power (LP) UART. These UARTs are compatible with various UART devices, and support Infrared Data Association (IrDA) and RS485 communication. In this chapter, the four regular UART controllers are referred to as UART*n*, in which *n* denotes 0, 1, 2, 3. LP UART is the cut-down version of the regular UART, with a separate group of registers. For differences between UART and LP UART, please refer to Table 21.2-1.

21.2 Features

Table 21.2-1 lists the feature comparison between UART and LP UART:

Table 21.2-1. UART and LP UART Feature Comparison

UART Feature	LP UART Feature
Programmable baud rate up to 5 MBaud	
128 x 8 bit RAM respectively for the TX channel and RX channel of a UART controller	20 x 8-bit RAM, shared by the TX FIFO and RX FIFO of LP UART
Full-duplex asynchronous communication	
Data bits (5 to 8 bits)	
Stop bits (1, 1.5, or 2 bits)	
Parity bit	
Special character AT_CMD detection	
RS485 protocol	—
IrDA protocol	—

Cont'd on next page

Table 21.2-1 – cont'd from previous page

UART Feature	LP_UART Feature
High-speed data communication using GDMA	—
Receive timeout	
UART as wakeup source	
Software and hardware flow control	
Three prescalable clock sources: 1. XTAL_CLK 2. RC_FAST_CLK 3. PLL_F80M_CLK	Three prescalable clock sources 1. RC_FAST_CLK 2. XTAL_DIV_CLK 3. PLL_F8M_CLK

The following description mainly covers regular UART controllers.

21.3 UART Structure

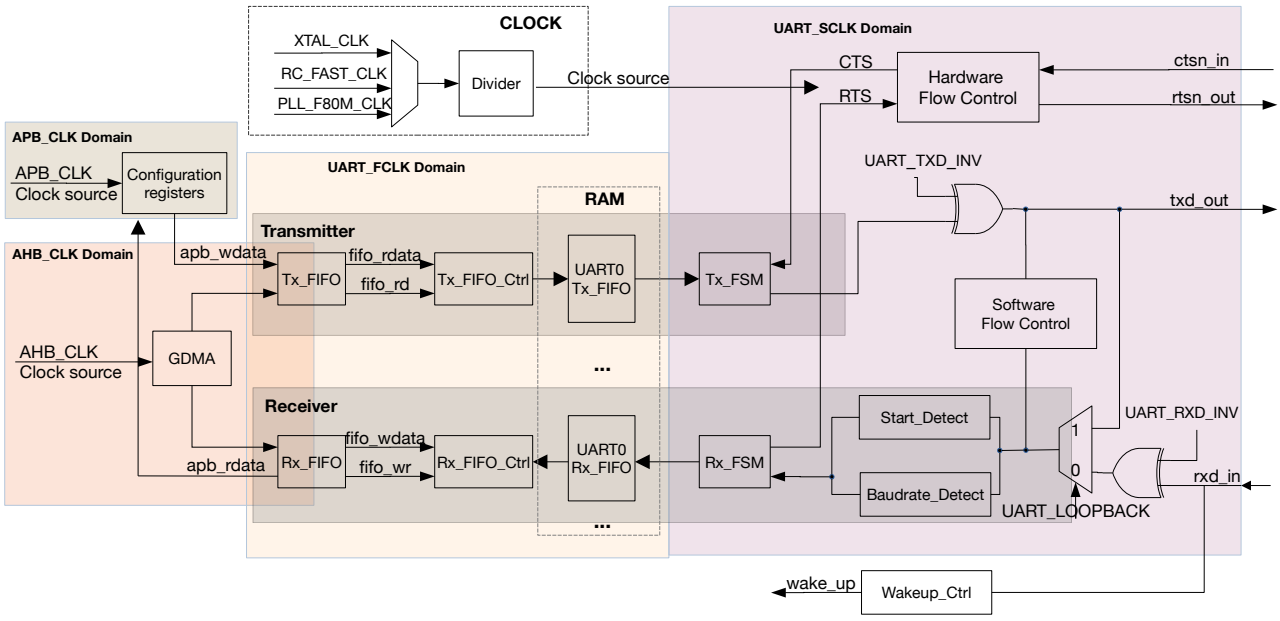


Figure 21.3-1. UART Structure

Figure 21.3-1 shows the basic structure of a UART controller. A UART controller works in four clock domains, namely APB_CLK, AHB_CLK, UART_SCLK, and UART_FCLK. APB_CLK and AHB_CLK are synchronized but with different frequencies (APB_CLK is derived from AHB_CLK by division), and likewise UART_SCLK and UART_FCLK are synchronized but with different frequencies (UART_SCLK is derived from UART_FCLK by division). UART_FCLK has three clock sources: PLL_F80M_CLK, RC_FAST_CLK, and crystal clock XTAL_CLK (for details, please refer to Chapter *7 Reset and Clock [to be added later]*), 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. Only regular UART has such a divider; LP UART does not.

A UART controller can be broken down into two parts based on functions: a transmitter and a receiver.

The transmitter contains a TX FIFO (i.e., Tx_FIFO in Figure 21.3-1), 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 21.3-1), 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 [to be added later]* for more details), a wake_up signal can be generated in four ways and sent to PMU, which then wakes up the ESP32-S31 chip. For more information about wakeup, please refer to Section 21.4.8.

21.4 Functional Description

21.4.1 Clock and Reset

UART controllers are asynchronous. Their register configuration module works in the APB_CLK domain. TX FIFO and RX FIFO work across the AHB_CLK and UART_FCLK domains. The UART RAM control unit works in the UART_FCLK domain. The UART transmission and reception control module works in the UART_SCLK domain, i.e., UART Core's clock domain.

When the frequency of the UART_SCLK is higher than the frequency needed to generate the baud rate, the UART Core can be clocked at a lower frequency by the divider, in order to reduce power consumption. Usually, the UART Core's clock frequency is lower than the APB_CLK's frequency, and can be divided by the largest divisor when higher than the frequency needed to generate the baud rate. The frequency of the UART Core's clock can also be at most twice higher than the APB_CLK. The clock for the UART transmitter and the UART receiver can be controlled independently. To enable the clock for the UART transmitter, UART_TX_SCLK_EN shall be set; to enable the clock for the UART receiver, UART_RX_SCLK_EN shall be set.

To ensure that the configured register values are synchronized from APB_CLK domain to the UART_SCLK domain, please follow the procedures in Section21.6.

To reset the whole UART, please:

- Enable the UART Core's clock by setting PCR_UART*n*_CLK_EN to 1.
- Write 1 to PCR_UART*n*_RST_EN.
- Clear PCR_UART*n*_RST_EN to 0.

21.4.2 UART FIFO

The transmitter and the receiver on the UART controller each use a 128 x 8-bit RAM, and access their respective RAM through a separate 4 x 8-bit asynchronous FIFO interface. The RAM and asynchronous FIFO interface for the transmitter and the receiver are independent and cannot be shared.

UART Tx_FIFO is reset by setting UART_TXFIFO_RST. UART Rx_FIFO is reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using GDMA, read automatically, and converted from a frame into a bitstream by hardware Tx_FSM. Data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using GDMA. The 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.

21.4.3 Baud Rate Generation and Detection

21.4.3.1 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 21.4-1, 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 21.4-1 below, to make the output timing more uniform:

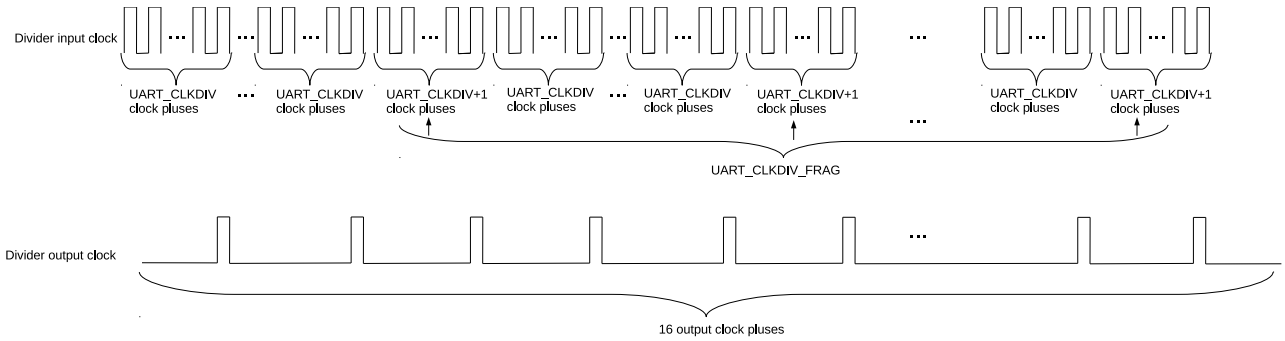


Figure 21.4-1. UART Controllers Division

To support IrDA (see Section 21.4.7 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.

21.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 21.4-2 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.

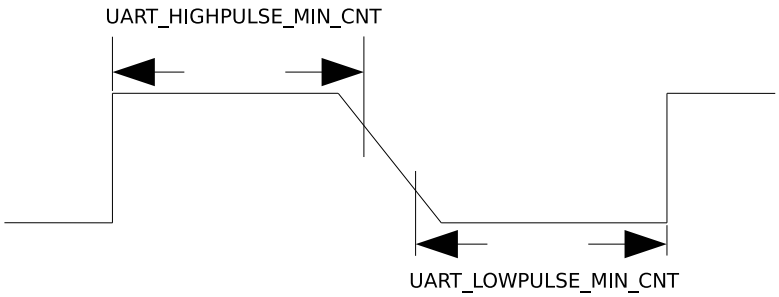


Figure 21.4-2. The Timing Diagram of Weak UART Signals Along Negative Edges

The baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along positive or negative edges in a metastable state, which results in the inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a

weighted average of these two values to eliminate errors for 1-bit pulses. In this case, the baud rate is calculated as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

2. If UART signals are weak along negative edges as shown in Figure 21.4-2, 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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

21.4.4 UART Data Frame

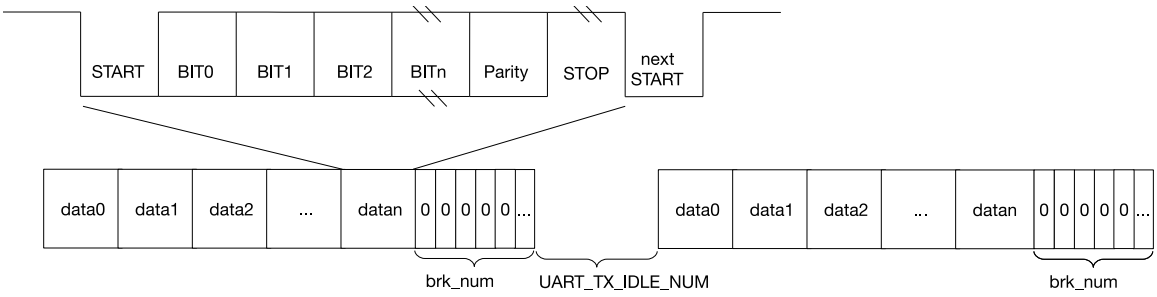


Figure 21.4-3. Structure of UART Data Frame

Figure 21.4-3 shows the basic structure of a data frame. A frame starts with one start bit, and ends with stop bits which can be 1, 1.5, or 2 bit-long, configured by UART_STOP_BIT_NUM (in RS485 mode turnaround delay may be added. See details in Section 21.4.6.2). The start bit is logical low, whereas stop bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in the data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received will still be stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set, then the transmitter will enter the break condition and send several NULL characters. After the NULL characters are sent, the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK_IDLE_DONE_INT interrupt is generated.

The receiver can also detect the Break conditions when the RX data line detects any low logical level for one NULL character transmission, and a UART_BRK_DET_INT interrupt will be triggered to detect when a break condition has been completed.

The receiver can detect the current bus state through the timeout interrupt UART_RXFIFO_TOUT_INT. The UART_RXFIFO_TOUT_INT interrupt will be triggered when the bus is in the idle state for more than UART_RX_TOUT_THRHD bit time on current baud rate after the receiver has received at least one byte. You can use this interrupt to detect whether all the data from the transmitter has been sent.

21.4.5 AT_CMD Character Structure

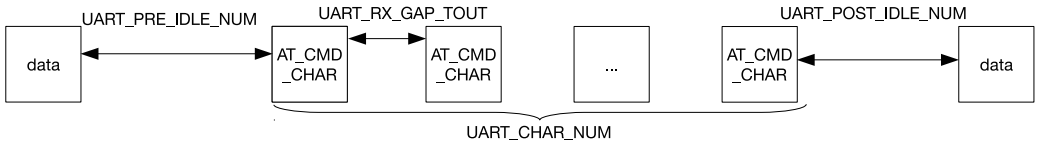


Figure 21.4-4. AT_CMD Character Structure

Figure 21.4-4 is the structure of a special character AT_CMD. If the receiver constantly receives AT_CMD_CHAR and the following conditions are met, a UART_AT_CMD_CHAR_DET_INT interrupt is generated.

- The interval between the first AT_CMD_CHAR and the last non-AT_CMD_CHAR character is at least UART_PRE_IDLE_NUM cycles.
- The interval between two AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT in the unit of baud rate cycles.
- The number of AT_CMD_CHAR characters is equal to or greater than UART_CHAR_NUM.
- The interval between the last AT_CMD_CHAR character and next non-AT_CMD_CHAR character is at least UART_POST_IDLE_NUM cycles.

Note: Given that the interval between AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT in the unit of baud rate cycles, the APB_CLK frequency is suggested not to be lower than 8 MHz.

21.4.6 RS485

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.

21.4.6.1 Driver Control

As shown in Figure 21.4-5, 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 21.4-5, DE is connected to dtrn_out of UART (please refer to Section 21.4.9.1 for more details).

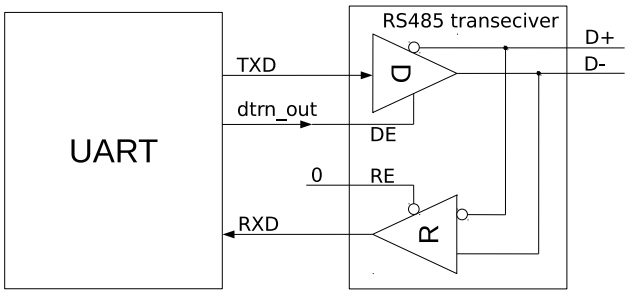


Figure 21.4-5. Driver Control Diagram in RS485 Mode

21.4.6.2 Turnaround Delay

By default, the UART controllers work in receiver mode. When a UART controller is switched from transmitter mode to receiver mode, the RS485 protocol requires a turnaround delay of one cycle after the stop bit. The UART transmitter supports adding a turnaround delay of one cycle before the start bit or after the stop bit. When UART_DL0_EN is set, a turnaround delay of one cycle is added before the start bit; when UART_DL1_EN is set, a turnaround delay of one cycle is added after the stop bit.

21.4.6.3 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 21.4-5, 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.

UART controllers can snoop the data sent by themselves. In transmitter mode, when a UART controller monitors a collision between the data sent and the data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitors a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a parity bit error, a UART_RS485_PARITY_ERR_INT is generated.

21.4.7 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol, and the link management protocol. The UART controllers implement IrDA's physical layer. In IrDA encoding, a UART controller supports data rates up to 115.2 kbit/s (SIR, or serial infrared mode). As shown in Figure 21.4-6, the IrDA encoder converts a non-return to zero code (NRZ) signal to a return to zero inverted code (RZI) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits to indicate logic "0", and low levels to indicate logic "1". The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the parity bit of the encoder output is opposite to that of the decoder input. If a low pulse is detected, it indicates that a start bit has been received.

When IrDA function is enabled, one bit is divided into 16 clock cycles. If the bit to be sent is zero, then the 9th, 10th, and 11th clock cycle are high.

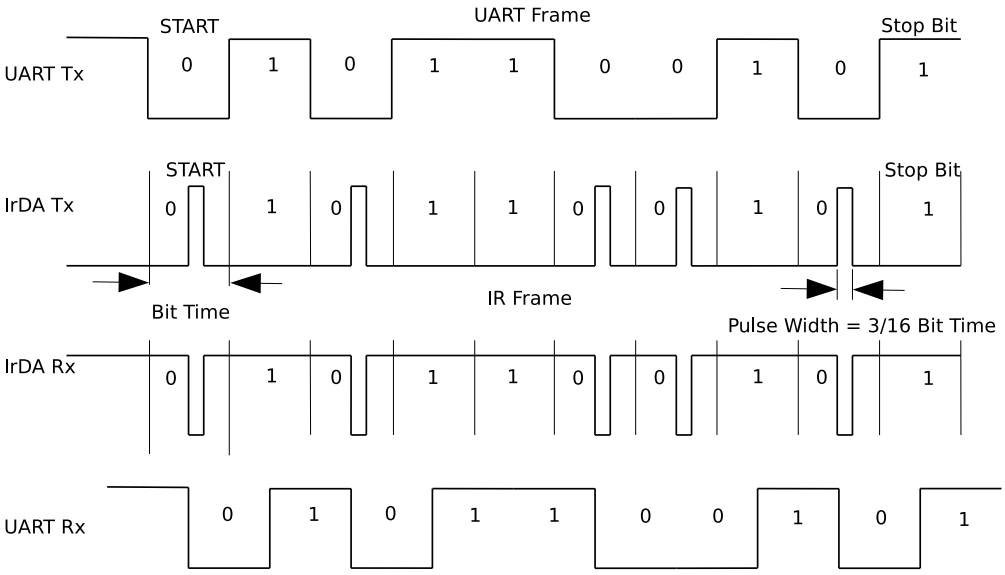


Figure 21.4-6. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 21.4-7, 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.

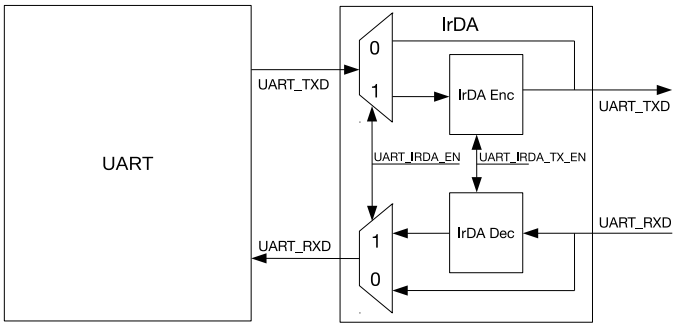


Figure 21.4-7. IrDA Encoding and Decoding Diagram

21.4.8 Wakeup

UART can be set as wakeup source. When a UART controller is in Light-sleep mode, a wake_up signal can be generated in four ways and be sent to the PMU module, which then wakes up ESP32-S31.

- UART_WK_MODE_SEL = 0: When all the clocks are disabled, the chip can be woken up by reverting RXD for multiple cycles until the number of positive edges is greater than or equal to UART_ACTIVE_THRESHOLD + 3.
- UART_WK_MODE_SEL = 1: UART Core keeps working, so the UART receiver can still receive data and store the received data in RX FIFO. When the number of data bytes in RX FIFO is greater than UART_RX_WAKE_UP_THRHD, the chip can be woken up from the Light-sleep mode.

- UART_WK_MODE_SEL = 2: When the UART receiver detects a start bit, the chip will be woken up.
- UART_WK_MODE_SEL = 3: When the UART receiver receives a specific character sequence, the chip will be woken up. The wakeup characters can be defined by configuring UART_WK_CHAR0, UART_WK_CHAR1, UART_WK_CHAR2, UART_WK_CHAR3, and UART_WK_CHAR4. These characters can be formed into different character sequences by configuring UART_CHAR_NUM and UART_WK_CHAR_MASK, as shown in Table 21.4-1. Once the sequence is detected, the chip will be woken up. For the last configuration in Table 21.4-1, UART will detect for CHAR0 ~ CHAR4 in order.

Table 21.4-1. 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.

21.4.9 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.

21.4.9.1 Hardware Flow Control

Figure 21.4-8 shows the hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal ctsn_in. Figure 21.4-9 illustrates how these signals are connected between UART on ESP32-S31 (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

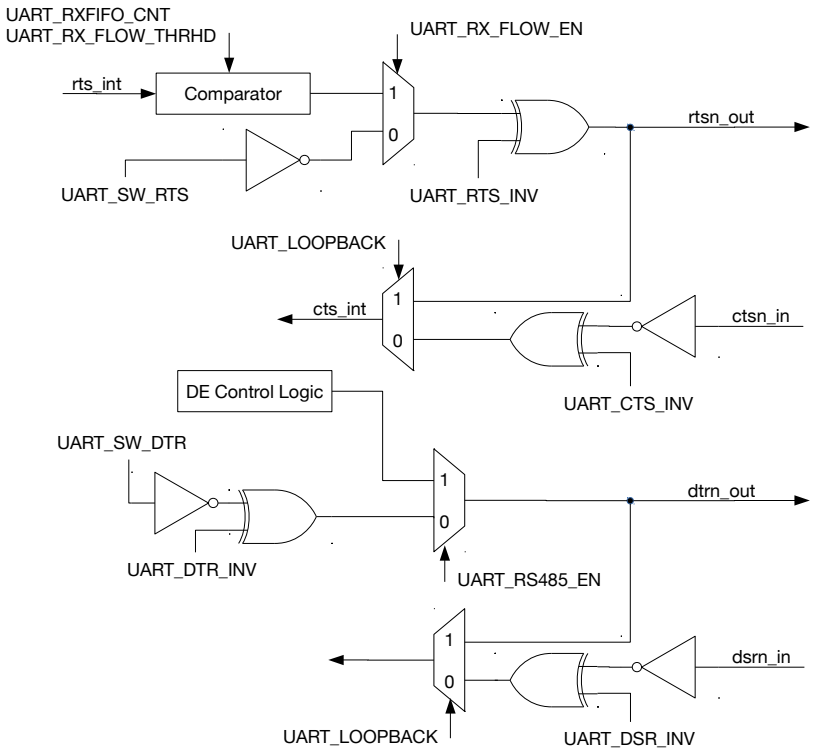


Figure 21.4-8. Hardware Flow Control Diagram

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.

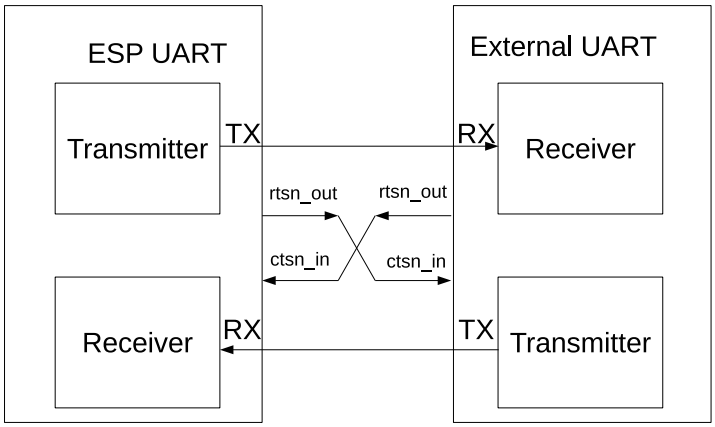


Figure 21.4-9. Connection between Hardware Flow Control Signals

When ctsn_in of IU0 is low, IU0 is allowed to send data; when ctsn_in is high, IU0 is not allowed to send data. When IU0 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.

21.4.9.2 Software Flow Control

Instead of CTS/RTS lines, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting UART_SW_FLOW_CON_EN to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in the data flow received, and generate a UART_SW_XOFF_INT or a UART_SW_XON_INT interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting UART_FORCE_XOFF, or to start sending data by setting UART_FORCE_XON.

Software determines whether to insert flow control characters based on the remaining room in RX FIFO. When UART_SEND_XOFF is set, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission; when UART_SEND_XON is set, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores more data than UART_XOFF_THRESHOLD, UART_SEND_XOFF is set by hardware. As a result, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores less data than UART_XON_THRESHOLD, UART_SEND_XON is set by hardware. As a result, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission.

In full-duplex mode, when the UART receiver receives an XOFF character, the UART transmitter is not allowed to send any data including XOFF even if the UART receiver receives more data than its threshold. To avoid deadlocks in software flow control or overflow caused thereby, you can set UART_XON_XOFF_STILL_SEND. In this way, the UART transmitter can still send an XOFF character when it is not allowed to send any data.

21.4.10 GDMA Mode

The UART controllers on ESP32-S31 share one TX/RX GDMA (General Direct Memory Access) channel via UHCI (Universal Host Controller Interface). In GDMA mode, UART controllers support the decoding and encoding of HCI data packets. The UHCI_UART_SEL field determines which UART controller occupies the GDMA TX/RX channel.

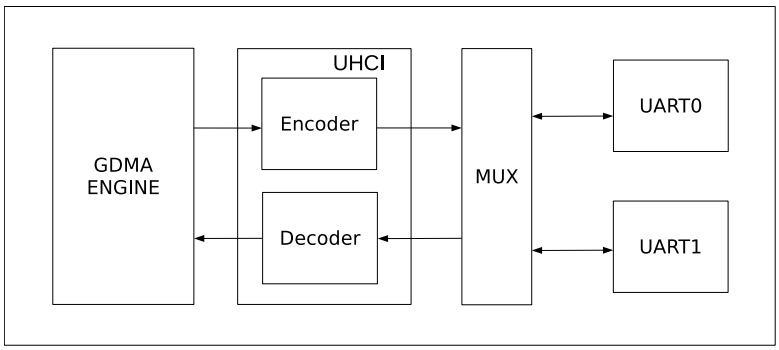


Figure 21.4-10. Data Transfer in GDMA Mode

Figure 21.4-10 shows how data is transferred using GDMA. Before GDMA receives data, software prepares an inlink. AHB_DMA_INLINK_ADDR_CH*n* points to the first receive descriptor in the inlink. After AHB_DMA_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. AHB_DMA_OUTLINK_ADDR_CH*n* points to the first transmit descriptor in the outlink. After AHB_DMA_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.

21.5 Interrupts

ESP32-S31's UART*n* and UHCI can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- UART*n*_INTR
- UHCI_INTR

There are several internal interrupt sources from UART*n* and UHCI that can generate the above interrupt signals.

UART*n* interrupt sources are listed as follows:

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.

- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.
- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver has received at least one byte, and the bus remains idle for UART_RX_TOUT_THRHD bit time.
- UART_BRK_DET_INT: Triggered when the receiver detects a NULL character (i.e., logic 0 for one NULL character transmission) after stop bits.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of CTSn signals.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of DSRn signals.
- UART_RXFIFO_OVF_INT: Triggered when the amount of data received by the receiver exceeds the storage capacity of the FIFO.
- UART_FRM_ERR_INT: Triggered when the receiver detects a data frame error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

UHCI interrupt sources are listed as follows:

- UHCI_APP_CTRL1_INT: Triggered when software sets UHCI_APP_CTRL1_INT_RAW.
- UHCI_APP_CTRL0_INT: Triggered when software sets UHCI_APP_CTRL0_INT_RAW.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_SEND_A_REG_Q_INT: Triggered when UHCI has sent a series of short packets using always_send.
- UHCI_SEND_S_REG_Q_INT: Triggered when UHCI has sent a series of short packets using single_send.
- UHCI_TX_HUNG_INT: Triggered when UHCI takes too long to read RAM using a GDMA transmit channel.
- UHCI_RX_HUNG_INT: Triggered when UHCI takes too long to receive data using a GDMA receive channel.
- UHCI_TX_START_INT: Triggered when GDMA detects a separator character.

- UHCI_RX_START_INT: Triggered when a separator character has been sent.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 21.7 *Register Summary*.

21.6 Programming Procedures

21.6.1 Register Type

All UART registers are in the APB_CLK domain.

UART configuration registers can be classified into two groups. One group of registers are read in the APB_CLK or AHB_CLK domains, so once such registers are configured no extra operations are required. The other group of registers are read in the UART_SCLK domain, and therefore need to implement the clock domain crossing design. Once these registers are configured, the configured values need to be synchronized to the UART Core's clock domain by writing to UART_REG_UPDATE. Once all values have been synchronized, UART_REG_UPDATE will be automatically cleared by hardware. After configuring registers that need synchronization, it is recommended to check whether UART_REG_UPDATE is 0. This is to ensure that register values configured before have already been synchronized.

To distinguish between these two groups of registers easily, all registers that implement the clock domain crossing design have the _SYNC suffix, and are put together in Section 21.7. Those without the _SYNC suffix in Section 21.7 are configuration registers that require no clock domain crossing.

21.6.2 Detailed Steps

Figure 21.6-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

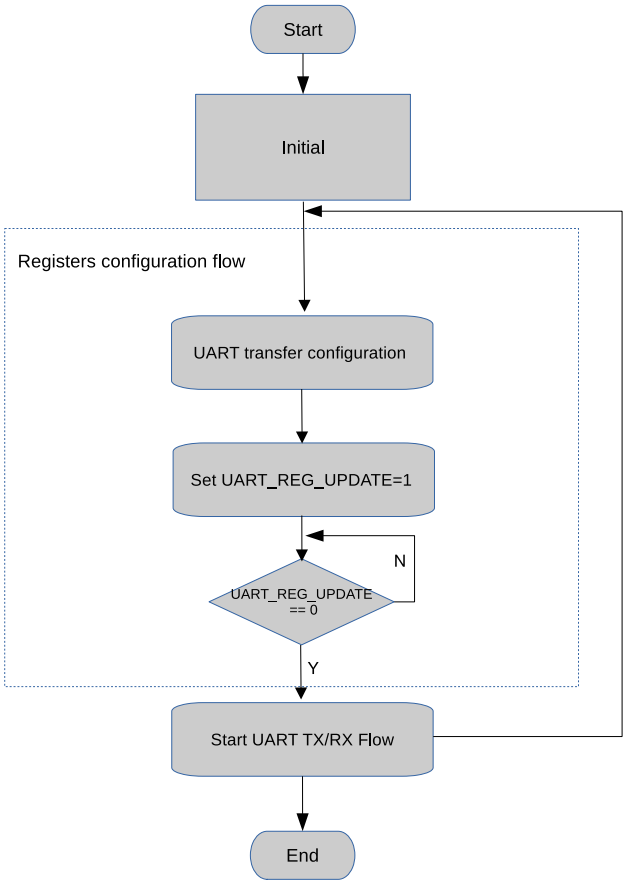


Figure 21.6-1. UART Programming Procedures

21.6.2.1 Initializing UART*n*

To initialize UART*n*:

- Enable APB_CLK for UART*n* by setting PCR_UART*n*_CLK_EN to 1.
- Write 1 to PCR_UART*n*_RST_EN.
- Clear PCR_UART*n*_RST_EN to 0.

21.6.2.2 Configuring UART*n* Communication

To configure UART*n* communication:

- Wait for UART_REG_UPDATE to become 0, which indicates the completion of the last synchronization.
- Select the clock source via PCR_UART*n*_SCLK_SEL.
- Configure divisor of the divider via PCR_UART*n*_SCLK_DIV_NUM, PCR_UART*n*_SCLK_DIV_A, and PCR_UART*n*_SCLK_DIV_B.
- Configure the baud rate for transmission via UART_CLKDIV and UART_CLKDIV_FRAG.

- Configure data length via UART_BIT_NUM.
- Configure odd or even parity check via UART_PARITY_EN and UART_PARITY.
- Optional steps depending on application ...
- Synchronize the configured values to the Core Clock domain by writing 1 to UART_REG_UPDATE.

21.6.2.3 Enabling UART*n*

To enable UART*n* transmitter:

- Configure TX FIFO's empty threshold via UART_TXFIFO_EMPTY_THRHD.
- Disable UART_TXFIFO_EMPTY_INT interrupt by clearing UART_TXFIFO_EMPTY_INT_ENA.
- Write data to be sent to UART_RXFIFO_RD_BYTE.
- Clear UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_CLR.
- Enable UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_ENA.
- Check UART_TXFIFO_EMPTY_INT_ST and wait for the completion of data transmission.

To enable UART*n* receiver:

- Configure RX FIFO's full threshold via UART_RXFIFO_FULL_THRHD.
- Enable UART_RXFIFO_FULL_INT interrupt by setting UART_RXFIFO_FULL_INT_ENA.
- Check UART_RXFIFO_FULL_INT_ST and wait until the RX FIFO is full.
- Read data from RX FIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RX FIFO via UART_RXFIFO_CNT.

21.7 Register Summary

21.7.1 UART Register Summary

The addresses in this section are relative to UART Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
FIFO Configuration Register			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART threshold and allocation configuration	0x0064	R/W
UART Interrupt Register			
UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
UART_SLEEP_CONF0_REG	UART sleep configuration register 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART sleep configuration register 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART sleep configuration register 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 mode configuration	0x004C	R/W
UART_CLK_CONF_REG	UART core clock configuration	0x0088	R/W
UART_REG_UPDATE_REG	UART register configuration update	0x0098	R/W/SC
UART_ID_REG	UART ID register	0x009C	R/W
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
UART_MEM_RX_STATUS_REG	Rx FIFO write and read offset address	0x006C	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0070	RO
UART_AFIFO_STATUS_REG	UART asynchronous FIFO status	0x0090	RO
AT Escape Sequence Selection Configuration Register			
UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W

Name	Description	Address	Access
UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Autobaud Register			
UART_POSPULSE_REG	Autobaud high pulse register	0x0074	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0078	RO
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x007C	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x0080	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0084	RO
Version Register			
UART_DATE_REG	UART version control register	0x008C	R/W

21.7.2 LP UART Register Summary

The addresses in this section are relative to LP UART base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
FIFO Configuration Register			
LP_UART_FIFO_REG	FIFO data register	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART threshold and allocation configuration	0x0064	R/W
LP UART Interrupt Register			
LP_UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	Masked interrupt status	0x0008	RO
LP_UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
LP_UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
LP_UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
LP_UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
LP_UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
LP_UART_CONF1_REG	Configuration register 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
LP_UART_SLEEP_CONF0_REG	LP UART sleep configuration register 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART sleep configuration register 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART sleep configuration register 2	0x0038	R/W
LP_UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
LP_UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
LP_UART_DELAY_CONF_SYNC_REG	RS485 mode configuration	0x004C	R/W

Name	Description	Address	Access
LP_UART_CLK_CONF_REG	LP UART core clock configuration	0x0088	R/W
LP_UART_REG_UPDATE_REG	LP UART register configuration update register	0x0098	R/W/SC
LP_UART_ID_REG	LP UART ID register	0x009C	R/W
Status Register			
LP_UART_STATUS_REG	LP UART status register	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART transmit and receive status	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART asynchronous FIFO Status	0x0090	RO
AT Escape Sequence Selection Configuration Register			
LP_UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
LP_UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Version Register			
LP_UART_DATE_REG	LP UART version register	0x008C	R/W

21.7.3 UHCI Register Summary

The addresses in this section are relative to UHCI base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x0014	varies
UHCI_ESCAPE_CONF_REG	Escape character configuration	0x0020	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0024	R/W
UHCI_ACK_NUM_REG	UHCI ACK number configuration	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 quick send register	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 quick send register	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 quick send register	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 quick send register	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 quick send register	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 quick send register	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 quick send register	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 quick send register	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 quick send register	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 quick send register	0x0058	R/W

Name	Description	Address	Access
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 register	0x0068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x006C	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x0070	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x0074	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x0078	R/W
UHCI_PKT_THRES_REG	Configuration register for packet length	0x007C	R/W
UHCI Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	varies
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WT
UHCI Status Register			
UHCI_STATE0_REG	UHCI receive status	0x0018	RO
UHCI_STATE1_REG	UHCI transmit status	0x001C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x002C	RO
Version Register			
UHCI_DATE_REG	UHCI version control register	0x0080	R/W

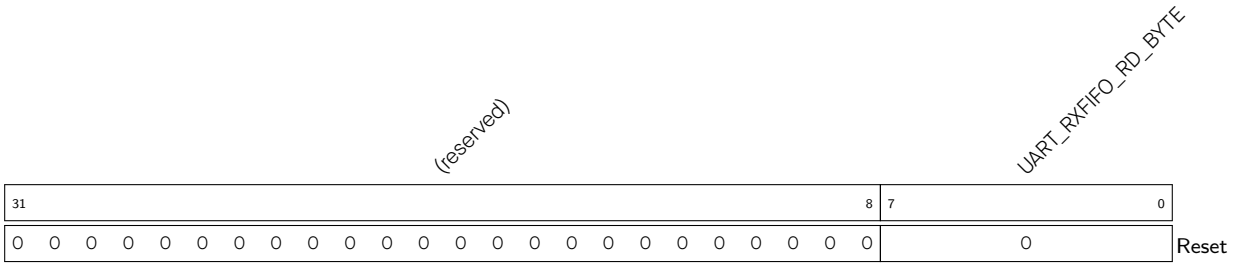
21.8 Registers

21.8.1 UART Registers

The addresses in this section are relative to UART Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

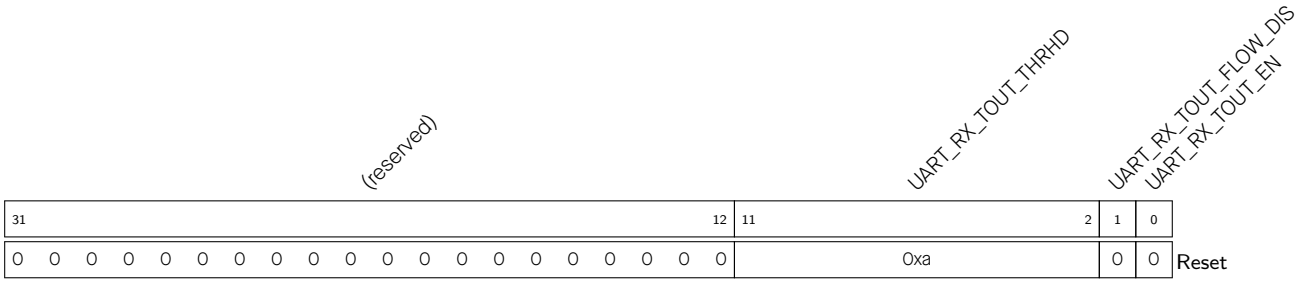
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 21.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE Represents the data UART*n* reads from FIFO.
Measurement unit: byte. (RO)

Register 21.2. UART_TOUT_CONF_SYNC_REG (0x0064)

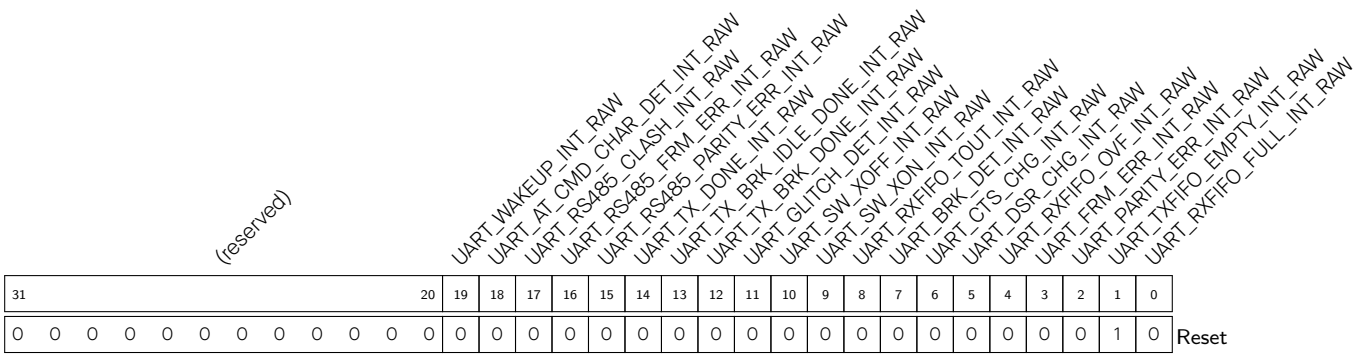


UART_RX_TOUT_EN Configures whether or not to enable UART receiver's timeout function.
0: Disable
1: Enable
(R/W)

UART_RX_TOUT_FLOW_DIS Configures whether or not to disable the idle status counter when hardware flow control is enabled.
0: Enable
1: Disable
(R/W)

UART_RX_TOUT_THRHD Configures the amount of time that the bus can remain idle before timeout.
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW The raw interrupt status of UART_RXFIFO_FULL_INT. (R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW The raw interrupt status of UART_TXFIFO_EMPTY_INT. (R/WTC/SS)

UART_PARITY_ERR_INT_RAW The raw interrupt status of UART_PARITY_ERR_INT. (R/WTC/SS)

UART_FRM_ERR_INT_RAW The raw interrupt status of UART_FRM_ERR_INT. (R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW The raw interrupt status of UART_RXFIFO_OVF_INT. (R/WTC/SS)

UART_DSR_CHG_INT_RAW The raw interrupt status of UART_DSR_CHG_INT. (R/WTC/SS)

UART_CTS_CHG_INT_RAW The raw interrupt status of UART_CTS_CHG_INT. (R/WTC/SS)

UART_BRK_DET_INT_RAW The raw interrupt status of UART_BRK_DET_INT. (R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW The raw interrupt status of UART_RXFIFO_TOUT_INT. (R/WTC/SS)

UART_SW_XON_INT_RAW The raw interrupt status of UART_SW_XON_INT. (R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT. (R/WTC/SS)

UART_GLITCH_DET_INT_RAW The raw interrupt status of UART_GLITCH_DET_INT. (R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW The raw interrupt status of UART_TX_BRK_DONE_INT. (R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW The raw interrupt status of UART_TX_BRK_IDLE_DONE_INT. (R/WTC/SS)

UART_TX_DONE_INT_RAW The raw interrupt status of UART_TX_DONE_INT. (R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW The raw interrupt status of UART_RS485_PARITY_ERR_INT. (R/WTC/SS)

UART_RS485_FRM_ERR_INT_RAW The raw interrupt status of UART_RS485_FRM_ERR_INT. (R/WTC/SS)

Continued on the next page...

Register 21.3. UART_INT_RAW_REG (0x0004)

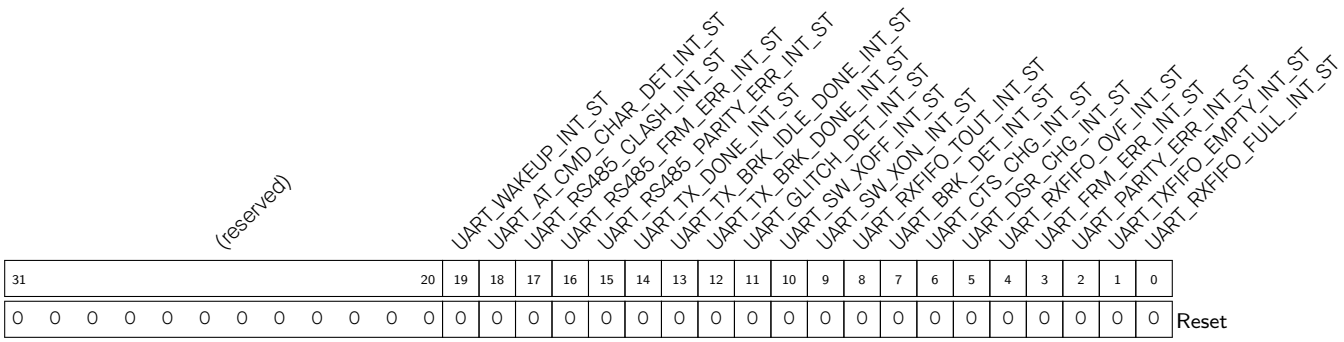
Continued from the previous page...

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 21.4. 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 21.5. 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)

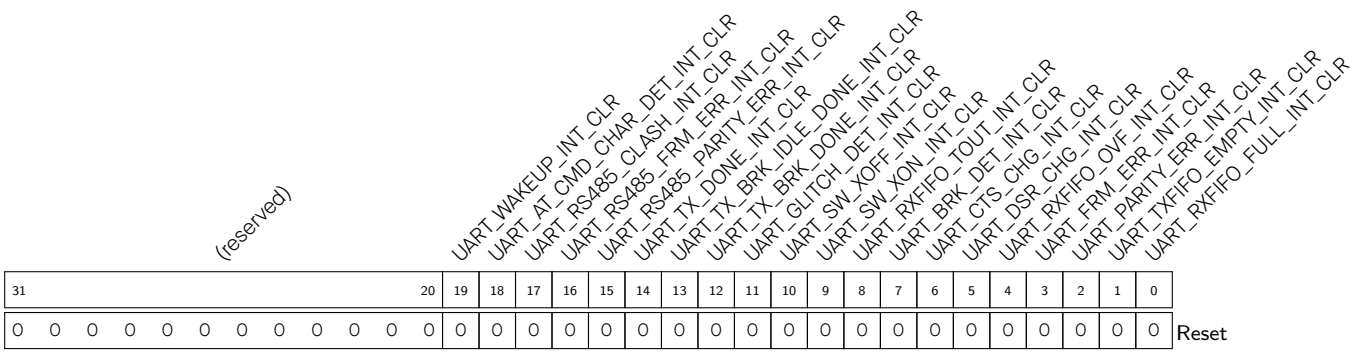
UART_RS485_FRM_ERR_INT_ENA Write 1 to enable UART_RS485_FRM_ERR_INT. (R/W)

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 21.6. 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)

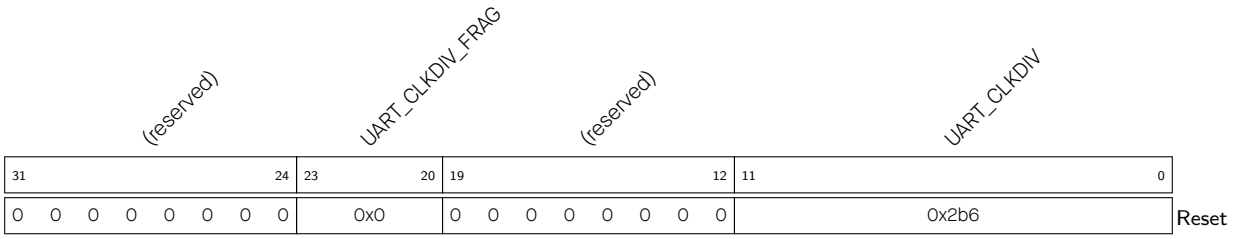
UART_RS485_FRM_ERR_INT_CLR Write 1 to clear UART_RS485_FRM_ERR_INT. (WT)

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)

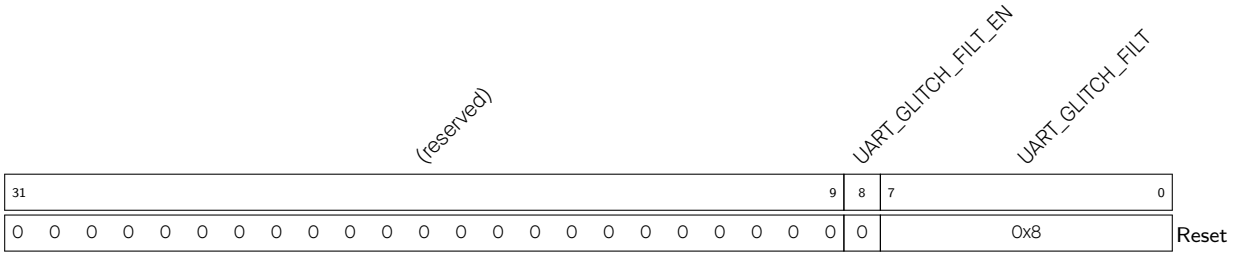
Register 21.7. UART_CLKDIV_SYNC_REG (0x0014)



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 21.8. 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)

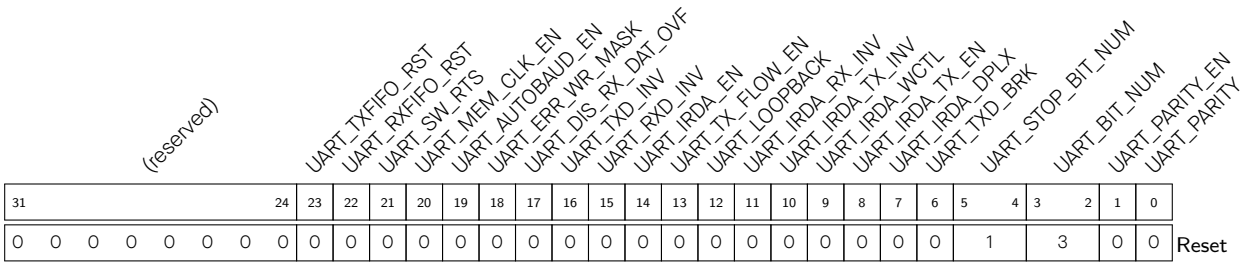
UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 21.9. 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 21.9. 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 21.9. 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)

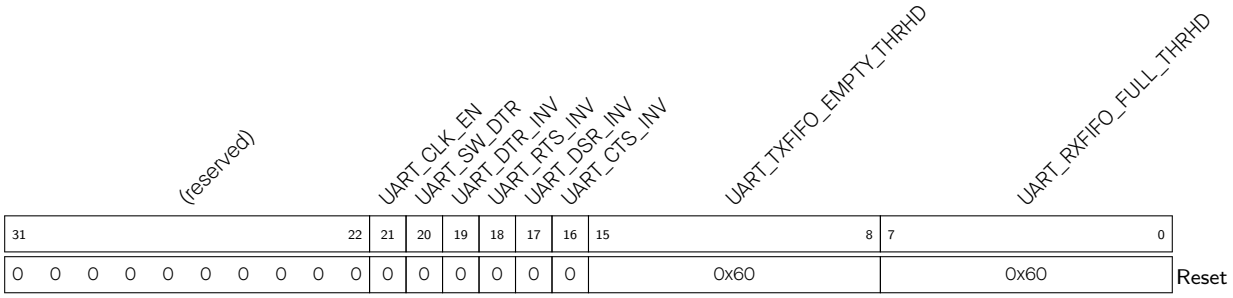
UART_TXFIFO_RST Configures whether or not to reset the UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 21.10. 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)

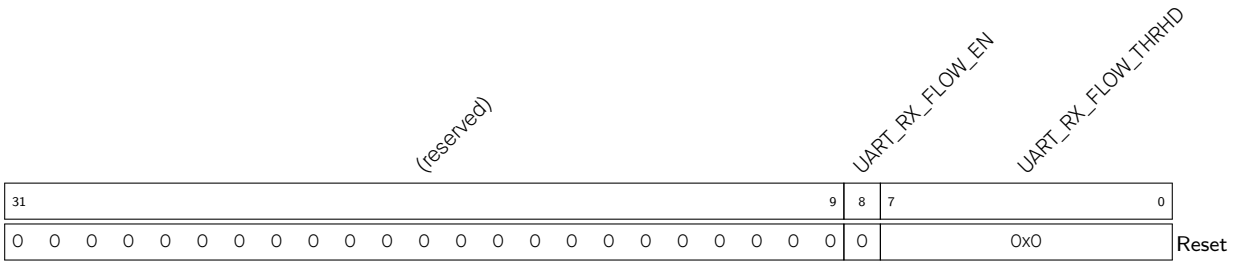
UART_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 21.11. 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)

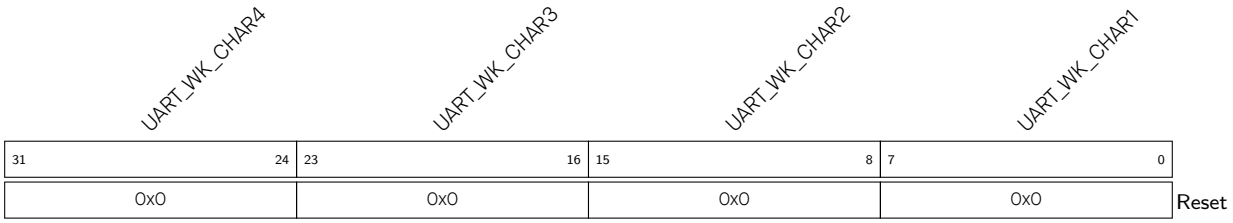
UART_RX_FLOW_EN Configures whether or not to enable the UART receiver.

0: Disable

1: Enable

(R/W)

Register 21.12. UART_SLEEP_CONF0_REG (0x0030)



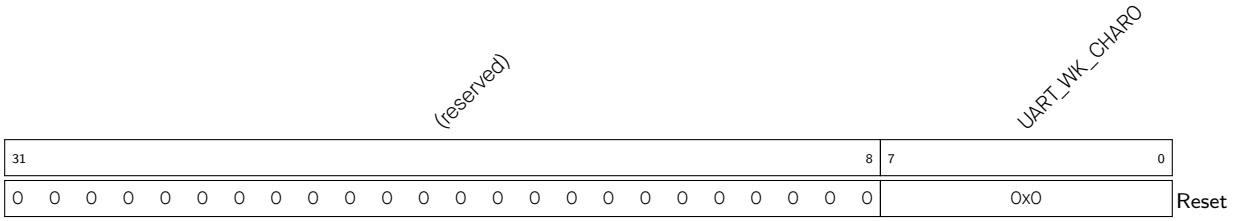
UART_WK_CHAR1 Configures wakeup character 1. (R/W)

UART_WK_CHAR2 Configures wakeup character 2. (R/W)

UART_WK_CHAR3 Configures wakeup character 3. (R/W)

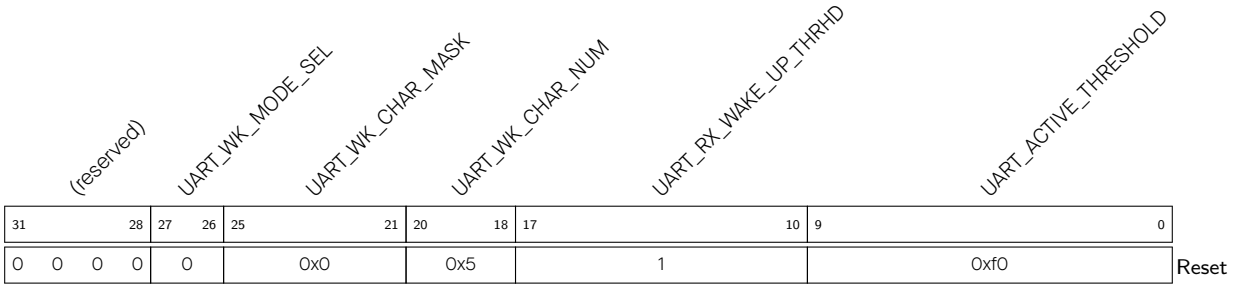
UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 21.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 21.14. UART_SLEEP_CONF2_REG (0x0038)



UART_ACTIVE_THRESHOLD Configures the number of RXD edge changes to wake up the chip in wakeup mode 0. (R/W)

UART_RX_WAKE_UP_THRHD Configures the number of received data bytes to wake up the chip in wakeup mode 1. (R/W)

UART_WK_CHAR_NUM Configures the number of wakeup characters. (R/W)

UART_WK_CHAR_MASK Configures whether or not to mask wakeup characters.

0: Not mask

1: Mask

(R/W)

UART_WK_MODE_SEL Configures which wakeup mode to select. See Section 21.4.8 *Wakeup* for the explanation of each mode.

0: Mode 0

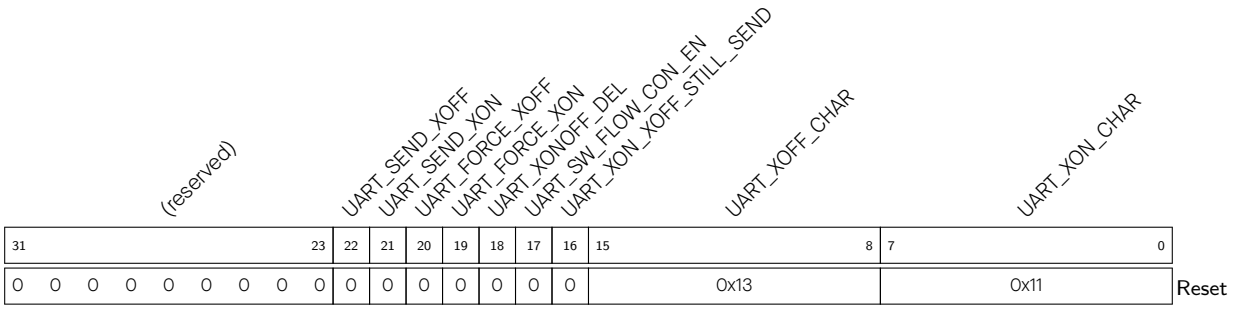
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 21.15. UART_SWFC_CONF0_SYNC_REG (0x003C)



UART_XON_CHAR Configures the XON character for flow control. (R/W)

UART_XOFF_CHAR Configures the XOFF character for flow control. (R/W)

UART_XON_XOFF_STILL_SEND Configures whether the UART transmitter can send XON or XOFF characters when it is disabled.

0: Cannot send

1: Can send

(R/W)

UART_SW_FLOW_CON_EN Configures whether or not to enable software flow control.

0: Disable

1: Enable

(R/W)

UART_XONOFF_DEL Configures whether or not to remove flow control characters from the received data.

0: Not move

1: Move

(R/W)

UART_FORCE_XON Configures whether the transmitter continues to sending data.

0: Not send

1: Send

(R/W)

UART_FORCE_XOFF Configures whether or not to stop the transmitter from sending data.

0: Not stop

1: Stop

(R/W)

UART_SEND_XON Configures whether or not to send XON characters.

0: Not send

1: Send

(R/W/SS/SC)

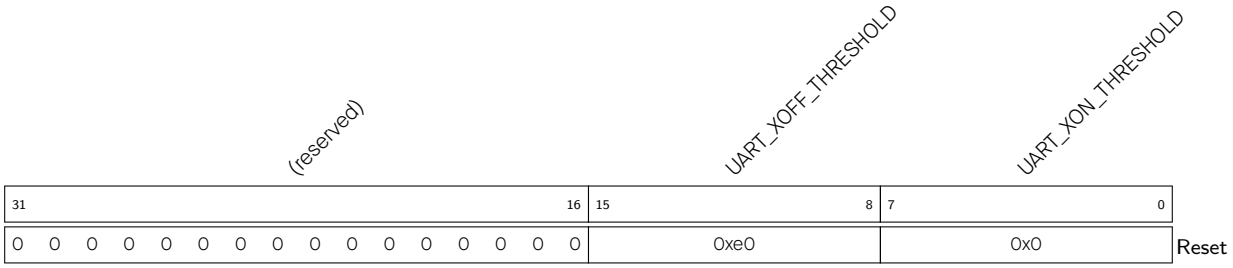
UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 21.16. UART_SWFC_CONF1_REG (0x0040)



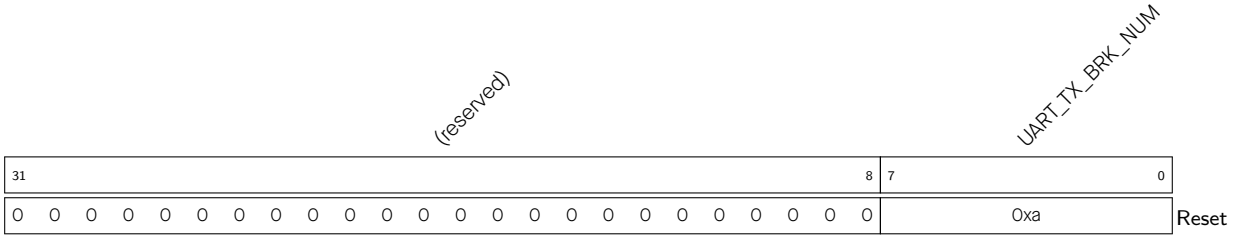
UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

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)

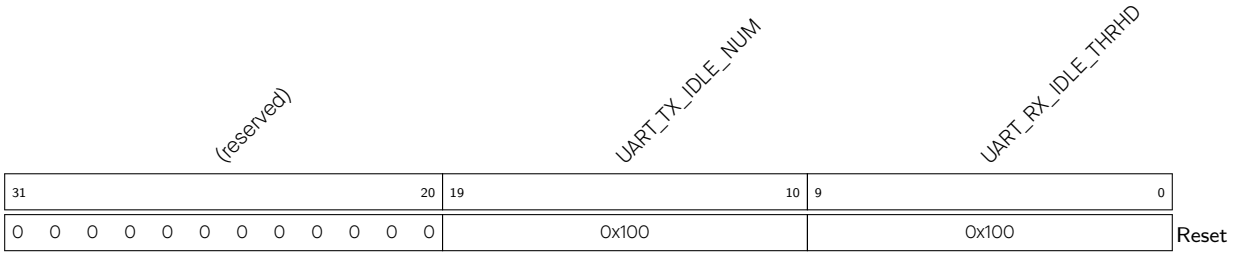
Register 21.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



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 21.18. UART_IDLE_CONF_SYNC_REG (0x0048)



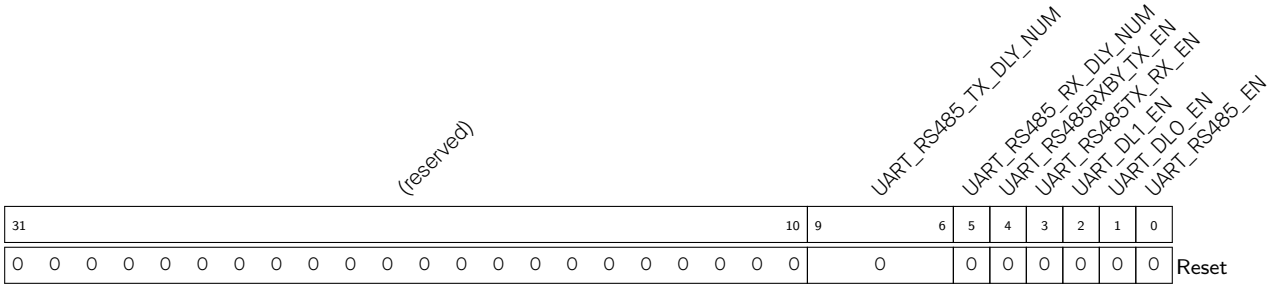
UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

UART_TX_IDLE_NUM Configures the interval between two data transfers.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.19. UART_RS485_CONF_SYNC_REG (0x004C)



UART_RS485_EN Configures whether or not to enable RS485 mode.

0: Disable

1: Enable

(R/W)

UART_DL0_EN Configures whether or not to add a turnaround delay of 1 bit before the start bit.

0: Not add

1: Add

(R/W)

UART_DL1_EN Configures whether or not to add a turnaround delay of 1 bit after the stop bit.

0: Not add

1: Add

(R/W)

UART_RS485TX_RX_EN Configures whether or not to enable the receiver for data reception when the transmitter is transmitting data in RS485 mode.

0: Disable

1: Enable

(R/W)

UART_RS485RXBY_TX_EN Configures whether to enable the RS485 transmitter for data transmission when the RS485 receiver is busy.

0: Disable

1: Enable

(R/W)

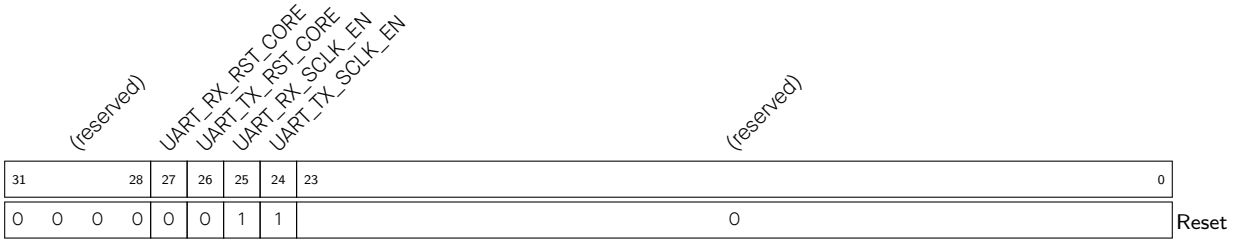
UART_RS485_RX_DLY_NUM Configures the delay of internal data signals in the receiver.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

UART_RS485_TX_DLY_NUM Configures the delay of internal data signals in the transmitter.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.20. 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

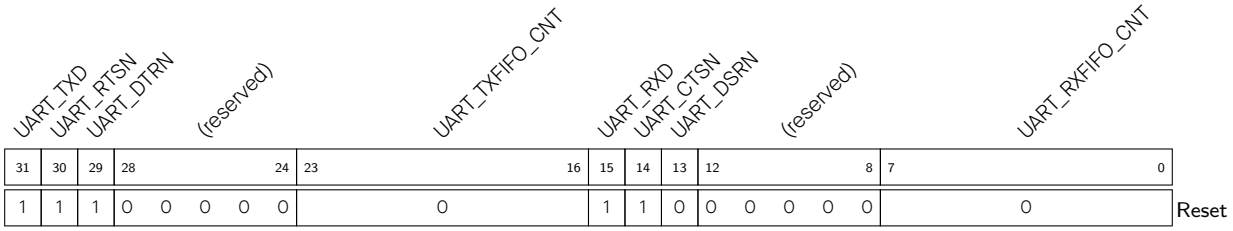
1: Enable

(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 21.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

UART_DSRN Represents the level of the internal UART DSR signal. (RO)

UART_CTSN Represents the level of the internal UART CTS signal. (RO)

UART_RXD Represents the level of the internal UART RXD signal. (RO)

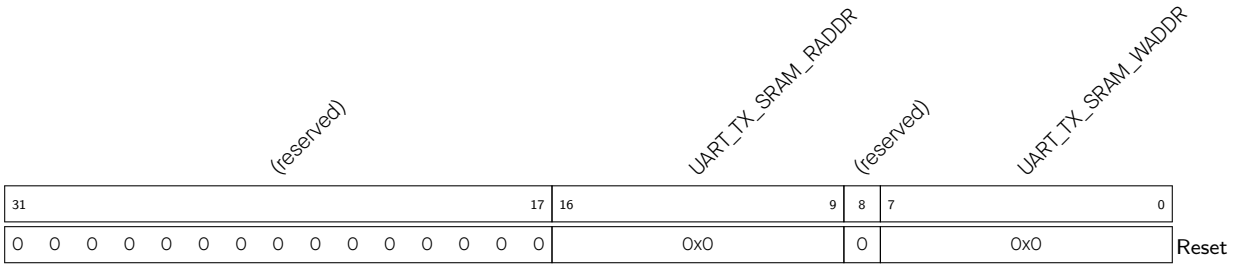
UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

UART_DTRN Represents the level of the internal UART DTR signal. (RO)

UART_RTSN Represents the level of the internal UART RTS signal. (RO)

UART_TXD Represents the level of the internal UART TXD signal. (RO)

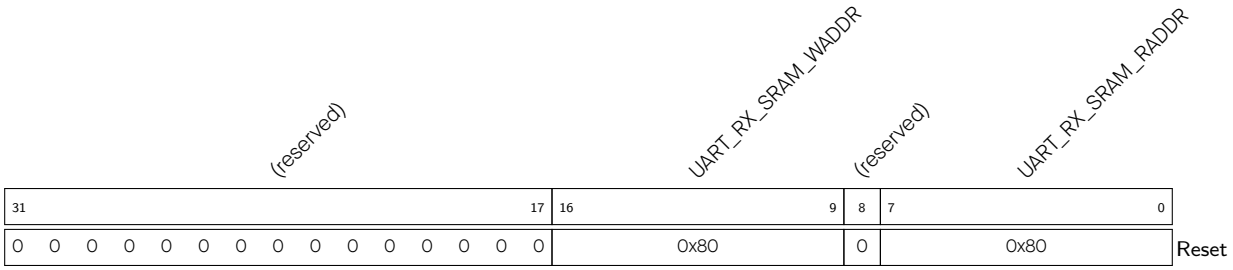
Register 21.22. UART_MEM_TX_STATUS_REG (0x0068)



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)

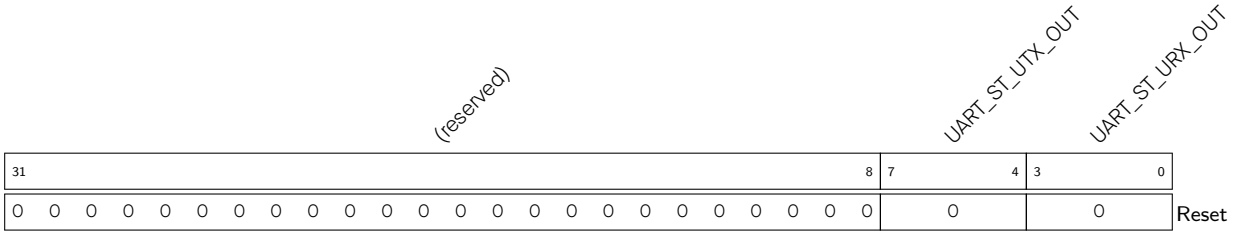
Register 21.23. UART_MEM_RX_STATUS_REG (0x006C)



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)

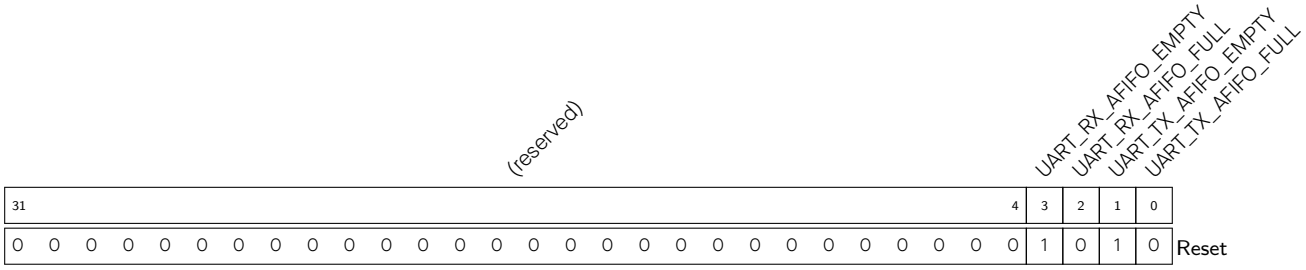
Register 21.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT Represents the status of the receiver. (RO)

UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 21.25. 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)

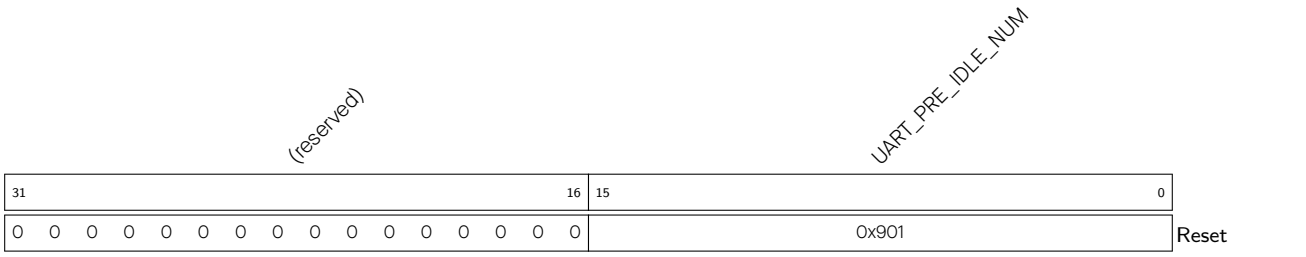
UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

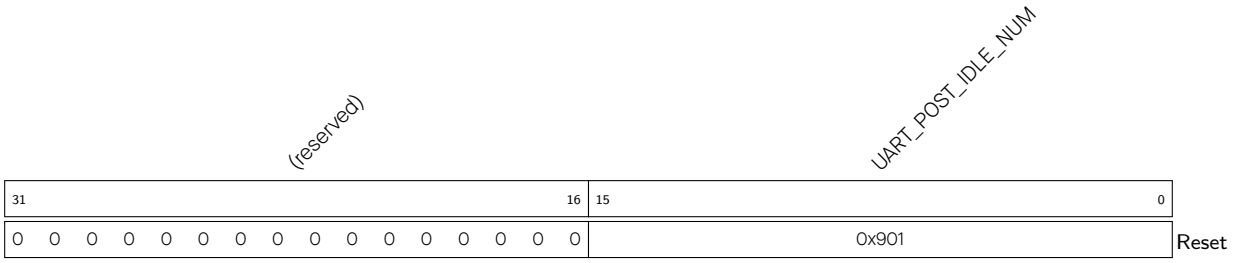
Register 21.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD.

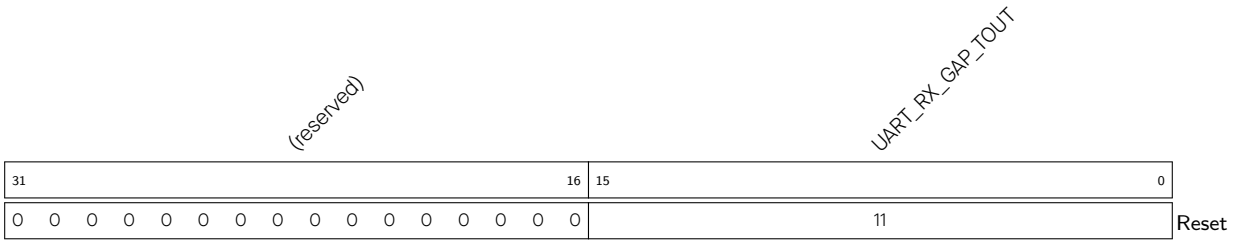
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



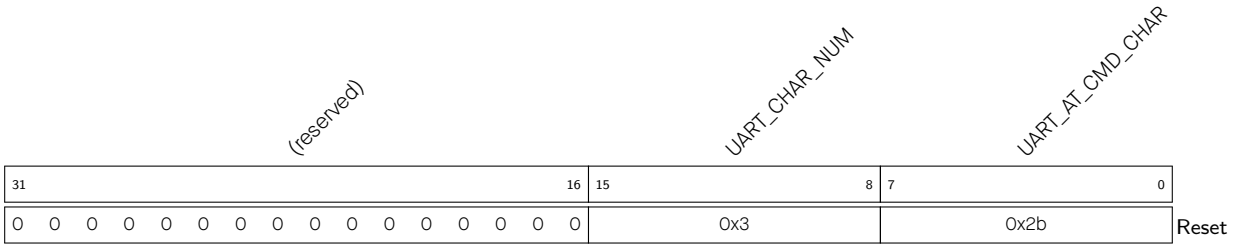
UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.28. 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)

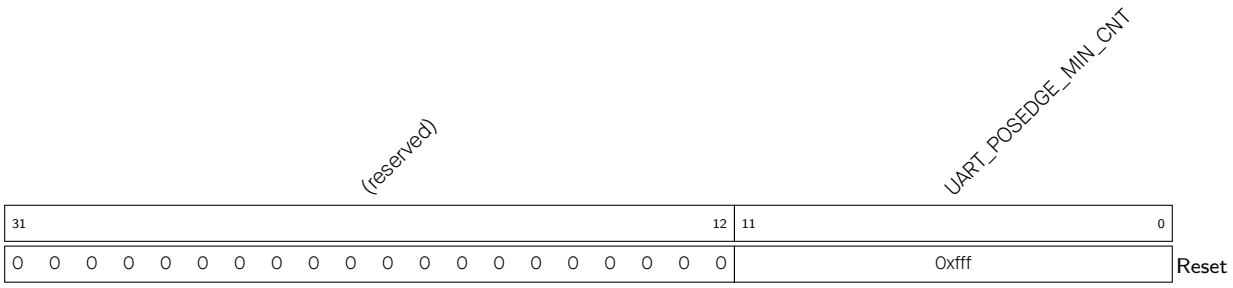
Register 21.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

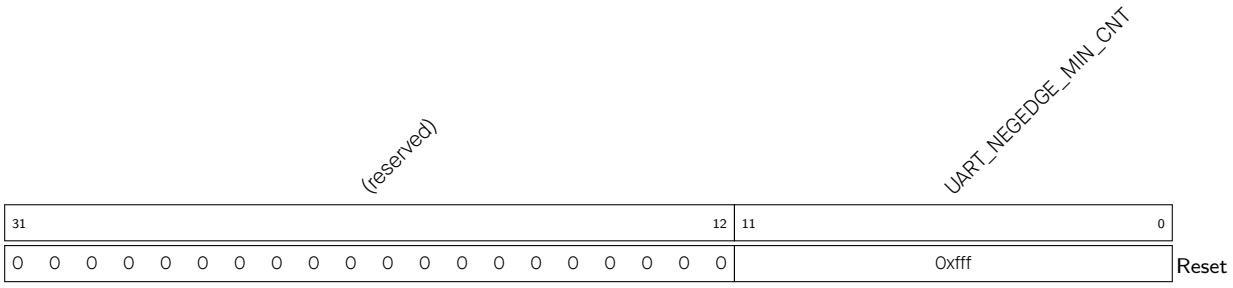
UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 21.30. UART_POSPULSE_REG (0x0074)



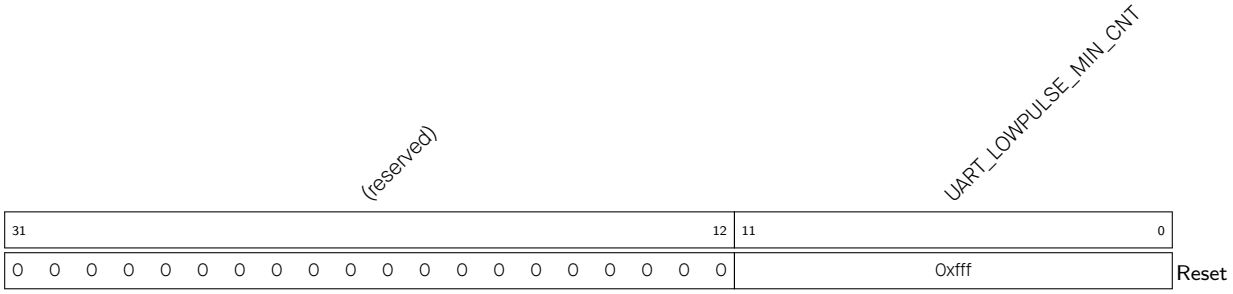
UART_POSEDGE_MIN_CNT Represents the minimal input clock counter value between two positive edges. It is used for baud rate detection. (RO)

Register 21.31. 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)

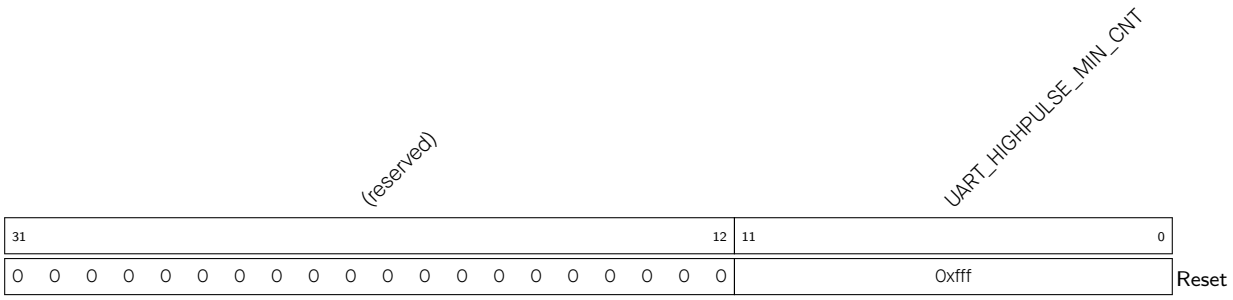
Register 21.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT Represents the minimum duration time of a low-level pulse. It is used for baud rate detection.

Measurement unit: APB_CLK clock cycle. (RO)

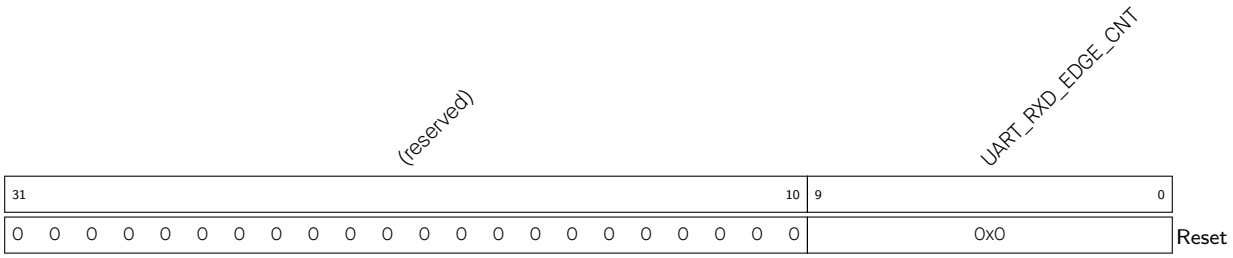
Register 21.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT Represents the maximum duration time for a high-level pulse. It is used for baud rate detection.

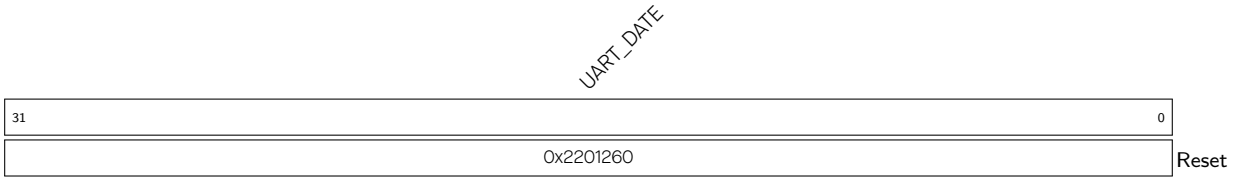
Measurement unit: APB_CLK clock cycle. (RO)

Register 21.34. UART_RXD_CNT_REG (0x0084)



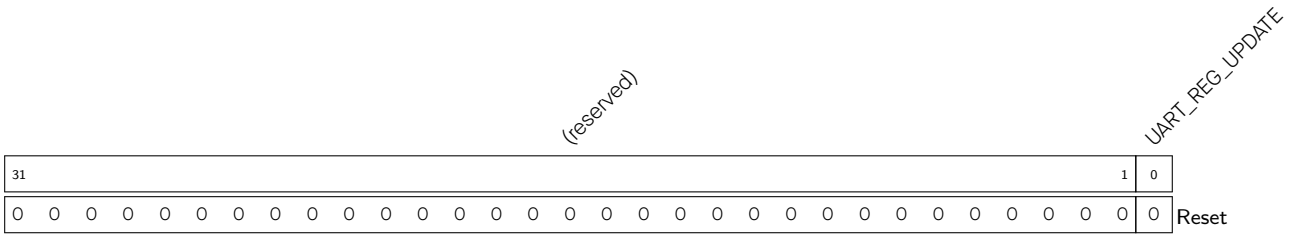
UART_RXD_EDGE_CNT Represents the number of RXD edge changes. It is used for baud rate detection. (RO)

Register 21.35. UART_DATE_REG (0x008C)



UART_DATE Version control register. (R/W)

Register 21.36. UART_REG_UPDATE_REG (0x0098)



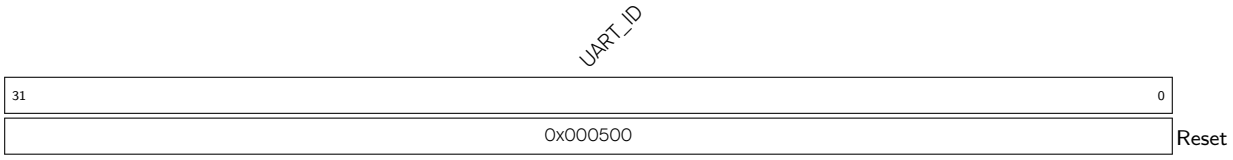
UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 21.37. UART_ID_REG (0x009C)



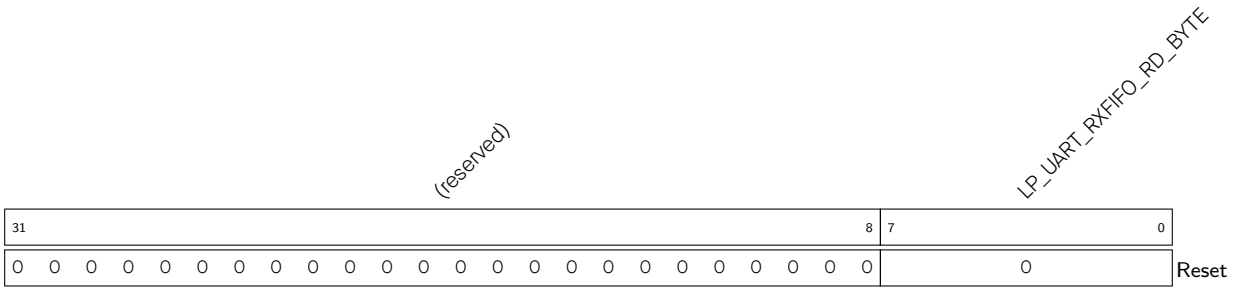
UART_ID Configures the UART ID. (R/W)

21.8.2 LP UART Registers

The addresses in this section are relative to LP UART base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

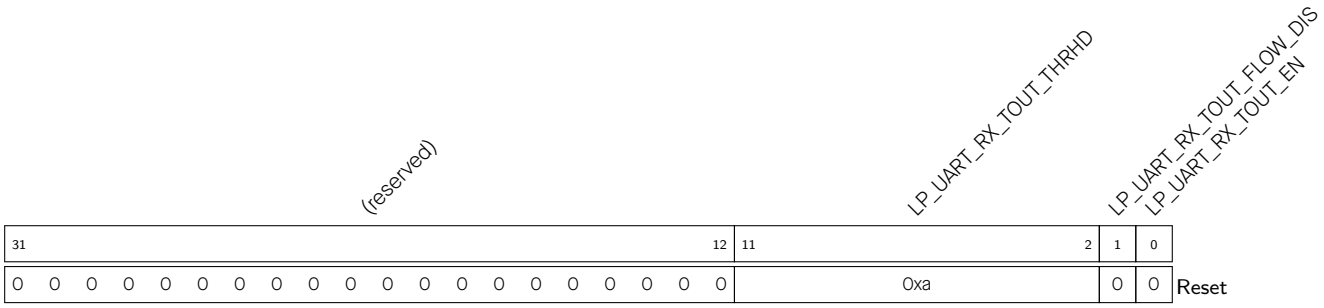
Register 21.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE Represents the data LP UART *n* read from FIFO.

Measurement unit: byte. (RO)

Register 21.39. LP_UART_TOUT_CONF_SYNC_REG (0x0064)



LP_UART_RX_TOUT_EN Configures whether or not to enable LP UART receiver's timeout function.

0: Disable

1: Enable

(R/W)

LP_UART_RX_TOUT_FLOW_DIS Configures whether or not to disable the idle status counter when hardware flow control is enabled.

0: Invalid. No effect

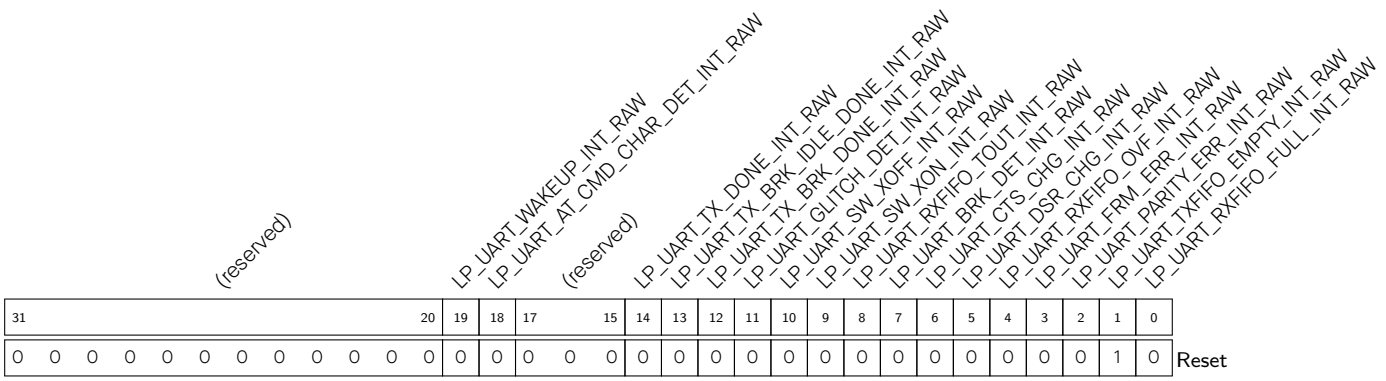
1: Disable

(R/W)

LP_UART_RX_TOUT_THRHD Configures the amount of time that the bus can remain idle before time-out.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.40. LP_UART_INT_RAW_REG (0x0004)



LP_UART_RXFIFO_FULL_INT_RAW The raw interrupt status of LP_UART_RXFIFO_FULL_INT. (R/WTC/SS)

LP_UART_TXFIFO_EMPTY_INT_RAW The raw interrupt status of LP_UART_TXFIFO_EMPTY_INT. (R/WTC/SS)

LP_UART_PARITY_ERR_INT_RAW The raw interrupt status of LP_UART_PARITY_ERR_INT. (R/WTC/SS)

LP_UART_FRM_ERR_INT_RAW The raw interrupt status of LP_UART_FRM_ERR_INT. (R/WTC/SS)

LP_UART_RXFIFO_OVF_INT_RAW The raw interrupt status of LP_UART_RXFIFO_OVF_INT. (R/WTC/SS)

LP_UART_DSR_CHG_INT_RAW The raw interrupt status of LP_UART_DSR_CHG_INT. (R/WTC/SS)

LP_UART_CTS_CHG_INT_RAW The raw interrupt status of LP_UART_CTS_CHG_INT. (R/WTC/SS)

LP_UART_BRK_DET_INT_RAW The raw interrupt status of LP_UART_BRK_DET_INT. (R/WTC/SS)

LP_UART_RXFIFO_TOUT_INT_RAW The raw interrupt status of LP_UART_RXFIFO_TOUT_INT. (R/WTC/SS)

LP_UART_SW_XON_INT_RAW The raw interrupt status of LP_UART_SW_XON_INT. (R/WTC/SS)

LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XOFF_INT. (R/WTC/SS)

LP_UART_GLITCH_DET_INT_RAW The raw interrupt status of LP_UART_GLITCH_DET_INT. (R/WTC/SS)

LP_UART_TX_BRK_DONE_INT_RAW The raw interrupt status of LP_UART_TX_BRK_DONE_INT. (R/WTC/SS)

LP_UART_TX_BRK_IDLE_DONE_INT_RAW The raw interrupt status of LP_UART_TX_BRK_IDLE_DONE_INT. (R/WTC/SS)

Continued on the next page...

Register 21.40. LP_UART_INT_RAW_REG (0x0004)

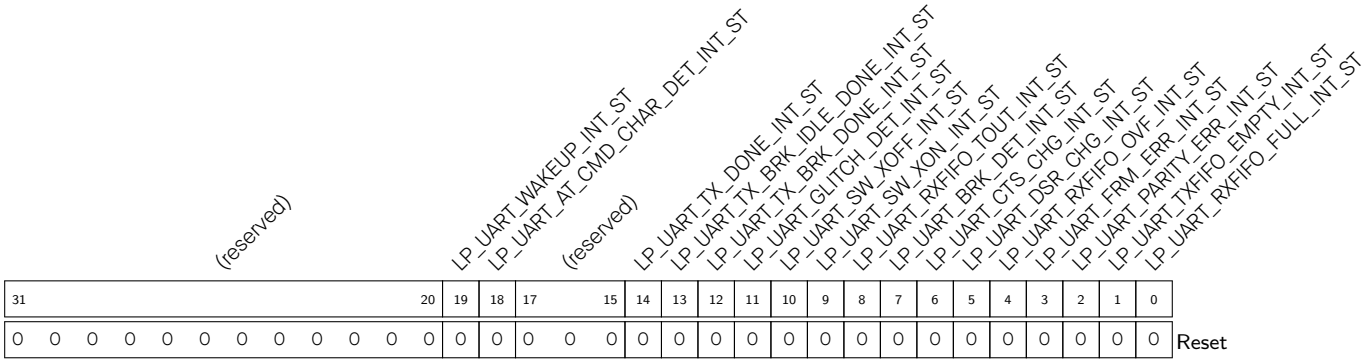
Continued from the previous page...

LP_UART_TX_DONE_INT_RAW The raw interrupt status of LP_UART_TX_DONE_INT. (R/WTC/SS)

LP_UART_AT_CMD_CHAR_DET_INT_RAW The raw interrupt status of LP_UART_AT_CMD_CHAR_DET_INT. (R/WTC/SS)

LP_UART_WAKEUP_INT_RAW The raw interrupt status of LP_UART_WAKEUP_INT. (R/WTC/SS)

Register 21.41. LP_UART_INT_ST_REG (0x0008)



LP_UART_RXFIFO_FULL_INT_ST The masked interrupt status of LP_UART_RXFIFO_FULL_INT.(RO)

LP_UART_TXFIFO_EMPTY_INT_ST The masked interrupt status of LP_UART_TXFIFO_EMPTY_INT. (RO)

LP_UART_PARITY_ERR_INT_ST The masked interrupt status of LP_UART_PARITY_ERR_INT. (RO)

LP_UART_FRM_ERR_INT_ST The masked interrupt status of LP_UART_FRM_ERR_INT. (RO)

LP_UART_RXFIFO_OVF_INT_ST The masked interrupt status of LP_UART_RXFIFO_OVF_INT. (RO)

LP_UART_DSR_CHG_INT_ST The masked interrupt status of LP_UART_DSR_CHG_INT. (RO)

LP_UART_CTS_CHG_INT_ST The masked interrupt status of LP_UART_CTS_CHG_INT. (RO)

LP_UART_BRK_DET_INT_ST The masked interrupt status of LP_UART_BRK_DET_INT. (RO)

LP_UART_RXFIFO_TOUT_INT_ST The masked interrupt status of LP_UART_RXFIFO_TOUT_INT. (RO)

LP_UART_SW_XON_INT_ST The masked interrupt status of LP_UART_SW_XON_INT. (RO)

LP_UART_SW_XOFF_INT_ST The masked interrupt status of LP_UART_SW_XOFF_INT. (RO)

LP_UART_GLITCH_DET_INT_ST The masked interrupt status of LP_UART_GLITCH_DET_INT. (RO)

LP_UART_TX_BRK_DONE_INT_ST The masked interrupt status of LP_UART_TX_BRK_DONE_INT. (RO)

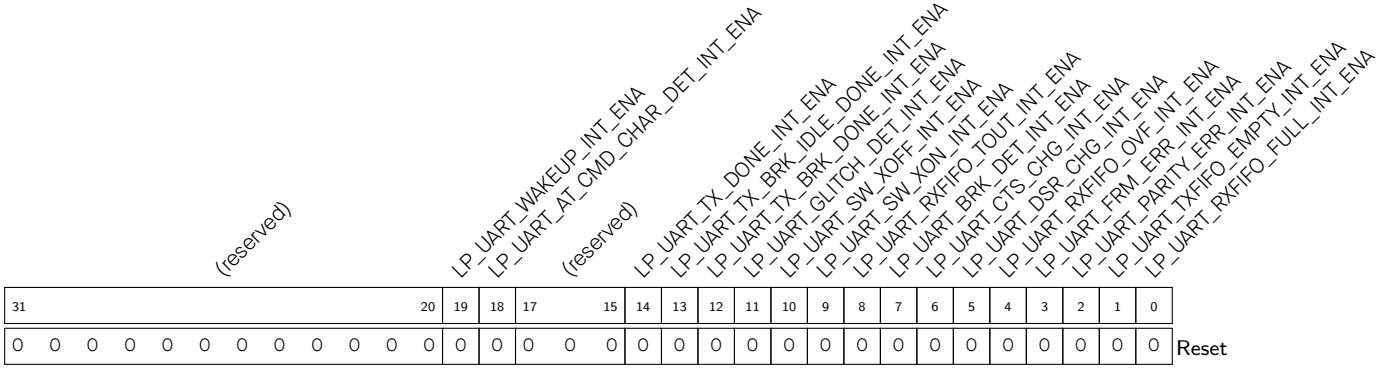
LP_UART_TX_BRK_IDLE_DONE_INT_ST The masked interrupt status of LP_UART_TX_BRK_IDLE_DONE_INT. (RO)

LP_UART_TX_DONE_INT_ST The masked interrupt status of LP_UART_TX_DONE_INT. (RO)

LP_UART_AT_CMD_CHAR_DET_INT_ST The masked interrupt status of LP_UART_AT_CMD_CHAR_DET_INT. (RO)

LP_UART_WAKEUP_INT_ST The masked interrupt status of LP_UART_WAKEUP_INT. (RO)

Register 21.42. LP_UART_INT_ENA_REG (0x000C)



LP_UART_RXFIFO_FULL_INT_ENA Write 1 to enable LP_UART_RXFIFO_FULL_INT. (R/W)

LP_UART_TXFIFO_EMPTY_INT_ENA Write 1 to enable LP_UART_TXFIFO_EMPTY_INT. (R/W)

LP_UART_PARITY_ERR_INT_ENA Write 1 to enable LP_UART_PARITY_ERR_INT. (R/W)

LP_UART_FRM_ERR_INT_ENA Write 1 to enable LP_UART_FRM_ERR_INT. (R/W)

LP_UART_RXFIFO_OVF_INT_ENA Write 1 to enable LP_UART_RXFIFO_OVF_INT. (R/W)

LP_UART_DSR_CHG_INT_ENA Write 1 to enable LP_UART_DSR_CHG_INT. (R/W)

LP_UART_CTS_CHG_INT_ENA Write 1 to enable LP_UART_CTS_CHG_INT. (R/W)

LP_UART_BRK_DET_INT_ENA Write 1 to enable LP_UART_BRK_DET_INT. (R/W)

LP_UART_RXFIFO_TOUT_INT_ENA Write 1 to enable LP_UART_RXFIFO_TOUT_INT. (R/W)

LP_UART_SW_XON_INT_ENA Write 1 to enable LP_UART_SW_XON_INT.(R/W)

LP_UART_SW_XOFF_INT_ENA Write 1 to enable LP_UART_SW_XOFF_INT. (R/W)

LP_UART_GLITCH_DET_INT_ENA Write 1 to enable LP_UART_GLITCH_DET_INT. (R/W)

LP_UART_TX_BRK_DONE_INT_ENA Write 1 to enable LP_UART_TX_BRK_DONE_INT. (R/W)

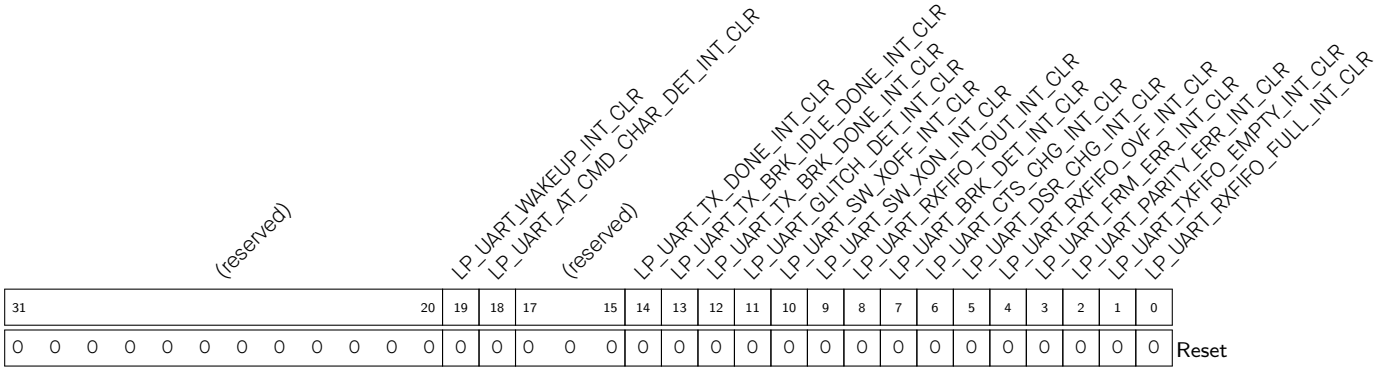
LP_UART_TX_BRK_IDLE_DONE_INT_ENA Write 1 to enable LP_UART_TX_BRK_IDLE_DONE_INT. (R/W)

LP_UART_TX_DONE_INT_ENA Write 1 to enable LP_UART_TX_DONE_INT. (R/W)

LP_UART_AT_CMD_CHAR_DET_INT_ENA Write 1 to enable LP_UART_AT_CMD_CHAR_DET_INT. (R/W)

LP_UART_WAKEUP_INT_ENA Write 1 to enable LP_UART_WAKEUP_INT. (R/W)

Register 21.43. LP_UART_INT_CLR_REG (0x0010)



LP_UART_RXFIFO_FULL_INT_CLR Write 1 to clear LP_UART_RXFIFO_FULL_INT. (WT)

LP_UART_TXFIFO_EMPTY_INT_CLR Write 1 to clear LP_UART_TXFIFO_EMPTY_INT. (WT)

LP_UART_PARITY_ERR_INT_CLR Write 1 to clear LP_UART_PARITY_ERR_INT. (WT)

LP_UART_FRM_ERR_INT_CLR Write 1 to clear LP_UART_FRM_ERR_INT. (WT)

LP_UART_RXFIFO_OVF_INT_CLR Write 1 to clear LP_UART_RXFIFO_OVF_INT. (WT)

LP_UART_DSR_CHG_INT_CLR Write 1 to clear LP_UART_DSR_CHG_INT. (WT)

LP_UART_CTS_CHG_INT_CLR Write 1 to clear LP_UART_CTS_CHG_INT. (WT)

LP_UART_BRK_DET_INT_CLR Write 1 to clear LP_UART_BRK_DET_INT. (WT)

LP_UART_RXFIFO_TOUT_INT_CLR Write 1 to clear LP_UART_RXFIFO_TOUT_INT. (WT)

LP_UART_SW_XON_INT_CLR Write 1 to clear LP_UART_SW_XON_INT. (WT)

LP_UART_SW_XOFF_INT_CLR Write 1 to clear LP_UART_SW_XOFF_INT. (WT)

LP_UART_GLITCH_DET_INT_CLR Write 1 to clear LP_UART_GLITCH_DET_INT. (WT)

LP_UART_TX_BRK_DONE_INT_CLR Write 1 to clear LP_UART_TX_BRK_DONE_INT. (WT)

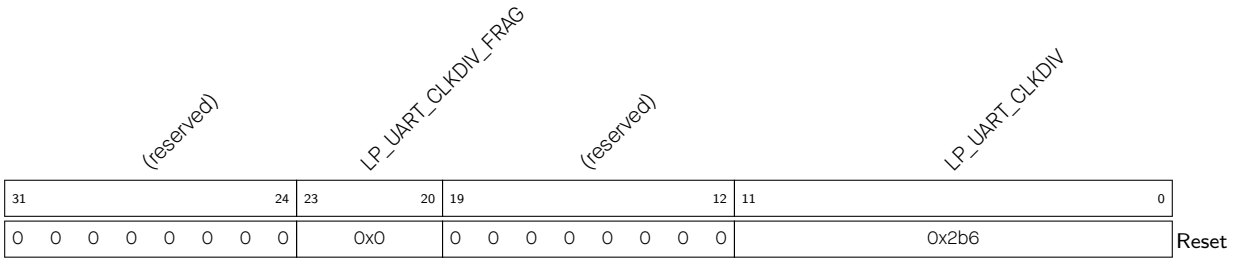
LP_UART_TX_BRK_IDLE_DONE_INT_CLR Write 1 to clear LP_UART_TX_BRK_IDLE_DONE_INT. (WT)

LP_UART_TX_DONE_INT_CLR Write 1 to clear LP_UART_TX_DONE_INT. (WT)

LP_UART_AT_CMD_CHAR_DET_INT_CLR Write 1 to clear LP_UART_AT_CMD_CHAR_DET_INT. (WT)

LP_UART_WAKEUP_INT_CLR Write 1 to clear LP_UART_WAKEUP_INT. (WT)

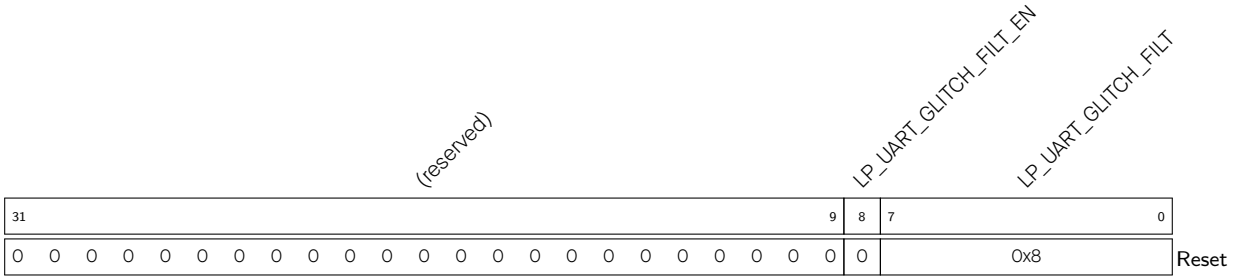
Register 21.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



LP_UART_CLKDIV Configures the integral part of the divisor for baud rate generation. (R/W)

LP_UART_CLKDIV_FRAG Configures the fractional part of the divisor for baud rate generation. (R/W)

Register 21.45. LP_UART_RX_FILT_REG (0x0018)



LP_UART_GLITCH_FILT Configures the width of a pulse to be filtered.

Measurement unit: UART Core's clock cycle.

Pulses whose width is lower than this value will be ignored. (R/W)

LP_UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 21.46. LP_UART_CONF0_SYNC_REG (0x0020)

Bits	Field	Reset
31:24	(reserved)	0 0 0 0 0 0 0 0
23	LP_UART_TXFIFO_RST	0
22	LP_UART_RXFIFO_RST	0
21	LP_UART_SW_RTS	0
20	LP_UART_MEM_CLK_EN	0
19	(reserved)	0
18	LP_UART_ERR_WR_MASK	0
17	LP_UART_DIS_RX_DAT_OVF	0
16	LP_UART_TXD_INV	0
15	LP_UART_RXD_INV	0
14	(reserved)	0
13	LP_UART_TX_FLOW_EN	0
12	LP_UART_LOOPBACK	0
11:7	(reserved)	0
6	LP_UART_TXD_BRK	0
5:4	LP_UART_STOP_BIT_NUM	1
3:2	LP_UART_BIT_NUM	3
1	LP_UART_PARITY_EN	0
0	LP_UART_PARITY	0

LP_UART_PARITY Configures the parity check mode.
0: Even parity
1: Odd parity
(R/W)

LP_UART_PARITY_EN Configures whether or not to enable LP UART parity check.
0: Disable
1: Enable
(R/W)

LP_UART_BIT_NUM Configures the number of data bits.
0: 5 bits
1: 6 bits
2: 7 bits
3: 8 bits
(R/W)

LP_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)

LP_UART_TXD_BRK Configures whether or not to send NULL characters when finishing data transmission.
0: Not send
1: Send
(R/W)

LP_UART_LOOPBACK Configures whether or not to enable LP UART loopback test.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 21.46. LP_UART_CONF0_SYNC_REG (0x0020)

Continued from the previous page...

LP_UART_TX_FLOW_EN Configures whether or not to enable flow control for the transmitter.
0: Disable
1: Enable
(R/W)

LP_UART_RXD_INV Configures whether or not to invert the level of LP UART RXD signal.
0: Not invert
1: Invert
(R/W)

LP_UART_TXD_INV Configures whether or not to invert the level of LP UART TXD signal.
0: Not invert
1: Invert
(R/W)

LP_UART_DIS_RX_DAT_OVF Configures whether or not to disable data overflow detection for the LP UART receiver.
0: Enable
1: Disable
(R/W)

LP_UART_ERR_WR_MASK Configures whether or not to store the received data with errors into FIFO.
0: Store
1: Not store
(R/W)

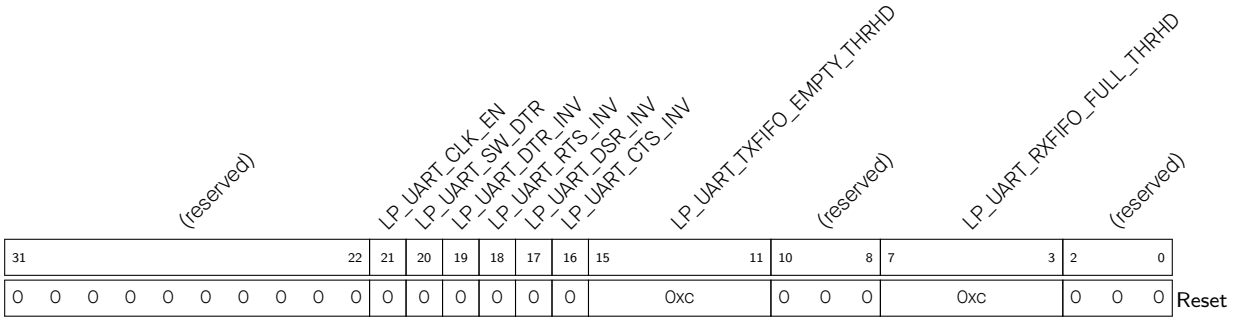
LP_UART_MEM_CLK_EN Configures whether or not to enable clock gating for LP UART memory.
0: Disable
1: Enable
(R/W)

LP_UART_SW_RTS Configures the RTS signal used in software flow control.
0: The LP UART transmitter is not allowed to send data.
1: The LP UART transmitted is allowed to send data.
(R/W)

LP_UART_RXFIFO_RST Configures whether or not to reset the LP UART RX FIFO.
0: Not reset
1: Reset
(R/W)

LP_UART_TXFIFO_RST Configures whether or not to reset the LP UART TX FIFO.
0: Not reset
1: Reset
(R/W)

Register 21.47. LP_UART_CONF1_REG (0x0024)



LP_UART_RXFIFO_FULL_THRHD Configures the threshold for RX FIFO being full.

Measurement unit: byte. (R/W)

LP_UART_TXFIFO_EMPTY_THRHD Configures the threshold for TX FIFO being empty.

Measurement unit: byte. (R/W)

LP_UART_CTS_INV Configures whether or not to invert the level of LP UART CTS signal.

0: Not invert

1: Invert

(R/W)

LP_UART_DSR_INV Configures whether or not to invert the level of LP UART DSR signal.

0: Not invert

1: Invert

(R/W)

LP_UART_RTS_INV Configures whether or not to invert the level of LP UART RTS signal.

0: Not invert

1: Invert

(R/W)

LP_UART_DTR_INV Configures whether or not to invert the level of LP UART DTR signal.

0: Not invert

1: Invert

(R/W)

LP_UART_SW_DTR Configures the DTR signal used in software flow control.

0: Data to be transmitted is not ready.

1: Data to be transmitted is ready.

(R/W)

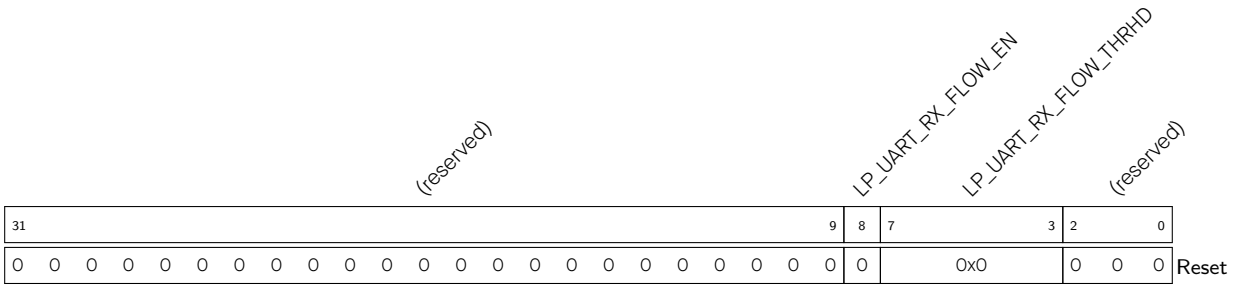
LP_UART_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 21.48. LP_UART_HWFC_CONF_SYNC_REG (0x002C)



LP_UART_RX_FLOW_THRHD Configures the maximum number of data bytes that can be received during hardware flow control.

Measurement unit: byte. (R/W)

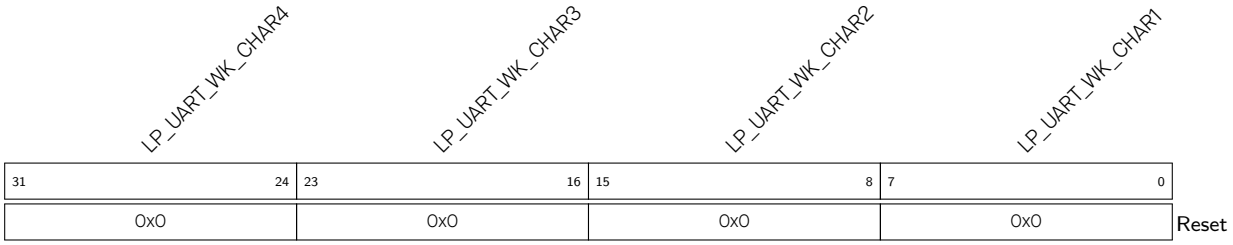
LP_UART_RX_FLOW_EN Configures whether or not to enable the LP UART receiver.

0: Disable

1: Enable

(R/W)

Register 21.49. LP_UART_SLEEP_CONF0_REG (0x0030)



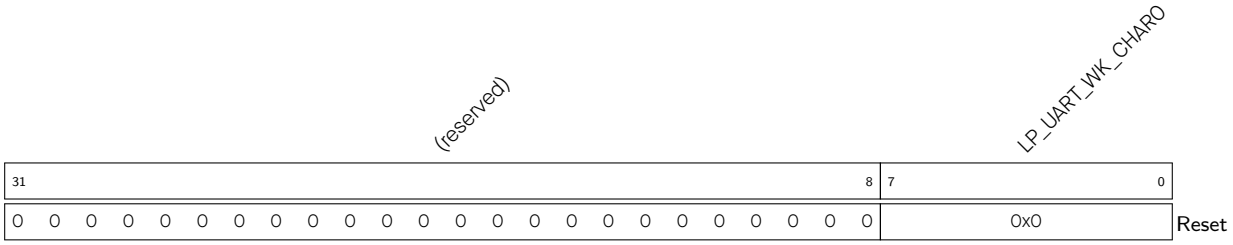
LP_UART_WK_CHAR1 Configures wakeup character 1. (R/W)

LP_UART_WK_CHAR2 Configures wakeup character 2. (R/W)

LP_UART_WK_CHAR3 Configures wakeup character 3. (R/W)

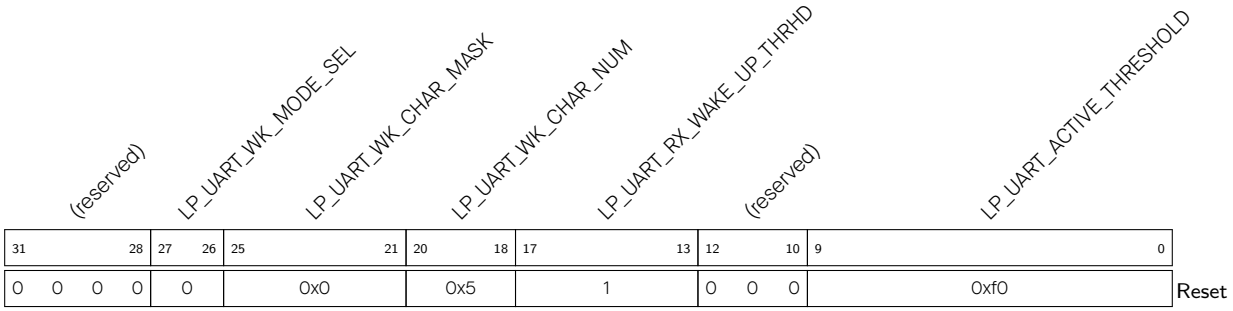
LP_UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 21.50. LP_UART_SLEEP_CONF1_REG (0x0034)



LP_UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 21.51. LP_UART_SLEEP_CONF2_REG (0x0038)



LP_UART_ACTIVE_THRESHOLD Configures the number of RXD edge changes to wake up the chip in wakeup mode 0. (R/W)

LP_UART_RX_WAKE_UP_THRHD Configures the number of received data bytes to wake up the chip in wakeup mode 1. (R/W)

LP_UART_WK_CHAR_NUM Configures the number of wakeup characters. (R/W)

LP_UART_WK_CHAR_MASK Configures whether or not to mask wakeup characters.

0: Not mask

1: Mask

(R/W)

LP_UART_WK_MODE_SEL Configures which wakeup mode to select.

0: Mode 0

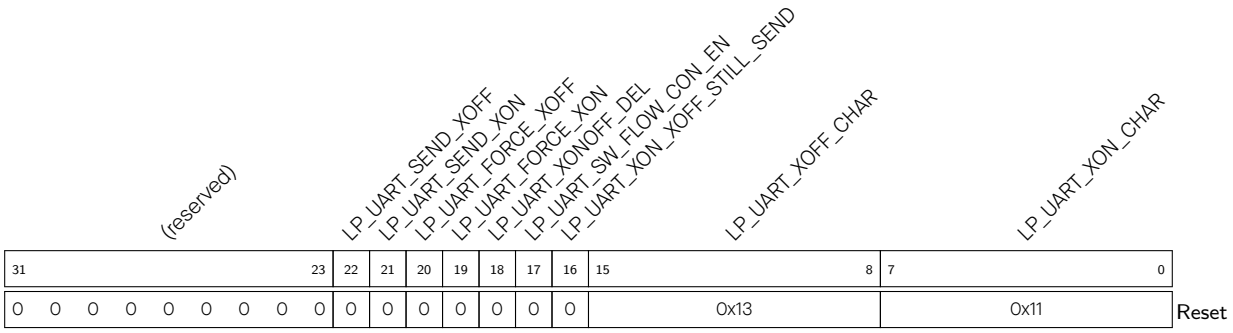
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 21.52. LP_UART_SWFC_CONF0_SYNC_REG (0x003C)



LP_UART_XON_CHAR Configures the XON character for flow control. (R/W)

LP_UART_XOFF_CHAR Configures the XOFF character for flow control. (R/W)

LP_UART_XON_XOFF_STILL_SEND Configures whether the LP UART transmitter can send XON or XOFF characters when it is disabled.

0: Cannot send

1: Can send

(R/W)

LP_UART_SW_FLOW_CON_EN Configures whether or not to enable software flow control.

0: Disable

1: Enable

(R/W)

LP_UART_XONOFF_DEL Configures whether or not to remove flow control characters from the received data.

0: Not move

1: Move

(R/W)

LP_UART_FORCE_XON Configures whether the transmitter continues to sending data.

0: Not send

1: Send

(R/W)

LP_UART_FORCE_XOFF Configures whether or not to stop the transmitter from sending data.

0: Not stop

1: Stop

(R/W)

LP_UART_SEND_XON Configures whether or not to send XON characters.

0: Not send

1: Send

(R/W/SS/SC)

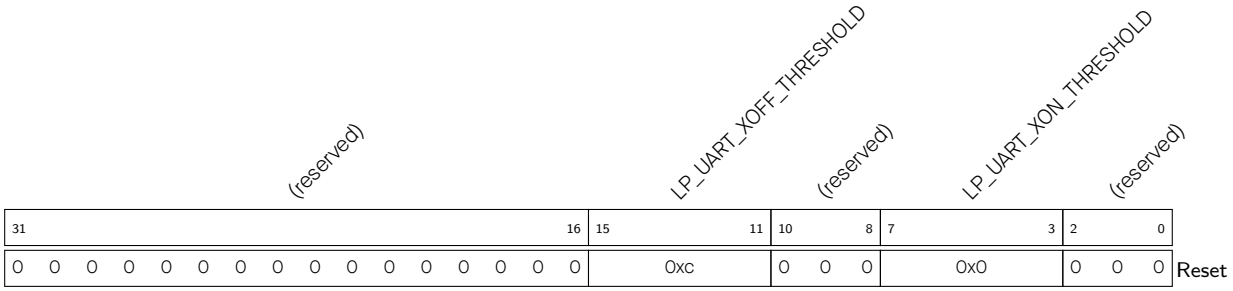
LP_UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 21.53. LP_UART_SWFC_CONF1_REG (0x0040)



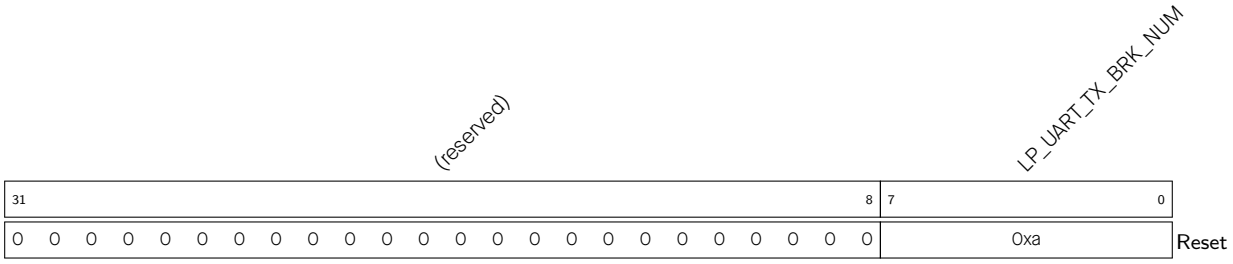
LP_UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

Measurement unit: byte. (R/W)

LP_UART_XOFF_THRESHOLD Configures the threshold for data in RX FIFO to send XOFF characters in software flow control.

Measurement unit: byte. (R/W)

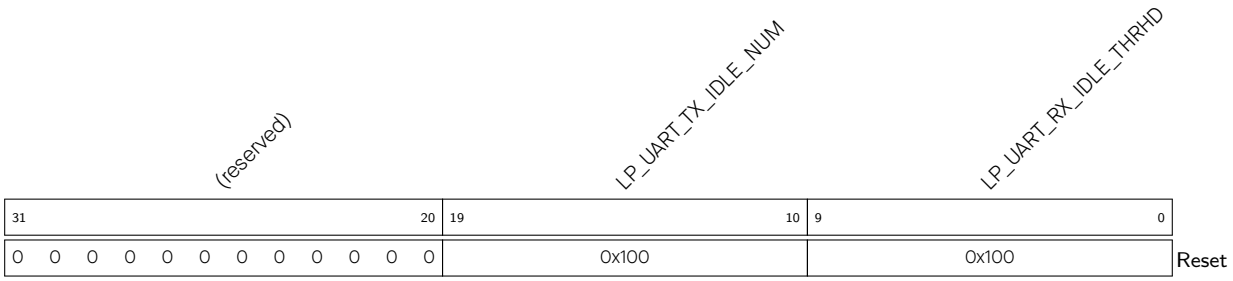
Register 21.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)



LP_UART_TX_BRK_NUM Configures the number of NULL characters to be sent after finishing data transmission.

Valid only when LP_UART_TXD_BRK is 1. (R/W)

Register 21.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)



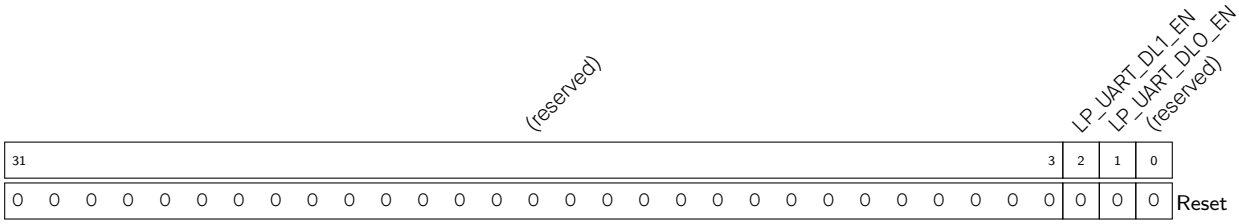
LP_UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

LP_UART_TX_IDLE_NUM Configures the interval between two data transfers.

Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.56. LP_UART_DELAY_CONF_SYNC_REG (0x004C)



LP_UART_DL0_EN Configures whether to add a turnaround delay of 1 bit before the start bit.

0: Not add

1: Add

(R/W)

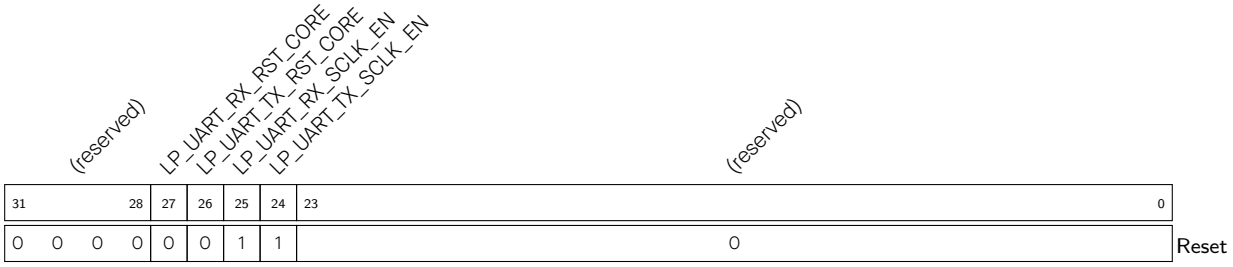
LP_UART_DL1_EN Configures whether to add a turnaround delay of 1 bit after the stop bit.

0: Not add

1: Add

(R/W)

Register 21.57. LP_UART_CLK_CONF_REG (0x0088)



LP_UART_TX_SCLK_EN Configures whether or not to enable LP UART TX clock.

0: Disable

1: Enable

(R/W)

LP_UART_RX_SCLK_EN Configures whether or not to enable LP UART RX clock.

0: Disable

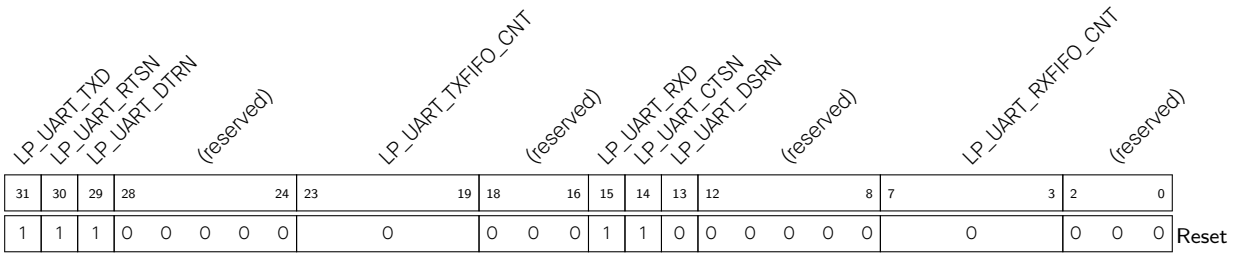
1: Enable

(R/W)

LP_UART_TX_RST_CORE Write 1 and then write 0 to reset LP UART TX. (R/W)

LP_UART_RX_RST_CORE Write 1 and then write 0 to reset LP UART RX. (R/W)

Register 21.58. LP_UART_STATUS_REG (0x001C)



LP_UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DSRN Represents the level of the internal LP UART DSR signal. (RO)

LP_UART_CTSN Represents the level of the internal LP UART CTS signal. (RO)

LP_UART_RXD Represents the level of the internal LP UART RXD signal. (RO)

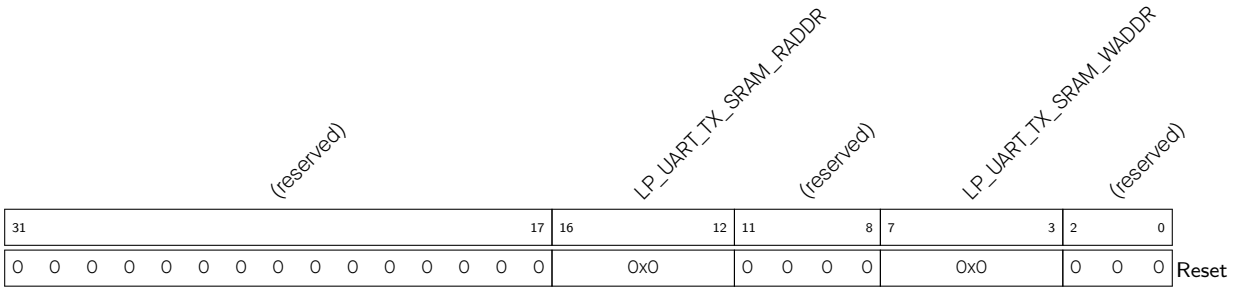
LP_UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DTRN Represents the level of the internal LP UART DTR signal. (RO)

LP_UART_RTSN Represents the level of the internal LP UART RTS signal. (RO)

LP_UART_TXD Represents the level of the internal LP UART TXD signal. (RO)

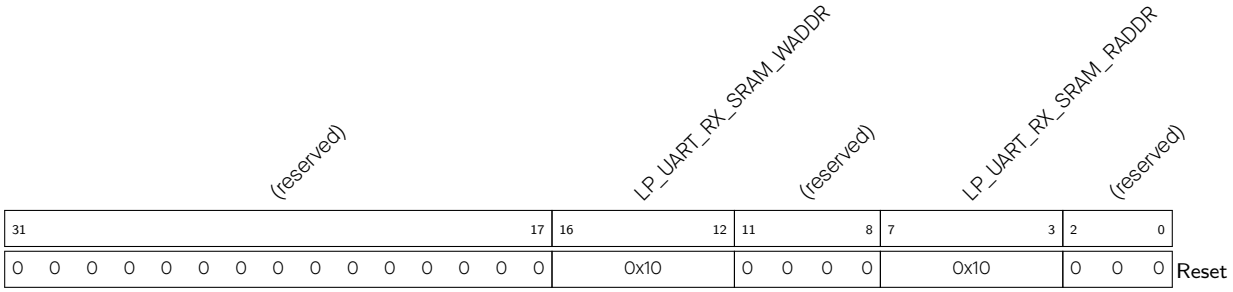
Register 21.59. LP_UART_MEM_TX_STATUS_REG (0x0068)



LP_UART_TX_SRAM_WADDR Represents the offset address to write TX FIFO. (RO)

LP_UART_TX_SRAM_RADDR Represents the offset address to read TX FIFO. (RO)

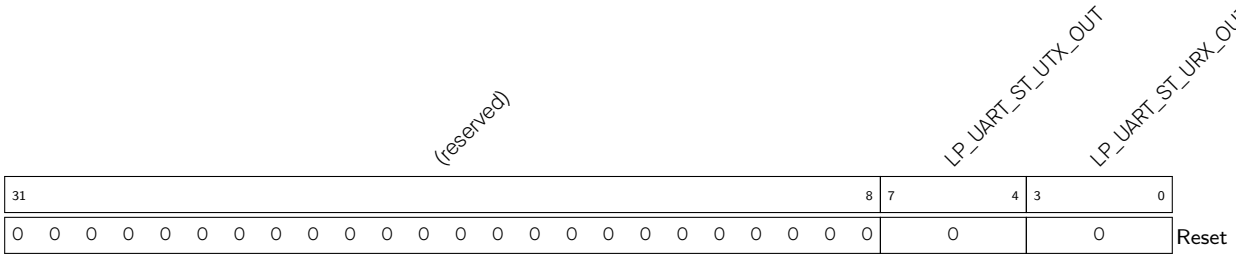
Register 21.60. LP_UART_MEM_RX_STATUS_REG (0x006C)



LP_UART_RX_SRAM_RADDR Represents the offset address to read RX FIFO. (RO)

LP_UART_RX_SRAM_WADDR Represents the offset address to write RX FIFO. (RO)

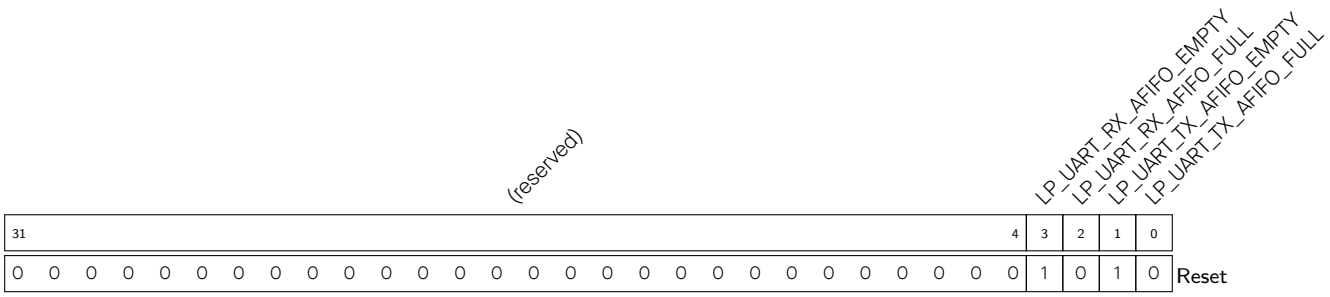
Register 21.61. LP_UART_FSM_STATUS_REG (0x0070)



LP_UART_ST_URX_OUT Represents the status of the receiver. (RO)

LP_UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 21.62. LP_UART_AFIFO_STATUS_REG (0x0090)



LP_UART_TX_AFIFO_FULL Represents whether or not the APB TX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

LP_UART_TX_AFIFO_EMPTY Represents whether or not the APB TX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

LP_UART_RX_AFIFO_FULL Represents whether or not the APB RX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

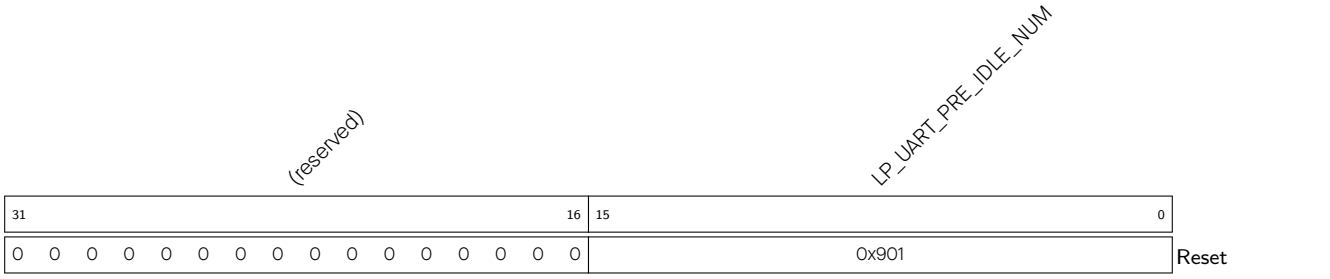
LP_UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

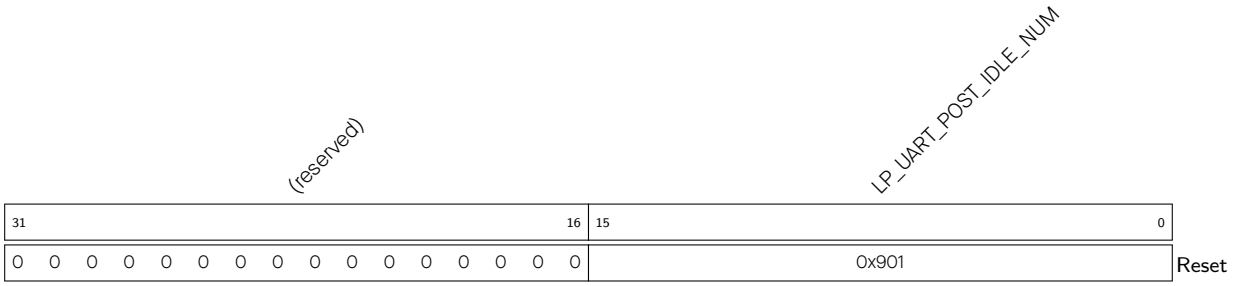
(RO)

Register 21.63. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



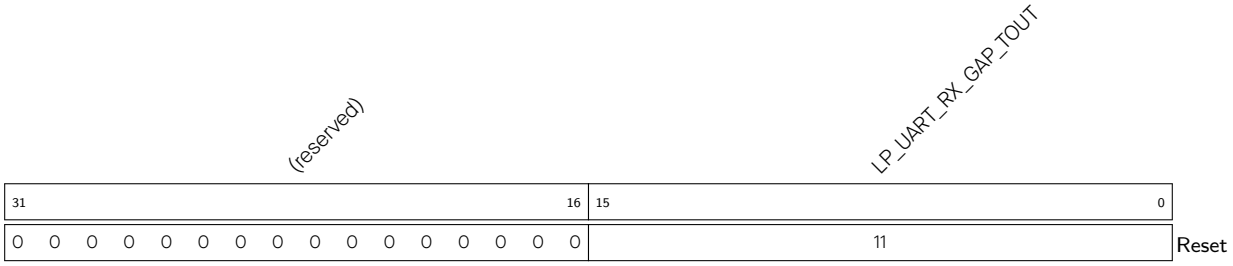
LP_UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.64. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



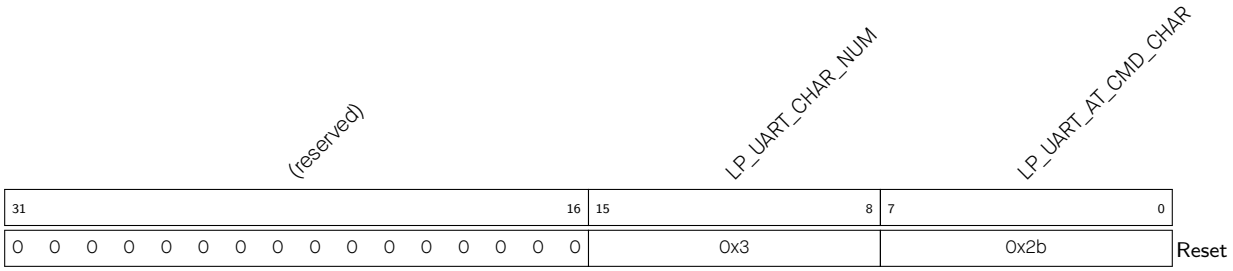
LP_UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 21.65. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



LP_UART_RX_GAP_TOUT Configures the interval between two AT_CMD characters. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

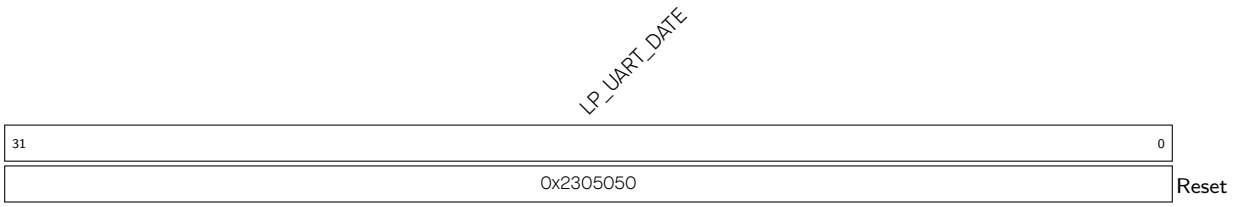
Register 21.66. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)



LP_UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

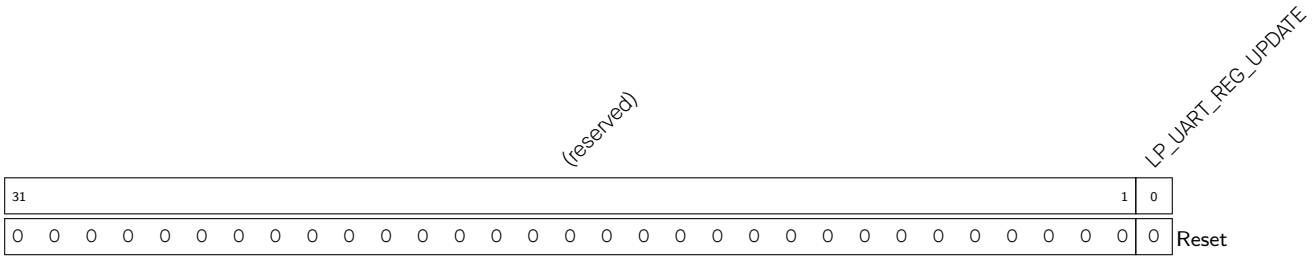
LP_UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 21.67. LP_UART_DATE_REG (0x008C)



LP_UART_DATE Version control register. (R/W)

Register 21.68. LP_UART_REG_UPDATE_REG (0x0098)



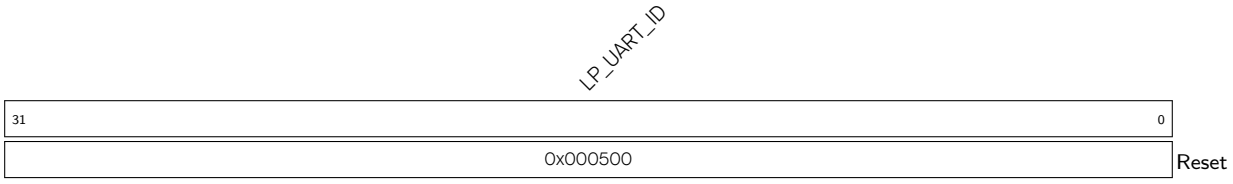
LP_UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 21.69. LP_UART_ID_REG (0x009C)



LP_UART_ID Configures the LP UART ID. (R/W)

21.8.3 UHCI Registers

The addresses in this section are relative to UHCI base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 21.70. 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_UART_SEL Select one UART from UART 0 ~ 3 to connect with UHCI.

0: Select UART0

1: Select UART1

2: Select UART2

3: Select UART3

4 ~ 7: No effect, as no UART will connect with UHCI

(R/W)

UHCI_UART1_CE Configures whether or not to connect UHCI with UART1.

0: Not connect

1: Connect

(R/W)

UHCI_SEPER_EN Configures whether or not to separate the data frame with a special character.

0: Not separate

1: Separate

(R/W)

UHCI_HEAD_EN Configures whether or not to encode the data packet with a formatting header.

0: Not use formatting header

1: Use formatting header

(R/W)

Continued on the next page...

Register 21.70. UHCI_CONF0_REG (0x0000)

Continued from the previous page...

UHCI_CRC_REC_EN Configures whether or not to enable the reception of the 16-bit CRC.

0: Disable

1: Enable

(R/W)

UHCI_UART_IDLE_EOF_EN Configures whether or not to stop receiving data when UART is idle.

0: Not stop

1: Stop

(R/W)

UHCI_LEN_EOF_EN Configures when the UHCI decoder stops receiving data.

0: Stops after receiving 0xC0

1: Stops when the number of received data bytes reach the specified value. When UHCI_HEAD_EN is 1, the specified value is the data length indicated by the UHCI packet header; when UHCI_HEAD_EN is 0, the specified value is the configured value.

(R/W)

UHCI_ENCODE_CRC_EN Configures whether or not to enable data integrity check by appending a 16 bit CCITT-CRC to the end of the data.

0: Disable

1: Enable

(R/W)

UHCI_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

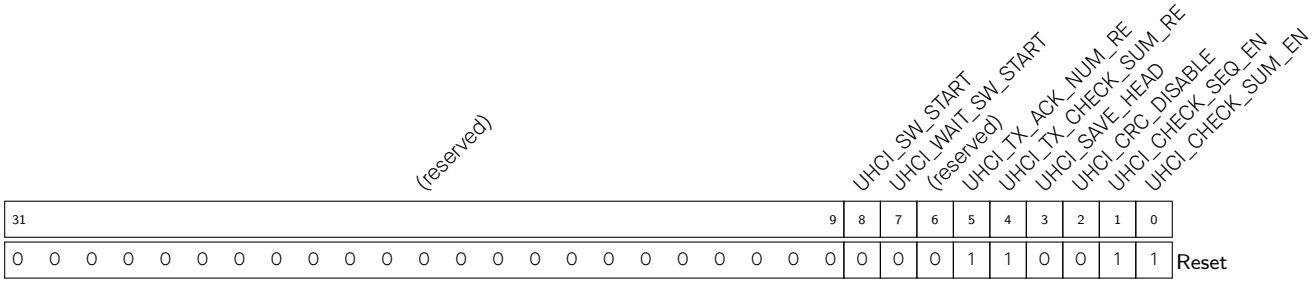
UHCI_UART_RX_BRK_EOF_EN Configures whether or not to stop UHCI from receiving data after UART has received a NULL frame.

0: Not stop

1: Stop

(R/W)

Register 21.71. 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 21.71. 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)

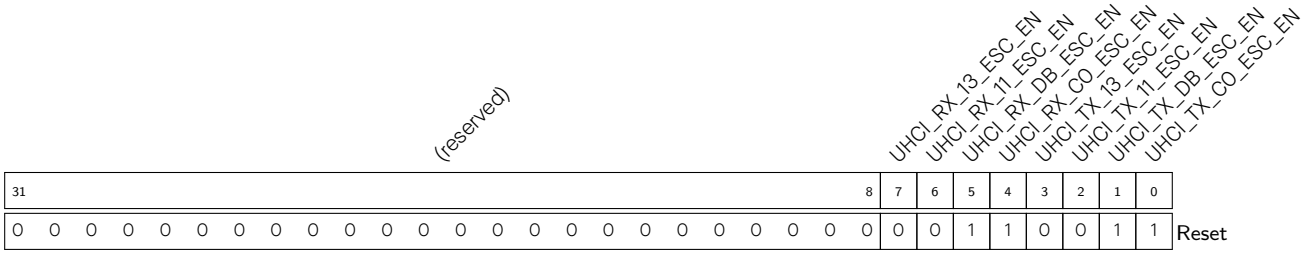
UHCI_SW_START Configures whether or not to send data packets when the encoder state machine is in ST_SW_WAIT state.

0: Not send

1: Send

(R/W/SC)

Register 21.72. 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...

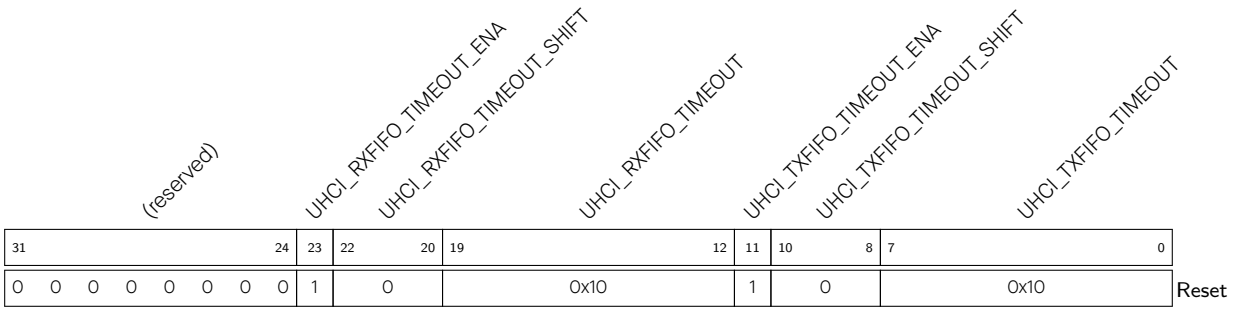
Register 21.72. UHCI_ESCAPE_CONF_REG (0x0020)

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 21.73. 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)

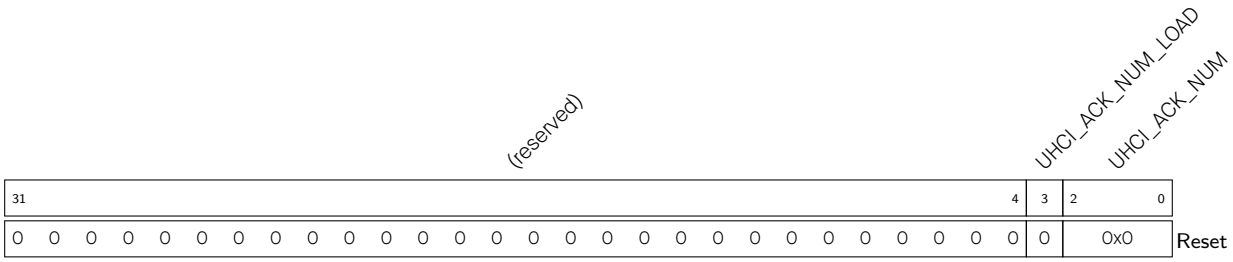
UHCI_TXFIFO_TIMEOUT_ENA Configures whether or not to enable the data reception timeout for TX FIFO.
0: Disable
1: Enable
(R/W)

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 21.74. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM Configures the number of acknowledgements used in software flow control. (R/W)

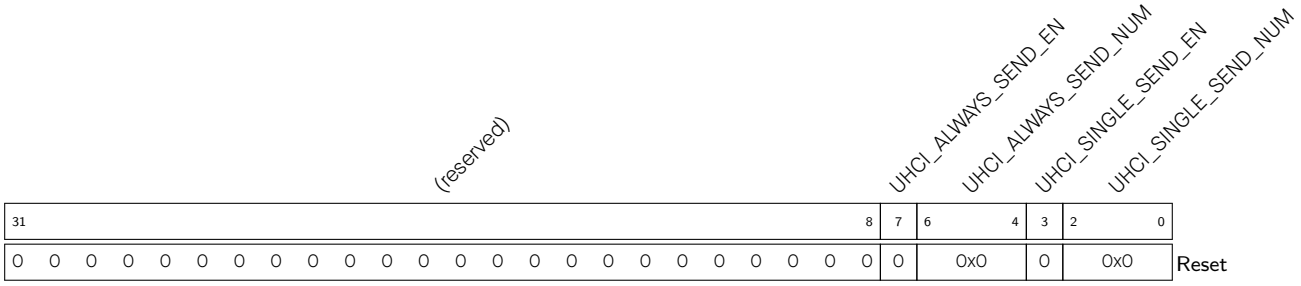
UHCI_ACK_NUM_LOAD Configures whether or not load acknowledgements.

0: Not load

1: Load

(WT)

Register 21.75. 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)

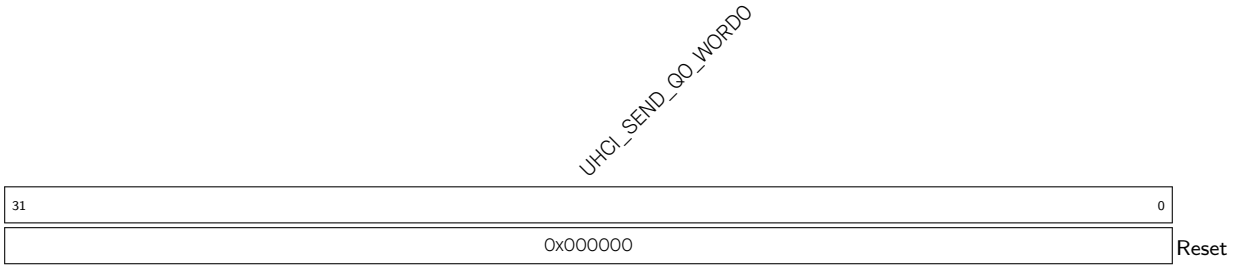
UHCI_ALWAYS_SEND_EN Configures whether or not to enable always_send mode.

0: Disable

1: Enable

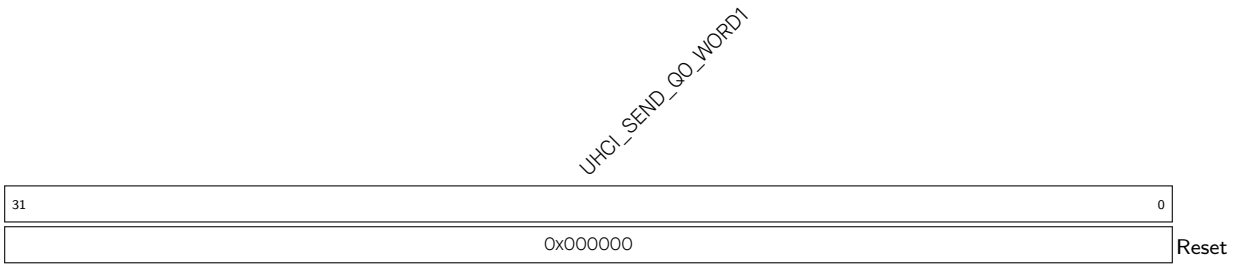
(R/W)

Register 21.76. UHCI_REG_Q0_WORD0_REG (0x0034)



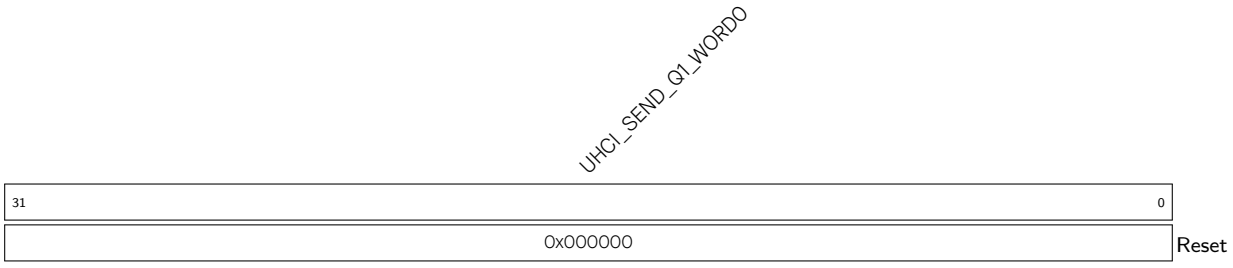
UHCI_SEND_Q0_WORD0 Data to be transmitted in Q0 register. (R/W)

Register 21.77. UHCI_REG_Q0_WORD1_REG (0x0038)



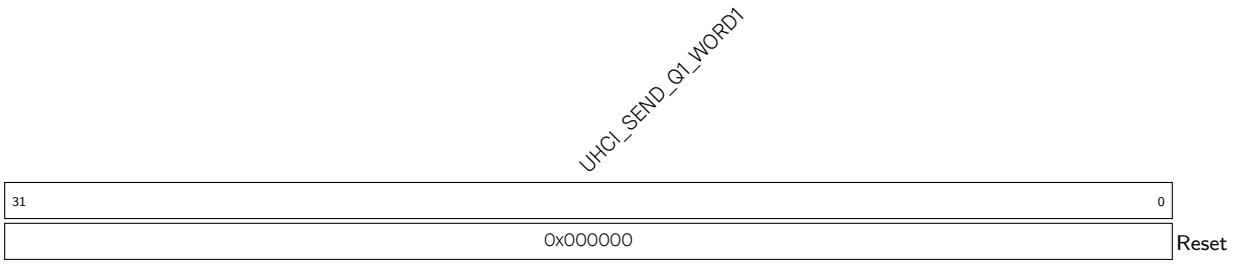
UHCI_SEND_Q0_WORD1 Data to be transmitted in Q0 register. (R/W)

Register 21.78. UHCI_REG_Q1_WORD0_REG (0x003C)



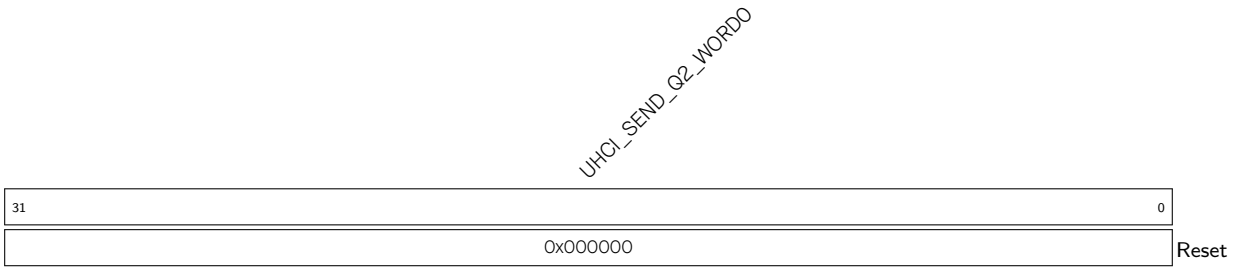
UHCI_SEND_Q1_WORD0 Data to be transmitted in Q1 register. (R/W)

Register 21.79. UHCI_REG_Q1_WORD1_REG (0x0040)



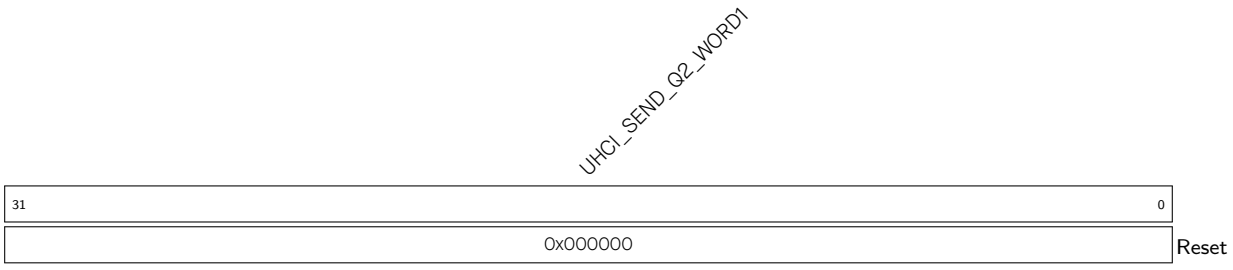
UHCI_SEND_Q1_WORD1 Data to be transmitted in Q1 register. (R/W)

Register 21.80. UHCI_REG_Q2_WORD0_REG (0x0044)



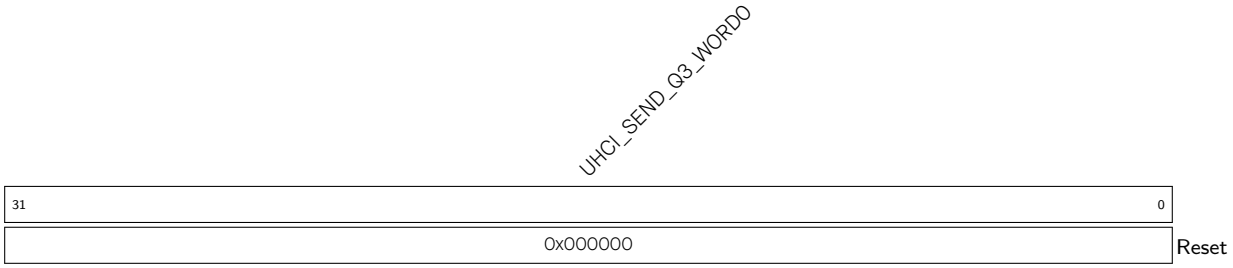
UHCI_SEND_Q2_WORD0 Data to be transmitted in Q2 register. (R/W)

Register 21.81. UHCI_REG_Q2_WORD1_REG (0x0048)



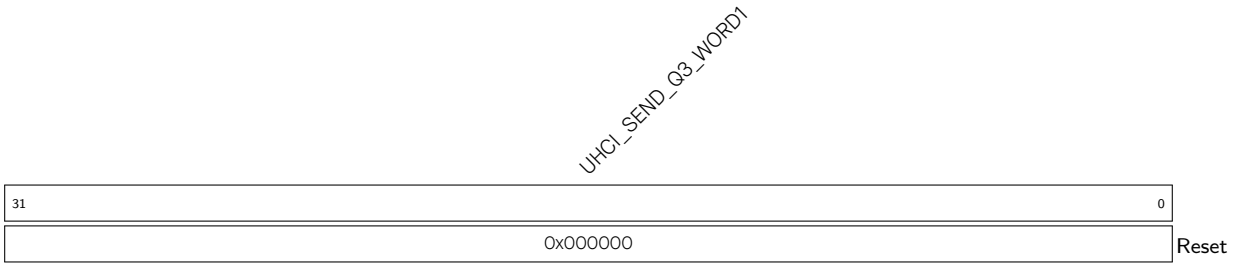
UHCI_SEND_Q2_WORD1 Data to be transmitted in Q2 register. (R/W)

Register 21.82. UHCI_REG_Q3_WORD0_REG (0x004C)



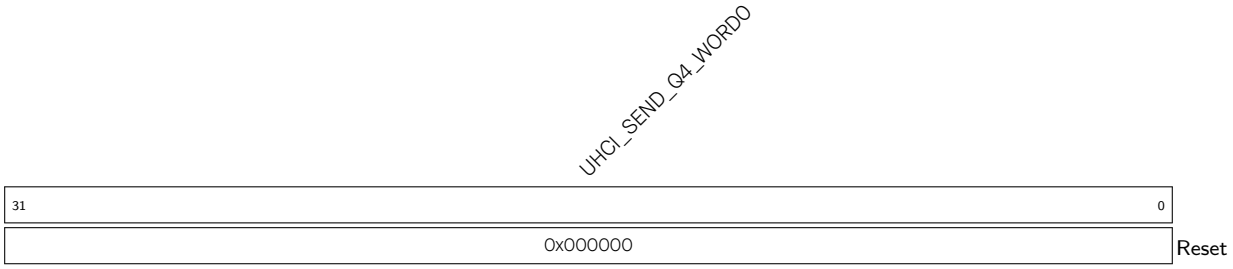
UHCI_SEND_Q3_WORD0 Data to be transmitted in Q3 register. (R/W)

Register 21.83. UHCI_REG_Q3_WORD1_REG (0x0050)



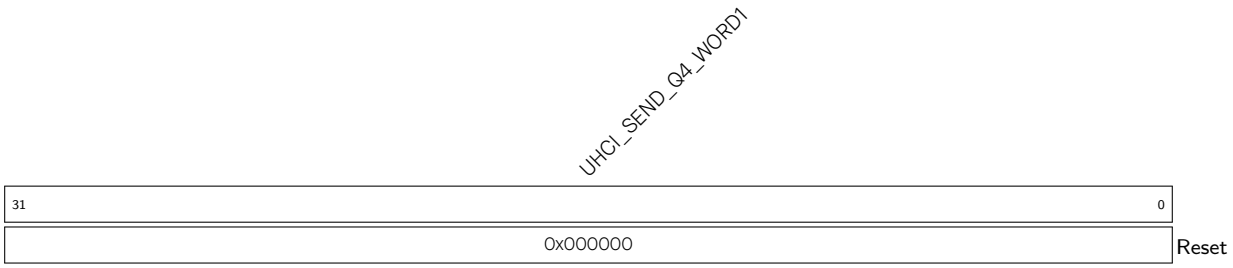
UHCI_SEND_Q3_WORD1 Data to be transmitted in Q3 register. (R/W)

Register 21.84. UHCI_REG_Q4_WORD0_REG (0x0054)



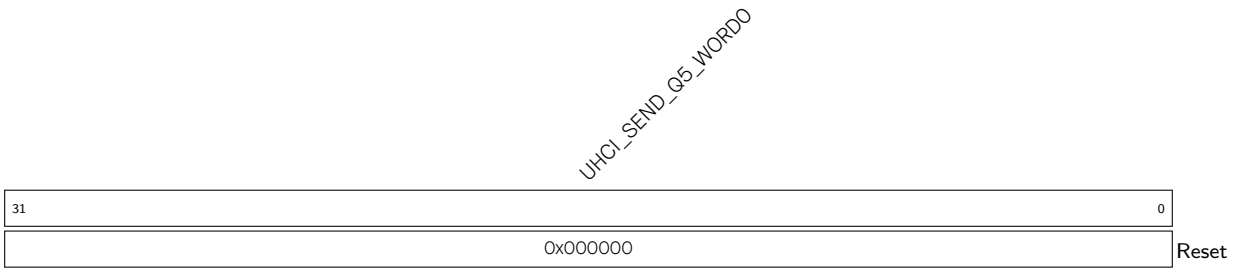
UHCI_SEND_Q4_WORD0 Data to be transmitted in Q4 register. (R/W)

Register 21.85. UHCI_REG_Q4_WORD1_REG (0x0058)



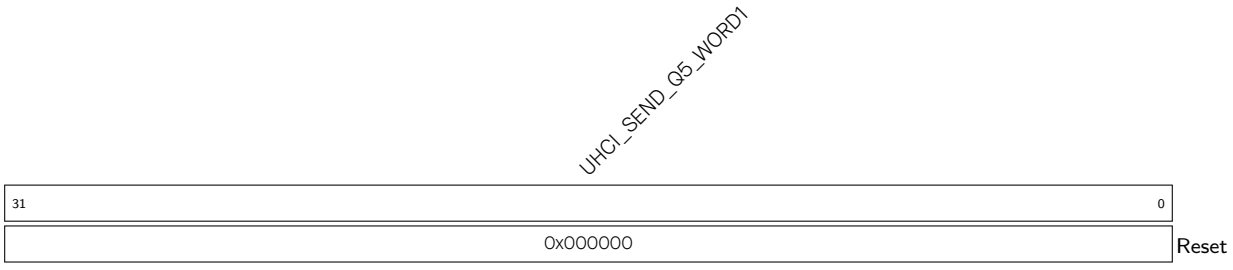
UHCI_SEND_Q4_WORD1 Data to be transmitted in Q4 register. (R/W)

Register 21.86. UHCI_REG_Q5_WORD0_REG (0x005C)



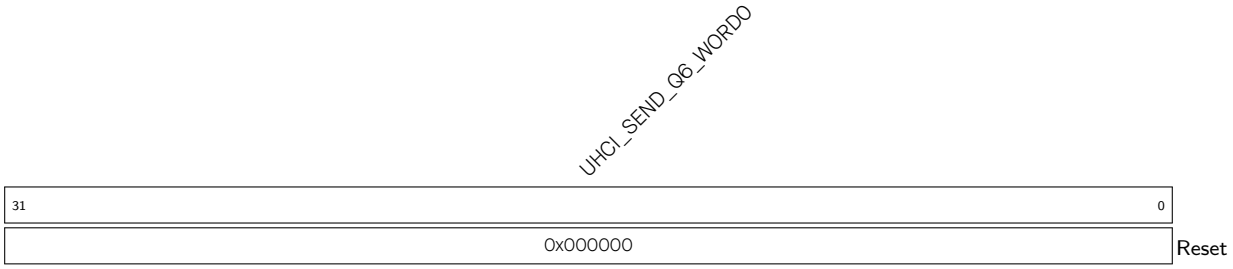
UHCI_SEND_Q5_WORD0 Data to be transmitted in Q5 register. (R/W)

Register 21.87. UHCI_REG_Q5_WORD1_REG (0x0060)



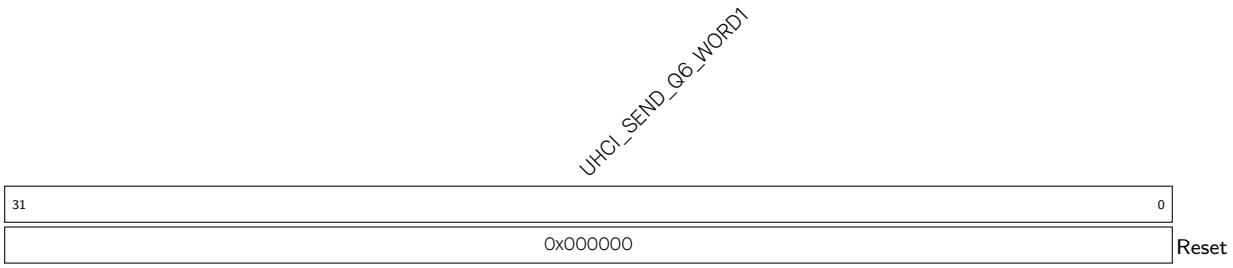
UHCI_SEND_Q5_WORD1 Data to be transmitted in Q5 register. (R/W)

Register 21.88. UHCI_REG_Q6_WORD0_REG (0x0064)



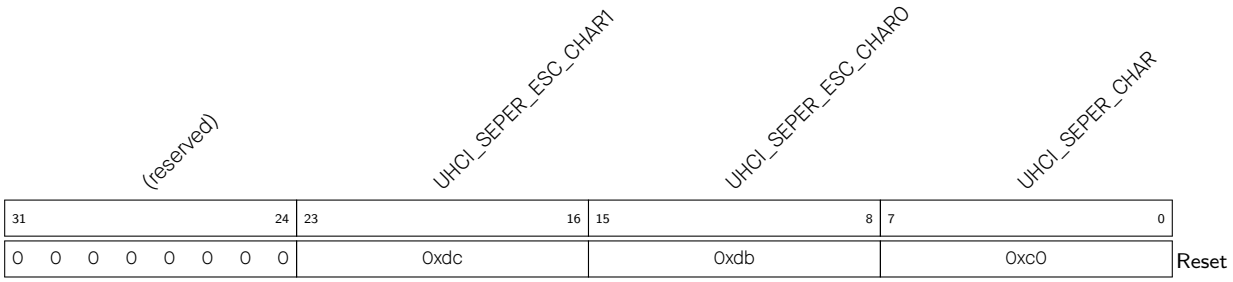
UHCI_SEND_Q6_WORD0 Data to be transmitted in Q6 register. (R/W)

Register 21.89. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Data to be transmitted in Q6 register. (R/W)

Register 21.90. UHCI_ESC_CONF0_REG (0x006C)

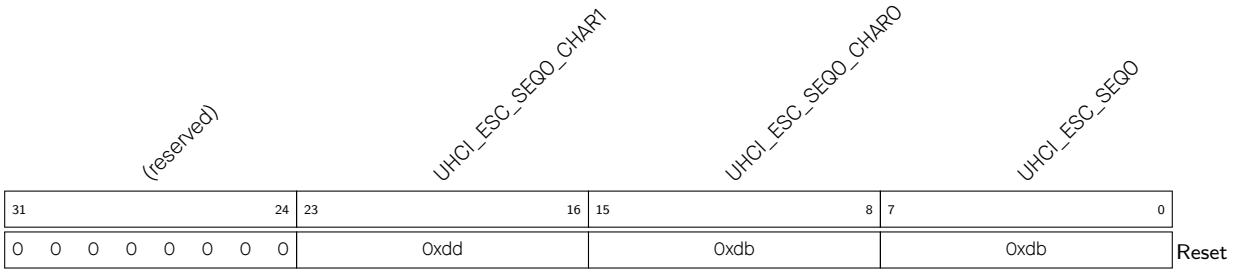


UHCI_SEPER_CHAR Configures separators to encode data packets. The default value is 0xC0. (R/W)

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 21.91. UHCI_ESC_CONF1_REG (0x0070)

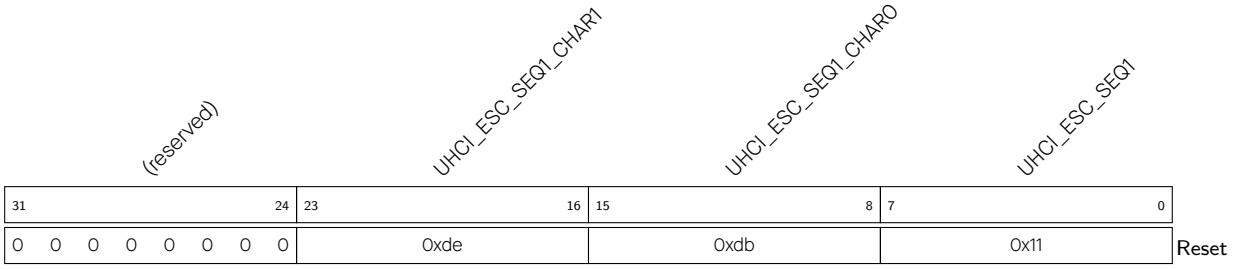


UHCI_ESC_SEQ0 Configures the character that needs to be encoded. The default value is 0xDB used as the first character of SLIP escape sequence. (R/W)

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 21.92. UHCI_ESC_CONF2_REG (0x0074)

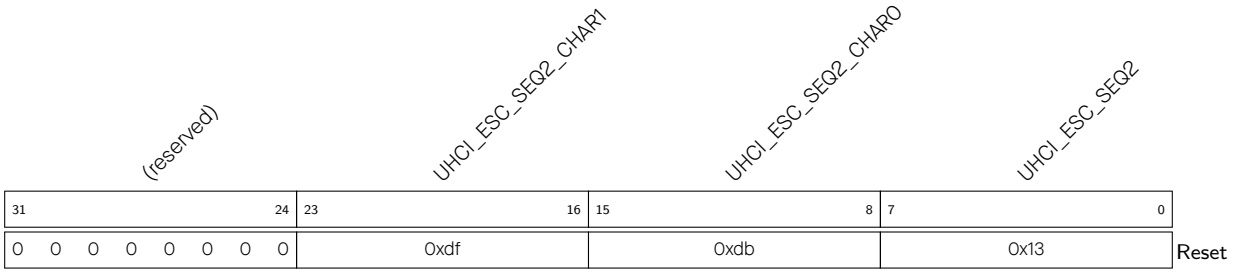


UHCI_ESC_SEQ1 Configures a character that need to be encoded. The default value is 0x11 used as a flow control character. (R/W)

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 21.93. UHCI_ESC_CONF3_REG (0x0078)

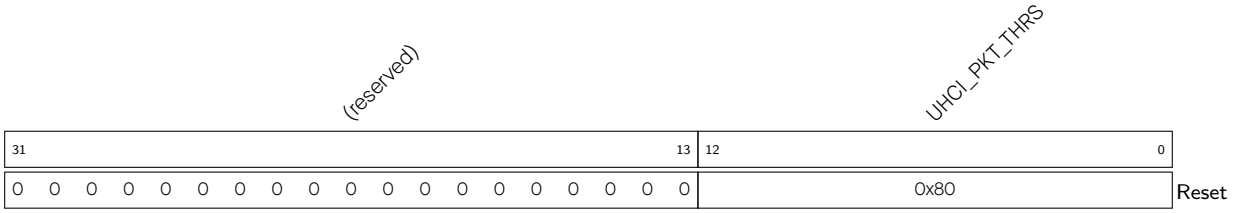


UHCI_ESC_SEQ2 Configures the character that needs to be decoded. The default value is 0x13 used as a flow control character. (R/W)

UHCI_ESC_SEQ2_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

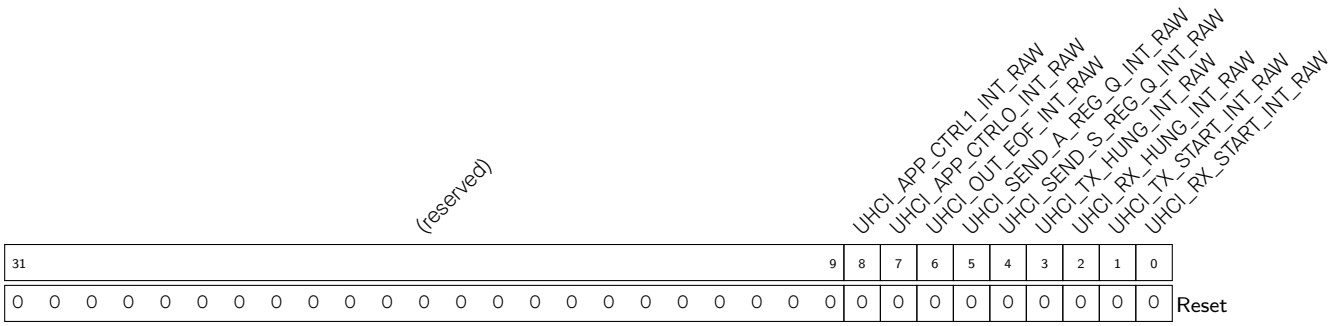
UHCI_ESC_SEQ2_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDF. (R/W)

Register 21.94. 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 21.95. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW The raw interrupt status of UHCI_RX_START_INT. (R/WTC/SS)

UHCI_TX_START_INT_RAW The raw interrupt status of UHCI_TX_START_INT. (R/WTC/SS)

UHCI_RX_HUNG_INT_RAW The raw interrupt status of UHCI_RX_HUNG_INT. (R/WTC/SS)

UHCI_TX_HUNG_INT_RAW The raw interrupt status of UHCI_TX_HUNG_INT. (R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_S_REG_Q_INT. (R/WTC/SS)

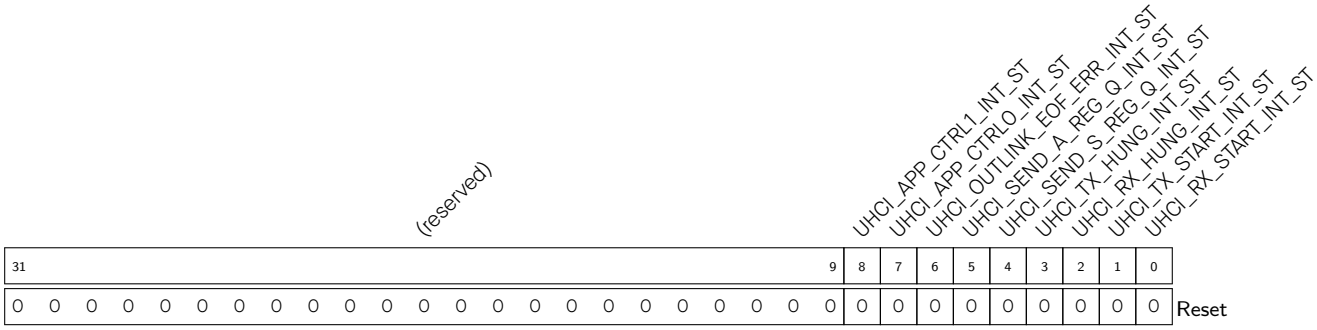
UHCI_SEND_A_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_A_REG_Q_INT. (R/WTC/SS)

UHCI_OUT_EOF_INT_RAW The raw interrupt status of UHCI_OUT_EOF_INT. (R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW The raw interrupt status of UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_RAW The raw interrupt status of UHCI_APP_CTRL1_INT. (R/W)

Register 21.96. 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)

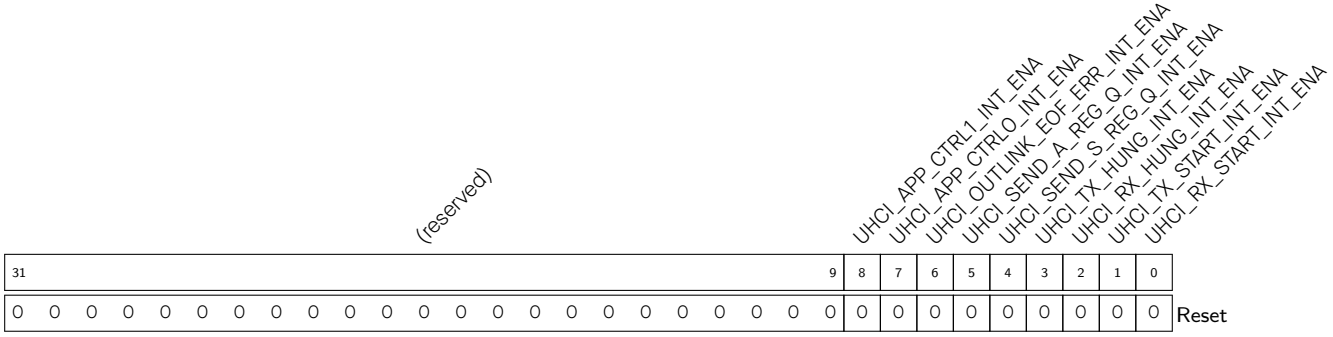
UHCI_SEND_A_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_A_REG_Q_INT. (RO)

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 21.97. 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)

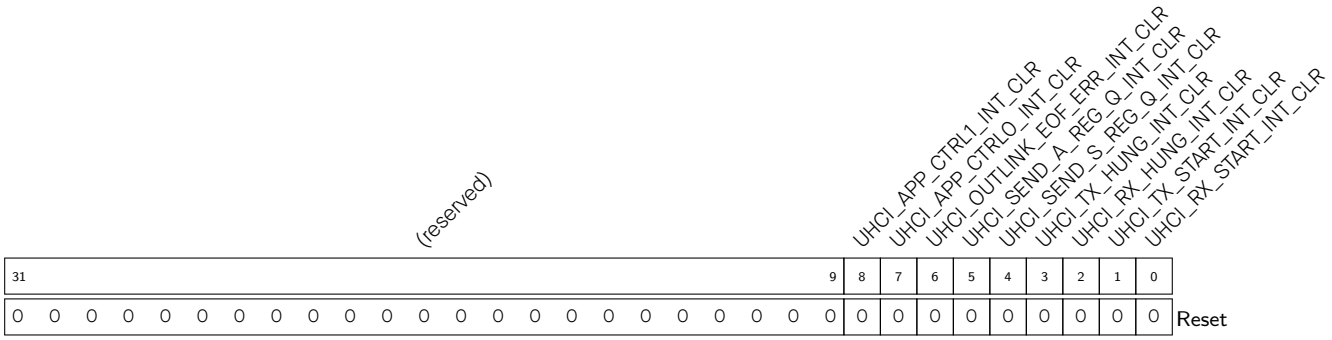
UHCI_SEND_A_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_A_REG_Q_INT. (R/W)

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 21.98. 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)

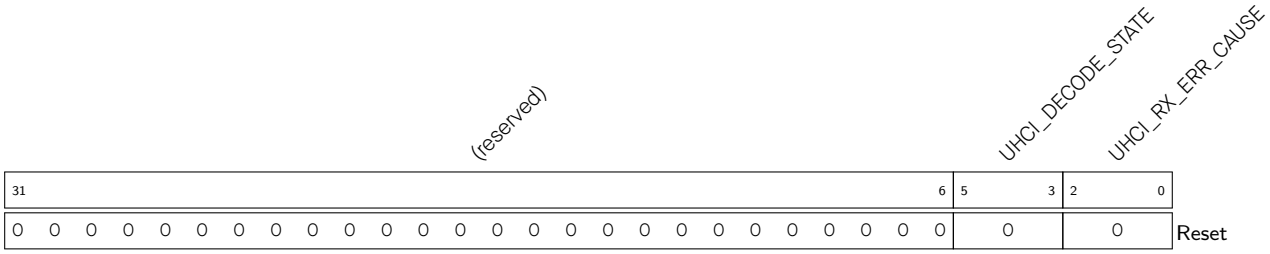
UHCI_SEND_A_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_A_REG_Q_INT. (WT)

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 21.99. 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

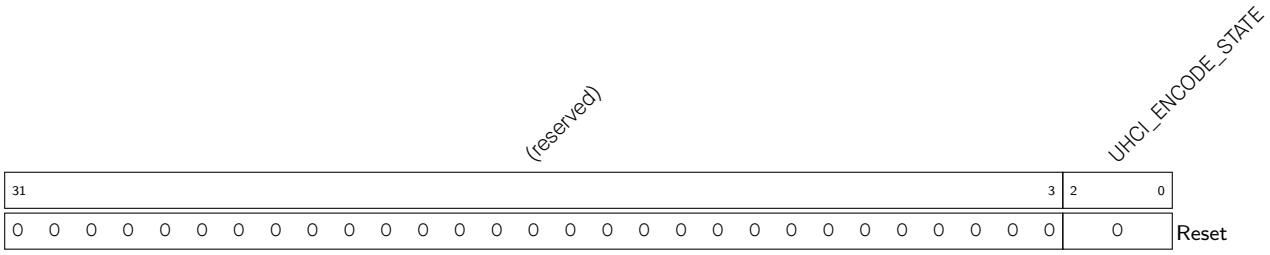
6: CRC check error

7: Invalid. No effect

(RO)

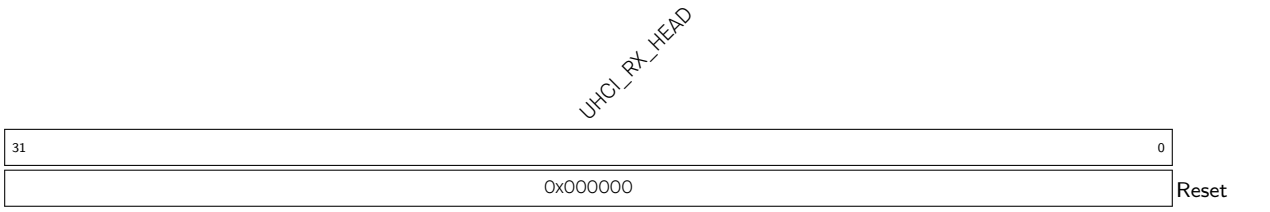
UHCI_DECODE_STATE Represents the UHCI decoder status. (RO)

Register 21.100. UHCI_STATE1_REG (0x001C)



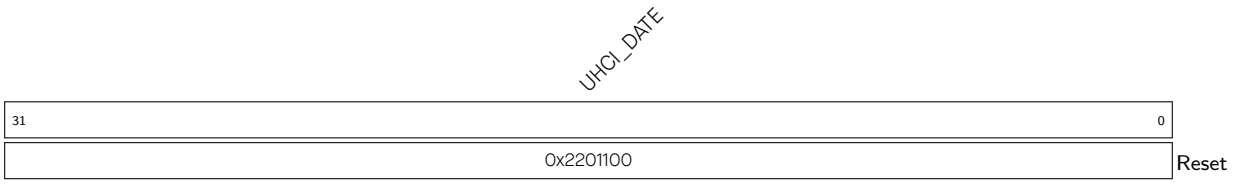
UHCI_ENCODE_STATE Represents the UHCI encoder status. (RO)

Register 21.101. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD Represents the header of the current received packet. (RO)

Register 21.102. UHCI_DATE_REG (0x0080)



UHCI_DATE Version control register. (R/W)

Chapter 22

USB Serial/JTAG Controller

ESP32-S31 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.

22.1 Overview

While programming and debugging an ESP32-S31 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-S31 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-S31.

22.2 Feature List

The USB Serial/JTAG controller has the following features:

- USB Full-speed device; Hardwired for CDC-ACM (Communication Device Class - Abstract Control Model) and JTAG adapter functionality
- CDC-ACM:
  - Integrates CDC-ACM adherent serial port emulation (plug-and-play on most modern OSes)
  - Supports host controllable chip reset and entry into download mode
- JTAG adapter functionality:
  - Allows fast communication with CPU debugging core using a compact representation of JTAG instructions
- A control endpoint, a dummy interrupt endpoint, two bulk input endpoints, and two bulk output endpoints; Up to 64-byte data payload size
- Internal PHY: very few or no external components needed to connect to a host computer

Note:
The SDIO Slave controller can be used together with the USB Serial/JTAG controller in single SPI mode, but not in quad

SPI mode.

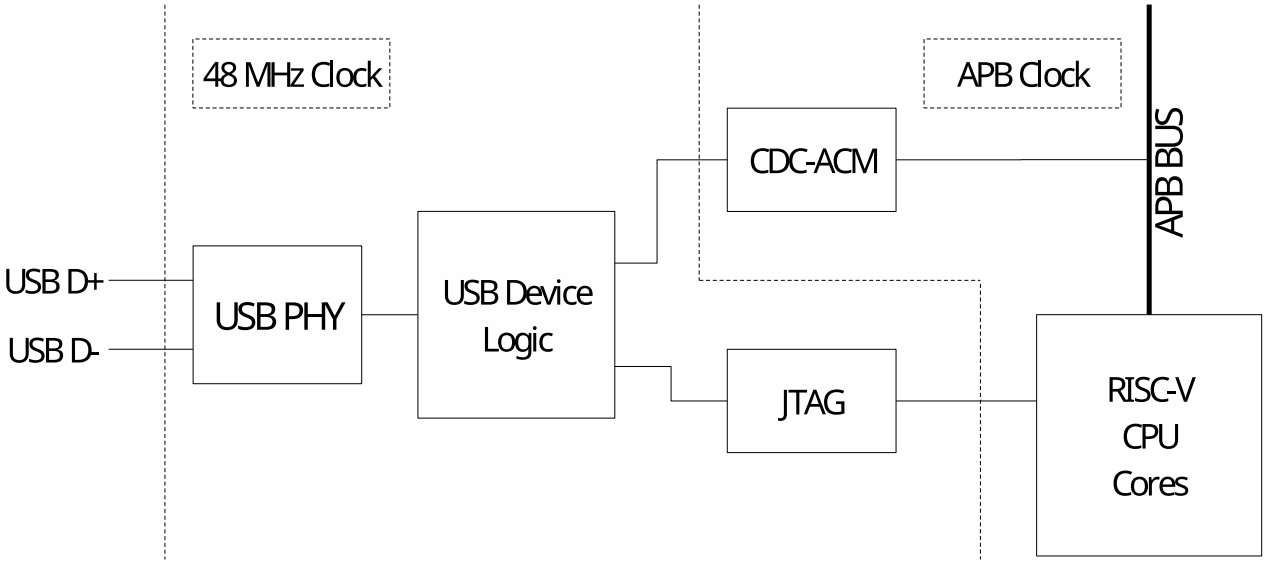


Figure 22.2-1. USB Serial/JTAG High Level Diagram

As shown in Figure 22.2-1, the USB Serial/JTAG controller consists of a USB PHY, a USB device interface, a JTAG command processor, a response capture unit, and the CDC-ACM registers. The PHY and device interface are clocked from a 48 MHz clock derived from the baseband PLL (BBPLL); the software-accessible side of the CDC-ACM block is clocked from APB_CLK. The JTAG command processor is connected to the JTAG debugging unit of the main processor; the CDC-ACM registers are connected to the APB bus and as such can be read from and written to by software running on the main CPU.

Note that while the USB Serial/JTAG device supports USB 2.0 standard, it only supports Full-speed (12 Mbps) mode but not other modes that the USB 2.0 standard introduced, e.g., the High-speed (480 Mbps) mode.

Figure 22.2-2 shows the internal details of the USB Serial/JTAG controller on the USB side. The USB Serial/JTAG controller consists of a USB 2.0 Full-speed device. It contains a control endpoint, a dummy interrupt endpoint, two bulk input endpoints, and two bulk output endpoints. Together, these form a USB composite device, which consists of a CDC-ACM USB class device as well as a vendor-specific device implementing the JTAG interface. On the SoC side, the JTAG interface is directly connected to the RISC-V CPU's debugging interface, allowing debugging of programs running on that core. Meanwhile, the CDC-ACM device is exposed as a set of registers, allowing a program on the CPU to read and write from it. Additionally, the ROM startup code of the SoC contains code that allows the user to reprogram attached flash memory using this interface.

Note:

To get specific information about what endpoint numbers are used for what functionality, please parse the relevant USB descriptors as returned by the device. This guarantees compatibility with other devices with the same functionality but a different implementation.

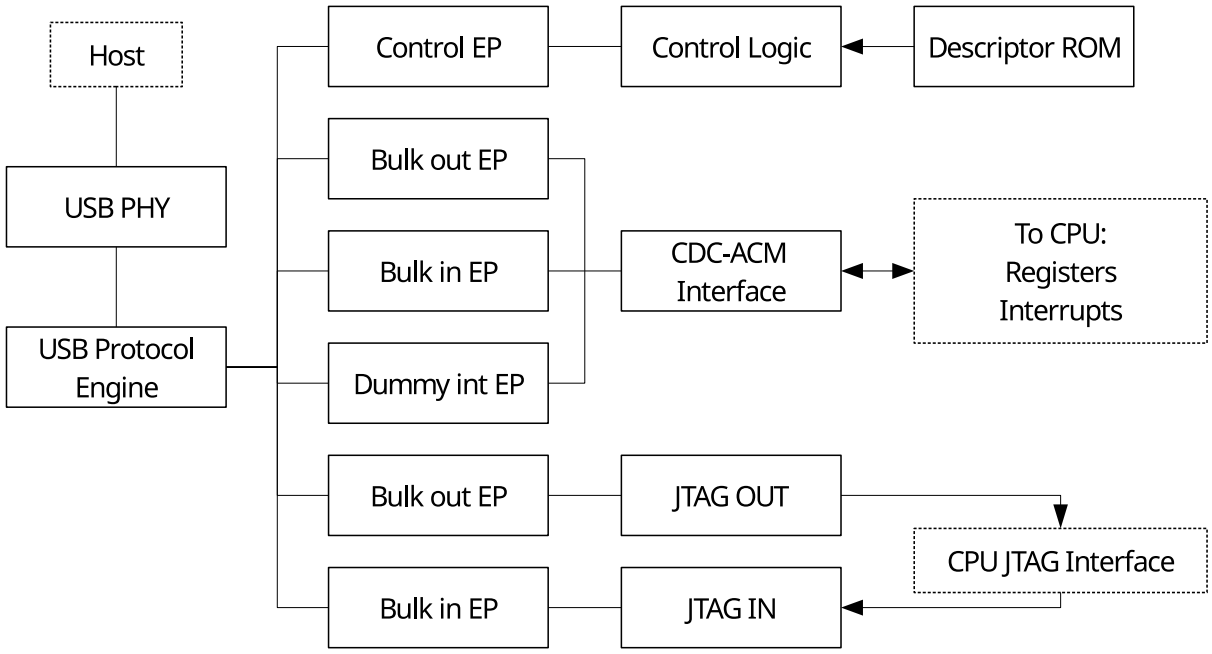


Figure 22.2-2. USB Serial/JTAG Block Diagram

22.3 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.

22.3.1 CDC-ACM USB Interface Functional Description

The CDC-ACM interface adheres to the standard USB CDC-ACM class for serial port emulation. It contains a dummy interrupt endpoint (which will never send any events, as they are not implemented nor needed) and a Bulk IN as well as a Bulk OUT endpoint for the host's received and sent serial data respectively. These endpoints can handle 64-bytes packet at a time, allowing high throughput. As CDC-ACM is a standard USB device class, a host generally can function without any special installation procedures. That is to say, when the USB debugging device is properly connected to a host, the operating system should show a new serial port moments later.

The CDC-ACM interface accepts the following standard CDC-ACM control requests:

Table 22.3-1. Standard CDC-ACM Control Requests

Command	Action
SEND_BREAK	Accepted but ignored (dummy)
SET_LINE_CODING	Accepted, value sent is readable in software
GET_LINE_CODING	By default, returns 9600 baud, no parity, 8 data bits, 1 stop bit (Can be changed through software)
SET_CONTROL_LINE_STATE	Set the state of the RTS/DTR lines. See Table 22.3-2

Aside from general-purpose communication, the CDC-ACM interface can also be used to reset ESP32-S31 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 22.3-2. 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-S31
1	1	No action

Note that if the download mode flag is set when ESP32-S31 is reset, ESP32-S31 will reboot into download mode. When this flag is cleared and the chip is reset, ESP32-S31 will boot from flash. For specific sequences, please refer to Section 22.5. All these functions can also be disabled by programming various eFuses. Please refer to Chapter *4 eFuse Controller (eFuse) [to be added later]* for more details.

22.3.2 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-S31. This is mainly used to read and write data from and to the virtual serial port on the attached host.

USB CDC-ACM serial data is sent to and received from the host in packets of 0 to 64 bytes in size. When enough CDC-ACM data has accumulated in the host, the host sends a packet to the CDC-ACM receive endpoint, and the USB Serial/JTAG controller accepts this packet if it has a free buffer. Conversely, the host checks periodically if the USB Serial/JTAG controller has a packet ready to be sent to the host, and if so, receives this packet.

Firmware can get notified of new data from the host in one of the following two ways. First of all, the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit will remain set as long as there still is unread host data in the buffer. Secondly, the availability of data will trigger the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT interrupt. When data is available, it can be read by firmware through repeatedly reading bytes from USB_SERIAL_JTAG_EP1_REG. The amount of bytes to read can be determined by checking the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit after reading each byte to see if there is more data to read. After all data is read, the USB device is automatically readied to receive a new data packet from the host.

Generally, after sending data, the attached host will block until the data is actually read by firmware in one of the described methods. However, it is possible that a given firmware program does not read data at all. This situation generally is not expected by host programs which are written to talk to a discrete USB-serial converter; the program e.g. stops executing until the data is read. The ESP32-S31 USB-serial-JTAG adapter can simulate the behavior of a discrete USB-serial-JTAG converter: by writing a timeout to USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX and writing an 1 to USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN, any unread data will be automatically flushed after the given timeout.

When the firmware has data to send, it can put the data in the send buffer and trigger a flush to allow the host to receive the data in a USB packet. In order to do so, there needs to be space available in the send buffer. Firmware can check this by reading USB_REG_SERIAL_IN_EP_DATA_FREE. A 1 in this register field indicates

there is still free room in the buffer, and firmware can fill the buffer by writing bytes to the USB_SERIAL_JTAG_EP1_REG register. Writing the buffer does not immediately trigger sending data to the host until the buffer is flushed. After the flush, the entire buffer will be ready to be received by the USB host at once. A flush can be triggered in two ways: 1) after the 64th byte is written to the buffer, the USB hardware will automatically flush the buffer to the host; or 2) firmware can trigger a flush by writing 1 to USB_REG_SERIAL_WR_DONE.

Regardless of how a flush is triggered, the send buffer will be unavailable for firmware to write into until it has been fully read by the host. As soon as the send buffer has been fully read, the USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT interrupt will be triggered, indicating that the send buffer can receive another 64 bytes.

It is possible to handle some out-of-band serial requests in software, specifically, the host setting DTR and RTS and changing the line state. If the CDC-ACM interface receives a SET_LINE_CODING request, the peripheral can be configured to trigger a USB_SERIAL_JTAG_SET_LINE_CODE_INT interrupt, at which point the line coding can be read from the USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG register. Similarly, SET_CONTROL_LINE_STATE requests will trigger USB_SERIAL_JTAG_RTS_CHG_INT and USB_SERIAL_JTAG_DTR_CHG_INT interrupts if they change the state of these lines. Software can then read the specific state through the USB_SERIAL_JTAG_RTS and USB_SERIAL_JTAG_DTR bits. Note that as described earlier, certain RTS/DTR sequences lead to hardware reset of ESP32-S31. 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.

22.3.3 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-S31.

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-S31. 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-S31 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 22.3-3. 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-S31.
- 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-S31.
- CMD_REP repeats the last (non-CMD_REP) command for a certain number of times. The purpose is to compress command streams which repeat the CMD_CLK instruction for multiple times. A command such as CMD_CLK can be followed by multiple CMD_REP commands. The number of repetitions done by one CMD_REP can be expressed as $repetition\_count = (R1 \times 2 + R0) \times (4^{cmd\_rep\_count})$, where cmd_rep_count indicates the number of the CMD_REP instruction that went directly before it. Note that the CMD_REP command is only intended to repeat a CMD_CLK command. Specifically, using it on a CMD_FLUSH command may lead to an unresponsive USB device, and a USB reset will be required to recover it.

22.3.4 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.

22.3.5 USB-to-JTAG Interface: Response Capture Unit

The response capture unit reads the TDO line of the internal JTAG bus and captures its value when the command parser executes a CMD_CLK with cap=1. It puts this bit into an internal shift register, and writes a byte into the USB buffer when 8 bits have been collected. Of these 8 bits, the least significant one is the one that is read from TDO the earliest.

As soon as either 64 bytes (512 bits) have been collected or a CMD_FLUSH command is executed, the response capture unit will make the buffer available for the host to receive. Note that the interface to the USB logic is double-buffered. Therefore, as long as the USB throughput is sufficient, the response capture unit can always receive more data. That is to say, while one of the buffers is waiting to be sent to the host, the other can receive more data. When the host has received data from its buffer and the response capture unit flushes its buffer, the two buffers exchange position.

This also means that a command stream can cause at most 128 bytes of capture data generated (less if there are flush commands in the stream) without the host acting to receive the generated data. If more data is generated anyway, the command stream will pause and the device will not accept more commands until the generated capture data is read out.

Note that in general, the logic of the response capture unit tries not to send zero-byte responses. For instance, sending a series of CMD_FLUSH commands will not cause a series of 0-byte USB responses to be sent. However, in the current implementation, some zero-0 responses may be generated in extraordinary circumstances. It is recommended to ignore these responses.

22.3.6 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 22.3-4. 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 22.3-5). 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-S31 are as follows. Note that all 16-bit values are little-endian.

Table 22.3-5. 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

22.4 Interrupts

ESP32-S31's USB Serial/JTAG Controller can generate the USB_SERIAL_JTAG_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from the USB Serial/JTAG Controller that can generate this interrupt signal.

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT: triggered when flush cmd is received for the JTAG bulk out endpoint.

- USB_SERIAL_JTAG_SOF_INT: triggered when SOF frame is received.
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT: triggered when Serial Port OUT Endpoint receives one packet.
- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT: triggered when Serial Port IN Endpoint is empty.
- USB_SERIAL_JTAG_PID_ERR_INT: triggered when PID error is detected.
- USB_SERIAL_JTAG_CRC5_ERR_INT: triggered when CRC5 error is detected.
- USB_SERIAL_JTAG_CRC16_ERR_INT: triggered when CRC16 error is detected.
- USB_SERIAL_JTAG_STUFF_ERR_INT: triggered when a bit stuffing error is detected.
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT: triggered when IN token for IN endpoint 1 is received.
- USB_SERIAL_JTAG_USB_BUS_RESET_INT: triggered when USB bus reset is detected.
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT: triggered when OUT endpoint 1 receives packet with zero payload.
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT: triggered when OUT endpoint 2 receives packet with zero payload.
- USB_SERIAL_JTAG_RTS_CHG_INT: triggered when level of RTS from USB serial channel is changed.
- USB_SERIAL_JTAG_DTR_CHG_INT: triggered when level of DTR from USB serial channel is changed.
- USB_SERIAL_JTAG_GET_LINE_CODE_INT: triggered when level of GET_LINE_CODING request is received.
- USB_SERIAL_JTAG_SET_LINE_CODE_INT: triggered when level of SET_LINE_CODING request is received.

22.5 Programming Procedures

Little setup is needed for using the USB Serial/JTAG device. The USB-to-JTAG hardware itself does not need any setup aside from the standard USB initialization that the host operating system already does. Apart from that, the CDC-ACM emulation on the host side is also plug-and-play.

On the firmware side, very little initialization is needed either. The USB hardware is self-initialized and after boot-up, if a host is connected and listening on the CDC-ACM interface, data can be exchanged as described above without any specific setup except for the situation when the firmware optionally sets up an interrupt service handler.

One thing to note is that there may be situations where either the host is not attached or the CDC-ACM virtual port is not opened. In such cases, the packets that are flushed to the host will never be picked up and the send buffer will never be empty. It is important to detect these situations and implement timeout, as this is the only way to reliably detect whether the port on the host side is closed or not.

Another thing to note is that the USB device is dependent on the BBPLL for the 48 MHz USB PHY clock. If this PLL is disabled, the USB communication will cease to function.

One scenario where this happens is Deep-sleep. The USB Serial/JTAG controller (as well as the attached RISC-V CPU) will be entirely powered down in Deep-sleep mode. If a device needs to be debugged in this

mode, it may be preferable to use an external JTAG debugger and a serial interface instead.

The CDC-ACM interface can also be used to reset the SoC and take it into or out of download mode. Generating the correct sequence of handshake signals can be a bit complicated, since most operating systems only allow setting or resetting DTR and RTS separately, but not in tandem. Additionally, some drivers (e.g., the standard CDC-ACM driver on Windows) do not set DTR until RTS is set and the user needs to explicitly set RTS in order to 'propagate' the DTR value. The recommended procedures are introduced below.

To reset the SoC into download mode:

Table 22.5-1. Reset SoC into Download Mode

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	Initialize to known values
Clear RTS	RTS=0, DTR=0	-
Set DTR	RTS=0, DTR=1	Set download mode flag
Clear RTS	RTS=0, DTR=1	Propagate DTR
Set RTS	RTS=1, DTR=1	-
Clear DTR	RTS=1, DTR=0	Reset SoC
Set RTS	RTS=1, DTR=0	Propagate DTR
Clear RTS	RTS=0, DTR=0	Clear download flag

To reset the SoC into booting from flash:

Table 22.5-2. Reset SoC into Booting from flash

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	-
Clear RTS	RTS=0, DTR=0	Clear download flag
Set RTS	RTS=1, DTR=0	Reset SoC
Clear RTS	RTS=0, DTR=0	Exit reset

22.6 Register Summary

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Registers			
USB_SERIAL_JTAG_EP1_REG	FIFO access for the CDC-ACM data IN and OUT endpoints	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	Configuration and control registers for the CDC-ACM FIFOs	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY hardware configuration	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	Registers used for debugging the PHY	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	Clock enable control	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	Memory power control	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM chip reset control	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	W0 of GET_LINE_CODING command	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	W1 of GET_LINE_CODING command	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	Configuration registers' value update	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	Serial AFIFO configure register	0x0064	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG	USB UART out endpoint timeout configuration	0x006C	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG	USB UART out endpoint timeout configuration	0x0070	R/W
Interrupt Registers			
USB_SERIAL_JTAG_INT_RAW_REG	Interrupt raw status register	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	Interrupt status register	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	Interrupt enable status register	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	Interrupt clear status register	0x0014	WT
Status Registers			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO status and control registers	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	Last received SOF frame index register	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	Control IN endpoint status information	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM IN endpoint status information	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM interrupt IN endpoint status information	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG IN endpoint status information	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	Control OUT endpoint status information	0x0038	RO

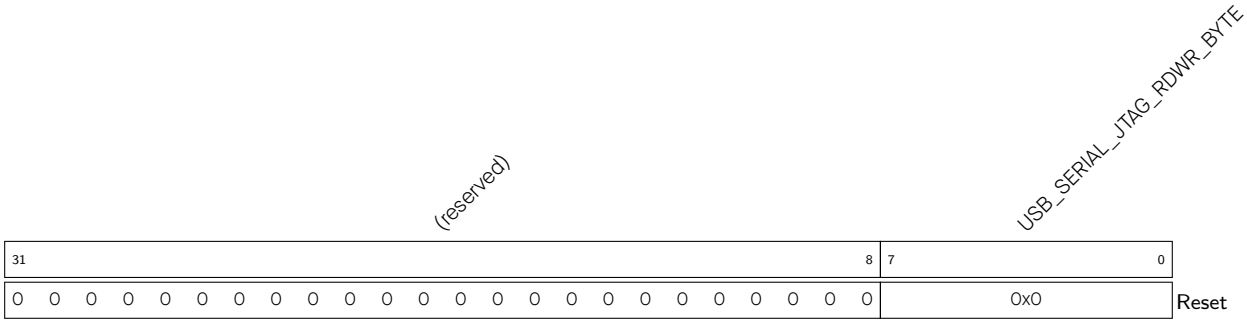
Name	Description	Address	Access
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM OUT endpoint status information	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG OUT endpoint status information	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	W0 of SET_LINE_CODING command	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	W1 of SET_LINE_CODING command	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB Bus reset status register	0x0068	RO
Version Registers			
USB_SERIAL_JTAG_DATE_REG	Date register	0x0080	R/W

22.7 Registers

The addresses in this section are relative to USB Serial/JTAG controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 22.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



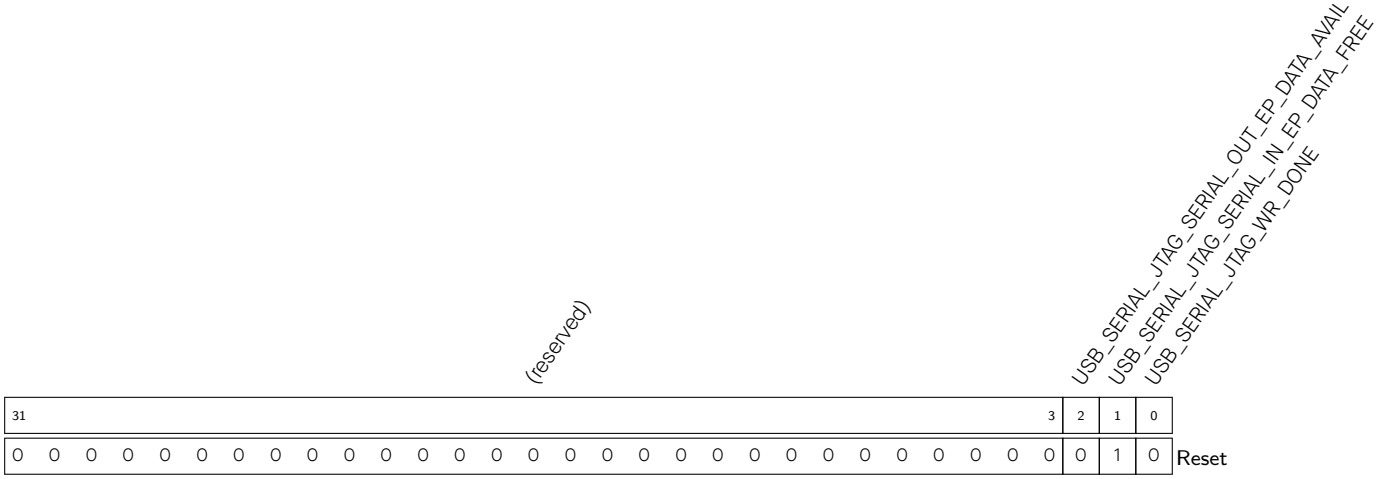
USB_SERIAL_JTAG_RDWR_BYTE A write to this register pushes the written data into the CDC TX FIFO; a read from this register pops a byte from the CDC RX FIFO and returns it.

When USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT is set, users can write data (up to 64 bytes) into CDC-ACM TX FIFO through this register.

When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is set, users can check how many data is received through USB_SERIAL_JTAG_OUT_EP1_WR_ADDR, then read data from CDC-ACM RX FIFO through this register.

(R/W)

Register 22.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE Configures whether to represent writing byte data to CDC-ACM TX FIFO is done.

0: No effect

1: Represents writing byte data to CDC-ACM TX FIFO is done

This bit then stays 0 until data in CDC-ACM TX FIFO is read by the USB Host.

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE Represents whether CDC-ACM TX FIFO has space available.

0: CDC-ACM TX FIFO is full and no data should be written into it

1: CDC-ACM TX FIFO is not full and data can be written into it

After writing USB_SERIAL_JTAG_WR_DONE, this bit will be 0 until data in CDC-ACM TX FIFO is read by USB Host.

(RO)

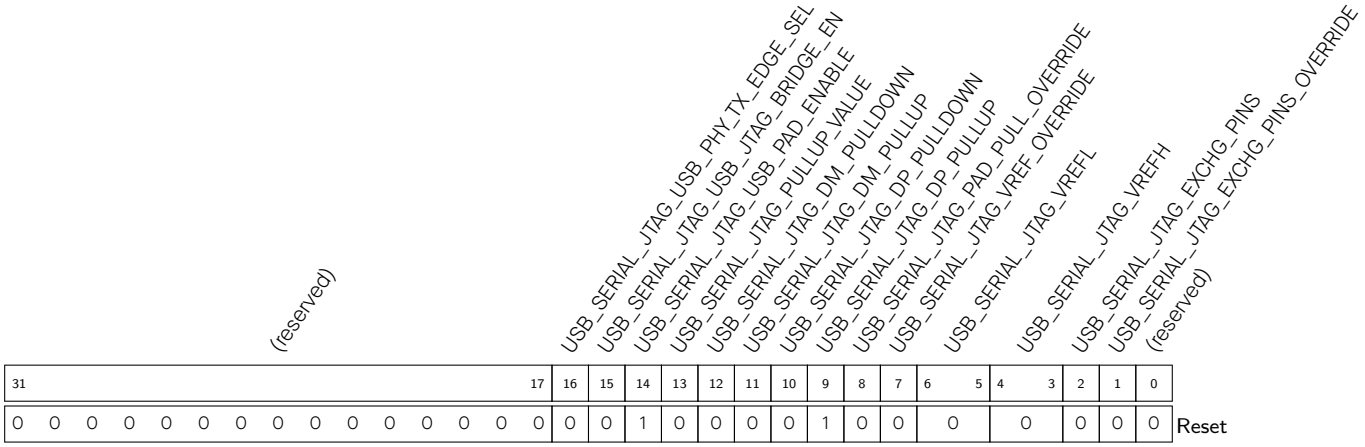
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL Represents whether there is data in CDC-ACM RX FIFO.

0: There is no data in CDC-ACM RX FIFO

1: There is data in CDC-ACM RX FIFO

(RO)

Register 22.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE Configures whether to enable software control USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS Configures whether to enable USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_VREFH Configures single-end input high threshold.

0: 1.76 V

1: 1.84 V

2: 1.92 V

3: 2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL Configures single-end input low threshold.

0: 0.80 V

1: 0.88 V

2: 0.96 V

3: 1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE Configures whether to enable software control input threshold.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 22.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE Configures whether to enable software to control USB D+ D- pullup and pulldown.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLUP Configures whether to enable USB D+ pull up when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN Configures whether to enable USB D+ pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DM_PULLUP Configures whether to enable USB D- pull up when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DM_PULLDOWN Configures whether to enable USB D- pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE Configures the pull up value when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: 2.2 K
1: 1.1 K
(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE Configures whether to enable USB pad function.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 22.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN Configures whether to disconnect USB_JTAG and internal JTAG.

0: USB_JTAG is connected to the internal JTAG port of CPU

1: USB_JTAG and the internal JTAG are disconnected, MTMS, MTDI, and MTCK are output through GPIO Matrix, and MTDO is input through GPIO Matrix

(R/W)

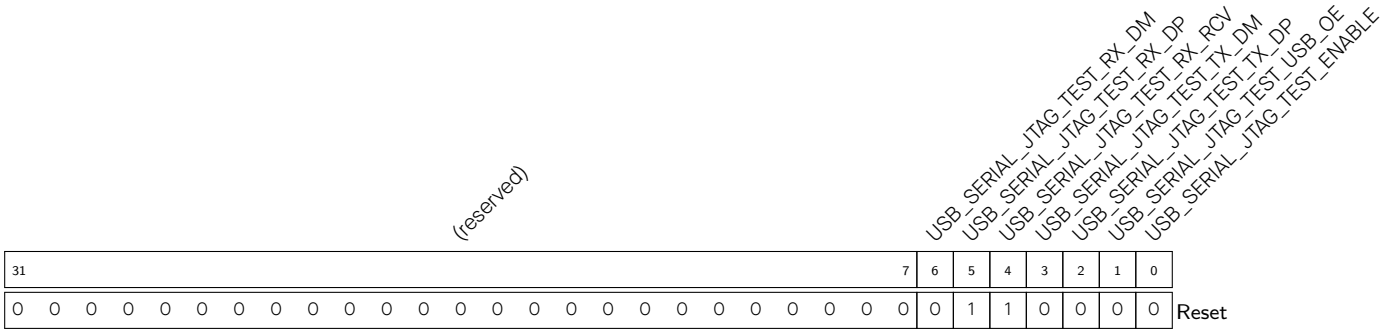
USB_SERIAL_JTAG_USB_PHY_TX_EDGE_SEL Configures the clock edge at which DP and DM signals are transmitted to the USB PHY.

0: Transmit on the falling edge of the clock

1: Transmit on the rising edge of the clock

(R/W)

Register 22.4. 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+

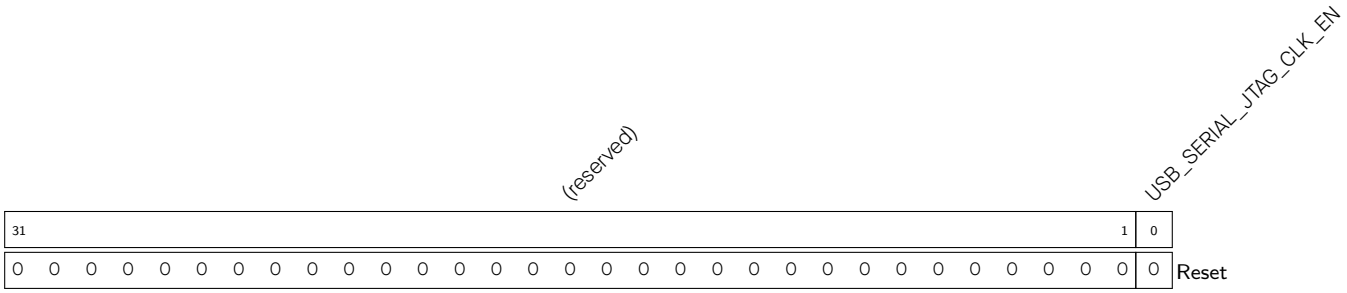
1: USB D+ voltage is higher than USB D-

(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 22.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



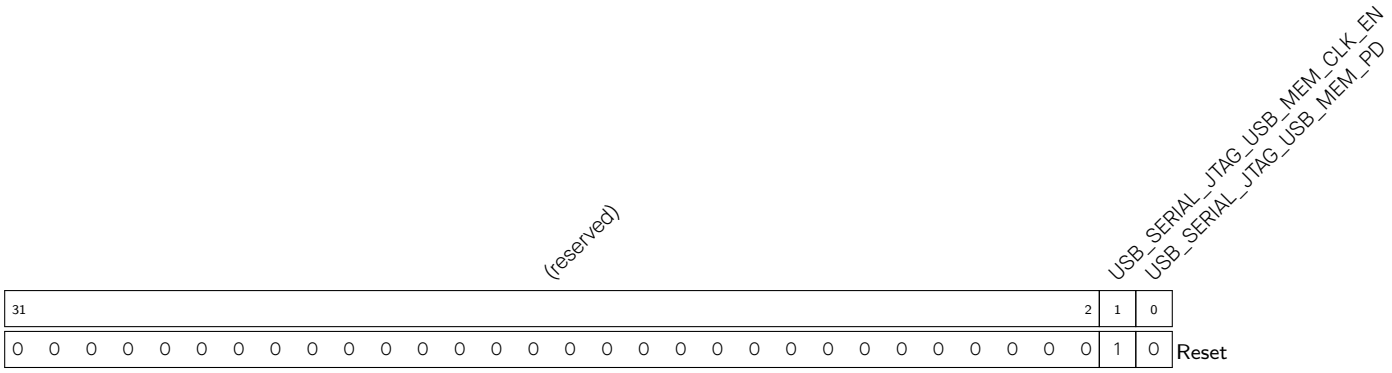
USB_SERIAL_JTAG_CLK_EN Configures whether to force clock on for register.

0: Support clock only when application writes registers

1: Force clock on for register

(R/W)

Register 22.6. 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)

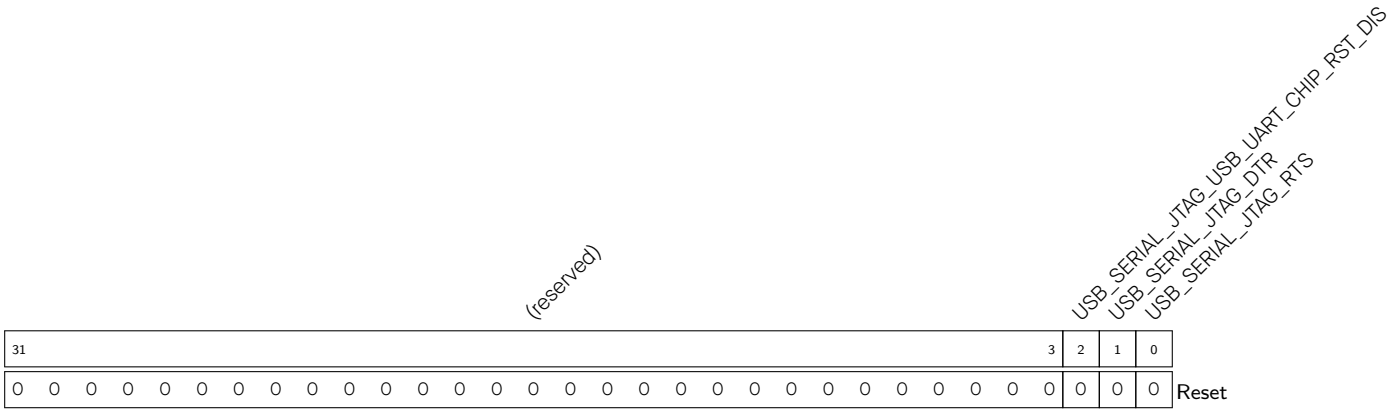
USB_SERIAL_JTAG_USB_MEM_CLK_EN Configures whether to force clock on for USB memory.

0: No effect

1: Force

(R/W)

Register 22.7. 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)

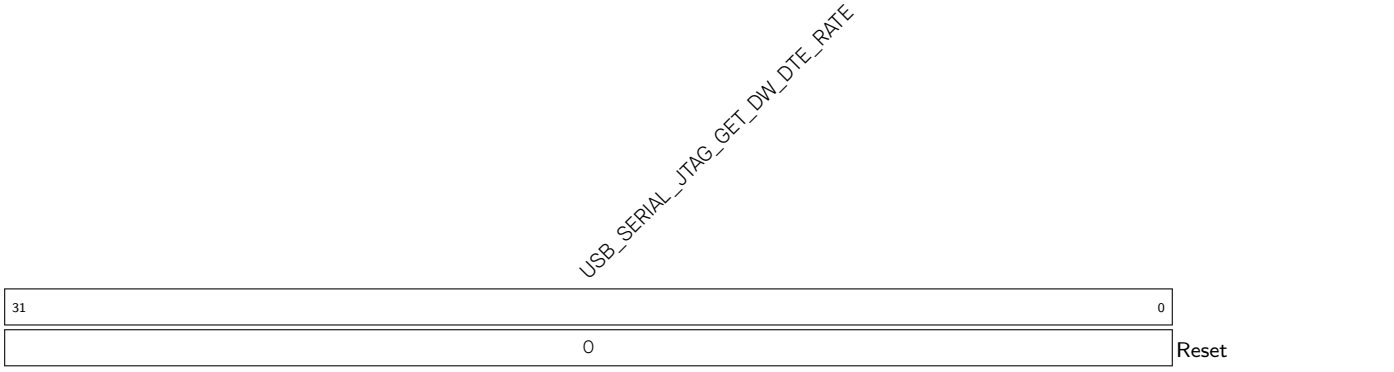
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS Configures whether to disable chip reset from USB serial channel.

0: No effect

1: Disable

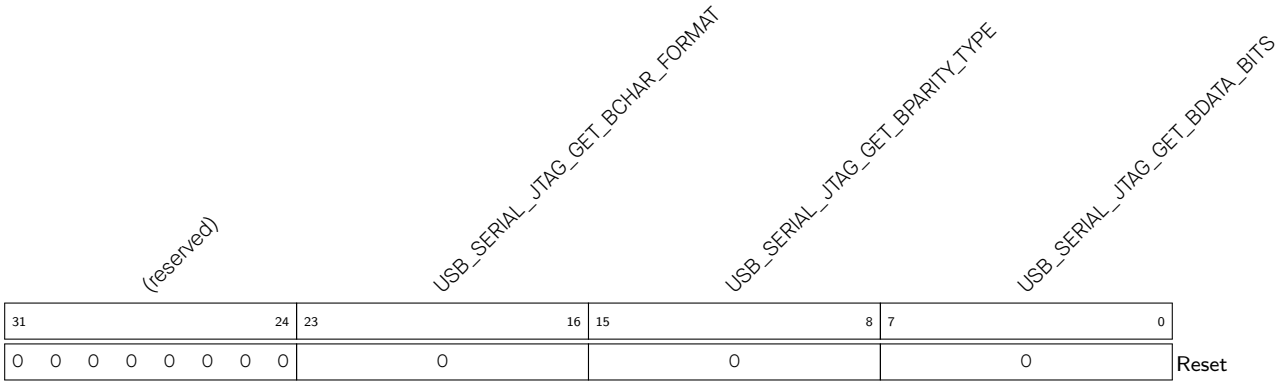
(R/W)

Register 22.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE Configures the value of dwDTERate set by software, which is requested by GET_LINE_CODING command. (R/W)

Register 22.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

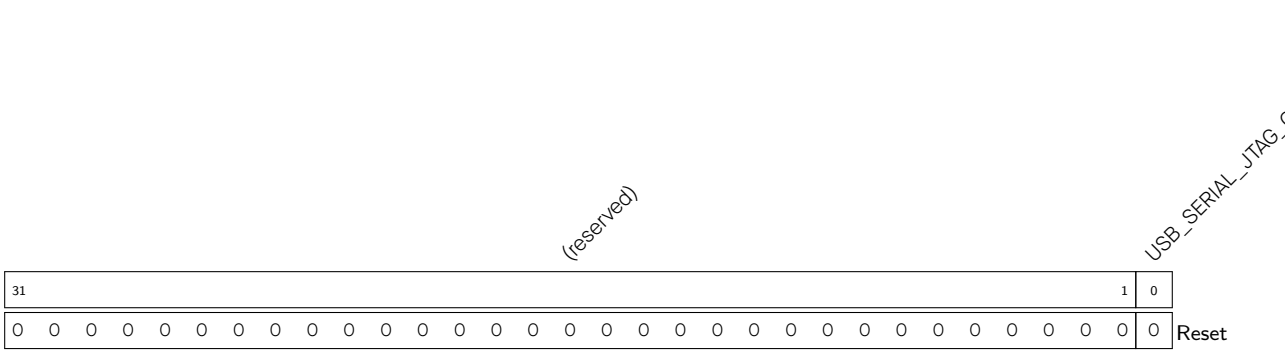


USB_SERIAL_JTAG_GET_BDATA_BITS Configures the value of bDataBits set by software, which is requested by GET_LINE_CODING command. (R/W)

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 22.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



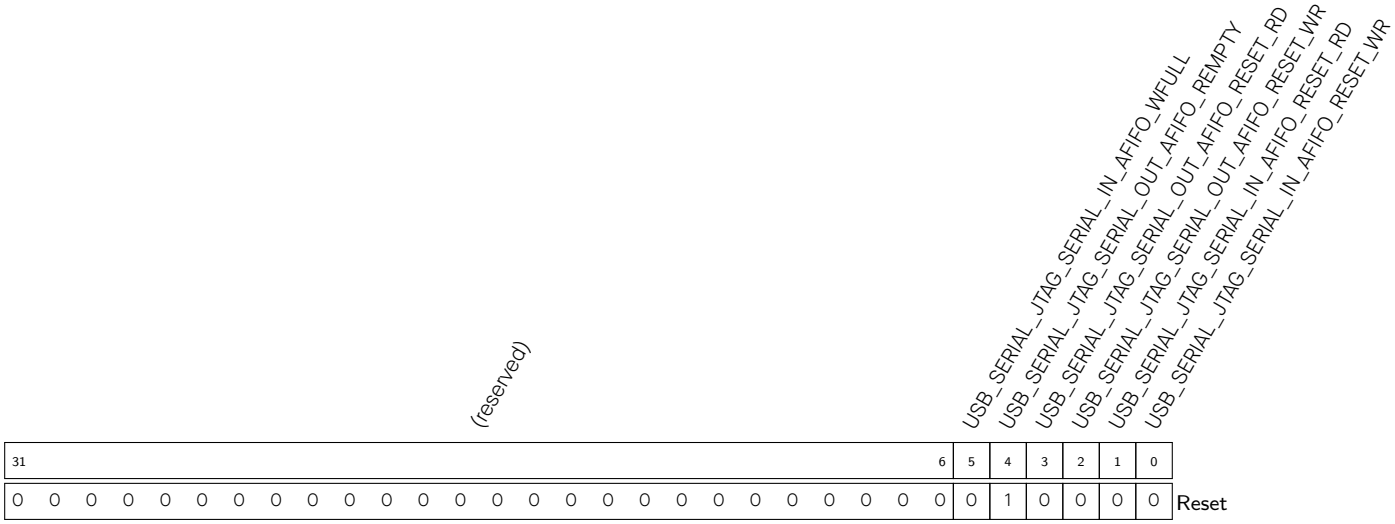
USB_SERIAL_JTAG_CONFIG_UPDATE Configures whether to update the value of configuration registers from APB clock domain to 48 MHz clock domain.

0: No effect

1: Update

(WT)

Register 22.11. 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

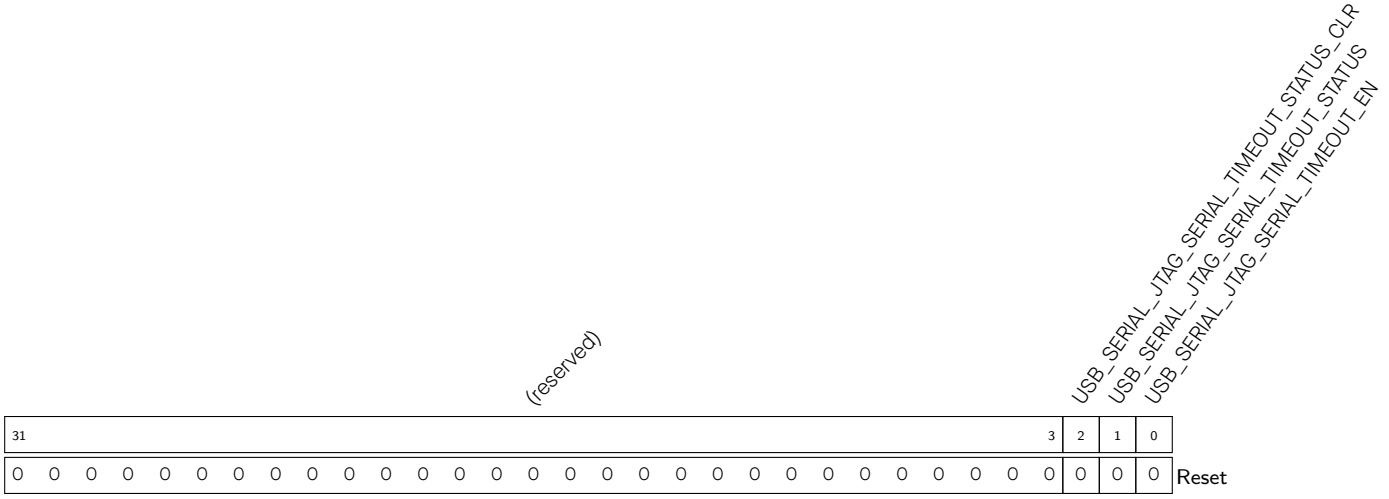
1: Reset

(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 22.12. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG (0x006C)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN USB serial out endpoint timeout enable. When a timeout event occurs, serial out endpoint buffer is automatically cleared and USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUSis asserted.

0: timeout disabled, buffer will not be cleared automatically

1: timeout enabled

(R/W)

USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS Timeout status bit

0: No timeout occurred.

1: Serial out endpoint has triggered a timeout event.

(R/WTC/SS)

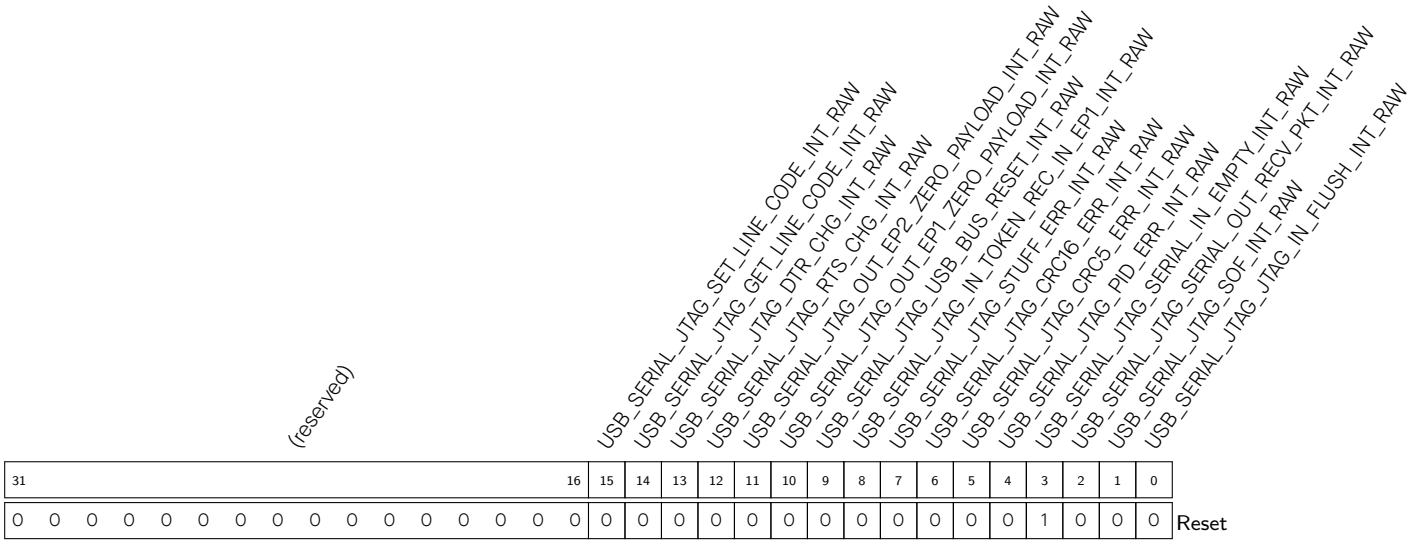
USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS. (WT)

Register 22.13. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG (0x0070)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX USB serial out endpoint timeout max threshold value, indicates the maximum time that waiting for chips to take away data in memory. This value is in steps of 20.83 ns. (R/W)

Register 22.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SOF_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/WTC/SS)

Continued on the next page...

Register 22.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

Continued from the previous page...

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/WTC/SS)

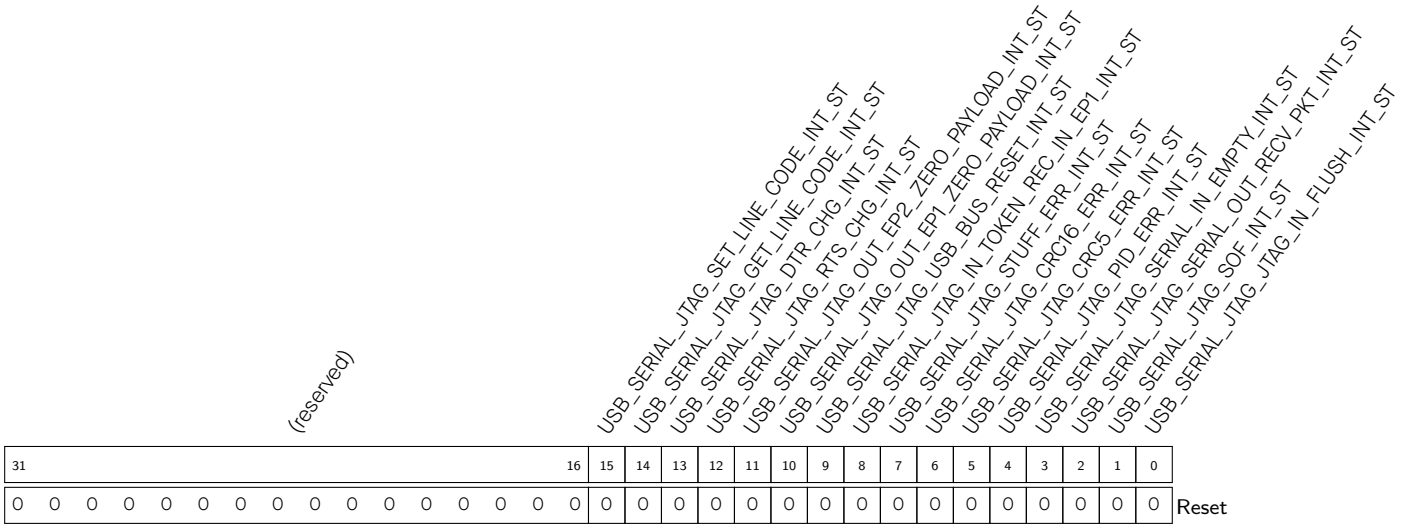
USB_SERIAL_JTAG_RTS_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/WTC/SS)

Register 22.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST The masked interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (RO)

USB_SERIAL_JTAG_SOF_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SOF_INT. (RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST The masked interrupt status of the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST The masked interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST The masked interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT (RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (RO)

Continued on the next page...

Register 22.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

Continued from the previous page...

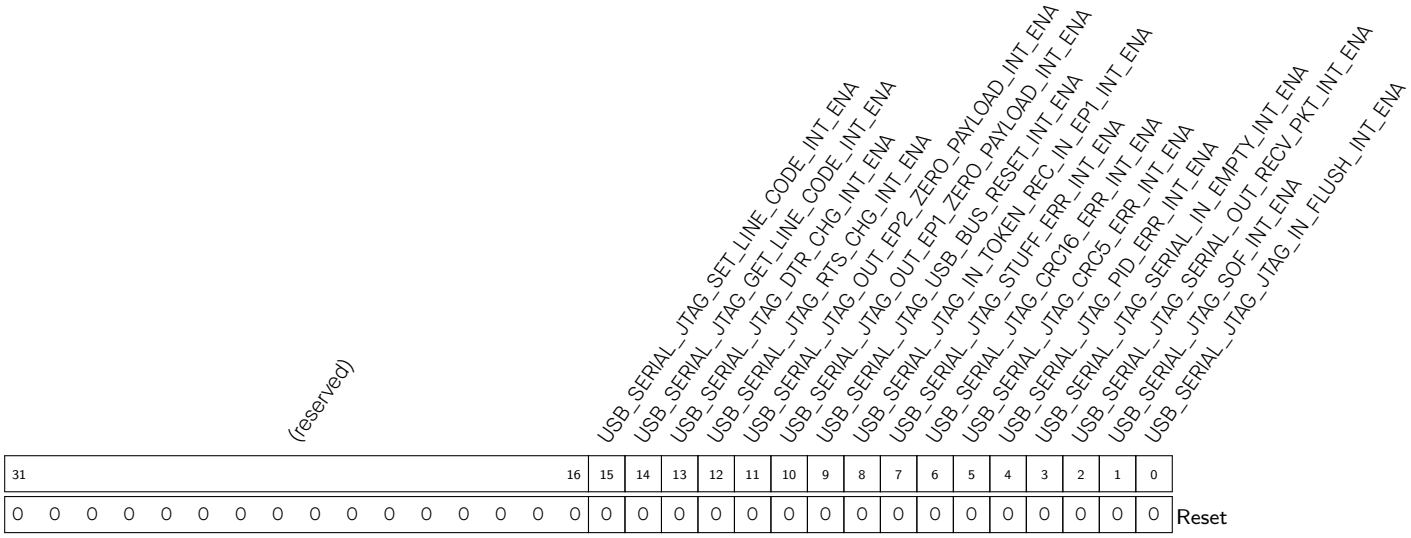
USB_SERIAL_JTAG_RTS_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (RO)

Register 22.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA Write 1 to enable USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/W)

USB_SERIAL_JTAG_SOF_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SOF_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_PID_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC5_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC16_ERR_INT. (R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_STUFF_ERR_INT. (R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA Write 1 to enable USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA Write 1 to enable USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/W)

Continued on the next page...

Register 22.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

Continued from the previous page...

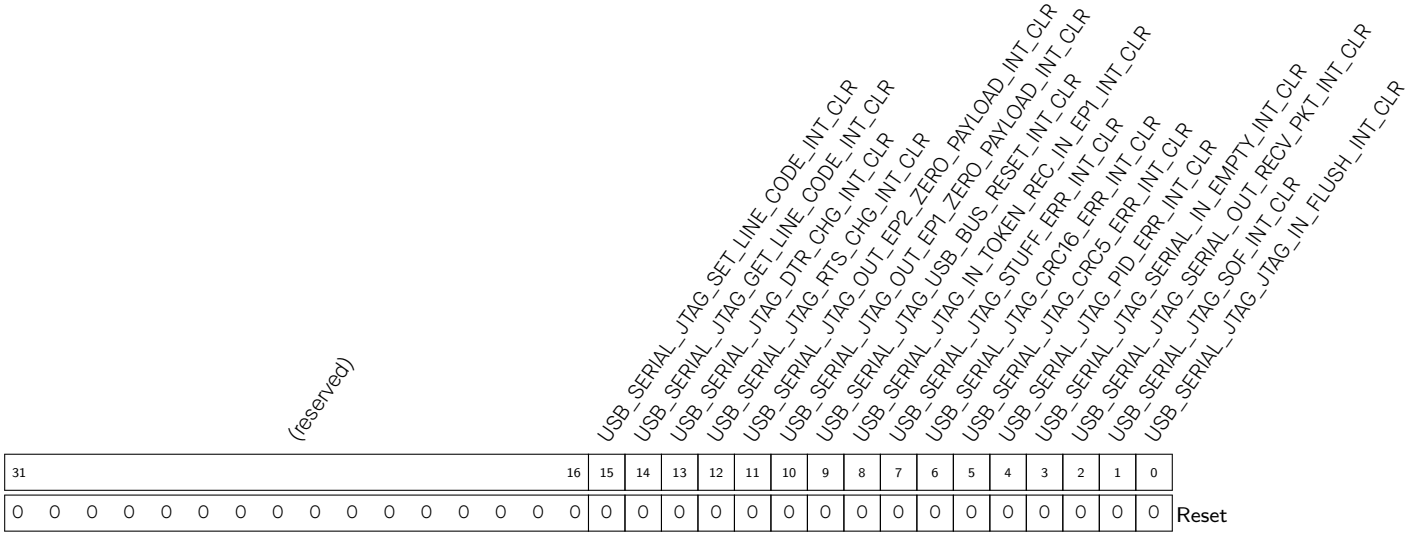
USB_SERIAL_JTAG_RTS_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_RTS_CHG_INT. (R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_DTR_CHG_INT. (R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/W)

Register 22.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR Write 1 to clear USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (WT)

USB_SERIAL_JTAG_SOF_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SOF_INT. (WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_PID_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC5_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC16_ERR_INT. (WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_STUFF_ERR_INT. (WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR Write 1 to clear USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR Write 1 to clear USB_SERIAL_JTAG_USB_BUS_RESET_INT. (WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (WT)

Continued on the next page...

Register 22.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

Continued from the previous page...

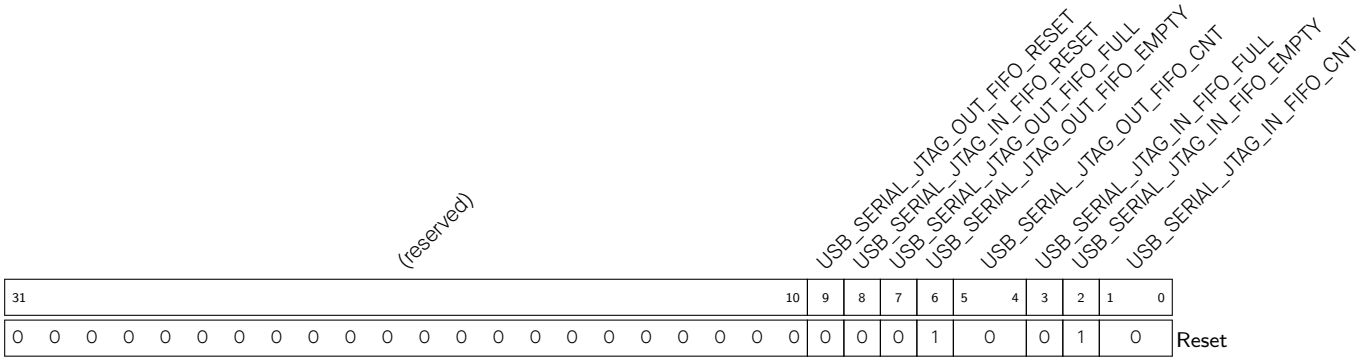
USB_SERIAL_JTAG_RTS_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_RTS_CHG_INT. (WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_DTR_CHG_INT. (WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_GET_LINE_CODE_INT. (WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SET_LINE_CODE_INT. (WT)

Register 22.18. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT Represents JTAG IN FIFO counter. (RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY Represents whether JTAG IN FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL Represents whether JTAG IN FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT Represents JTAG OUT FIFO counter. (RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY Represents whether JTAG OUT FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL Represents whether JTAG OUT FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET Configures whether to reset JTAG IN FIFO.

0: No effect

1: Reset

(R/W)

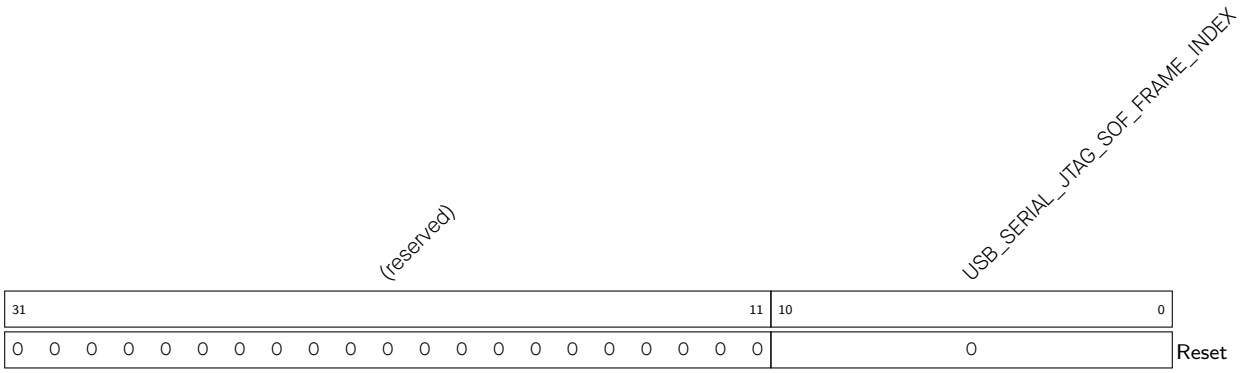
USB_SERIAL_JTAG_OUT_FIFO_RESET Configures whether to reset JTAG OUT FIFO.

0: No effect

1: Reset

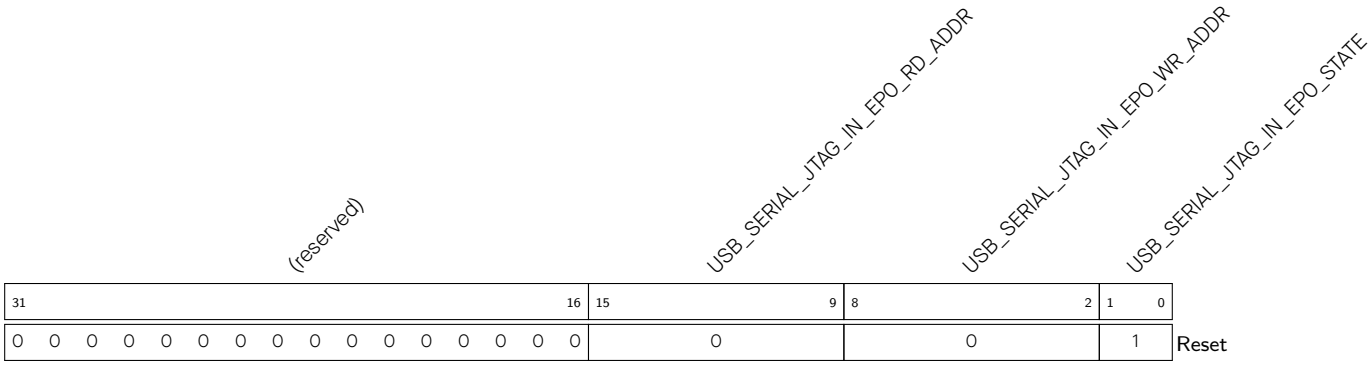
(R/W)

Register 22.19. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX Represents frame index of received SOF frame. (RO)

Register 22.20. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

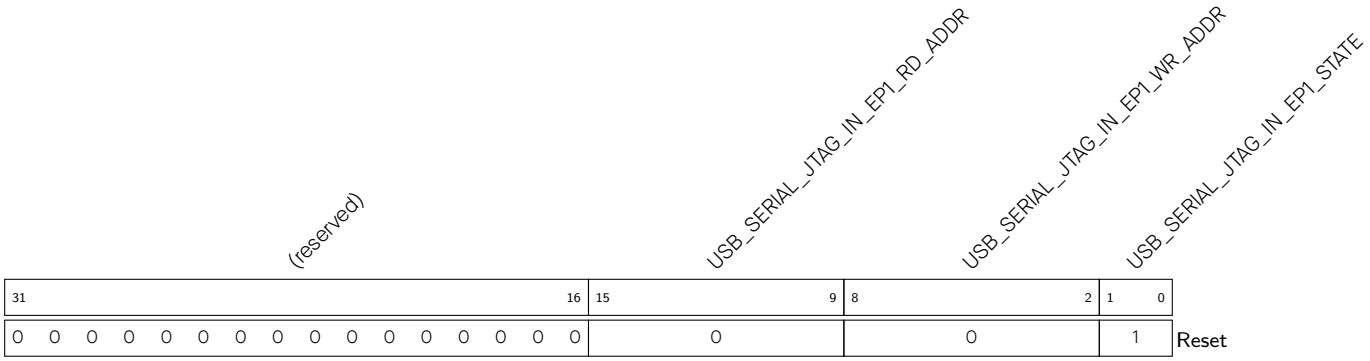


USB_SERIAL_JTAG_IN_EP0_STATE Represents state of IN Endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR Represents write data address of IN endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR Represents read data address of IN endpoint 0. (RO)

Register 22.21. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

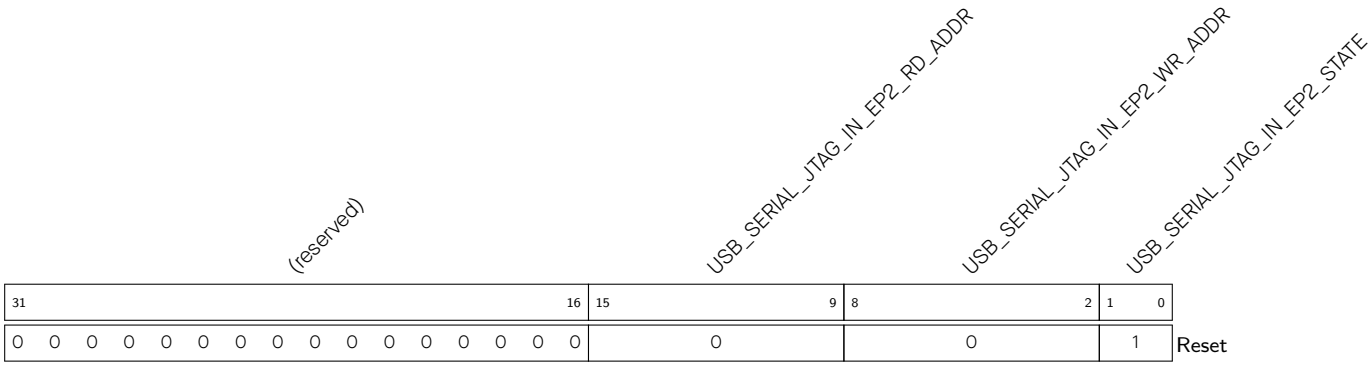


USB_SERIAL_JTAG_IN_EP1_STATE Represents state of IN Endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR Represents write data address of IN endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR Represents read data address of IN endpoint 1. (RO)

Register 22.22. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

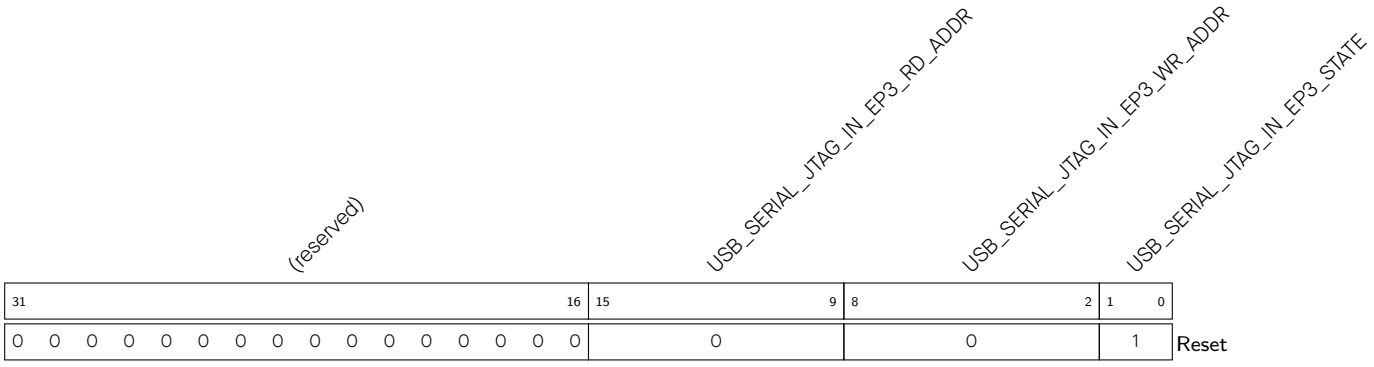


USB_SERIAL_JTAG_IN_EP2_STATE Represents state of IN Endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR Represents write data address of IN endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR Represents read data address of IN endpoint 2. (RO)

Register 22.23. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

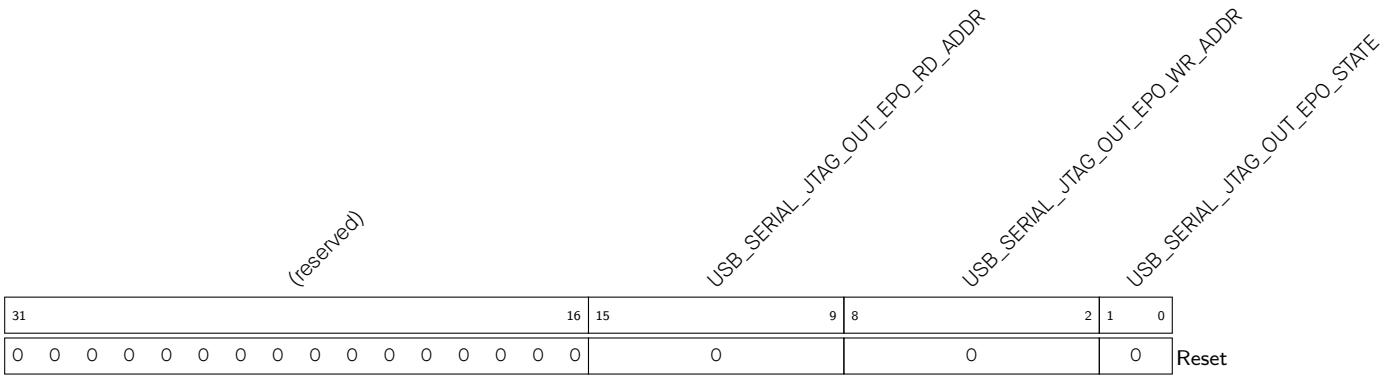


USB_SERIAL_JTAG_IN_EP3_STATE Represents state of IN Endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR Represents write data address of IN endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR Represents read data address of IN endpoint 3. (RO)

Register 22.24. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)



USB_SERIAL_JTAG_OUT_EP0_STATE Represents state of OUT Endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR Represents write data address of OUT endpoint 0. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) bytes data in OUT endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR Represents read data address of OUT endpoint 0. (RO)

Register 22.25. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



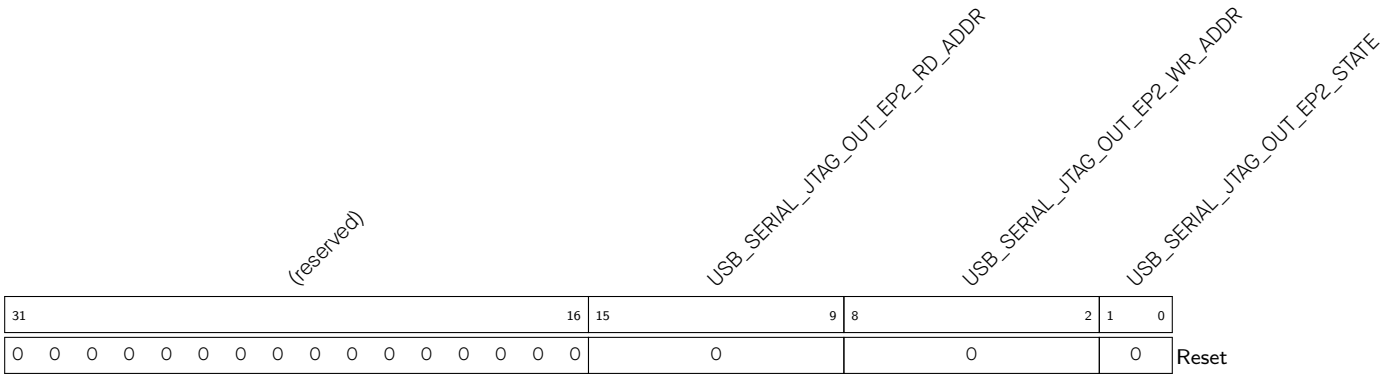
USB_SERIAL_JTAG_OUT_EP1_STATE Represents state of OUT Endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR Represents write data address of OUT endpoint 1. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR − 2) bytes data in OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR Represents read data address of OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT Represents data count in OUT endpoint 1 when one packet is received. (RO)

Register 22.26. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)

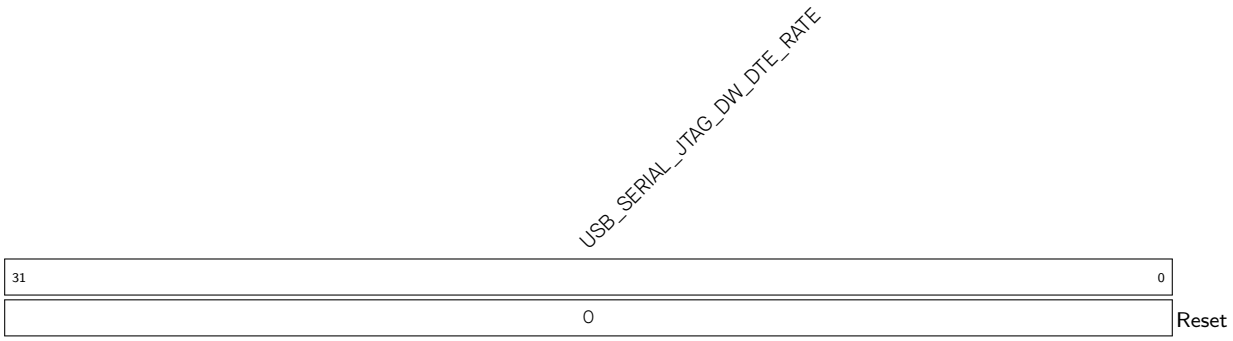


USB_SERIAL_JTAG_OUT_EP2_STATE Represents state of OUT Endpoint 2. (RO)

USB_SERIAL_JTAG_OUT_EP2_WR_ADDR Represents write data address of OUT endpoint 2. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected there are (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR − 2) bytes data in OUT endpoint 2. (RO)

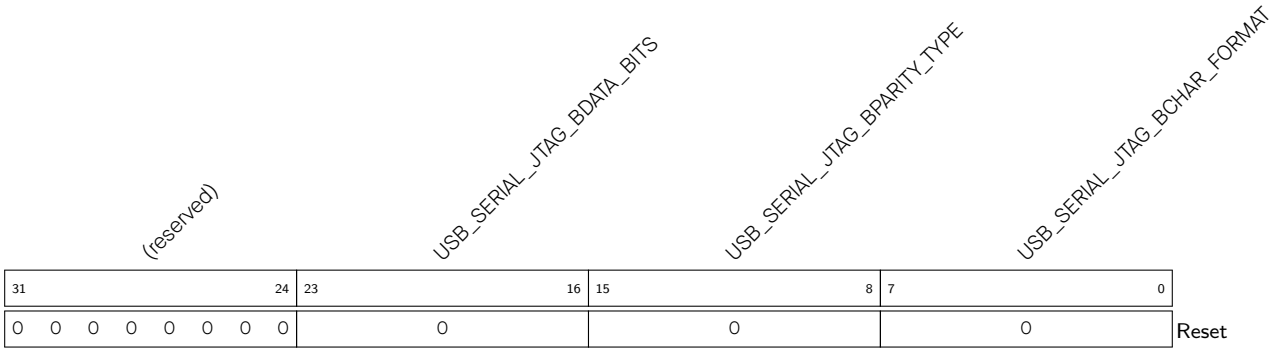
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR Represents read data address of OUT endpoint 2. (RO)

Register 22.27. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE Represents the value of dwDTERate set by host through SET_LINE_CODING command. (RO)

Register 22.28. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

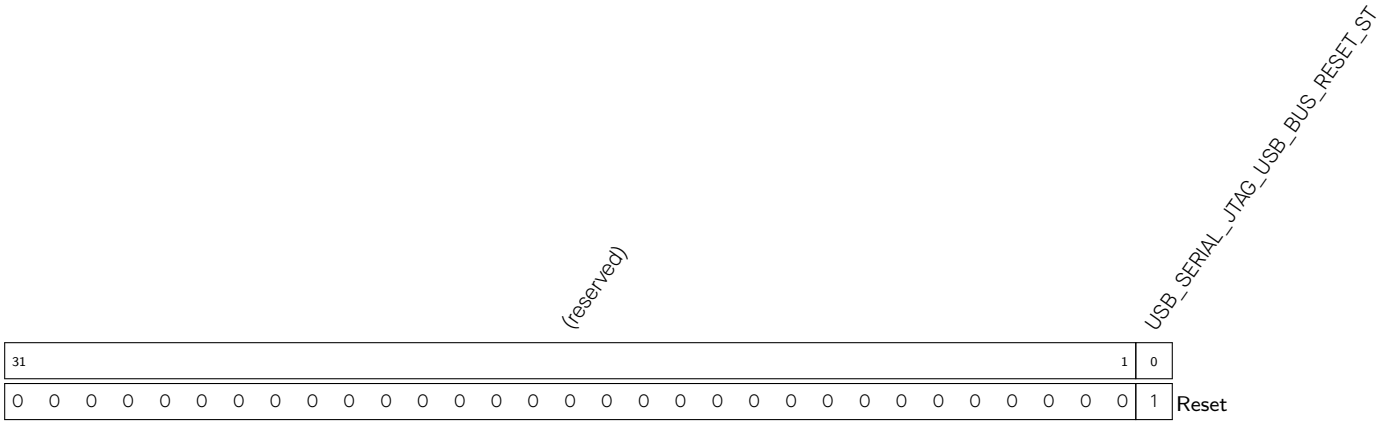


USB_SERIAL_JTAG_BCHAR_FORMAT Represents the value of bCharFormat set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BPARITY_TYPE Represents the value of bParityTpye set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BDATA_BITS Represents the value of bDataBits set by host through SET_LINE_CODING command. (RO)

Register 22.29. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



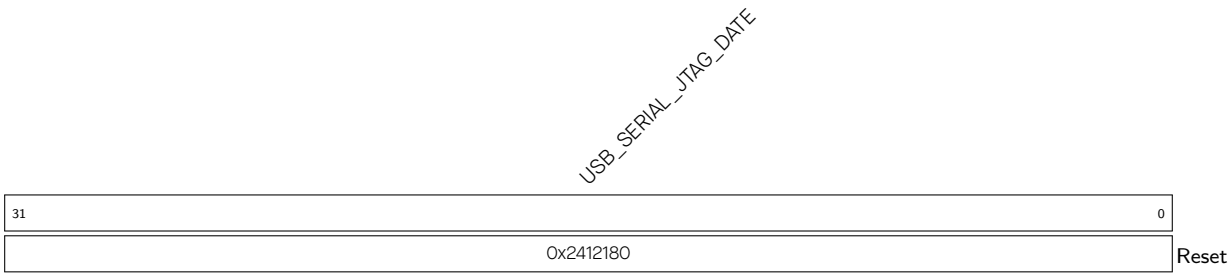
USB_SERIAL_JTAG_USB_BUS_RESET_ST Represents whether USB bus reset is released.

0: USB Serial/JTAG is in USB bus reset status

1: USB bus reset is released

(RO)

Register 22.30. USB_SERIAL_JTAG_DATE_REG (0x0080)



USB_SERIAL_JTAG_DATE Version control register. (R/W)

Chapter 23

LED PWM Controller (LEDC)

23.1 Overview

The LED PWM Controller is a peripheral designed to generate pulse-width modulation (PWM) signals for LED control. It has specialized features such as automatic duty cycle fading, and can also be used to generate PWM signals for other purposes.

23.2 Features

The LED PWM Controller has the following features:

- Two independent LED PWM controllers (LEDC0, LEDC1), each containing 8 independent PWM generation channels (16 channels in total)
- Maximum PWM duty cycle resolution: 20 bits
- Four independent timers per LED PWM controller, supporting fractional division
- Adjustable phase of PWM signal output
- PWM duty cycle dithering
- Automatic duty cycle fading – gradual increase or decrease of a PWM's duty cycle without processor intervention. An interrupt is 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, the step size, the number of fades, and the fading frequency
- PWM signal output in low-power mode (Light-sleep mode)
- Event generation and task response managed by the Event Task Matrix (ETM) peripheral

The two independent LED PWM controllers are identical. The following sections describe one of them as an example.

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the eight PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 7).

23.3 Architectural Overview

Figure 23.3-1 shows the architecture of the LED PWM Controller.

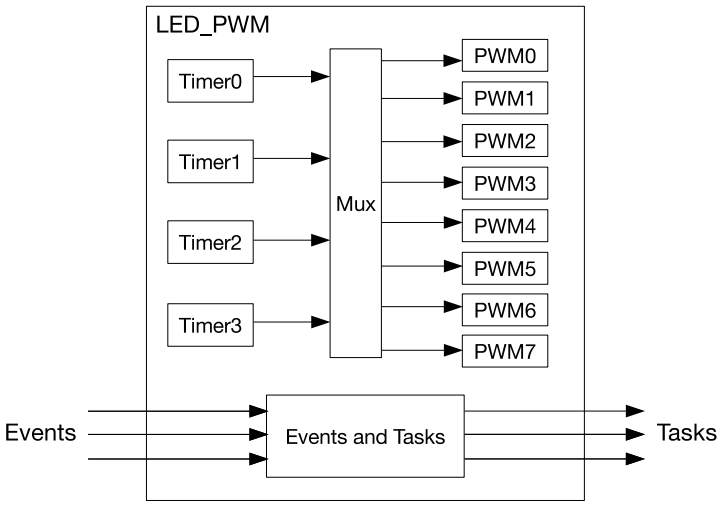


Figure 23.3-1. LED PWM Architecture

Each of the four timers has an internal timebase counter (i.e., a counter that counts on cycles of a reference clock) and thus can be independently configured (i.e., configurable clock divider, and counter overflow value). Each PWM generator selects one of the four timers by configuring the LEDC_TIMER_SEL_CH*n*, and uses the timer's counter value timer*x*_cnt as a reference to generate its PWM signal.

Figure 23.3-2 illustrates the main functional blocks of the timer and the PWM generator.

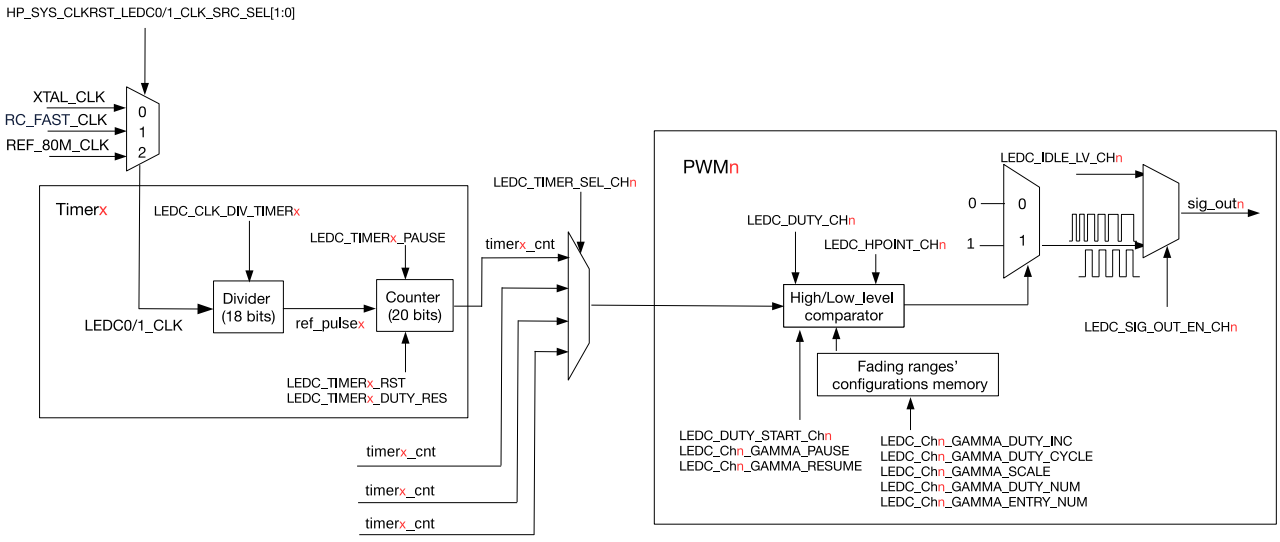


Figure 23.3-2. Timer and PWM Generator Block Diagram

23.4 Functional Description

23.4.1 Clock

The LED PWM Controller uses two types of clocks:

- APB_CLK: bus clock used to configure LED PWM registers
- LEDC0/1_CLK: function clock for the LED PWM Controller to operate

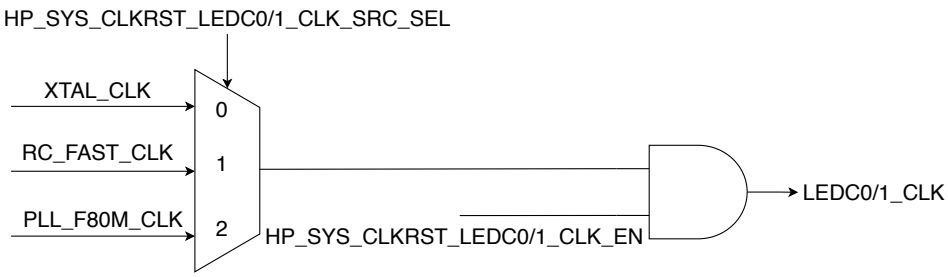


Figure 23.4-1. LEDC_CLK Diagram

As shown in 23.4-1 *LEDC_CLK Diagram*, the function clock LEDC_CLK can be sourced from:

- XTAL_CLK
- RC_FAST_CLK
- PLL_F80M_CLK

For more information on clocks, see Chapter 7 *Reset and Clock [to be added later]*.

For detailed clock programming procedures, see Section 23.7 *Programming Procedures*.

23.4.2 Reset

The LED PWM Controller can be fully reset by setting the HP_SYS_CLKRST_LEDC0_RST_EN or HP_SYS_CLKRST_LEDC1_RST_EN fields in the HP_SYS_CLKRST_LEDC_CTRL0_REG register.

23.4.3 Timers

Each timer in the LED PWM Controller internally maintains a timebase counter. Referring to Figure 23.3-2, the clock signal used by the timebase counter is named ref_pulse*x*. All timers use the same clock source, LEDC_CLK, which is then passed through a clock divider to generate ref_pulse*x* for the counter. Each timer can be independently enabled by setting the corresponding LEDC_TIMER*x*_POWER_UP field, and disabled by clearing the corresponding LEDC_TIMER*x*_POWER_UP field.

23.4.3.1 Clock Divider Configuration

The LEDC_CLK signal is passed through a clock divider to generate the ref_pulse*x* signal for the counter. The frequency of ref_pulse*x* is equal to the frequency of LEDC_CLK divided by the divisor LEDC_CLK_DIV (see Figure 23.3-2).

The clock divider is a fractional divider. Thus, the divisor LEDC_CLK_DIV can be a non-integer value. LEDC_CLK_DIV is configured according to the following equation.

$$\text{LEDC\_CLK\_DIV} = A + \frac{B}{256}$$

- The integer part *A* corresponds to the most significant 10 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[22:13])
- The fractional part *B* corresponds to the least significant 8 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[12:5])

When the fractional part *B* is 0, LEDC_CLK_DIV is equivalent to an integer divisor (i.e., an integer prescaler). In other words, a ref_pulse*x* clock pulse is generated after every *A* number of LEDC_CLK clock pulses.

However, when *B* is not 0, LEDC_CLK_DIV becomes a non-integer divisor. The clock divider implements non-integer frequency division by alternating between *A* and (*A*+1) LEDC_CLK clock pulses per ref_pulse*x* clock pulse. In this way, the average frequency of ref_pulse*x* clock pulse will be the desired frequency (i.e., the non-integer divided frequency). For every 256 ref_pulse*x* clock pulses:

- A number of *B* ref_pulse*x* clock pulses are generated every (*A*+1) LEDC_CLK clock pulses
- A number of (256-*B*) ref_pulse*x* clock pulses are generated every *A* LEDC_CLK clock pulses
- The ref_pulse*x* clock pulses generated every (*A*+1) pulses are evenly distributed amongst those generated every *A* pulses

Figure 23.4-2 illustrates the relation between LEDC_CLK clock pulses and ref_pulse*x* clock pulses when LEDC_CLK_DIV is a non-integer value.

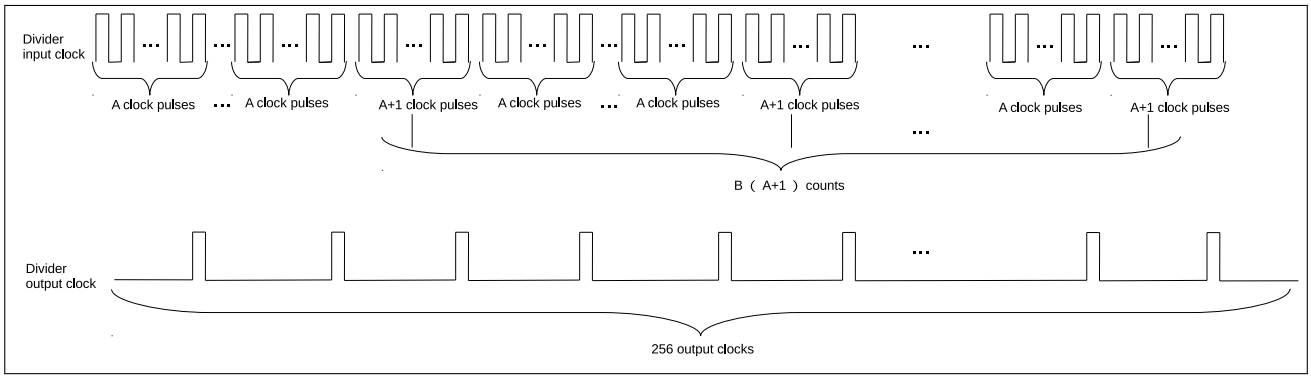


Figure 23.4-2. Frequency Division When LEDC_CLK_DIV is a Non-Integer Value

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.

23.4.3.2 20-Bit Counter

Each timer contains a 20-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 23.3-2). The LEDC_TIMER*x*_DUTY_RES field configures the actual used bit width of this 20-bit counter. Hence, the maximum resolution of the PWM signal is 20 bits. The counter counts up to $2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} - 1$, overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software. Figure 23.4-3 shows the relationship between the counter and PWM resolution.

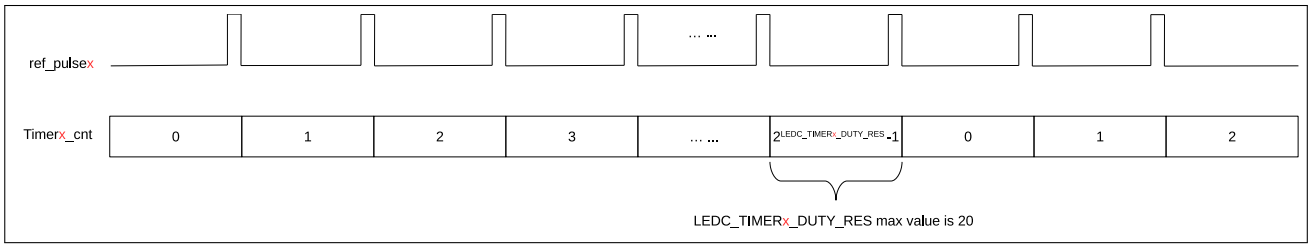


Figure 23.4-3. Relationship Between Counter And Resolution

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 after overflowing LEDC_OVF_NUM_CH*n* + 1 times. To configure LEDC_OVF_CNT_CH*n*_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CH*n* to select the timer for the PWM generator
2. Enable the overflow counter by setting LEDC_OVF_CNT_EN_CH*n*
3. Configure LEDC_OVF_NUM_CH*n* with the number of counter overflows (that triggers an interrupt) minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CH*n*_INT_ENA
5. Configure LEDC_TIMER*x*_DUTY_RES to specify the counter bit width for the selected Timer*x* and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP or LEDC_PARA_UP_CH*n*. LEDC_TIMER*x*_PARA_UP and LEDC_PARA_UP_CH*n* will be automatically cleared by hardware.

Referring to Figure 23.3-2, the frequency of a PWM generator output signal (sig_out*n*) is dependent on the frequency of the timer's clock source LEDC_CLK, the clock divisor LEDC_CLK_DIV, and the duty resolution (counter width) LEDC_TIMER*x*_DUTY_RES:

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC\_TIMER}x\text{\_DUTY\_RES} = \log_2\left(\frac{f_{\text{LEDC\_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}}\right)$$

Table 23.4-1 lists the commonly-used frequencies and their corresponding resolutions.

Table 23.4-1. Commonly-used Frequencies and Resolutions

LEDC_CLK	PWM Frequency	Highest Resolution (bit) [1]	Lowest Resolution (bit) [2]
REF_80M_CLK (80 MHz)	1 kHz	16	7
REF_80M_CLK (80 MHz)	5 kHz	13	4
REF_80M_CLK (80 MHz)	10 kHz	12	3
XTAL_40M_CLK (40 MHz)	1 kHz	15	6
XTAL_40M_CLK (40 MHz)	4 kHz	13	4
FOSC_20M_CLK (20 MHz)	1 kHz	14	5
FOSC_20M_CLK (20 MHz)	2 kHz	13	4

[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.

23.4.3.3 Timer Configuration Constraints

The clock divisor LEDC_CLK_DIV and the duty resolution LEDC_TIMER*x*_DUTY_RES of the timer's 20-bit counter must satisfy the following condition:

$$\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} \geq 8 \tag{23.1}$$

23.4.4 PWM Generators

Each LED PWM generator can be independently enabled by setting the corresponding LEDC_CH*n*_POWER_UP field.

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 23.3-2, each PWM generator mainly consists of 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 can output a high or low level:

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Figure 23.4-4 illustrates how Hpoint*n* and Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

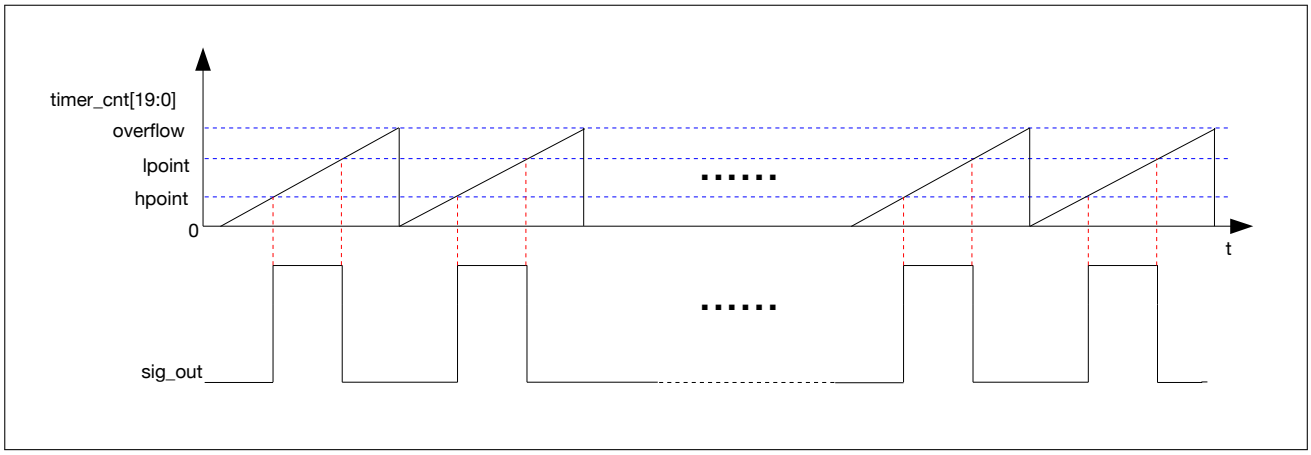


Figure 23.4-4. LED PWM Output Signal Diagram

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*.

23.4.4.1 PWM Output Signal Dithering

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.

23.4.4.2 Applying the Changed Configuration

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.

To disable an LED PWM generator, clear its corresponding LEDC_CH*n*_POWER_UP field.

23.4.5 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 the fading direction (increase or decrease), the step size, the number of fades, and the fading frequency. If duty cycle fading is enabled, every range's Lpoint*n* value will change according to its fading configuration.

23.4.5.1 Linear Duty Cycle Fading

Linear fading PWM signals can be generated by configuring the direction, the step size, the number of fades, and the fading frequency of the first duty cycle range.

The programming procedures will be described in detail in the section 23.7 *Programming Procedures*.

After the procedures, the PWM generator can fade the duty cycle of a PWM signal once per LEDC_CH*n*_GAMMA_DUTY_CYCLE times of counter overflows. Every time when the PWM signal is faded, Lpoint*n* increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by LEDC_CH*n*_GAMMA_SCALE; accordingly, the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by ΔDC.

$$\Delta DC = \frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} \tag{23.2}$$

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 23.4-5 shows a linear fading PWM signal.

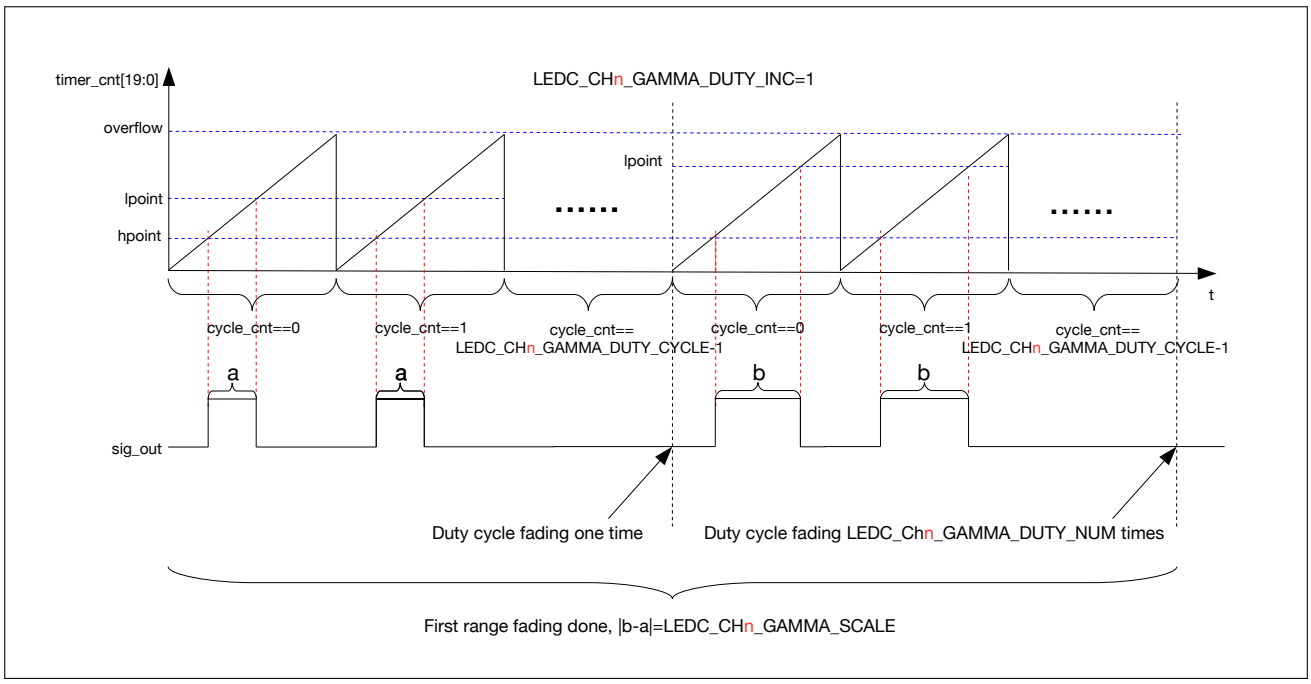


Figure 23.4-5. Output Signal of Linear Duty Cycle Fading

23.4.5.2 Gamma Curve Fading

Gamma curve fading PWM signals can be generated by configuring the fading direction, the step size, the number of fades, and the fading frequency of multiple duty cycle fading ranges.

The programming procedures will be described in detail in the section 23.7 *Programming Procedures*.

After the procedures, the PWM generator can generate a PWM signal with LEDC_CH*n*_GAMMA_ENTRY_NUM ranges. The duty cycle of the PWM signal fades according to the configurations of range 0 first, and then range 1, till range (LEDC_CH*n*_GAMMA_ENTRY_NUM − 1) (the last range) where duty cycle fading ends. The PWM signal fades independently in each range. For each range, the PWM generator fades the duty cycle of the 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; accordingly, the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by ΔDC.

$$\Delta DC = \frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} \tag{23.3}$$

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 can be fitted to a specific gamma curve.

Figure 23.4-6 illustrates a gamma curve fading PWM signal.

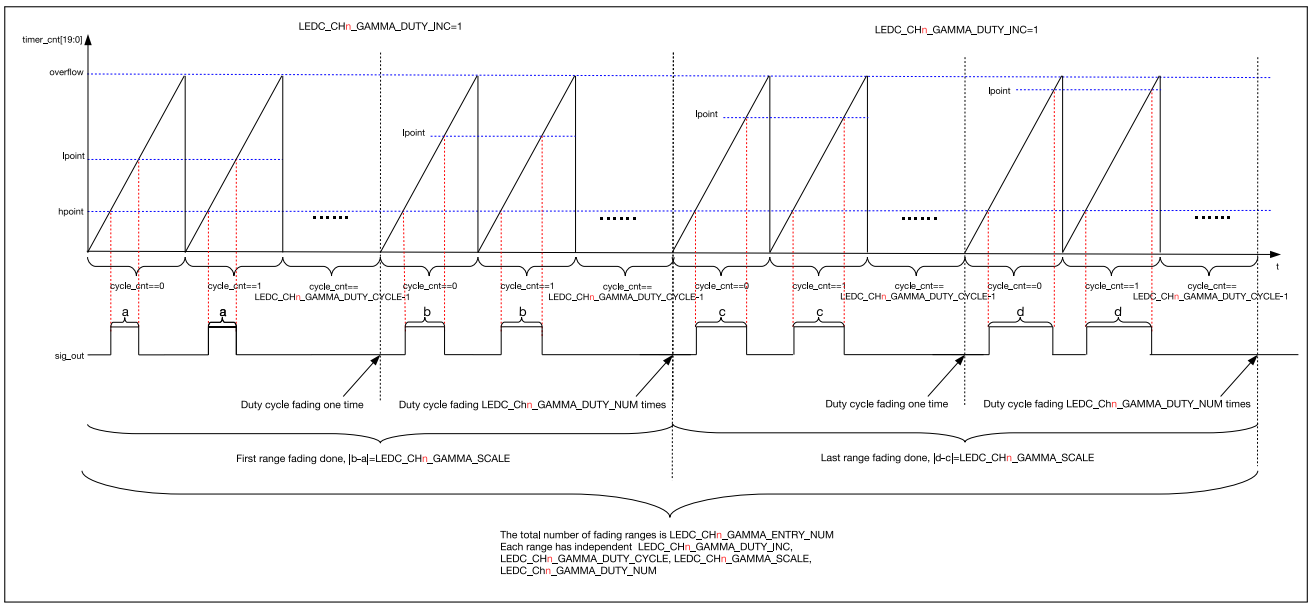


Figure 23.4-6. Output Signal of Gamma Curve Fading

23.4.5.3 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. At this point, the PWM signal will continue the fading process from the paused state, following the original range configuration. The hardware will automatically clear the LEDC_CH*n*_GAMMA_PAUSE field after LEDC_CH*n*_GAMMA_RESUME is set to 1.

23.5 Event Task Matrix Feature

The LEDC on ESP32-S31 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. For more information on ETM, please refer to Chapter 10 Event Task Matrix (ETM) [to be added later]. This section only introduces the ETM tasks and events related to LEDC.

ETM-related events and tasks are enabled by configuring corresponding fields of the LEDC_EVT_TASK_EN0_REG, LEDC_EVT_TASK_EN1_REG and LEDC_EVT_TASK_EN2_REG registers. For the correspondence between events/tasks and the fields of the above registers, please refer to Section 23.9.

In this section, *n* denotes the PWM channel index (ranges from 0 to 7), and *x* denotes the timer index (ranges from 0 to 3).

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* restarts generation of 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 23.3-2.
- LEDC_TASK_OVF_CNT_RST_CH*n*: If the LEDC_TASK_OVF_CNT_RST_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_OVF_CNT_RST_CH*n* task, PWM*n* timer's overflow counter is reset to 0.
- LEDC_TASK_TIMER*x*_RST: If the LEDC_TASK_TIMER*x*_RST_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RST task, Timer*x*'s counter is reset to 0.
- LEDC_TASK_TIMER*x*_PAUSE: If the LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_PAUSE task, the corresponding timer (Timer*x*) will be paused.
- LEDC_TASK_TIMER*x*_RESUME: If the LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RESUME task, the corresponding timer (Timer*x*) will be resumed if it is currently in a paused state.
- 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 generation of 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 the counter of the timer used by PWM*n* overflows LEDC_OVF_NUM_CH*n* + 1 times.
- LEDC_EVT_TIME_OVF_TIMER*x*: Generated when the LEDC_EVT_TIME_OVF_TIMER*x*_EN field is enabled and Timer*x*'s counter overflows.
- LEDC_EVT_TIMER*x*_CMP: Generated when the LEDC_EVT_TIMER*x*_CMP_EN field is enabled and the value of Timer*x*'s counter reaches that of the LEDC_TIMER*x*_CMP field of register LEDC_TIMER*x*_CMP_REG.

In practical applications, LEDC's ETM events can trigger its own ETM tasks. For example, LEDC_EVT_DUTY_CHNG_END_CH*n* event can trigger the LEDC_TASK_GAMMA_RESTART_CH*n* task, thus starting the next fading directly after the current fading is completed.

23.6 Interrupts

ESP32-S31's LEDC can generate the LEDC_INT interrupt signal, which will be sent to the Interrupt Matrix.

Interrupt signal LEDC_INT can be generated by the following internal interrupt sources:

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for LEDC_OVF_NUM_CH*n* + 1 times and the register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on PWM*n* has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when Timer*x* has reached its maximum counter value.

The above interrupt sources can only be triggered when the interrupt enable register *XXX*_ENA is set to 1. Writing 1 to *XXX*_CLR clears the interrupt (*XXX* is the interrupt source name).

23.7 Programming Procedures

The configuration process of the LED PWM controller to generate PWM signals is as follows:

1. Configure the clock
   (a) Enable LEDC0 APB_CLK by setting the HP_SYS_CLKRST_LEDC0_APB_CLK_EN field to 1; or Enable LEDC1 APB_CLK by setting the HP_SYS_CLKRST_LEDC1_APB_CLK_EN field to 1.
   (b) Configure the HP_SYS_CLKRST_LEDC0_CLK_SRC_SEL field to select the clock source for LEDC0_CLK; or configure the HP_SYS_CLKRST_LEDC1_CLK_SRC_SEL field to select the clock source for LEDC1_CLK.
      - XTAL_CLK: Set HP_SYS_CLKRST_LEDC0_CLK_SRC_SEL/HP_SYS_CLKRST_LEDC1_CLK_SRC_SEL to 0
      - RC_FAST_CLK: Set HP_SYS_CLKRST_LEDC0_CLK_SRC_SEL/HP_SYS_CLKRST_LEDC1_CLK_SRC_SEL to 1
      - PLL_F80M_CLK: Set HP_SYS_CLKRST_LEDC0_CLK_SRC_SEL/HP_SYS_CLKRST_LEDC1_CLK_SRC_SEL to 2
   (c) Enable LEDC0_CLK by setting the HP_SYS_CLKRST_LEDC0_CLK_EN field to 1; or enable LEDC1_CLK by setting the HP_SYS_CLKRST_LEDC1_CLK_EN field to 1.
2. Configure the Timer*x* according to section 23.4.3. After that, the clock source, divider, and counter are ready.
3. Configure HP_SYS_LEDC0_MEM_LP_CTRL_REG or HP_SYS_LEDC1_MEM_LP_CTRL_REG to power up the LEDC memory. This memory is powered down by default, and subsequent PWM*n* generator configurations cannot be performed until it is powered up.
4. Configure the PWM*n* generator according to section 23.4.4. After that, one of the four timers is selected for PWM*n*, and the phase and duty cycle of the PWM*n* output are set.

5. Choose the corresponding configuration process based on the different duty cycle fading types:

   (a) Linear duty cycle fading:

      i. Configure the LEDC_DUTY_CH*n* field with the initial value of Lpoint*n*.

      ii. Set the LEDC_DUTY_START_CH*n* field to enable duty cycle fading. When this field is cleared, duty cycle fading will be disabled.

      iii. Write the linear duty cycle fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4). The LEDC_CH*n*_MEM starting address is provided in Table 23.8-1. Please note that the following field names are just for easier description, and there are no such register fields.

         - Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increase or decrease in the current configured range.
         - Bits 1 to 10 of the configuration data are used to configure the number of times the counter overflows per increment or decrement of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
         - Bits 11 to 20 of the configuration data are used to configure the amount by which Lpoint*n* increases or decreases in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
         - Bits 21 to 30 of the configuration data are used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
         - The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For linear duty cycle fading only the first range needs to be configured, so the duty cycle range number should be configured as 0.

      iv. Configure the number of ranges per each fading (1 in this case) via the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG. Once the specified number of ranges have been faded, duty cycle fading stops and the PWM generator triggers the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. For linear duty cycle fading there is only one duty cycle range (i.e., the first one), so configure LEDC_CH*n*_GAMMA_ENTRY_NUM as 1.

      v. Set the LEDC_PARA_UP_CH*n* field to apply the above configurations. After this field is set, the configurations for duty cycle fading will take effect upon the next overflow of the counter, and the PWM generator will output a linear fading PWM signal according to the specified configuration. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

   (b) Gamma curve fading:

      i. The same as Step 1 for linear duty cycle fading.

      ii. The same as Step 2 for linear duty cycle fading.

      iii. Configure multiple duty cycle ranges with the following steps. Write the gamma curve fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4). The LEDC_CH*n*_MEM starting address is provided in Table 23.8-1.

- Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increment or decrement in the current configured range.
- Bits 1 to 10 of the configuration data are used to configure the number of times the counter overflows per increase or decrease of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
- Bits 11 to 20 of the configuration data are used to configure the amount by which Lpoint*n* increases or decreases in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
- Bits 21 to 30 of the configuration data are used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
- The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For gamma curve fading, it must start from 0 and increase by 1 for the next range to be configured.
- Once the above procedures are finished, the configuration for one range is complete. Other ranges are configured by repeating the same set of procedures. You can configure any number of ranges from 0 to 15, and each can be configured independently.

iv. After all required ranges are configured, write the total number of ranges configured in Step 3 to the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG register.

v. Set the LEDC_PARA_UP_CH*n* field to apply the above configuration. After this field is set, the configurations for duty cycle fading will take effect upon the next overflow of the counter, and the PWM generator will output a gamma curve fading PWM signal following the configurations. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

After the above procedures, the LED PWM controller will generate the desired PWM signals.

At any time, duty cycle fading can be suspended or resumed. For more details, see section 23.4.5.3. If the duty cycle fading configurations need to be changed before the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has not been triggered), the duty cycle fading needs to be suspended before changing the configurations. However, if the duty cycle fading configurations need to be changed after the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has been triggered), the configurations can be changed directly.

At any time, PWM signal output can be enabled or disabled by software, and the output signal level can be changed. For details, see section 23.4.4.

23.8 Memory Blocks

Each LEDC channel has a memory block to store duty cycle fading configuration data. Each memory can store configuration data for up to 16 ranges. The addresses in this section are relative to LED PWM Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Table 23.8-1. LEDC Memory Blocks

Name	Description	Size	Starting Address	Ending Address	Access
LEDC_CH*n*_MEM	Memory of PWM*n*	64 B	0x400 + PWM*n* * 64	0x400 + (PWM*n* + 1) * 64	WR

23.9 Register Summary

The addresses in this section are relative to LED PWM Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH0_DUTY_REG	Initial duty cycle register for channel 0	0x0008	R/W
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W/SC
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH1_DUTY_REG	Initial duty cycle register for channel 1	0x001C	R/W
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	R/W/SC
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH2_DUTY_REG	Initial duty cycle register for channel 2	0x0030	R/W
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	R/W/SC
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH3_DUTY_REG	Initial duty cycle register for channel 3	0x0044	R/W
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	R/W/SC
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH4_DUTY_REG	Initial duty cycle register for channel 4	0x0058	R/W
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	R/W/SC
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
LEDC_CH5_DUTY_REG	Initial duty cycle register for channel 5	0x006C	R/W
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W/SC
LEDC_CH6_CONF0_REG	Configuration register 0 for channel 6	0x0078	varies
LEDC_CH6_HPOINT_REG	High point register for channel 6	0x007C	R/W
LEDC_CH6_DUTY_REG	Initial duty cycle register for channel 6	0x0080	R/W
LEDC_CH6_CONF1_REG	Configuration register 1 for channel 6	0x0084	R/W/SC
LEDC_CH7_CONF0_REG	Configuration register 0 for channel 7	0x008C	varies
LEDC_CH7_HPOINT_REG	High point register for channel 7	0x0090	R/W
LEDC_CH7_DUTY_REG	Initial duty cycle register for channel 7	0x0094	R/W
LEDC_CH7_CONF1_REG	Configuration register 1 for channel 7	0x0098	R/W/SC
LEDC_TIMER0_CONF_REG	Timer 0 configuration register	0x00A0	varies
LEDC_TIMER1_CONF_REG	Timer 1 configuration register	0x00A8	varies
LEDC_TIMER2_CONF_REG	Timer 2 configuration register	0x00B0	varies
LEDC_TIMER3_CONF_REG	Timer 3 configuration register	0x00B8	varies

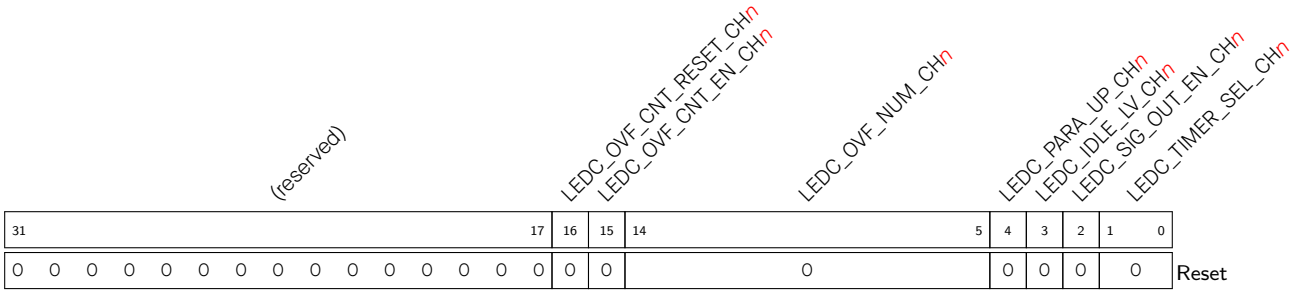
Name	Description	Address	Access
LEDC_CH0_GAMMA_CONF_REG	LEDC channel 0 gamma config register	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	LEDC channel 1 gamma config register	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	LEDC channel 2 gamma config register	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	LEDC channel 3 gamma config register	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	LEDC channel 4 gamma config register	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	LEDC channel 5 gamma config register	0x0114	varies
LEDC_CH6_GAMMA_CONF_REG	LEDC channel 6 gamma config register	0x0118	varies
LEDC_CH7_GAMMA_CONF_REG	LEDC channel 7 gamma config register	0x011C	varies
LEDC_EVT_TASK_EN0_REG	LEDC event task enable bit register 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC event task enable bit register 1	0x0124	R/W
LEDC_EVT_TASK_EN2_REG	LEDC event task enable bit register 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	LEDC timer 0 compare value register	0x0140	R/W
LEDC_TIMER1_CMP_REG	LEDC timer 1 compare value register	0x0144	R/W
LEDC_TIMER2_CMP_REG	LEDC timer 2 compare value register	0x0148	R/W
LEDC_TIMER3_CMP_REG	LEDC timer 3 compare value register	0x014C	R/W
LEDC_CONF_REG	LEDC global configuration register	0x0170	R/W
LEDC_CH_POWER_UP_CONF_REG	LEDC channel power up configuration register	0x0174	R/W
LEDC_TIMER_POWER_UP_CONF_REG	LEDC timer power up configuration register	0x0178	R/W
Status Register			
LEDC_CH0_DUTY_R_REG	Current duty cycle register for channel 0	0x0010	RO
LEDC_CH1_DUTY_R_REG	Current duty cycle register for channel 1	0x0024	RO
LEDC_CH2_DUTY_R_REG	Current duty cycle register for channel 2	0x0038	RO
LEDC_CH3_DUTY_R_REG	Current duty cycle register for channel 3	0x004C	RO
LEDC_CH4_DUTY_R_REG	Current duty cycle register for channel 4	0x0060	RO
LEDC_CH5_DUTY_R_REG	Current duty cycle register for channel 5	0x0074	RO
LEDC_CH6_DUTY_R_REG	Current duty cycle register for channel 6	0x0088	RO
LEDC_CH7_DUTY_R_REG	Current duty cycle register for channel 7	0x009C	RO
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value register	0x00A4	RO
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value register	0x00AC	RO
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value register	0x00B4	RO
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value register	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	LEDC timer 0 captured count value register	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	LEDC timer 1 captured count value register	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	LEDC timer 2 captured count value register	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	LEDC timer 3 captured count value register	0x015C	RO
Interrupt Register			
LEDC_INT_RAW_REG	Interrupt raw status register	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	Interrupt masked status register	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable register	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear register	0x00CC	WT
Version Register			
LEDC_DATE_REG	Version control register	0x017C	R/W

23.10 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 23.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* Configures which timer is selected by channel *n*.

0: Select Timer 0

1: Select Timer 1

2: Select Timer 2

3: Select Timer 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* Configures whether to enable signal output on channel *n*.

0: Signal output disable

1: Signal output enable

(R/W)

LEDC_IDLE_LV_CH*n* Configures the output value when channel *n* is inactive. Valid only when LEDC_SIG_OUT_EN_CH*n* is 0.

0: Output level is low

1: Output level is high

(R/W)

LEDC_PARA_UP_CH*n* Configures whether to update LEDC_HPOINT_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_SIG_OUT_EN_CH*n*, LEDC_TIMER_SEL_CH*n*, LEDC_OVF_CNT_EN_CH*n* fields and duty cycle range configurations for channel *n*. This field will be automatically cleared by hardware.

0: Invalid. No effect

1: Update

(WT)

LEDC_OVF_NUM_CH*n* Configures the maximum times of overflow minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when when the timer used by channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* Configures whether to enable the overflow counter of channel *n*.

0: Disable

1: Enable

(R/W)

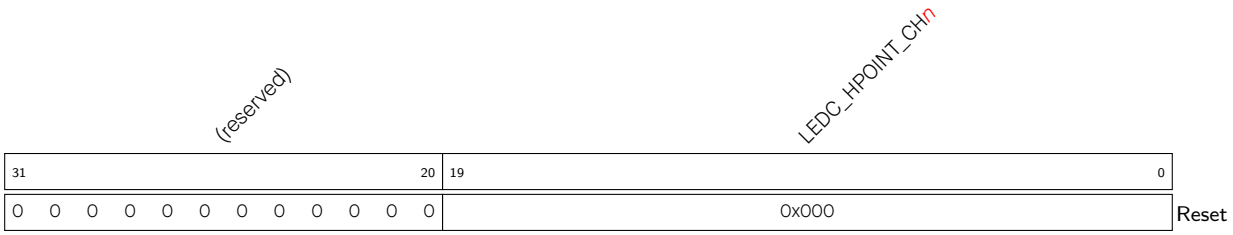
LEDC_OVF_CNT_RESET_CH*n* Configures whether to reset the overflow counter of channel *n*.

0: Invalid. No effect

1: Reset the overflow counter

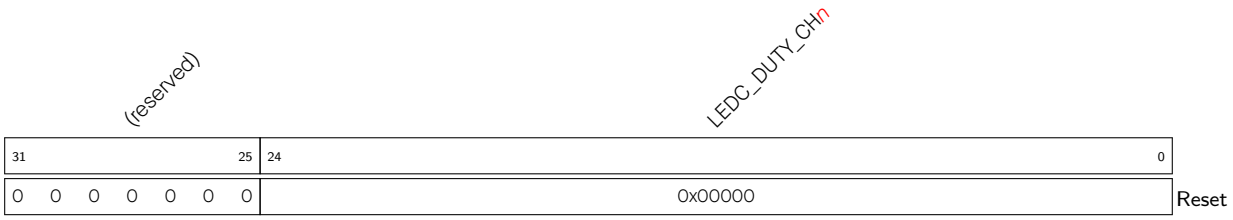
(WT)

Register 23.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+0x14*n*)



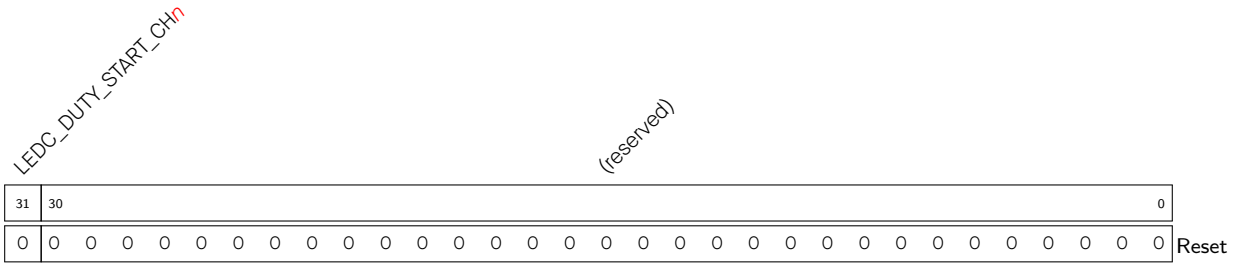
LEDC_HPOINT_CH*n* Configures the Hpoint value for signal output on channel *n*. The output value changes to high when the selected timers have reached the value specified by this register. (R/W)

Register 23.3. LEDC_CH*n*_DUTY_REG (*n*: 0-7) (0x0008+0x14*n*)



LEDC_DUTY_CH*n* Configures the duty cycle of signal output on channel *n*. (R/W)

Register 23.4. LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+0x14*n*)



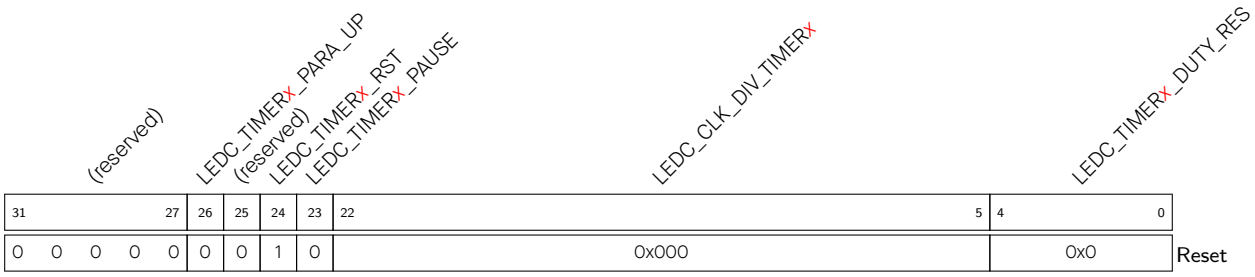
LEDC_DUTY_START_CH*n* Configures whether the duty cycle fading configurations take effect.

0: Not take effect

1: Take effect

(R/W/SC)

Register 23.5. LEDC_TIMERx_CONF_REG (x: 0-3) (0x00A0+0x8*x)



LEDC_TIMERx_DUTY_RES Configures the bit width of the counter in timer *x*. Valid values are 1 to 20. (R/W)

LEDC_CLK_DIV_TIMERx Configures the divisor for the divider in timer *x*.

The least significant eight bits represent the fractional part. The most significant ten bits represent the integer part. (R/W)

LEDC_TIMERx_PAUSE Configures whether to pause the counter in timer *x*.

0: Normal

1: Pause

(R/W)

LEDC_TIMERx_RST Configures whether to reset the counter of timer *x*. The counter value will be 0 after reset.

0: Not reset

1: Reset

(R/W)

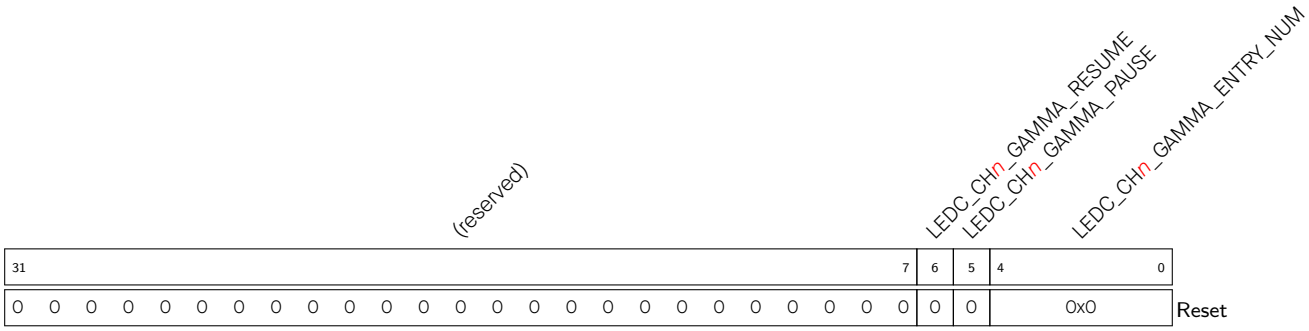
LEDC_TIMERx_PARA_UP Configures whether to update LEDC_CLK_DIV_TIMERx and LEDC_TIMERx_DUTY_RES.

0: Invalid. No effect

1: Update

(WT)

Register 23.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-7) (0x0100+0x4**n*)



LEDC_CH*n*_GAMMA_ENTRY_NUM Configures the number of duty cycle fading ranges for LEDC channel *n*. (R/W)

LEDC_CH*n*_GAMMA_PAUSE Configures whether to pause duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Pause

(WT)

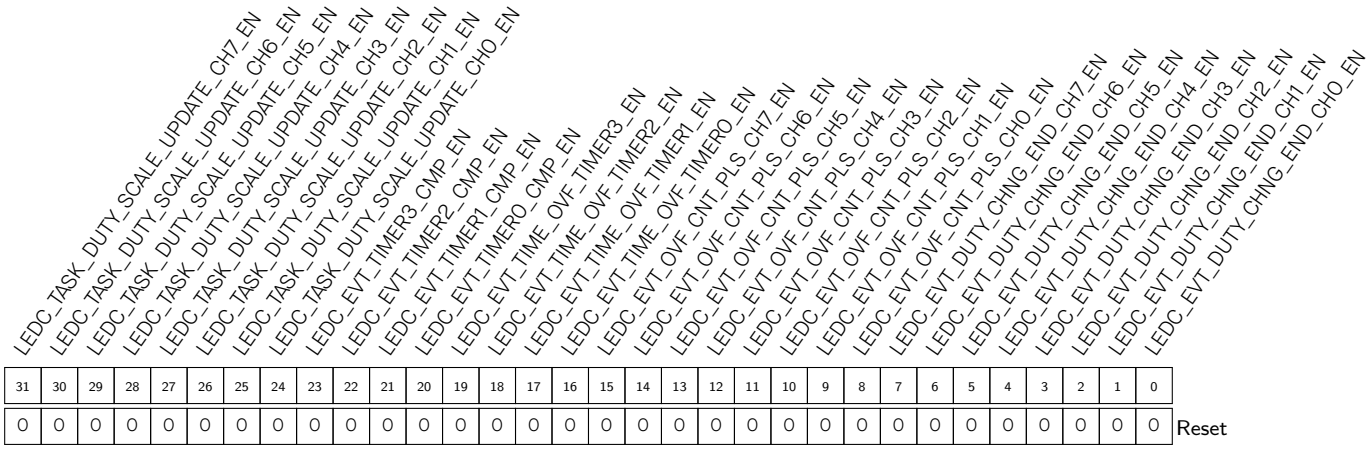
LEDC_CH*n*_GAMMA_RESUME Configures whether to resume duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Resume

(WT)

Register 23.7. LEDC_EVT_TASK_EN0_REG (0x0120)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_EVT_DUTY_CHNG_END_CH*n* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_EVT_OVF_CNT_PLS_CH*n* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN (*x*: 0-3) Configures whether to enable the LEDC_EVT_TIME_OVF_TIMER*x* event.

0: Disable

1: Enable

(R/W)

LEDC_EVT_TIMER*x*_CMP_EN (*x*: 0-3) Configures whether to enable the LEDC_EVT_TIMER*x*_CMP event.

0: Disable

1: Enable

(R/W)

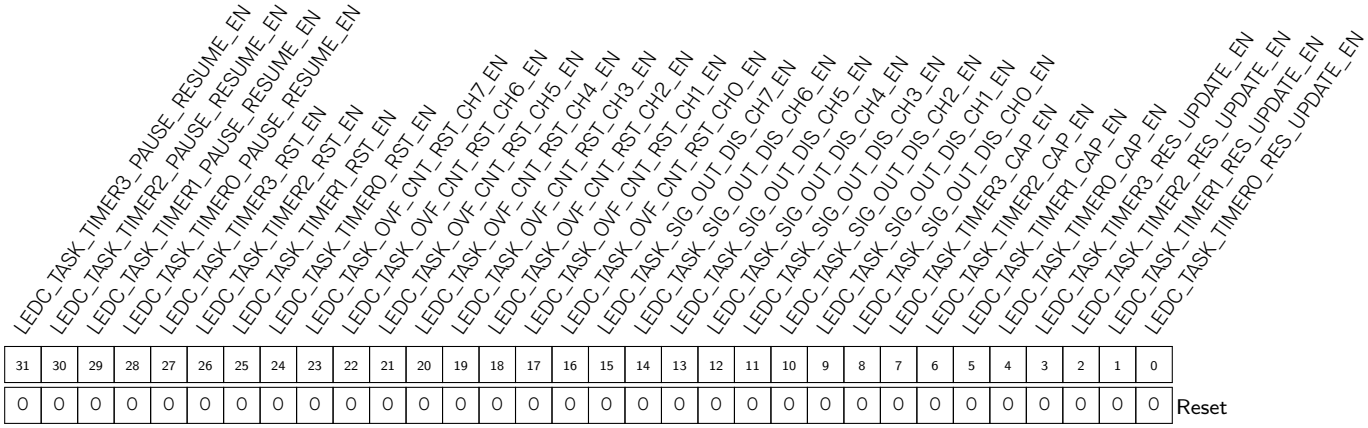
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 23.8. LEDC_EVT_TASK_EN1_REG (0x0124)



LEDC_TASK_TIMER*x*_RES_UPDATE_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_RES_UPDATE task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_CAP task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_SIG_OUT_DIS_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_OVF_CNT_RST_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMER*x*_RST_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_RST task.

0: Disable

1: Enable

(R/W)

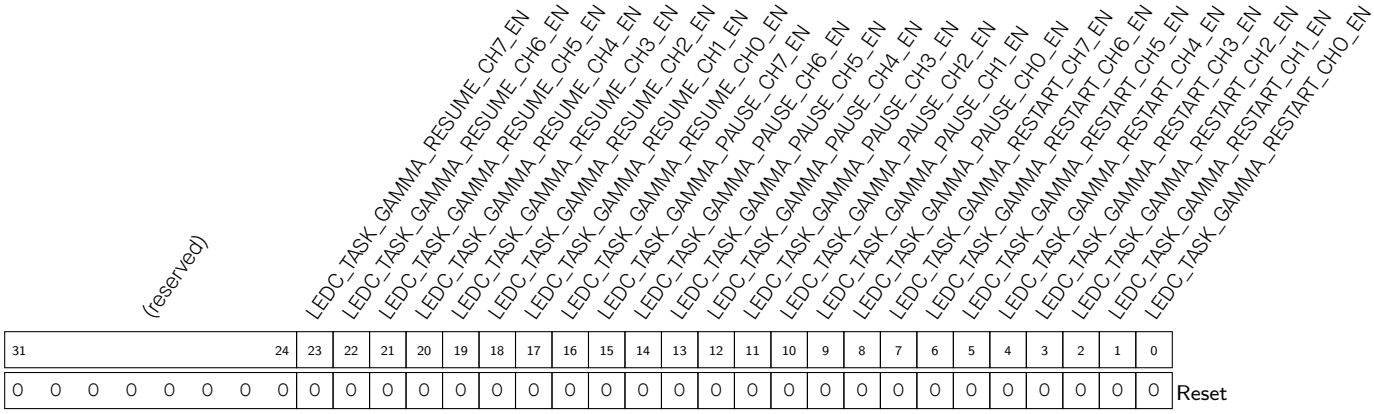
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN (*x*: 0-3) Configures whether to enable LEDC_TASK_TIMER*x*_PAUSE and LEDC_TASK_TIMER*x*_RESUME task.

0: Disable

1: Enable

(R/W)

Register 23.9. LEDC_EVT_TASK_EN2_REG (0x0128)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_RESTART_CH*n* task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_PAUSE_CH*n* task.

0: Disable

1: Enable

(R/W)

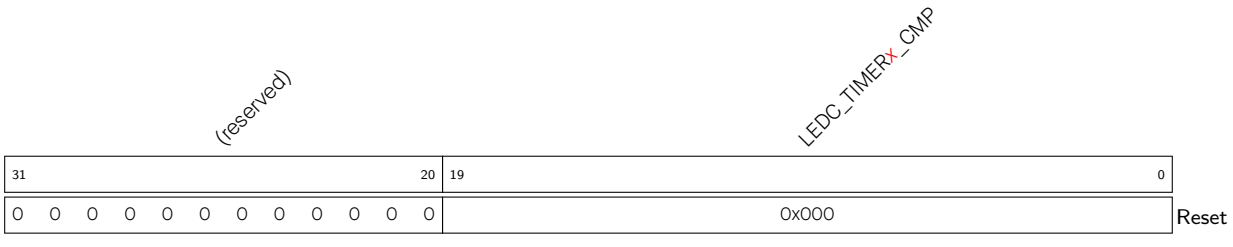
LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_RESUME_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 23.10. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x0140+0x4**x*)



LEDC_TIMER*x*_CMP Configures the comparison value for LEDC timer *x*. (R/W)

Register 23.11. LEDC_CONF_REG (0x0170)

Bit	31	30–10	9	8	7	6	5	4	3	2	1–0	
Field	LEDC_CLK_EN	(reserved)	LEDC_GAMMA_RAM_CLK_EN_CH7	LEDC_GAMMA_RAM_CLK_EN_CH6	LEDC_GAMMA_RAM_CLK_EN_CH5	LEDC_GAMMA_RAM_CLK_EN_CH4	LEDC_GAMMA_RAM_CLK_EN_CH3	LEDC_GAMMA_RAM_CLK_EN_CH2	LEDC_GAMMA_RAM_CLK_EN_CH1	LEDC_GAMMA_RAM_CLK_EN_CH0	(reserved)	
Value	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_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-7) Configures whether to force on the gamma RAM clock gate for LEDC channel *n*.

0: Enable the clock gate only when an application reads from or writes to the gamma RAM of LEDC channel *n*

1: Force on the clock gate for the gamma RAM of LEDC channel *n*

(R/W)

LEDC_CLK_EN Configures whether to force on the register clock gate.

0: Enable the clock gate only when an application writes registers

1: Force on the clock gate for register

(R/W)

Register 23.12. LEDC_CH_POWER_UP_CONF_REG (0x0174)

Bit	31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field	(reserved)	LEDC_CH7_CLK_GATE_FORCE_ON	LEDC_CH6_CLK_GATE_FORCE_ON	LEDC_CH5_CLK_GATE_FORCE_ON	LEDC_CH4_CLK_GATE_FORCE_ON	LEDC_CH3_CLK_GATE_FORCE_ON	LEDC_CH2_CLK_GATE_FORCE_ON	LEDC_CH1_CLK_GATE_FORCE_ON	LEDC_CH0_CLK_GATE_FORCE_ON	LEDC_CH7_POWER_UP	LEDC_CH6_POWER_UP	LEDC_CH5_POWER_UP	LEDC_CH4_POWER_UP	LEDC_CH3_POWER_UP	LEDC_CH2_POWER_UP	LEDC_CH1_POWER_UP	LEDC_CH0_POWER_UP	
Value	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_CH*n*_POWER_UP (*n*: 0-7) Configures whether to power up channel *n*.

0: Power down

1: Power up

(R/W)

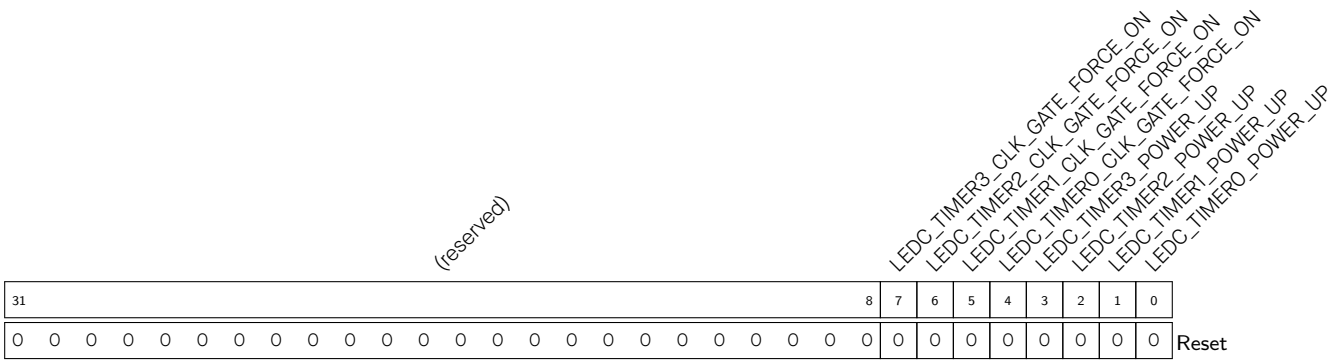
LEDC_CH*n*_CLK_GATE_FORCE_ON (*n*: 0-7) Configures whether to force on the clock gate for channel *n*.

0: Enable the clock gate only when channel *n* is power up

1: Force on the clock gate for channel *n*

(R/W)

Register 23.13. LEDC_TIMER_POWER_UP_CONF_REG (0x0178)



LEDC_TIMER*x*_POWER_UP (*x*: 0-3) Configures whether to power up Timer*x*.

0: Power down

1: Power up

(R/W)

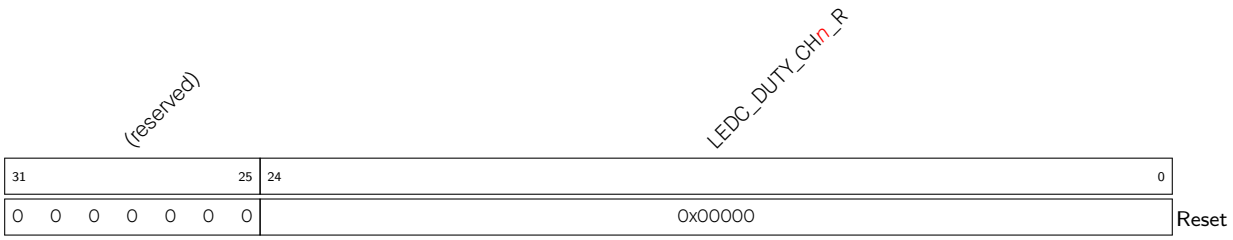
LEDC_TIMER*x*_CLK_GATE_FORCE_ON (*x*: 0-3) Configures whether to force on the clock gate for Timer*x*.

0: Enable the clock gate only when Timer*x* is power up

1: Force on the clock gate for Timer*x*

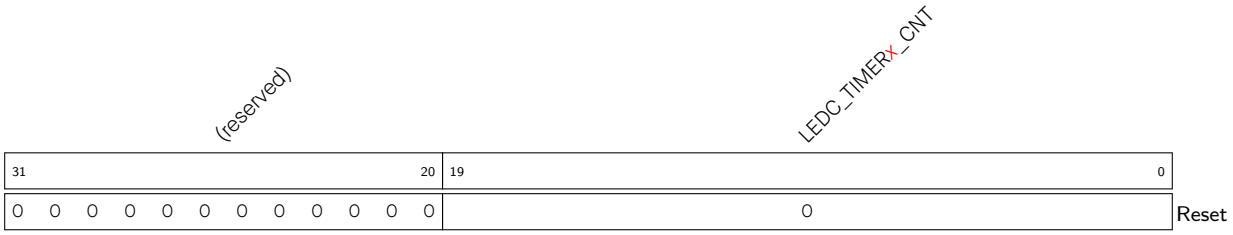
(R/W)

Register 23.14. LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+0x14**n*)



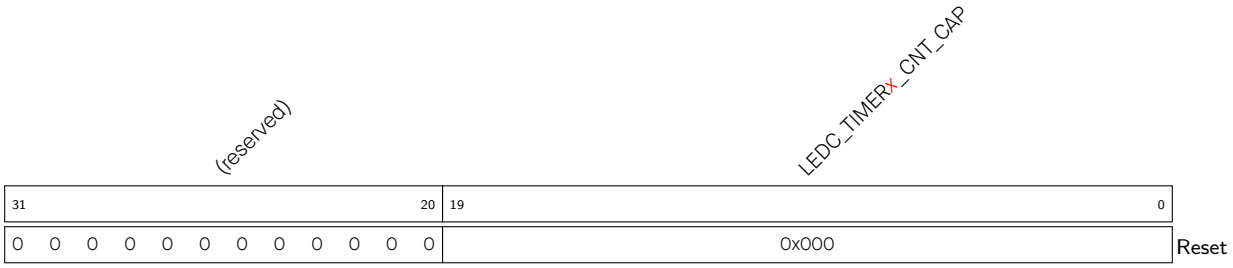
LEDC_DUTY_CH*n*_R Represents the current duty cycle of output signal on channel *n*. (RO)

Register 23.15. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8**x*)



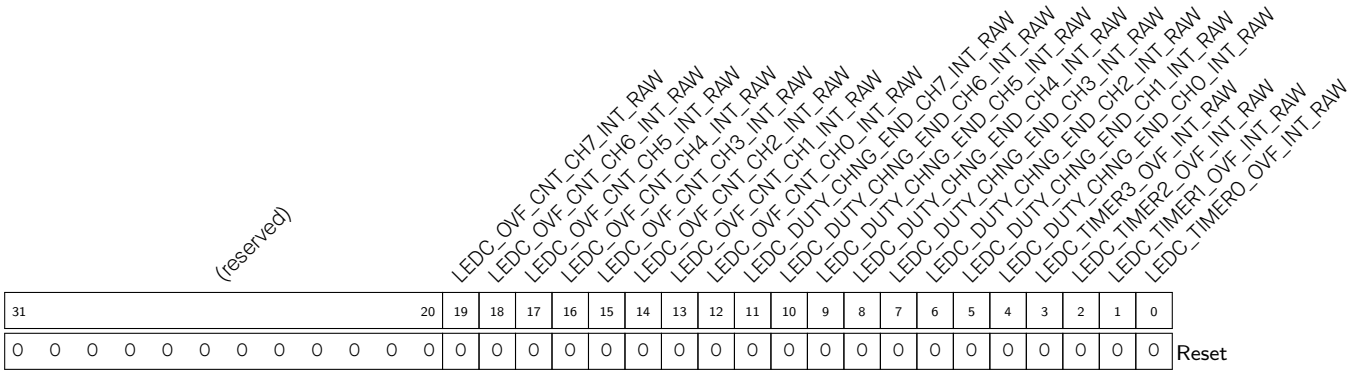
LEDC_TIMER*x*_CNT Represents the current counter value of timer *x*. (RO)

Register 23.16. LEDC_TIMERx_CNT_CAP_REG (x: 0-3) (0x0150+0x4*x)



LEDC_TIMERx_CNT_CAP Represents the captured LEDC timer *x* count value. (RO)

Register 23.17. LEDC_INT_RAW_REG (0x00C0)

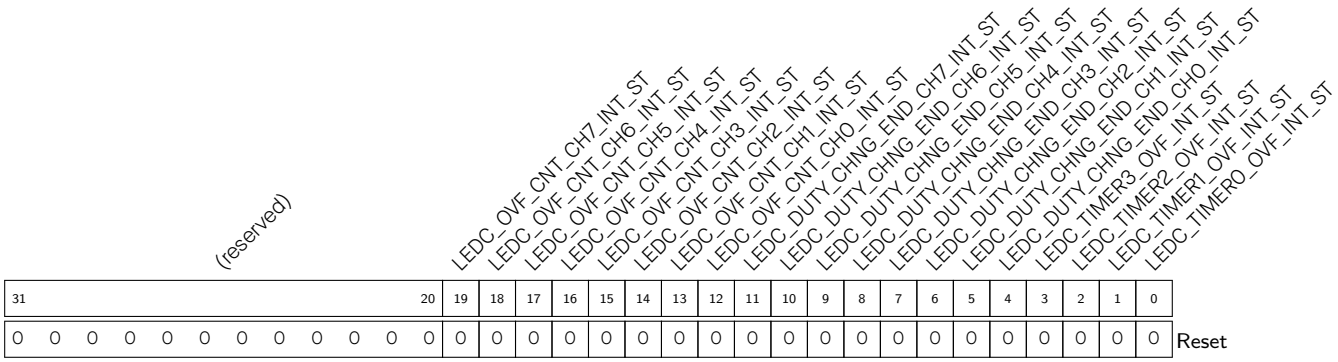


LEDC_TIMERx_OVF_INT_RAW (x: 0-3) Raw status bit: The raw interrupt status of LEDC_TIMERx_OVF_INT. Triggered when Timerx has reached its maximum counter value. (R/WTC/SS)

LEDC_DUTY_CHNG_END_CHn_INT_RAW (n: 0-7) Raw status bit: The raw interrupt status of LEDC_DUTY_CHNG_END_CHn_INT. Triggered when duty cycle fading has finished. (R/WTC/SS)

LEDC_OVF_CNT_CHn_INT_RAW (n: 0-7) Raw status bit: The raw interrupt status of LEDC_OVF_CNT_CHn_INT. Triggered when the overflow counter has reached the value specified by LEDC_OVF_NUM_CHn. (R/WTC/SS)

Register 23.18. LEDC_INT_ST_REG (0x00C4)

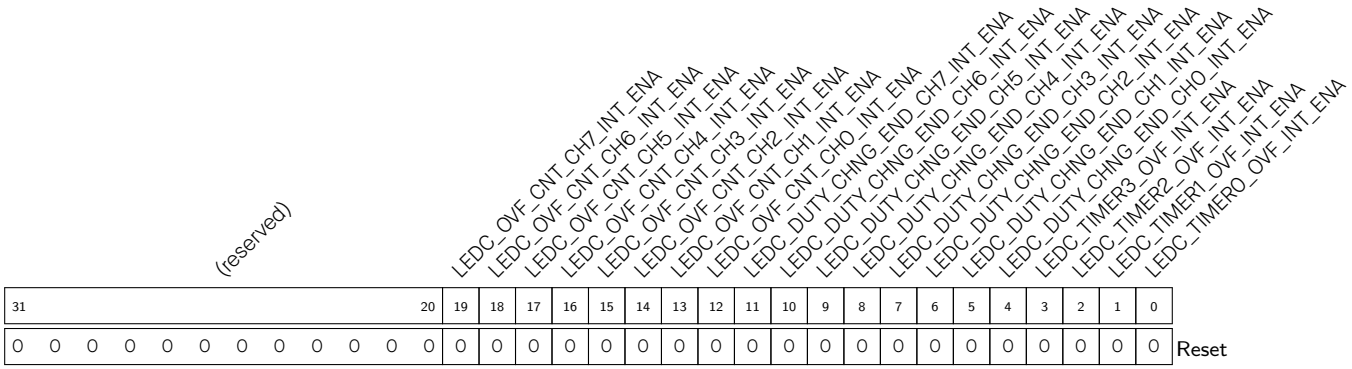


LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3) Masked status bit: The masked interrupt status of LEDC_TIMER*x*_OVF_INT. Valid only when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_DUTY_CHNG_END_CH*n*_INT. Valid only when LEDC_DUTY_CHNG_END_CH*n*_INT_ENA is set to 1. (RO)

LEDC_OVF_CNT_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_OVF_CNT_CH*n*_INT. Valid only when LEDC_OVF_CNT_CH*n*_INT_ENA is set to 1. (RO)

Register 23.19. LEDC_INT_ENA_REG (0x00C8)

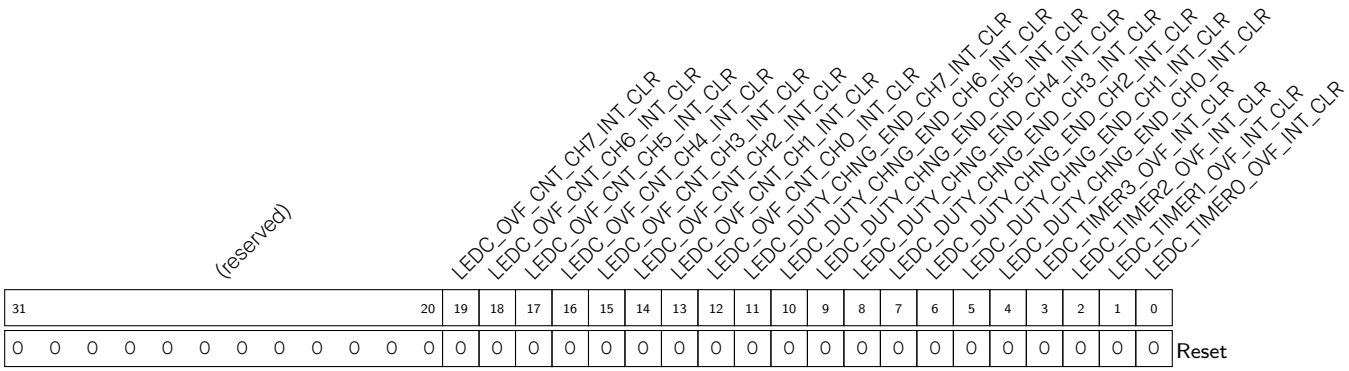


LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3) Enable bit: Write 1 to enable LEDC_TIMER*x*_OVF_INT. (R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_DUTY_CHNG_END_CH*n*_INT. (R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_OVF_CNT_CH*n*_INT. (R/W)

Register 23.20. LEDC_INT_CLR_REG (0x00CC)

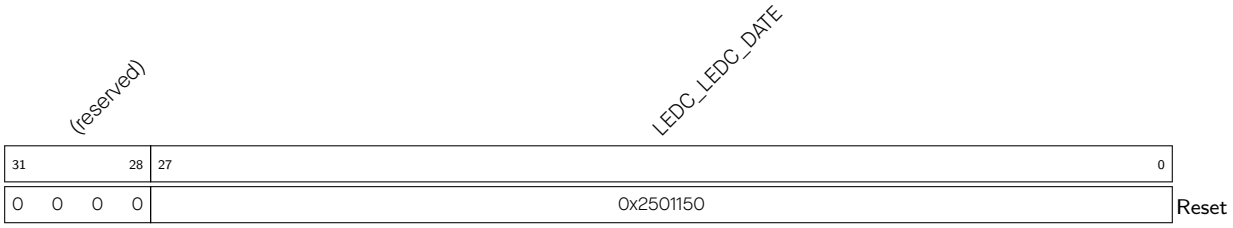


LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3) Clear bit: Write 1 to clear LEDC_TIMER*x*_OVF_INT. (WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_DUTY_CHNG_END_CH*n*_INT. (WT)

LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_OVF_CNT_CH*n*_INT. (WT)

Register 23.21. LEDC_DATE_REG (0x017C)



LEDC_LEDC_DATE Version control register. (R/W)

Chapter 24

Motor Control PWM (MCPWM)

24.1 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-S31 has one MCPWM peripheral, which is MCPWM0.

24.2 Features

An MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, a Capture module, an ETM module, and a Fault Detection module. MCPWM's core clock can be selected from three clock sources: PLL_F160M_CLK, XTAL_CLK, and RC_FAST_CLK (configured by the PCR_PWM_CLKM_SEL field of the PCR_PWM_CLK_CONF_REG register). Figure 24.2-1 shows the submodules inside MCPWM and the signals on the interface. PWM timers are used for generating timing references. The PWM operators generate the desired waveform based on the timing references. Any PWM operator can be configured to use the timing references of any PWM timers. Different PWM operators can use the same PWM timer's timing reference to generate PWM signals, or different PWM timers' values to generate separate PWM signals. Different PWM timers can also be synchronized together.

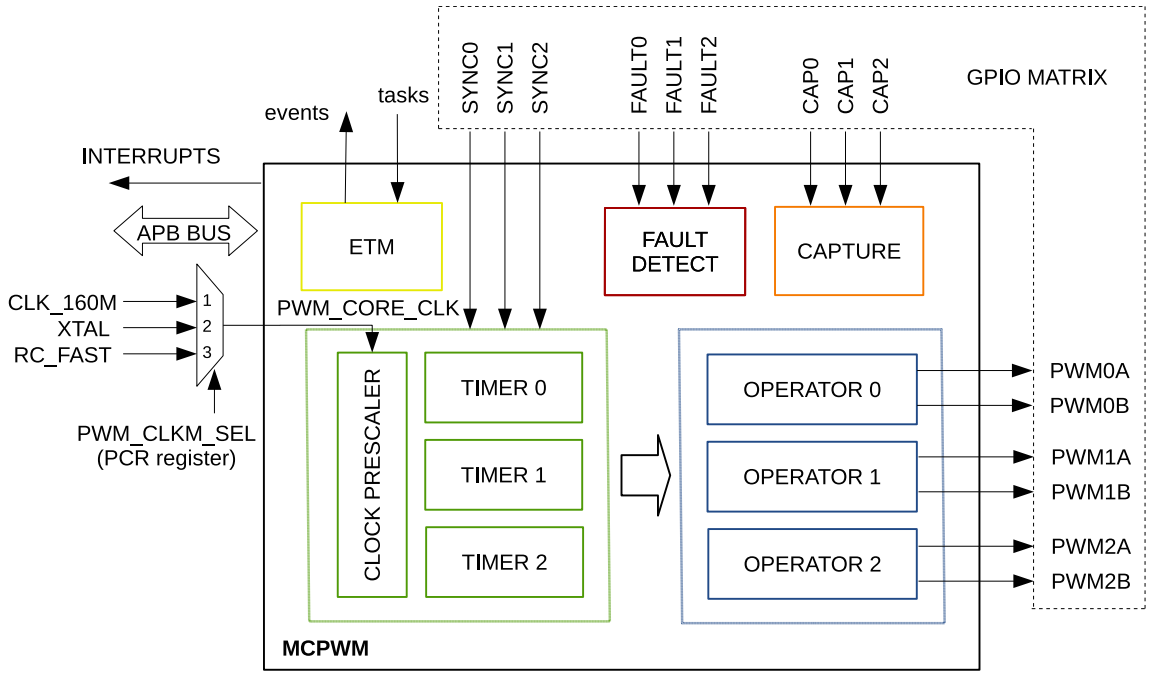


Figure 24.2-1. MCPWM Module Overview

Below is an overview of the submodules' functionality in Figure 24.2-1:

- PWM Timers 0, 1, and 2:
  - Every PWM timer has a dedicated 8-bit clock prescaler.
  - The 16-bit counter in the PWM timer can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode.
  - A hardware sync or software sync can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler's restart, so that the timer's clock can also be synced. The source of the hard sync can come from any GPIO or any other PWM timer's sync_out. The source of the soft sync comes from writing toggle value to the MCPWM_TIMER*n*_SYNC_SW bit.
- PWM Operators 0, 1, and 2:
  - Every PWM operator has two PWM outputs: PWM*x*A and PWM*x*B. They can work independently, in symmetric or asymmetric configurations.
  - The control of the PWM signal can be updated asynchronously.
  - Configurable dead time on rising and falling edges; each set up independently.
  - All events can trigger CPU interrupts.
  - Modulating of PWM output by high-frequency carrier signals, useful when gate drivers are insulated with a transformer.
  - Period, time-stamp, 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.

24.3 Modules

24.3.1 Overview

The key modules in each MCPWM are timer module, operator module, fault detection module, capture module, and ETM module. See their functions in the following sections.

24.3.1.1 Timer Module

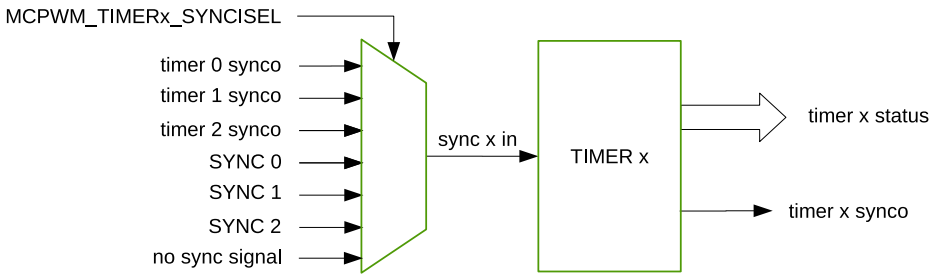


Figure 24.3-1. Timer Module

This module contains a timer, which can count at a specified period. It can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode. It supports different synchronization input sources (a total of seven optional input sources) for reloading the count value and direction, allowing synchronization among multiple timers. Furthermore, it provides synchronization output options (a total of four optional output sources) for use by other timers.

The timer*x* status in the figure represents the timer status output by the timer module, such as the counting mode, and count value equal to zero or period.

24.3.1.2 Operator Module

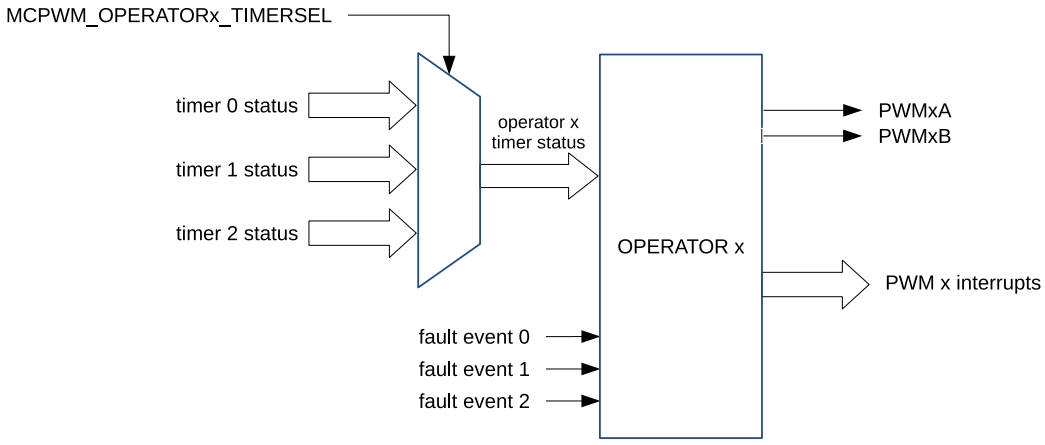


Figure 24.3-2. Operator Module

A PWM operator contains a PWM generator, a dead time generator, and a PWM carrier module. Their functions are shown in Table 24.3-1.

Table 24.3-1. Functions of Each Submodule in the Operator Module

Submodule	Functions
PWM Generator	Generate PWM signals according to the configured duty cycle and waveform.
Dead Time Generator	Choose to add dead time to the rising and falling edges of the PWM signal generated by the PWM generator (add delays to the rising and falling edges), and perform operations such as inversion.
PWM Carrier	Choose to add a carrier to the PWM signal generated by the dead time generator. The carrier frequency and the first pulse duration of the PWM waveform after adding the carrier can be configured.
Fault Detector	Detect faults and generate corresponding fault events, respond to fault events based on the specified interval mode (one-shot mode or cycle-by-cycle mode), and when a fault event occurs, take action on the PWM signal generated by the PWM carrier in the specified way.

24.3.1.3 Capture Module

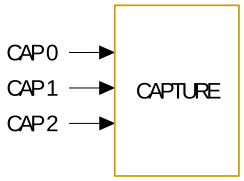


Figure 24.3-3. Capture Module

The module contains a timer that can be synchronized. When the input capture signal is valid, the count value of the timer is captured.

24.3.1.4 ETM Module

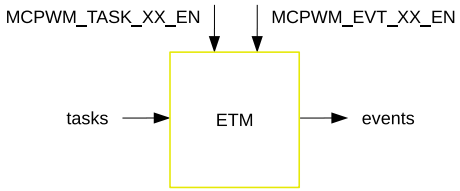


Figure 24.3-4. ETM Module

The module responds to received tasks, or generate and output events. Each event and task can be enabled independently.

24.3.2 PWM Timer Module

MCPWM has three PWM timer modules. Any of them can determine the necessary event timing for any of the three PWM operator modules. By using the synchronization signals from the GPIO matrix, built-in synchronization logic allows multiple PWM timer modules in one or more MCPWM peripherals to work together as a system.

24.3.2.1 Configurations of the PWM Timer Module

Users can configure the following functions of the PWM timer module:

- Control how often events occur by specifying the PWM timer frequency or period.
- Configure a particular PWM timer to synchronize phases with other PWM timers or modules.
- Configure the following timer counting modes: count-up, count-down, count-up-down.
- Change the rate of the PWM timer clock (PT_CLK) with a prescaler. Each timer has its own prescaler configured with MCPWM_TIMER*n*_PRESCALE of the register MCPWM_TIMER0_CFG0_REG. The PWM timer increments or decrements at a slower pace, depending on the setting of this field. The new MCPWM_TIMER*n*_PRESCALE configuration value will take effect when the timer stops and starts counting again.

24.3.2.2 PWM Timer's Working Modes and Timing Event Generation

The PWM timer has three working modes, selected by the PWM*x* timer mode field:

- Count-Up Mode:
  The PWM timer increments from zero until reaching the value configured in the period field. Once done, the PWM timer returns to zero and starts increasing again. PWM period = the value of the period field + 1.
  Note: The period field is MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2), i.e., MCPWM_TIMER0_PERIOD, MCPWM_TIMER1_PERIOD, MCPWM_TIMER2_PERIOD.
- Count-Down Mode:
  The PWM timer decrements to zero, starting from the value configured in the period field. Once done, the PWM timer returns to the period value and starts decrementing again. In this case, the PWM period = the value of period field + 1.
- Count-Up-Down Mode:
  This is a combination of the two modes mentioned above. The PWM timer starts increasing from zero until the period value is reached. Then, the timer decreases back to zero. The PWM timer cycles incrementally and decrementally in this mode. The PWM period = the value of the period field × 2.

Figures 24.3-5 to 24.3-8 show PWM timer waveforms in different modes, including timer behavior during synchronization events. In Count-Up mode, the counting direction after synchronization is always counting up. In Count-Down mode, the counting direction after synchronization is always counting down. In Count-Up-Down Mode, the counting direction after synchronization can be chosen by setting the MCPWM_TIMER*n*_PHASE_DIRECTION.

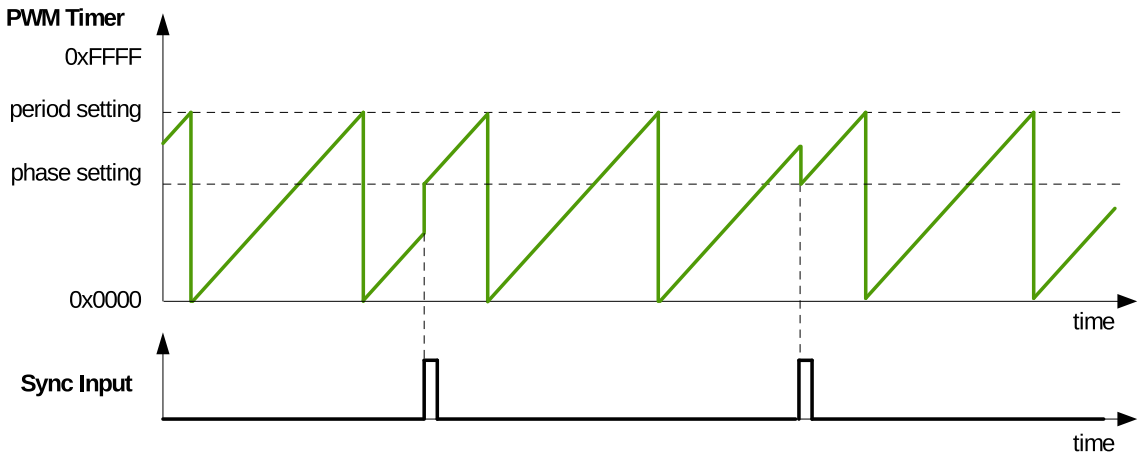


Figure 24.3-5. Count-Up Mode Waveform

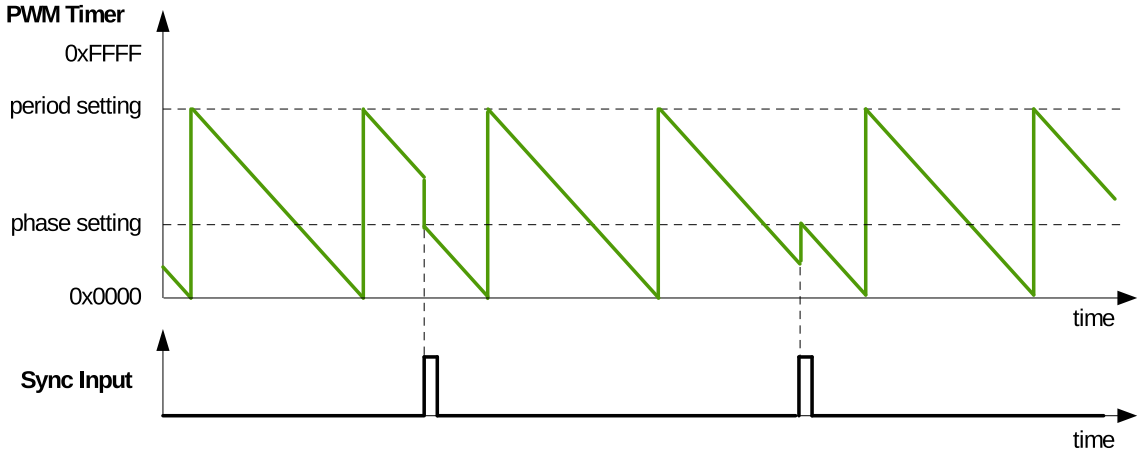


Figure 24.3-6. Count-Down Mode Waveforms

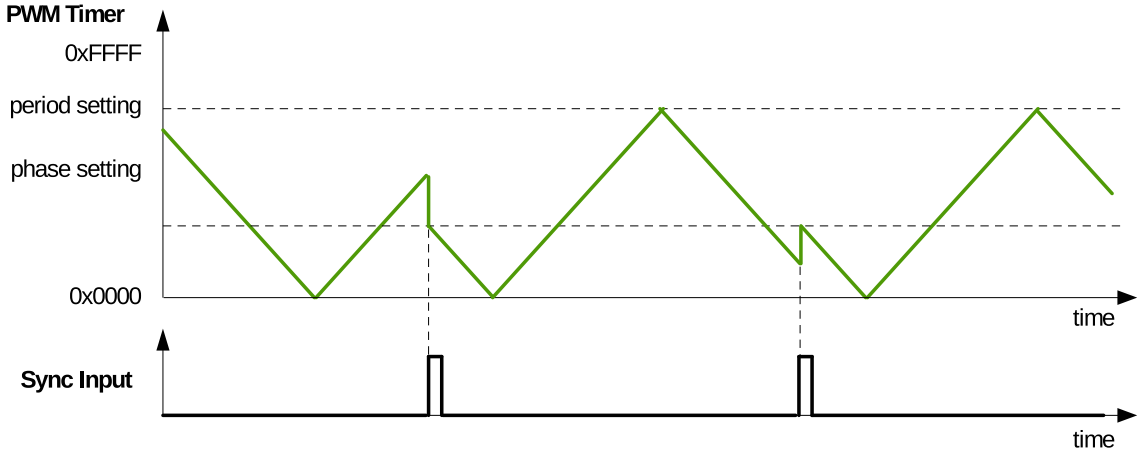


Figure 24.3-7. Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

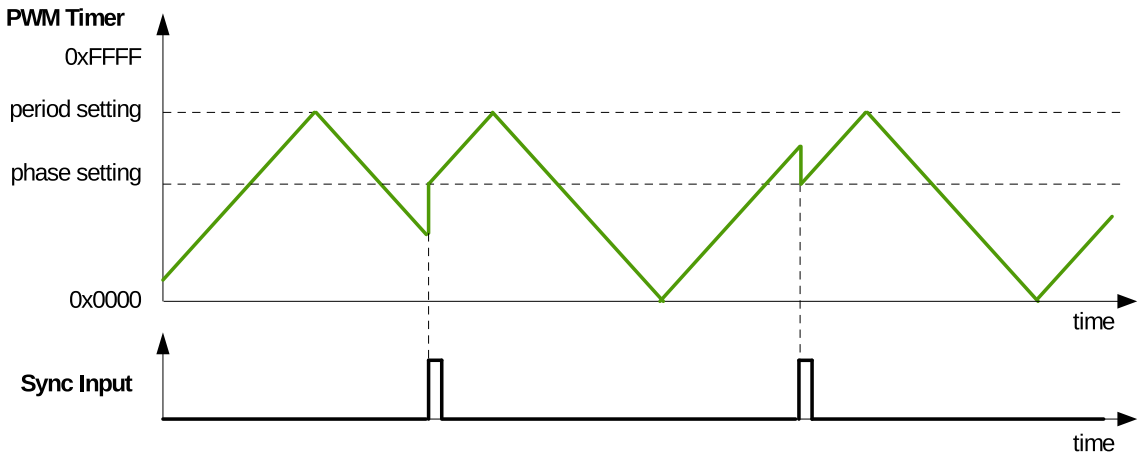


Figure 24.3-8. Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

When the PWM timer is running, it generates the following timing events periodically and automatically:

- UTEP: The timing event generated when the PWM timer's value is equal to the value of the period field (MCPWM_TIMER*n*_PERIOD) and when the PWM timer is increasing.
- UTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is increasing.
- DTEP: The timing event generated when the PWM timer's value equals the value of the period field (MCPWM_TIMER*n*_PERIOD) and when the PWM timer is decreasing.
- DTEZ: The timing event generated when the PWM timer's value equals zero and when the PWM timer is decreasing.

Figures 24.3-9 to 24.3-11 show the timing waveforms of U/DTEP and U/DTEZ.

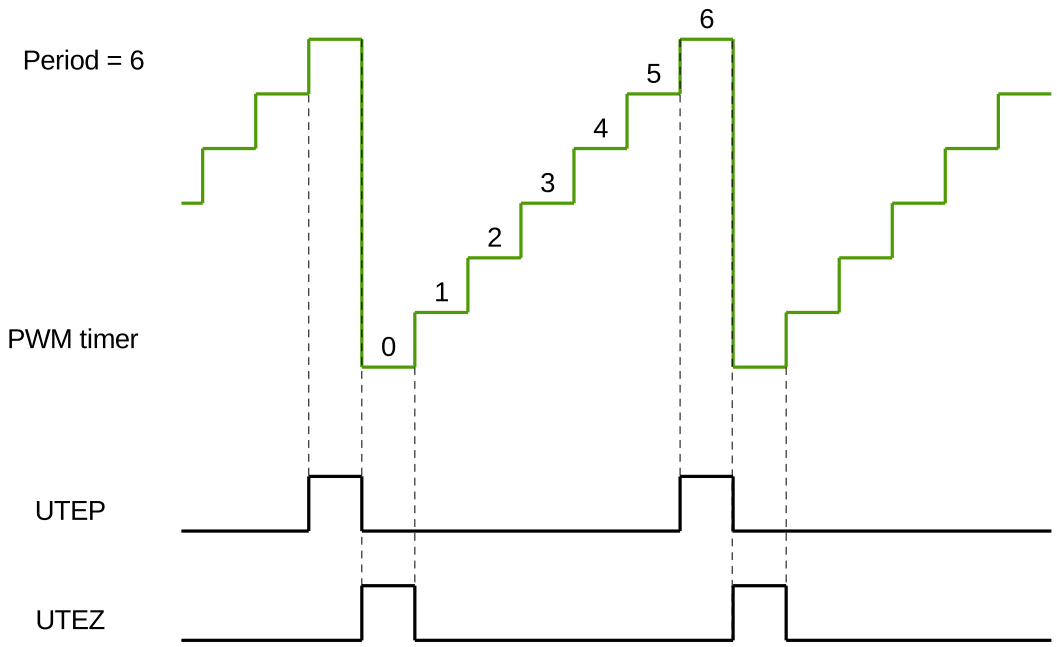


Figure 24.3-9. UTEP and UTEZ Generation in Count-Up Mode

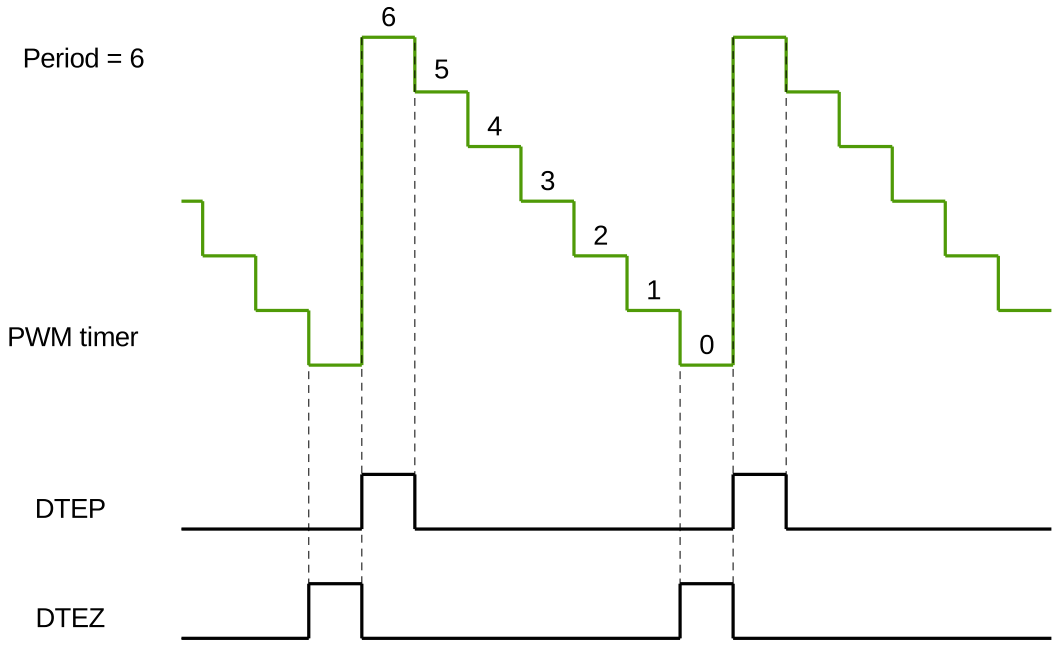


Figure 24.3-10. DTEP and DTEZ Generation in Count-Down Mode

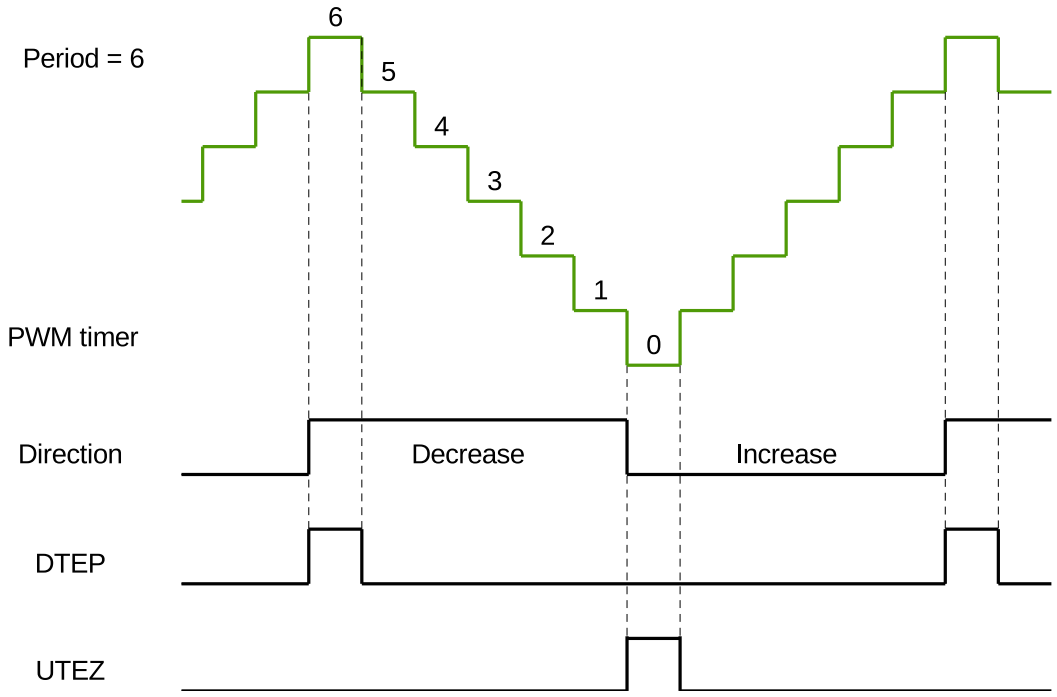


Figure 24.3-11. DTEP and UTEZ Generation in Count-Up-Down Mode

Please note that in the Count-Up-Down Mode, when the counting direction is increasing, the timer range is [0, period value - 1], and when the counting direction is decreasing, the timer range is [period value, 1]. That is, in this mode, when synchronizing the timer to 0, decreasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 1. Similarly, when synchronizing the timer to period value, increasing counting direction will be illegal, namely, MCPWM_TIMER*n*_PHASE_DIRECTION cannot be set to 0. Therefore, when the timer is synchronized to 0, the counting direction can only be increasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 0. When the timer is synchronized to the period value, the counting direction can only be decreasing, and MCPWM_TIMER*n*_PHASE_DIRECTION will be 1.

24.3.2.3 Shadow Register of PWM Timer

The PWM timer's period register and the PWM timer's clock prescaler register have shadow registers. The shadow registers can back up the values that are about to be written to the valid registers. It also supports writing the values saved into the active register at a specific moment of hardware synchronization. The functionality of both register types is as follows:

- Active Register: Directly responsible for controlling all actions performed by hardware.
- Shadow Register: Acts as a temporary buffer for a value to be written to the active register. At a specific, user-configured point in time, the value saved in the shadow register is copied to the active register. Before this happens, the content of the shadow register has no direct effect on the controlled hardware. This helps to prevent erroneous operation of the hardware, which may happen when a register is asynchronously modified by software. Both the shadow register and the active register have the same memory address. The software always writes into, or reads from the shadow register.

  The moment of updating the clock prescaler's active register is at the time when the timer starts operating. When MCPWM_GLOBAL_UP_EN is set to 1, the moment of updating the period active register can be selected by the following ways:

  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 0, the update will start immediately.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 1, the update can start when the PWM timer is equal to zero.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 2, the update can start when the PWM timer is synchronized.
  - By configuring the update method register MCPWM_TIMER*n*_PERIOD_UPMETHOD to 3, the update can start when the PWM timer is equal to zero or is synchronized.
  - Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers.

24.3.2.4 PWM Timer Synchronization and Phase Locking

The PWM modules adopt a flexible synchronization method. Each PWM timer has a synchronization input and a synchronization output. The synchronization input can be selected from three synchronization outputs of the three PWM timers and three synchronization signals from the GPIO matrix. The synchronization output can be generated when the PWM timer's value equals to period or zero, the synchronization input signal is active, or software synchronizes. Thus, each PWM timer can flexibly select the synchronization input for different phase synchronization methods. During synchronization, the PWM timer clock prescaler will reset its counter and restart dividing the clock in order to ensure the correctness of the timer.

24.3.3 PWM Operator Module

The PWM Operator module has the following functions:

- Generates a PWM signal pair, based on timing references obtained from the corresponding PWM timer.

- Each signal out of the PWM signal pair includes a specific pattern of dead time (i.e., adding delays to the rising and falling edges of the PWM signal).
- Superimposes a carrier on the PWM signal, if configured to do so.
- Handles response under fault conditions.

Figure 24.3-12 shows the block diagram of a PWM operator.

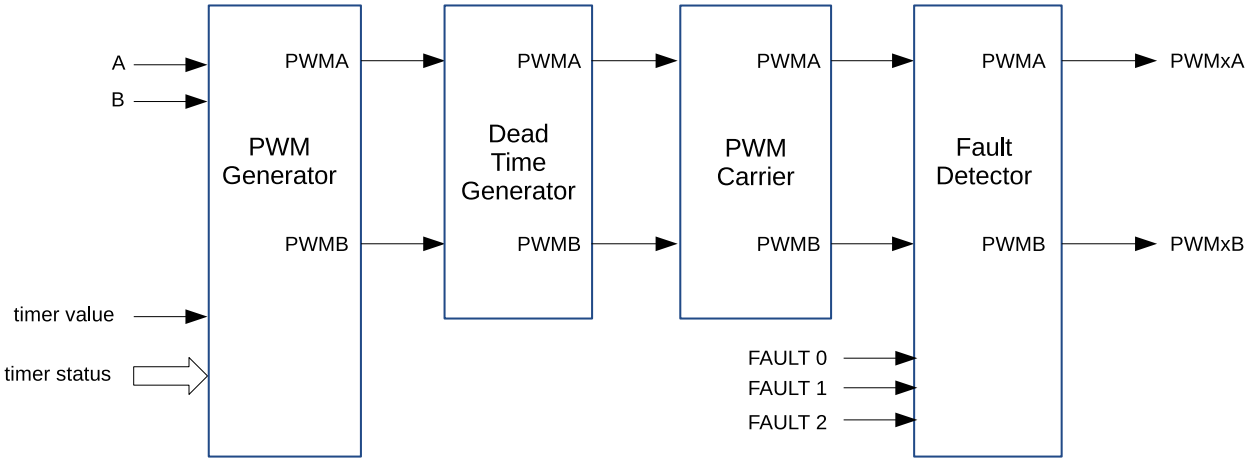


Figure 24.3-12. Block Diagram of A PWM Operator

24.3.3.1 PWM Generator Module

Purpose of the PWM Generator Module

In this module, important timing events are generated or imported. The events are then converted into specific actions to generate the desired waveforms at the PWM*x*A and PWM*x*B outputs.

The PWM generator module performs the following actions:

- Generation of timing events based on time stamps configured using the A and B registers. Events happen when the following conditions are met (for the configuration of registers A and B, see section 24.3.3.1):
  - UTEA: the PWM timer is counting up and its value is equal to register A.
  - UTEB: the PWM timer is counting up and its value is equal to register B.
  - DTEA: the PWM timer is counting down and its value is equal to register A.
  - DTEB: the PWM timer is counting down and its value is equal to register B.
- Generation of U/DT0, U/DT1 timing events based on fault or synchronization events.
  - UT0: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*n*_T0_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T0_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T0_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T0_SEL is set to 3).
  - UT1: the PWM timer is counting up and FAULT0 detected (field MCPWM_GEN*n*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T1_SEL is set to 3).

  - DT0: the PWM timer is counting down and FAULT0 detected (field MCPWM_GEN*n*_T0_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T0_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T0_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T0_SEL is set to 3).
  - DT1: the PWM timer is counting down and FAULT0 detected(field MCPWM_GEN*n*_T1_SEL is set to 0) or FAULT1 detected (field MCPWM_GEN*n*_T1_SEL is set to 1) or FAULT2 detected (field MCPWM_GEN*n*_T1_SEL is set to 2) or synchronized (field MCPWM_GEN*n*_T1_SEL is set to 3).
- Management of priority when these timing events occur concurrently.
- Generation of set, clear, and toggle actions, based on the timing events.
- Controlling of the PWM duty cycle, depending on the configuration of the PWM generator module.
- Handling of new time stamp values, using shadow registers to prevent glitches in the PWM cycle.

Shadow Register of PWM Generator

The time stamp registers A and B used by the hardware have shadow registers, which are registers MCPWM_GEN*n*_A_REG and MCPWM_GEN*n*_B_REG. Shadowing provides a way of updating registers in sync with the hardware.

When MCPWM_GLOBAL_UP_EN is set to 1, the shadow registers can be written to the active register at a specified time. The update method field for MCPWM_GEN*n*_A_REG and MCPWM_GEN*n*_B_REG is MCPWM_GEN*n*_CFG_UPMETHOD. Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For a description of the shadow registers, please see Section 24.3.2.3.

Timing Events

For convenience, all timing signals and events are summarized in Table 24.3-2.

Table 24.3-2. Timing Events Used in PWM Generator

Signal	Event Description	PWM Timer Operation
DTEP	PWM timer value is equal to the period register value	PWM timer counts down
DTEZ	PWM timer value is equal to zero	
DTEA	PWM timer value is equal to register A	
DTEB	PWM timer value is equal to register B	
DT0 event	Based on fault or synchronization events	
DT1 event	Based on fault or synchronization events	
UTEP	PWM timer value is equal to the period register value	PWM timer counts up
UTEZ	PWM timer value is equal to zero	
UTEA	PWM timer value is equal to register A	
UTEB	PWM timer value is equal to register B	
UT0 event	Based on fault or synchronization events	
UT1 event	Based on fault or synchronization events	
Software-force event	Software-initiated asynchronous event	N/A

The purpose of a software-force event is to impose non-continuous or continuous changes on the PWM*x*A and PWM*x*B outputs. The change is done asynchronously. Software-force control is handled by the

MCPWM_GEN*n*_FORCE_REG registers.

The selection and configuration of T0/T1 in the PWM generator module is independent of the configuration of fault events in the fault handler module. A particular trip event may or may not be configured to cause trip action in the fault handler submodule, but the same event can be used by the PWM generator to trigger T0/T1 for controlling PWM waveforms.

It is important to know that when the PWM timer is in Count-Up-Down Mode. It will always decrement after a TEP event, and increment after a TEZ event. So, when the PWM timer is in Count-Up-Down Mode, DTEP and UTEZ events will occur, while UTEP and DTEZ events never occurs.

The PWM generator can handle multiple events at the same time. Events are prioritized by the hardware and relevant details are provided in Table 24.3-3 and Table 24.3-4. 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 24.3-3. Timing Events Priority When PWM Timer Increments

Priority Level	Event
1 (highest)	Software-forced event
2	UTEP
3	UT0
4	UT1
5	UTEB
6	UTEA
7 (lowest)	UTEZ

Table 24.3-4. 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 24.3-9, so their action on PWM signals will not interrupt each other. When the PWM timer is in Count-Up-Down Mode, UTEP will not occur.
2. DTEP and DTEZ do not happen simultaneously. When the PWM timer is in Count-Down Mode, DTEZ will always happen one cycle earlier than DTEP, as demonstrated in Figure 24.3-10, so their action on PWM

signals will not interrupt each other. When the PWM timer is in Count-Up-Down Mode, DTEZ will not occur.

PWM Signal Generation

The PWM generator module controls the behavior of outputting PWM*x*A and PWM*x*B when a particular timing event occurs. The timing events are further qualified by the PWM timer's counting mode (increment or decrement). Knowing the counting mode, the module may then perform an independent action at each stage of the PWM timer counting up or down.

The following actions may be configured on PWM*x*A and PWM*x*B outputs:

- Set High: Set the output of PWM*x*A or PWM*x*B to a high level.
- Clear Low: Clear the output of PWM*x*A or PWM*x*B by setting it to a low level.
- Toggle: Change the current output level of PWM*x*A or PWM*x*B to the opposite value. If it is currently pulled up, then pull it down, or vice versa.
- Do Nothing: Keep both outputs PWM*x*A and PWM*x*B unchanged. In this state, interrupts can still be triggered.

Actions on outputs is configured by using registers MCPWM_GEN*n*_A_REG and MCPWM_GEN*n*_B_REG. So, the action to be taken on each output is set independently. Also, there is great flexibility in selecting actions to be taken on a given output based on events. More specifically, any event listed in Table 24.3-2 can operate on either output of PWM*x*A or PWM*x*B. To check out registers for particular generator 0, 1, or 2, please refer to the register description in Section 24.5.

Waveforms for Common Configurations

In the configuration of various waveforms below, period refers to the configuration value stored in the period register of the PWM timer selected by the PWM operator (i.e. MCPWM_TIMER*n* _PERIOD (*x* = 0, 1, 2)). For configuration methods, please refer to the section 24.3.2. A and B denote the time stamp registers A and B used by the hardware. For register configuration, please refer to the section 24.3.3.1. UTEA/B, UTEZ/P, DTEA/B, DTEZ/P, DTO/1, and UTO/1 represent different timing events of the PWM generator. For event definition, please refer to the section 24.3.3.1. The upward arrow, downward arrow, and T mark signify setting the PWM waveform high, setting it low, toggling it when the timing event occurs. See section 24.3.3.1 for configuration. PWM*x*A and PWM*x*B are the output signals of PWM generator*x*.

Figure 24.3-13 presents the symmetric PWM waveform generated when the PWM timer is in Count-Up-Down mode. DC 0%–100% modulation can be calculated via the formula below:

$$Duty = (Period - A) \div Period$$

If A matches the PWM timer value and the PWM timer is incrementing, then the PWM output is pulled up. If A matches the PWM timer value while the PWM timer is decrementing, then the PWM output is pulled low.

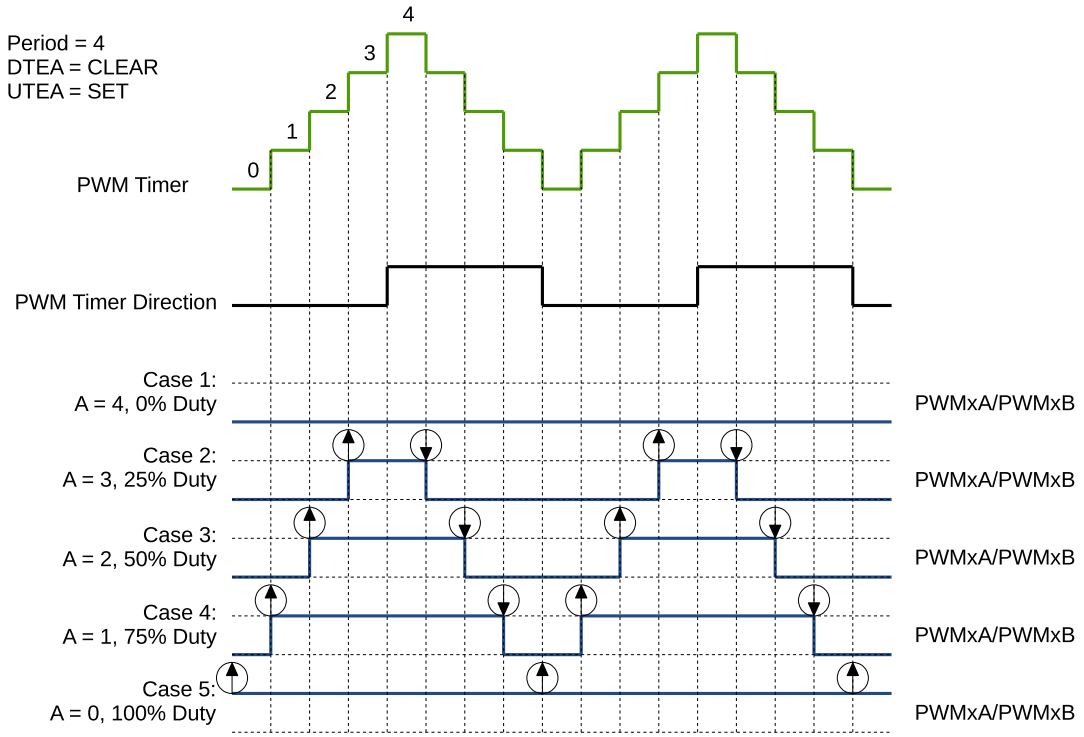


Figure 24.3-13. Symmetrical Waveform in Count-Up-Down Mode

The PWM waveforms in Figures 24.3-14 to 24.3-17 show some common PWM generator configurations.

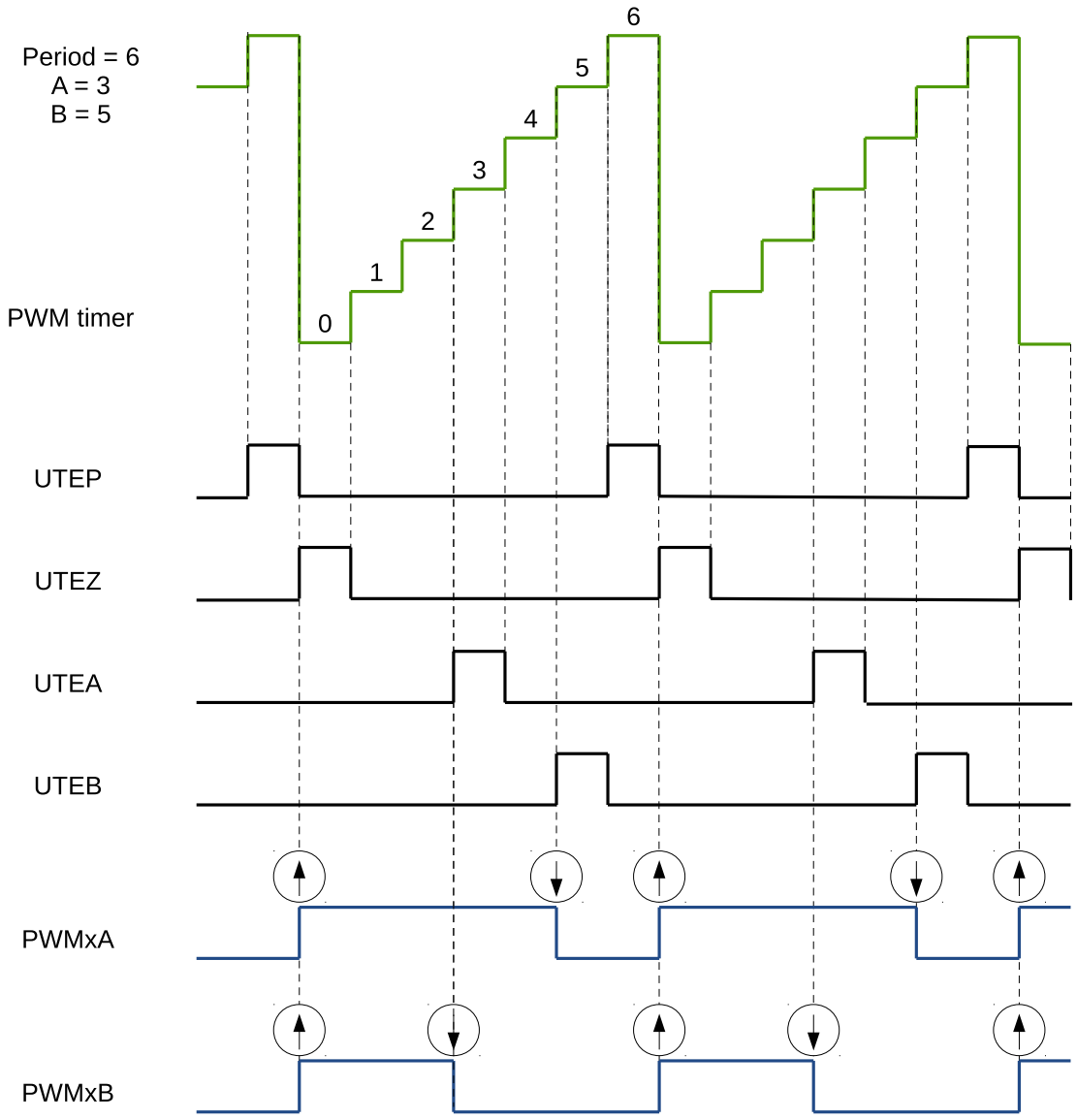


Figure 24.3-14. Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by B, active high and proportional to B.
The duty modulation for PWM*x*B is set by A, active high and proportional to A.

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

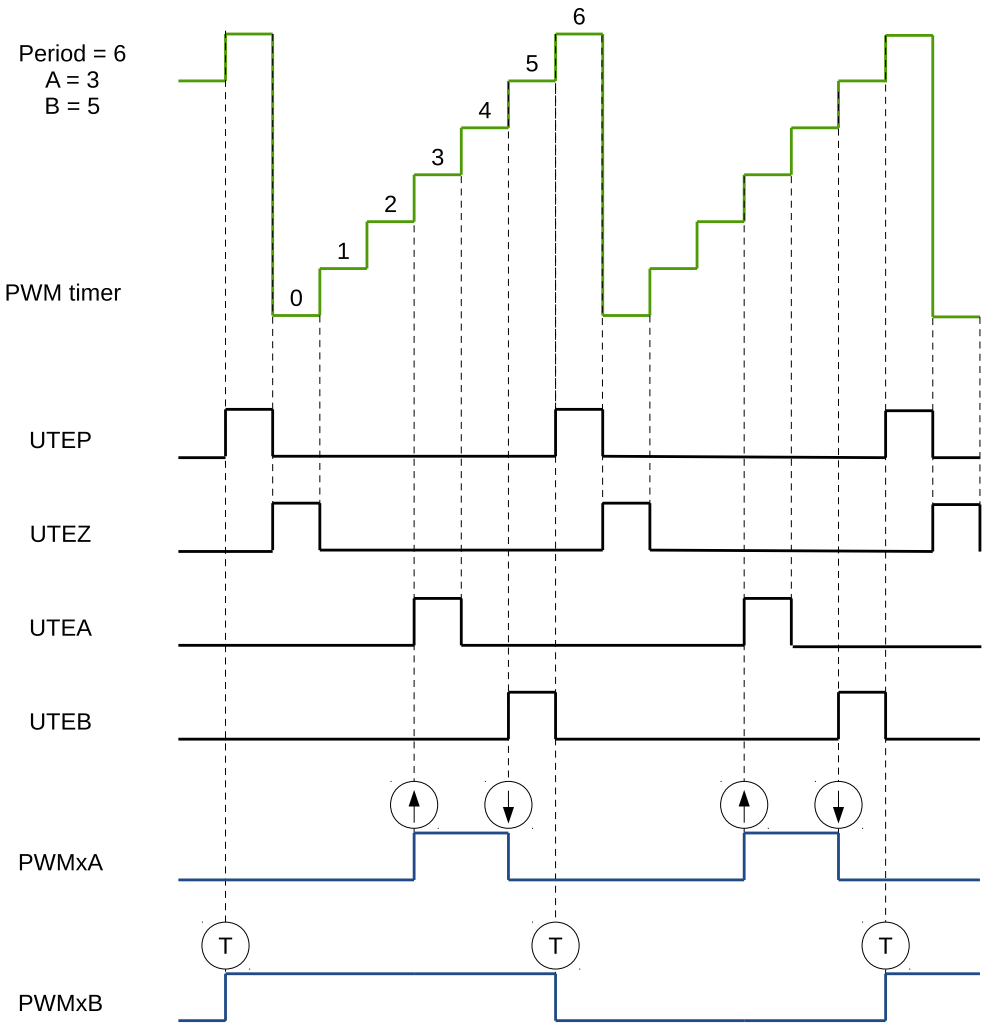


Figure 24.3-15. Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A

Pulses may be generated anywhere within the PWM cycle (zero to period).

PWM*x*A's high time duty is proportional to (B – A).

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

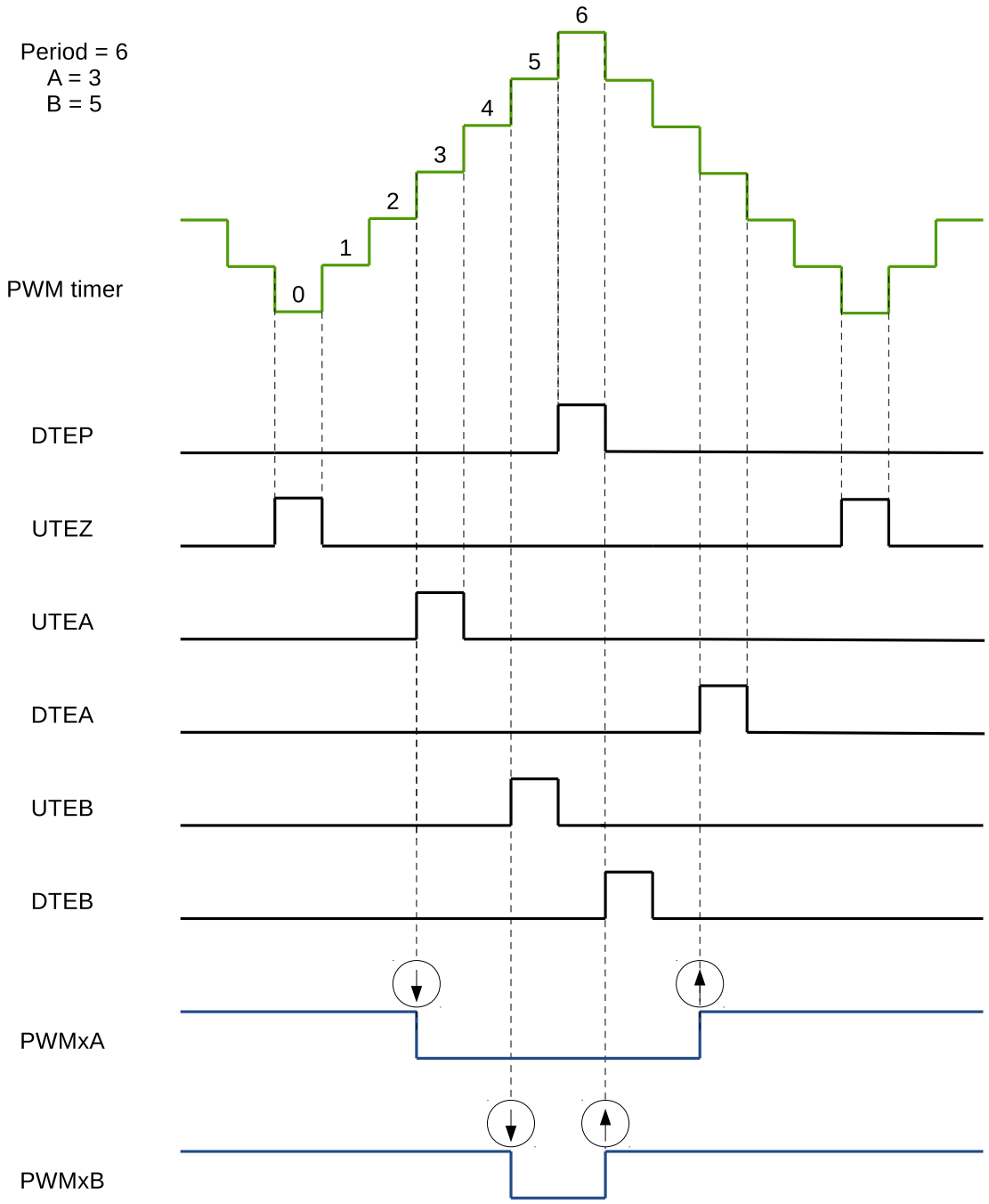


Figure 24.3-16. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by A, active high and proportional to A.
The duty modulation for PWM*x*B is set by B, active high and proportional to B.
Outputting PWM*x*A and PWM*x*B can drive separate switches.

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

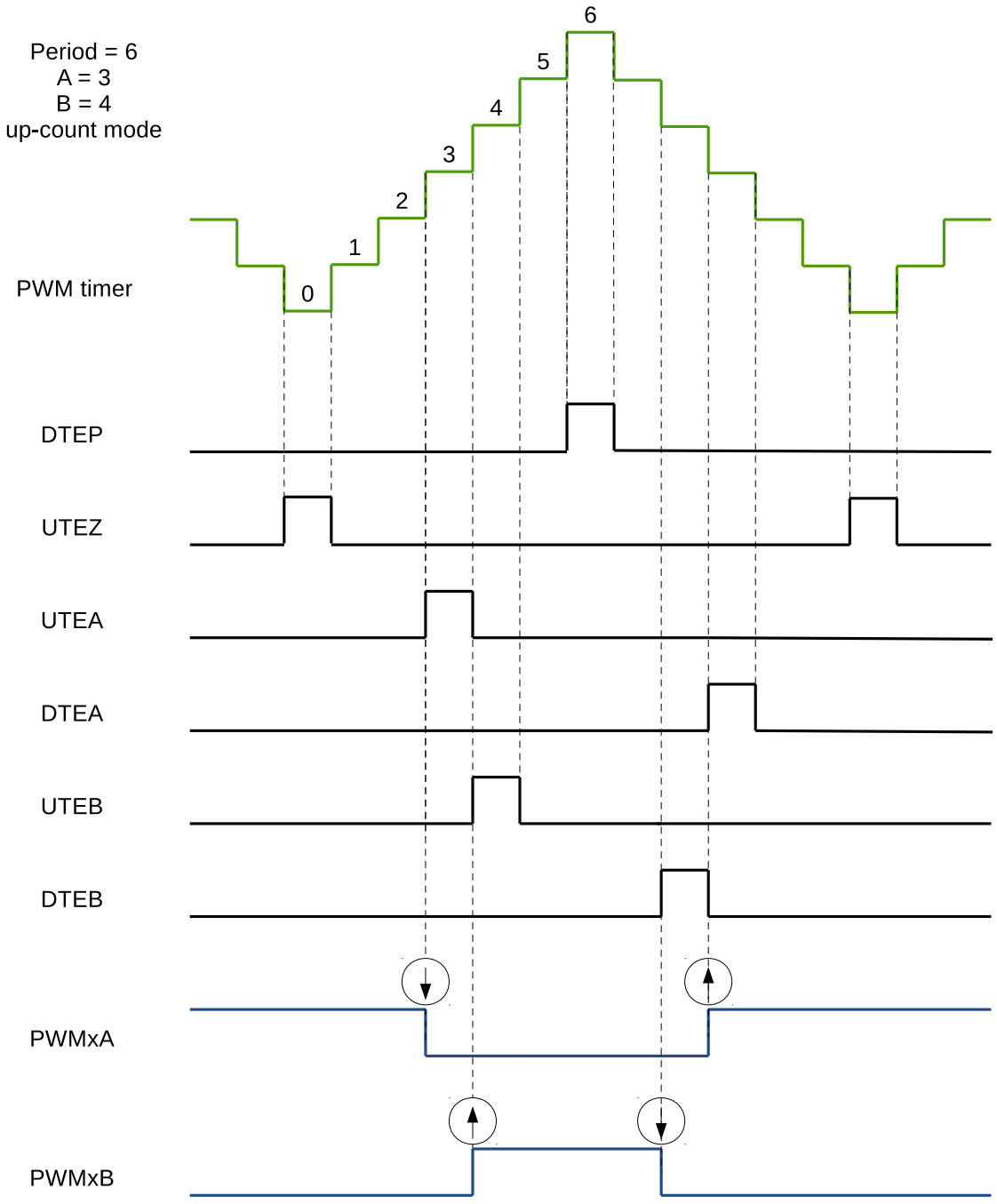


Figure 24.3-17. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary

The duty modulation of PWM*x*A is set by A, is active high and proportional to A.
The duty modulation of PWM*x*B is set by B, is active low and proportional to B.
Outputs PWM*x* can drive upper/lower (complementary) switches.
Dead time = B − A. Edge placement is configurable by software. Dead time generator module supports configuring edge delay methods when required.

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

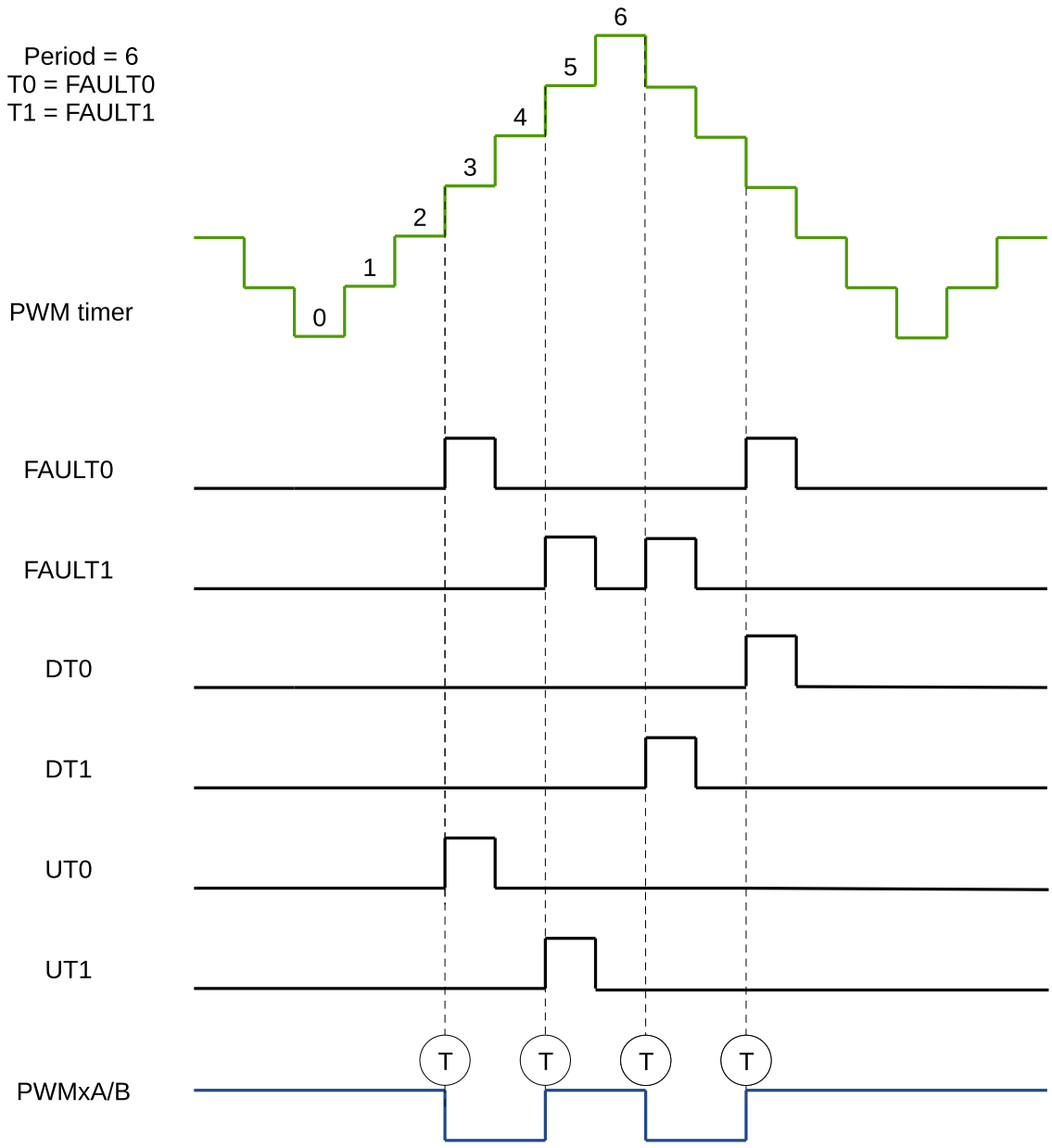


Figure 24.3-18. Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B

Figure 24.3-18 shows a waveform when UT0/1 and DT0/1 events are generated. In this example, T0 selects FAULT0 and T1 selects FAULT1. The events selected by T0 and T1 can be configured independently, these events can be FAULT0, FAULT1, FAULT2 or synchronous. For detailed configuration, see section 24.3.3.1.

Software-Force Events

There are two types of software-force events inside the PWM generator:

- Non-continuous-immediate (NCI) software-force events: Such types of events are immediately effective on PWM outputs when triggered by software. The forcing is non-continuous, which means the next active timing events will be able to alter the PWM outputs.
- Continuous (CNTU) software-force events: Such types of events are continuous. The forced PWM

outputs will continue until they are released by software. The events' triggers are configurable. They can be configured to be timing events or immediate events.

For NCI software-force events, set register MCPWM_GEN*n*_A_NCIFORCE or MCPWM_GEN*n*_B_NCIFORCE to 1 and then to 0 to trigger the NCI software-forced events of PWM*x*A or PWM*x*B. Configure register MCPWM_GEN*n*_A_NCIFORCE_MODE or MCPWM_GEN*n*_B_NCIFORCE_MODE to specify the operation on the PWM*x*A or PWM*x*B waveform when the NCI software-force event occurs. Configuring the registers to 0 or 3 means no operation, 1 means low level, and 2 means high level.

For CNTU software-force events, set register MCPWM_GEN*n*_CNTUFORCE_UPMETHOD to specify the trigger method of CNTU software-force events. Setting all bits to 0 means trigger CNTU software-force events immediately, setting bit 0 to 1 means the TEZ event triggers CNTU software-force events, setting bit 1 to 1 means the TEP event triggers CNTU software-force events, setting bit 2 to 1 means the TEA event triggers CNTU software-force events, setting bit 3 to 1 means the TEB event triggers CNTU software-force events, setting bit 4 to 1 means CNTU software-force events triggered when the PWM timer selected by the PWM generator is synchronized, and setting bit 5 to 1 means the CNTU software-forced events will not be triggered. Configure register MCPWM_GEN*n*_A_CNTUFORCE_MODE or MCPWM_GEN*n*_B_CNTUFORCE_MODE to specify the operation on the PWM*x*A or PWM*x*B waveform when the CNTU software-force events occur. Configuring the registers to 0 or 3 means no operation, 1 means low level, and 2 means high level.

Figure 24.3-19 shows a waveform of NCI software-force events. NCI events are used to force PWM*x*A output low. Forcing on PWM*x*B is disabled in this case.

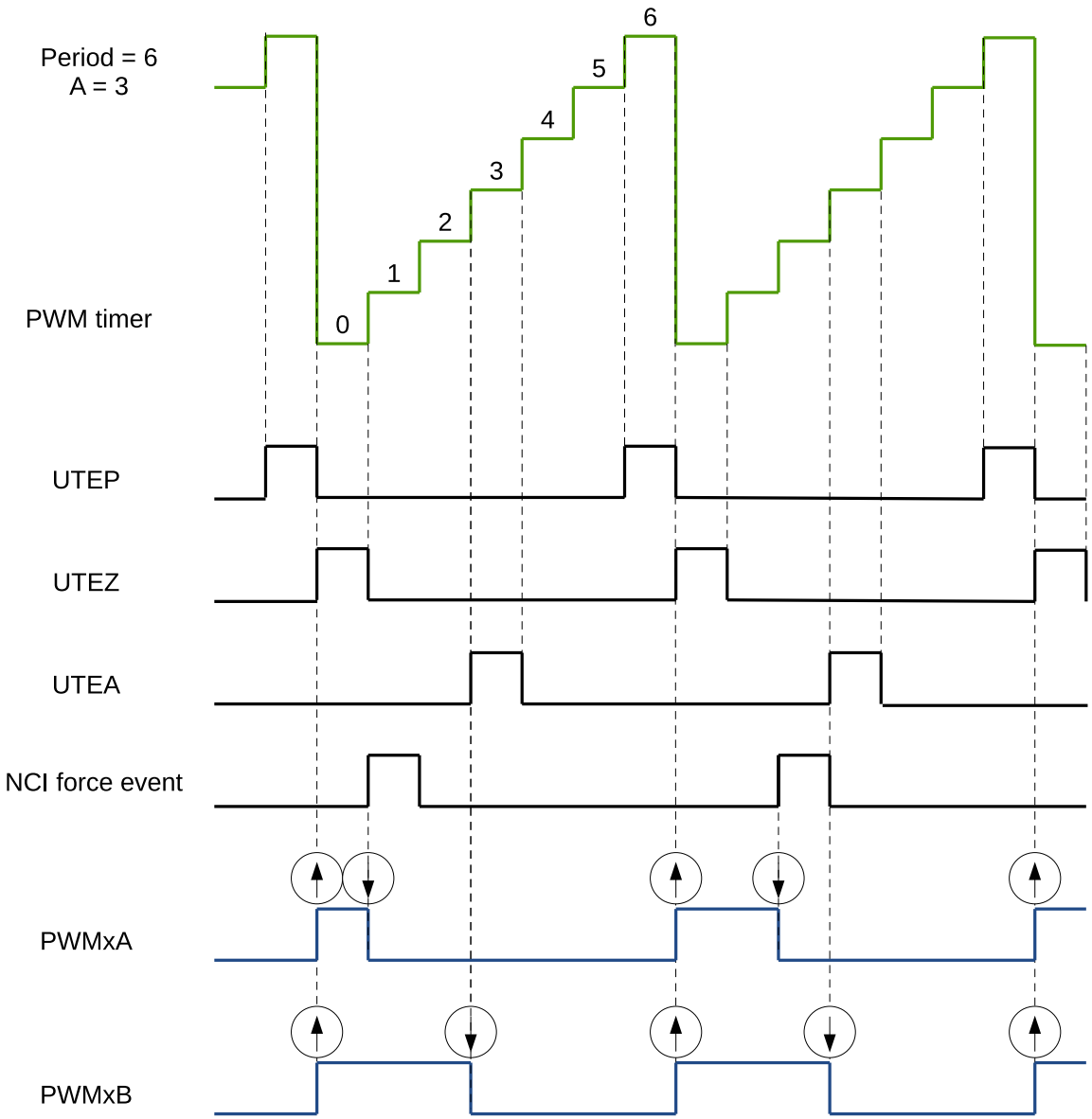


Figure 24.3-19. Example of an NCI Software-Force Event on PWM*x*A

Figure 24.3-20 shows a waveform of CNTU software-force events. TEZ events are selected as triggers for CNTU software-force events. CNTU is used to force the PWM*x*B output low. Forcing on PWM*x*A is disabled.

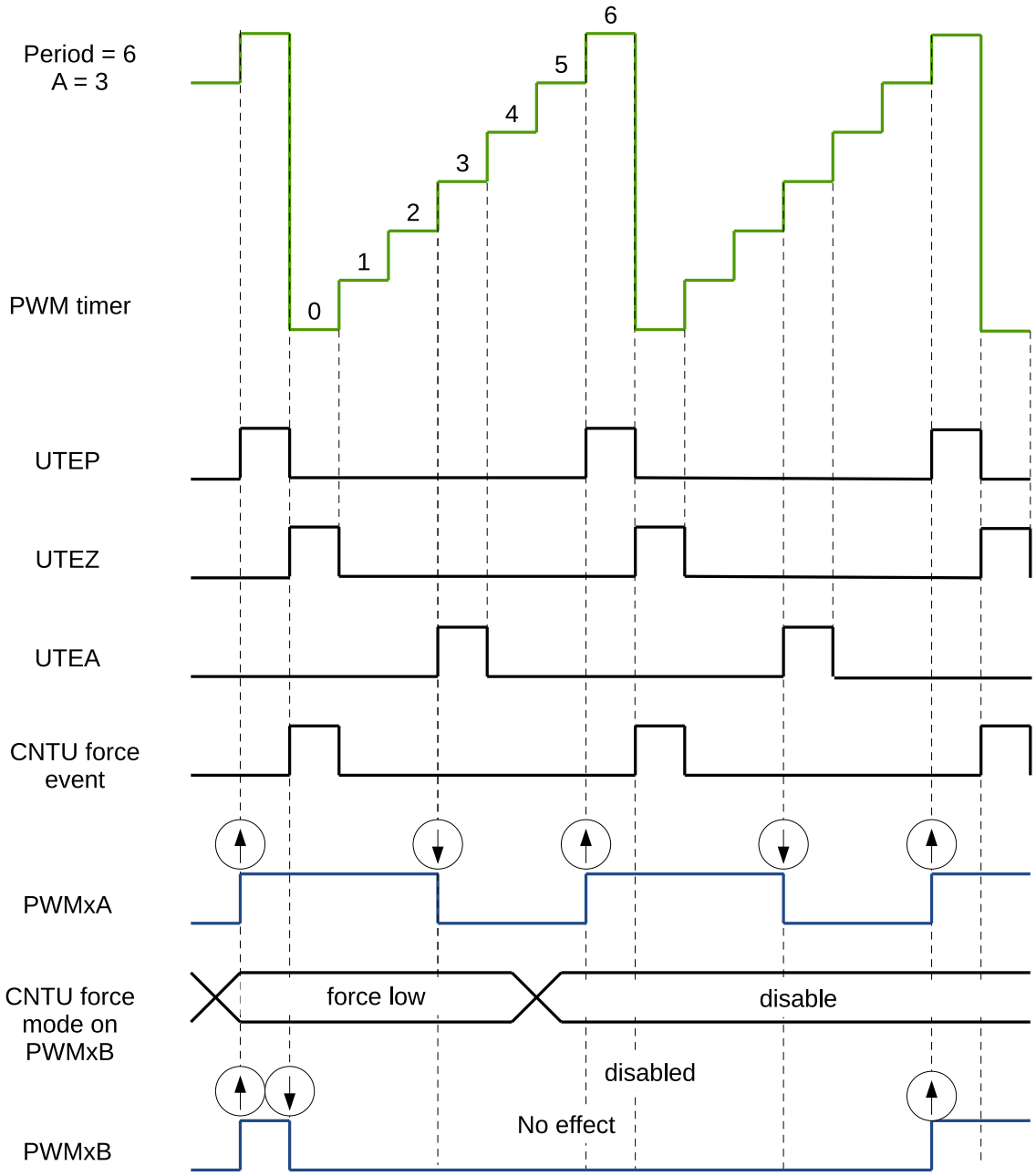


Figure 24.3-20. Example of a CNTU Software-Force Event on PWM*x*B

24.3.3.2 Dead Time Generator Module

Purpose of the Dead Time Generator Module

Section 24.3.3.1 introduced several options to generate signals on PWM*x*A and PWM*x*B outputs, with a specific placement of signal edges. The required dead time is obtained by altering the edge placement between signals and by setting the signal's duty cycle. Another option to control the dead time is to use a specialized module – Dead Time Generator.

The key functions of the Dead Time Generator module are as follows:

- Generating output signal pairs (PWM*x*A and PWM*x*B) with a dead time from a common source, which can be either the input signal PWM*x*A or PWM*x*B.
- Creating a dead time by adding delay to signal edges:
  - Rising edge delay (RED)
  - Falling edge delay (FED)
- Can generate PWM signal in various format. The typical dead time configurations are:
  - Active high complementary (AHC)
  - Active low complementary (ALC)
  - Active high (AH)
  - Active low (AL)
- This module may also be bypassed, if the dead time is configured directly in the generator module.

Shadow Register of Dead Time Generator

Delay registers RED and FED are shadowed with registers MCPWM_DT*n*_RED_CFG_REG and MCPWM_DT*n*_FED_CFG_REG. When MCPWM_GLOBAL_UP_EN is set to 1, the values saved in the shadow registers can be written to the active register at the specified time. The update method register for MCPWM_DT*n*_RED_CFG_REG is MCPWM_DB*n*_RED_UPMETHOD. The update method register for MCPWM_DT*n*_FED_CFG_REG is MCPWM_DB*n*_FED_UPMETHOD. The Software can also trigger a globally forced update bit MCPWM_GLOBAL_FORCE_UP which will prompt all registers in the module to be updated according to shadow registers. For the description of shadow registers, please see section 24.3.2.3.

Highlights for Operation of the Dead Time Generator

Options for setting up the dead time module are shown in Figure 24.3-21.

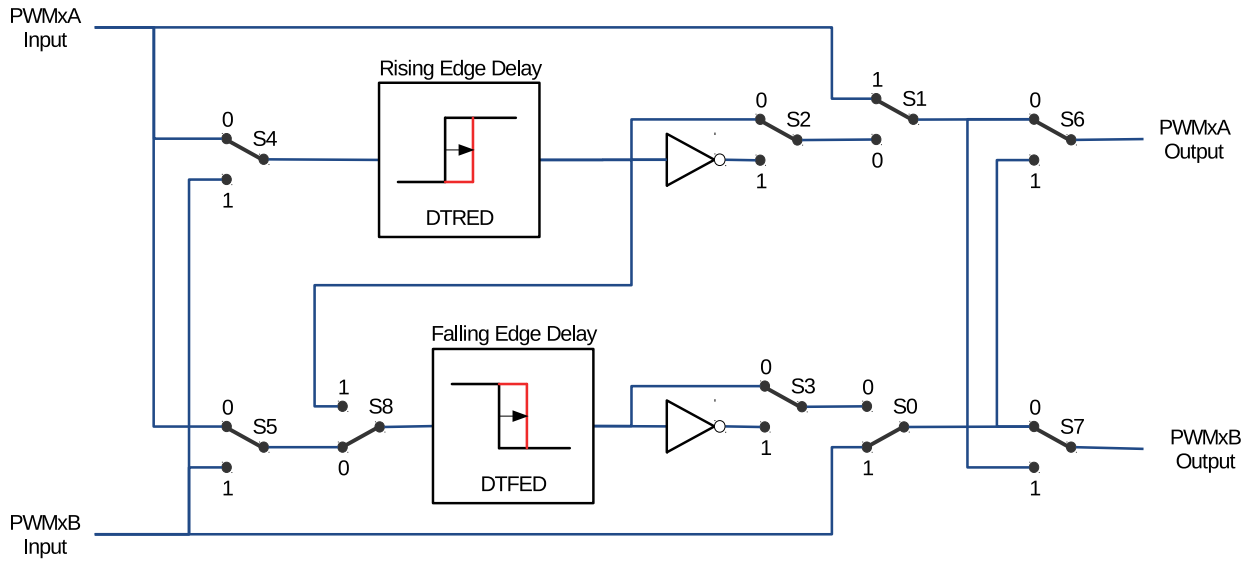


Figure 24.3-21. Options for Setting up the Dead Time Generator Module

S0-S8 in the figure above are switches controlled by fields in register MCPWM_DT*n*_CFG_REG shown in Table 24.3-5.

Table 24.3-5. Dead Time Generator Switches Control Fields

Switch	Field
S0	MCPWM_DB*n*_B_OUTBYPASS
S1	MCPWM_DB*n*_A_OUTBYPASS
S2	MCPWM_DB*n*_RED_OUTINVERT
S3	MCPWM_DB*n*_FED_OUTINVERT
S4	MCPWM_DB*n*_RED_INSEL
S5	MCPWM_DB*n*_FED_INSEL
S6	MCPWM_DB*n*_A_OUTSWAP
S7	MCPWM_DB*n*_B_OUTSWAP
S8	MCPWM_DB*n*_DEB_MODE

All switch combinations are supported, but not all of them represent the typical modes of use. Table 24.3-6 documents some typical dead time configurations. In these configurations, the position of S4 and S5 sets PWM*x*A as the common source of both falling edge delay (FED) and rising edge delay (RED). The modes presented in table 24.3-6 may be categorized as follows:

- **Mode 1: Bypass delays on both FED and RED**
  In this mode, the dead time module is disabled. Signals of PWM*x*A and PWM*x*B pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**

These four modes represent typical configurations of polarity and should cover the active-high/low modes in available industry power switch gate drivers. The typical waveforms are shown in Figures 24.3-22 to 24.3-25.

- **Modes 6 and 7: Bypass delay on the falling edge (FED) or rising edge (RED)**
  In these two modes, either RED or FED is bypassed. As a result, the corresponding delay is not applied.

Table 24.3-6. 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 24.3-22	0	0	0	1
3	Active Low Complementary (ALC), see Figure 24.3-23	0	0	1	0
4	Active High (AH), see Figure 24.3-24	0	0	0	0
5	Active Low (AL), see Figure 24.3-25	0	0	1	1
6	PWM*x*A Output = PWM*x*A In (No Delay) PWM*x*B Output = PWM*x*A Input with Falling Edge Delay	0	1	0 or 1	0 or 1
7	PWM*x*A Output = PWM*x*A Input with Rising Edge Delay PWM*x*B Output = PWM*x*B Input with No Delay	1	0	0 or 1	0 or 1

Note:
For all the modes above, the position of the binary switches S4 to S8 is set to 0.

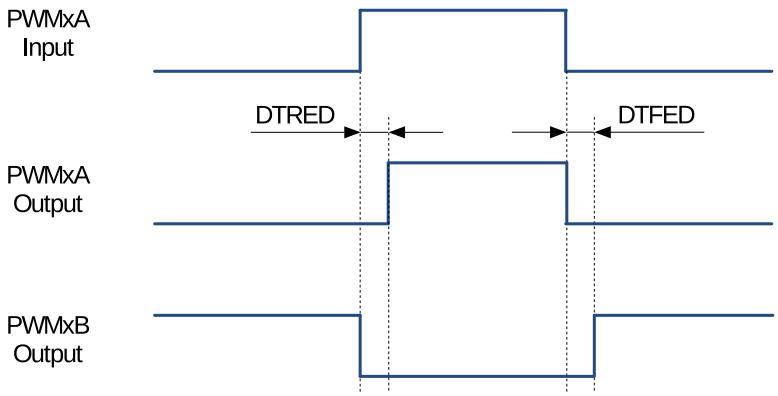


Figure 24.3-22. Active High Complementary (AHC) Dead Time Waveforms

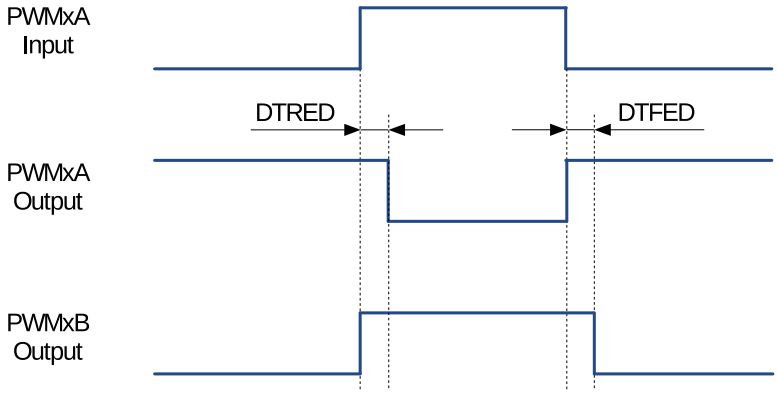


Figure 24.3-23. Active Low Complementary (ALC) Dead Time Waveforms

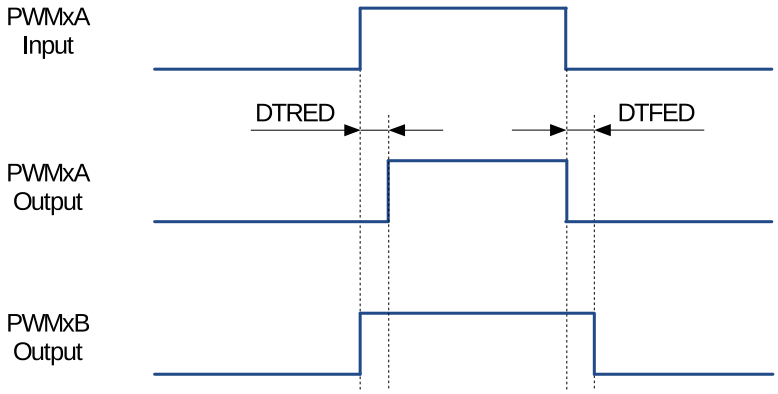


Figure 24.3-24. Active High (AH) Dead Time Waveforms

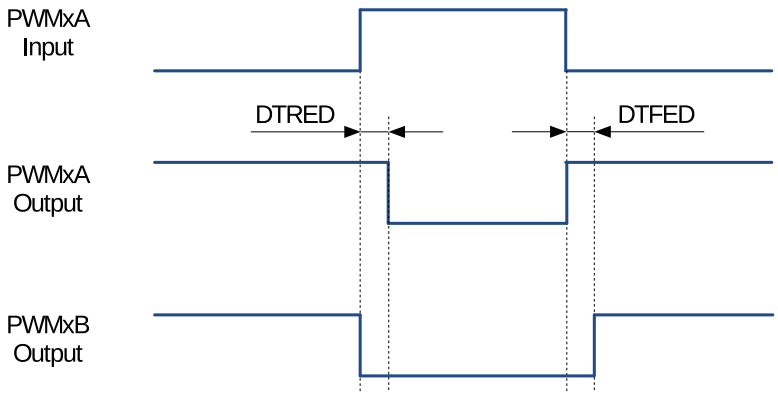


Figure 24.3-25. Active Low (AL) Dead Time Waveforms

RED and FED delays may be set up independently. The delay value is programmed using the 16-bit field MCPWM_DB*n*_RED and MCPWM_DB*n*_FED. The register value represents the number of clock (DT_clk) periods by which a signal edge is delayed. DT_clk can be selected from PWM_clk or PT_clk through register MCPWM_DB*n*_CLK_SEL.

To calculate the delay on the falling edge (FED) and rising edge (RED), use the following formulas:

$$FED = MCPWM\_DTn\_FED \times T_{DT\_clk}$$

$$RED = MCPWM\_DTn\_RED \times T_{DT\_clk}$$

24.3.3.3 PWM Carrier Module

The coupling of PWM output to a motor driver may need isolation with a transformer. Transformers deliver only AC signals, while the duty cycle of a PWM signal may range anywhere from 0% to 100%. The PWM carrier module passes such a PWM signal through a transformer by using a high frequency carrier to modulate the signal.

Function Overview

The following key characteristics of this module are configurable:

- Carrier frequency
- Pulse width of the first pulse
- Duty cycle of the second and the subsequent pulses
- Enabling/disabling the carrier function

Operational Highlights

The PWM carrier clock (PC_clk) is derived from PWM_CLK. The frequency and duty cycle are configured by the MCPWM_CHOPPER*n*_PRESCALE and MCPWM_CHOPPER*n*_DUTY bits in the MCPWM_CARRIER*n*_CFG_REG register. The purpose of one-shot pulses is to provide the high-energy impulse to reliably turn on the power switch. Subsequent pulses sustain the power-on status. The width of a one-shot pulse is configurable with the MCPWM_CHOPPER*n*_OSHTWTH field. Enabling/disabling of the carrier module is done with the MCPWM_CHOPPER*n*_EN bit.

Waveform Examples

Figure 24.3-26 shows an example of waveforms, where a carrier is superimposed on original PWM pulses. This figure does not show the first one-shot pulse and the duty-cycle control. Related details are covered in the following two sections.

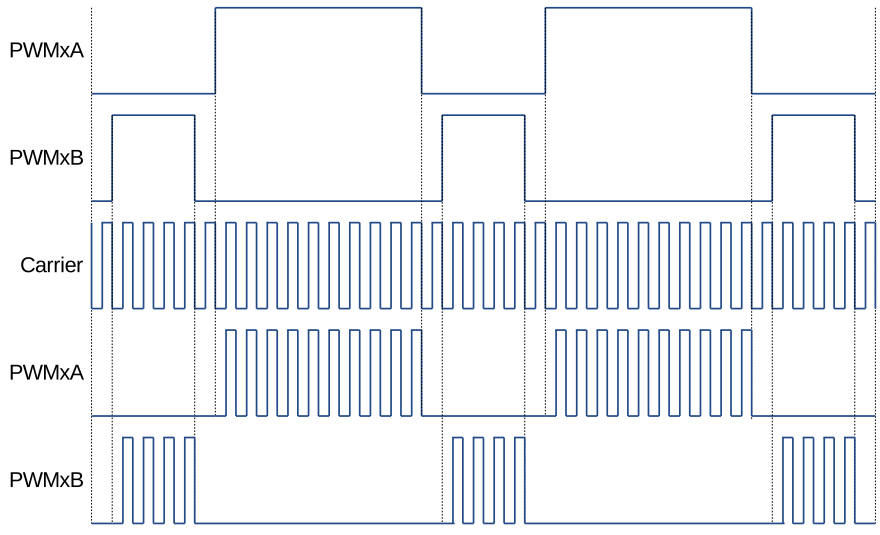


Figure 24.3-26. Example of Waveforms Showing PWM Carrier Action

One-Shot Pulse

The width of the first pulse can be configured to 16 different values, which can be calculated by the following equation:

$$T_{1stpulse} = T_{PWM\_clk} \times 8 \times (MCPWM\_CARRIERn\_PRESCALE + 1) \times (MCPWM\_CARRIERn\_OSHTWTH + 1)$$

Where:

- T_{PMW_clk} is the period of the PWM clock (PWM_clk).
- $(MCPWM\_CARRIERn\_OSHTWTH + 1)$ is the width of the first pulse (whose value ranges from 1 to 16).
- $(MCPWM\_CARRIERn\_PRESCALE + 1)$ is the PWM carrier clock's (PC_clk) prescaler value.

The first one-shot pulse and subsequent sustaining pulses are shown in Figure 24.3-27.

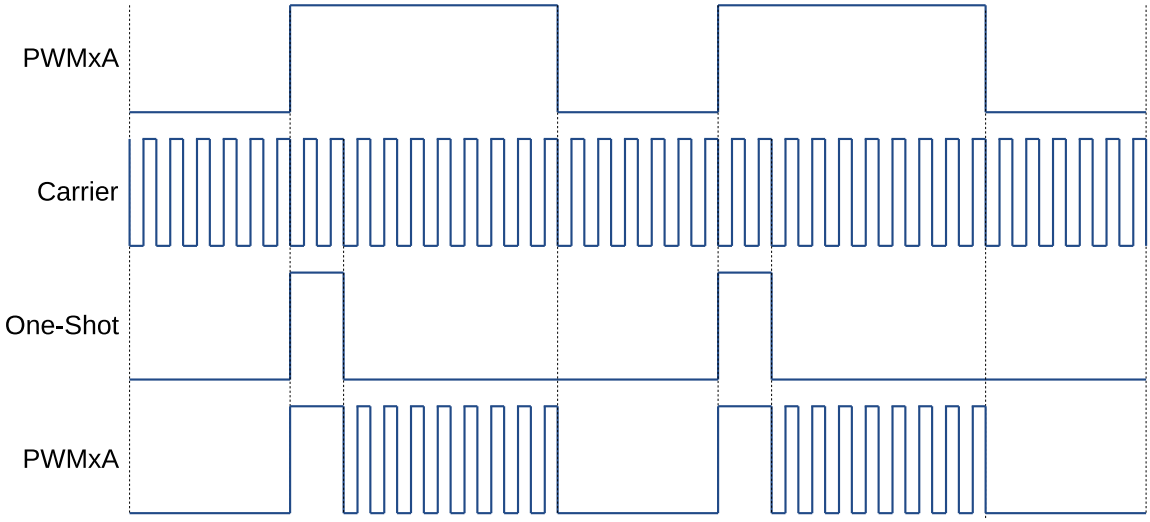


Figure 24.3-27. Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule

Duty Cycle Control

After issuing the first one-shot pulse, the remaining PWM signal is modulated according to the carrier frequency. Users can configure the duty cycle of this signal. Tuning of duty may be required, so that the signal passes through the isolating transformer and can still operate (turn on/off) the motor drive, changing rotation speed and direction.

The duty cycle may be set to one of seven values, using MCPWM_CHOPPERn_DUTY, or bits [7:5] of register MCPWM_CARRIERn_CFG_REG.

Below is the formula for calculating the duty cycle:

$$Duty = MCPWM\_CARRIERn\_DUTY \div 8$$

All seven settings of the duty cycle are shown in Figure 24.3-28.

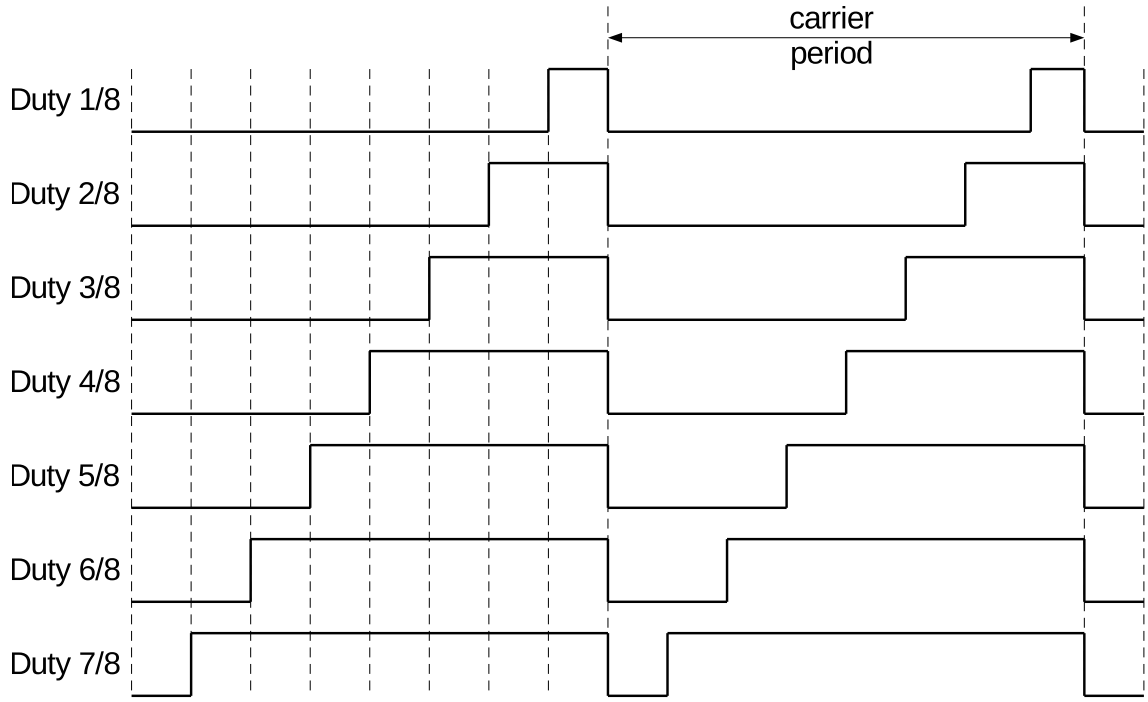


Figure 24.3-28. Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

24.3.3.4 Fault Detection Module

Each MCPWM peripheral is connected to three fault signals (FAULT0, FAULT1, and FAULT2) which are sourced from the GPIO matrix. These signals are intended to indicate external fault conditions, and may be preprocessed by the Fault Detection module to generate fault events. Fault events can then execute the user code to control MCPWM outputs in response to specific faults.

Function of Fault Detection Module

The key actions performed by the fault detection module are:

- Forcing outputs PWM*x*A and PWM*x*B, upon detected fault, to one of the following states:
  - High
  - Low
  - Toggle
  - No action taken
- Execution of one-shot trip (OST) upon detection of over-current conditions/short circuits.
- Cycle-by-cycle trip (CBC) to provide current-limiting operation.
- Allocation of either one-shot or cycle-by-cycle operation for each fault signal.
- Generation of interrupts for each fault input.
- Support for software-force tripping.
- Enabling or disabling of module function as required.

Operation and Configuration Tips

This section provides the operational tips and set-up options for the Fault Detection module.

Fault signals coming from pins are sampled and synced in the GPIO matrix. In order to guarantee the successful sampling of fault pulses, each pulse duration must be at least two APB clock cycles. The Fault Detection module will then sample fault signals by using PWM_clk. So, the duration of fault pulses coming from GPIO matrix must be at least one PWM_clk cycle. Differently put, regardless of the period relation between APB clock and PWM_clk, the width of fault signal pulses on pins must be at least equal to the sum of two APB clock cycles and one PWM_clk cycle.

Each level of fault signals, FAULT0 to FAULT2, can be used by the Fault Detection module to generate fault events (fault_event0 to fault_event2). Every fault event can be configured individually to provide CBC action, OST action, or none.

- **Cycle-by-Cycle (CBC) action:**
  When CBC action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately according to the configuration of fields MCPWM_TZ*n*_A_CBC_U/D and MCPWM_TZ*n*_B_CBC_U/D. Different actions can be indicated when the PWM timer is incrementing or decrementing. Different CBC action interrupts can be triggered for different fault events. Status field MCPWM_TZ*n*_CBC_ON indicates whether a CBC action is on or off. When the fault event is no longer present, CBC actions on PWM*x*A/B will be cleared at a specified point, which is either a D/UTEP or D/UTEZ event. Field MCPWM_TZ*n*_CBCPULSE determines at which event PWM*x*A and PWM*x*B will be able to resume normal actions. Therefore, in this mode, the CBC action is cleared or refreshed upon every PWM cycle.
- **One-Shot (OST) action:**
  When OST action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately, depending on the setting of fields MCPWM_TZ*n*_A_OST_U/D and MCPWM_TZ*n*_B_OST_U/D. Different actions can be configured when PWM timer is incrementing or decrementing. Different OST action interrupts can be triggered from different fault events. Status field MCPWM_TZ*n*_OST_ON indicates whether an OST action is on or off. The OST actions on PWM*x*A/B are not automatically cleared when the fault event is no longer present. One-shot actions must be cleared manually by setting the MCPWM_TZ*n*_CLR_OST bit.

24.3.4 Capture Module

24.3.4.1 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.

24.3.4.2 Capture Timer

The capture timer is a 32-bit counter incrementing continuously. It is enabled by setting MCPWM_CAP_TIMER_EN to 1. Its operating clock source is MCPWM core clock. When MCPWM_CAP_SYNCI_EN is configured, the counter will be loaded with phase stored in register MCPWM_CAP_TIMER_PHASE_REG at the time of a sync event. Sync events can select from PWM timers sync-out, or PWM module sync-in by configuring MCPWM_CAP_SYNCI_SEL. Sync events can also be generated by setting MCPWM_CAP_SYNC_SW. The capture timer provides timing references for all three capture channels.

24.3.4.3 Capture Channel

The capture signal coming to a capture channel will be inverted first, if needed, and then prescaled. Each capture channel has a prescaler register of MCPWM_CAP*n*_PRESCALE. Finally, specified edges of preprocessed capture signal will trigger capture events. Setting MCPWM_CAP*n*_EN to enable a capture channel. The capture event occurs at the time selected by the MCPWM_CAP*n*_MODE. When a capture event occurs, the capture timer's value is stored in time-stamp register MCPWM_CAP_CH*n*_REG. Different interrupts can be generated for different capture channels at capture events. The edge that triggers a capture event is recorded in register MCPWM_CAP*n*_EDGE. The capture event can be also forced by software setting MCPWM_CAP*n*_SW.

24.3.5 ETM Module

24.3.5.1 Overview

The MCPWM peripheral on ESP32-S31 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 (ETM) [to be added later].

24.3.5.2 MCPWM-Related ETM Events

When setting enable field to 1, after the generation condition is met, the corresponding event would be generated. For details, please refer to Table 24.3-7 below:

Table 24.3-7. MCPWM-Related ETM Events

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_CAP*n*_EN	CAP*n* capture event occurs	MCPWM0_EVT_CAP*n*
MCPWM_EVT_TZ*n*_OST_EN	PWM operator *n* performs a One-Shot trip (OST) operation	MCPWM0_EVT_TZ*n*_OST
MCPWM_EVT_TZ*n*_CBC_EN	PWM operator *n* performs a cycle-by-cycle trip (CBC) operation	MCPWM0_EVT_TZ*n*_CBC
MCPWM_EVT_F*n*_CLR_EN	Fault event fault_event*n* is cleared	MCPWM0_EVT_F*n*_CLR

[1] See Section 24.3.3.1 for a detailed description of timer stamp A and B.

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_F*n*_EN	Fault event fault_event*n* is generated	MCPWM0_EVT_F*n*
MCPWM_EVT_OP*n*_TEB_EN	The count value of the timer that PWM operator *n* selects is equal to the value of timer stamp B[1]	MCPWM0_EVT_OP*n*_TEB
MCPWM_EVT_OP*n*_TEA_EN	The count value of the timer that PWM operator *n* selects is equal to the value of timer stamp A[1]	MCPWM0_EVT_OP*n*_TEA
MCPWM_EVT_OP*n*_TEE1_EN	The count value of the timer that PWM operator *n* selects is equal to the value of register MCPWM_OP*n*_TSTMP_E1_REG	MCPWM0_EVT_OP*n*_TEE1
MCPWM_EVT_OP*n*_TEE2_EN	The count value of the timer that PWM operator *n* selects is equal to the value of register MCPWM_OP*n*_TSTMP_E2_REG	MCPWM0_EVT_OP*n*_TEE2
MCPWM_EVT_TIMER*n*_TEP_EN	The count value of timer *n* is equal to the period value MCPWM_TIMER*n*_PERIOD	MCPWM0_EVT_TIMER*n*_TEP
MCPWM_EVT_TIMER*n*_TEZ_EN	The count value of timer *n* is equal to 0	MCPWM0_EVT_TIMER*n*_TEZ
MCPWM_EVT_TIMER*n*_STOP_EN	Timer *n*'s count stops	MCPWM0_EVT_TIMER*n*_STOP

[1] See Section 24.3.3.1 for a detailed description of timer stamp A and B.

24.3.5.3 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 24.3-8 below:

Table 24.3-8. ETM Related Tasks

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CAP*n*_EN	MCPWM0_TASK_CAP*n*	CAP*n* channel performs a capture operation
MCPWM_TASK_CLR*n*_OST_EN	MCPWM0_TASK_CLR*n*_OST	PWM operator *n* clears the One-Shot Trip operation
MCPWM_TASK_TZ*n*_OST_EN	MCPWM0_TASK_TZ*n*_OST	PWM operator *n* performs a One-Shot Trip (OST) operation
MCPWM_TASK_TIMER*n*_PERIOD_UP_EN	MCPWM0_TASK_TIMER*n*_PERIOD_UP	The period of timer *n* is updated to the value configured in the period register MCPWM_TIMER*n*_PERIOD
MCPWM_TASK_TIMER*n*_SYNC_EN	MCPWM0_TASK_TIMER*n*_SYN	Timer *n* performs a sync operation
MCPWM_TASK_GEN_STOP_EN	MCPWM0_TASK_GEN_STOP	All the timers stop counting and the PWM signals output by all PWM operators remain unchanged

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CMPR*n*_B_UP_EN	MCPWM0_TASK_CMPR*n*_B_UP	Timer stamp B of the PWM operator *n* is updated to the value of the shadow register MCPWM_GEN*n*_B
MCPWM_TASK_CMPR*n*_A_UP_EN	MCPWM0_TASK_CMPR*n*_A_UP	Timer stamp A of the PWM operator *n* is updated to the value of the shadow register MCPWM_GEN*n*_A

24.4 Interrupts

ESP32-S31's MCPWM0 can generate the PWM0_INTR interrupt signal. The PWM0_INTR will be sent to the Interrupt Matrix.

Interrupt signal PWM0_INTR can be generated by the following internal interrupt sources:

- MCPWM_TIMER*x*_STOP_INT: Triggered when timer*x* stops.
- MCPWM_TIMER*x*_TEZ_INT: Triggered by the TEZ event of PWM timer*x*.
- MCPWM_TIMER*x*_TEP_INT: Triggered by the TEP event of PWM timer*x*.
- MCPWM_FAULT*x*_INT: Triggered when fault_event*x* starts.
- MCPWM_FAULT*x*_CLR_INT: Triggered after fault_event*x* ends.
- MCPWM_CMPR*x*_TEA_INT: Triggered by the TEA event of PWM operator*x*.
- MCPWM_CMPR*x*_TEB_INT: Triggered by the TEB event of PWM operator*x*.
- MCPWM_TZ*x*_CBC_INT: Triggered by the CBC action of PWM*x*.
- MCPWM_TZ*x*_OST_INT: Triggered by the OST action of PWM*x*.
- MCPWM_CAP*x*_INT: Triggered by the capture event on channel*x*.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

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 24.5 *Register Summary*.

24.5 Register Summary

The addresses in this section are relative to Motor Control PWM base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Prescaler Configuration			
MCPWM_CLK_CFG_REG	PWM clock prescaler register	0x0000	R/W
MCPWM Timer 0 Configuration and Status			
MCPWM_TIMER0_CFG0_REG	PWM timer0 period and update method configuration register	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	PWM timer0 working mode and start/stop control configuration register	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM timer0 sync function configuration register	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM timer0 status register	0x0010	RO
MCPWM Timer 1 Configuration and Status			
MCPWM_TIMER1_CFG0_REG	PWM timer1 period and update method configuration register	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	PWM timer1 working mode and start/stop control configuration register	0x0018	varies
MCPWM_TIMER1_SYNC_REG	PWM timer1 sync function configuration register	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM timer1 status register	0x0020	RO
MCPWM Timer 2 Configuration and status			
MCPWM_TIMER2_CFG0_REG	PWM timer2 period and update method configuration register	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	PWM timer2 working mode and start/stop control configuration register	0x0028	varies
MCPWM_TIMER2_SYNC_REG	PWM timer2 sync function configuration register	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM timer2 status register	0x0030	RO
Common Configuration for MCPWM Timers			
MCPWM_TIMER_SYNCI_CFG_REG	Synchronization input selection for three PWM timers	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	Select specific timer for PWM operators	0x0038	R/W
MCPWM Operator 0 Configuration and Status			
MCPWM_GEN0_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	Shadow register for register A	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	Shadow register for register B	0x0044	R/W
MCPWM_GEN0_CFG0_REG	Fault event T0 and T1 handling	0x0048	R/W
MCPWM_GEN0_FORCE_REG	Permissives to force PWM0A and PWM0B outputs by software	0x004C	R/W

Name	Description	Address	Access
MCPWM_GEN0_A_REG	Actions triggered by events on PWM0A	0x0050	R/W
MCPWM_GEN0_B_REG	Actions triggered by events on PWM0B	0x0054	R/W
MCPWM_DT0_CFG_REG	Dead time type selection and configuration	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	Carrier enable and configuration	0x0064	R/W
MCPWM_FH0_CFG0_REG	Actions on PWM0A and PWM0B trip events	0x0068	R/W
MCPWM_FH0_CFG1_REG	Software triggers for fault handler actions	0x006C	R/W
MCPWM_FH0_STATUS_REG	Status of fault events	0x0070	RO
MCPWM Operator 1 Configuration and Status			
MCPWM_GEN1_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x0074	varies
MCPWM_GEN1_TSTMP_A_REG	Shadow register for register A	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	Shadow register for register B	0x007C	R/W
MCPWM_GEN1_CFG0_REG	Fault event T0 and T1 handling	0x0080	R/W
MCPWM_GEN1_FORCE_REG	Permissives to force PWM1A and PWM1B outputs by software	0x0084	R/W
MCPWM_GEN1_A_REG	Actions triggered by events on PWM1A	0x0088	R/W
MCPWM_GEN1_B_REG	Actions triggered by events on PWM1B	0x008C	R/W
MCPWM_DT1_CFG_REG	Dead time type selection and configuration	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	Carrier enable and configuration	0x009C	R/W
MCPWM_FH1_CFG0_REG	Actions on PWM1A and PWM1B trip events	0x00A0	R/W
MCPWM_FH1_CFG1_REG	Software triggers for fault handler actions	0x00A4	R/W
MCPWM_FH1_STATUS_REG	Status of fault events	0x00A8	RO
MCPWM Operator 2 Configuration and Status			
MCPWM_GEN2_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	Shadow register for register A	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	Shadow register for register B	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	Fault event T0 and T1 handling	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	Permissives to force PWM2A and PWM2B outputs by software	0x00BC	R/W
MCPWM_GEN2_A_REG	Actions triggered by events on PWM2A	0x00C0	R/W
MCPWM_GEN2_B_REG	Actions triggered by events on PWM2B	0x00C4	R/W
MCPWM_DT2_CFG_REG	Dead time type selection and configuration	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	Carrier enable and configuration	0x00D4	R/W
MCPWM_FH2_CFG0_REG	Actions on PWM2A and PWM2B trip events	0x00D8	R/W
MCPWM_FH2_CFG1_REG	Software triggers for fault handler actions	0x00DC	R/W
MCPWM_FH2_STATUS_REG	Status of fault events	0x00E0	RO

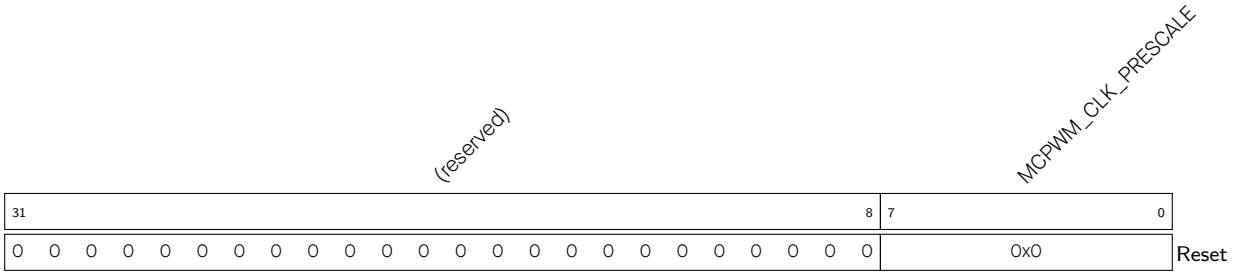
Name	Description	Address	Access
Fault Detection Configuration and Status			
MCPWM_FAULT_DETECT_REG	Fault detection configuration and status	0x00E4	varies
Capture Configuration and Status			
MCPWM_CAP_TIMER_CFG_REG	Configure capture timer	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	Phase for capture timer sync	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	Capture channel 0 configuration and enable	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	Capture channel 1 configuration and enable	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	Capture channel 2 configuration and enable	0x00F8	varies
MCPWM_CAP_CH0_REG	ch0 capture value status register	0x00FC	RO
MCPWM_CAP_CH1_REG	ch1 capture value status register	0x0100	RO
MCPWM_CAP_CH2_REG	ch2 capture value status register	0x0104	RO
MCPWM_CAP_STATUS_REG	Edge of last capture trigger	0x0108	RO
Enable Update of Active Registers			
MCPWM_UPDATE_CFG_REG	Enable update	0x010C	R/W
Manage Interrupts			
MCPWM_INT_ENA_REG	Interrupt enable bits	0x0110	R/W
MCPWM_INT_RAW_REG	Raw interrupt status	0x0114	R/WTC/SS
MCPWM_INT_ST_REG	Masked interrupt status	0x0118	RO
MCPWM_INT_CLR_REG	Interrupt clear bits	0x011C	WT
MCPWM Event Enable Registers			
MCPWM_EVT_EN_REG	MCPWM event enable register	0x0120	R/W
MCPWM_EVT_EN2_REG	MCWM event enable register2	0x0128	R/W
MCPWM Task Enable Register			
MCPWM_TASK_EN_REG	MCPWM task enable register	0x0124	R/W
MCPWM Generator Configuration Registers			
MCPWM_OP0_TSTMP_E1_REG	Generator0 time stamp E1 value configuration register	0x012C	R/W
MCPWM_OP0_TSTMP_E2_REG	Generator0 time stamp E2 value configuration register	0x0130	R/W
MCPWM_OP1_TSTMP_E1_REG	Generator1 time stamp E1 value configuration register	0x0134	R/W
MCPWM_OP1_TSTMP_E2_REG	Generator1 time stamp E2 value configuration register	0x0138	R/W
MCPWM_OP2_TSTMP_E1_REG	Generator2 time stamp E1 value configuration register	0x013C	R/W
MCPWM_OP2_TSTMP_E2_REG	Generator2 time stamp E2 value configuration register	0x0140	R/W
MCPWM APB Configuration Register			
MCPWM_CLK_REG	MCPWM APB configuration register	0x0144	R/W
Version Register			
MCPWM_VERSION_REG	Version control register	0x0148	R/W

24.6 Registers

The addresses in this section are relative to Motor Control PWM base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

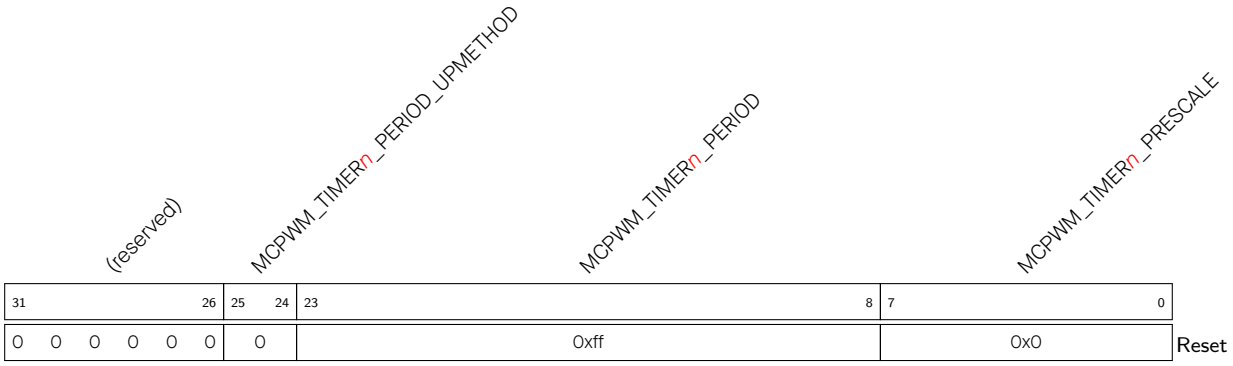
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 24.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE Configures the prescaler value of the clock, so that the period of PWM_clk = 6.25 ns * (PWM_CLK_PRESCALE + 1). (R/W)

Register 24.2. MCPWM_TIMER*n*_CFG0_REG (*n*: 0-2) (0x0004+0x10n*)**



MCPWM_TIMER*n*_PRESCALE Configures the prescaler value of timer*n*, so that the period of PTO_clk = Period of PWM_clk * (PWM_TIMER*n*_PRESCALE + 1). (R/W)

MCPWM_TIMER*n*_PERIOD Configures the period shadow of PWM timer*n*. (R/W)

MCPWM_TIMER*n*_PERIOD_UPMETHOD Configures the update method for active register of PWM timer*n* period.

0: Immediate

1: TEZ

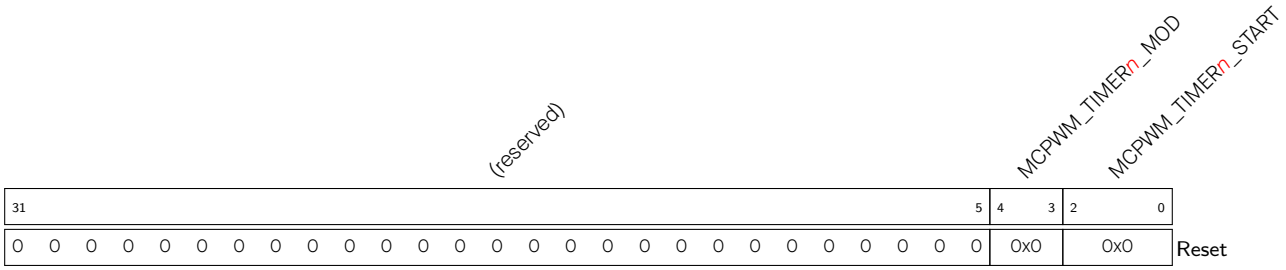
2: Sync

3: TEZ or sync

TEZ here and below means timer equals zero event.

(R/W)

Register 24.3. MCPWM_TIMER*n*_CFG1_REG(*n*: 0-2) (0x0008+0x10n*)**



MCPWM_TIMER*n*_START Configures conditions to start/stop PWM timer*n*.

0: If PWM timer0 starts, then stops at TEZ

1: If timer0 starts, then stops at TEP

2: PWM timer0 starts and runs on

3: Timer0 starts and stops at the next TEZ

4: Timer0 starts and stops at the next TEP

5: Invalid. No effect

6: Invalid. No effect

7: Invalid. No effect

TEP here and below means the event that happens when the timer equals to period.

(R/W/SC)

MCPWM_TIMER*n*_MOD Configures the working mode of PWM timer*n*.

0: Freeze

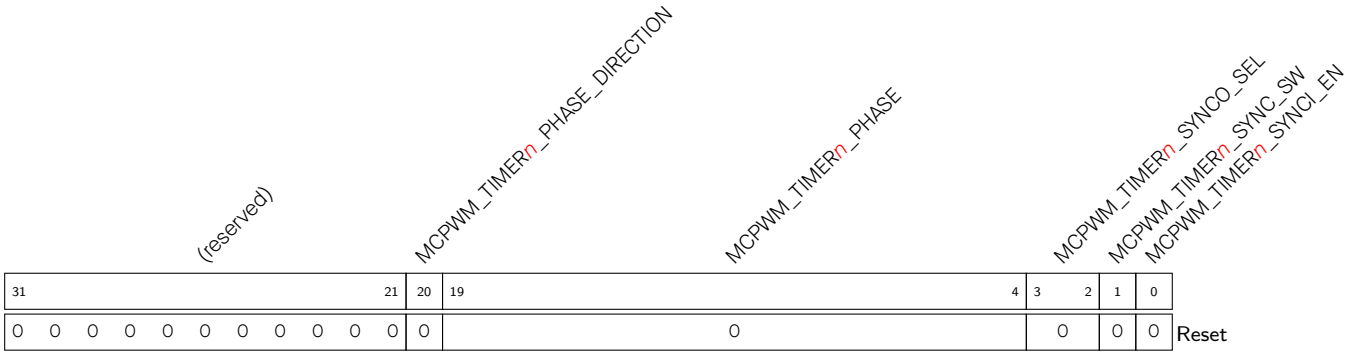
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 24.4. MCPWM_TIMER*n*_SYNC_REG (*n*: 0-2) (0x000C+0x10**n*)



MCPWM_TIMER*n*_SYNCI_EN Configures whether to enable timer*n* reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER*n*_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER*n*_SYNCO_SEL Configures PWM timer*n* sync out selection.

0: sync_in. The sync out will always generate when toggling the MCPWM_TIMER*n*_SYNC_SW bit.

1: TEZ

2: TEP

3: No effect

(R/W)

MCPWM_TIMER*n*_PHASE Configures the phase for timer*n* reload on sync event.

(R/W)

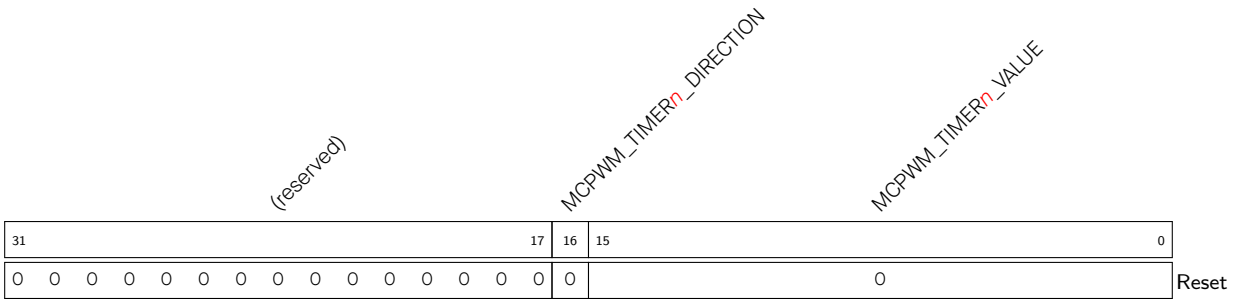
MCPWM_TIMER*n*_PHASE_DIRECTION Configures the PWM timer*n*'s direction when timer*n* mode is up-down mode.

0: Increase

1: Decrease

(R/W)

Register 24.5. MCPWM_TIMER*n*_STATUS_REG(*n*: 0-2) (0x0010+0x10**n*)



MCPWM_TIMER*n*_VALUE Represents current PWM timer*n* counter value. (RO)

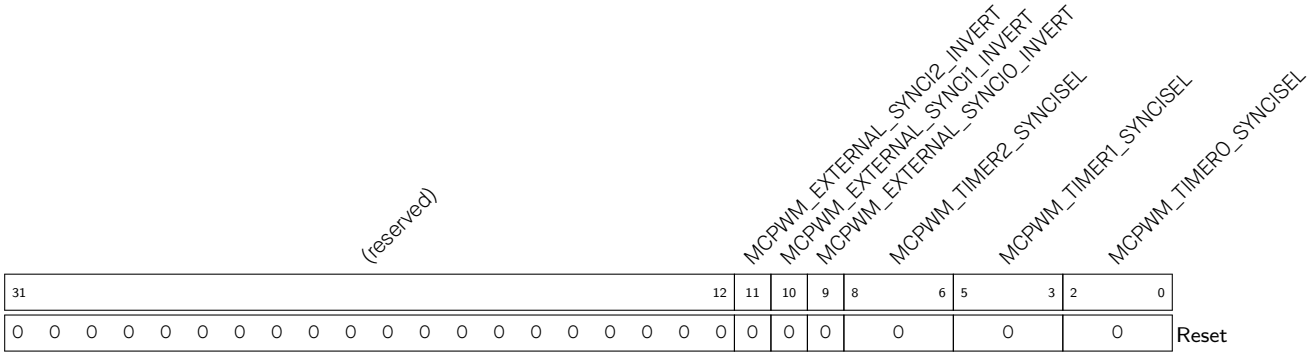
MCPWM_TIMER*n*_DIRECTION Represents current PWM timer*n* counter direction.

0: Increment

1: Decrement

(RO)

Register 24.6. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER*n*_SYNCISEL Configures sync input for PWM timer*n*.

1: PWM timer0 sync out

2: PWM timer1 sync out

3: PWM timer2 sync out

4: SYNC0 from GPIO matrix

5: SYNC1 from GPIO matrix

6: SYNC2 from GPIO matrix

Other values: no sync input selected

(R/W)

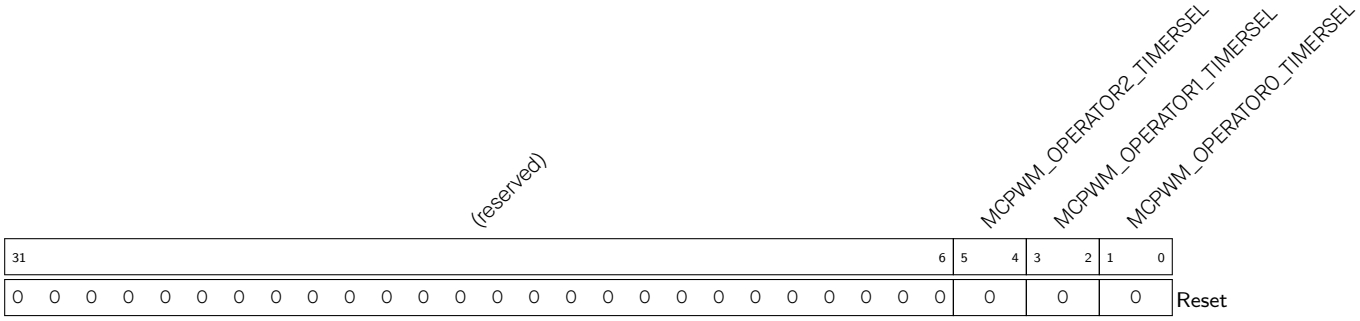
MCPWM_EXTERNAL_SYNCI*n*_INVERT Configures whether to invert SYNC*n* from GPIO matrix.

0: Not invert

1: Invert

(R/W)

Register 24.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator*n*.

0: timer0

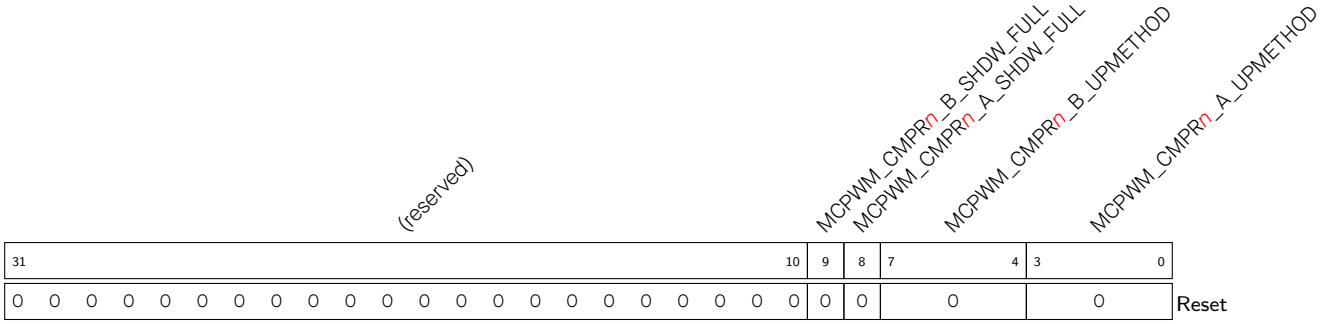
1: timer1

2: timer2

3: Invalid, will select timer2

(R/W)

Register 24.8. MCPWM_GEN*n*_STMP_CFG_REG (*n*: 0-2) (0x003C+0x38n*)**



MCPWM_CMPR*n*_A_UPMETHOD Configures the update method for PWM generator*n* time stamp A's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_CMPR*n*_B_UPMETHOD Configures the update method for PWM generator*n* time stamp B's active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_CMPR*n*_A_SHDW_FULL Configures whether the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to A's active register

1: Write the value to the PWM generator*n* time stamp A's shadow register, and wait to be transferred to A's active register

(R/SC/WTC)

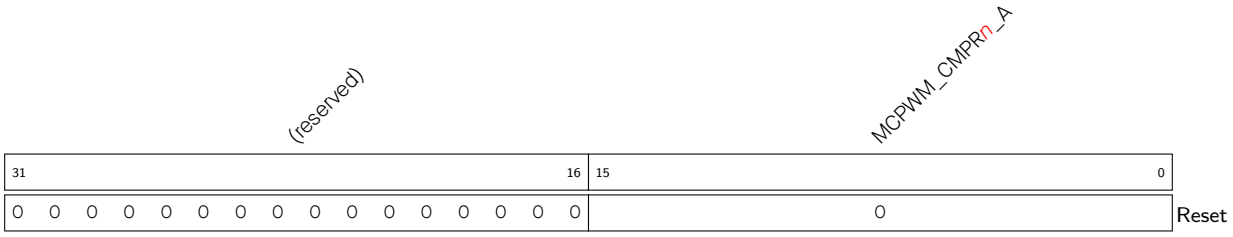
MCPWM_CMPR*n*_B_SHDW_FULL Configures whether the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

0: Write the latest value in the shadow register to B's active register

1: Write the value to the PWM generator*n* time stamp B's shadow register, and wait to be transferred to B's active register

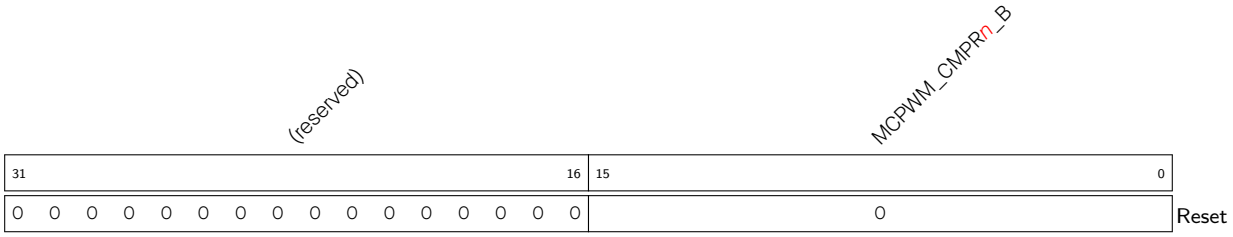
(R/SC/WTC)

Register 24.9. MCPWM_GEN*n*_TSTMP_A_REG(*n*: 0-2) (0x0040+0x38n*)**



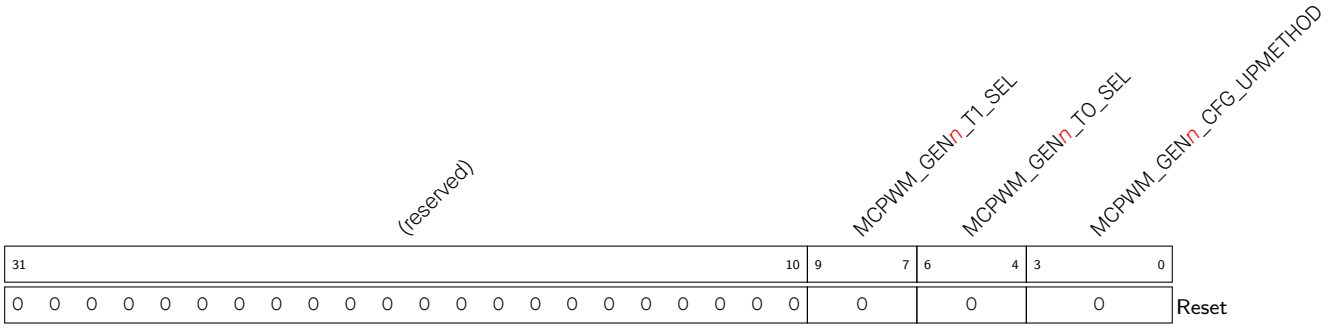
MCPWM_CMPR*n*_A Configures the value of PWM generator *n* time stamp A's shadow register. (R/W)

Register 24.10. MCPWM_GEN*n*_TSTMP_B_REG(*n*: 0-2) (0x0044+0x38n*)**



MCPWM_CMPR*n*_B Configures the value of PWM generator *n* time stamp B's shadow register. (R/W)

Register 24.11. MCPWM_GEN*n*_CFG0_REG(*n*: 0-2) (0x0048+0x38**n*)



MCPWM_GEN*n*_CFG_UPMETHOD Configures the update method for PWM generator*n*'s active register.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: Sync

When bit3 is set to 1: Disable the update

(R/W)

MCPWM_GEN*n*_T0_SEL Configures source selection for PWM generator*n* event_t0, take effect immediately.

0: fault_event0

1: fault_event1

2: fault_event2

3: sync_taken

4: Invalid, selects nothing

(R/W)

MCPWM_GEN*n*_T1_SEL Configures source selection for PWM generator*n* event_t1, take effect immediately.

0: fault_event0

1: fault_event1

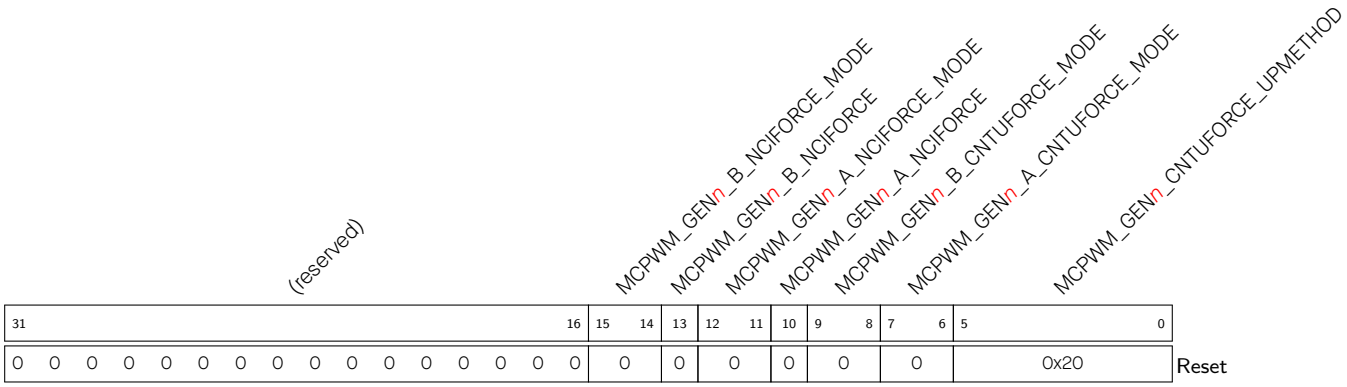
2: fault_event2

3: sync_taken

4: Invalid, selects nothing

(R/W)

Register 24.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)



MCPWM_GEN*n*_CNTUFORCE_UPMETHOD Configures the update method for continuous software force of PWM generator*n*.

When all bits are set to 0: Immediately

When bit0 is set to 1: TEZ

When bit1 is set to 1: TEP

When bit2 is set to 1: TEA

When bit3 is set to 1: TEB

When bit4 is set to 1: Sync

When bit5 is set to 1: Disable update

TEA/B means an event generated when the timer's value equals to that of register A/B. (R/W)

MCPWM_GEN*n*_A_CNTUFORCE_MODE Configures the continuous software force mode for PWM*n* A.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN*n*_B_CNTUFORCE_MODE Configures the continuous software force mode for PWM*n* B.

0: Disabled

1: Low

2: High

3: Disabled

(R/W)

MCPWM_GEN*n*_A_NCIFORCE Configures whether to trigger the non-continuous immediate software-force event for PWM*n* A.

0: Invalid

1: Trigger a force event

(R/W)

Continued on the next page...

Register 24.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)

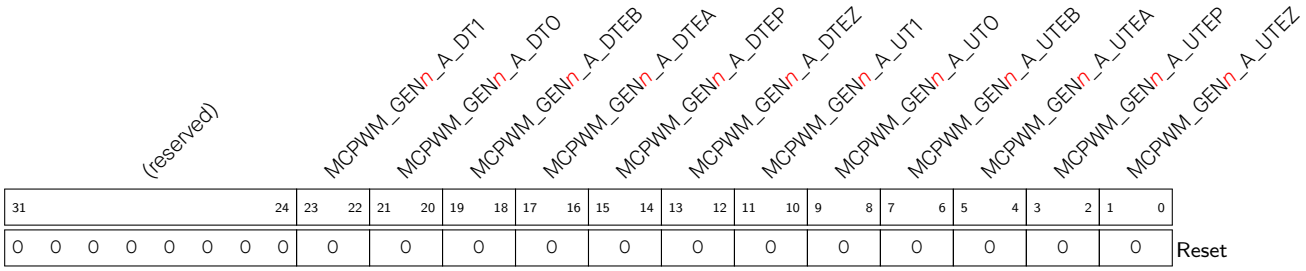
Continued from the previous page...

MCPWM_GEN*n*_A_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM*n* A.
0: Disabled
1: Low
2: High
3: Disabled
(R/W)

MCPWM_GEN*n*_B_NCIFORCE Configures whether to trigger the non-continuous immediate software-force event for PWM*n* B.
0: Invalid
1: Trigger a force event
(R/W)

MCPWM_GEN*n*_B_NCIFORCE_MODE Configures non-continuous immediate software force mode for PWM*n* B.
0: Disabled
1: Low
2: High
3: Disabled
(R/W)

Register 24.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)



MCPWM_GEN*n*_A_UTEZ Configures action on PWM*n* A triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEP Configures action on PWM*n* A triggered by event TEP when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEA Configures action on PWM*n* A triggered by event TEA when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_A_UTEB Configures action on PWM*n* A triggered by event TEB when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

Continued on the next page...

Register 24.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_A_UT0 Configures action on PWM*n* A triggered by event_t0 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_UT1 Configures action on PWM*n* A triggered by event_t1 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEZ Configures action on PWM*n* A triggered by event TEZ when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEP Configures action on PWM*n* A triggered by event TEP when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DTEA Configures action on PWM*n* A triggered by event TEA when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

Continued on the next page...

Register 24.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

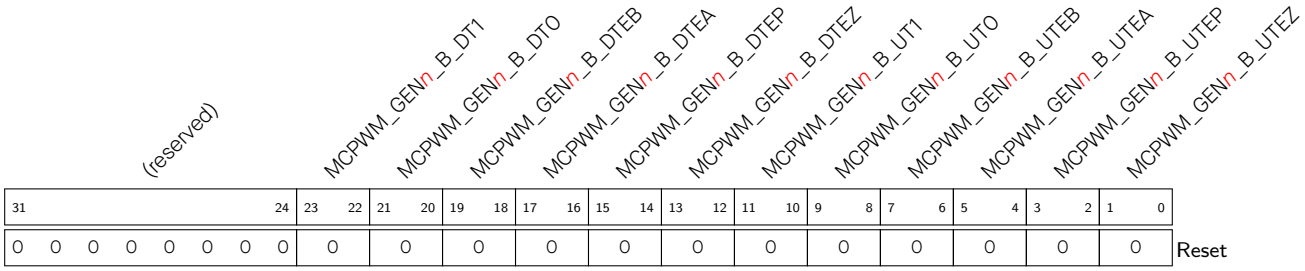
Continued from the previous page...

MCPWM_GEN*n*_A_DTEB Configures action on PWM*n* A triggered by event TEB when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DT0 Configures action on PWM*n* A triggered by event_t0 when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_A_DT1 Configures action on PWM*n* A triggered by event_t1 when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

Register 24.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)



MCPWM_GEN*n*_B_UTEZ Configures action on PWM*n* B triggered by event TEZ when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEP Configures action on PWM*n* B triggered by event TEP when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEA Configures action on PWM*n* B triggered by event TEA when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UTEB Configures action on PWM*n* B triggered by event TEB when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_UT0 Configures action on PWM*n* B triggered by event_t0 when timer increasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

Continued on the next page...

Register 24.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_B_UT1 Configures action on PWM*n* B triggered by event_t1 when timer increasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEZ Configures action on PWM*n* B triggered by event TEZ when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEP Configures action on PWM*n* B triggered by event TEP when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEA Configures action on PWM*n* B triggered by event TEA when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

MCPWM_GEN*n*_B_DTEB Configures action on PWM*n* B triggered by event TEB when timer decreasing.
0: No change
1: Low
2: High
3: Toggle
(R/W)

Continued on the next page...

Register 24.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

Continued from the previous page...

MCPWM_GEN*n*_B_DT0 Configures action on PWM*n* B triggered by event_t0 when timer decreasing.

0: No change

1: Low

2: High

3: Toggle

(R/W)

MCPWM_GEN*n*_B_DT1 Configures action on PWM*n* B triggered by event_t1 when timer decreasing.

0: No change

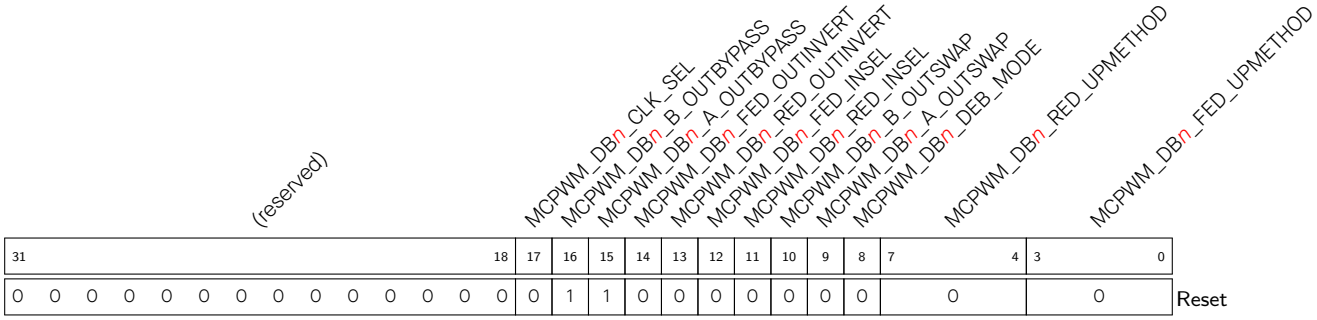
1: Low

2: High

3: Toggle

(R/W)

Register 24.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)



MCPWM_DB*n*_FED_UPMETHOD Configures the update method for FED (Falling edge delay) active register.

0: Immediate

Bit0 is set to 1: TEZ

Bit1 is set to 1: TEP

Bit2 is set to 1: Sync

Bit3 is set to 1: Disable the update

(R/W)

MCPWM_DB*n*_RED_UPMETHOD Configures the update method for RED (Rising edge delay) active register.

0: Immediate

Bit0 is set to 1: TEZ

Bit1 is set to 1: TEP

Bit2 is set to 1: Sync

Bit3 is set to 1: Disable the update

(R/W)

MCPWM_DB*n*_DEB_MODE Configures the S8 switch in Table 24.3-5.

0: FED/RED take effect on different paths separately

1: FED/RED take effect on B path, A out is in bypass or dulpB mode

(R/W)

MCPWM_DB*n*_A_OUTSWAP Configures the S6 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_B_OUTSWAP Configures the S7 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_RED_INSEL Configures the S4 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_FED_INSEL Configures the S5 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_RED_OUTINVERT Configures the S2 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

Continued on the next page...

Register 24.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)

Continued from the previous page...

MCPWM_DB*n*_FED_OUTINVERT Configures the S3 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_A_OUTBYPASS Configures the S1 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

MCPWM_DB*n*_B_OUTBYPASS Configures the S0 switch in Table 24.3-5. For typical configurations, please refer to Table 24.3-6. (R/W)

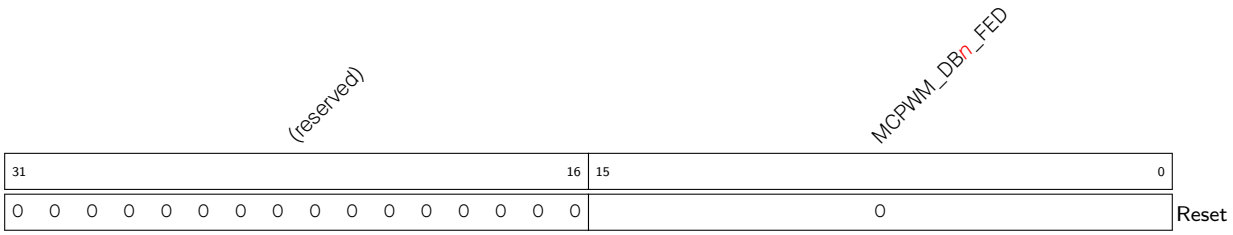
MCPWM_DB*n*_CLK_SEL Configures dead time generator *n* clock selection.

0: PWM_CLK

1: PT_CLK

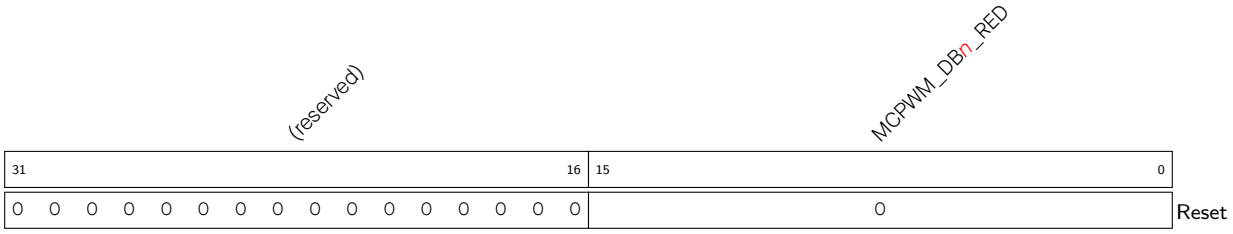
(R/W)

Register 24.16. MCPWM_DT*n*_FED_CFG_REG(*n*: 0-2) (0x005C+0x38**n*)



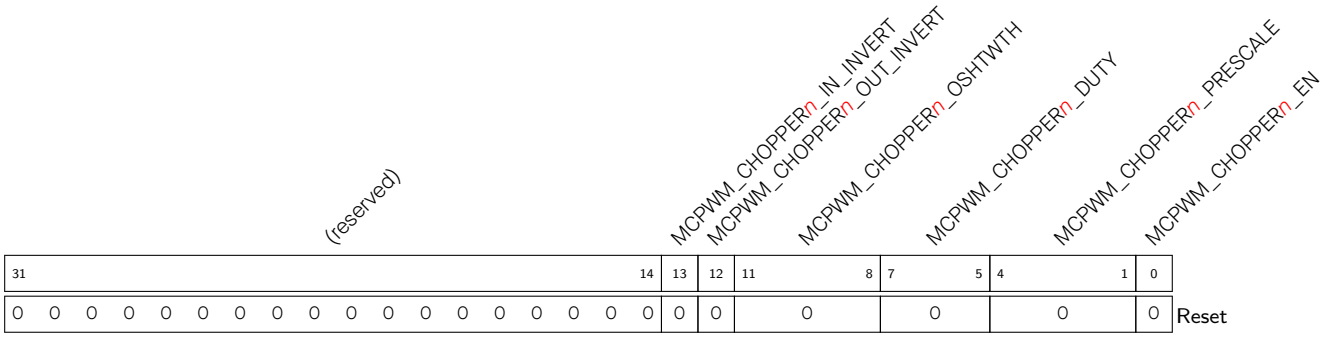
MCPWM_DB*n*_FED Configures the shadow register for FED. (R/W)

Register 24.17. MCPWM_DT*n*_RED_CFG_REG(*n*: 0-2) (0x0060+0x38**n*)



MCPWM_DB*n*_RED Configures the shadow register for RED. (R/W)

Register 24.18. MCPWM_CARRIER*n*_CFG_REG(*n*: 0-2) (0x0064+0x38n*)**



MCPWM_CHOPPER*n*_EN Configures whether to enable carrier*n*.

0: Bypassed

1: Enabled

(R/W)

MCPWM_CHOPPER*n*_PRESCALE Configures the prescale value of PWM carrier*n* clock (PC_clk), so that period of PC_clk = period of PWM_clk * (PWM_CARRIER*n*_PRESCALE + 1). (R/W)

MCPWM_CHOPPER*n*_DUTY Configures carrier duty. Duty = PWM_CARRIER*n*_DUTY / 8. (R/W)

MCPWM_CHOPPER*n*_OSHTWTH Configures width of the first pulse. Measurement unit: Periods of the carrier. (R/W)

MCPWM_CHOPPER*n*_OUT_INVERT Configures whether to invert the output of PWM*n* A and PWM*n* B for this submodule.

0: Normal

1: Invert

(R/W)

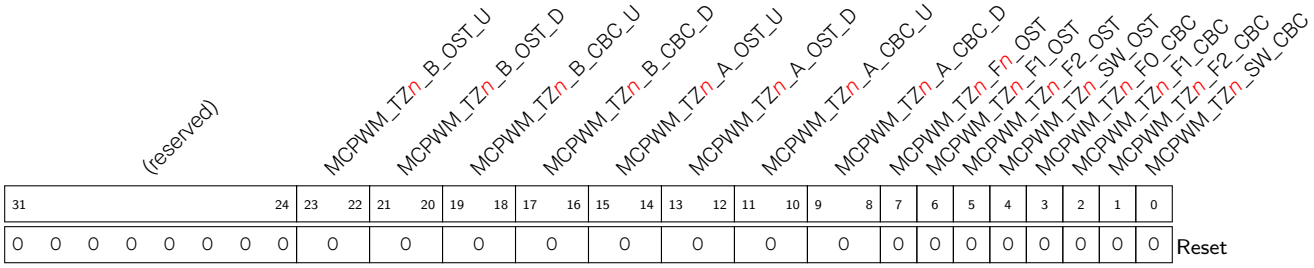
MCPWM_CHOPPER*n*_IN_INVERT Configures whether to invert the input of PWM*n* A and PWM*n* B for this submodule.

0: Normal

1: Invert

(R/W)

Register 24.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)



MCPWM_TZ*n*_SW_CBC Configures whether to enable software force cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F2_CBC Configures whether event_f2 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F1_CBC Configures whether event_f1 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F0_CBC Configures whether event_f0 will trigger cycle-by-cycle mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_SW_OST Configures whether to enable software force one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F2_OST Configures whether event_f2 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F1_OST Configures whether event_f1 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

MCPWM_TZ*n*_F*n*_OST Configures whether event_f0 will trigger one-shot mode action.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 24.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

Continued from the previous page...

MCPWM_TZ*n*_A_CBC_D Configures cycle-by-cycle mode action on PWM*n* A when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_CBC_U Configures cycle-by-cycle mode action on PWM*n* A when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_OST_D Configures one-shot mode action on PWM*n* A when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_A_OST_U Configures one-shot mode action on PWM*n* A when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_CBC_D Configures cycle-by-cycle mode action on PWM*n* B when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

Continued on the next page...

Register 24.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

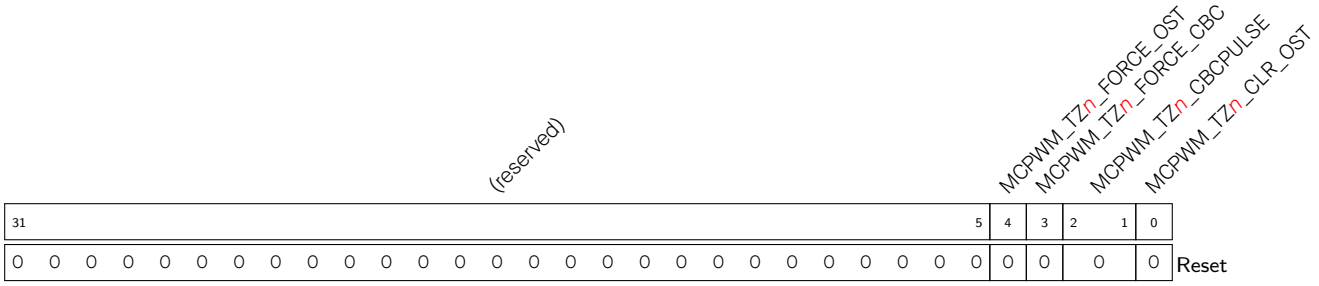
Continued from the previous page...

MCPWM_TZ*n*_B_CBC_U Configures cycle-by-cycle mode action on PWM*n* B when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_OST_D Configures one-shot mode action on PWM*n* B when fault event occurs and timer is decreasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

MCPWM_TZ*n*_B_OST_U Configures one-shot mode action on PWM*n* B when fault event occurs and timer is increasing.
0: Do nothing
1: Force low
2: Force high
3: Toggle
(R/W)

Register 24.20. MCPWM_FH*n*_CFG1_REG(*n*: 0-2) (0x006C+0x38**n*)



MCPWM_TZ*n*_CLR_OST Configures whether to generate a one-shot mode action clear by software.

0: No effect

1: Triggers a clear for ongoing one-shot mode action by software

(R/W)

MCPWM_TZ*n*_CBCPULSE Configures the refresh moment selection of cycle-by-cycle mode action.

0: Select nothing, will not refresh

Bit0 is set to 1: TEZ

Bit1 is set to 1: TEP

(R/W)

MCPWM_TZ*n*_FORCE_CBC Configures whether to generate a software cycle-by-cycle mode action.

0: No effect

1: Triggers a cycle-by-cycle mode action by software

(R/W)

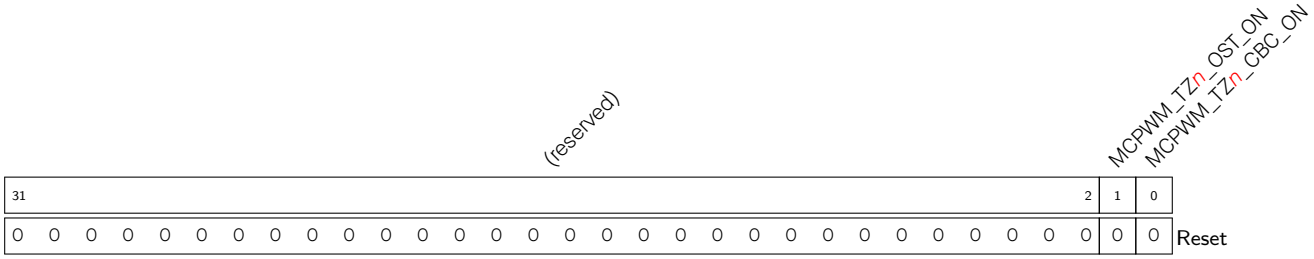
MCPWM_TZ*n*_FORCE_OST Configures whether to generate a software one-shot mode action.

0: No effect

1: Triggers a one-shot mode action by software

(R/W)

Register 24.21. MCPWM_FH*n*_STATUS_REG(*n*: 0-2) (0x0070+0x38**n*)



MCPWM_TZ*n*_CBC_ON Represents whether an cycle-by-cycle mode action is on going.

0: No action

1: Ongoing

(RO)

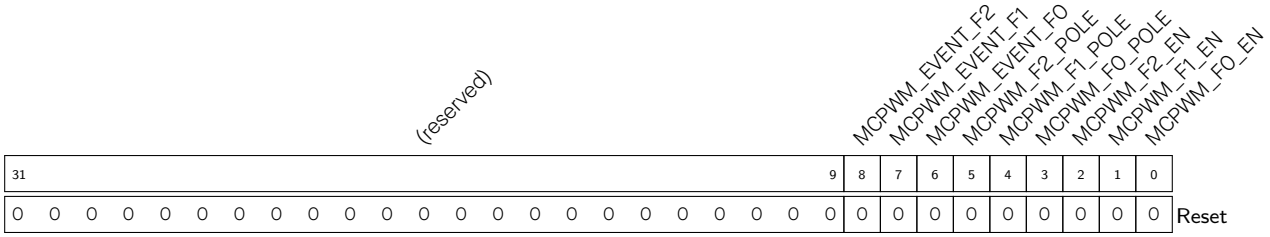
MCPWM_TZ*n*_OST_ON Represents whether a one-shot mode action is ongoing.

0: No action

1: Ongoing

(RO)

Register 24.22. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F*n*_EN Configures whether to enable event_f*n* generation.

0: Disable

1: Enable

(R/W)

MCPWM_F*n*_POLE Configures event_f*n* trigger polarity on FAULT*n* source from GPIO matrix.

0: Level low

1: Level high

(R/W)

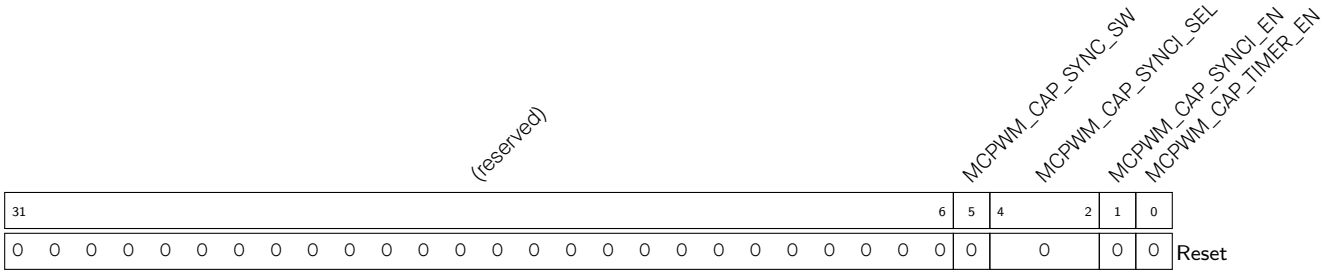
MCPWM_EVENT_F*n* Represents whether an event_f*n* is ongoing. This field is set and reset by hardware.

0: No action

1: Ongoing

(RO)

Register 24.23. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN Configures whether to enable capture timer increment incrementing under APB_CLK.

0: Disable

1: Enable

(R/W)

MCPWM_CAP_SYNCI_EN Configures whether to enable capture timer sync.

0: Disable

1: Enable

(R/W)

MCPWM_CAP_SYNCI_SEL Configures the selection of capture module sync input.

0: None

1: Timer0 sync_out

2: Timer1 sync_out

3: Timer2 sync_out

4: SYNC0 from GPIO matrix

5: SYNC1 from GPIO matrix

6: SYNC2 from GPIO matrix

7: None

(R/W)

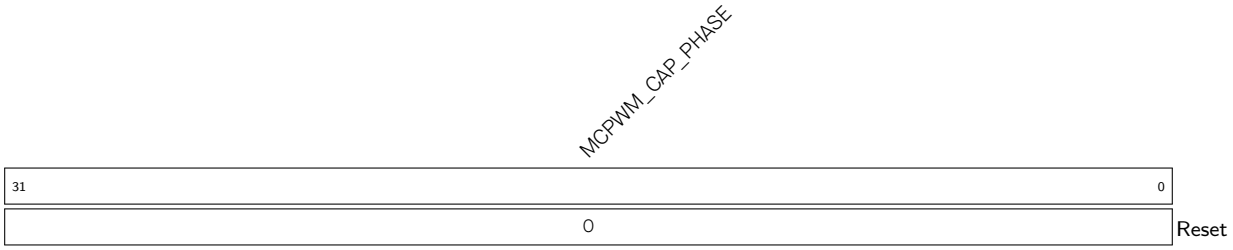
MCPWM_CAP_SYNC_SW When MCPWM_CAP_SYNCI_EN is set to 1, configures whether to trigger a capture timer sync so that capture timer is loaded with value in phase register.

0: No effect

1: Trigger a capture timer sync

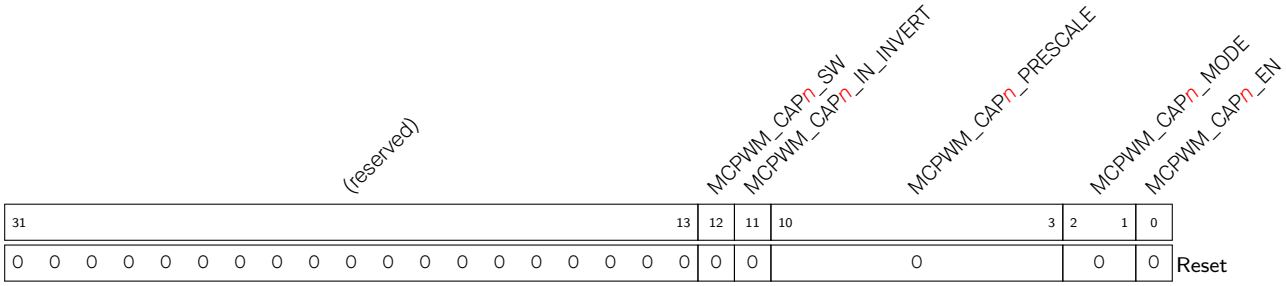
(WT)

Register 24.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_PHASE Configures phase value for capture timer sync operation. (R/W)

Register 24.25. MCPWM_CAP_CH*n*_CFG_REG (*n*: 0-2) (0x00F0+0x4**n*)



MCPWM_CAP*n*_EN Configures whether to enable capture on channel *n*.

0: Disable

1: Enable

(R/W)

MCPWM_CAP*n*_MODE Configures the edge of capture on channel 0 after prescaling.

When bit0 is set to 1: enable capture on the falling edge.

When bit1 is set to 1: enable capture on the rising edge.

(R/W)

MCPWM_CAP*n*_PRESCALE Configures the prescale value on the rising edge of CAP0. Prescale value = PWM_CAP0_PRESCALE + 1. (R/W)

MCPWM_CAP*n*_IN_INVERT Configures whether to invert CAP*n* from GPIO matrix before prescale.

0: Normal

1: Invert

(R/W)

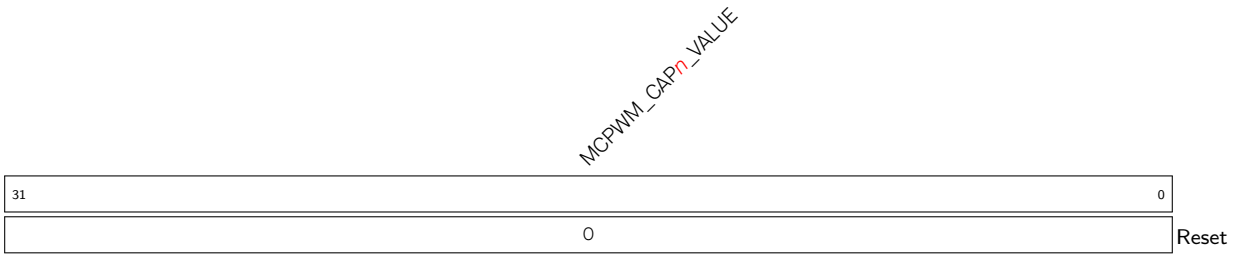
MCPWM_CAP*n*_SW Configures whether to trigger a software-forced capture on channel 0.

0: Not trigger

1: Trigger

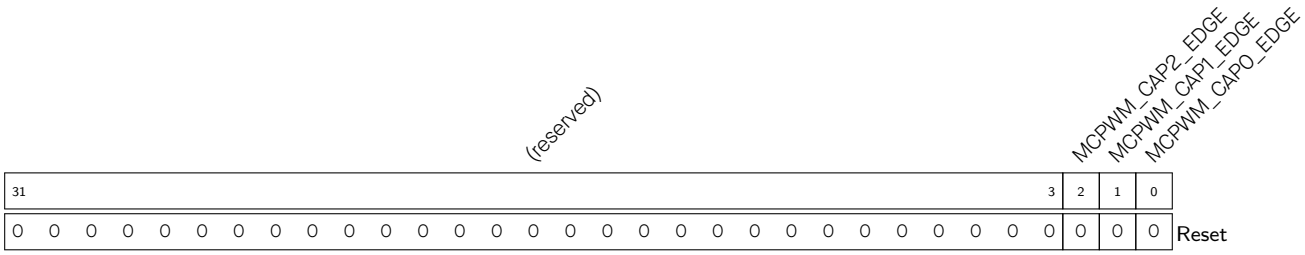
(WT)

Register 24.26. MCPWM_CAP_CH*n*_REG (*n*: 0-2) (0x00FC+0x4**n*)



MCPWM_CAP*n*_VALUE Represents value of last capture on CAP*n*. (RO)

Register 24.27. MCPWM_CAP_STATUS_REG (0x0108)



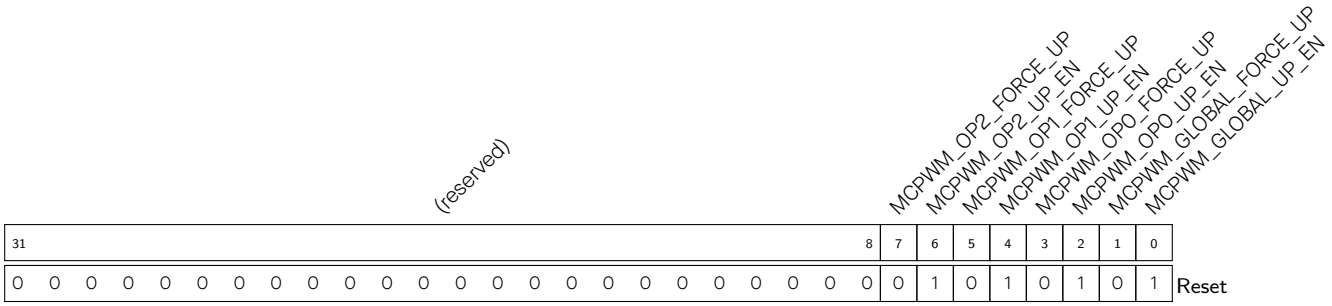
MCPWM_CAP*n*_EDGE Represents the edge of the last capture trigger on channel *n*.

0: Rising edge

1: Falling edge

(RO)

Register 24.28. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN Configures whether to globally update all active registers.

0: No effect

1: Update all active registers globally

(R/W)

MCPWM_GLOBAL_FORCE_UP Configures whether to trigger a forced update of all active registers globally.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP0_UP_EN Configures whether to update active registers in PWM operator 0 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 0

(R/W)

MCPWM_OP0_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 0.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP1_UP_EN Configures whether to update active registers in PWM operator 1 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 1

(R/W)

Continued on the next page...

Register 24.28. MCPWM_UPDATE_CFG_REG (0x010C)

Continued from the previous page...

MCPWM_OP1_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 1.

0: No effect

1: Trigger a forced update

(R/W)

MCPWM_OP2_UP_EN Configures whether to update active registers in PWM operator 2 when MCPWM_GLOBAL_UP_EN is set to 1.

0: No effect

1: Update active registers in PWM operator 2

(R/W)

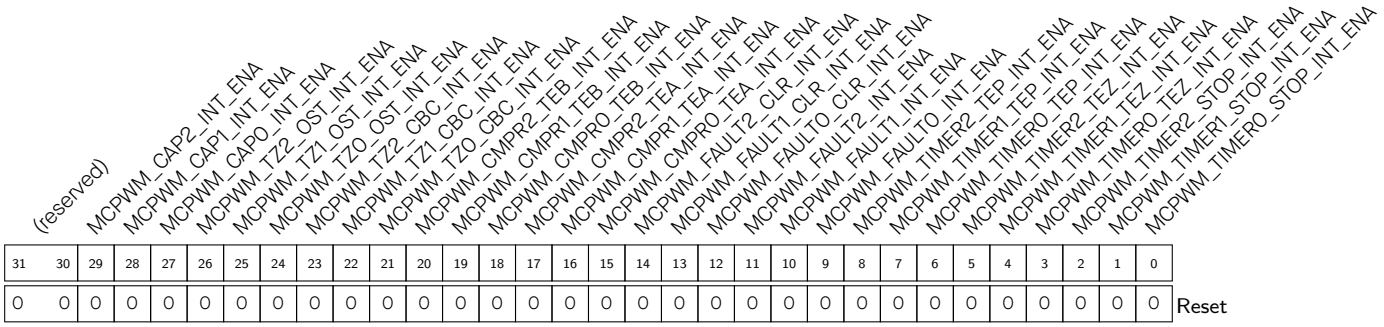
MCPWM_OP2_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 2.

0: No effect

1: Trigger a forced update

(R/W)

Register 24.29. MCPWM_INT_ENA_REG (0x0110)



MCPWM_TIMER0_STOP_INT_ENA Write 1 to enable MCPWM_TIMER0_STOP_INT. (R/W)

MCPWM_TIMER1_STOP_INT_ENA Write 1 to enable MCPWM_TIMER1_STOP_INT. (R/W)

MCPWM_TIMER2_STOP_INT_ENA Write 1 to enable MCPWM_TIMER2_STOP_INT. (R/W)

MCPWM_TIMER0_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER0_TEZ_INT. (R/W)

MCPWM_TIMER1_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER1_TEZ_INT. (R/W)

MCPWM_TIMER2_TEZ_INT_ENA Write 1 to enable MCPWM_TIMER2_TEZ_INT. (R/W)

MCPWM_TIMER0_TEP_INT_ENA Write 1 to enable MCPWM_TIMER0_TEP_INT. (R/W)

MCPWM_TIMER1_TEP_INT_ENA Write 1 to enable MCPWM_TIMER1_TEP_INT. (R/W)

MCPWM_TIMER2_TEP_INT_ENA Write 1 to enable MCPWM_TIMER2_TEP_INT. (R/W)

MCPWM_FAULT0_INT_ENA Write 1 to enable MCPWM_FAULT0_INT. (R/W)

MCPWM_FAULT1_INT_ENA Write 1 to enable MCPWM_FAULT1_INT. (R/W)

MCPWM_FAULT2_INT_ENA Write 1 to enable MCPWM_FAULT2_INT. (R/W)

MCPWM_FAULT0_CLR_INT_ENA Write 1 to enable MCPWM_FAULT0_CLR_INT. (R/W)

MCPWM_FAULT1_CLR_INT_ENA Write 1 to enable MCPWM_FAULT1_CLR_INT. (R/W)

MCPWM_FAULT2_CLR_INT_ENA Write 1 to enable MCPWM_FAULT2_CLR_INT. (R/W)

MCPWM_CMPR0_TEA_INT_ENA Write 1 to enable MCPWM_CMPR0_TEA_INT. (R/W)

MCPWM_CMPR1_TEA_INT_ENA Write 1 to enable MCPWM_CMPR1_TEA_INT. (R/W)

MCPWM_CMPR2_TEA_INT_ENA Write 1 to enable MCPWM_CMPR2_TEA_INT. (R/W)

MCPWM_CMPR0_TEB_INT_ENA Write 1 to enable MCPWM_CMPR0_TEB_INT. (R/W)

MCPWM_CMPR1_TEB_INT_ENA Write 1 to enable MCPWM_CMPR1_TEB_INT. (R/W)

MCPWM_CMPR2_TEB_INT_ENA Write 1 to enable MCPWM_CMPR2_TEB_INT. (R/W)

MCPWM_TZ0_CBC_INT_ENA Write 1 to enable MCPWM_TZ0_CBC_INT. (R/W)

Continued on the next page...

Register 24.29. MCPWM_INT_ENA_REG (0x0110)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_ENA Write 1 to enable MCPWM_TZ1_CBC_INT. (R/W)

MCPWM_TZ2_CBC_INT_ENA Write 1 to enable MCPWM_TZ2_CBC_INT. (R/W)

MCPWM_TZ0_OST_INT_ENA Write 1 to enable MCPWM_TZ0_OST_INT. (R/W)

MCPWM_TZ1_OST_INT_ENA Write 1 to enable MCPWM_TZ1_OST_INT. (R/W)

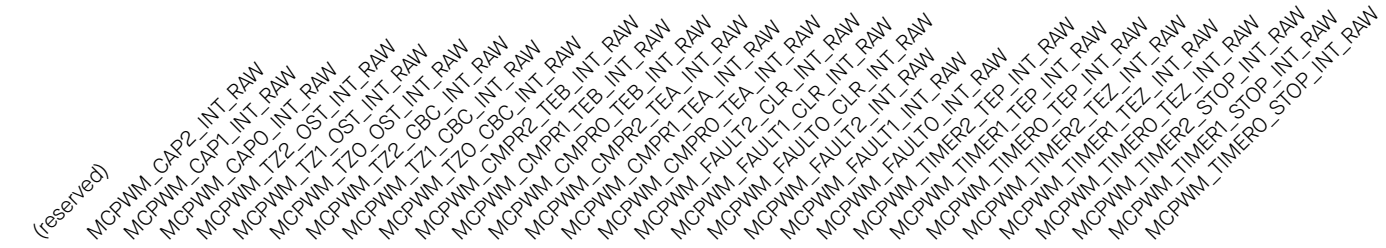
MCPWM_TZ2_OST_INT_ENA Write 1 to enable MCPWM_TZ2_OST_INT. (R/W)

MCPWM_CAP0_INT_ENA Write 1 to enable MCPWM_CAP0_INT. (R/W)

MCPWM_CAP1_INT_ENA Write 1 to enable MCPWM_CAP1_INT. (R/W)

MCPWM_CAP2_INT_ENA Write 1 to enable MCPWM_CAP2_INT. (R/W)

Register 24.30. MCPWM_INT_RAW_REG (0x0114)



31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_RAW Represents the raw status of MCPWM_TIMER0_STOP_INT. (R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW Represents the raw status of MCPWM_TIMER1_STOP_INT. (R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW Represents the raw status of MCPWM_TIMER2_STOP_INT. (R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER0_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER1_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW Represents the raw status of MCPWM_TIMER2_TEZ_INT. (R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW Represents the raw status of MCPWM_TIMER0_TEP_INT. (R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW Represents the raw status of MCPWM_TIMER1_TEP_INT. (R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW Represents the raw status of MCPWM_TIMER2_TEP_INT. (R/WTC/SS)

MCPWM_FAULT0_INT_RAW Represents the raw status of MCPWM_FAULT0_INT. (R/WTC/SS)

MCPWM_FAULT1_INT_RAW Represents the raw status of MCPWM_FAULT1_INT. (R/WTC/SS)

MCPWM_FAULT2_INT_RAW Represents the raw status of MCPWM_FAULT2_INT. (R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW Represents the raw status of MCPWM_FAULT0_CLR_INT. (R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW Represents the raw status of MCPWM_FAULT1_CLR_INT. (R/WTC/SS)

Continued on the next page...

Register 24.30. MCPWM_INT_RAW_REG (0x0114)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_RAW Represents the raw status of MCPWM_FAULT2_CLR_INT. (R/WTC/SS)

MCPWM_CMPR0_TEA_INT_RAW Represents the raw status of MCPWM_CMPR0_TEA_INT. (R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW Represents the raw status of MCPWM_CMPR1_TEA_INT. (R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW Represents the raw status of MCPWM_CMPR2_TEA_INT. (R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW Represents the raw status of MCPWM_CMPR0_TEB_INT. (R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW Represents the raw status of MCPWM_CMPR1_TEB_INT. (R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW Represents the raw status of MCPWM_CMPR2_TEB_INT. (R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW Represents the raw status of MCPWM_TZ0_CBC_INT. (R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW Represents the raw status of MCPWM_TZ1_CBC_INT. (R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW Represents the raw status of MCPWM_TZ2_CBC_INT. (R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW Represents the raw status of MCPWM_TZ0_OST_INT. (R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW Represents the raw status of MCPWM_TZ1_OST_INT. (R/WTC/SS)

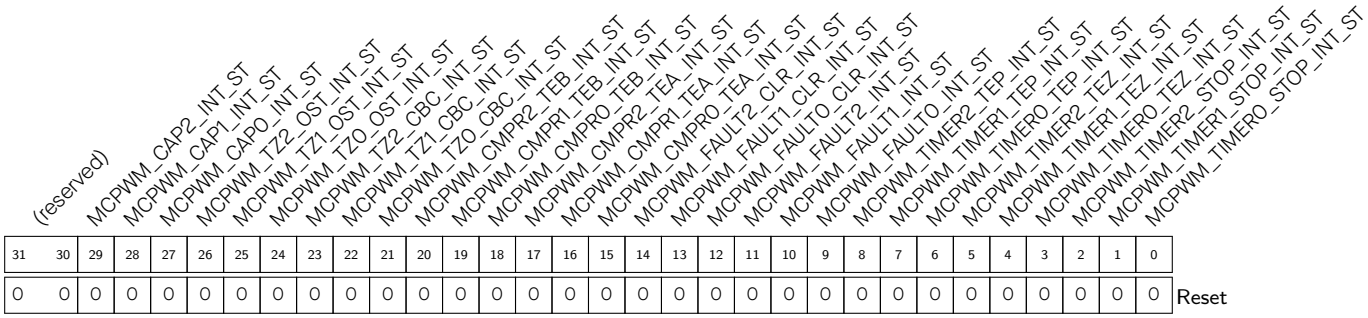
MCPWM_TZ2_OST_INT_RAW Represents the raw status of MCPWM_TZ2_OST_INT. (R/WTC/SS)

MCPWM_CAP0_INT_RAW Represents the raw status of MCPWM_CAP0_INT. (R/WTC/SS)

MCPWM_CAP1_INT_RAW Represents the raw status of MCPWM_CAP1_INT. (R/WTC/SS)

MCPWM_CAP2_INT_RAW Represents the raw status of MCPWM_CAP2_INT. (R/WTC/SS)

Register 24.31. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST Represents the masked status of MCPWM_TIMER0_STOP_INT. (RO)

MCPWM_TIMER1_STOP_INT_ST Represents the masked status of MCPWM_TIMER1_STOP_INT. (RO)

MCPWM_TIMER2_STOP_INT_ST Represents the masked status of MCPWM_TIMER2_STOP_INT. (RO)

MCPWM_TIMER0_TEZ_INT_ST Represents the masked status of MCPWM_TIMER0_TEZ_INT. (RO)

MCPWM_TIMER1_TEZ_INT_ST Represents the masked status of MCPWM_TIMER1_TEZ_INT. (RO)

MCPWM_TIMER2_TEZ_INT_ST Represents the masked status of MCPWM_TIMER2_TEZ_INT. (RO)

MCPWM_TIMER0_TEP_INT_ST Represents the masked status of MCPWM_TIMER0_TEP_INT. (RO)

MCPWM_TIMER1_TEP_INT_ST Represents the masked status of MCPWM_TIMER1_TEP_INT. (RO)

MCPWM_TIMER2_TEP_INT_ST Represents the masked status of MCPWM_TIMER2_TEP_INT. (RO)

MCPWM_FAULT0_INT_ST Represents the masked status of MCPWM_FAULT0_INT. (RO)

MCPWM_FAULT1_INT_ST Represents the masked status of MCPWM_FAULT1_INT. (RO)

MCPWM_FAULT2_INT_ST Represents the masked status of MCPWM_FAULT2_INT. (RO)

MCPWM_FAULT0_CLR_INT_ST Represents the masked status of MCPWM_FAULT0_CLR_INT. (RO)

MCPWM_FAULT1_CLR_INT_ST Represents the masked status of MCPWM_FAULT1_CLR_INT. (RO)

Continued on the next page...

Register 24.31. MCPWM_INT_ST_REG (0x0118)

Continued from the previous page...

MCPWM_FAULT2_CLR_INT_ST Represents the masked status of MCPWM_FAULT2_CLR_INT. (RO)

MCPWM_CMPR0_TEA_INT_ST Represents the masked status of MCPWM_CMPR0_TEA_INT. (RO)

MCPWM_CMPR1_TEA_INT_ST Represents the masked status of MCPWM_CMPR1_TEA_INT. (RO)

MCPWM_CMPR2_TEA_INT_ST Represents the masked status of MCPWM_CMPR2_TEA_INT. (RO)

MCPWM_CMPR0_TEB_INT_ST Represents the masked status of MCPWM_CMPR0_TEB_INT. (RO)

MCPWM_CMPR1_TEB_INT_ST Represents the masked status of MCPWM_CMPR1_TEB_INT. (RO)

MCPWM_CMPR2_TEB_INT_ST Represents the masked status of MCPWM_CMPR2_TEB_INT. (RO)

MCPWM_TZ0_CBC_INT_ST Represents the masked status of MCPWM_TZ0_CBC_INT_ST. (RO)

MCPWM_TZ1_CBC_INT_ST Represents the masked status of MCPWM_TZ1_CBC_INT_ST. (RO)

MCPWM_TZ2_CBC_INT_ST Represents the masked status of MCPWM_TZ2_CBC_INT_ST. (RO)

MCPWM_TZ0_OST_INT_ST Represents the masked status of MCPWM_TZ0_OST_INT. (RO)

MCPWM_TZ1_OST_INT_ST Represents the masked status of MCPWM_TZ1_OST_INT. (RO)

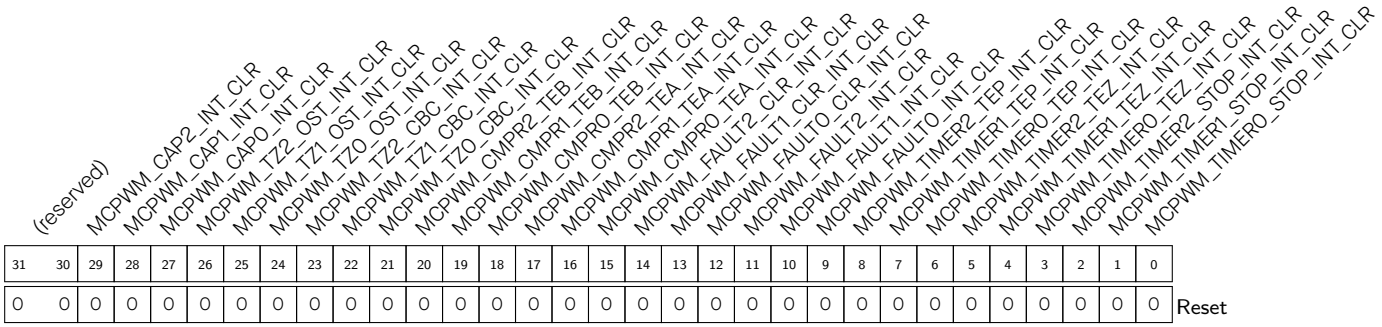
MCPWM_TZ2_OST_INT_ST Represents the masked status of MCPWM_TZ2_OST_INT. (RO)

MCPWM_CAP0_INT_ST Represents the masked status of MCPWM_CAP0_INT. (RO)

MCPWM_CAP1_INT_ST Represents the masked status of MCPWM_CAP1_INT. (RO)

MCPWM_CAP2_INT_ST Represents the masked status of MCPWM_CAP2_INT. (RO)

Register 24.32. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR Write 1 to clear MCPWM_TIMER0_STOP_INT. (WT)

MCPWM_TIMER1_STOP_INT_CLR Write 1 to clear MCPWM_TIMER1_STOP_INT. (WT)

MCPWM_TIMER2_STOP_INT_CLR Write 1 to clear MCPWM_TIMER2_STOP_INT. (WT)

MCPWM_TIMER0_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER0_TEZ_INT. (WT)

MCPWM_TIMER1_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER1_TEZ_INT. (WT)

MCPWM_TIMER2_TEZ_INT_CLR Write 1 to clear MCPWM_TIMER2_TEZ_INT. (WT)

MCPWM_TIMER0_TEP_INT_CLR Write 1 to clear MCPWM_TIMER0_TEP_INT. (WT)

MCPWM_TIMER1_TEP_INT_CLR Write 1 to clear MCPWM_TIMER1_TEP_INT. (WT)

MCPWM_TIMER2_TEP_INT_CLR Write 1 to clear MCPWM_TIMER2_TEP_INT. (WT)

MCPWM_FAULT0_INT_CLR Write 1 to clear MCPWM_FAULT0_INT. (WT)

MCPWM_FAULT1_INT_CLR Write 1 to clear MCPWM_FAULT1_INT. (WT)

MCPWM_FAULT2_INT_CLR Write 1 to clear MCPWM_FAULT2_INT. (WT)

MCPWM_FAULT0_CLR_INT_CLR Write 1 to clear MCPWM_FAULT0_CLR_INT. (WT)

MCPWM_FAULT1_CLR_INT_CLR Write 1 to clear MCPWM_FAULT1_CLR_INT. (WT)

MCPWM_FAULT2_CLR_INT_CLR Write 1 to clear MCPWM_FAULT2_CLR_INT. (WT)

MCPWM_CMPR0_TEA_INT_CLR Write 1 to clear MCPWM_CMPR0_TEA_INT. (WT)

MCPWM_CMPR1_TEA_INT_CLR Write 1 to clear MCPWM_CMPR1_TEA_INT. (WT)

MCPWM_CMPR2_TEA_INT_CLR Write 1 to clear MCPWM_CMPR2_TEA_INT. (WT)

MCPWM_CMPR0_TEB_INT_CLR Write 1 to clear MCPWM_CMPR0_TEB_INT. (WT)

MCPWM_CMPR1_TEB_INT_CLR Write 1 to clear MCPWM_CMPR1_TEB_INT. (WT)

MCPWM_CMPR2_TEB_INT_CLR Write 1 to clear MCPWM_CMPR2_TEB_INT. (WT)

MCPWM_TZ0_CBC_INT_CLR Write 1 to clear MCPWM_TZ0_CBC_INT. (WT)

Continued on the next page...

Register 24.32. MCPWM_INT_CLR_REG (0x011C)

Continued from the previous page...

MCPWM_TZ1_CBC_INT_CLR Write 1 to clear MCPWM_TZ1_CBC_INT. (WT)

MCPWM_TZ2_CBC_INT_CLR Write 1 to clear MCPWM_TZ2_CBC_INT. (WT)

MCPWM_TZ0_OST_INT_CLR Write 1 to clear MCPWM_TZ0_OST_INT. (WT)

MCPWM_TZ1_OST_INT_CLR Write 1 to clear MCPWM_TZ1_OST_INT. (WT)

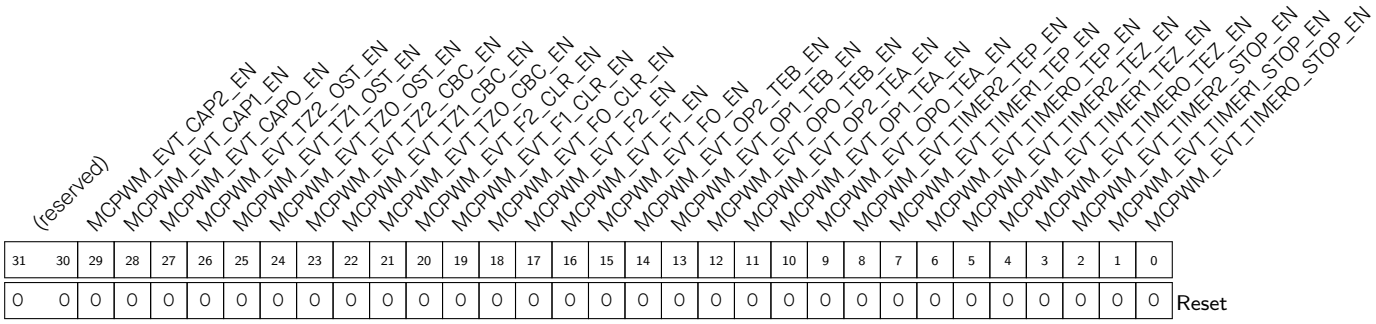
MCPWM_TZ2_OST_INT_CLR Write 1 to clear MCPWM_TZ2_OST_INT. (WT)

MCPWM_CAP0_INT_CLR Write 1 to clear MCPWM_CAP0_INT. (WT)

MCPWM_CAP1_INT_CLR Write 1 to clear MCPWM_CAP1_INT. (WT)

MCPWM_CAP2_INT_CLR Write 1 to clear MCPWM_CAP2_INT. (WT)

Register 24.33. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN Configures whether to enable timer0 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_STOP_EN Configures whether to enable timer1 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_STOP_EN Configures whether to enable timer2 stop event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN Configures whether to enable timer0 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN Configures whether to enable timer1 equal zero event generation.

0: Disable

1: Enable

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN Configures whether to enable timer2 equal zero event generation.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 24.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TIMER0_TEP_EN Configures whether to enable timer0 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER1_TEP_EN Configures whether to enable timer1 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TIMER2_TEP_EN Configures whether to enable timer2 equal period event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEA_EN Configures whether to enable PWM generator0 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP1_TEA_EN Configures whether to enable PWM generator1 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEA_EN Configures whether to enable PWM generator2 timer equal A event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP0_TEB_EN Configures whether to enable PWM generator0 timer equal B event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 24.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_OP1_TEB_EN Configures whether to enable PWM generator1 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_OP2_TEB_EN Configures whether to enable PWM generator2 timer equal B event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_EN Configures whether to enable FAULT0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_EN Configures whether to enable FAULT1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_EN Configures whether to enable FAULT2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F0_CLR_EN Configures whether to enable FAULT0 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F1_CLR_EN Configures whether to enable FAULT1 clear event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_F2_CLR_EN Configures whether to enable FAULT2 clear event generation.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 24.33. MCPWM_EVT_EN_REG (0x0120)

Continued from the previous page...

MCPWM_EVT_TZ0_CBC_EN Configures whether to enable cycle by cycle trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_CBC_EN Configures whether to enable cycle by cycle trip1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ2_CBC_EN Configures whether to enable cycle by cycle trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ0_OST_EN Configures whether to enable one shot trip0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_TZ1_OST_EN Configures whether to enable one shot trip1 event generation.
0: Disable
1: Enable
(R/W)

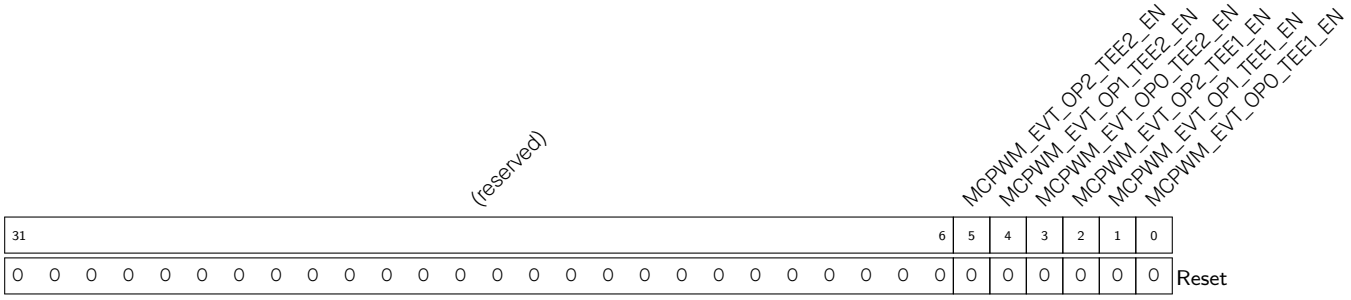
MCPWM_EVT_TZ2_OST_EN Configures whether to enable one shot trip2 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP0_EN Configures whether to enable capture0 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP1_EN Configures whether to enable capture1 event generation.
0: Disable
1: Enable
(R/W)

MCPWM_EVT_CAP2_EN Configures whether to enable capture2 event generation.
0: Disable
1: Enable
(R/W)

Register 24.34. MCPWM_EVT_EN2_REG (0x0128)



MCPWM_EVT_OP*n*_TEE1_EN Configures whether to generate the MCPWM_EVT_OP*n*_TEE1 event when the PWM generator*n* timer equals OP*n*_TSTMP_E1_REG.

0: Not generate

1: Generate

(R/W)

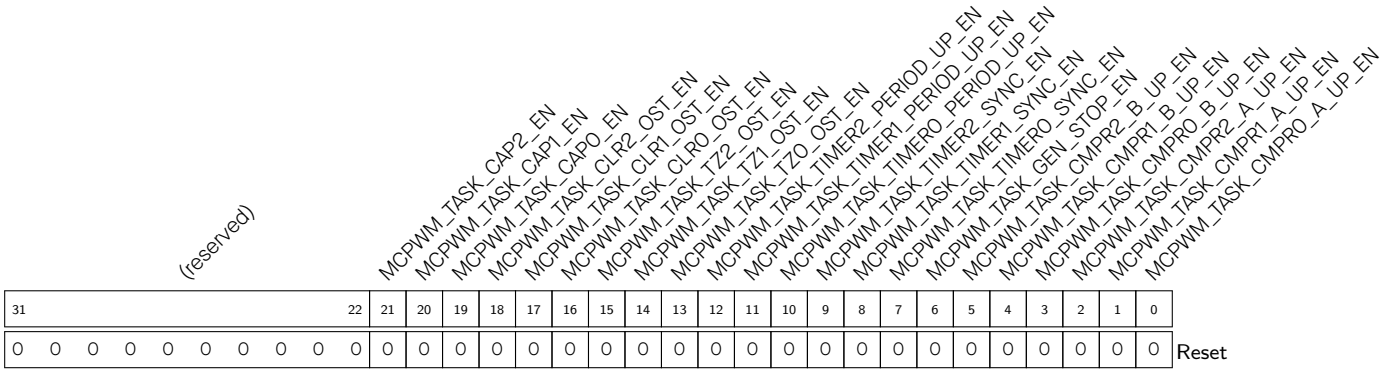
MCPWM_EVT_OP*n*_TEE2_EN Configures whether to generate the MCPWM_EVT_OP*n*_TEE2 event when the PWM generator*n* timer equals OP*n*_TSTMP_E2_REG.

0: Not generate

1: Generate

(R/W)

Register 24.35. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN Configures whether to receive update task of PWM generator0 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN Configures whether to receive update task of PWM generator1 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN Configures whether to receive update task of PWM generator2 timer stamp A's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN Configures whether to receive update task of PWM generator0 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN Configures whether to receive update task of PWM generator1 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN Configures whether to receive update task of PWM generator2 timer stamp B's shadow register.

0: No effect

1: Receive

(R/W)

Continued on the next page...

Register 24.35. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_GEN_STOP_EN Configures whether to receive all PWM generate stop task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN Configures whether to receive timer0 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN Configures whether to receive timer1 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN Configures whether to receive timer2 sync task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN Configures whether to receive timer0 period update task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN Configures whether to receive timer1 period update task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN Configures whether to receive timer2 period update task.

0: No effect

1: Receive

(R/W)

Continued on the next page...

Register 24.35. MCPWM_TASK_EN_REG (0x0124)

Continued from the previous page...

MCPWM_TASK_TZ0_OST_EN Configures whether to receive one shot trip0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ1_OST_EN Configures whether to receive one shot trip1 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_TZ2_OST_EN Configures whether to receive one shot trip2 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR0_OST_EN Configures whether to receive one shot trip0 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR1_OST_EN Configures whether to receive one shot trip1 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CLR2_OST_EN Configures whether to receive one shot trip2 clear task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP0_EN Configures whether to receive capture0 task.

0: No effect

1: Receive

(R/W)

MCPWM_TASK_CAP1_EN Configures whether to receive capture1 task.

0: No effect

1: Receive

(R/W)

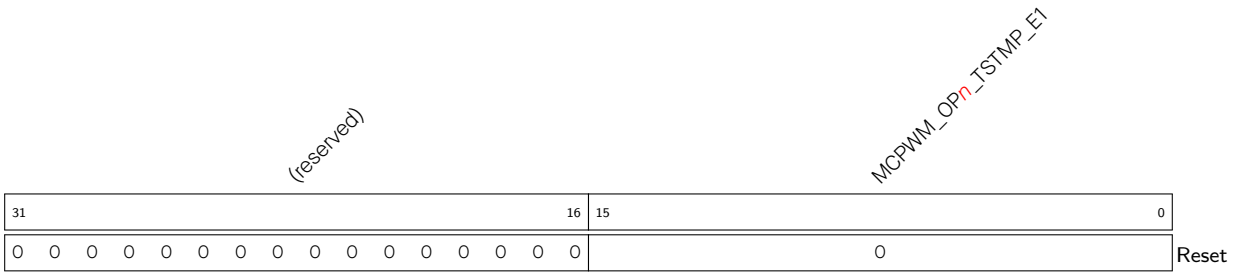
MCPWM_TASK_CAP2_EN Configures whether to receive capture2 task.

0: No effect

1: Receive

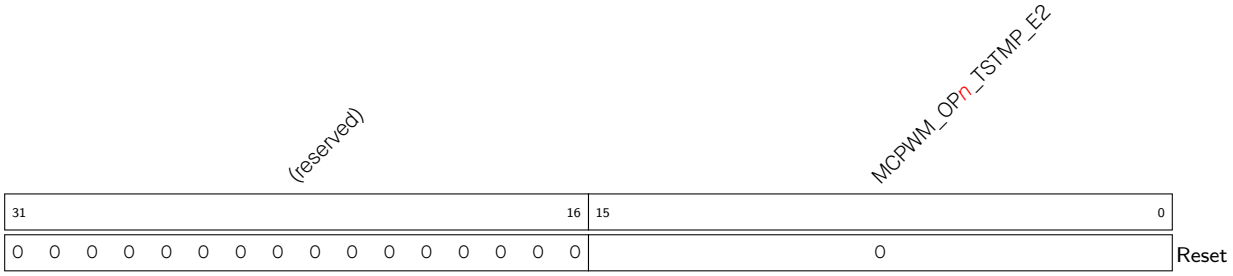
(R/W)

Register 24.36. MCPWM_OP*n*_TSTMP_E1_REG (*n*: 0-2) (0x012C+0x8**n*)



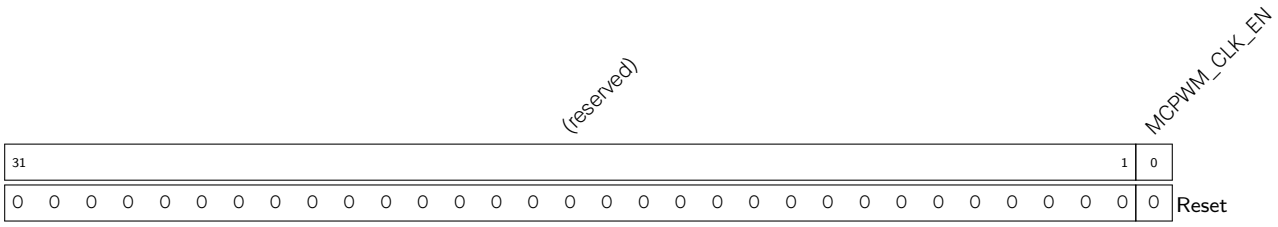
MCPWM_OP*n*_TSTMP_E1 Configures the time stamp E1 value of the generator*n*. (R/W)

Register 24.37. MCPWM_OP*n*_TSTMP_E2_REG(*n*: 0-2) (0x0130+0x8**n*)



MCPWM_OP*n*_TSTMP_E2 Configures the time stamp E2 value of the generator*n*. (R/W)

Register 24.38. MCPWM_CLK_REG (0x0144)



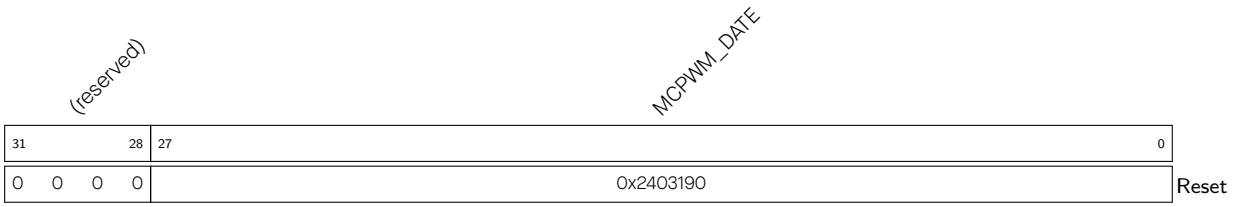
MCPWM_CLK_EN Configures whether to force open register clock gate.

0: Open the clock gate only when application writes registers

1: Force open the clock gate for register

(R/W)

Register 24.39. MCPWM_VERSION_REG (0x0148)



MCPWM_DATE Version control register. (R/W)

Chapter 25

Remote Control Peripheral (RMT)

25.1 Overview

The Remote Control (RMT) module is designed to transmit and receive infrared remote control signals. A variety of remote control protocols can be encoded/decoded via software based on the RMT module. The RMT module converts pulse codes stored in the module's built-in RAM into output signals, or converts input signals into pulse codes and stores them in RAM. In addition, the RMT module optionally modulates its output signals with a carrier wave, or optionally demodulates and filters its input signals.

The RMT module has eight channels, numbered from zero to seven. Each channel is able to independently transmit or receive signals.

- Channel 0 ~ 3 (TX channel) are dedicated to transmitting signals;
- Channel 4 ~ 7 (RX channel) are dedicated to receiving signals.

Each TX/RX channel has the same functionality controlled by a dedicated set of registers and is able to independently transmit or receive data. Channel 3 and channel 7 support GDMA access, so the two channels also have a set of GDMA-related control and status registers. TX channels are indicated by *n* which is used as a placeholder for the channel number, and by *m* for RX channels.

25.2 Features

The RMT module has the following features:

- Eight channels:
  - TX channels 0 ~ 3
  - RX channels 4 ~ 7
  - Eight channels share a 384 x 32-bit RAM
- The transmitter supports:
  - Normal TX mode
  - Wrap TX mode
  - Continuous TX mode
  - Modulation on TX pulses
  - Multiple channels transmitting data simultaneously (programmable)
  - GDMA access supported by TX channel 3

- The receiver supports:
  - Normal RX mode
  - Wrap RX mode
  - RX filtering
  - Demodulation on RX pulses
  - GDMA access supported by RX channel 7

25.3 Functional Description

25.3.1 Architecture

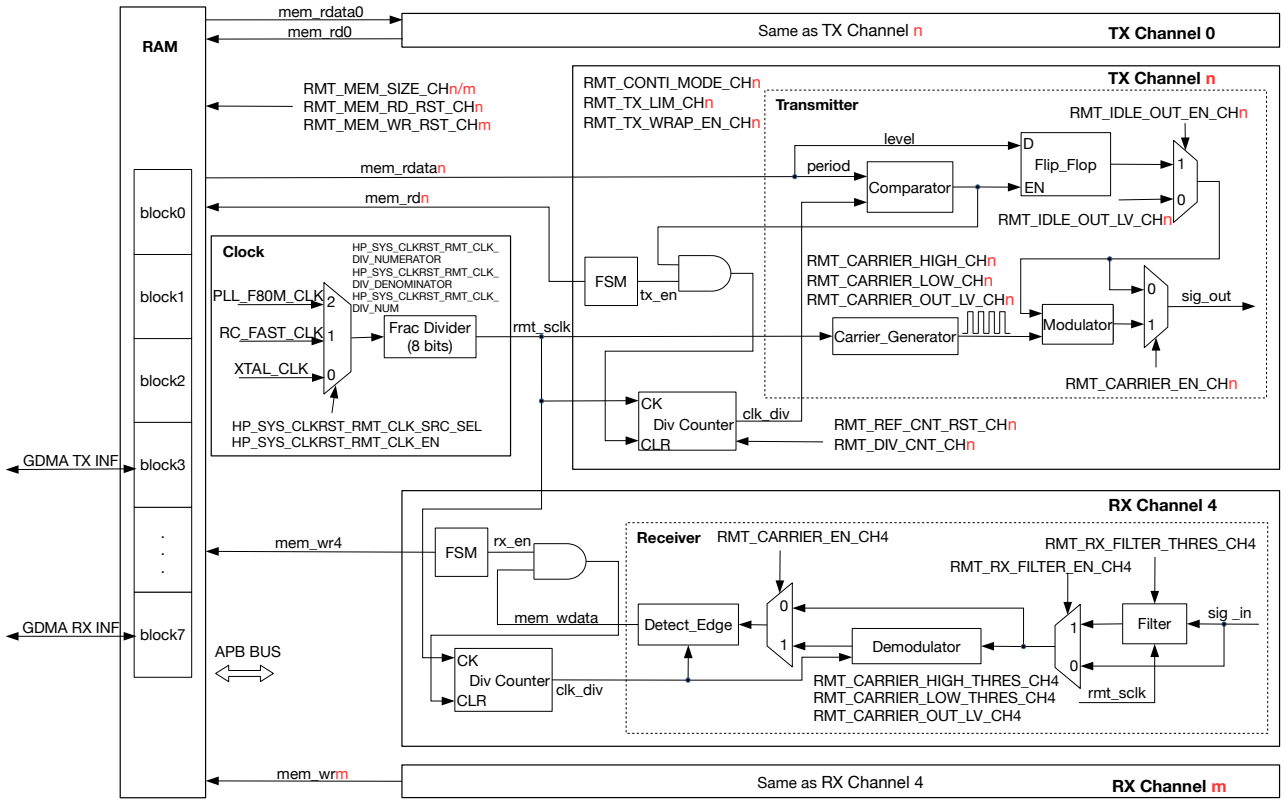


Figure 25.3-1. RMT Architecture

As shown in Figure 25.3-1, each TX channel has:

- 1 x clock divider counter (Div Counter)
- 1 x state machine (FSM)
- 1 x transmitter

Each RX channel has:

- 1 x clock divider counter (Div Counter)

- 1 x state machine (FSM)
- 1 x receiver

The eight channels share a 384 x 32-bit RAM.

25.3.2 RAM

25.3.2.1 Structure of RAM

Figure 25.3-2 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 25.3-1) that the level lasts for.

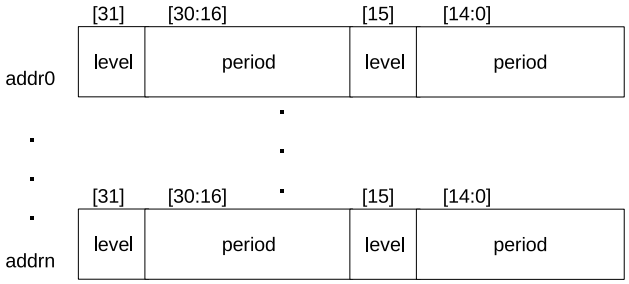


Figure 25.3-2. Format of Pulse Code in RAM

The minimum value for the period is zero (0) and is interpreted as a transmission end-marker. For a non-zero period (i.e., not an end-marker), its value is limited by APB clock and RMT clock according to the formula below:

$$5 \times T_{apb\_clk} + 6 \times T_{rmt\_sclk} < period \times T_{clk\_div} \tag{25.1}$$

Note:
RMT clock and APB clock must satisfy the following relation:

$$1.5 \times T_{apb\_clk} < 9 \times T_{rmt\_sclk} \tag{25.2}$$

According to the formula above and the frequency of rmt_sclk, the pulse width (i.e., $period \times T_{clk\_div}$) able to be captured by RMT is limited as follows:

- The minimum value of pulse width should be larger than $(5 \times T_{apb\_clk} + 6 \times T_{rmt\_sclk})$.
- The maximum value of pulse width should be smaller than or equal to (the maximum period $\times$ the maximum T_{clk_div}), i.e., $((2^{15} - 1) \times$ the maximum $T_{rmt\_sclk} \times 256)$.

For more information about the rmt_sclk frequency, or APB_CLK frequency, see Section 25.3.3, or Chapter 7 *Reset and Clock [to be added later]*.

25.3.2.2 Use of RAM

The RAM is divided into eight 48 x 32-bit blocks. By default, each channel uses one block (block 0 for channel 0, block 1 for channel 1, and so on).

If the data size of one single transfer is larger than one block size of TX channel *n* or RX channel *m*, users can configure the channel:

- to enable wrap mode by setting RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m*;
- or to use more blocks by configuring RMT_MEM_SIZE_CH*n*/*m*.

Setting RMT_MEM_SIZE_CH*n*/*m* > 1 allows channel *n*/*m* to use the memory of the subsequent channels, i.e., block (*n*/*m*) ~ block (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* - 1). In such case, the subsequent channels (*n*/*m* + 1) ~ (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* - 1) can not be used since their RAM blocks are occupied. For example, if channel 0 is configured to use block 0 and block 1, then channel 1 will be unavailable since its block is occupied, while channel 2 and channel 3 are not affected and can be used normally.

Note that the RAM used by each channel is mapped from low address to high address. Under such mapping, channel 0 is able to use the RAM blocks for channels 1, 2 ... and 7 by setting RMT_MEM_SIZE_CH0, but channel 7 can not use the blocks for channels 0, 1, ... or 6. Therefore, the value of RMT_MEM_SIZE_CH*n* should not exceed (8 - *n*) and the value of RMT_MEM_SIZE_CH*m* should not exceed (8 - *m*).

The RMT RAM can be accessed via APB bus, or read by the transmitter and written by the receiver. To avoid any possible access conflict between the receiver writing RAM and the APB bus reading RAM, RMT can be configured to designate the RAM block's owner, be it the receiver or the APB bus, by configuring RMT_MEM_OWNER_CH*m*. If this ownership is violated, a flag signal RMT_MEM_OWNER_ERR_CH*m* will be generated.

25.3.2.3 RAM Access

APB bus is able to access RAM in FIFO mode and in NONFIFO (Direct Address) mode, depending on the configuration of RMT_APB_FIFO_MASK:

- 0: Use FIFO mode;
- 1: Use NONFIFO mode.

Channels 3 and 7 also support GDMA access.

FIFO Mode

In FIFO mode, the APB reads data from or writes data to RAM via a fixed address stored in RMT_CH*n*/*m*DATA_REG.

NONFIFO Mode

In NONFIFO mode, the APB writes data to or reads data from a continuous address range.

- The write-starting address of TX channel *n* is: RMT base address + 0x800 + *n*x48. The access address for the second data and the following data are RMT base address + 0x800 + *n*x48 + 0x4, and so on, incremented by 0x4.
- The read-starting address of RX channel *m* is: RMT base address + 0x8C0 + *m*x48. The access address for the second data and the following data are RMT base address + 0x8C0 + *m*x48 + 0x4, and so on, incremented by 0x4.

GDMA Mode

Channel 3 also supports GDMA access. If RMT_DMA_ACCESS_EN_CH3 is set, RAM of channel 3 only allows GDMA access. FIFO access or NOFIFO access to channel 3 by the APB bus are forbidden, otherwise **unpredictable consequences may occur.**

To ensure correct data transmission,

1. GDMA should be started first.
2. RMT can only start transmitting data after GDMA channel gets data ready, otherwise, unexpected data may be sent.

In normal TX mode, when the RAM of channel 3 is fully written by GDMA, an RMT_APB_MEM_WR_ERR_CH3 interrupt is triggered. Setting RMT_MEM_TX_WRAP_EN_CH3 allows channel 3 to transmit more data than one block can fit, with no software operation needed.

Channel 7 also supports GDMA access. If RMT_DMA_ACCESS_EN_CH7 is set, the RAM of channel 7 is allowed to transmit data to GDMA. Note in this mode, channel 7's RAM can also be accessed by APB via NONFIFO mode.

In normal RX mode, when the size of data read by GDMA from channel 7 is equal to its RAM size, an RMT_APB_MEM_RD_ERR_CH7 is triggered and the subsequent data is discarded. If RMT_MEM_RX_WRAP_EN_CH7 is set, data of more than one block size can be received with no software wrap operation needed. If channel 7's RAM is full but the GDMA still does not start receiving data from the channel, the newly received data by this channel will replace the previous data.

Note:
When channel 7 receives an end-maker, a GDMA in_suc_eof interrupt is generated. Two bytes are written to GDMA if the period[14:0] is 0, and four bytes to GDMA if the period[30:16] is 0.

25.3.3 Clock

The clock source of RMT can be PLL_F80M_CLK, RC_FAST_CLK, or XTAL_CLK, depending on the configuration of HP_SYS_CLKRST_RMT_CLK_SRC_SEL. RMT clock can be enabled by setting HP_SYS_CLKRST_RMT_CLK_EN. RMT working clock (see rmt_sclk in Figure 25.3-1) is obtained by dividing the selected clock source with a fractional divider. The divider is HP_SYS_CLKRST_RMT_CLK_DIV_NUM + 1 + HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR/HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR.

For more information, see Chapter 7 *Reset and Clock [to be added later]*. RMT_DIV_CNT_CH*n*/*m* is used to configure the divider coefficient of internal clock divider for RMT channels. The coefficient is normally equal to the value of RMT_DIV_CNT_CH*n*/*m*, except value 0 that represents divider 256. The clock divider can be reset by setting RMT_REF_CNT_RST_CH*n*/*m*. The clock generated from the divider can be used by the counter (see Figure 25.3-1).

25.3.4 Transmitter

Note:
Updating the configuration described in this and subsequent sections requires setting RMT_CONF_UPDATE_CH*n*/*m* first. See Section 25.3.6.

25.3.4.1 Normal TX Mode

When RMT_TX_START_CH*n* is set, the transmitter of channel *n* starts reading and transmitting pulse codes from the starting address of its RAM block. The codes are sent starting from low-address entry. When an end-marker (a zero period) is encountered, the transmitter stops the transmission, returns to idle state and generates an RMT_CH*n*_TX_END_INT interrupt. Setting RMT_TX_STOP_CH*n* to 1 also stops the transmission and immediately sets the transmitter back to idle. The output level of a transmitter in idle state is determined by the "level" field of the end-marker or by the content of RMT_IDLE_OUT_LV_CH*n*, depending on the configuration of RMT_IDLE_OUT_EN_CH*n*:

- 0: The level in idle state is determined by the "level" field of the end-marker;
- 1: The level is determined by RMT_IDLE_OUT_LV_CH*n*.

25.3.4.2 Wrap TX Mode

To transmit more pulse codes than that can be fitted in the channel's RAM, users can enable wrap TX mode for channel *n* by setting RMT_MEM_TX_WRAP_EN_CH*n*. In this mode, the transmitter transmits the data from RAM in loops till an end-marker is encountered. For example, if RMT_MEM_SIZE_CH*n* = 1, the transmitter starts transmitting data from the address 48 x *n*, and then the data from higher RAM address. Once the transmitter finishes transmitting data from (48 x (*n* + 1) - 1), it continues transmitting data from 48 x *n* again till an end-marker is encountered. Wrap mode is also applicable for RMT_MEM_SIZE_CH*n* > 1.

When the size of transmitted pulse codes is larger than or equal to the value set by RMT_TX_LIM_CH*n*, an RMT_CH*n*_TX_THR_EVENT_INT interrupt is generated. In wrap mode, RMT_TX_LIM_CH*n* can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*n*_TX_THR_EVENT_INT interrupt is detected by software, the already used RAM region can be updated by new pulse codes. In such way, the transmitter can seamlessly transmit unlimited pulse codes in wrap mode.

Note:
If RAM is accessed by GDMA, more pulse codes than one block size can be transmitted with no additional operation needed. If accessed by APB bus, wrap mode has to be enabled via software to transmit more data than one block size.

25.3.4.3 TX Modulation

Transmitter output can be modulated with a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier waveform is configurable. In a carrier cycle, the high level lasts for (RMT_CARRIER_HIGH_CH*n* + 1) rmt_sclk cycles, while the low level lasts for (RMT_CARRIER_LOW_CH*n* + 1) rmt_sclk cycles. When RMT_CARRIER_OUT_LV_CH*n* is set, carrier wave is added on the high-level of output signals; while RMT_CARRIER_OUT_LV_CH*n* is cleared, carrier wave is added on the low-level of output signals. Carrier wave can be added on all output signals during modulation, or just added on valid pulse codes (the data stored in RAM), which can be set by configuring RMT_CARRIER_EFF_EN_CH*n*:

- 0: Add carrier wave on all output signals;

- 1: Add carrier wave only on valid signals.

25.3.4.4 Continuous TX Mode

The continuous TX mode can be enabled by setting RMT_TX_CONTI_MODE_CH*n*. In this mode, the transmitter transmits the pulse codes from RAM in loops:

- If an end-marker is encountered, the transmitter starts transmitting from the first data of the channel's RAM again.
- If no end-marker is encountered, the transmitter starts transmitting from the first data again after the last data is transmitted.

If RMT_TX_LOOP_CNT_EN_CH*n* is set, the loop counting is incremented by 1 each time an end-marker is encountered. If the counting reaches the value set by RMT_TX_LOOP_NUM_CH*n*, an RMT_CH*n*_TX_LOOP_INT interrupt is generated. If RMT_LOOP_STOP_EN_CH*n* is set, the transmission stops immediately once an RMT_CH*n*_TX_LOOP_INT interrupt is generated, otherwise, the transmission will continue. In an end-maker, if its period[14:0] is 0, then the period of the previous data must satisfy:

$$10 \times T_{apb\_clk} + 19 \times T_{rmt\_sclk} < period \times T_{clk\_div} \tag{25.3}$$

The period of the other data only need to satisfy non-zero period relation.

25.3.4.5 Simultaneous TX Mode

RMT module supports multiple channels transmitting data simultaneously. To use this function, follow the steps below:

1. Configure RMT_TX_SIM_CH*n* to choose which multiple channels are used to transmit data simultaneously.
2. Set RMT_TX_SIM_EN to enable this transmission mode.
3. Set RMT_TX_START_CH*n* for each selected channel, to start data transmitting.

The transmission starts once the final channel is configured. RMT module also supports simultaneous transmission of channels 0 ~ 2's RAM accessed by APB bus and channel 3's RAM accessed by GDMA.

25.3.5 Receiver

25.3.5.1 Normal RX Mode

The receiver of channel *m* is controlled by RMT_RX_EN_CH*m*:

- 0: The receiver stops receiving data;
- 1: The receiver starts working.

When the receiver becomes active, it starts counting from the first edge of the signal, detecting signal levels and counting clock cycles the level lasts for. Each cycle count (period) is then written back to RAM together with the level information (level). When the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*, the receiver will stop receiving data, return to idle state, and generate an RMT_CH*m*_RX_END_INT interrupt. Please note that RMT_IDLE_THRES_CH*m* should be

configured to a maximum value according to your application, otherwise a valid received level may be mistaken as a level in idle state. If the RAM space of this RX channel is used up by the received data, the receiver stops receiving data, and an RMT_CH*m*_ERR_INT interrupt is triggered by RAM FULL event.

25.3.5.2 Wrap RX Mode

To receive more pulse codes than can be fitted in the channel's RAM, users can enable wrap mode for channel *m* by configuring RMT_MEM_RX_WRAP_EN_CH*m*. But if RAM is accessed by GDMA, more pulse codes than one block size can be received with no additional operation needed. If accessed by APB bus, wrap mode has to be enabled to transmit more data than one block size. In wrap mode, the receiver stores the received data to RAM space of this channel in loops. Receiving ends, when the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*m*. The receiver returns to idle state and generates an RMT_CH*m*_RX_END_INT interrupt. For example, if RMT_MEM_SIZE_CH*m* is set to 1, the receiver starts receiving data and stores the data to address 48 * *m*, and then to higher RAM address. When the receiver finishes storing the received data to (48 * (*m* + 1) - 1), the receiver continues receiving data and storing data to the address 48 * *m* again, till no change is detected on a signal level for more than RMT_IDLE_THRES_CH*m* clock cycles. Wrap mode is also applicable for RMT_MEM_SIZE_CH*m* > 1.

An RMT_CH*m*_RX_THR_EVENT_INT interrupt is generated when the size of received pulse codes is larger than or equal to the value set by RMT_CH*m*_RX_LIM_REG. In wrap mode, RMT_CH*m*_RX_LIM_REG can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*m*_RX_THR_EVENT_INT interrupt is detected, the already used RAM region can be updated by subsequent data.

25.3.5.3 RX Filtering

Users can enable the receiver to filter input signals by setting RMT_RX_FILTER_EN_CH*m* for channel *m*. The filter samples input signals continuously, and detects the signals which remain unchanged for a continuous RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles as valid, otherwise, the signals will be detected as invalid. Only the valid signals can pass through this filter. The filter removes pulses with a length of less than RMT_RX_FILTER_THRES_CH*m* rmt_sclk cycles.

25.3.5.4 RX Demodulation

Users can enable RX demodulation on input signals or on filtered signals by setting RMT_CARRIER_EN_CH*m*. RX demodulation can be applied to high-level carrier wave or low-level carrier wave, depending on the configuration of RMT_CARRIER_OUT_LV_CH*m*:

- 0: Demodulate low-level carrier wave;
- 1: Demodulate high-level carrier wave.

Users can configure RMT_CARRIER_HIGH_THRES_CH*m* and RMT_CARRIER_LOW_THRES_CH*m* to set the thresholds to demodulate high-level carrier or low-level carrier. If the high-level of a signal lasts for less than RMT_CARRIER_HIGH_THRES_CH*m* clk_div cycles, or the low-level lasts for less than RMT_CARRIER_LOW_THRES_CH*m* clk_div cycles, such level is detected as a carrier and then is filtered out.

25.3.6 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 25.3-1 should follow this rule.

Table 25.3-1. Configuration Update

Register	Bit/Field Configuration Update
TX Channel	
RMT_CH*n*CONFO_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_TX_STOP_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
RX Channel	
RMT_CH*m*CONFO_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

25.4 Interrupts

ESP32-S31's RMT can generate the RMT_INTR interrupt signal that will be sent to the Interrupt Matrix.

The following interrupt sources can generate the RMT_INTR interrupt signal:

- RMT_CH*n*_ERR_INT: Triggered when channel *n* does not read or write data correctly. For example, the transmitter still tries to read data from RAM when the RAM is empty.
- RMT_CH*m*_ERR_INT: Triggered when channel *m* does not read or write data correctly. For example, the receiver still tries to write data into RAM when the RAM is full.
- RMT_CH*n*_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the value of RMT_CH*n*_TX_LIM_REG.
- RMT_CH*m*_RX_THR_EVENT_INT: Triggered each time when the amount of data received by the receiver reaches the value set in RMT_CH*m*_RX_LIM_REG.
- RMT_CH*n*_TX_END_INT: Triggered when the transmitter has finished transmitting signals.

- RMT_CH*m*_RX_END_INT: Triggered when the receiver has finished receiving signals.
- RMT_CH*n*_TX_LOOP_INT: Triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CH*n* in continuous TX mode.
- RMT_CH3_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries written to channel 3's RAM - the entries transmitted by channel 3) is larger than channel 3's RAM size, but DAM keeps writing data to this channel.
- RMT_CH7_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries received by channel 7's RAM - the entries read by GDMA) is larger than channel 7's RAM size, but channel 7 keeps receiving data.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

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 25.5 *Register Summary*.

25.5 Register Summary

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
FIFO R/W registers			
RMT_CH0DATA_REG	The read and write data register for channel 0 by APB FIFO access	0x0000	HRO
RMT_CH1DATA_REG	The read and write data register for channel 1 by APB FIFO access	0x0004	HRO
RMT_CH2DATA_REG	The read and write data register for channel 2 by APB FIFO access	0x0008	HRO
RMT_CH3DATA_REG	The read and write data register for channel 3 by APB FIFO access	0x000C	HRO
RMT_CH4DATA_REG	The read and write data register for channel 4 by APB FIFO access	0x0010	HRO
RMT_CH5DATA_REG	The read and write data register for channel 5 by APB FIFO access	0x0014	HRO
RMT_CH6DATA_REG	The read and write data register for channel 6 by APB FIFO access	0x0018	HRO
RMT_CH7DATA_REG	The read and write data register for channel 7 by APB FIFO access	0x001C	HRO
Configuration registers			
RMT_CH0CONF0_REG	Configuration register 0 for channel 0	0x0020	varies
RMT_CH1CONF0_REG	Configuration register 0 for channel 1	0x0024	varies
RMT_CH2CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
RMT_CH3CONF0_REG	Configuration register 0 for channel 3	0x002C	varies
RMT_CH4CONF0_REG	Configuration register 0 for channel 4	0x0030	R/W
RMT_CH4CONF1_REG	Configuration register 1 for channel 4	0x0034	varies
RMT_CH5CONF0_REG	Configuration register 0 for channel 5	0x0038	R/W
RMT_CH5CONF1_REG	Configuration register 1 for channel 5	0x003C	varies
RMT_CH6CONF0_REG	Configuration register 0 for channel 6	0x0040	R/W
RMT_CH6CONF1_REG	Configuration register 1 for channel 6	0x0044	varies
RMT_CH7CONF0_REG	Configuration register 0 for channel 7	0x0048	R/W
RMT_CH7CONF1_REG	Configuration register 1 for channel 7	0x004C	varies
RMT_CH4_RX_CARRIER_RM_REG	Carrier remove register for channel 4	0x0090	R/W
RMT_CH5_RX_CARRIER_RM_REG	Carrier remove register for channel 5	0x0094	R/W
RMT_CH6_RX_CARRIER_RM_REG	Carrier remove register for channel 6	0x0098	R/W
RMT_CH7_RX_CARRIER_RM_REG	Carrier remove register for channel 7	0x009C	R/W
RMT_SYS_CONF_REG	RMT APB configuration register	0x00C0	R/W
RMT_REF_CNT_RST_REG	RMT clock divider reset register	0x00C8	WT
Status registers			

Name	Description	Address	Access
RMT_CH0STATUS_REG	Status register for channel 0	0x0050	RO
RMT_CH1STATUS_REG	Status register for channel 1	0x0054	RO
RMT_CH2STATUS_REG	Status register for channel 2	0x0058	RO
RMT_CH3STATUS_REG	Status register for channel 3	0x005C	RO
RMT_CH4STATUS_REG	Status register for channel 4	0x0060	RO
RMT_CH5STATUS_REG	Status register for channel 5	0x0064	RO
RMT_CH6STATUS_REG	Status register for channel 6	0x0068	RO
RMT_CH7STATUS_REG	Status register for channel 7	0x006C	RO
Interrupt registers			
RMT_INT_RAW_REG	Interrupt raw signal status register	0x0070	R/WTC/SS
RMT_INT_ST_REG	Interrupt signal status register	0x0074	RO
RMT_INT_ENA_REG	Interrupt enable signal configuration register	0x0078	R/W
RMT_INT_CLR_REG	Interrupt clear signal configuration register	0x007C	WT
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Duty cycle configuration register for channel 0	0x0080	R/W
RMT_CH1CARRIER_DUTY_REG	Duty cycle configuration register for channel 1	0x0084	R/W
RMT_CH2CARRIER_DUTY_REG	Duty cycle configuration register for channel 2	0x0088	R/W
RMT_CH3CARRIER_DUTY_REG	Duty cycle configuration register for channel 3	0x008C	R/W
TX event configuration registers			
RMT_CH0_TX_LIM_REG	Configuration register for channel 0 TX event	0x00A0	varies
RMT_CH1_TX_LIM_REG	Configuration register for channel 1 TX event	0x00A4	varies
RMT_CH2_TX_LIM_REG	Configuration register for channel 2 TX event	0x00A8	varies
RMT_CH3_TX_LIM_REG	Configuration register for channel 3 TX event	0x00AC	varies
RMT_TX_SIM_REG	RMT TX synchronous register	0x00C4	R/W
RX event configuration registers			
RMT_CH4_RX_LIM_REG	Configuration register for channel 4 RX event	0x00B0	R/W
RMT_CH5_RX_LIM_REG	Configuration register for channel 5 RX event	0x00B4	R/W
RMT_CH6_RX_LIM_REG	Configuration register for channel 6 RX event	0x00B8	R/W
RMT_CH7_RX_LIM_REG	Configuration register for channel 7 RX event	0x00BC	R/W
Version register			
RMT_DATE_REG	Version control register	0x00CC	R/W

25.6 Registers

The addresses in this section are relative to Remote Control Peripheral base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

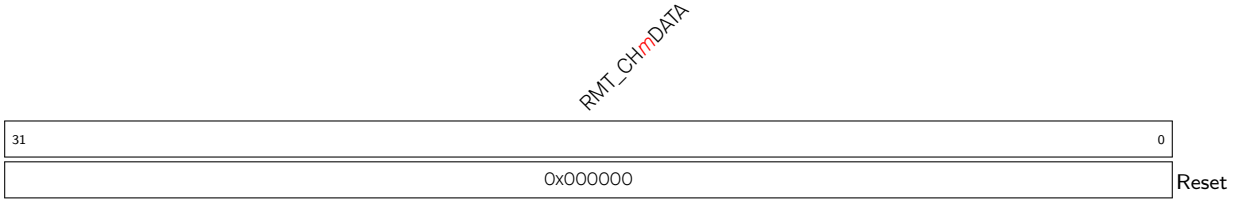
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 25.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4n*)**



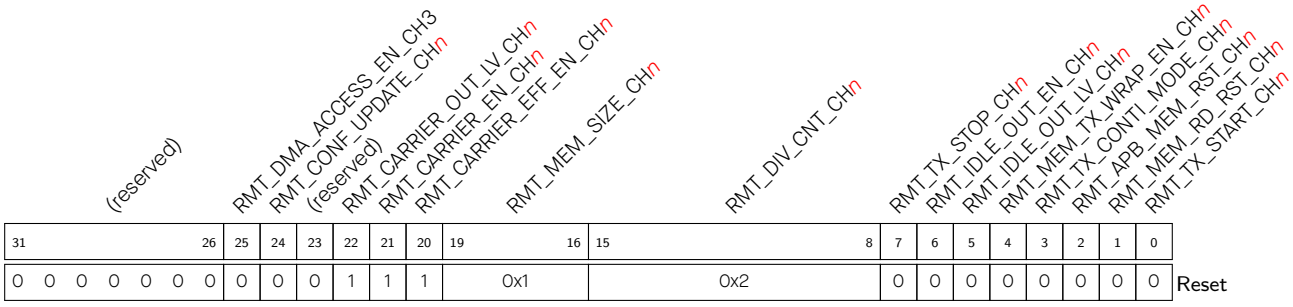
RMT_CH*n*DATA Read and write data for channel *n* via APB FIFO. (HRO)

Register 25.2. RMT_CH*m*DATA_REG (*m*: 4-7) (0x0000+0x4m*)**



RMT_CH*m*DATA Read and write data for channel *n* via APB FIFO. (HRO)

Register 25.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4*n*)



RMT_TX_START_CH*n* Configures whether to enable sending data in channel *n*.

0: No effect

1: Enable

(WT)

RMT_MEM_RD_RST_CH*n* Configures whether to reset RAM read address accessed by the transmitter for channel *n*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*n* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *n*.

0: No effect

1: Reset

(WT)

RMT_TX_CONTI_MODE_CH*n* Configures whether to enable continuous TX mode for channel *n*.

0: No Effect

1: Enable

In this mode, the transmitter starts transmission from the first data. If an end-marker is encountered, the transmitter starts transmitting data from the first data again; if no end-marker is encountered, the transmitter starts transmitting the first data again when the last data is transmitted.

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* Configures whether to enable wrap TX mode for channel *n*.

0: No effect

1: Enable

In this mode, if the TX data size is larger than the channel's RAM block size, the transmitter continues transmitting the first data to the last data in loops.

(R/W)

RMT_IDLE_OUT_LV_CH*n* Configures the level of output signal for channel *n* when the transmitter is in idle state. (R/W)

Continued on the next page...

Register 25.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4**n*)

Continued from the previous page...

RMT_IDLE_OUT_EN_CH*n* Configures whether to enable the output for channel *n* in idle state.
0: No effect
1: Enable
(R/W)

RMT_TX_STOP_CH*n* Configures whether to stop the transmitter of channel *n* sending data out.
0: No effect
1: Stop
(R/W/SC)

RMT_DIV_CNT_CH*n* Configures the divider for clock of channel *n*.
Measurement unit: rmt_sclk
(R/W)

RMT_MEM_SIZE_CH*n* Configures the maximum number of memory blocks allocated to channel *n*.
(R/W)

RMT_CARRIER_EFF_EN_CH*n* Configures whether to add carrier modulation on the output signal only at data-sending state for channel *n*.
0: Add carrier modulation on the output signal at data-sending state and idle state for channel *n*
1: Add carrier modulation on the output signal only at data-sending state for channel *n*
Only valid when RMT_CARRIER_EN_CH*n* is 1.
(R/W)

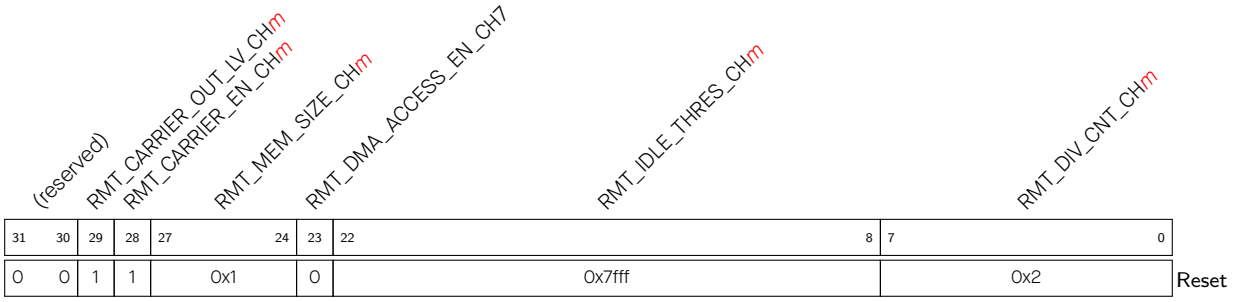
RMT_CARRIER_EN_CH*n* Configures whether to enable the carrier modulation on output signal for channel *n*.
0: Disable
1: Enable
(R/W)

RMT_CARRIER_OUT_LV_CH*n* Configures the position of carrier wave for channel *n*.
0: Add carrier wave on low level
1: Add carrier wave on high level
(R/W)

RMT_CONF_UPDATE_CH*n* Synchronization bit for channel *n*. (WT)

RMT_DMA_ACCESS_EN_CH3 Configures whether to enable the DMA access function for channel 3. This field is reserved for channel 0 ~ 2.
0: Disable
1: Enable
(R/W)

Register 25.4. RMT_CH*m*CONF0_REG (*m*: 4-7) (0x0030, 0x0038, 0x0040, 0x0048)



RMT_DIV_CNT_CH*m* Configures the clock divider of channel *m*.

Measurement unit: rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* Configures RX threshold. When no edge is detected on the input signal for continuous clock cycles longer than this field value, the receiver stops receiving data.

Measurement unit: clk_div

(R/W)

RMT_DMA_ACCESS_EN_CH7 Configures whether to enable the DMA access function for channel 7. This field is reserved for channel 4 ~ 6.

0: Disable

1: Enable

(R/W)

RMT_MEM_SIZE_CH*m* Configures the maximum number of memory blocks allocated to channel *m*.

(R/W)

RMT_CARRIER_EN_CH*m* Configures whether to enable carrier demodulation on output signal for channel *m*.

0: Disable

1: Enable

(R/W)

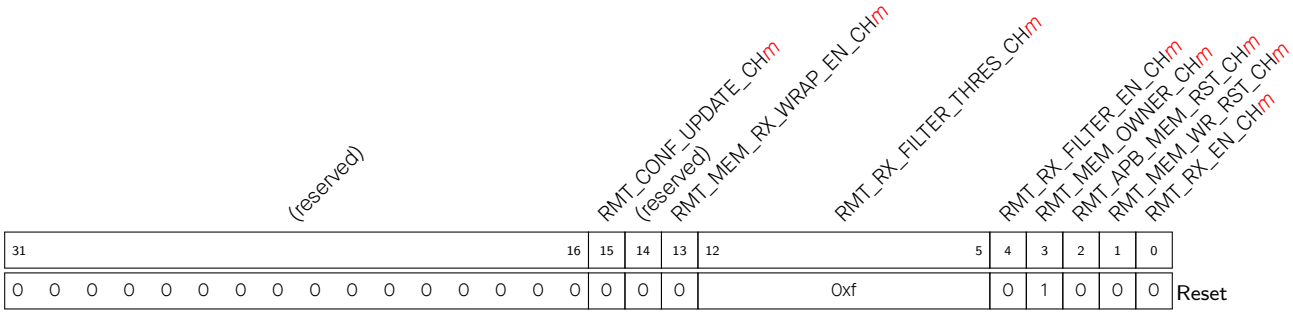
RMT_CARRIER_OUT_LV_CH*m* Configures the position of carrier wave for channel *m*.

0: Demodulate carrier wave on low level

1: Demodulate carrier wave on high level

(R/W)

Register 25.5. RMT_CH*m*CONF1_REG (*m*: 4-7) (0x0034, 0x003C, 0x0044, 0x004C)



RMT_RX_EN_CH*m* Configures whether to enable the receiver to start receiving data in channel *m*.

0: Disable

1: Enable

(R/W)

RMT_MEM_WR_RST_CH*m* Configures whether to reset RAM write address accessed by the receiver for channel *m*.

0: No effect

1: Reset

(WT)

RMT_APB_MEM_RST_CH*m* Configures whether to reset RAM W/R address accessed by APB FIFO for channel *m*.

0: No effect

1: Reset

(WT)

RMT_MEM_OWNER_CH*m* Configures the ownership of channel *m*'s RAM block.

0: APB bus is using the RAM

1: Receiver is using the RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* Configures whether to enable the receiver's filter for channel *m*.

0: Disable

1: Enable

(R/W)

RMT_RX_FILTER_THRES_CH*m* Configures whether the receiver, when receiving data, ignores the input pulse when its width is shorter than this register value in units of rmt_sclk cycles.

0: No effect

1: Reset

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* Configures whether to enable wrap RX mode for channel *m*.

0: Disable

1: Enable

In this mode, if the RX data size is larger than channel *m*'s RAM block size, the receiver stores the RX data from the first address to the last address in loops.

(R/W)

RMT_CONF_UPDATE_CH*m* Synchronization bit for channel *m*. (WT)

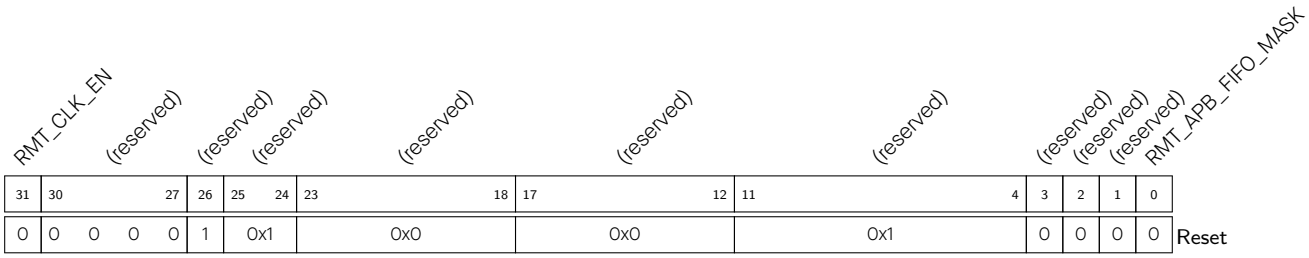
Register 25.6. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 4-7) (0x0090, 0x0094, 0x0098, 0x009C)



RMT_CARRIER_LOW_THRES_CH*m* Configures the low level period in a carrier modulation mode for channel *m*, which is equal to (RMT_CARRIER_LOW_THRES_CH*m* + 1). (R/W)

RMT_CARRIER_HIGH_THRES_CH*m* Configures the high level period in a carrier modulation mode for channel *m*, which is equal to (RMT_CARRIER_HIGH_THRES_CH*m* + 1). (R/W)

Register 25.7. RMT_SYS_CONF_REG (0x00C0)



RMT_APB_FIFO_MASK Configures the memory access mode.

0: Access memory by FIFO

1: Access memory directly

(R/W)

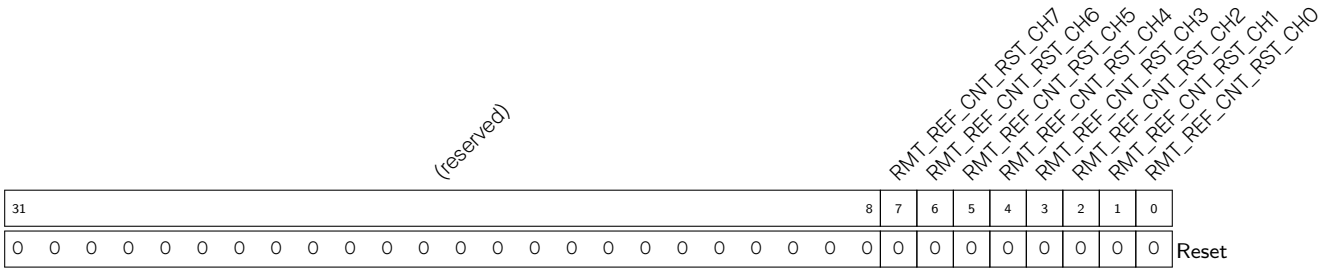
RMT_CLK_EN Configures whether to enable signal of RMT register clock gate.

0: Power down the drive clock of registers

1: Power up the drive clock of registers

(R/W)

Register 25.8. RMT_REF_CNT_RST_REG (0x00C8)



RMT_REF_CNT_RST_CH*n* (*n*: 0-3) Configures whether to reset the clock divider of channel *n*.

0: No effect

1: Reset

(WT)

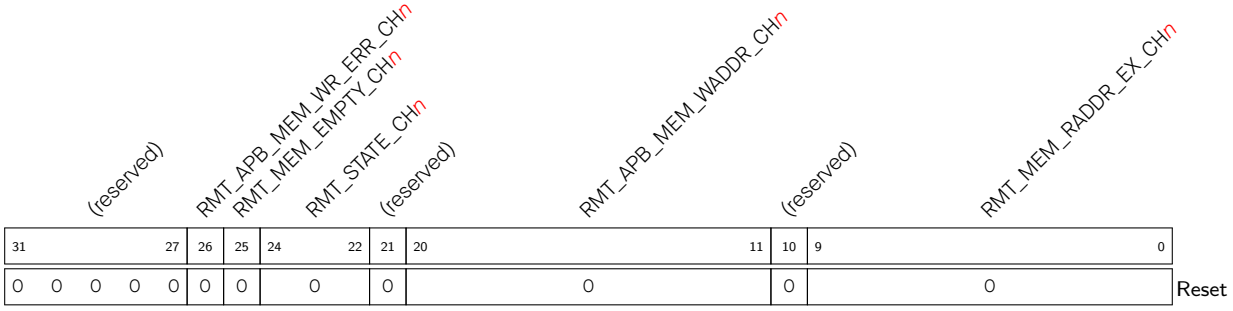
RMT_REF_CNT_RST_CH*m* (*m*: 4-7) Configures whether to reset the clock divider of channel *m*.

0: No effect

1: Reset

(WT)

Register 25.9. RMT_CH*n*STATUS_REG (*n*: 0-3) (0x0050+0x4n*)**



RMT_MEM_RADDR_EX_CH*n* Represents the memory address offset when transmitter of channel *n* is using the RAM. (RO)

RMT_APB_MEM_WADDR_CH*n* Represents the memory address offset when writes RAM over APB bus. (RO)

RMT_STATE_CH*n* Represents the FSM status of channel *n*. (RO)

RMT_MEM_EMPTY_CH*n* Represents whether the TX data size exceeds the memory size and the wrap TX mode is disabled.

0: Not exceed

1: Exceed

(RO)

RMT_APB_MEM_WR_ERR_CH*n* Represents whether the offset address exceeds memory size (overflows) when writes via APB bus.

0: Not exceed

1: Exceed

(RO)

Register 25.10. RMT_CH*m*STATUS_REG (*m*: 4-7) (0x0050+0x4m*)**



RMT_MEM_WADDR_EX_CH*m* Represents the memory address offset when receiver of channel *m* is using the RAM. (RO)

RMT_APB_MEM_RADDR_CH*m* Represents the memory address offset when reads RAM over APB bus. (RO)

RMT_STATE_CH*m* Represents the FSM status of channel *m*. (RO)

RMT_MEM_OWNER_ERR_CH*m* Represents whether the ownership of memory block is wrong.

0: The ownership of memory block is correct

1: The ownership of memory block is wrong

(RO)

RMT_MEM_FULL_CH*m* Represents whether the receiver receives more data than the memory can fit.

0: The receiver does not receive more data than the memory can fit

1: The receiver receives more data than the memory can fit

(RO)

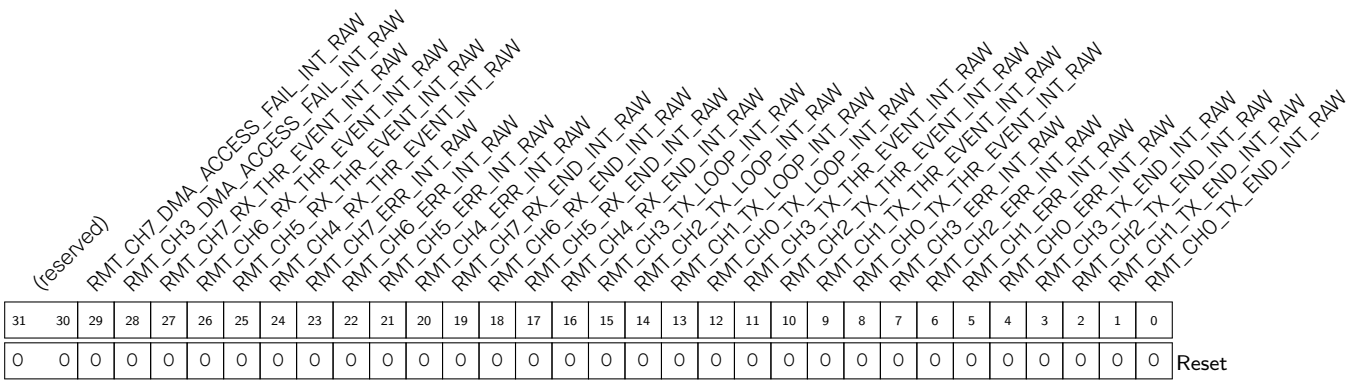
RMT_APB_MEM_RD_ERR_CH*m* Represents whether the offset address exceeds memory size (overflows) when reads RAM via APB bus.

0: Not exceed

1: Exceed

(RO)

Register 25.11. RMT_INT_RAW_REG (0x0070)



RMT_CH*n*_TX_END_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_END_INT. (R/WTC/SS)

RMT_CH*n*_ERR_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_ERR_INT. (R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. (R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW (*n*: 0-3) The raw interrupt status of RMT_CH*n*_TX_LOOP_INT. (R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_RX_END_INT. (R/WTC/SS)

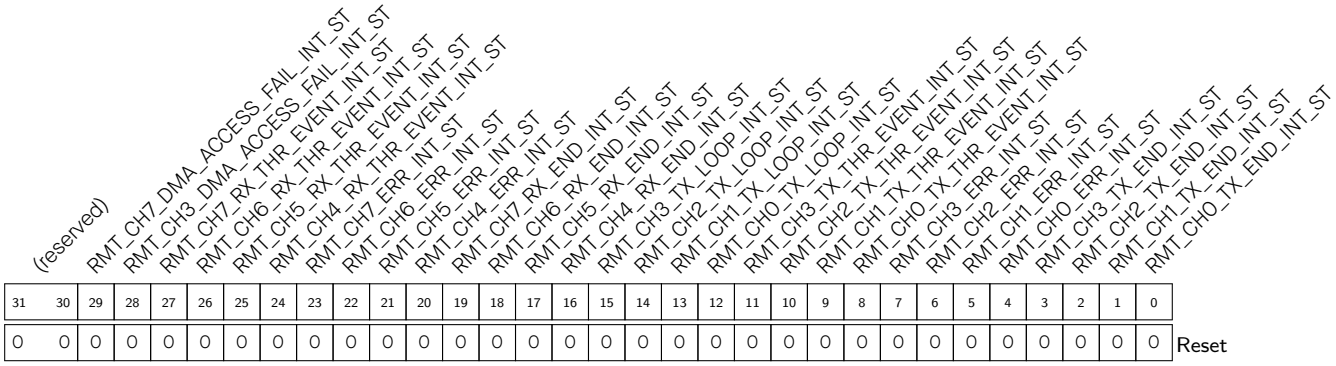
RMT_CH*m*_ERR_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_ERR_INT. (R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. (R/WTC/SS)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_RAW (*n*: 3) The raw interrupt status of RMT_CH*n*_DMA_ACCESS_FAIL_INT. (R/WTC/SS)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_RAW (*m*: 7) The raw interrupt status of RMT_CH*m*_DMA_ACCESS_FAIL_INT. (R/WTC/SS)

Register 25.12. RMT_INT_ST_REG (0x0074)



RMT_CH*n*_TX_END_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_END_INT. (RO)

RMT_CH*n*_ERR_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_ERR_INT. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_THR_EVENT_INT. (RO)

RMT_CH*n*_TX_LOOP_INT_ST (*n*: 0-3) The masked interrupt status of RMT_CH*n*_TX_LOOP_INT. (RO)

RMT_CH*m*_RX_END_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_RX_END_INT. (RO)

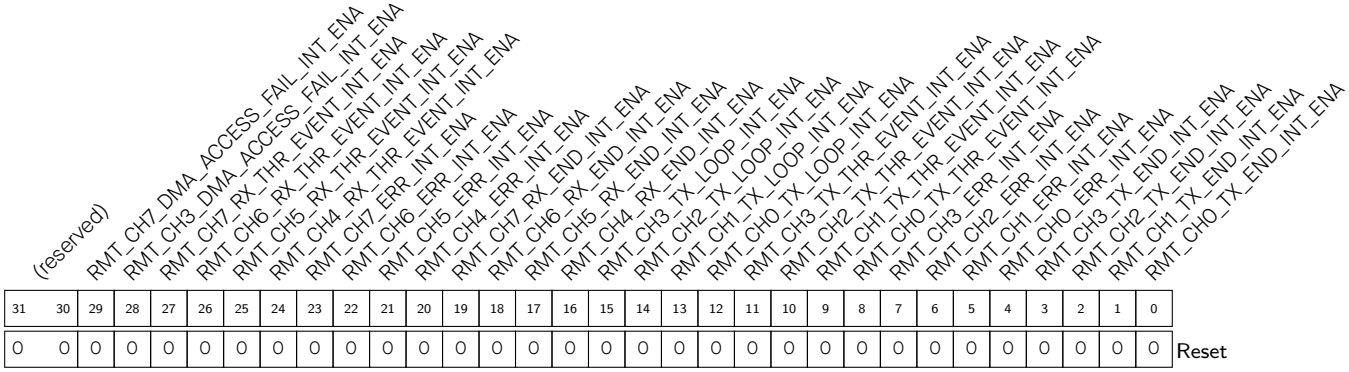
RMT_CH*m*_ERR_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_ERR_INT. (RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_RX_THR_EVENT_INT. (RO)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ST (*n*: 3) The masked interrupt status of RMT_CH*n*_DMA_ACCESS_FAIL_INT. (RO)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ST (*m*: 7) The masked interrupt status of RMT_CH*m*_DMA_ACCESS_FAIL_INT. (RO)

Register 25.13. RMT_INT_ENA_REG (0x0078)



RMT_CH*n*_TX_END_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_END_INT. (R/W)

RMT_CH*n*_ERR_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_ERR_INT. (R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_THR_EVENT_INT. (R/W)

RMT_CH*n*_TX_LOOP_INT_ENA (*n*: 0-3) Write 1 to enable RMT_CH*n*_TX_LOOP_INT. (R/W)

RMT_CH*m*_RX_END_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_RX_END_INT. (R/W)

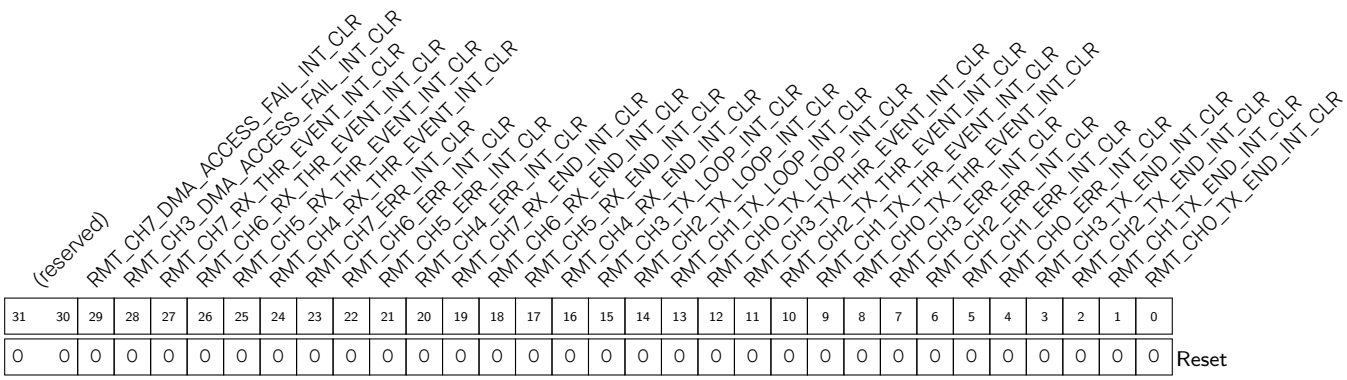
RMT_CH*m*_ERR_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_ERR_INT. (R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_RX_THR_EVENT_INT. (R/W)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ENA (*n*: 3) Write 1 to enable RMT_CH*n*_DMA_ACCESS_FAIL_INT. (R/W)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ENA (*m*: 7) Write 1 to enable RMT_CH*m*_DMA_ACCESS_FAIL_INT. (R/W)

Register 25.14. RMT_INT_CLR_REG (0x007C)



RMT_CH*n*_TX_END_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_END_INT. (WT)

RMT_CH*n*_ERR_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_ERR_INT. (WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_THR_EVENT_INT. (WT)

RMT_CH*n*_TX_LOOP_INT_CLR (*n*: 0-3) Write 1 to clear RMT_CH*n*_TX_LOOP_INT. (WT)

RMT_CH*m*_RX_END_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_RX_END_INT. (WT)

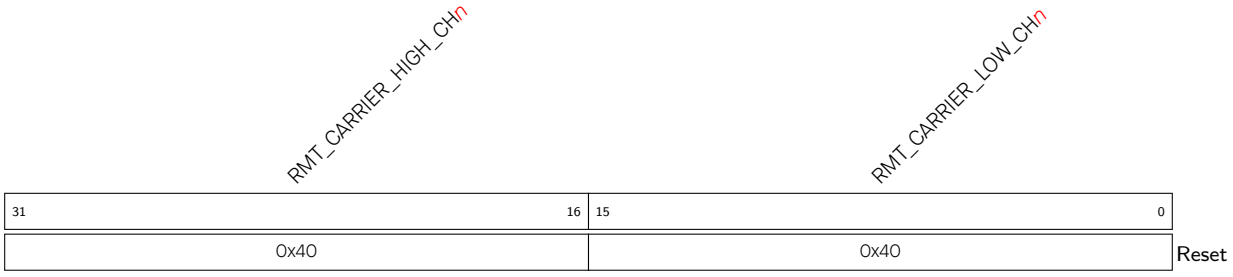
RMT_CH*m*_ERR_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_ERR_INT. (WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_RX_THR_EVENT_INT. (WT)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_CLR (*n*: 3) Write 1 to clear RMT_CH*n*_DMA_ACCESS_FAIL_INT. (WT)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_CLR (*m*: 7) Write 1 to clear RMT_CH*m*_DMA_ACCESS_FAIL_INT. (WT)

Register 25.15. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-3) (0x0080+0x4**n*)



RMT_CARRIER_LOW_CH*n* Configures carrier wave's low level clock period for channel *n*.

Measurement unit: rmt_sclk

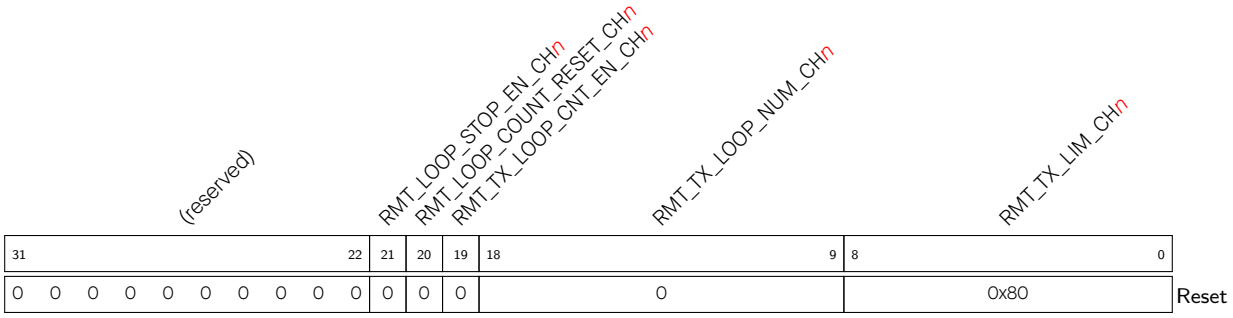
(R/W)

RMT_CARRIER_HIGH_CH*n* Configures carrier wave's high level clock period for channel *n*.

Measurement unit: rmt_sclk

(R/W)

Register 25.16. RMT_CH*n*_TX_LIM_REG (*n*: 0-3) (0x00A0+0x4n*)**



RMT_TX_LIM_CH*n* Configures the maximum entries that channel *n* can send out. (R/W)

RMT_TX_LOOP_NUM_CH*n* Configures the maximum loop count when Continuous TX mode is valid. (R/W)

RMT_TX_LOOP_CNT_EN_CH*n* Configures whether to enable loop count.

0: No effect

1: Enable

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* Configures whether to reset the loop count when continuous TX mode is enabled.

0: No effect

1: Reset

(WT)

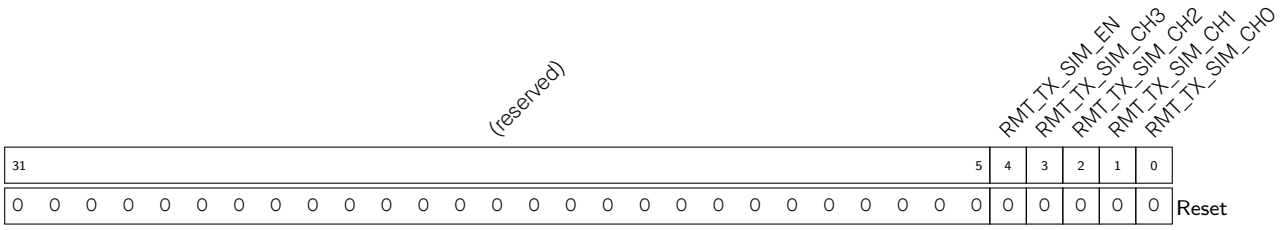
RMT_LOOP_STOP_EN_CH*n* Configures whether to enable the loop send stop function after the loop counter counts to loop number for channel *n*.

0: No effect

1: Enable

(R/W)

Register 25.17. RMT_TX_SIM_REG (0x00C4)



RMT_TX_SIM_CH*n* (*n*: 0-3) Configures whether to enable channel *n* to start sending data synchronously with other enabled channels.

0: No effect

1: Enable

(R/W)

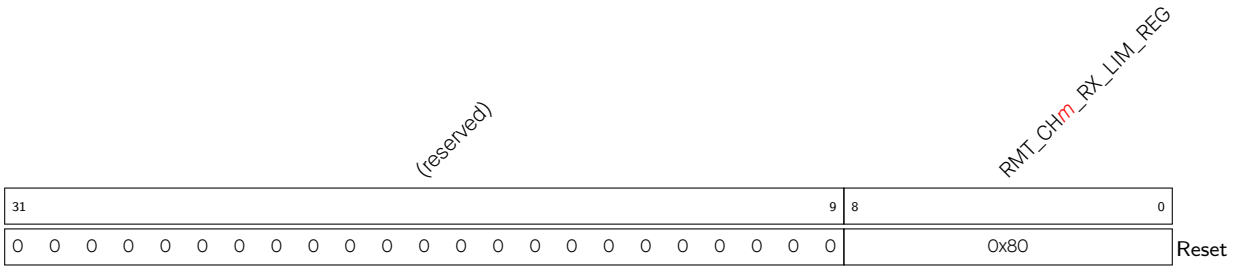
RMT_TX_SIM_EN Configures whether to enable multiple of channels to start sending data synchronously.

0: No effect

1: Enable

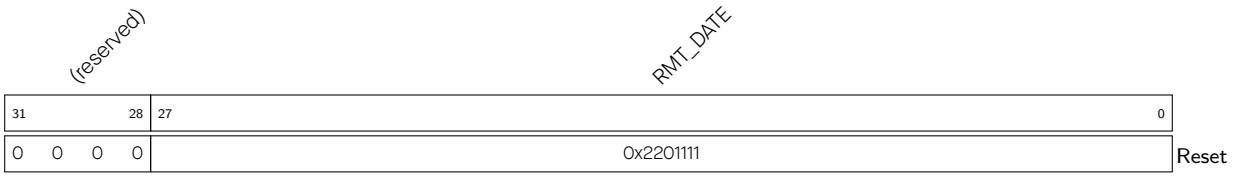
(R/W)

Register 25.18. RMT_CH*m*_RX_LIM_REG (*m*: 4-7) (0x00B0, 0x00B4, 0x00B8, 0x00BC)



RMT_CH*m*_RX_LIM_REG Configures the maximum entries that channel *m* can receive. (R/W)

Register 25.19. RMT_DATE_REG (0x00CC)



RMT_DATE Version control register. (R/W)

Chapter 26

Parallel IO Controller (PARLIO)

26.1 Introduction

ESP32-S31 contains a Parallel IO controller (PARLIO) capable of transferring data between external devices and internal memory on a parallel bus through General Direct Memory Access (GDMA). The PARLIO consists of a TX unit and an RX unit, serving as a transmitter and a receiver respectively. With the two units combined, PARLIO achieves full-duplex communication.

Due to its flexibility, PARLIO can function as a general interface to connect various peripherals. For example, with SPI as the master device and PARLIO as the slave device, a peer-to-peer transfer can be achieved. For detailed application examples, refer to Section 26.8.

26.2 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.

26.3 Features

The PARLIO module has the following main features:

- Various clock sources:
  - Including external IO clock PAD_CLK_TX/RX and internal system clock XTAL_CLK, PLL_F160M_CLK, and RC_FAST_CLK
  - Maximum IO clock frequency of 40 MHz
  - Integer and fractional clock frequency division
- 1/2/4/8/16-bit configurable data bus width
- Full-duplex communication with 16-bit data bus width
- Bit reversal when data bus width is 1/2/4-bit
- RX unit for receiving IO parallel data, which supports:
  - Output clock gating
  - RX unit input and output clock inverse
  - Various receive modes
  - Configurable GDMA SUC EOF generation
  - Configurable IO pin of external enable signal
- TX unit for sending IO parallel data, which supports:
  -
  - Chip select function
  - Output clock gating
  - TX unit input and output clock inverse
  - Configurable TX EOF generation
  - Configurable bus idle value

26.4 Architectural Overview

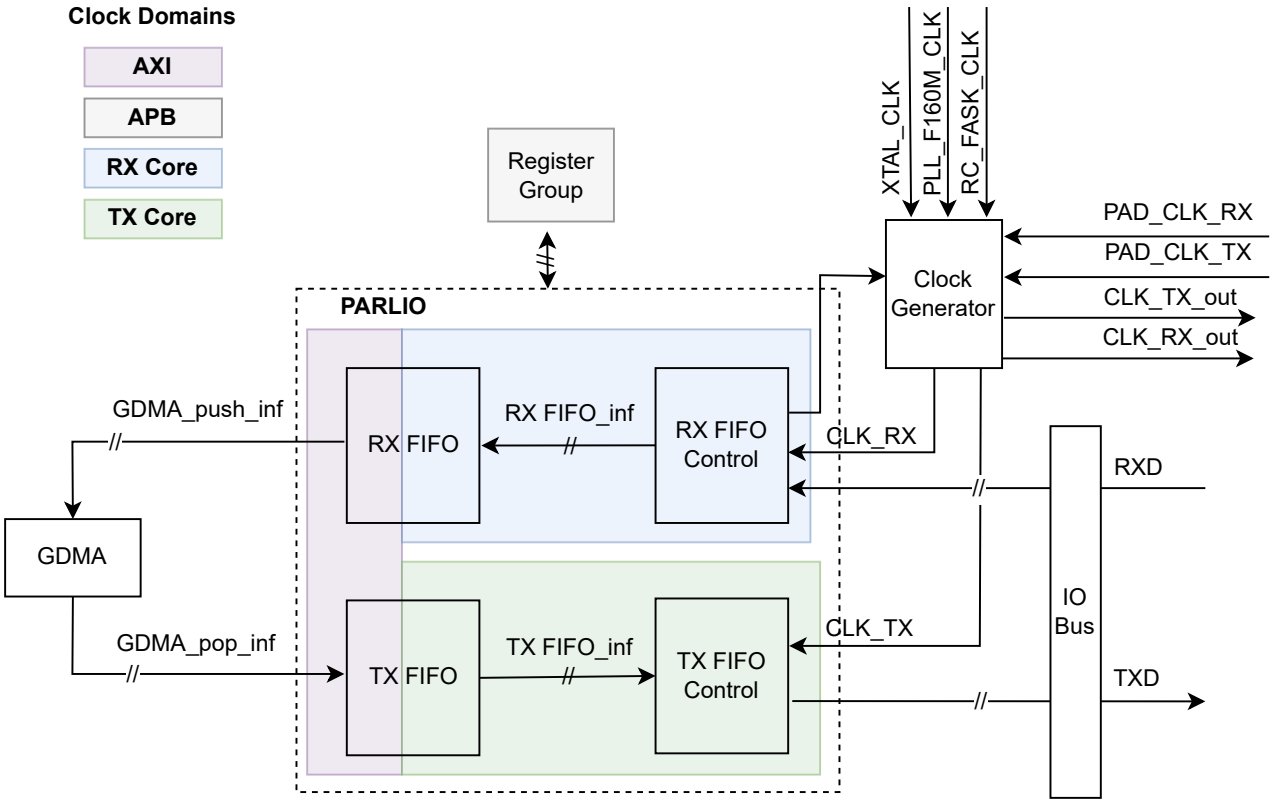


Figure 26.4-1. PARLIO Architecture

Figure 26.4-1 shows the architecture of PARLIO. In addition to the RX unit and the TX unit, a group of status configuration registers is also included.

The RX unit receives the RXD, stores it in an asynchronous FIFO after the serial-to-parallel conversion, and then stores the data to internal memory via GDMA.

The TX unit fetches data from the internal memory via GDMA, stores it in an asynchronous FIFO, and outputs the data from the IO bus via the TXD signal after the parallel-to-serial conversion.

26.5 Functional Description

26.5.1 Clock Generator

There are four input clock domains in PARLIO, namely, RX Core, TX Core, AXI, and APB, as shown in Figure 26.4-1.

The status configuration register group works in the APB clock domain.

The GDMA interface logic works in the AXI clock domain.

The RX Core clock domain has four clock sources for selection, i.e., the internal system clock source XTAL_CLK, RC_FAST_CLK, PLL_F160M_CLK, and the external clock source (PAD_CLK_RX), as shown in Figure 26.5-1. Clock sources can be selected by configuring HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL.

The clock can be divided by configuring HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM, HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR, and HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR. The clock division factor can be configured up to $(2^8 - 1)$.

The TX Core clock domain has four clock sources for selection, i.e., the internal system clock sources XTAL_CLK, RC_FAST_CLK, PLL_F160M_CLK and the external clock source (PAD_CLK_TX), as shown in Figure 26.5-1. Clock sources can be selected by configuring HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL. The clock can be divided by configuring HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM, HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR, and HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR. The clock division factor can be configured up to $(2^8 - 1)$.

The input clock of the TX and RX units can be inverted. The operating clock of the TX and RX units can also be inverted before being output to IO. The TX and RX units also support clock gating of the output clock.

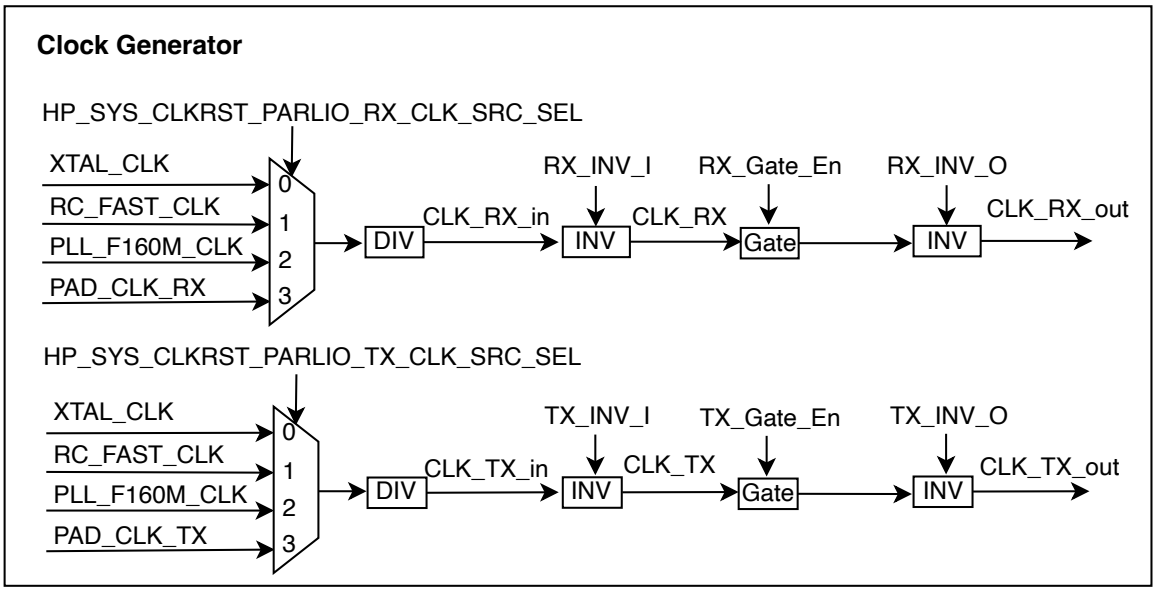


Figure 26.5-1. PARLIO Clock Generation

26.5.2 Clock & Reset Restriction

Due to the versatility of PARLIO, the PAD clocks (PAD_CLK_TX/RX) of PARLIO may come from different masters (external devices or internal clock sources). These clocks might be either free-running clock or not. If the clock is not free-running, some internal control signals of PARLIO cannot process CDC, so there are certain restrictions during the operation.

1. During the reset of the asynchronous FIFO, it takes two clock cycles to synchronize within AXI clock domain and Core clock domain. Therefore, if the reset of AXI clock domain is performed with a clock that is not free-running, the reset synchronization must be performed two clock cycles in advance. The specific operation is shown in the table below.

Table 26.5-1. Operations to Reset AXI Clock Domain with Clock Restrictions

Clock Restriction		Specific Operation
The Current Frame	The Next Frame	
Free-running clock	Not free-running clock	Users can reset the next frame transfer before switching to the clock that is not free-running. After the reset is completed, users can switch the clock.
Not free-running clock	Free-running clock	The next frame can be reset freely. Users only need to ensure that there is an interval of two clock cycles between the reset and the start of the transfer.
Not free-running clock	Not free-running clock	If the next frame transfer needs to be reset, users need to first switch to the internal free-running clock, and then switch to the actual clock after the reset is completed.

2. Due to the restrictions caused by a clock that is not free-running, PARL_IO_RX_START and PARL_IO_TX_START cannot perform CDC processing. Therefore, it is necessary to wait until PARL_IO_RX_START and PARL_IO_TX_START are stable before starting the data transfer. Otherwise, the transfer might enter a metastable state.

   Here are the specific operation steps for the RX unit:

   - Clear HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn off RX Core clock domain;
   - Write 1 to PARL_IO_RX_START;
   - Set HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn on RX Core clock domain;
   - Operate the external device to start sending data;
   - Clear HP_SYS_CLKRST_PARLIO_RX_CLK_EN to turn off RX Core clock domain;
   - Write 0 to PARL_IO_RX_START.

   Here are the specific operation steps for the TX unit:

   - Clear HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn off TX Core clock domain;
   - Write 1 to PARL_IO_TX_START;
   - Set HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn on TX Core clock domain;
   - Operate the external device to start receiving data;
   - Clear HP_SYS_CLKRST_PARLIO_TX_CLK_EN to turn off TX Core clock domain;
   - Write 0 to PARL_IO_TX_START.

3. Reset should follow the requirements below:
   - The clock reset during the chip start-up should follow the sequence below:
     (a) Reset APB clock domain;
     (b) Reset GDMA clock domain;
     (c) Reset Core clock domain.
   - Inter-frame transfer requires Core clock domain reset and async FIFO reset.

26.5.3 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 26.5-2. 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 26.5-2)	The master clock should sample TXD at the falling edge.
Not free-running clock	Negative waveform (Figure 26.5-3)	The master device should invert the original clock and convert it to the waveform as Figure 26.5-2 shows before output.

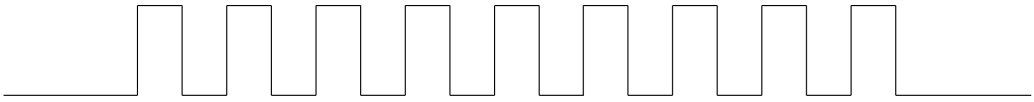


Figure 26.5-2. Positive Waveform

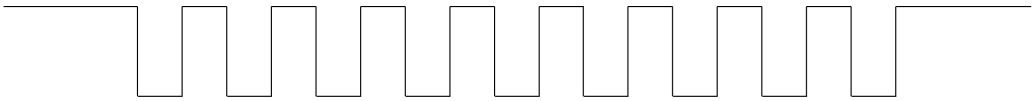


Figure 26.5-3. Negative Waveform

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 26.5-3. 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 26.5-2)	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 26.5-3)	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).

26.5.4 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.

26.5.4.1 Level Enable Mode

Figure 26.5-4 shows the Level Enable mode. In this mode, an active level on the external enable signal must be aligned with valid data. Since the external level enable signal occupies one IO pin, there are at most 15 IO pins left usable for RXD.

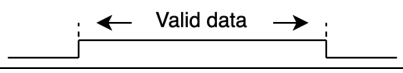
Mode	Sub-mode	Description
LEVEL_ENABLE	\	***signal level high*** 

Figure 26.5-4. Level Enable Mode for RX Unit

26.5.4.2 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 26.5-5.

Sub-modes 1 ~ 4 all contain start pulse and end pulse. The difference lies in whether start pulse and end pulse are aligned with valid data.

Sub-modes 5 ~ 6 only contain start pulse and the end of valid data is signaled by configuring PARL_IO_RX_BITLEN.

Since the external pulse enable signal occupies one IO pin, there are at most 15 IO pins left usable for RXD. However, in sub-mode 5 and sub-mode 6, as the data is considered valid after the pulse's first edge and before the pulse's last edge, the enable signal IO pin can serve as a data IO pin at the same time. Therefore, there are 16 IO pins usable for RXD in these two sub-modes.

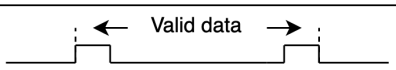
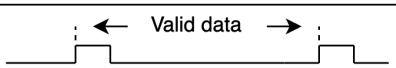
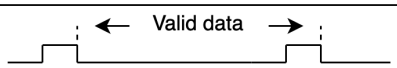
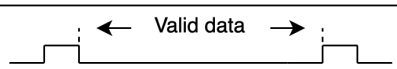
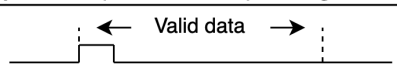
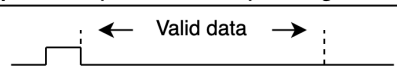
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	***pulse start(data bit include) && pulse end(data bit include)*** 
	sub-mode2	***pulse start(data bit include) && pulse end(data bit exclude)*** 
	sub-mode3	***pulse start(data bit exclude) && pulse end(data bit include)*** 
	sub-mode4	***pulse start(data bit exclude) && pulse end(data bit exclude)*** 
	sub-mode5	***pulse start(data bit include) && length end*** 
	sub-mode6	***pulse start(data bit exclude) && length end*** 

Figure 26.5-5. Sub-Modes of Pulse Enable Mode for RX Unit

26.5.4.3 Software Enable Mode

The enable signal in Software Enable mode is determined by the internal configuration register. If users switch to this mode, the receive will only be activated when both PARL_IO_RX_SW_EN and PARL_IO_RX_START are set to 1.

Since the enable signal does not occupy IO pins on the interface, there are at most 16 IO pins usable by the RXD. Due to the differences of clock domains, the enable signal cannot be aligned with valid data. Thus, the validity of data needs to be identified by the valid clock edge. In this case, the RX Core clock needs to be aligned with valid data.

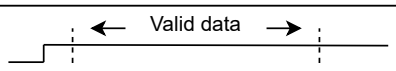
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

Figure 26.5-6. Software Enable Mode for RX Unit

26.5.5 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.

26.5.6 TX Unit EOF Generation

The TX unit generates an EOF signal and triggers the TX_EOF_INT interrupt to indicate the end of current frame transfer. The TX EOF signal can be generated by a GDMA-configured EOF signal or triggered by the internally configured bit length.

- When TX EOF is generated by the internally configured bit length, PARL_IO_TX_BITLEN must be configured. If the configured value of PARL_IO_TX_BITLEN is less than the actual transmitted data, the TX EOF signal will be triggered in advance.
- When TX EOF is generated by a GDMA-configured EOF signal, the transfer is not affected by the value of PARL_IO_TX_BITLEN, and the transferred data length of the frame is not limited. The transfer does not end until the GDMA-configured EOF signal is received.

26.5.7 RX Unit Timeout

The RX unit supports the receive timeout. If the RX FIFO has not been receiving valid data for a long time, the timeout will be triggered. In such case, a GDMA ERR EOF signal will be generated and sent to the GDMA interface to indicate the end of the receiving. Configure PARL_IO_RX_TIMEOUT_THRES to set the timeout threshold.

The timeout function is enabled by default and can be disabled by users. The upper threshold of the configurable timeout is $(2^{16} - 1)$ cycles of GDMA clock domain, and the lower threshold depends on the relative frequency relationship between GDMA clock domain and RX Core clock domain. It is recommended to set a relatively large value for PARL_IO_RX_TIMEOUT_THRES to avoid undesired GDMA ERR EOF signals.

26.5.8 Unlimited Length Data Transmission of TX Unit

By default, the end of transmission for the TX unit is determined by the configured value of PARL_IO_TX_BITLEN. When the transmitted data reaches the configured value, the system generates a GDMA SUC EOF interrupt signal, indicating the completion of the current data frame transmission. This approach limits the maximum length of a single frame transmission to the bit width defined by PARL_IO_TX_BITLEN, preventing it from achieving unlimited length data transmission.

To enable larger single-frame transmissions, set PARL_IO_TX_EOF_GEN_SEL to 1. In this mode, the hardware uses the EOF configured in the GDMA inlink list as the end-of-transmission signal for a single frame. As a result, data transmission is no longer constrained by the bit width of PARL_IO_TX_BITLEN, allowing for significantly larger data frames.

26.5.9 TX Chip Select Function

To emulate the LCD I8080 protocol, the TX unit is equipped with an output chip select (CS) signal, which is active low. Configure PARL_IO_TX_CS_START_DELAY and PARL_IO_TX_CS_STOP_DELAY to control the phase relationship between the GDMA output data and the CS signal. Note that when clock gating is enabled and the CS signal is used as the control source for clock gating, configuring these two registers not only implements the basic function, but also allows control over the phase relationship between the output clock and the CS signal.

26.5.10 Output Clock Gating of TX Unit

The TX unit supports output clock gating. The clock gating function is disabled by default, and can be enabled by software via setting PARL_IO_TX_GATING_EN.

Currently, the most significant bit (MSB) of TXD can be configured to be controlled by one of two sources: GDMA output data or the chip select signal (CS). Set PARL_IO_TX_VALID_OUTPUT_EN to 0 to select the GDMA output data as the control source. In this case, the TX unit must be configured with the maximum bit width, because when the control source is the GDMA output data, the gating signal is fixed to the MSB of TXD. The clock signal can be toggled when this bit is at a high level. Note that when selecting GDMA output data as the control source, if the peripheral is in an idle state, the clock gating will be controlled by the MSB of the user-configured bus idle value. For configuration guidelines, refer to Section 26.5.11 Bus Idle Value of TX Unit. When the gating function is enabled while using GDMA output data as the control source, there are at most 15 IO pins left usable for TXD as the gating signal occupies one IO.

Set PARL_IO_TX_VALID_OUTPUT_EN to 1 to select the chip select signal (CS) as the control source. In this case, there is no limit to the bit width.

26.5.11 Bus Idle Value of TX Unit

The TX unit is regarded as in idle state when it is not transmitting data. It supports a configurable bus idle value.

The bus idle value is 0x0 by default, and its maximum configurable value is 0xFF. Note that the configured idle value should not conflict with other enabled functions. For example, when the MSB of TXD is used as the valid signal, users should avoid configuring the MSB of the idle value as 1.

26.5.12 Data Transfer in a Single Frame

The RX unit and the TX unit transfer data in the unit of bits, i.e., a single frame transfers 1 bit of data at least.

When the RX unit generates GDMA EOF signals through bit length, the maximum length of the single-frame transmission is $(2^{19} - 1)$ bits. When the RX unit generates GDMA EOF signals through the enable signal from an external device, there is no limit to the amount of bits of the single-frame transmission.

When the TX unit generates EOF signals through bit length, the maximum length of a single frame transmission is $(2^{19} - 1)$ bits. When the TX unit generates EOF signals through GDMA-configured EOF signals, there is no limit to the amount of bits of the single-frame transmission.

Since PARLIO transfers data in the unit of bytes on the GDMA side, it will process the IO data when it is not aligned with the bytes.

- When receiving data, PARLIO will automatically pad 0 to the high bits of the data stored in memory via GDMA to make it aligned with bytes.
- When sending data, PARLIO will truncate the data retrieved from memory according to the configured value of PARL_IO_TX_BITLEN. Data exceeding the value will not be sent.

When the configured data bus width is 2/4/8/16-bit, the bit length must be configured as a multiple of the corresponding bus width.

26.5.13 Bit Reversal in One Byte

The sequence of data within one byte can be reversed when data bus width is 1/2/4-bit. Taking the RX unit as an example, when the configured bus width is 2 bits, the data needs to be packed into one byte before being written into the RX FIFO.

Presume that the original bit sequence is:

$$\{\{b_0, b_1\}, \{b_2, b_3\}, \{b_4, b_5\}, \{b_6, b_7\}\}$$

If the bit reversal is enabled, the sequence will be reordered to:

$$\{\{b_6, b_7\}, \{b_4, b_5\}, \{b_2, b_3\}, \{b_0, b_1\}\}$$

26.6 Interrupts

ESP32-S31's PARLIO module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

There are several internal interrupt sources from PARLIO that can generate the above interrupt signals. The interrupt sources from PARLIO are listed with their trigger conditions and the resulted interrupt signals in Table 26.6-1.

Table 26.6-1. PARLIO's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TX_FIFO_REMPTY_INT	TX FIFO is empty, indicating possible error in the data sent by TX	PARL_IO_TX_INTR
RX_FIFO_WOVF_INT	RX FIFO is full, indicating possible error in the data received by RX	PARL_IO_RX_INTR
TX_EOF_INT	TX finishes sending a complete frame of data	PARL_IO_TX_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 26.9 *Register Summary*.

26.7 Programming Procedures

26.7.1 Data Receiving Operation Process

This section introduces the programming procedure for receiving data through RX unit. To receive parallel data from IO pins that are connected to external devices and store the data in the internal memory, perform the following procedure. For a detailed description of the clock and reset operation restrictions in the RX unit, refer to Section 26.5.2.

1. Reset the RX unit. For specific reset scenarios and sequences, refer to Section 26.5.2.
2. Set PARL_IO_RX_FIFO_WOVF_INT_CLR and PARL_IO_RX_FIFO_WOVF_INT_ENA.
3. Select the RXD IO pins. If a PAD clock is used, the clock IO pin also needs to be configured.
4. Select the clock source and divide the clock by configuring clock registers.
5. Turn off the clock of RX Core clock domain.
6. Select the receive mode and enable functions required as described in Sections 26.3 and 26.5.
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.

26.7.2 Data Transmitting Operation Process

This section introduces the programming procedure for transmitting data through TX unit. To transmit parallel data from internal memory to the IO pins that are connected to external devices, perform the following procedure. For detailed description of the clock and reset operation restrictions for the TX unit, refer to Section 26.5.2.

1. Reset the TX unit. For specific reset scenarios and sequences, refer to Section 26.5.2.

2. Set PARL_IO_TX_FIFO_REMPTY_INT_CLR, PARL_IO_TX_EOF_INT_CLR, PARL_IO_TX_FIFO_REMPTY_INT_ENA, and PARL_IO_TX_EOF_INT_ENA in sequence.
3. Select the TXD IO pins. If a PAD clock is used, the clock IO PAD also needs to be configured.
4. Select the clock source and divide the clock by configuring clock registers.
5. Turn off the clock of TX Core clock domain.
6. Select the functions required as described in Section 26.5.
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.

26.8 Application Examples

This section introduces some PARLIO application examples and their detailed operation process. All peripherals used in the examples are from ESP series chips and can work with PARLIO to form a complete data path.

Note:
The data paths constructed in the examples may not be the optimal. For example, users can use the SPI peripherals on two identical ESP chips to complete the peer-to-peer transfer in real case instead of using PARLIO to work with SPI. However, these examples demonstrate the flexibility of the PARLIO interface to a certain extent.

26.8.1 Co-working with SPI

In this example, external SPI sends data as a master device and PARLIO RX unit receives data as a slave device, and at the same time, PARLIO TX sends data as a master device and SPI receives data as a slave device, thus achieving a peer-to-peer serial data transfer.

- Follow the operation process below to achieve SPI transmit and PARLIO receive:
  1. Configure SPI clock.
  2. Configure SPI as the master device.
  3. Configure signal pins. Connect FSPICLK to PAD_CLK_RX, FSPICS0 to RXD[15], and FSPID to RXD[0].
  4. Write the data sent into the SPI buffer and configure the bit length of the data sent.

5. Set SPI_UPDATE to update the configured register value.
6. Reset PARLIO RX unit.
7. Configure PARLIO RX unit clock.
8. Turn off the PARLIO RX Core clock domain.
9. Configure PARLIO receive mode as sub-mode 1 of LEVEL Enable mode. Configure RX unit data bus width as 1 bit. Configure PARL_IO_RX_BITLEN according to the sending length of SPI. Set PARL_IO_RX_REG_UPDATE.
10. Configure PARLIO GDMA inlink list.
11. Set PARL_IO_RX_START.
12. Turn on the PARLIO RX Core clock domain.
13. Set SPI_USR to start transmitting data of SPI.
14. Poll GDMA SUC EOF interrupt.
15. Clear PARL_IO_RX_START.

- Follow the operation process below to achieve PARLIO transmit and SPI receive:

1. Configure SPI clock.
2. Configure SPI as the slave device.
3. Configure signal pins. Connect FSPICLK to PAD_CLK_TX, FSPICS0 to TXD[15], and FSPID to TXD[0].
4. Set SPI_RD_BIT_ORDER to invert the bit order.
5. Set SPI_UPDATE to update the configured register value.
6. Reset PARLIO TX unit.
7. Set PARL_IO_TX_EOF_INT_CLR and PARL_IO_TX_EOF_INT_ENA.
8. Configure PARLIO TX unit clock.
9. Turn off the clock of TX Core clock domain.
10. Configure data bus width as 1 bit. Write 1 to PARL_IO_TX_VALID_OUTPUT_EN. Configure PARL_IO_TX_BITLEN.
11. Configure GDMA outlink list.
12. Poll PARL_IO_TX_READY.
13. Write 1 to PARL_IO_TX_START.
14. Turn on the clock of TX Core clock domain.
15. Start data transfer.
16. Poll PARL_IO_TX_EOF_INT_ST.
17. Set PARL_IO_TX_EOF_INT_CLR.
18. Turn off the clock of TX Core clock domain.
19. Clear PARL_IO_TX_START.

26.8.2 Co-working with I2S

In this example, external I2S sends data as a master device and PARLIO RX unit receives data as a slave device. PARLIO supports the transmission of the I2S TDM MSB alignment standard and the TDM PCM standard. When the I2S transfer protocol is the TDM MSB alignment standard, it is required to configure the receive mode of PARLIO as Level Enable mode. When the I2S transfer protocol is the TDM PCM standard, it is required to configure the receive mode of PARLIO as the sub-mode 4 of Pulse Enable mode and set PARL_IO_RX_EXT_EN_INV.

This section takes the TDM PCM alignment standard as an example. The specific operation process is as follows:

1. Configure I2S clock.
2. Configure signal pins. Connect I2SO_BCK_out to PAD_CLK_RX, I2SO_WS_out to RXD[15], and I2SO_Data_out to RXD[0].
3. Configure I2S as the master device.
4. Configure the I2S TX data mode and channel mode required. Set I2S_TX_UPDATE.
5. Reset I2S TX unit and TX FIFO.
6. Enable I2S_TX_DONE_INT.
7. Configure I2S GDMA outlink list.
8. Set I2S_TX_STOP_EN.
9. Reset PARLIO RX unit.
10. Configure PARLIO RX unit clock.
11. Turn off PARLIO RX Core clock domain.
12. Configure PARLIO receive mode as sub-mode 4 of Pulse Enable mode and set PARL_IO_RX_EXT_EN_INV to configure the RX unit data bus width as 1 bit. Configure PARL_IO_TX_BITLEN according to the length of the data sent by I2S. Set PARL_IO_RX_REG_UPDATE.
13. Configure PARLIO GDMA inlink list.
14. Set PARL_IO_RX_START.
15. Turn on PARLIO RX Core clock domain.
16. Set I2S_TX_START to start transmitting data.
17. Poll I2S_TX_DONE_INT.
18. Poll GDMA SUC EOF interrupt.
19. Clear I2S_TX_START.
20. Clear PARL_IO_RX_START.

26.8.3 Co-working with LCD

In this example, PARLIO TX unit sends data as a master device and external LCD controller receives data as a slave device. The I8080/MOTO6800 format is used. 8-bit parallel data is transferred between devices. The

specific operation process is as follows:

1. Configure signal pins. Connect CLK_TX_out to LCD pixel clock PCLK, TXD[7:0] to LCD data input pin, TXD[8] to LCD CS pin, TXD[9] to LCD CD pin.
2. Reset PARLIO TX unit.
3. Set PARL_IO_TX_FIFO_REMPTY_INT_CLR, PARL_IO_TX_EOF_INT_CLR, PARL_IO_TX_FIFO_REMPTY_INT_ENA, and PARL_IO_TX_EOF_INT_ENA in sequence.
4. Configure PARLIO TX unit clock.
5. Turn off the clock of TX Core clock domain.
6. Configure data bus width as 16 bit. Configure PARL_IO_TX_BITLEN.
7. Configure GDMA outlink list. Note that the data sent in the linked list should conform to I8080/MOTO6800 format. The lower eight bits are valid parallel data. The 9th and 10th bits are respectively CS, CD. The MSB is the constant 1. The remaining bits are arbitrary values.
8. Poll PARL_IO_TX_READY.
9. Write 1 to PARL_IO_TX_START.
10. Turn on the clock of TX Core clock domain.
11. Start data transfer.
12. Poll PARL_IO_TX_EOF_INT_ST.
13. Set PARL_IO_TX_EOF_INT_CLR.
14. Turn off the clock of TX Core clock domain.
15. Clear PARL_IO_TX_START.

26.9 Register Summary

The addresses in this section are relative to Parallel IO Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

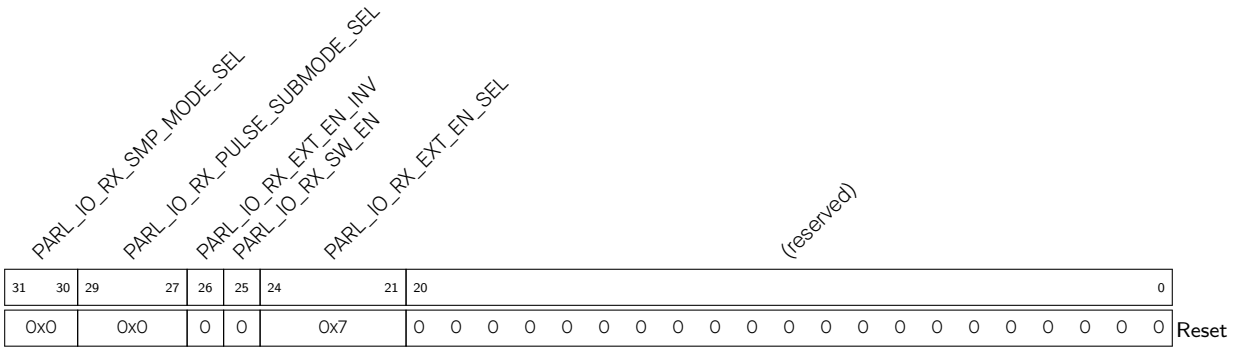
Name	Description	Address	Access
PARLIO RX Configuration Registers			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX sampling mode configuration register	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX data configuration register	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX general configuration register	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX start configuration register	0x000C	R/W
PARLIO TX Configuration Registers			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX data configuration register	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX start configuration register	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX general configuration register	0x0018	R/W
PARLIO Configuration and Status Registers			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO configuration register	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO register update configuration register	0x0020	WT
PARL_IO_ST_REG	PARLIO module status register	0x0024	RO
PARLIO Interrupt Configuration and Status Registers			
PARL_IO_INT_ENA_REG	PARLIO interrupt enable signal configuration register	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO interrupt raw signal status register	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO interrupt signal status register	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO interrupt clear signal configuration register	0x0034	WT
PARLIO RX/TX Status Registers			
PARL_IO_RX_ST0_REG	PARLIO RX status register 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX status register 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX status register 0	0x0040	RO
PARLIO Clock Configuration Registers			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX clock configuration register	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX clock configuration register	0x0048	R/W
PARL_IO_CLK_REG	PARLIO clock configuration register	0x0120	R/W
PARLIO TX CS Configuration Register			
PARL_IO_TX_CS_CFG_REG	PARLIO TX CS configuration register	0x004C	R/W
PARLIO Version Register			
PARL_IO_VERSION_REG	Version control register	0x03FC	R/W

26.10 Registers

The addresses in this section are relative to Parallel IO Controller base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 26.1. 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.

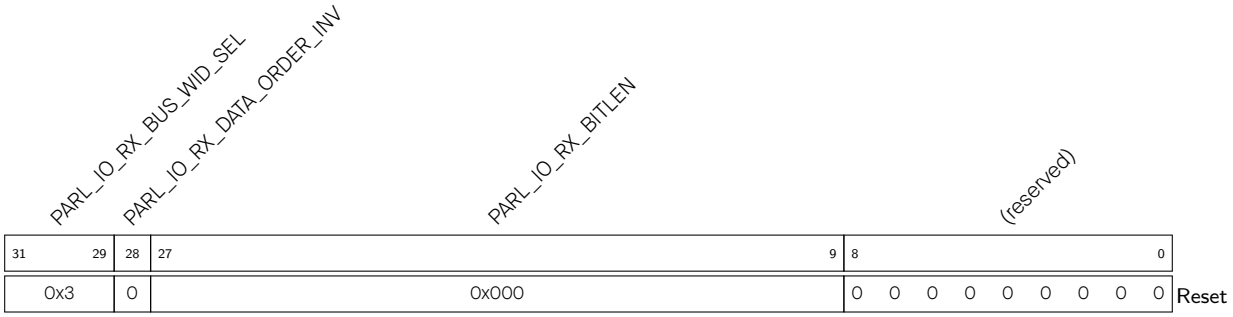
0: External Level Enable mode

1: External Pulse Enable mode

2: Internal Software Enable mode

(R/W)

Register 26.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN Configures the expected bit number of RXD. (R/W)

PARL_IO_RX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from RX FIFO to GDMA. (R/W)

PARL_IO_RX_BUS_WID_SEL Configures the RXD bus width.

0: 1 bit

1: 2 bits

2: 4 bits

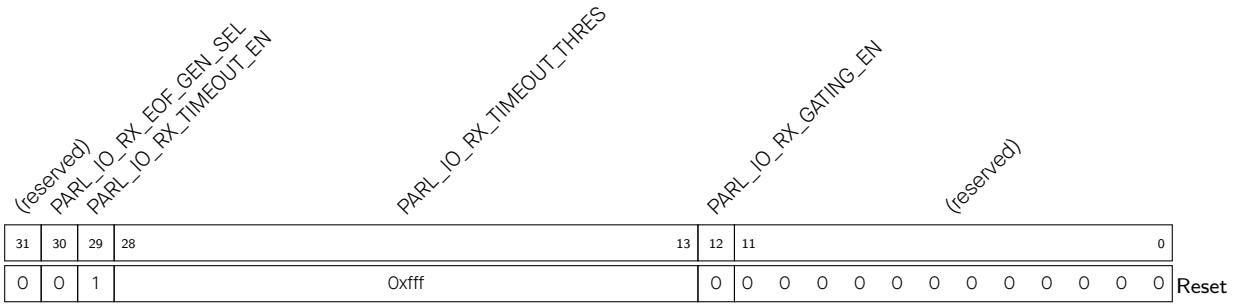
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 26.3. 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)

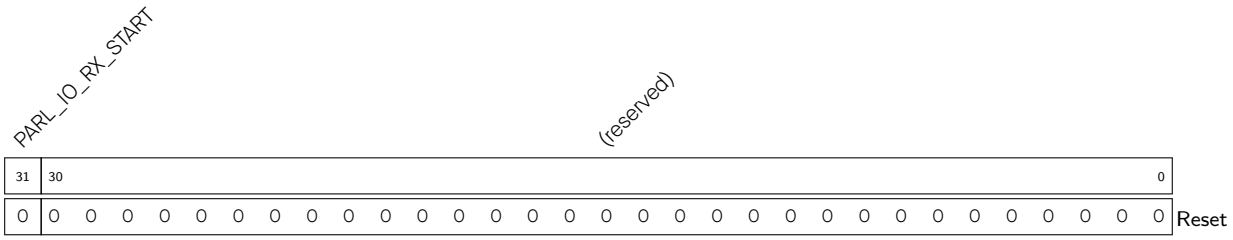
PARL_IO_RX_EOF_GEN_SEL Configures the generation mechanism of GDMA SUC EOF.

0: Generate GDMA SUC EOF by the configured data bit length

1: Generate GDMA SUC EOF by the external enable signal

(R/W)

Register 26.4. PARL_IO_RX_START_CFG_REG (0x000C)



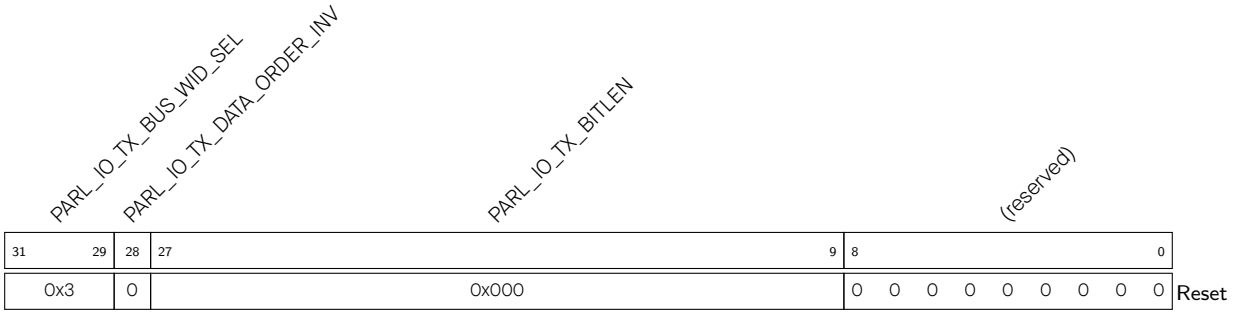
PARL_IO_RX_START Configures whether to start RX data sampling.

0: No effect

1: Start

(R/W)

Register 26.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN Configures the expected bit number of TXD. (R/W)

PARL_IO_TX_DATA_ORDER_INV Configures whether to invert the bit order of one byte sent from TX FIFO to IO data.

0: No effect

1: Invert

(R/W)

PARL_IO_TX_BUS_WID_SEL Configures the TXD bus width.

0: 1 bit

1: 2 bits

2: 4 bits

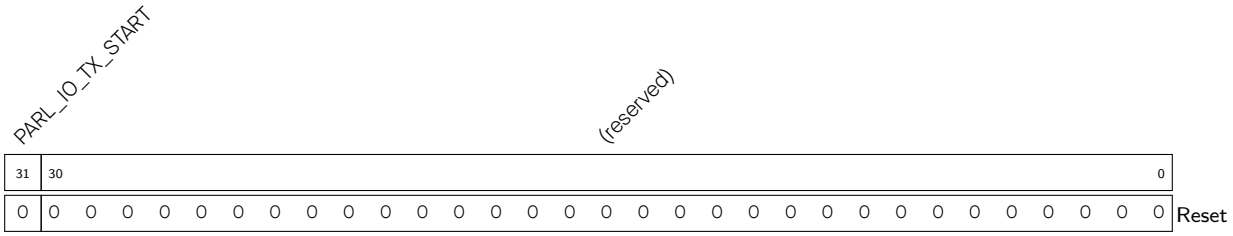
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 26.6. PARL_IO_TX_START_CFG_REG (0x0014)



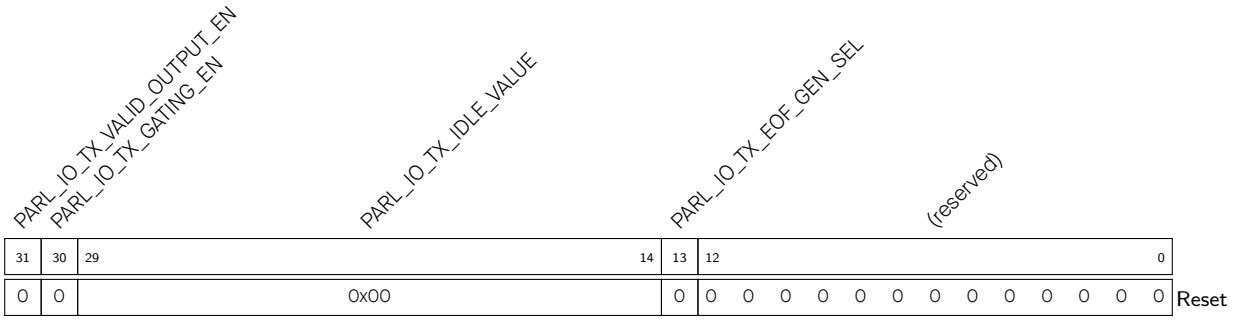
PARL_IO_TX_START Configures whether to start TX data transmission.

0: No effect

1: Start

(R/W)

Register 26.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL Configures the generation mechanism of TX EOF.

0: Generate TX EOF by the configured data bit length

1: Generate TX EOF by the GDMA EOF signal.

(R/W)

PARL_IO_TX_IDLE_VALUE Configures the data value on TX bus in idle state. (R/W)

PARL_IO_TX_GATING_EN Configures whether to enable the clock gating of the TX output clock.

0: Disable

1: Enable

(R/W)

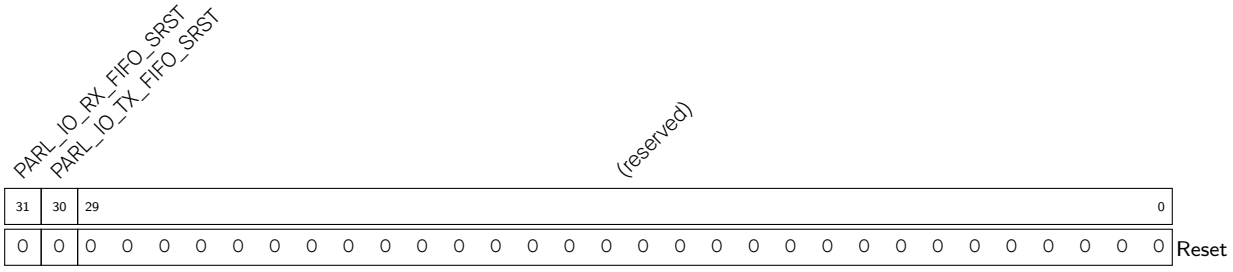
PARL_IO_TX_VALID_OUTPUT_EN Configures whether to select DMA output data or the chip select signal (CS) as the control source of TXD MSB.

0: Select the DMA output data as the control source

1: Select the chip select signal (CS) as the control source

(R/W)

Register 26.8. 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)

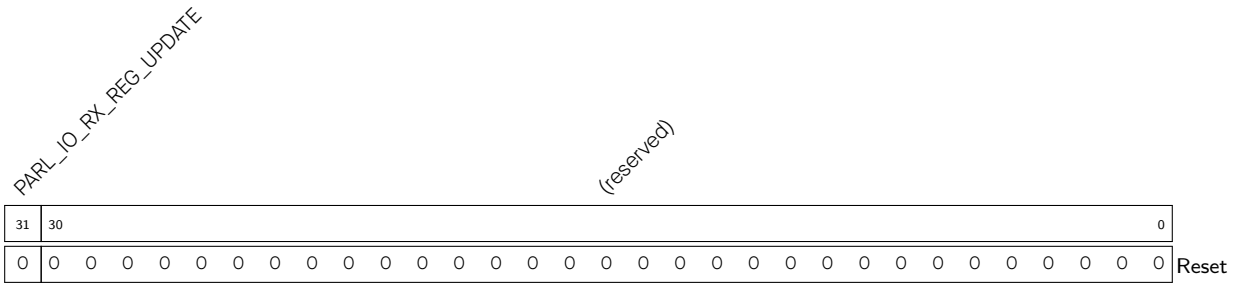
PARL_IO_RX_FIFO_SRST Configures whether to reset async FIFO in the RX unit.

0: No effect

1: Reset

(R/W)

Register 26.9. PARL_IO_REG_UPDATE_REG (0x0020)



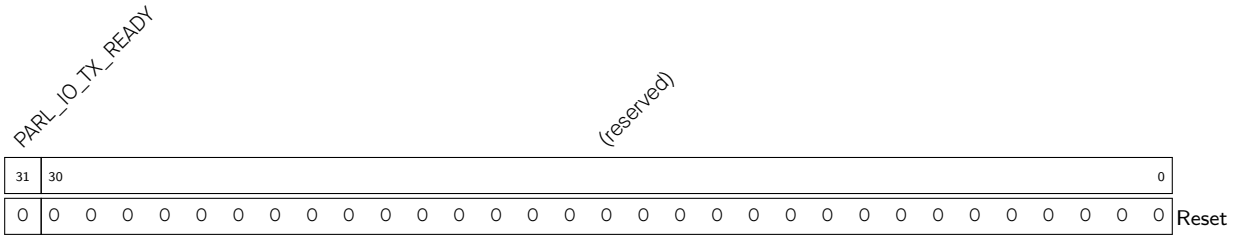
PARL_IO_RX_REG_UPDATE Configures whether to update RX register configuration.

0: No effect

1: Update

(WT)

Register 26.10. PARL_IO_ST_REG (0x0024)



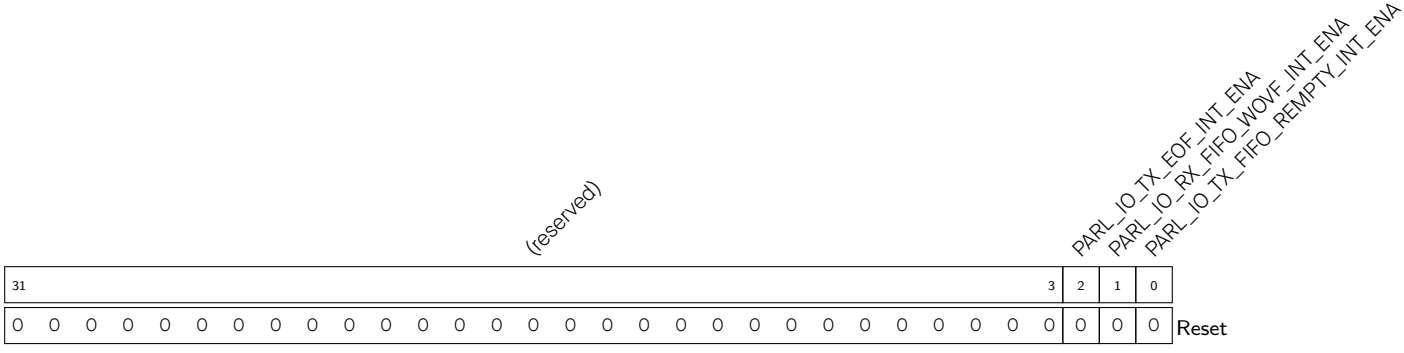
PARL_IO_TX_READY Represents whether TX is ready to transmit data.

0: Not ready

1: Ready

(RO)

Register 26.11. PARL_IO_INT_ENA_REG (0x0028)

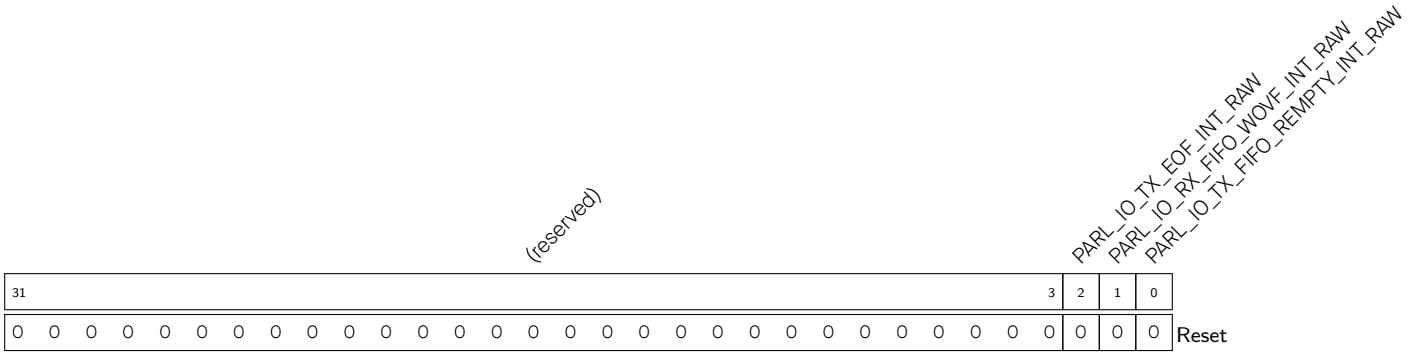


PARL_IO_TX_FIFO_REMPTY_INT_ENA Write 1 to enable TX_FIFO_REMPTY_INT. (R/W)

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 26.12. PARL_IO_INT_RAW_REG (0x002C)

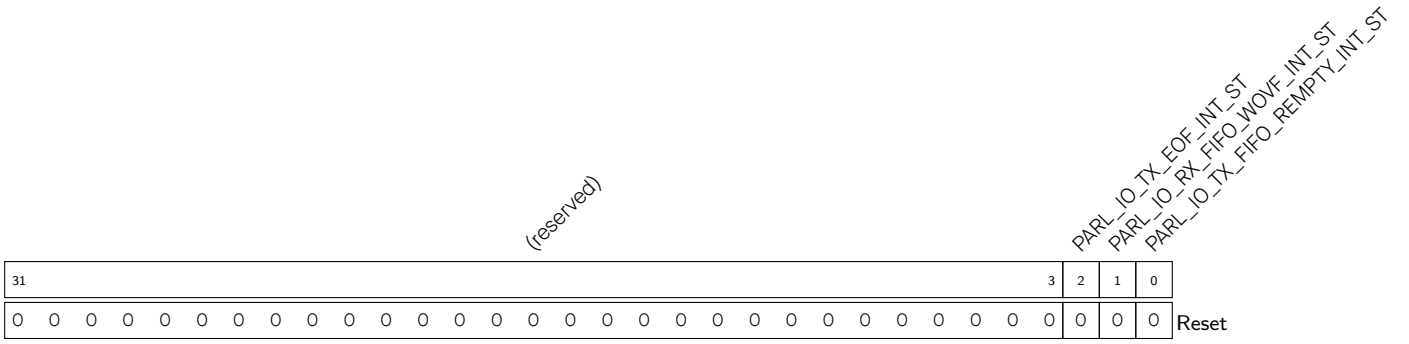


PARL_IO_TX_FIFO_REMPTY_INT_RAW The raw interrupt status of TX_FIFO_REMPTY_INT. (R/WTC/SS)

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 26.13. PARL_IO_INT_ST_REG (0x0030)

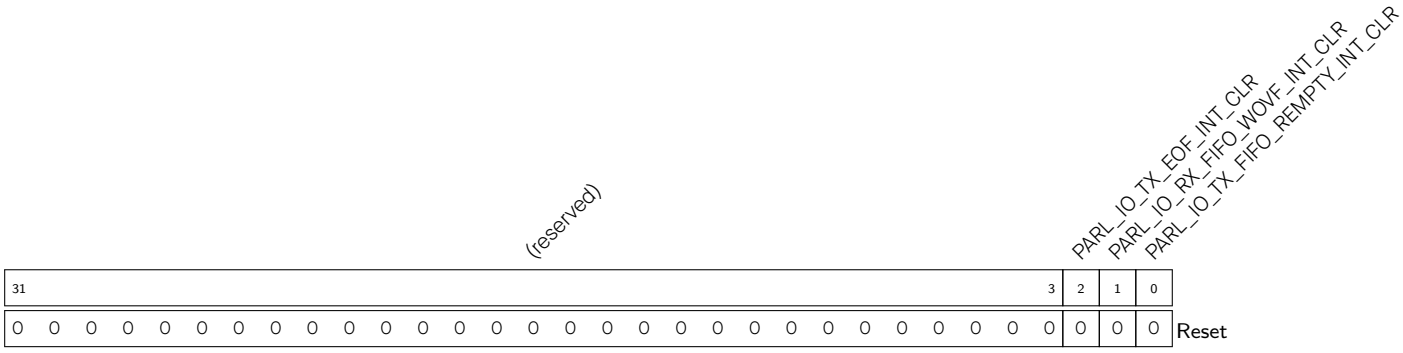


PARL_IO_TX_FIFO_REMPTY_INT_ST The masked interrupt status of TX_FIFO_REMPTY_INT. (RO)

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 26.14. PARL_IO_INT_CLR_REG (0x0034)

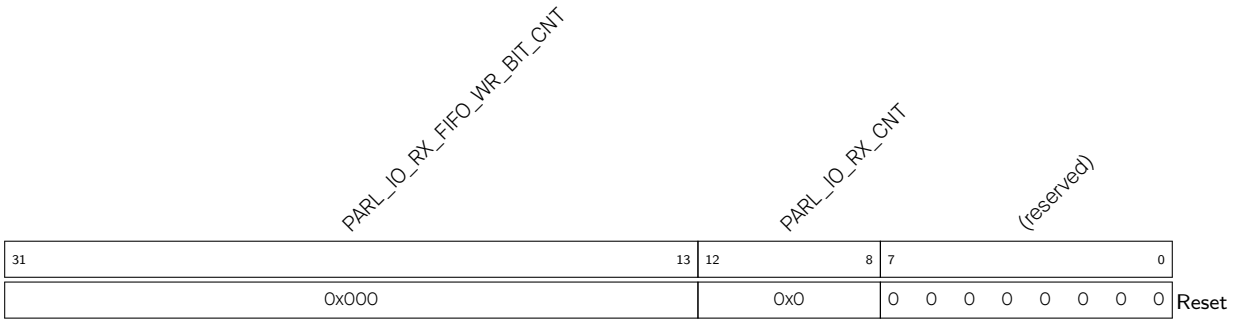


PARL_IO_TX_FIFO_REMPTY_INT_CLR Write 1 to clear TX_FIFO_REMPTY_INT. (WT)

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)

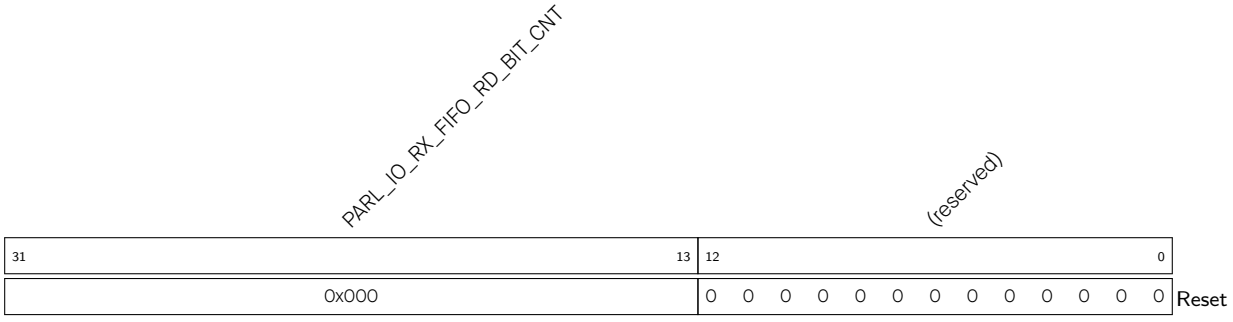
Register 26.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT Represents the clock cycle number of reading the RX FIFO. (RO)

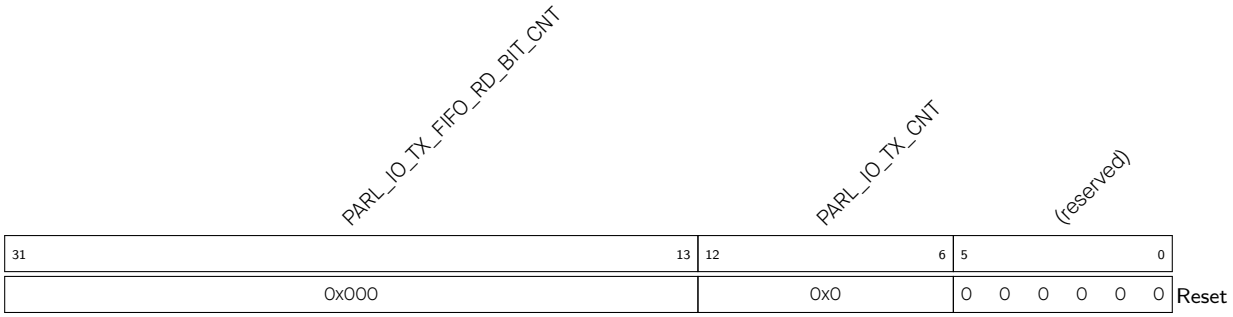
PARL_IO_RX_FIFO_WR_BIT_CNT Represents the bit number currently written into the RX FIFO. (RO)

Register 26.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT Represents the bit number currently read from the RX FIFO. (RO)

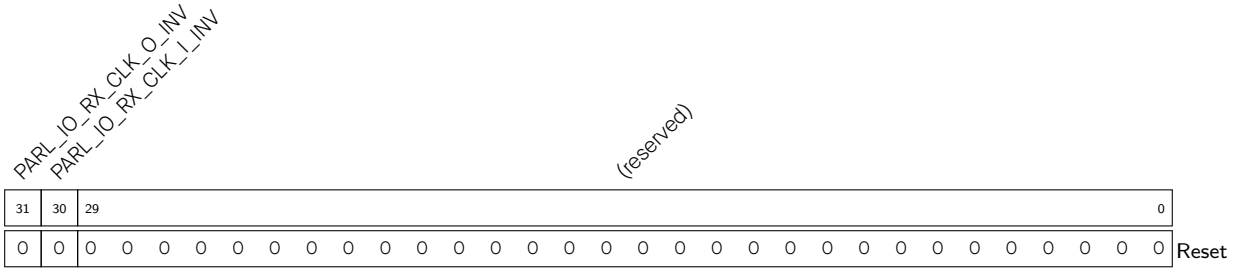
Register 26.17. PARL_IO_TX_ST0_REG (0x0040)



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 26.18. 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)

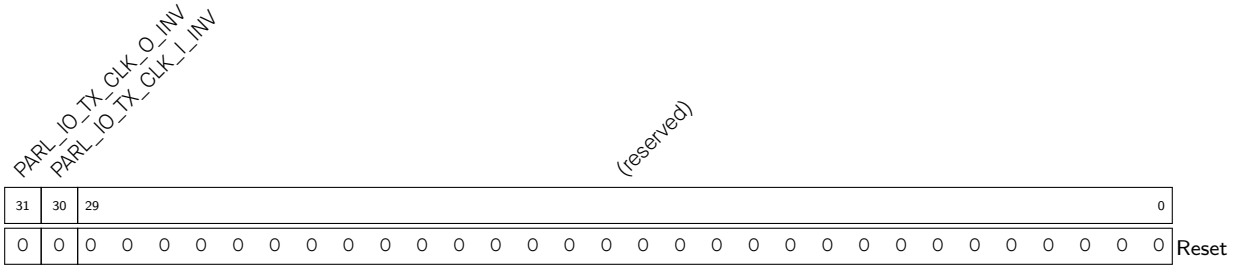
PARL_IO_RX_CLK_O_INV Configures whether to invert the RX output Core clock.

0: No effect

1: Invert

(R/W)

Register 26.19. 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)

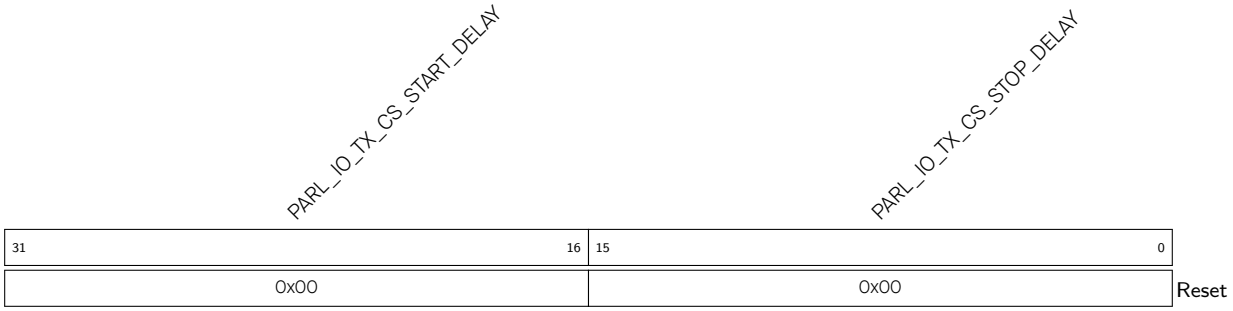
PARL_IO_TX_CLK_O_INV Configures whether to invert the TX output Core clock.

0: No effect

1: Invert

(R/W)

Register 26.20. PARL_IO_TX_CS_CFG_REG (0x004C)



PARL_IO_TX_CS_STOP_DELAY Configures the delay between the end of TX data transmission and the rising edge of TX_CS_O.

Unit: TX Core clock cycle

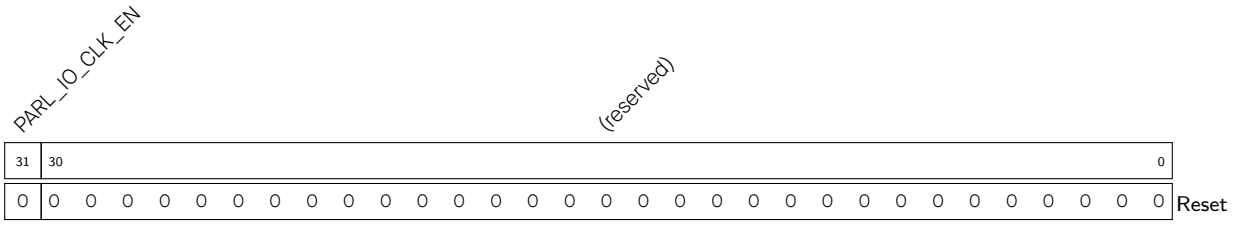
(R/W)

PARL_IO_TX_CS_START_DELAY Configures the delay between the falling edge of TX_CS_O and the start of TX data transmission.

Unit: TX Core clock cycle

(R/W)

Register 26.21. PARL_IO_CLK_REG (0x0120)



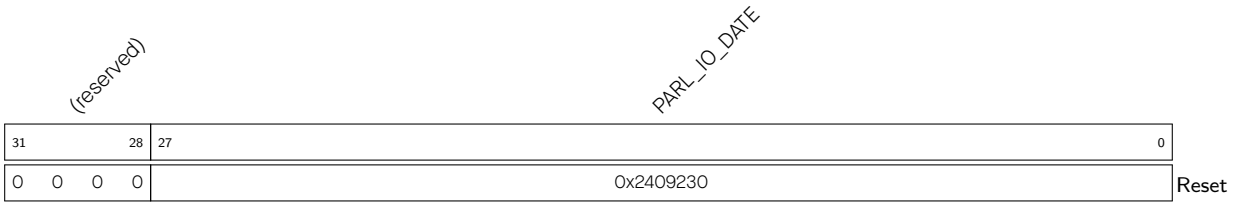
PARL_IO_CLK_EN Register clock gating.

0: Enable the register clock only when reading or writing registers

1: Always enable the register clock

(R/W)

Register 26.22. 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, voltage comparison, on-chip sensors, and features such as temperature and touch sensing, demonstrating the system's capabilities in handling analog signals.

Chapter 27

Touch Sensor (TOUCH)

ESP32-S31 has 14 capacitive touch sensor units, each of which can be connected to an external touch panel via a touch pin (GPIO pin) to constitute a touch sensor system. The ESP32-S31 touch sensor system is mainly used in human-computer interactions to detect finger touch or proximity. It also supports frequency hopping sampling and features moisture resistance and waterproof design.

27.1 Terminology

To better illustrate the functions of capacitive touch sensors, the following terms are used in this section.

Touch Sensor	Touch-related internal sensing circuitry.
Touch Pin	Pins with touch sensing feature.
Touch Panel	The external device connected to touch sensor via channel (touch pin) to detect finger touch.
Touch Sensor System	Capacitive touch sensing system, consisting of touch sensor, touch pin, traces, and touch panel.

Note:
In subsequent description, "touch panel is sampled/scanned/measured" indicates the same action to touch pin, i.e. "touch pin is sampled/scanned/measured".

27.2 Feature List

The ESP32-S31 touch sensor has the following features:

- Detection of 14 capacitive touch pins
- Sampling triggered by software or dedicated hardware timer
- Two sampling methods:
  - Pulses from the touch pins used as clock signals to count the sampling period
  - Pulses from the touch pins used as digital signals; sample the rising edge of the digital signal with the system clock to count the sampling period
- Scan mode, supporting sequential sampling of multiple touch pins by configuring the Touch FSM.
- Timeout mechanism to monitor channel abnormality
- Frequency hopping to increase the anti-interference of detection
- Proximity sensing mode with up to three configurable channels

- Configuration of individual touch sensors to operate normally in sleep mode
- Wake-up by touch sensor
- Moisture resistance
- Waterproof design

27.3 Architectural Overview

27.3.1 Touch Panel

A touch panel consists of the following components:

- Electrode: Changes in capacitance when touched by a finger.
- Substrate: The base material on which the protective cover, electrode, and the electrode's connector to the channel are built.
- Protective cover: To physically separate the other components from the external environment.

When developing applications using the touch sensing feature, users should decide on the placement, the material, or the arrangement of the touch panels during mechanical design. For more information about the design guidelines, please refer to Touch Sensor Application Note. Touch panel can be connected to ESP32-S31 touch sensor via touch pin (channel). See Figure 27.3-1.

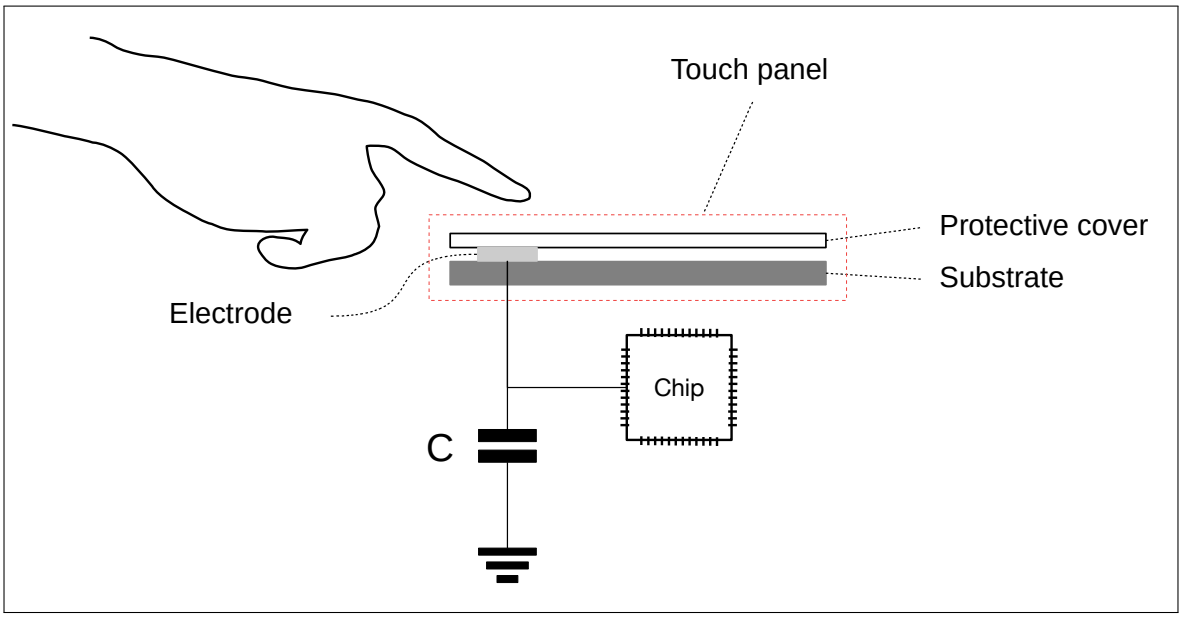


Figure 27.3-1. Touch Sensor System

When users touch the protective cover, the capacitance of the electrode increases. If the electrode is connected to one of the ESP32-S31 touch sensors via a channel, the touch sensor will be able to detect the electrode's change in capacitance. If the change of capacitance exceeds the configurable threshold, the touch sensor will trigger an interrupt.

27.3.2 Capacitive Touch Pin

ESP32-S31 provides 14 capacitive touch pins labeled as T0 ~ T13, each connected to the chip's pin to monitor finger touch from the external environment. The mapping between the touch sensors and the chip pins is shown in Table 27.3-1.

Table 27.3-1. ESP32-S31 Capacitive Touch Pins

Capacitive Touch Pin	Connected GPIO Pin
T0	GPIO6
T1	GPIO7
T2	GPIO8
T3	GPIO9
T4	GPIO10
T5	GPIO11
T6	GPIO12
T7	GPIO13
T8	GPIO14
T9	GPIO15
T10	GPIO16
T11	GPIO17
T12	GPIO18
T13	GPIO19

27.3.3 Touch Sensor

The touch sensor charges and discharges the touch pin with a fixed current source, and each charge and discharge generates a pulse signal, the TOUCH_OUT signal. The touch sensor detects the capacitance change of the touch pin. If the touch pin is touched or approached by a finger, the touch pin's capacitance increases and the charging and discharging time is extended. By measuring the time required for charging and discharging the touch pin a fixed number of times, it can be inferred whether the touch pin has been touched or not.

Figure 27.3-2 illustrates the internal structure of the touch sensors. Each of the touch sensors has a set of input and output signals, some of which are connected to the Touch FSM (see Section 27.4.1). Table 27.3-2 shows the output signals of the touch sensor and their functions.

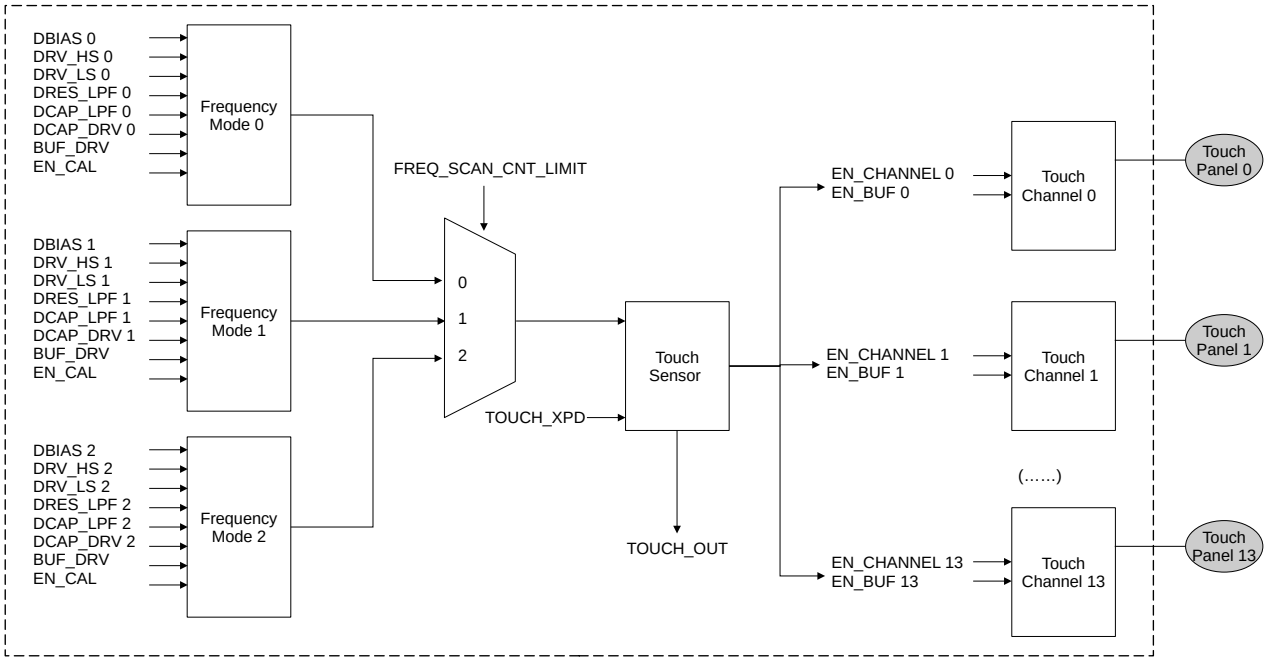


Figure 27.3-2. Touch Sensor Internal Structure

Table 27.3-2. Capacitive Touch Sensor Output Signal

Signal	Source	Function
TOUCH_XPD	Touch FSM	Controls the power-up and power-down of the touch sensor
TOUCH_START	Touch FSM	Controls the touch sensor to initiate a measurement. The measurement ends when the Touch FSM pulls down the TOUCH_START signal
DBIAS	Touch FSM	The FSM selects different DBIAS based on the frequency mode to adjust the voltage of the touch sensor's internal LDO, which affects the charging and discharging speed of the touch sensor
DRV_HS	Touch FSM	The FSM selects different DRV_HS based on the frequency mode to adjust the high-speed driving capability of the touch sensor
DRV_LS	Touch FSM	The FSM selects different DRV_LS based on the frequency mode to adjust the low-speed driving capability of the touch sensor
DRES_LPF	Touch FSM	The FSM selects different DRES_LPF based on the frequency mode to adjust the RC low-pass filter, which can be configured as 0 kΩ, 3 kΩ, 4.5 kΩ, or 7.5 kΩ
DCAP_LPF	Touch FSM	The FSM selects different DCAP_LPF based on the frequency mode to adjust the RC low-pass filter with an adjustment range of 0-2.54 pF and a step of 20 fF
DCAP_DRV	Touch FSM	Used to adjust the internal capacitance connected to touch channels during measurement, usually set to 0
EN_CAL	Touch FSM	Used to disconnect from the touch pins and to test the capacitance in the touch sensor (the touch panel needs to be disabled)
BUF_DRV	Touch FSM	Adjusts the driving capability of TOUCH BUFF
EN_BUF	Touch FSM	Enables TOUCH BUFF, a software-configurable touch panel, to mitigate the effects of water droplets on the sensor for moisture tolerance

Signal	Source	Function
TOUCH_OUT	Touch sensor	The Touch FSM counts the pulses in the TOUCH_OUT signal and measures the time it takes to generate N pulses

27.4 Functional Description

27.4.1 Touch FSM

The Touch FSM performs a measurement by selecting a touch sensor and controlling the necessary signals to and from a touch sensor. Figure 27.4-1 shows the internal structure and operating mechanism of the Touch FSM. The Touch FSM is clocked by LP_DYN_FAST_CLK. For more information on LP_DYN_FAST_CLK, see Chapter 7 *Reset and Clock [to be added later]*.

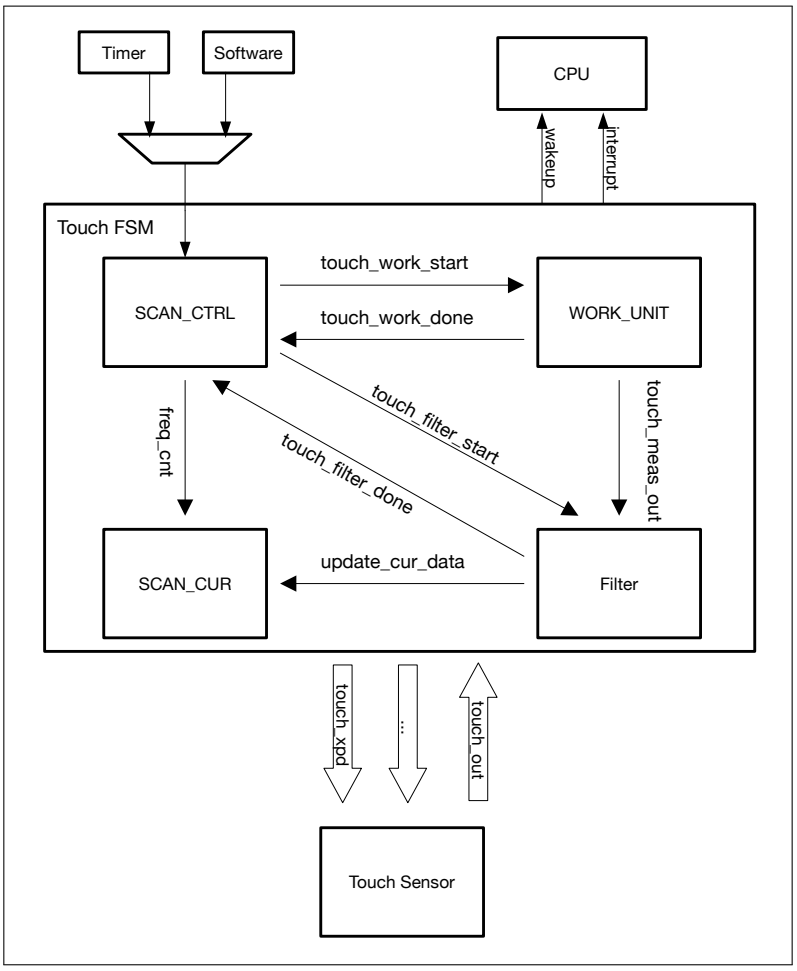


Figure 27.4-1. Touch FSM Structure and Mechanism

The following introduces the modules in the Touch FSM.

- SCAN_CTRL: Select the touch pin to measure, set the parameters such as hopping frequency, and controls the start and end of measurement for each touch sensor in scan mode.
- WORK_UNIT: Sample the selected touch sensors during a measurement.

- Filter: If enabled, each touch sensor can filter a series of measurements via an infinite impulse response (IIR) filter. The filtered value will be returned as the sampled value. See Section 27.4.3.1 for details.
- SCAN_CUR: Cache the parameters of different touch sensors at different frequency modes during the measurement to be used for the next measurement.

27.4.2 Sampled Signal Preprocessing

The core mechanism of the touch sensor is to reflect whether the touch panel is touched or not based on the change of the output pulse wave, i.e., the TOUCH_OUT signal, through detecting the change of capacitance on the touch panel. Therefore, the sampling of the TOUCH_OUT signal is critical. Figure 27.4-2 shows the preprocessing of the TOUCH_OUT signal.

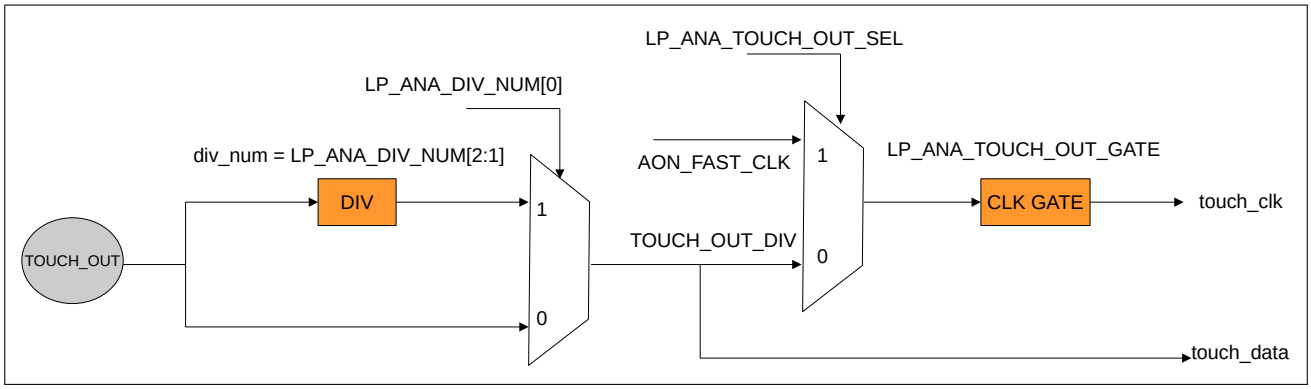


Figure 27.4-2. TOUCH_OUT Signal Preprocessing

The frequency of TOUCH_OUT might be much higher than the sampling frequency. Therefore, in order to improve the accuracy of the signal sampling, users can divide the TOUCH_OUT signal by configuring TOUCH_AON_DIV_NUM*n*. *n* in this field represents different frequency modes in frequency hopping (refer to Section 27.4.2.4) with the value 0-2. The lowest bit configures whether to enable or disable frequency division. Set the lowest bit to 0 to disable frequency division, and set it to 1 to enable division. The highest two bits indicate the frequency division coefficient with a value of 0/1/2/3, corresponding to the 0/2/4/6 divisions.

If the frequency is still higher than twice the sampling frequency after frequency division, set TOUCH_AON_TOUCH_OUT_SEL to 1 to directly use the divided TOUCH_OUT signal as a clock. Set TOUCH_AON_TOUCH_OUT_GATE to configure whether to enable this clock.

27.4.2.1 Measurement Process

Depending on whether frequency hopping is enabled, a measurement of a touch pin can be divided into the following two processes:

- Frequency hopping not enabled:
  1. The Touch FSM selects the touch sensor to be measured. The relevant signals are routed to that touch sensor.
  2. The Touch FSM drives the START signal to the touch sensor to initiate the measurement. Internally, the Touch FSM starts a touch counter to time the duration of the measurement.

3. A pulse counter in the Touch FSM is incremented on each pulse received from the touch sensor's TOUCH_OUT signal.
4. When the pulse counter reaches the count threshold set in TOUCH_AON_TOUCH_MEAS_NUMO, the measurement is complete. The START signal is de-asserted, and the touch counter is stopped. The value of the stopped touch counter × LP_DYN_SLOW_CLK cycles indicates the time taken to charge and discharge the touch pin for TOUCH_AON_TOUCH_MEAS_NUMO cycles.

- Frequency hopping enabled:

1. The Touch FSM selects the touch sensor to be measured. The relevant signals are routed to that touch sensor.
2. Set TOUCH_AON_FREQ_SCAN_EN to 1 to enable frequency hopping, configure TOUCH_AON_FREQ_SCAN_CNT_LIMIT to set the number of frequency modes supported for frequency hopping (up to three), and configure the parameters required for each frequency mode and the number of pulses for sampling the TOUCH_OUT signal.
3. The Touch FSM drives the START signal to the touch sensor to initiate the measurement. Internally, the Touch FSM starts a touch counter to time the duration of the measurement.
4. A pulse counter in the Touch FSM is incremented on each pulse received from the touch sensor's TOUCH_OUT signal.
5. When the pulse counter reaches the count threshold set in TOUCH_AON_TOUCH_MEAS_NUMO of the corresponding frequency mode, the measurement is complete. The value of the stopped touch counter indicates the time taken to charge and discharge the touch pin TOUCH_AON_TOUCH_MEAS_NUMO cycles with the current frequency.
6. The sampling frequency counter is incremented by 1 and the touch counter is cleared. The next frequency mode will be sampled after a small delay. When the samplings are finished for all frequency modes, the START signal is de-asserted, and the touch counter is cleared.

The sampled value (measured value) is the value of the touch counter indicated as touch_raw_data. The value of touch_raw_data can be read from TOUCH_PAD*n*_DATA. Note that the value returned by TOUCH_PAD*n*_DATA may also be the filtered touch_raw_data, i.e., touch_smooth_data and benchmark. Configure TOUCH_AON_TOUCH_DATA_SEL and TOUCH_AON_TOUCH_FREQ_SEL to select the type of the return value. For more information on the sampled value types and how to judge touch actions based on the data, refer to Section 27.4.3.1.

Note:
Note that if the pulse does not reach the set threshold for a long time while the touch counter has reached the timeout threshold set in TOUCH_AON_TOUCH_TIMEOUT_NUM with TOUCH_AON_TOUCH_TIMEOUT_EN set as 1, the TOUCH_TIMEOUT_INT_RAW timeout interrupt will be triggered, indicating a circuit abnormality.

27.4.2.2 Trigger Source of Measurement

The Touch FSM initiates a measurement by sending a START signal. The START signal can either be triggered by software, or by a dedicated hardware timer known as the "touch timer". The use of a touch timer allows periodic measurements to be taken without software intervention.

The touch timer is clocked by LP_DYN_SLOW_CLK and should be configured with a period in number of LP_DYN_SLOW_CLK cycles. The START signal is generated when the touch timer expires. The touch timer is reset when the measurement completes and restarts counting until the next expiration.

- To trigger the START signal by software:
  - Set TOUCH_AON_TOUCH_START_FORCE to 1 to trigger the START signal by software.
  - Software sets TOUCH_AON_TOUCH_START_EN to 1 and generates the START signal to initiate a measurement.
- To trigger the START signal by the touch timer:
  - Clear TOUCH_AON_TOUCH_START_FORCE.
  - Configure the wait time before the timer sends the START signal (unit: LP_DYN_SLOW_CLK cycle) via PMU_TOUCH_WAIT_CYCLES.
  - Configure the wait time after the timer receives the DONE signal upon the completion of the touch sensor sampling (unit: LP_DYN_SLOW_CLK cycle) via PMU_TOUCH_SLEEP_CYCLES.
  - Set PMU_TOUCH_SLEEP_TIMER_EN to 1 to enable the touch timer.
- To force exit the current measurement cycle of the touch timer by software, there are two approaches:
  1. Generate the end signal of the touch sensor to exit the current measurement cycle of the touch timer.
     - Set TOUCH_AON_TOUCH_DONE_FORCE to 1 to select the DONE signal to be triggered by software.
     - Software sets TOUCH_AON_TOUCH_DONE_EN to 1 to generate the DONE signal to end a measurement.
  2. Use the software to directly force the touch timer to exit the current measurement cycle.
     - Set PMU_TOUCH_FORCE_DONE to generate the DONE signal and force the touch timer to exit the current measurement cycle.

Note:
The touch timer waits for a period of time before generating the START signal and after receiving the DONE signal, so the interval between two consecutive measurements is (PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES)□and the timing clock is LP_DYN_SLOW_CLK.

27.4.2.3 Scan Mode

Scan mode involves the Touch FSM taking measurements of multiple touch sensors in sequential order. On every START signal, a new touch sensor is selected for measurement, thus allowing multiple touch pins to be monitored. The scan process is illustrated in Figure 27.4-3.

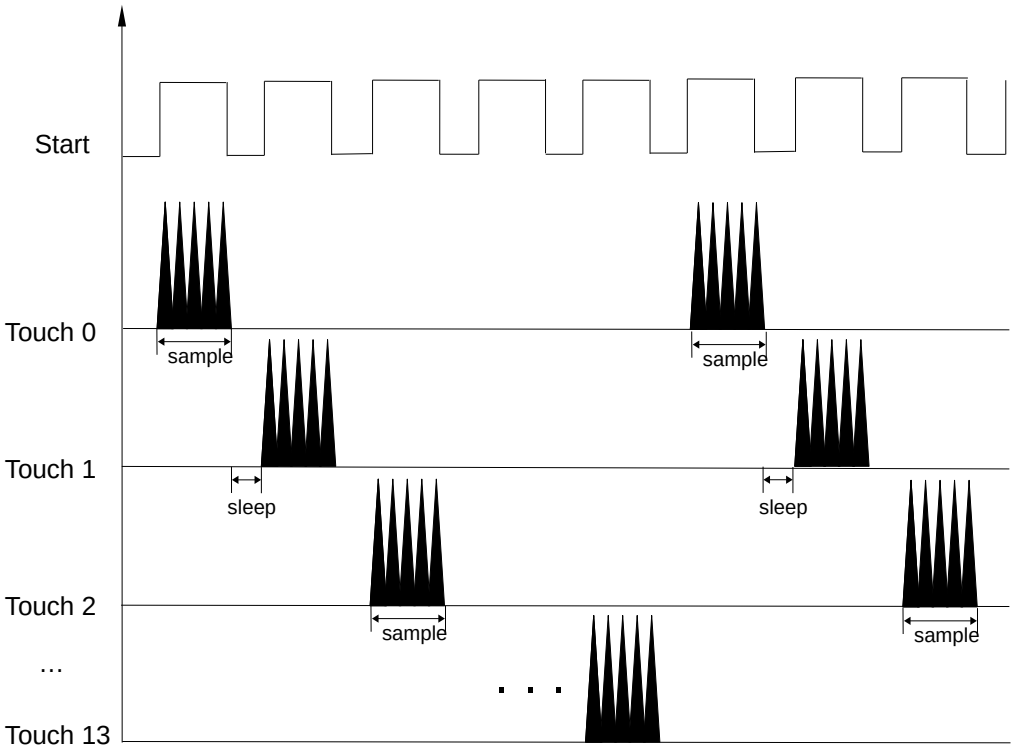


Figure 27.4-3. Timing Diagram of Scan Mode

To enable the scan mode, specify the bit map of the enabled touch pins in TOUCH_AON_TOUCH_SCAN_PAD_MAP. The Touch FSM will select one touch pin from the bit map of the enabled touch pins in turn to measure per START signal, i.e., only one measurement is conducted per START signal. As multiple START signals are generated, the Touch FSM will cycle through the enabled touch pins in sequential order.

When measuring for different touch sensors, the sleep interval between two measurements is PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES with the clock cycle of LP_DYN_SLOW_CLK.

Note:

- After the touch sensor receives the START signal from the touch timer, it waits a short period before starting the measurement to ensure the stability of the sampling signal. The wait time is configured by TOUCH_AON_TOUCH_XPD_WAIT. The timing clock is AON_FAST_CLK.
- If the touch timer is used to generate the START signal, the scanning can be conducted without software intervention.

27.4.2.4 Frequency Hopping

When frequency hopping is enabled, the Touch FSM can measure the touch sensor at different frequency modes. Up to three frequency modes are supported, each corresponding to a set of configuration parameters. When a START signal is generated, the measurement will be carried out for each frequency mode sequentially, and the DONE signal will be generated when the measurements are completed for all frequency modes. Frequency hopping can be enabled together with the scan mode. In such case, users can monitor

the touch condition of multiple touch sensors under different frequency modes by generating multiple START signals. The scanning process is shown in Figure 27.4-4.

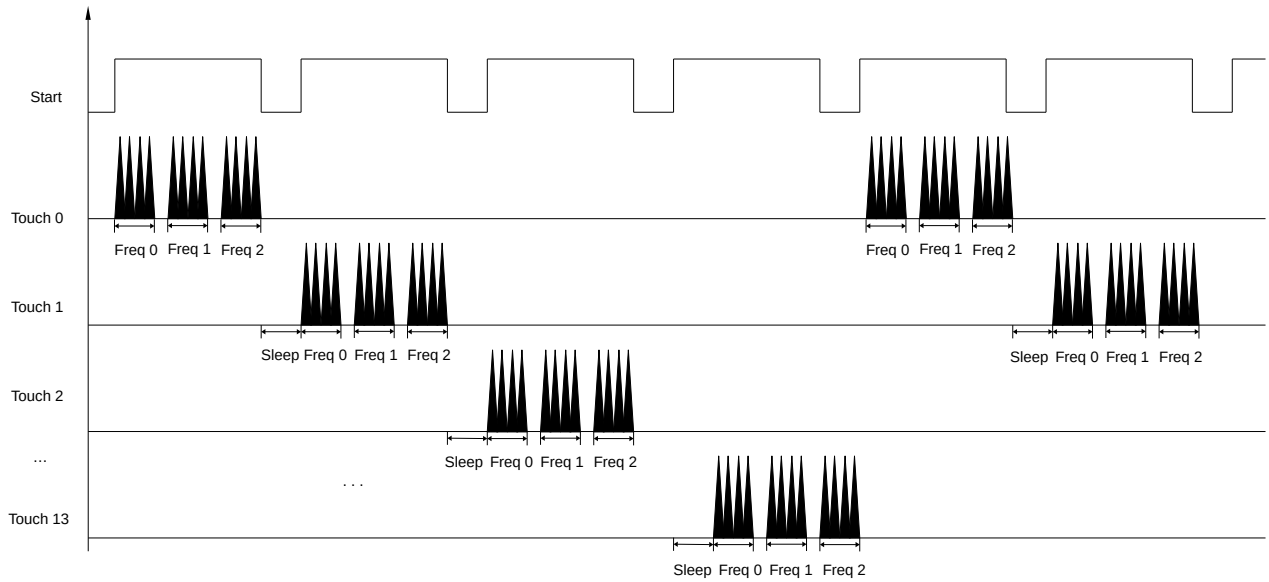


Figure 27.4-4. Touch Sensor Frequency Hopping Sequence Timing

Set TOUCH_AON_FREQ_SCAN_EN to 1 to enable frequency hopping, configure TOUCH_AON_FREQ_SCAN_CNT_LIMIT to select the maximum number of frequency mode supported, and configure the following analog parameter registers to adjust the sampling performance of the touch sensor at different frequency modes:

- TOUCH_AON_TOUCH_FREQ*n*_DBIAS (*n* = 0-2)
- TOUCH_AON_TOUCH_FREQ*n*_DRV_HS (*n* = 0-2)
- TOUCH_AON_TOUCH_FREQ*n*_DRV_LS (*n* = 0-2)
- TOUCH_AON_TOUCH_FREQ*n*_DRES_LPF (*n* = 0-2)
- TOUCH_AON_TOUCH_FREQ*n*_DCAP_LPF (*n* = 0-2)

Upon each START signal, the Touch FSM measures consecutively for frequency mode 0, 1, and 2. A START signal triggers a maximum of three measurement operations. When the measurements are completed for all frequency modes, the START signal is de-asserted until the next measurement. Therefore, the Touch FSM performs measurements on the enabled touch sensors in sequence upon multiple START signals. A touch on the same sensor is recognized only when the number of hit frequency points is greater than or equal to TOUCH_AON_FREQ_SCAN_CNT_RISE.

Note:

- When the same touch sensor is measured at different frequency modes, the wait time between measurements can be configured by TOUCH_AON_TOUCH_XPD_WAIT (count clock: AON_FAST_CLK).
- After receiving the START signal from the touch timer, the touch sensor waits for a short period of time before starting measuring to ensure the stability of the sampling signal. The wait time can be configured by TOUCH_AON_TOUCH_XPD_WAIT (count clock: AON_FAST_CLK).
- If a touch timer is used to generate the START signal, the scanning can be conducted without software inter-

vention.

27.4.3 Touch Detection

27.4.3.1 Sampled Value

As introduced in Section 27.4.2.1, the sampled value of a measurement can be read from TOUCH_PAD*n*_DATA. However, the specific type of sampled value stored in TOUCH_PAD*n*_DATA is determined together by TOUCH_AON_TOUCH_DATA_SEL and TOUCH_AON_TOUCH_FREQ_SEL. The following three sampled value types are supported:

1. **touch_raw_data**: When the measurement is completed, the touch counter's value indicates the time taken to charge and discharge the touch pin a fixed number of times in LP_DYN_SLOW_CLK cycles. After the IIR filter is turned on, the data is the filtered data.
2. **touch_smooth_data**: A series of moving averages generated by the touch_raw_data values sampled from a single touch sensor filtered by an IIR filter.

   touch_smooth_data is less prone to noise spikes or outlier samples, thus often used for hardware touch detection. If frequency hopping is enabled, each touch pin will have a separate set of smooth_touch_data at different frequency modes and do not affect each other. The IIR filter that generates touch_smooth_data is configured via TOUCH_AON_TOUCH_SMOOTH_LVL. See Table 27.4-1 for more information.

Table 27.4-1. touch_smooth_data Calculation

TOUCH_AON_TOUCH_SMOOTH_LVL	Type	Formula
0	—	touch_raw_data
1	IIR 1/2	1/2 touch_raw_data + 1/2 touch_smooth_data
2	IIR 1/4	1/4 touch_raw_data + 3/4 touch_smooth_data
3	IIR 1/8	1/8 touch_raw_data + 7/8 touch_smooth_data

3. **benchmark**: Also generated from the touch_raw_data values by IIR filtering. However, the moving average window for benchmark is much wider.

   The benchmark value is usually used to represent the stable reading of a touch pin without the effect from a touch action. The touch detection is based on the benchmark value. Each touch sensor has a separate set of benchmark values at different frequency modes with a default value of 0xFFFF. The IIR filter that generates benchmark is configured via TOUCH_AON_TOUCH_FILTER_MODE. Refer to Table 27.4-2 for more information.

   In addition to the IIR filtering, users can also update the benchmark value directly by software. The software update procedure is as follows:

   - Configure the target frequency point to be updated through TOUCH_AON_TOUCH_UPDATE_BENCHMARK_FREQ_SEL.
   - Configure the target channel to be updated through TOUCH_AON_TOUCH_UPDATE_BENCHMARK_PAD_SEL.
   - Configure the benchmark data value through TOUCH_AON_TOUCH_BENCHMARK_SW.

- Configure TOUCH_AON_TOUCH_UPDATE_BENCHMARK_SW to trigger the benchmark data update by software.

This method is generally used for software debugging. When the benchmark data of the target frequency point and channel has been updated, the interrupt TOUCH_BENCHMARK_UPDATE_INT will be generated.

Table 27.4-2. benchmark Algorithm

TOUCH_AON_TOUCH_FILTER_MODE	Type	Formula
0	IIR 1/4	1/4 touch_raw_data + 3/4 benchmark
1	IIR 1/8	1/8 touch_raw_data + 7/8 benchmark
2	IIR 1/16	1/16 touch_raw_data + 15/16 benchmark
3	IIR 1/32	1/32 touch_raw_data + 31/32 benchmark
4	IIR 1/64	1/64 touch_raw_data + 63/64 benchmark
5	IIR 1/128	1/128 touch_raw_data + 127/128 benchmark
6	IIR 1/256	1/256 touch_raw_data + 255/256 benchmark
7	JITTER	touch_raw_data +/– TOUCH_AON_TOUCH_JITTER_STEP

27.4.3.2 Hardware Touch Detection

In hardware touch detection, the touch sensor can detect finger touch or release and trigger an interrupt. To use hardware touch detection, the following key parameters need to be defined:

- **finger_threshold**: The threshold value used to determine a touch or a touch release. finger_threshold is configured via TOUCH_AON_TOUCH_PAD*x*_TH*y*, where *x* (0-13, corresponding to T0-T13) represents the touch pin index and *y* (0-2) represents the frequency mode configured in hopping frequency.
- **noise_threshold**: Indicates the maximum threshold of the upward and downward fluctuation of touch_smooth_data. noise_threshold is configured via TOUCH_AON_TOUCH_NOISE_THRES.
- **hysteresis**: Used to avoid touch misjudgments caused by fluctuations of touch_smooth_data around touch thresholds. The hysteresis value is configured via TOUCH_AON_TOUCH_HYSTERESIS.

The finger_threshold and noise_threshold values are not absolute thresholds, but offsets from the benchmark value, while the hysteresis value is an offset from finger_threshold. This prevents from misdiagnosing the gradual or slow changes of touch_raw_data caused by environmental factors such as temperature, power, or noise as a touch action.

After a series of calculations and comparisons based on the above parameters, the touch sensor will output the touch results detected on the touch pins via interrupts. Each touch sensor corresponds to a control register TOUCH_AON_TOUCH_OUTEN to configure whether to output the detection result.

27.4.4 Proximity Mode

When an object such as a finger is close to, but not touching, a touch pin, a small change in capacitance occurs on the touch pin, which is much smaller than the change caused by a physical touch. Proximity mode allows for the detection of these small changes.

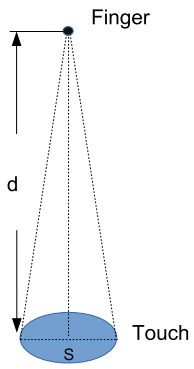


Figure 27.4-5. Sensing Area

- Detection distance "d" is positively correlated with sensing area "S". That is, with other conditions unchanged, a larger sensing area supports a longer detection distance. As shown in Figure 27.4-5, the current specification is a maximum detection distance of 16 cm and a sensing area of 20 cm^2.
- Up to three touch pins can be configured to operate simultaneously in proximity mode.
- Frequency hopping is also supported in proximity mode.

In proximity mode, a touch sensor samples for a fixed number of times and accumulates the sample values. If the final cumulative value exceeds the configured threshold, a proximity object is considered to be detected and an interrupt will be triggered. Note that since the sample values are accumulated, the values introduced in Section 27.4.3.1, i.e., touch_raw_data, touch_smooth_data, and benchmark, will not be generated in proximity mode.

To operate in proximity mode:

- Configure a touch sensor to operate in proximity mode by setting TOUCH_AON_TOUCH_APPROACH_PAD0, TOUCH_AON_TOUCH_APPROACH_PAD1, or TOUCH_AON_TOUCH_APPROACH_PAD2.
- Configure TOUCH_AON_TOUCH_APPROACH_LIMIT to adjust the number of samplings to generate the cumulative value. The touch sensor contains an internal sample counter to record the number of samplings.
- Set the threshold value via TOUCH_AON_TOUCH_PAD*x*_TH*y*, where *x* (0-13, corresponding to T0-T13) represents the touch pin index and *y* (0-2) represents the frequency mode configured in hopping frequency.
- When the sample counter reaches the number of samplings configured in TOUCH_AON_TOUCH_APPROACH_MEAS_NUM*n*:
  - If the cumulative value exceeds the threshold value, an interrupt will be triggered.
  - The sample counter and the cumulative value are reset to 0. The touch sensor restarts accumulating the sample values.

27.4.5 Sleep Mode

To reduce power consumption, the touch sensor also supports sleep mode. In sleep mode, users can designate one touch sensor to operate actively (hereinafter referred to as the sleeping touch sensor) to monitor whether a touch occurs in real time, while the remaining 13 touch sensors are all powered down in

sleep mode. Once the sleeping touch sensor detects a touch or a touch release, it will generate an interrupt to wake up the rest of the touch sensors. The sleeping touch sensor also supports frequency hopping and proximity mode. There is a separate set of threshold parameters for the sleeping touch sensor, which can be configured as follows:

- Configure TOUCH_AON_TOUCH_SLP_PAD to select a particular touch sensor as the sleeping touch sensor.
- Configure TOUCH_AON_TOUCH_SLP_TH0, TOUCH_AON_TOUCH_SLP_TH1, and TOUCH_AON_TOUCH_SLP_TH2 to set the touch threshold for the sleeping touch sensor at different frequency modes.
- Configure TOUCH_AON_TOUCH_SLP_APPROACH_EN to enable the proximity mode for the sleeping touch sensor.
- Configure TOUCH_AON_TOUCH_SLP_CHANNEL_CLR to clear the intermediate data for the sleeping touch sensor, such as benchmark, touch_smooth_data, and the number of samplings.

Note:
The configuration for the noise_threshold, hysteresis, touch_smooth_data, and benchmark values are the same for the sleeping touch sensor in sleep mode as in the normal mode.

27.4.6 Moisture Tolerance

If there are water droplets on the touch pads and the droplets are large enough to physically bridge two or more adjacent touch pads, the adjacent touch pads may be electrically coupled. Coupled touch pads can cause false touch detection due to capacitance changes caused by the coupling. If a water droplet is connected across the touch pad and ground, a large parasitic capacitance will be introduced, also leading to false detection. The moisture tolerance feature can mitigate the impact of water droplets.

To configure moisture tolerance:

- Configure TOUCH_AON_TOUCH_SHIELD_PAD_EN to enable moisture tolerance.
- Configure TOUCH_AON_TOUCH_BUFSEL to select a touch pin to be used for moisture tolerance.

Note:
The moisture tolerance feature only works at frequencies below 10 MHz. Therefore, if the touch sensor output sampling frequency exceeds 10 MHz, the moisture tolerance feature must be disabled.

27.4.7 Water Rejection

If the sensor array becomes wet, i.e., the majority of the sensor pads are soaked by water, most (if not all) of the touch pads will become unusable due to the possible false touch detections. ESP32-S31 supports the water rejection feature to shut down the sensor array if it is detected to be wet.

Configure TOUCH_AON_TOUCH_OUT_RING to select one of the touch pins to be used for the water rejection feature.

27.5 Interrupts

ESP32-S31's touch sensor can generate the LP_TOUCH_INTR signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from touch sensor that can generate the LP_TOUCH_INTR signal. The interrupt sources from touch sensor are listed with their trigger conditions and the resulted interrupt signal in Table 27.5-1.

Table 27.5-1. Touch Sensor's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TOUCH_APPROACH_LOOP_DONE_INT	Completion of proximity mode cumulative sampling	LP_TOUCH_INTR
TOUCH_TIMEOUT_INT	Timeout for the TOUCH_OUT signal sampling	
TOUCH_INACTIVE_INT	Touch release detected	
TOUCH_ACTIVE_INT	Touch detected	
TOUCH_DONE_INT	Completion of sampling an individual touch pin at a certain frequency mode for comparison	
TOUCH_SCAN_DONE_INT	Completion of sampling selected touch pins for comparison at all frequency modes	
TOUCH_BENCHMARK_UPDATE_INT	Completion of software benchmark data update	

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > Section 9.1 *Interrupt Terminology*.

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 27.6 *Register Summary*.

27.6 Register Summary

27.6.1 Interrupt and Status Register Summary

The addresses in this section are relative to Touch Sensor base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Interrupt Register			
TOUCH_INT_RAW_REG	Touch sensor interrupt raw register	0x0000	R/SS/WTC
TOUCH_INT_ST_REG	Touch sensor interrupt status register	0x0004	RO
TOUCH_INT_ENA_REG	Touch sensor interrupt enable register	0x0008	R/W
TOUCH_INT_CLR_REG	Touch sensor interrupt clear register	0x000C	WT
Status Register			
TOUCH_CHN_STATUS_REG	Scan mode status register	0x0010	RO
TOUCH_STATUS_0_REG	Touch pin 0 status register	0x0014	RO
TOUCH_STATUS_1_REG	Touch pin 1 status register	0x0018	RO
TOUCH_STATUS_2_REG	Touch pin 2 status register	0x001C	RO
TOUCH_STATUS_3_REG	Touch pin 3 status register	0x0020	RO
TOUCH_STATUS_4_REG	Touch pin 4 status register	0x0024	RO
TOUCH_STATUS_5_REG	Touch pin 5 status register	0x0028	RO
TOUCH_STATUS_6_REG	Touch pin 6 status register	0x002C	RO
TOUCH_STATUS_7_REG	Touch pin 7 status register	0x0030	RO
TOUCH_STATUS_8_REG	Touch pin 8 status register	0x0034	RO
TOUCH_STATUS_9_REG	Touch pin 9 status register	0x0038	RO
TOUCH_STATUS_10_REG	Touch pin 10 status register	0x003C	RO
TOUCH_STATUS_11_REG	Touch pin 11 status register	0x0040	RO
TOUCH_STATUS_12_REG	Touch pin 12 status register	0x0044	RO
TOUCH_STATUS_13_REG	Touch pin 13 status register	0x0048	RO
TOUCH_STATUS_15_REG	Sleep mode touch pin status register	0x0050	RO

Name	Description	Address	Access
TOUCH_STATUS_16_REG	Proximity mode status register	0x0054	RO
TOUCH_STATUS_17_REG	Frequency hopping status register	0x0058	RO
TOUCH_CHN_TMP_STATUS_REG	Touch status register	0x005C	RO
Version Control Register			
TOUCH_DATE_REG	Version control register	0x0100	R/W

27.6.2 Configuration Register Summary

The addresses in this section are relative to LP Analog Peripheral base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Name	Description	Address	Access
Configuration Registers			
TOUCH_AON_APPROACH_WORK_MEAS_NUM_REG	Configuration register for the number of measurements in proximity mode	0x0000	R/W
TOUCH_AON_SCAN_CTRL1_REG	Scan mode configuration register 1	0x0004	R/W
TOUCH_AON_SCAN_CTRL2_REG	Scan mode configuration register 2	0x0008	R/W
TOUCH_AON_WORK_REG	Sample preprocessing register	0x000C	varies
TOUCH_AON_WORK_MEAS_NUM_REG	Configuration register for number of measurements at different frequency modes	0x0010	R/W
TOUCH_AON_FILTER1_REG	Touch detection configuration register 1	0x0014	R/W
TOUCH_AON_FILTER2_REG	Touch detection configuration register 2	0x0018	R/W
TOUCH_AON_FILTER3_REG	Touch detection configuration register 3	0x001C	varies
TOUCH_AON_SLP0_REG	Sleep mode configuration register 1	0x0020	varies
TOUCH_AON_SLP1_REG	Sleep mode configuration register 2	0x0024	R/W
TOUCH_AON_CLR_REG	Status clear register	0x0028	WT
TOUCH_AON_APPROACH_REG	Proximity mode configuration register	0x002C	R/W
TOUCH_AON_FREQ0_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 0	0x0030	R/W
TOUCH_AON_FREQ1_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 1	0x0034	R/W
TOUCH_AON_FREQ2_SCAN_PARA_REG	Analog parameter configuration register for touch sensor for frequency mode 2	0x0038	R/W

Name	Description	Address	Access
TOUCH_AON_ANA_PARA_REG	Touch sensor analog parameter configuration register	0x003C	R/W
TOUCH_AON_MUX0_REG	DEBUG mode selection configuration register	0x0040	R/W
TOUCH_AON_MUX1_REG	DEBUG mode function configuration register	0x0044	R/W
TOUCH_AON_PAD0_TH0_REG	Touch threshold configuration register for touch pin 0 in frequency mode 0	0x0048	R/W
TOUCH_AON_PAD0_TH1_REG	Touch threshold configuration register for touch pin 0 in frequency mode 1	0x004C	R/W
TOUCH_AON_PAD0_TH2_REG	Touch threshold configuration register for touch pin 0 in frequency mode 2	0x0050	R/W
TOUCH_AON_PAD1_TH0_REG	Touch threshold configuration register for touch pin 1 in frequency mode 0	0x0054	R/W
TOUCH_AON_PAD1_TH1_REG	Touch threshold configuration register for touch pin 1 in frequency mode 1	0x0058	R/W
TOUCH_AON_PAD1_TH2_REG	Touch threshold configuration register for touch pin 1 in frequency mode 2	0x005C	R/W
TOUCH_AON_PAD2_TH0_REG	Touch threshold configuration register for touch pin 2 in frequency mode 0	0x0060	R/W
TOUCH_AON_PAD2_TH1_REG	Touch threshold configuration register for touch pin 2 in frequency mode 1	0x0064	R/W
TOUCH_AON_PAD2_TH2_REG	Touch threshold configuration register for touch pin 2 in frequency mode 2	0x0068	R/W
TOUCH_AON_PAD3_TH0_REG	Touch threshold configuration register for touch pin 3 in frequency mode 0	0x006C	R/W
TOUCH_AON_PAD3_TH1_REG	Touch threshold configuration register for touch pin 3 in frequency mode 1	0x0070	R/W
TOUCH_AON_PAD3_TH2_REG	Touch threshold configuration register for touch pin 3 in frequency mode 2	0x0074	R/W
TOUCH_AON_PAD4_TH0_REG	Touch threshold configuration register for touch pin 4 in frequency mode 0	0x0078	R/W
TOUCH_AON_PAD4_TH1_REG	Touch threshold configuration register for touch pin 4 in frequency mode 1	0x007C	R/W
TOUCH_AON_PAD4_TH2_REG	Touch threshold configuration register for touch pin 4 in frequency mode 2	0x0080	R/W
TOUCH_AON_PAD5_TH0_REG	Touch threshold configuration register for touch pin 5 in frequency mode 0	0x0084	R/W
TOUCH_AON_PAD5_TH1_REG	Touch threshold configuration register for touch pin 5 in frequency mode 1	0x0088	R/W
TOUCH_AON_PAD5_TH2_REG	Touch threshold configuration register for touch pin 5 in frequency mode 2	0x008C	R/W
TOUCH_AON_PAD6_TH0_REG	Touch threshold configuration register for touch pin 6 in frequency mode 0	0x0090	R/W
TOUCH_AON_PAD6_TH1_REG	Touch threshold configuration register for touch pin 6 in frequency mode 1	0x0094	R/W
TOUCH_AON_PAD6_TH2_REG	Touch threshold configuration register for touch pin 6 in frequency mode 2	0x0098	R/W
TOUCH_AON_PAD7_TH0_REG	Touch threshold configuration register for touch pin 7 in frequency mode 0	0x009C	R/W
TOUCH_AON_PAD7_TH1_REG	Touch threshold configuration register for touch pin 7 in frequency mode 1	0x00A0	R/W
TOUCH_AON_PAD7_TH2_REG	Touch threshold configuration register for touch pin 7 in frequency mode 2	0x00A4	R/W
TOUCH_AON_PAD8_TH0_REG	Touch threshold configuration register for touch pin 8 in frequency mode 0	0x00A8	R/W
TOUCH_AON_PAD8_TH1_REG	Touch threshold configuration register for touch pin 8 in frequency mode 1	0x00AC	R/W

Name	Description	Address	Access
TOUCH_AON_PAD8_TH2_REG	Touch threshold configuration register for touch pin 8 in frequency mode 2	0x00B0	R/W
TOUCH_AON_PAD9_TH0_REG	Touch threshold configuration register for touch pin 9 in frequency mode 0	0x00B4	R/W
TOUCH_AON_PAD9_TH1_REG	Touch threshold configuration register for touch pin 9 in frequency mode 1	0x00B8	R/W
TOUCH_AON_PAD9_TH2_REG	Touch threshold configuration register for touch pin 9 in frequency mode 2	0x00BC	R/W
TOUCH_AON_PAD10_TH0_REG	Touch threshold configuration register for touch pin 10 in frequency mode 0	0x00C0	R/W
TOUCH_AON_PAD10_TH1_REG	Touch threshold configuration register for touch pin 10 in frequency mode 1	0x00C4	R/W
TOUCH_AON_PAD10_TH2_REG	Touch threshold configuration register for touch pin 10 in frequency mode 2	0x00C8	R/W
TOUCH_AON_PAD11_TH0_REG	Touch threshold configuration register for touch pin 11 in frequency mode 0	0x00CC	R/W
TOUCH_AON_PAD11_TH1_REG	Touch threshold configuration register for touch pin 11 in frequency mode 1	0x00D0	R/W
TOUCH_AON_PAD11_TH2_REG	Touch threshold configuration register for touch pin 11 in frequency mode 2	0x00D4	R/W
TOUCH_AON_PAD12_TH0_REG	Touch threshold configuration register for touch pin 12 in frequency mode 0	0x00D8	R/W
TOUCH_AON_PAD12_TH1_REG	Touch threshold configuration register for touch pin 12 in frequency mode 1	0x00DC	R/W
TOUCH_AON_PAD12_TH2_REG	Touch threshold configuration register for touch pin 12 in frequency mode 2	0x00E0	R/W
TOUCH_AON_PAD13_TH0_REG	Touch threshold configuration register for touch pin 13 in frequency mode 0	0x00E4	R/W
TOUCH_AON_PAD13_TH1_REG	Touch threshold configuration register for touch pin 13 in frequency mode 1	0x00E8	R/W
TOUCH_AON_PAD13_TH2_REG	Touch threshold configuration register for touch pin 13 in frequency mode 2	0x00EC	R/W
TOUCH_AON_DATE_REG	Version control register	0x00FC	R/W

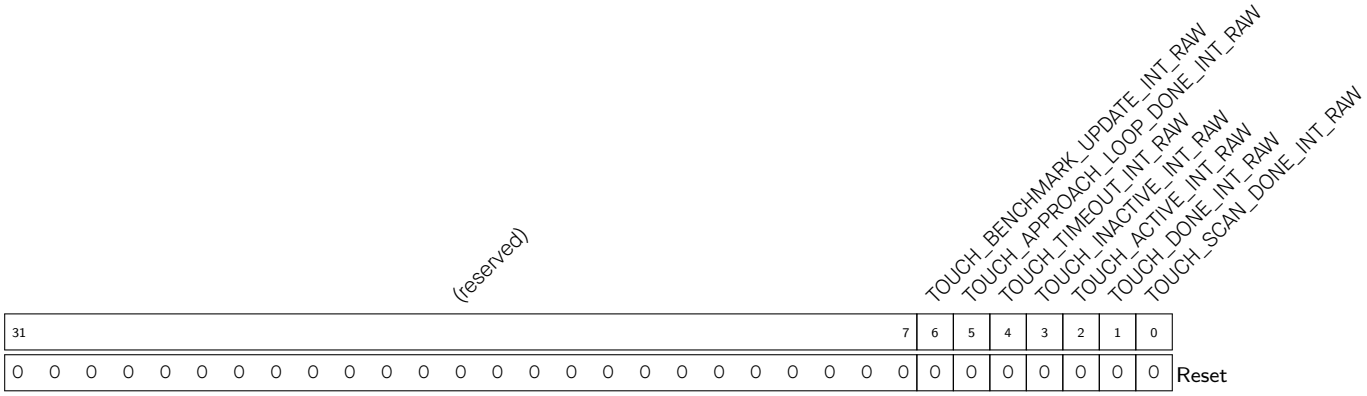
27.7 Registers

27.7.1 Interrupt and Status Registers

The addresses in this section are relative to Touch Sensor base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 27.1. TOUCH_INT_RAW_REG (0x0000)



TOUCH_SCAN_DONE_INT_RAW The raw interrupt status of TOUCH_SCAN_DONE_INT. (R/SS/WTC)

TOUCH_DONE_INT_RAW The raw interrupt status of TOUCH_DONE_INT. (R/SS/WTC)

TOUCH_ACTIVE_INT_RAW The raw interrupt status of TOUCH_ACTIVE_INT. (R/SS/WTC)

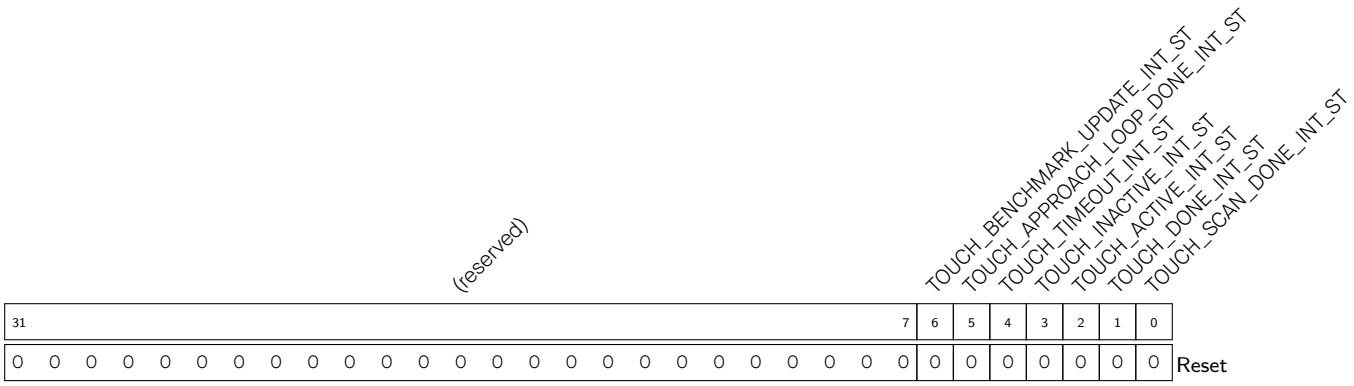
TOUCH_INACTIVE_INT_RAW The raw interrupt status of TOUCH_INACTIVE_INT. (R/SS/WTC)

TOUCH_TIMEOUT_INT_RAW The raw interrupt status of TOUCH_TIMEOUT_INT. (R/SS/WTC)

TOUCH_APPROACH_LOOP_DONE_INT_RAW The raw interrupt status of TOUCH_APPROACH_LOOP_DONE_INT. (R/SS/WTC)

TOUCH_BENCHMARK_UPDATE_INT_RAW The raw interrupt status of TOUCH_BENCHMARK_UPDATE_INT. (R/SS/WTC)

Register 27.2. TOUCH_INT_ST_REG (0x0004)



TOUCH_SCAN_DONE_INT_ST The masked interrupt status of TOUCH_SCAN_DONE_INT. (RO)

TOUCH_DONE_INT_ST The masked interrupt status of TOUCH_DONE_INT. (RO)

TOUCH_ACTIVE_INT_ST The masked interrupt status of TOUCH_ACTIVE_INT. (RO)

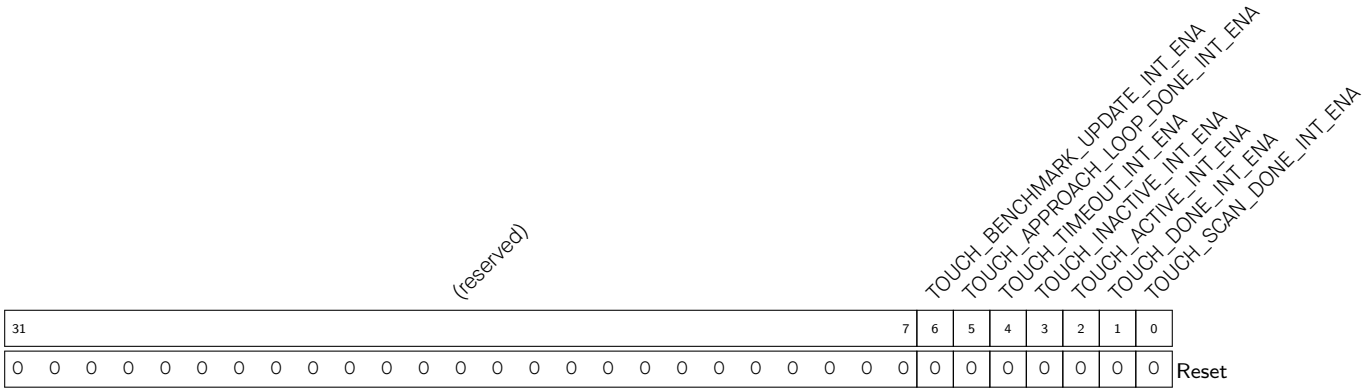
TOUCH_INACTIVE_INT_ST The masked interrupt status of TOUCH_INACTIVE_INT. (RO)

TOUCH_TIMEOUT_INT_ST The masked interrupt status of TOUCH_TIMEOUT_INT. (RO)

TOUCH_APPROACH_LOOP_DONE_INT_ST The masked interrupt status of TOUCH_APPROACH_LOOP_DONE_INT. (RO)

TOUCH_BENCHMARK_UPDATE_INT_ST The masked interrupt status of TOUCH_BENCHMARK_UPDATE_INT. (RO)

Register 27.3. TOUCH_INT_ENA_REG (0x0008)



TOUCH_SCAN_DONE_INT_ENA Write 1 to enable TOUCH_SCAN_DONE_INT. (R/W)

TOUCH_DONE_INT_ENA Write 1 to enable TOUCH_DONE_INT. (R/W)

TOUCH_ACTIVE_INT_ENA Write 1 to enable TOUCH_ACTIVE_INT. (R/W)

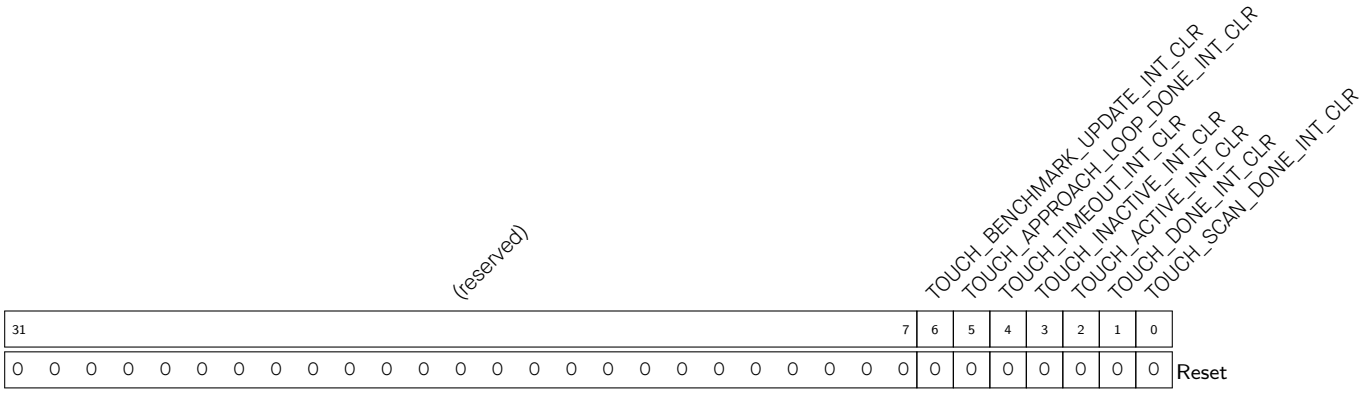
TOUCH_INACTIVE_INT_ENA Write 1 to enable TOUCH_INACTIVE_INT. (R/W)

TOUCH_TIMEOUT_INT_ENA Write 1 to enable TOUCH_TIMEOUT_INT. (R/W)

TOUCH_APPROACH_LOOP_DONE_INT_ENA Write 1 to enable TOUCH_APPROACH_LOOP_DONE_INT. (R/W)

TOUCH_BENCHMARK_UPDATE_INT_ENA Write 1 to enable TOUCH_BENCHMARK_UPDATE_INT. (R/W)

Register 27.4. TOUCH_INT_CLR_REG (0x000C)



TOUCH_SCAN_DONE_INT_CLR Write 1 to clear TOUCH_SCAN_DONE_INT. (WT)

TOUCH_DONE_INT_CLR Write 1 to clear TOUCH_DONE_INT. (WT)

TOUCH_ACTIVE_INT_CLR Write 1 to clear TOUCH_ACTIVE_INT. (WT)

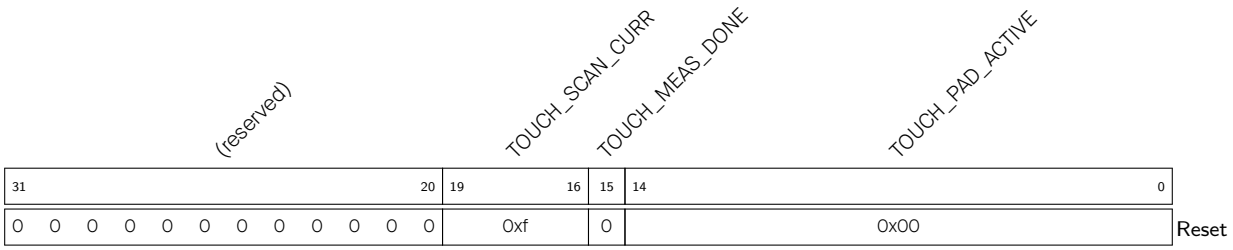
TOUCH_INACTIVE_INT_CLR Write 1 to clear TOUCH_INACTIVE_INT. (WT)

TOUCH_TIMEOUT_INT_CLR Write 1 to clear TOUCH_TIMEOUT_INT. (WT)

TOUCH_APPROACH_LOOP_DONE_INT_CLR Write 1 to clear TOUCH_APPROACH_LOOP_DONE_INT. (WT)

TOUCH_BENCHMARK_UPDATE_INT_CLR Write 1 to clear TOUCH_BENCHMARK_UPDATE_INT. (WT)

Register 27.5. TOUCH_CHN_STATUS_REG (0x0010)



TOUCH_PAD_ACTIVE Indicates whether the touch pin detects a touch. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: No touch detected

1: Touch detected

(RO)

TOUCH_MEAS_DONE Indicates whether the touch pins have been measured at all frequency modes.

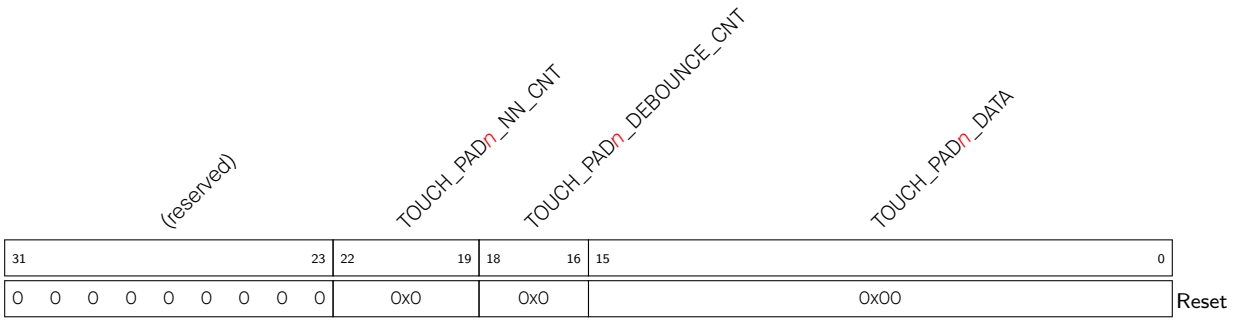
0: Not completed

1: Completed

(RO)

TOUCH_SCAN_CURR Indicates the index number of the currently scanned touch pin. (RO)

Register 27.6. TOUCH_STATUS_*n*_REG (*n*: 0-13) (0x0014+0x4**n*)

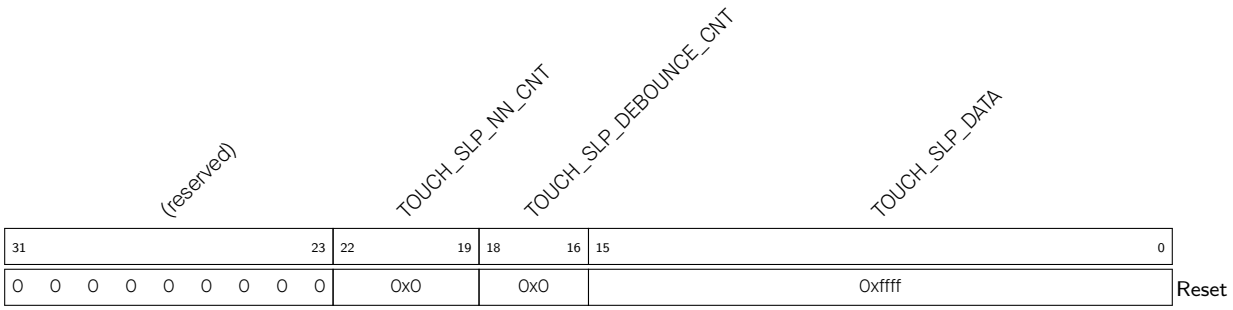


TOUCH_PAD*n*_DATA Indicates the touch_raw_data, touch_smooth_data, or benchmark sampled by touch pin *n*. (RO)

TOUCH_PAD*n*_DEBOUNCE_CNT Indicates the cumulative number of consecutive touch or touch release detection by touch pin *n*. (RO)

TOUCH_PAD*n*_NN_CNT Indicates the cumulative number of consecutive instances where $touch\_smooth\_data < benchmark - passive\_noise\_threshold$ has been detected by touch pin *n*. (RO)

Register 27.7. TOUCH_STATUS_15_REG (0x0050)

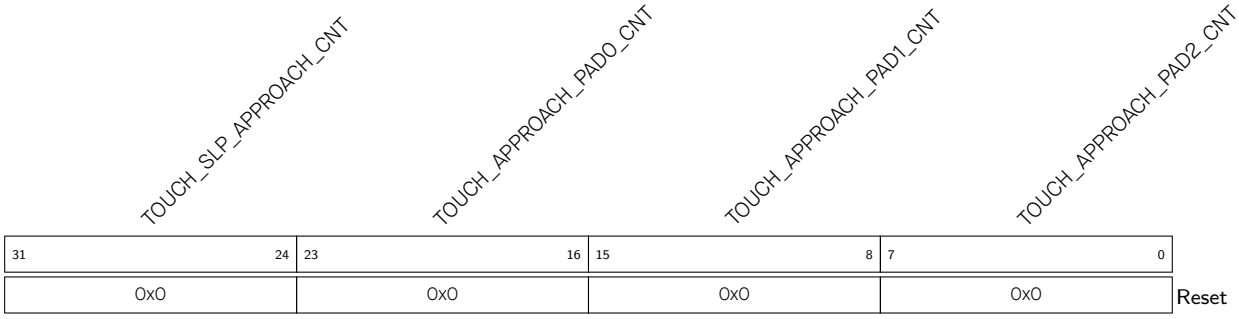


TOUCH_SLP_DATA Indicates the touch_smooth_data or benchmark sampled by touch pin in sleep mode. (RO)

TOUCH_SLP_DEBOUNCE_CNT Indicates the cumulative number of consecutive touch or touch release detection by touch pin in sleep mode. (RO)

TOUCH_SLP_NN_CNT Indicates the cumulative number of consecutive instances where $touch\_smooth\_data < benchmark - passive\_noise\_threshold$ has been detected by touch pin in sleep mode. (RO)

Register 27.8. TOUCH_STATUS_16_REG (0x0054)



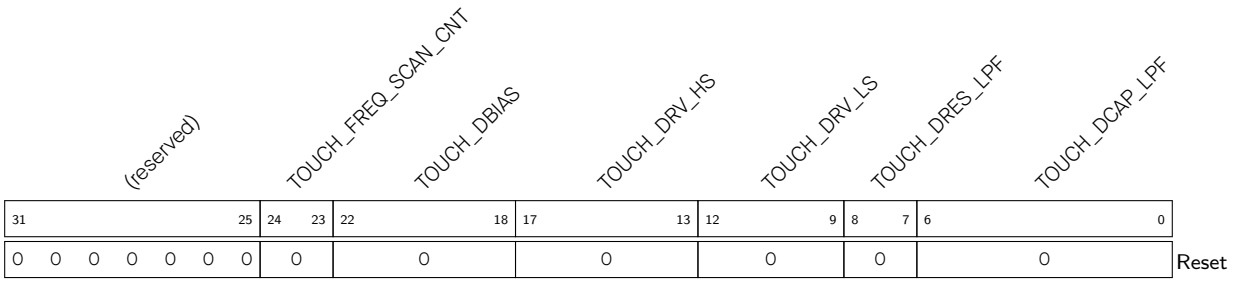
TOUCH_APPROACH_PAD2_CNT Indicates the cumulative number of samplings of touch pin 2 in proximity mode. (RO)

TOUCH_APPROACH_PAD1_CNT Indicates the cumulative number of samplings of touch pin 1 in proximity mode. (RO)

TOUCH_APPROACH_PAD0_CNT Indicates the cumulative number of samplings of touch pin 0 in proximity mode. (RO)

TOUCH_SLP_APPROACH_CNT Indicates the cumulative number of samplings of the sleeping touch pin in proximity mode. (RO)

Register 27.9. TOUCH_STATUS_17_REG (0x0058)



TOUCH_DCAP_LPF Indicates the DCAP_LPF value of the current touch pin. (RO)

TOUCH_DRES_LPF Indicates the DRES_LPF value of the current touch pin. (RO)

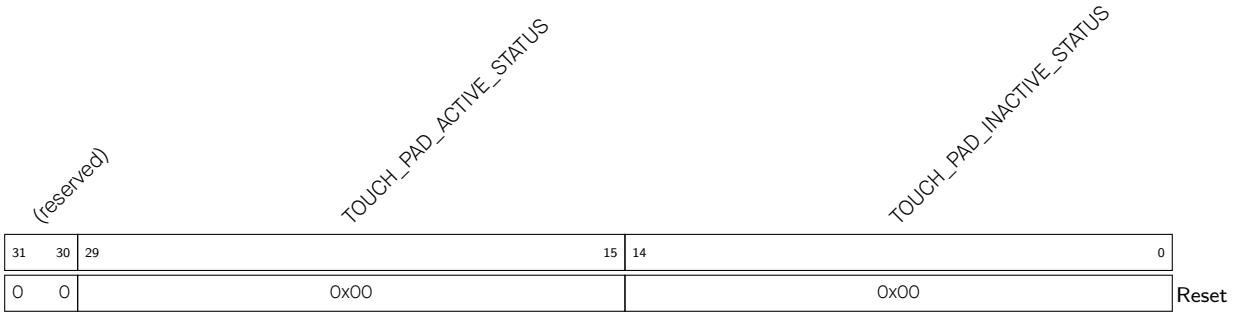
TOUCH_DRV_LS Indicates the DRV_LS value of the current touch pin. (RO)

TOUCH_DRV_HS Indicates the DRV_HS value of the current touch pin. (RO)

TOUCH_DBIAS Indicates the DBIAS value of the current touch pin. (RO)

TOUCH_FREQ_SCAN_CNT Indicates the SCAN_CNT value of the current touch pin. (RO)

Register 27.10. TOUCH_CHN_TMP_STATUS_REG (0x005C)



TOUCH_PAD_INACTIVE_STATUS Indicates whether the touch pin detects the touch pad from being touched to being released. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: Touch release not detected

1: Touch release detected

(RO)

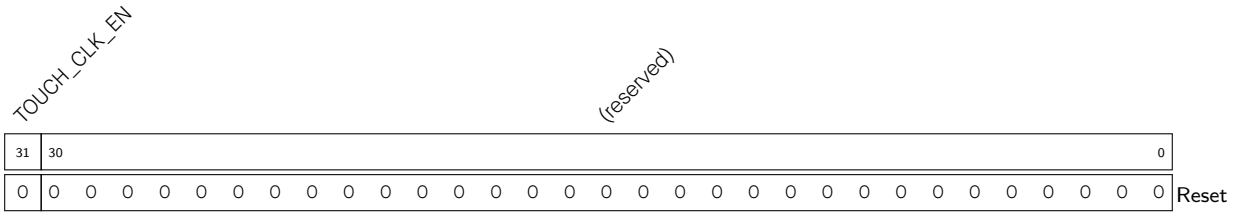
TOUCH_PAD_ACTIVE_STATUS Indicates whether the touch pin detects the process of being touched. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: Touch not detected

1: Touch detected

(RO)

Register 27.11. TOUCH_DATE_REG (0x0100)



TOUCH_CLK_EN Configures whether to enable the register read/write clock.

0: Disable

1: Enable

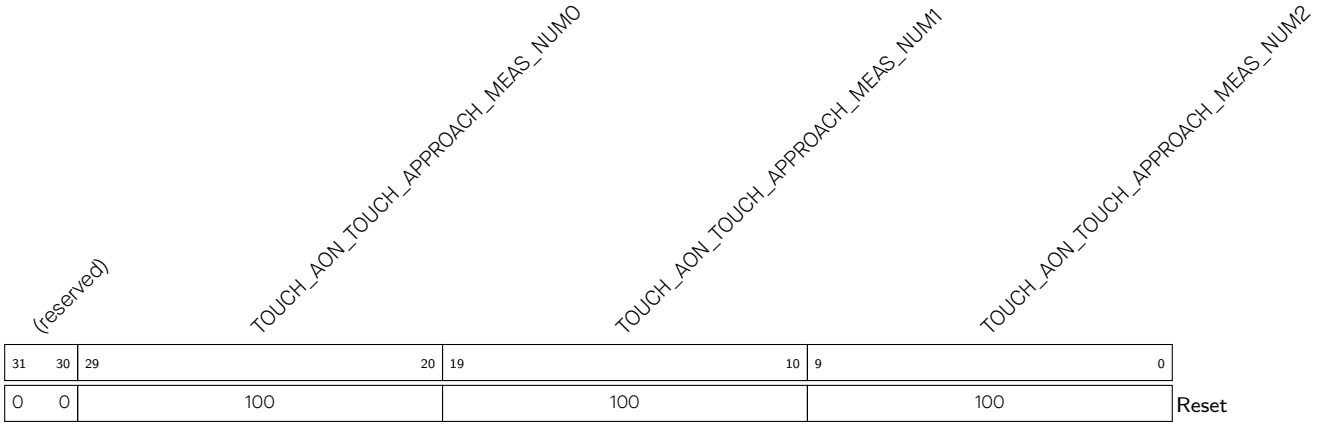
(R/W)

27.7.2 Configuration Registers

The addresses in this section are relative to LP Analog Peripheral base address provided in Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 27.12. TOUCH_AON_APPROACH_WORK_MEAS_NUM_REG (0x0000)

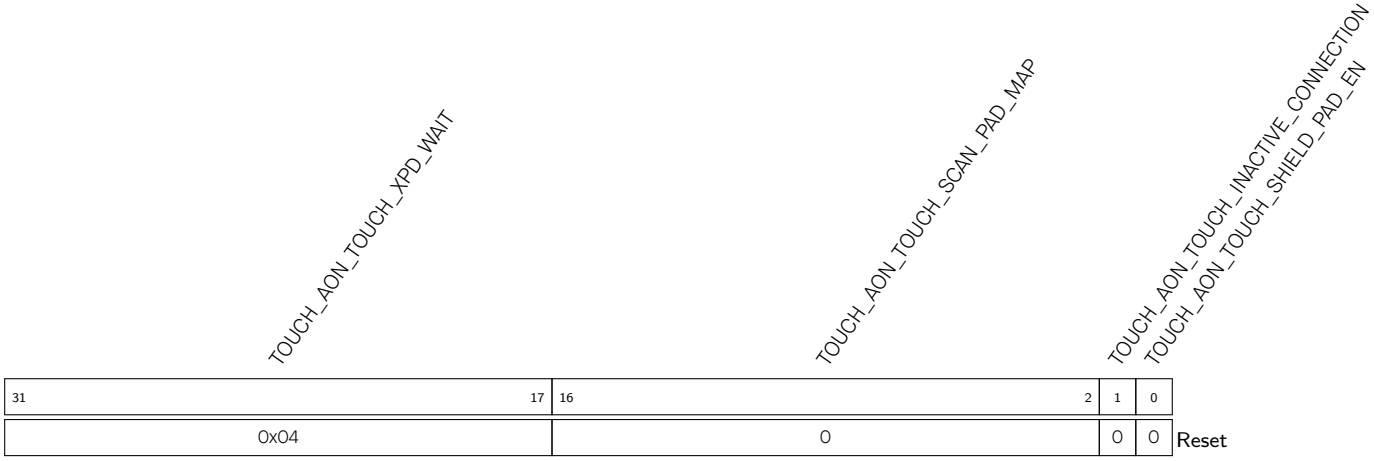


TOUCH_AON_TOUCH_APPROACH_MEAS_NUM2 Configures the number of measurements for sampling frequency mode 2 in proximity mode. (R/W)

TOUCH_AON_TOUCH_APPROACH_MEAS_NUM1 Configures the number of measurements for sampling frequency mode 1 in proximity mode. (R/W)

TOUCH_AON_TOUCH_APPROACH_MEAS_NUM0 Configures the number of measurements for sampling frequency mode 0 in proximity mode. (R/W)

Register 27.13. TOUCH_AON_SCAN_CTRL1_REG (0x0004)



TOUCH_AON_TOUCH_SHIELD_PAD_EN Configures whether to enable the moisture tolerance feature.

0: Disable

1: Enable

(R/W)

TOUCH_AON_TOUCH_INACTIVE_CONNECTION Configures whether to power up the touch pins involved in the measurement in scan mode.

0: Not power up

1: Power up

(R/W)

TOUCH_AON_TOUCH_SCAN_PAD_MAP Configures which touch pins to use in scan mode for measurement. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

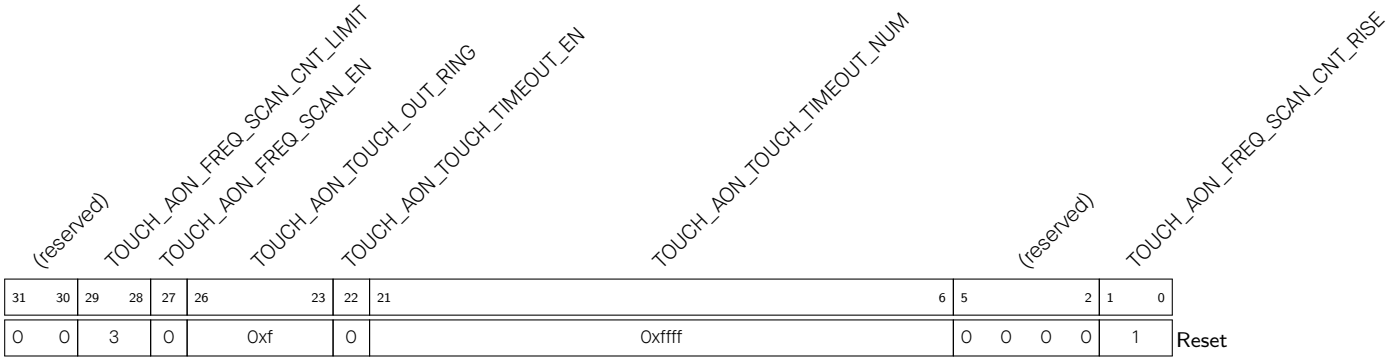
0: No effect

1: Enable the touch pin

(R/W)

TOUCH_AON_TOUCH_XPD_WAIT Configures the power-up wait time after initiating a measurement. (R/W)

Register 27.14. TOUCH_AON_SCAN_CTRL2_REG (0x0008)



TOUCH_AON_FREQ_SCAN_CNT_RISE Configures the number of hit frequency points to be evaluated for touch detection in frequency hopping mode. (R/W)

TOUCH_AON_TOUCH_TIMEOUT_NUM Configures the number of measurement timeouts. (R/W)

TOUCH_AON_TOUCH_TIMEOUT_EN Configures whether to enable the measurement timeout.

0: Disable

1: Enable

(R/W)

TOUCH_AON_TOUCH_OUT_RING Configures which touch pins to be used for water rejection. Configurable values are 0-13. Other values are invalid. (R/W)

TOUCH_AON_FREQ_SCAN_EN Configures whether to enable frequency hopping.

0: Disable

1: Enable

(R/W)

TOUCH_AON_FREQ_SCAN_CNT_LIMIT Configures the number of frequency hops.

0: 1

1: 2

2: 3

3: Invalid

(R/W)

Register 27.15. TOUCH_AON_WORK_REG (0x000C)



TOUCH_AON_DIV_NUM2 Configures whether to enable TOUCH_OUT signal division and sets the division coefficient for sampling frequency mode 2.

bit[0]: Configures whether to enable division.

0: Disable

1: Enable

bit[1-2]: Configures the TOUCH_OUT signal division coefficient.

0: 0

1: 2

2: 4

3: 6

(R/W)

TOUCH_AON_DIV_NUM1 Configures whether to enable TOUCH_OUT signal division and sets the division coefficient for sampling frequency mode 1. For detailed configuration, refer to TOUCH_AON_DIV_NUM2. (R/W)

TOUCH_AON_DIV_NUM0 Configures whether to enable TOUCH_OUT signal division and sets the division coefficient for sampling frequency mode 0. For detailed configuration, refer to TOUCH_AON_DIV_NUM2. (R/W)

TOUCH_AON_TOUCH_OUT_SEL Configures whether to use the TOUCH_OUT signal for clock or data.

0: TOUCH_OUT signal is used as data

1: TOUCH_OUT signal is used as clock

(R/W)

TOUCH_AON_TOUCH_OUT_RESET Write 1 to reset the divided TOUCH_OUT signal. (WT)

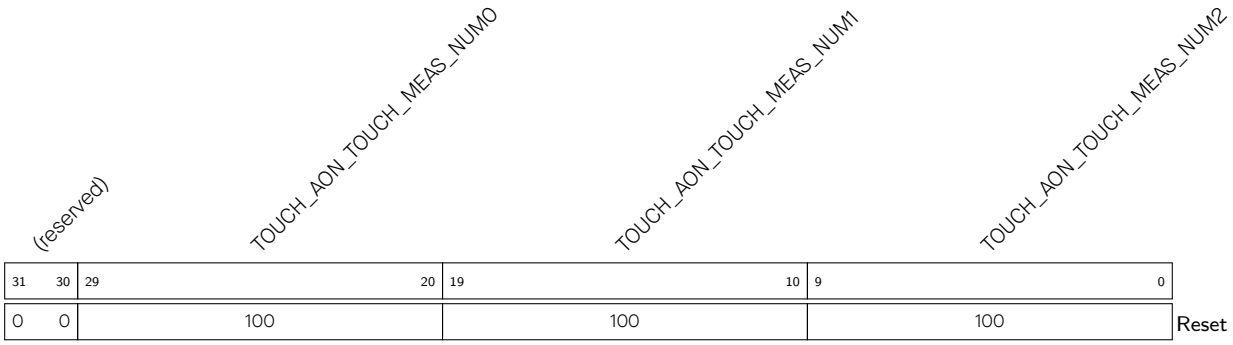
TOUCH_AON_TOUCH_OUT_GATE Configures whether to enable gating after enabling the TOUCH_OUT signal division.

0: Disable

1: Enable

(R/W)

Register 27.16. TOUCH_AON_WORK_MEAS_NUM_REG (0x0010)

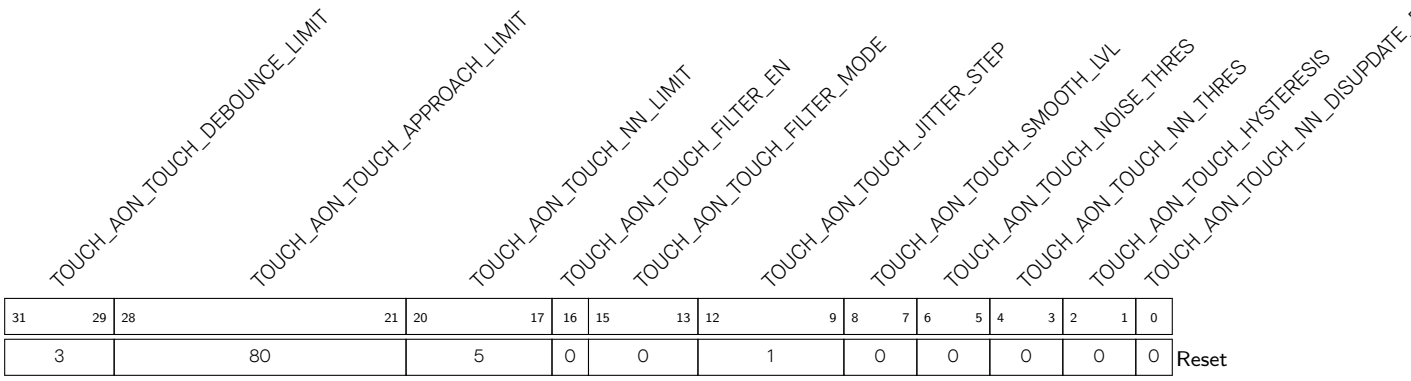


TOUCH_AON_TOUCH_MEAS_NUM2 Configures the number of measurements for sampling frequency mode 2 in normal operation. (R/W)

TOUCH_AON_TOUCH_MEAS_NUM1 Configures the number of measurements for sampling frequency mode 1 in normal operation. (R/W)

TOUCH_AON_TOUCH_MEAS_NUM0 Configures the number of measurements for sampling frequency mode 0 in normal operation. (R/W)

Register 27.17. TOUCH_AON_FILTER1_REG (0x0014)



TOUCH_AON_TOUCH_NN_DISUPDATE_BENCHMARK_EN Configures whether to turn off the benchmark data correction.

0: Do not turn off

1: Turn off

(R/W)

TOUCH_AON_TOUCH_HYSTERESIS Configures the sampling hysteresis. (R/W)

TOUCH_AON_TOUCH_NN_THRES Configures the out-of-phase noise_threshold. (R/W)

TOUCH_AON_TOUCH_NOISE_THRES Configures the noise_threshold. (R/W)

TOUCH_AON_TOUCH_SMOOTH_LVL Configures the IIR filter mode for touch_smooth_data. (R/W)

TOUCH_AON_TOUCH_JITTER_STEP Configures the jitter step parameter for benchmark data filtering. (R/W)

TOUCH_AON_TOUCH_FILTER_MODE Configures the filter mode for generating benchmark data. (R/W)

TOUCH_AON_TOUCH_FILTER_EN Configures whether to enable the filter mode for generating benchmark data.

0: Disable

1: Enable

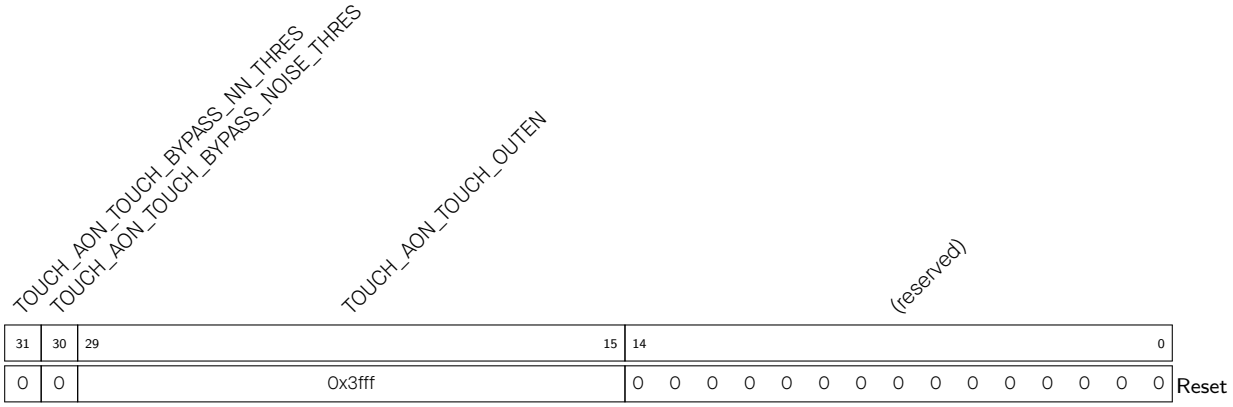
(R/W)

TOUCH_AON_TOUCH_NN_LIMIT Configures the number of consecutive samples exceeding the out-of-phase noise_threshold. (R/W)

TOUCH_AON_TOUCH_APPROACH_LIMIT Configures the number of sampling in proximity mode. (R/W)

TOUCH_AON_TOUCH_DEBOUNCE_LIMIT Configures the number of consecutive sampling for detecting touch state changes. (R/W)

Register 27.18. TOUCH_AON_FILTER2_REG (0x0018)



TOUCH_AON_TOUCH_OUTEN Configures whether to enable output of the 14 touch pin measurement results. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: Disable

1: Enable

(R/W)

TOUCH_AON_TOUCH_BYPASS_NOISE_THRES Configures whether to enable the bypass mode for touch_smooth_data.

0: Disable

1: Enable

(R/W)

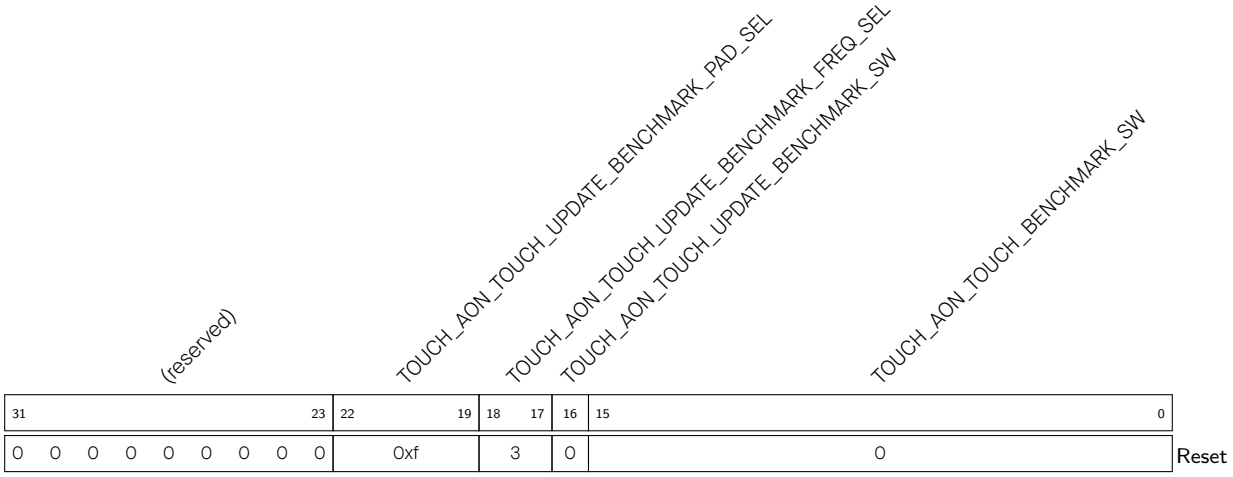
TOUCH_AON_TOUCH_BYPASS_NN_THRES Configures whether to enable the bypass mode for out-of-phase noise_threshold.

0: Disable

1: Enable

(R/W)

Register 27.19. TOUCH_AON_FILTER3_REG (0x001C)



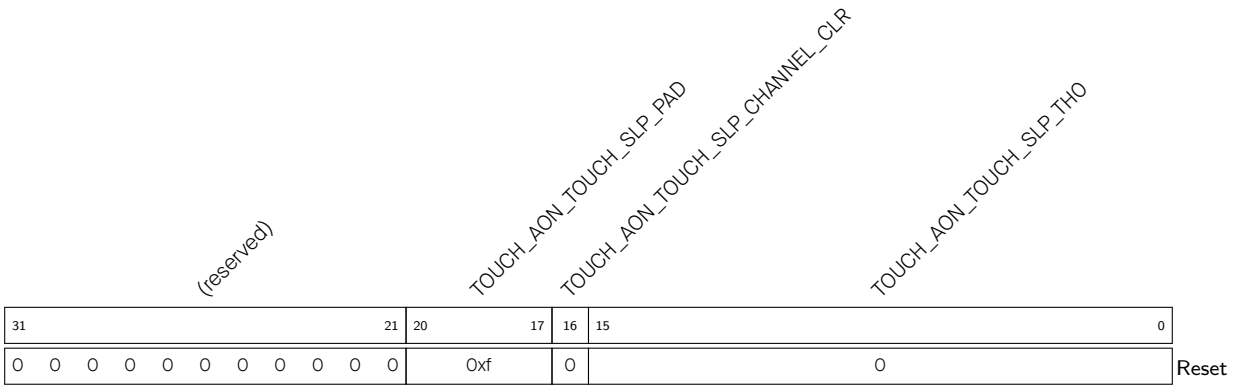
TOUCH_AON_TOUCH_BENCHMARK_SW Configures software to write the benchmark data. (R/W)

TOUCH_AON_TOUCH_UPDATE_BENCHMARK_SW Write 1 to enable the software to write the benchmark data. (WT)

TOUCH_AON_TOUCH_UPDATE_BENCHMARK_FREQ_SEL Configures the frequency point used by software to update the benchmark. (R/W)

TOUCH_AON_TOUCH_UPDATE_BENCHMARK_PAD_SEL Configures the channel used by software to update the benchmark. (R/W)

Register 27.20. TOUCH_AON_SLP0_REG (0x0020)

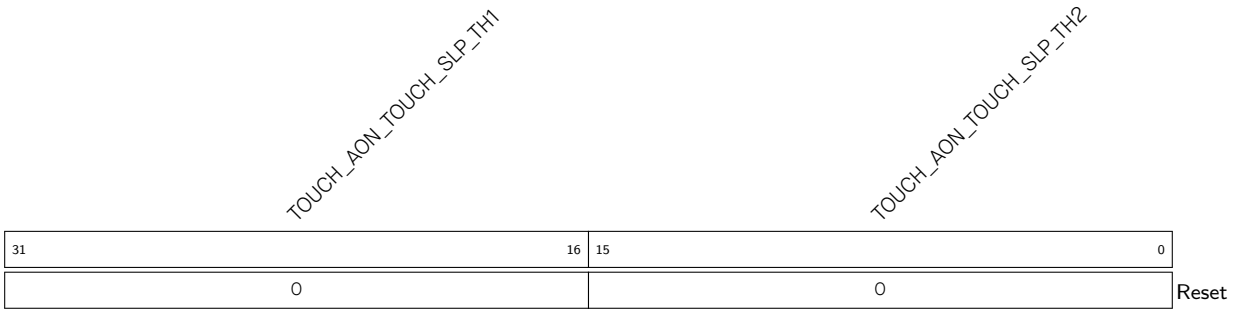


TOUCH_AON_TOUCH_SLP_TH0 Configures the touch threshold for sampling frequency mode 0 in sleep mode. (R/W)

TOUCH_AON_TOUCH_SLP_CHANNEL_CLR Write 1 to clear the benchmark data in sleep mode. (WT)

TOUCH_AON_TOUCH_SLP_PAD Configures the touch sensor number to enter sleep mode. Configurable values are 0-13, corresponding to touch pins 0-13 respectively. Other values are invalid. (R/W)

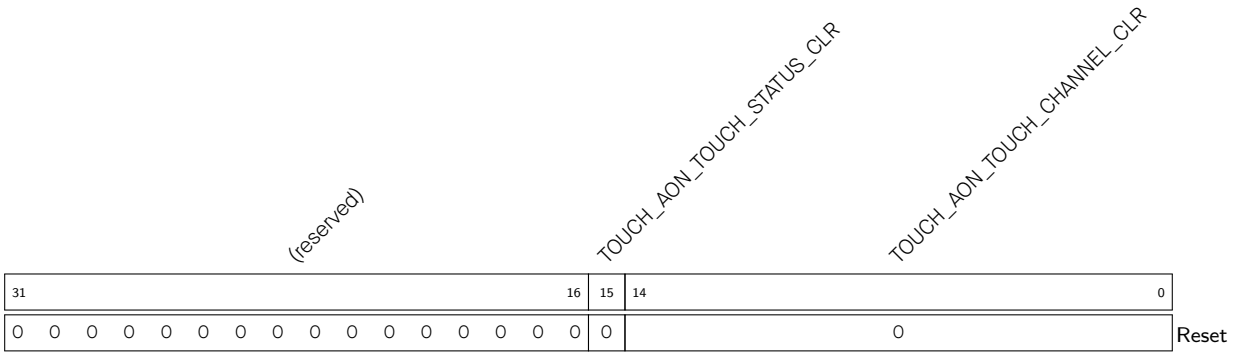
Register 27.21. TOUCH_AON_SLP1_REG (0x0024)



TOUCH_AON_TOUCH_SLP_TH2 Configures the touch threshold for sampling frequency mode 2 in sleep mode. (R/W)

TOUCH_AON_TOUCH_SLP_TH1 Configures the touch threshold for sampling frequency mode 1 in sleep mode. (R/W)

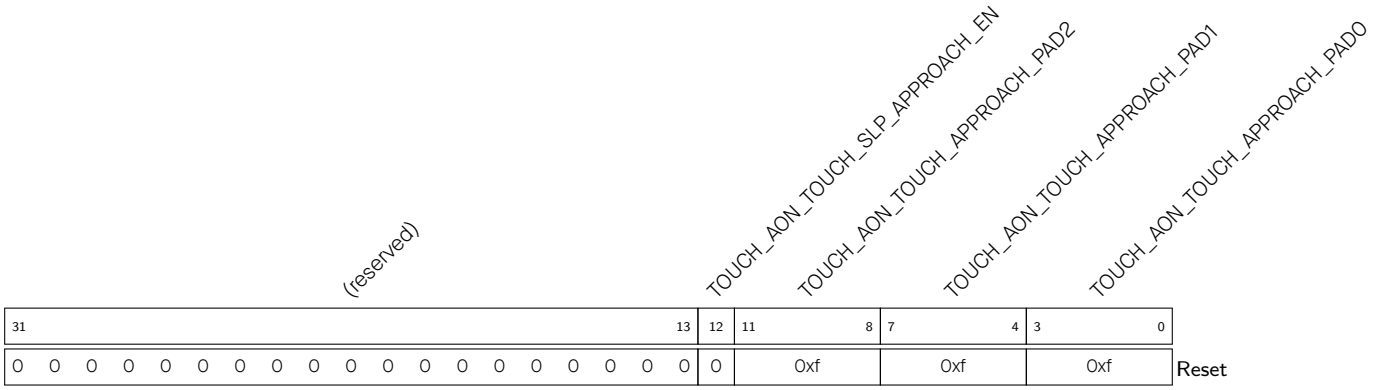
Register 27.22. TOUCH_AON_CLR_REG (0x0028)



TOUCH_AON_TOUCH_CHANNEL_CLR Write 1 to independently clear the touch state of the 14 touch sensors. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid. (WT)

TOUCH_AON_TOUCH_STATUS_CLR Write 1 to clear the touch state. (WT)

Register 27.23. TOUCH_AON_APPROACH_REG (0x002C)



TOUCH_AON_TOUCH_APPROACH_PAD0 Configure the first touch pin used in proximity mode. Configurable values are 0-13. Other values are invalid. (R/W)

TOUCH_AON_TOUCH_APPROACH_PAD1 Configure the second touch pin used in proximity mode. Configurable values are 0-13. Other values are invalid. (R/W)

TOUCH_AON_TOUCH_APPROACH_PAD2 Configure the third touch pin used in proximity mode. Configurable values are 0-13. Other values are invalid. (R/W)

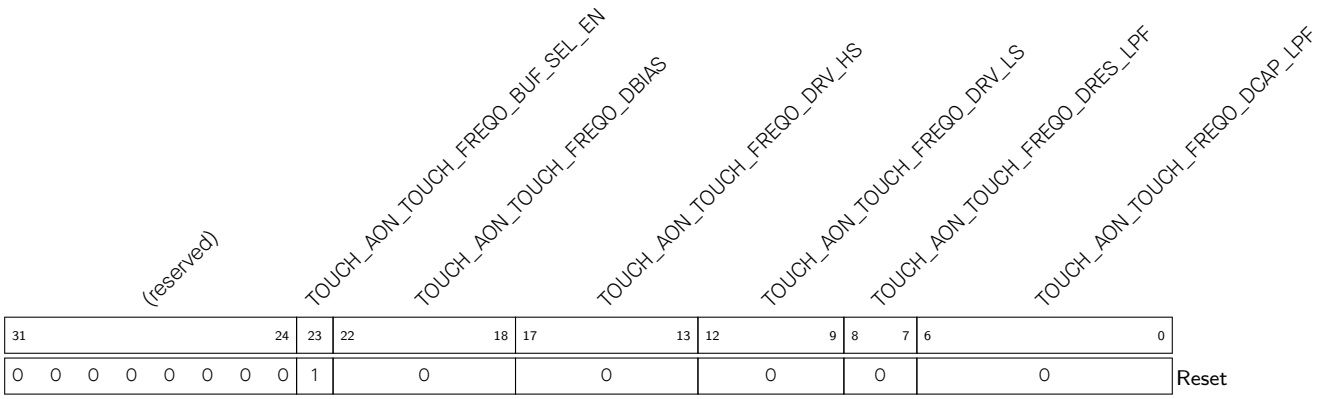
TOUCH_AON_TOUCH_SLP_APPROACH_EN Configures whether to enable proximity mode.

0: Disable

1: Enable

(R/W)

Register 27.24. TOUCH_AON_FREQ0_SCAN_PARA_REG (0x0030)



TOUCH_AON_TOUCH_FREQ0_DCAP_LPF Configures the touch sensor low-pass filter for sampling frequency mode 0. The adjustment range is 0-2.54 pF, and the step is 20 fF. (R/W)

TOUCH_AON_TOUCH_FREQ0_DRES_LPF Configures the touch sensor low-pass filter for sampling frequency mode 0.

0: 0 kΩ

1: 3 kΩ

2: 4.5 kΩ

3: 7.5 kΩ

(R/W)

TOUCH_AON_TOUCH_FREQ0_DRV_LS Configures the value of DRV_LS of the touch sensor for sampling frequency mode 0. (R/W)

TOUCH_AON_TOUCH_FREQ0_DRV_HS Configures the value of DRV_HS of the touch sensor for sampling frequency mode 0. (R/W)

TOUCH_AON_TOUCH_FREQ0_DBIAS Configures whether to enable the moisture tolerance function and configures the internal voltage of the touch sensor for sampling frequency mode 0.

bit[0]: Configures whether to enable the moisture tolerance function for sampling frequency mode 0.

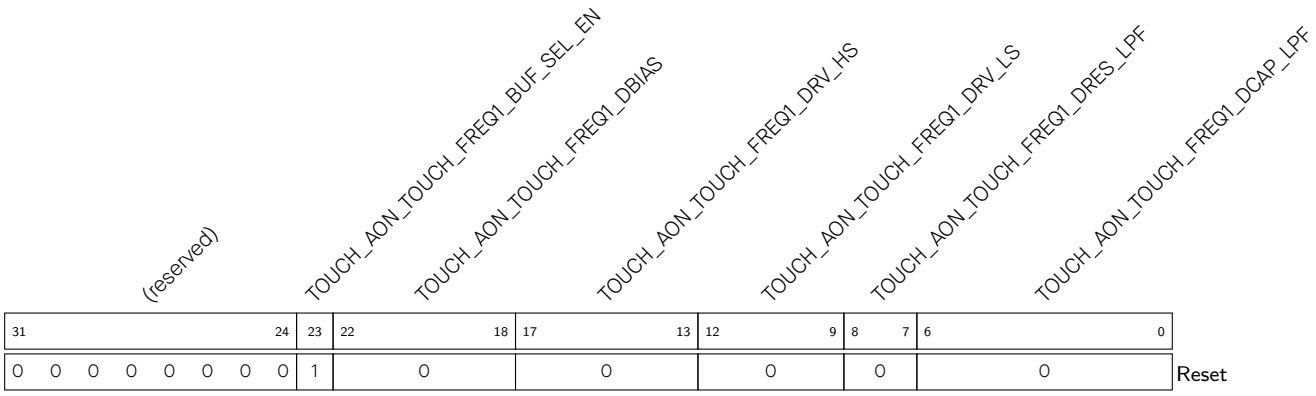
0: Disable

1: Enable

bit[1-4]: Configures the internal voltage of the touch sensor for sampling frequency mode 0. (R/W)

TOUCH_AON_TOUCH_FREQ0_BUF_SEL_EN Configures whether to enable the corresponding BUF_SEL in frequency mode 0. (R/W)

Register 27.25. TOUCH_AON_FREQ1_SCAN_PARA_REG (0x0034)



TOUCH_AON_TOUCH_FREQ1_DCAP_LPF Configures the touch sensor low-pass filter for sampling frequency mode 1. The adjustment range is 0-2.54 pF, and the step is 20 fF. (R/W)

TOUCH_AON_TOUCH_FREQ1_DRES_LPF Configures the touch sensor low-pass filter for sampling frequency mode 1.

0: 0 kΩ

1: 3 kΩ

2: 4.5 kΩ

3: 7.5 kΩ

(R/W)

TOUCH_AON_TOUCH_FREQ1_DRV_LS Configures the value of DRV_LS of the touch sensor for sampling frequency mode 1. (R/W)

TOUCH_AON_TOUCH_FREQ1_DRV_HS Configures the value of DRV_HS of the touch sensor for sampling frequency mode 1. (R/W)

TOUCH_AON_TOUCH_FREQ1_DBIAS Configures whether to enable the moisture tolerance function and configures the internal voltage of the touch sensor for sampling frequency mode 1.

bit[0]: Configures whether to enable the moisture tolerance function for sampling frequency mode 1.

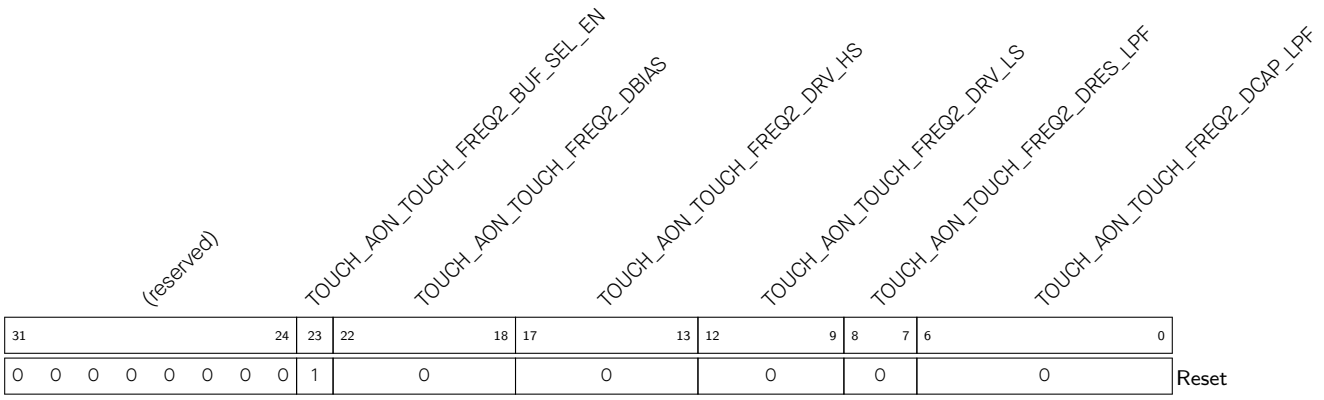
0: Disable

1: Enable

bit[1-4]: Configures the internal voltage of the touch sensor for sampling frequency mode 1. (R/W)

TOUCH_AON_TOUCH_FREQ1_BUF_SEL_EN Configures whether to enable the corresponding BUF_SEL in frequency mode 1. (R/W)

Register 27.26. TOUCH_AON_FREQ2_SCAN_PARA_REG (0x0038)



TOUCH_AON_TOUCH_FREQ2_DCAP_LPF Configures the touch sensor low-pass filter for sampling frequency mode 2. The adjustment range is 0-2.54 pF, and the step is 20 fF. (R/W)

TOUCH_AON_TOUCH_FREQ2_DRES_LPF Configures the touch sensor low-pass filter for sampling frequency mode 2.

0: 0 kΩ

1: 3 kΩ

2: 4.5 kΩ

3: 7.5 kΩ

(R/W)

TOUCH_AON_TOUCH_FREQ2_DRV_LS Configures the value of DRV_LS of the touch sensor for sampling frequency mode 2. (R/W)

TOUCH_AON_TOUCH_FREQ2_DRV_HS Configures the value of DRV_HS of the touch sensor for sampling frequency mode 2. (R/W)

TOUCH_AON_TOUCH_FREQ2_DBIAS Configures whether to enable the moisture tolerance function and configures the internal voltage of the touch sensor for sampling frequency mode 2.

bit[0]: Configures whether to enable the moisture tolerance function for sampling frequency mode 2.

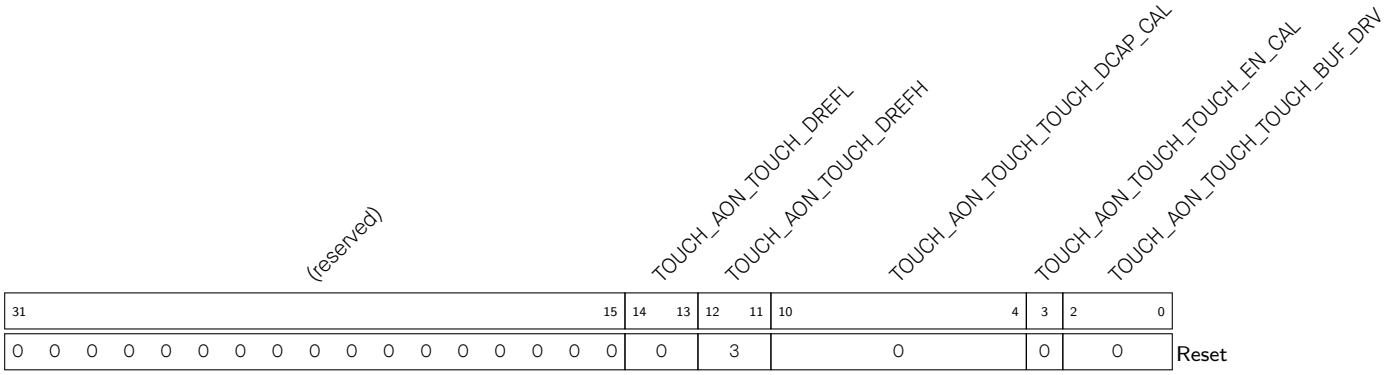
0: Disable

1: Enable

bit[1-4]: Configures the internal voltage of the touch sensor for sampling frequency mode 2. (R/W)

TOUCH_AON_TOUCH_FREQ2_BUF_SEL_EN Configures whether to enable the corresponding BUF_SEL in frequency mode 2. (R/W)

Register 27.27. TOUCH_AON_ANA_PARA_REG (0x003C)



TOUCH_AON_TOUCH_TOUCH_BUF_DRV Configures the value of BUF_DRV. (R/W)

TOUCH_AON_TOUCH_TOUCH_EN_CAL Configures whether to enable the touch sensor internal capacitance test mode.

0: Disable

1: Enable

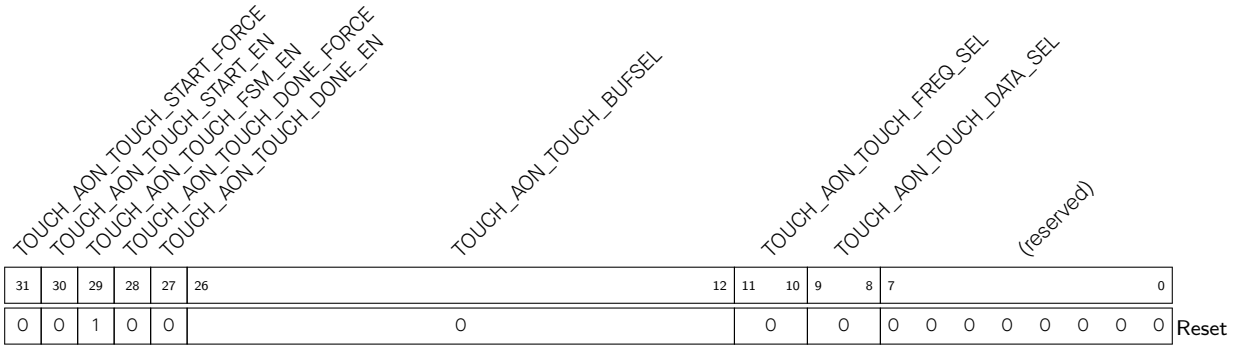
(R/W)

TOUCH_AON_TOUCH_TOUCH_DCAP_CAL Configures the internal capacitance connected to the touch pins. The default value is 0, and the adjustment range is 0-127 pF with a step of 1. (R/W)

TOUCH_AON_TOUCH_DREFH Configures the upper reference voltage of the touch sensor. (R/W)

TOUCH_AON_TOUCH_DREFL Configures the lower reference voltage of the touch sensor. (R/W)

Register 27.28. TOUCH_AON_MUX0_REG (0x0040)



TOUCH_AON_TOUCH_DATA_SEL Configures the type of the return value.

0/1: touch_raw_data

2: benchmark

3: touch_smooth_data

(R/W)

TOUCH_AON_TOUCH_FREQ_SEL Configures the corresponding sampling frequency mode of the return value.

0: 0

1: 1

2: 2

3: Invalid

(R/W)

TOUCH_AON_TOUCH_BUFSEL Configures whether to enable a touch pin to be used for moisture tolerance. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: Disable

1: Enable

(R/W)

TOUCH_AON_TOUCH_DONE_EN Configures whether to enable software to trigger the DONE signal that enables the measurement.

0: Disable

1: Enable

(R/W)

TOUCH_AON_TOUCH_DONE_FORCE Configures whether to generate the DONE signal to end the current measurement operation.

0: Not generate

1: Generate

(R/W)

TOUCH_AON_TOUCH_FSM_EN Configures the control source to power up and activate touch pins.

0: Controlled by software

1: Controlled by hardware

(R/W)

Continued on the next page...

Register 27.28. TOUCH_AON_MUX0_REG (0x0040)

Continued from the previous page...

TOUCH_AON_TOUCH_START_EN Configures whether to enable software to trigger the START signal to start the measurement.

0: Disable

1: Enable

(R/W)

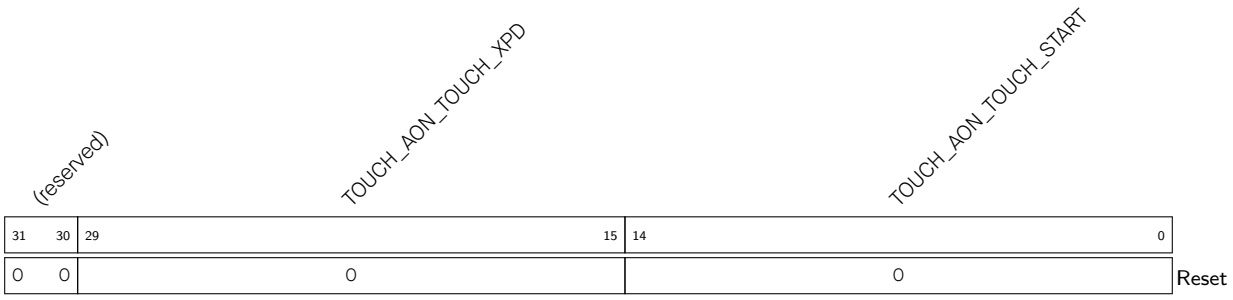
TOUCH_AON_TOUCH_START_FORCE Configures whether to generate the START signal to start a measurement.

0: Not generate

1: Generate

(R/W)

Register 27.29. TOUCH_AON_MUX1_REG (0x0044)



TOUCH_AON_TOUCH_START Configures whether to activate the 14 touch pins. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: No effect

1: Activate

(R/W)

TOUCH_AON_TOUCH_XPD Configures whether to power up the 14 touch pins. Bit 0-13 corresponds to touch pin 0-13 respectively. Other bits are invalid.

0: Not power up

1: Power up

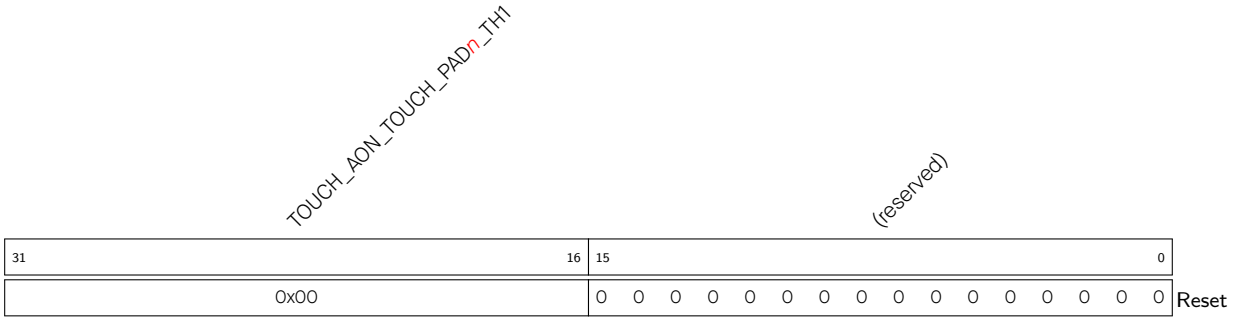
(R/W)

Register 27.30. TOUCH_AON_PADn_TH0_REG (n: 0-13) (0x0048+0xC*n)



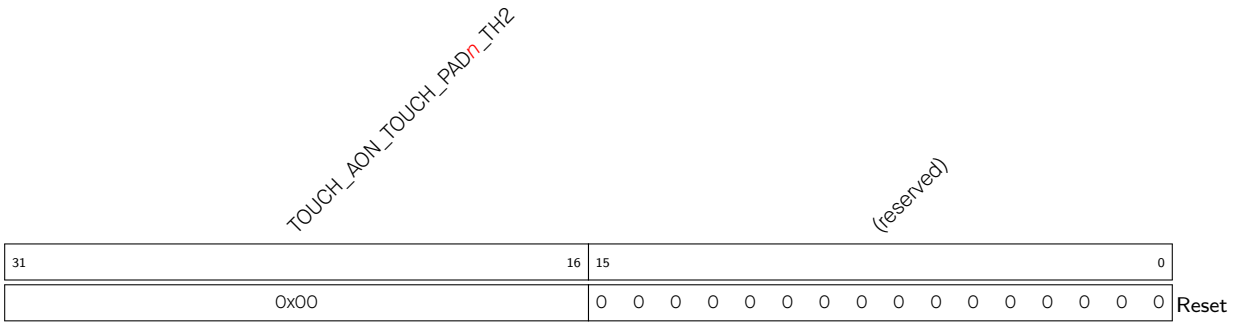
TOUCH_AON_TOUCH_PADn_TH0 Configures the touch threshold for touch sensor *n* for sampling frequency mode 0. (R/W)

Register 27.31. TOUCH_AON_PADn_TH1_REG (n: 0-13) (0x004C+0xC*n)



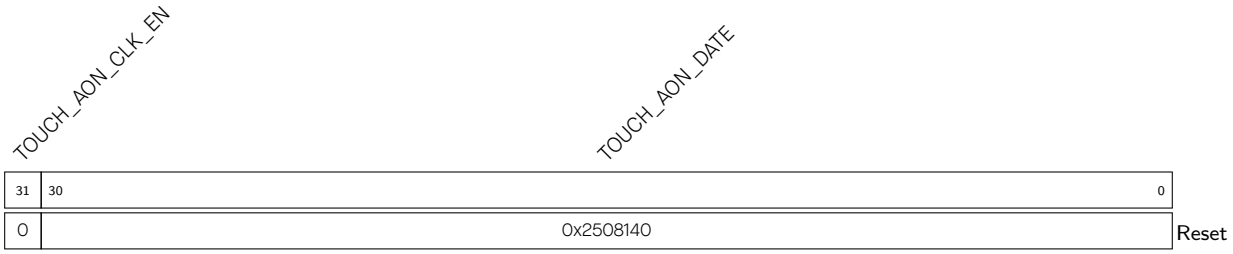
TOUCH_AON_TOUCH_PADn_TH1 Configures the touch threshold of touch sensor *n* for sampling frequency mode 1. (R/W)

Register 27.32. TOUCH_AON_PADn_TH2_REG (n: 0-13) (0x0050+0xC*n)



TOUCH_AON_TOUCH_PADn_TH2 Configures the touch threshold of touch sensor *n* for sampling frequency mode 2. (R/W)

Register 27.33. TOUCH_AON_DATE_REG (0x00FC)



TOUCH_AON_DATE Version control register. (R/W)

TOUCH_AON_CLK_EN Configures whether to enable the clock for accessing the registers.

0: Disable

1: Enable

(R/W)

Chapter 28

SAR ADC Digital Controller (ADC)

28.1 Overview

The ESP32-S31 SAR ADC digital controller is the digital control block of the HP ADC subsystem. It manages two independent SAR ADCs (SAR1 and SAR2), including sampling triggers, timing control, data filtering, threshold monitoring, and DMA transfers. The block exposes register access to the CPU via the APB bus and supports three trigger sources: software trigger, periodic timer scan, and ETM event trigger. Conversion results are transferred to the system bus through an asynchronous FIFO.

The block outputs one aggregated interrupt `adc_reg_int` formed from seven interrupt sources. Software accesses interrupts through the standard RAW / ST / ENA / CLR four-register flow.

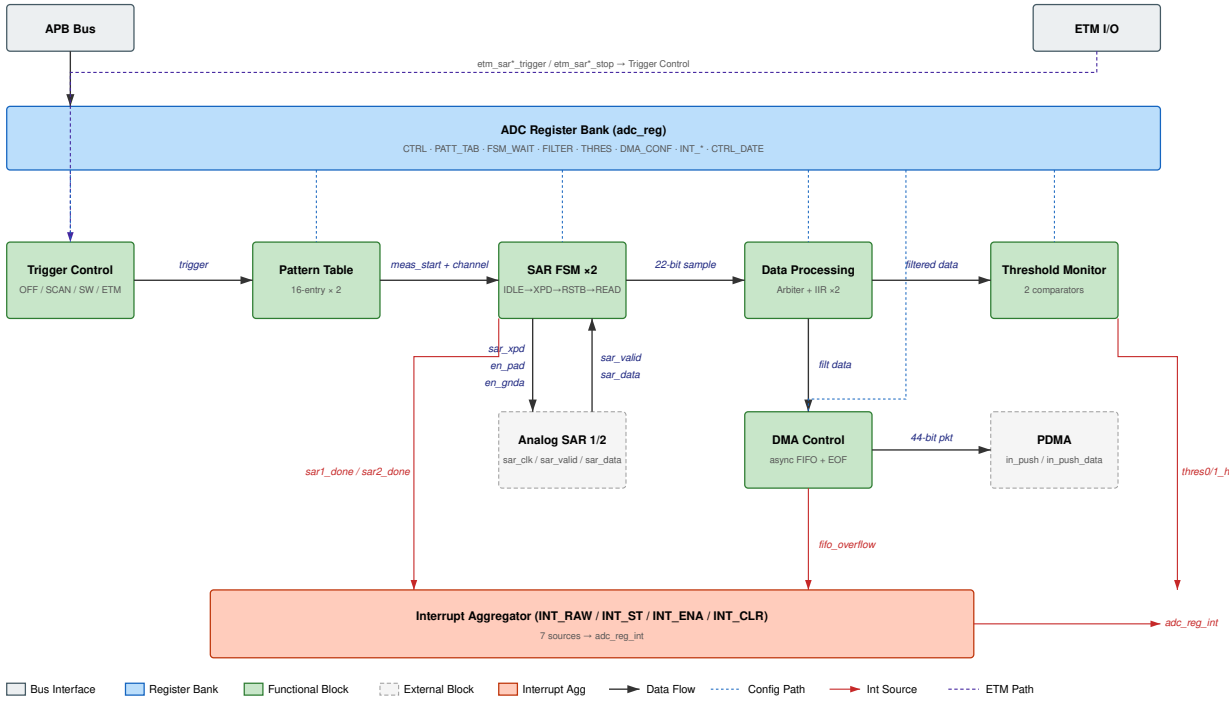


Figure 28.1-1. SAR ADC digital controller block diagram

28.2 Features

The SAR ADC digital controller has the following features:

- Two independent SAR ADCs (SAR1 / SAR2) operating in parallel
- Three trigger modes per SAR: SCAN (timer), SW (register write), ETM

- Per-SAR pattern table with 16 patterns for sequential scanning of up to 16 channels, supporting both single-ended and differential inputs
- Per-SAR four-state SAR control FSM (IDLE → XPD → RSTB → READ) with continuous sampling mode
- Two IIR low-pass filter channels with selectable factor 1 / 2 / 4 / 8 / 16 / 32 / 64
- Two threshold comparators with per-channel monitoring and thres0 all-channel monitoring
- DMA transfer path (asynchronous FIFO + EOF frame management); asynchronous FIFO depth is 4
- Seven aggregated interrupt sources: SAR conversion done, threshold violation, FIFO overflow
- ETM event input/output
- Register clock gating

28.2.1 ADC Channels

The SAR ADC has eight differential channel pairs connected to 16 chip pins. The ADC must first select the analog pin to measure through an internal multiplexer before sampling the analog signal.

Table 28.2-1 lists the chip pins used as ADC channels.

Table 28.2-1. SAR ADC channels

Pin No.	Pin Name	ADC Channel ID
55	GPIO42	ADC1_CHANNEL0_N
56	GPIO43	ADC1_CHANNEL0_P
57	GPIO44	ADC1_CHANNEL1_N
58	GPIO45	ADC1_CHANNEL1_P
59	GPIO46	ADC1_CHANNEL2_N
60	GPIO47	ADC1_CHANNEL2_P
61	GPIO48	ADC1_CHANNEL3_N
62	GPIO49	ADC1_CHANNEL3_P
65	GPIO50	ADC2_CHANNEL0_N
66	GPIO51	ADC2_CHANNEL0_P
67	GPIO52	ADC2_CHANNEL1_N
68	GPIO53	ADC2_CHANNEL1_P
69	GPIO54	ADC2_CHANNEL2_N
70	GPIO55	ADC2_CHANNEL2_P
71	GPIO56	ADC2_CHANNEL3_N
72	GPIO57	ADC2_CHANNEL3_P

28.2.2 ADC Clock

The SAR ADC system clock is ADC_CTRL_CLK. It is the working clock for the DIG ADC FSM and other logic (except the APB interface). It is derived from the source clock divided by LP_PERICLKRST_LP_ADC_DIV_NUM + 1.

The source clock for ADC_CTRL_CLK can be selected from two options: XTAL_CLK or RC_FAST_CLK, via register LP_PERICLKRST_LP_ADC_CLK_SEL.

For more clock information, see Chapter 7 *Reset and Clock [to be added later]*.

28.3 Trigger Control

28.3.1 Features

- SAR1 / SAR2 trigger modes are independently configurable
- Four trigger modes: OFF / SCAN / SW / ETM
- Built-in timer generates SCAN trigger pulses
- Software and ETM can stop conversion at any time and return the FSM to IDLE

28.3.2 Functional Description

28.3.2.1 Trigger Mode Selection

ADC_SAR1_TRIGGER_MODE selects the trigger mode for SAR1:

- 0: OFF — disables triggering; no sampling start pulses are generated
- 1: SCAN — periodic automatic trigger from the internal timer
- 2: SW — one-shot trigger by writing ADC_SAR1_TRIGGER_START = 1
- 3: ETM — triggered by external signal `etm_sar1_trigger`. See Section 28.9 *ETM Events* for details.

SAR2 uses ADC_SAR2_TRIGGER_MODE with the same behavior.

28.3.2.2 Timer Trigger

When the trigger mode is SCAN, the internal timer counts on the ADC working clock:

1. Software writes ADC_TIMER_EN = 1 to enable the timer
2. Software writes ADC_TIMER_TARGET to set the count period
3. When the count reaches the target value, the timer automatically issues one sampling trigger pulse and restarts; sampling period = 2 × ADC_TIMER_TARGET ADC working clocks
4. Writing ADC_TIMER_EN = 0 disables the timer and resets the counter

28.3.2.3 Software Trigger and Stop

- **Start**: write 1 to ADC_SAR1_TRIGGER_START (WT) — in SW mode this immediately produces one trigger pulse; SAR2 uses ADC_SAR2_TRIGGER_START
- **Stop**: write 1 to ADC_SAR1_TRIGGER_STOP (WT) — regardless of the current FSM state, returns to IDLE; SAR2 uses ADC_SAR2_TRIGGER_STOP
- The software stop command is OR'd with ETM `etm_sar1_stop` / `etm_sar2_stop`; either causes the FSM to return to IDLE

28.4 Pattern Table

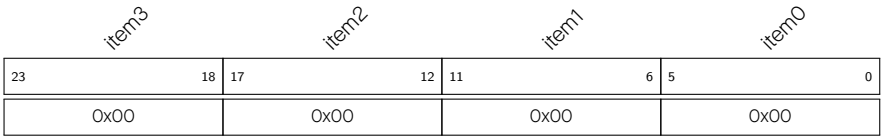
28.4.1 Features

- Per-SAR 16 independent patterns
- Each pattern contains a channel number and single-ended / differential mode
- Cyclic scanning with automatic pointer wrap-around
- Pattern table pointer update timing is configurable (on measurement done / measurement start)

28.4.2 Functional Description

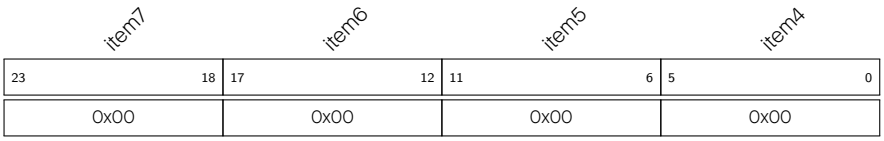
28.4.2.1 Pattern Table Structure

Each SAR holds 16 patterns in four 24-bit registers ADC_SAR1_PATT_TAB1_REG ~ ADC_SAR1_PATT_TAB4_REG. Within each register, four item bit fields are packed from MSB to LSB.



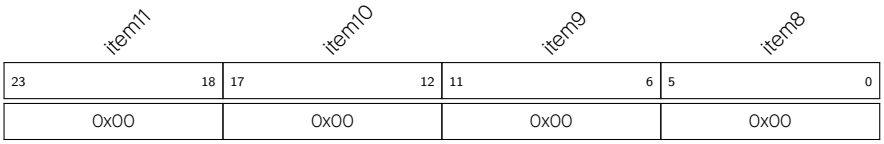
item *x* represents patterns 0–3.

Figure 28.4-1. ADC_SAR1_PATT_TAB1_REG contains patterns 0–3



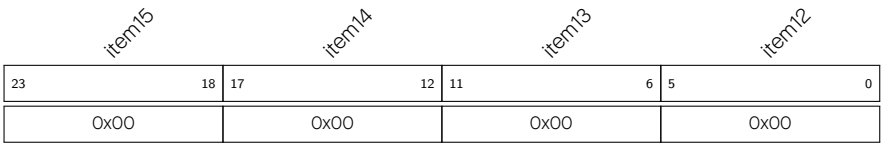
item *x* represents patterns 4–7.

Figure 28.4-2. ADC_SAR1_PATT_TAB2_REG contains patterns 4–7



item *x* represents patterns 8–11.

Figure 28.4-3. ADC_SAR1_PATT_TAB3_REG contains patterns 8–11



item *x* represents patterns 12–15.

Figure 28.4-4. ADC_SAR1_PATT_TAB4_REG contains patterns 12–15

The 16 patterns are arranged as follows: ADC_SAR1_PATT_TAB1_REG holds patterns 0–3, ADC_SAR1_PATT_TAB2_REG holds patterns 4–7, ADC_SAR1_PATT_TAB3_REG holds patterns 8–11, and ADC_SAR1_PATT_TAB4_REG holds patterns 12–15. Within each register, the item bit fields are packed from MSB to LSB ([23:18] is item3, [17:12] is item2, [11:6] is item1, and [5:0] is item0).

Each pattern is 6 bits and contains three subfields: mode, reserved, and channel. Figure 28.4-5 shows the pattern layout; subfield descriptions follow.

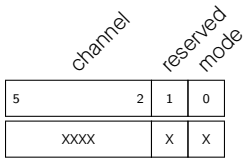


Figure 28.4-5. Pattern layout (6-bit)

mode Configures the sampling mode:
0: single-ended
1: differential
See Section 28.4.2.4 *Channel and Mode Selection*.

reserved Reserved field

channel Configures the sampling channel number:
0: CHANNEL_0_N
1: CHANNEL_0_P
2: CHANNEL_1_N
3: CHANNEL_1_P
4: CHANNEL_2_N
5: CHANNEL_2_P
6: CHANNEL_3_N
7: CHANNEL_3_P
8–15: reserved

SAR2 uses ADC_SAR2_PATT_TAB1_REG ~ ADC_SAR2_PATT_TAB4_REG with the same encoding.

28.4.2.2 Scan Length and Pointer

ADC_SAR1_PATT_LEN (4 bits) sets the pattern table length; values 0–15 correspond to lengths 1–16. The pattern pointer starts at 0, increments by 1 on each update request, and wraps to 0 after reaching ADC_SAR1_PATT_LEN.

Writing ADC_SAR1_PATT_P_CLEAR (WT) = 1 immediately clears the pattern pointer.

28.4.2.3 Update Timing

ADC_SAR1_PATT_TYPE controls when the pattern pointer advances:

- 0: switch to the next pattern after the current sample completes — use when the pattern must advance after every sample
- 1: switch to the next pattern when the next sample starts — use when the pattern must advance only on a timer or ETM trigger

28.4.2.4 Channel and Mode Selection

Based on the mode bit of the current pattern:

- **Single-ended** (mode = 0): enables the single pad selected by channel, with the matching GND reference
- **Differential** (mode = 1): enables the adjacent even/odd pad pair selected by channel[3:1]; GND reference is disabled

28.5 SAR Control FSM

28.5.1 Features

- Four-state FSM: IDLE → XPD → RSTB → READ
- Independent power-up wait and channel reset wait counts
- Continuous sampling mode
- Three-state SAR power control (auto / forced off / forced on)
- Independent FSM instances for SAR1 and SAR2

28.5.2 Functional Description

28.5.2.1 State Machine Flow

- **IDLE**: idle, waiting for a sampling trigger
- **XPD**: enables SAR power and waits for ADC_XPD_WAIT completed SAR conversions
- **RSTB**: pure wait state — waits for ADC_RSTB_WAIT completed SAR conversions, reserving analog settling time
- **READ**: waits for the analog SAR's next result, latches it and returns to IDLE

ADC_XPD_WAIT and ADC_RSTB_WAIT are configured in ADC_FSM_WAIT_REG, 4 bits each.

Once powered, the analog SAR converts continuously on its own, completing one conversion every 24 ADC working clocks (T_{conv}), with no handshake to the FSM. A conversion has two phases: the **first 5 clk are the sampling aperture**, during which the analog input is acquired onto the capacitor array; the following 19 clk are successive approximation, during which the input no longer affects the result. The analog channel must therefore have settled before the aperture opens.

Both wait values are counted in **completed SAR conversions**, so one count costs a whole T_{conv}. A value of 0 is the exception: the state then lasts a single ADC working clock, which means changing ADC_RSTB_WAIT from 0 to 1 jumps by a full T_{conv}. ADC_RSTB_WAIT is recommended to be set to 2.

28.5.2.2 Continuous Sampling Mode

With ADC_SAR1_CONTINUE_MODE_EN = 1 the FSM does not return to IDLE after READ: a new trigger sends it to RSTB, otherwise it **stays in READ**.

While parked in READ the FSM keeps sampling: every 24 clk it emits a sample, refreshes ADC_APB_SARADC1_DATA, raises one ADC_SAR1_DONE_INT interrupt, one DMA push and one ETM event,

and advances the pattern pointer by one entry. Continuous mode is therefore used by **triggering once and letting it free-run** — after start-up it streams one sample every 24 clk, cycling through the pattern table. SAR2 uses ADC_SAR2_CONTINUE_MODE_EN.

Re-triggering sends the FSM back to RSTB, which produces no samples. So a trigger in continuous mode does not request a sample — it **inserts an RSTB into the sample stream**.

Sampling cost:

- **Start-up latency** (trigger → first sample) ≈ (ADC_XPD_WAIT + ADC_RSTB_WAIT + 1) × 24 clk; about 264 clk with ADC_XPD_WAIT = 8 and ADC_RSTB_WAIT = 2
- **Steady-state sample period** = 24 clk, a throughput of 1 / 24 clk, independent of both wait values — they are traversed only once, at start-up

The three figures below compress T_{conv} to scale so that clk can be drawn, and use ADC_XPD_WAIT = 2, ADC_RSTB_WAIT = 1 (0 in scenario 2). See Figures 28.5-1, 28.5-2 and 28.5-3.

Scenario 1: free-running multi-channel scan (ADC_SAR1_PATT_TYPE = 0, advance on done) — the pointer advances once per sample, so the channel rolls.

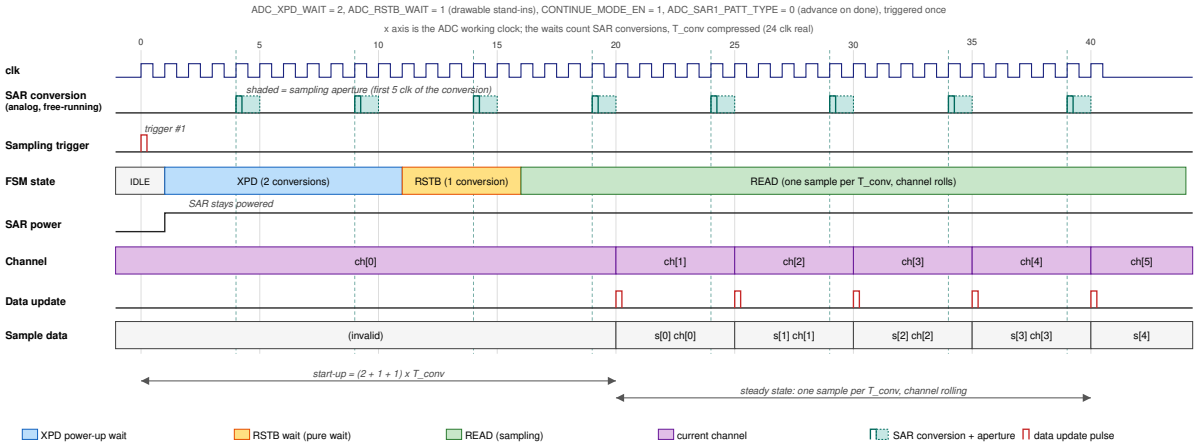


Figure 28.5-1. Continuous sampling with channel switching

Scenario 2: channel holds (ADC_SAR1_PATT_TYPE = 1, advance on start) — the pointer only advances on a trigger, and free-running has none, so the channel never moves.

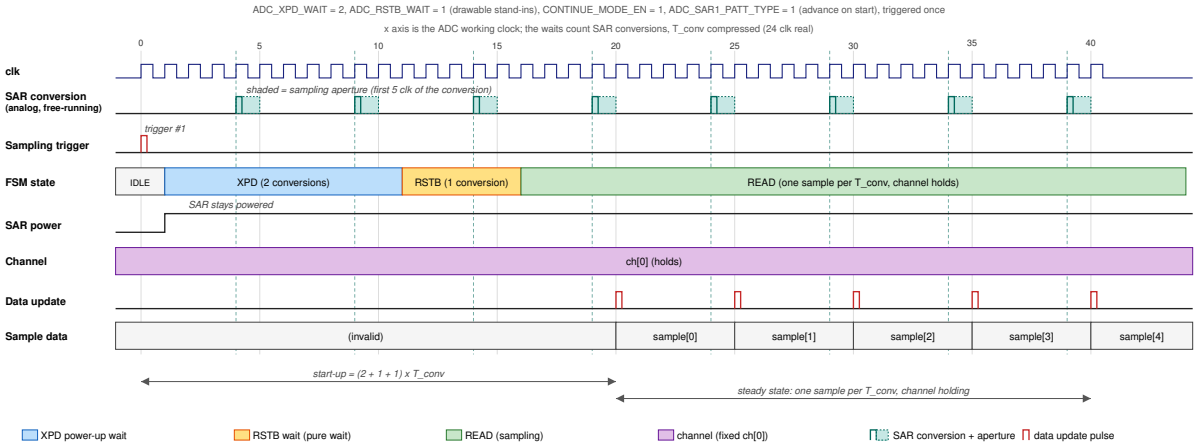


Figure 28.5-2. Continuous sampling with PATT_TYPE = 1 (channel holds)

Scenario 3: non-continuous mode (ADC_SAR1_CONTINUE_MODE_EN = 0), software triggered — each trigger runs IDLE → XPD → RSTB → READ and back to IDLE, with the SAR powered down between samples.

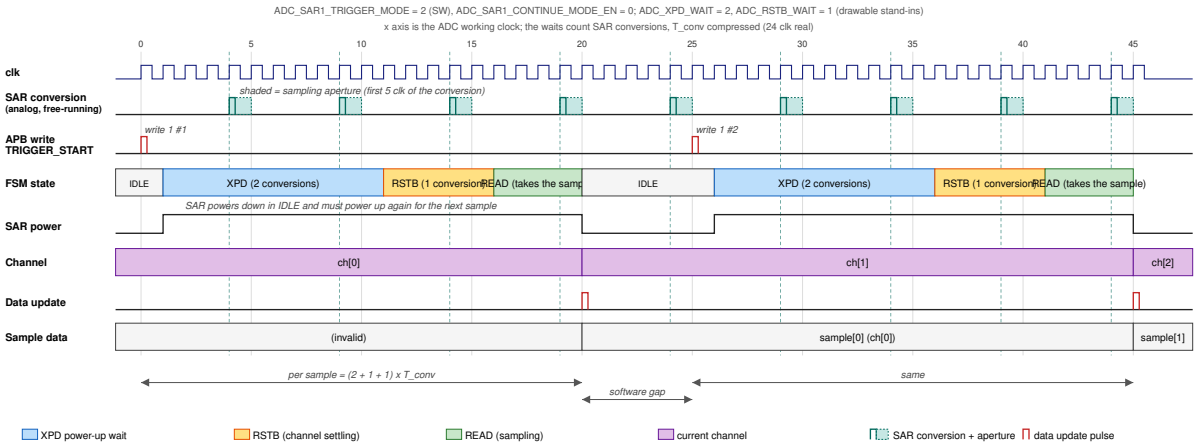


Figure 28.5-3. Non-continuous mode, software-triggered sampling

Where the channel switch lands: the pattern pointer advances as the sample is emitted and the channel switches 1–2 clk later, while the sampling aperture is the first 5 clk of each conversion. The switch therefore lands inside the aperture: its first 1–2 clk are still on the previous channel and the remaining 3–4 clk on the new one. Whether a free-running multi-channel sample is valid depends on the analog mux settling within those 3–4 clk.

When triggered once and left free-running, ADC_RSTB_WAIT is traversed only once, so the settling time it provides benefits only the first sample. **A multi-channel scan should instead be triggered once per sample**: each trigger routes the FSM through RSTB, giving the just-switched channel ADC_RSTB_WAIT × 24 clk to settle, after which the sample's aperture sits wholly on the new channel. When longer settling is required, use ADC_SAR1_CONTINUE_MODE_EN = 0 instead (scenario 3).

Choosing between the three usages:

Usage	Sample period	Channel settling	Use for
Continuous, single trigger	24 clk	≈ 3–4 clk in aperture	High-rate sampling with a fixed channel
Continuous, periodic trigger	(RSTB_WAIT + 1) × 24 clk, 72 clk at the recommended 2	RSTB_WAIT × 24 clk, ≈ 46 clk at the recommended 2	**Multi-channel scans**
CONTINUE_MODE_EN = 0	(XPD_WAIT + RSTB_WAIT + 1) × 24 clk, 264 clk at the recommended 8 / 2	(XPD_WAIT + RSTB_WAIT) × 24 clk, 240 clk at the recommended 8 / 2	When longer settling is required

In SCAN mode the timer period should be no shorter than the sample period above (trigger period = 2 × ADC_TIMER_TARGET ADC working clocks).

28.5.2.3 SAR Power Control

ADC_XPD_SAR1_FORCE[1:0] controls SAR1 power:

- 0–1: auto control — on when not IDLE, off in IDLE
- 2: force SAR power off
- 3: force SAR power on

SAR2 is controlled independently by ADC_XPD_SAR2_FORCE.

28.5.2.4 State Query

Software can read ADC_SAR1_STATE (RO) or ADC_SAR2_STATE (RO) to query the FSM state: 0=IDLE, 1=XPD, 2=RSTB, 3=READ.

28.6 Data Processing

28.6.1 Features

- SAR1 / SAR2 data arbitration
- Two independent IIR low-pass filter channels
- Filter factor is a power of 2 (1–64)
- Samples that do not match a filter channel pass through directly

28.6.2 Functional Description

28.6.2.1 Data Arbitration

When SAR1 and SAR2 conversion-done signals arrive, the data-processing block selects data as follows:

- Only SAR1 data arrives: process SAR1 data

- Only SAR2 data arrives: process SAR2 data
- Both arrive simultaneously: process SAR1 first, then SAR2 (results enter the next stage in order)

28.6.2.2 IIR Filter

The block provides two independent IIR filter channels. ADC_FILTER_CHANNEL0 and ADC_FILTER_CHANNEL1 each select the channel for one filter.

The filter factor is configured by ADC_FILTER_FACTOR0 / ADC_FILTER_FACTOR1 (3 bits):

- 0: k=1 (no filtering, pass-through)
- 1: k=2
- 2: k=4
- 3: k=8
- 4: k=16
- 5: k=32
- 6: k=64

The first-order IIR algorithm is:

$$sum_n = sum_{n-1} - \frac{sum_{n-1}}{k} + adc\_din_n$$

$$filtered\_data_n = \frac{sum_n}{k}$$

where `sum` is the internal integrator, `adc_din` is the raw ADC sample input, and `filtered_data` is the filtered ADC output.

Samples whose channel does not match a filter channel are not filtered and pass through directly. Writing ADC_FILTER_RESET (WT) = 1 clears the internal state of both filters.

28.6.2.3 Reading Latest Sample

The most recent conversion result can be read from ADC_APB_SARADC1_DATA (RO, 22 bits) and ADC_APB_SARADC2_DATA (RO, 22 bits): [21:17] is the channel number, [16:0] is the 17-bit ADC conversion result.

28.7 Threshold Monitor

28.7.1 Features

- Two independent threshold comparators: threshold 0 (thres0) and threshold 1 (thres1)
- Each comparator has its own monitored channel, upper threshold, and lower threshold
- thres0 supports all-channel monitoring mode
- A threshold violation sets the corresponding interrupt

28.7.2 Functional Description

28.7.2.1 Threshold Configuration

Each threshold comparator is configured through the following fields:

- Enable: ADC_THRESO_EN (write 1 to enable)
- Channel: ADC_THRESO_CHANNEL (5 bits)
- Upper threshold: ADC_THRESO_HIGH (17 bits)
- Lower threshold: ADC_THRESO_LOW (17 bits)

thres1 uses ADC_THRES1_EN / ADC_THRES1_CHANNEL / ADC_THRES1_HIGH / ADC_THRES1_LOW.

28.7.2.2 Comparison Logic

When a filtered sample arrives, if the channel number matches and the comparator is enabled:

- If adc_din $>$ ADC_THRESO_HIGH: raise a THRESO_HIGH interrupt pulse
- If adc_din $<$ ADC_THRESO_LOW: raise a THRESO_LOW interrupt pulse

Each violation sets the interrupt once; after software clears it by writing ADC_INT_CLR_REG, subsequent violations can trigger again.

28.7.2.3 All-Channel Monitoring Mode

When ADC_THRES_ALL_EN = 1, thres0 is no longer limited to the channel selected by ADC_THRESO_CHANNEL; it monitors samples from all channels. This applies to thres0 only; thres1 always monitors its configured channel.

28.8 DMA Control

28.8.1 Features

- Asynchronous FIFO bridges the ADC clock domain → DMA clock domain
- EOF frame markers ease software framing
- FIFO depth 4 with overflow protection and interrupt

28.8.2 Functional Description

28.8.2.1 DMA Enable

Setting ADC_APB_ADC_TRANS = 1 enables the DMA transfer path. While enabled, every sample from the data-processing block is written to the asynchronous FIFO and transferred to system memory by the DMA controller.

28.8.2.2 EOF Frame Management

ADC_APB_ADC_EOF_NUM (16 bits) sets the number of samples per frame. The DMA controller maintains an EOF counter:

1. Increment by 1 for every transferred sample
2. When the count reaches ADC_APB_ADC_EOF_NUM – 1, insert an EOF marker into the DMA data and clear the counter
3. Software can clear the counter by writing ADC_DMA_EOF_NUM_CLR (WT) = 1

28.8.2.3 DMA Data Format

From a software perspective, each DMA transfer delivers one 32-bit sample word (4 bytes), organized as:

- byte 3: reserved
- byte 2: `channel`+ `data_msb` — sample channel number + ADC sample MSB
- byte 1: `data_high` — ADC sample data, high 8 bits
- byte 0: `data_low` — ADC sample data, low 8 bits

The full ADC sample is the concatenation of `data_msb`, `data_high`, and `data_low`: `sample = (data_msb << 16) | (data_high << 8) | data_low`. Software reads DMA data 4 bytes at a time; internal alignment and packing are handled by the DMA controller.

28.8.2.4 FIFO Overflow

If the asynchronous FIFO is full and another sample is written, an ADC_FIFO_OVERFLOW interrupt is raised once. The interrupt fires only when the FIFO enters the overflow state; sustained overflow does not retrigger. After software clears the interrupt and drains the FIFO, a new overflow can trigger again.

28.9 ETM Events

28.9.1 Features

- ETM trigger inputs can start or stop ADC conversion
- Multiple event outputs for other system blocks
- Dedicated ETM reference clock output

28.9.2 Functional Description

28.9.2.1 ETM Trigger Inputs

- `etm_sar1_trigger` — SAR1 ETM start (when ADC_SAR1_TRIGGER_MODE = 3)
- `etm_sar2_trigger` — SAR2 ETM start (when ADC_SAR2_TRIGGER_MODE = 3)
- `etm_sar1_stop` — SAR1 ETM stop, OR'd with software ADC_SAR1_TRIGGER_STOP

- `etm_sar2_stop` — SAR2 ETM stop, OR'd with software ADC_SAR2_TRIGGER_STOP

28.9.2.2 ETM Event Outputs

- `etm_sar1_done` / `etm_sar2_done` — single conversion done
- `etm_event_sar1_start` / `etm_event_sar2_start` — sampling start (FSM enters READ from RSTB)
- `etm_event_thres0_high_int_etm` / `etm_event_thres1_high_int_etm` — threshold high violation
- `etm_event_thres0_low_int_etm` / `etm_event_thres1_low_int_etm` — threshold low violation

28.9.2.3 ETM Clock

`clk_adc_etm` outputs the ADC working clock as the ETM subsystem reference clock.

28.10 Interrupts

The SAR ADC block provides seven interrupt sources, aggregated into `adc_reg_int`:

Table 28.10-1. SAR ADC interrupt sources

Source	Description
ADC_SAR1_DONE_INT	SAR1 single conversion done
ADC_SAR2_DONE_INT	SAR2 single conversion done
ADC_THRES0_HIGH_INT	The filtered data is above the high threshold ADC_THRES0_HIGH
ADC_THRES1_HIGH_INT	The filtered data is above the high threshold ADC_THRES1_HIGH
ADC_THRES0_LOW_INT	The filtered data is below the low threshold ADC_THRES0_LOW
ADC_THRES1_LOW_INT	The filtered data is below the low threshold ADC_THRES1_LOW
ADC_FIFO_OVERFLOW_INT	DMA FIFO overflow

Interrupt register group: ADC_INT_ENA_REG (0x064), ADC_INT_RAW_REG (0x068), ADC_INT_ST_REG (0x06C), ADC_INT_CLR_REG (0x070).

Interrupt service flow:

1. After a hardware event, the corresponding bit in ADC_INT_RAW_REG is set automatically
2. ADC_INT_ST_REG = ADC_INT_RAW_REG & ADC_INT_ENA_REG
3. `adc_reg_int` is asserted when any bit of ADC_INT_ST_REG is non-zero
4. The CPU reads ADC_INT_ST_REG to identify the source
5. After service, software writes 1 to the corresponding bit of ADC_INT_CLR_REG

6. ADC_INT_RAW_REG does not self-clear; software must write ADC_INT_CLR_REG to clear

28.11 Register Summary

The addresses in this section are relative to the base address of the SAR ADC Controller in Table 3 in Chapter 3 *System and Memory [to be added later]*.

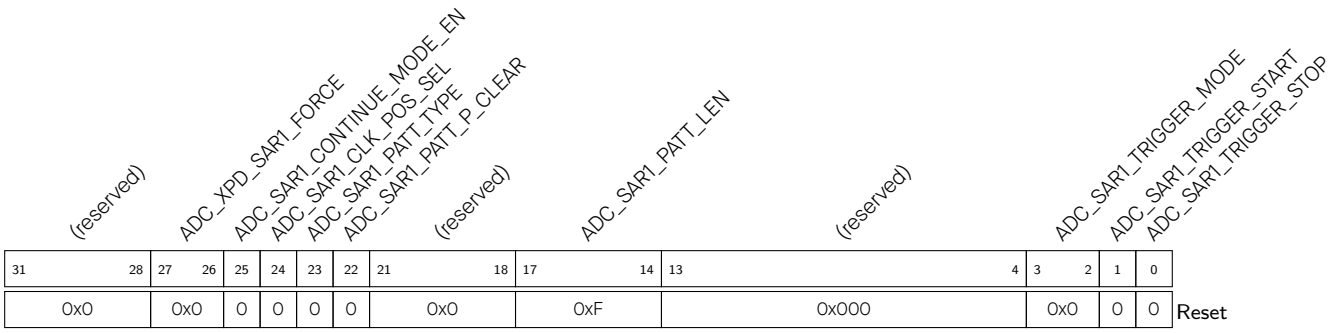
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
Configuration Registers			
ADC_CTRL0_REG	SAR1 control register	0x000	R/W
ADC_CTRL1_REG	SAR2 control register	0x004	R/W
ADC_CTRL2_REG	Timer configuration register	0x008	R/W
ADC_FILTER_CTRL1_REG	Filter factor configuration register	0x00C	R/W
ADC_FSM_WAIT_REG	FSM wait time configuration register	0x010	R/W
ADC_SAR1_STATUS_REG	SAR1 status register	0x014	RO
ADC_SAR2_STATUS_REG	SAR2 status register	0x018	RO
ADC_SAR1_PATT_TAB1_REG	SAR1 pattern table 0–3 register	0x01C	R/W
ADC_SAR1_PATT_TAB2_REG	SAR1 pattern table 4–7 register	0x020	R/W
ADC_SAR1_PATT_TAB3_REG	SAR1 pattern table 8–11 register	0x024	R/W
ADC_SAR1_PATT_TAB4_REG	SAR1 pattern table 12–15 register	0x028	R/W
ADC_SAR2_PATT_TAB1_REG	SAR2 pattern table 0–3 register	0x02C	R/W
ADC_SAR2_PATT_TAB2_REG	SAR2 pattern table 4–7 register	0x030	R/W
ADC_SAR2_PATT_TAB3_REG	SAR2 pattern table 8–11 register	0x034	R/W
ADC_SAR2_PATT_TAB4_REG	SAR2 pattern table 12–15 register	0x038	R/W
ADC_FILTER_CTRL0_REG	Filter control register	0x03C	R/W
ADC_SAR1_DATA_STATUS_REG	SAR1 conversion data status register	0x040	RO
ADC_SAR2_DATA_STATUS_REG	SAR2 conversion data status register	0x044	RO
ADC_THRES0_HIGH_THRES_REG	Threshold 0 upper limit register	0x048	R/W
ADC_THRES0_LOW_THRES_REG	Threshold 0 lower limit register	0x04C	R/W
ADC_THRES0_CTRL_REG	Threshold 0 control register	0x050	R/W
ADC_THRES1_HIGH_THRES_REG	Threshold 1 upper limit register	0x054	R/W
ADC_THRES1_LOW_THRES_REG	Threshold 1 lower limit register	0x058	R/W
ADC_THRES1_CTRL_REG	Threshold 1 control register	0x05C	R/W
ADC_THRES_CTRL_REG	Threshold 0 all-channel monitoring register	0x060	R/W
ADC_INT_ENA_REG	Interrupt enable register	0x064	R/W
ADC_INT_RAW_REG	Raw interrupt status register	0x068	R/WTC/SS
ADC_INT_ST_REG	Masked interrupt status register	0x06C	RO
ADC_INT_CLR_REG	Interrupt clear register	0x070	WT
ADC_DMA_CONF_REG	DMA configuration register	0x074	R/W
ADC_CTRL_DATE_REG	Clock gating and version control register	0x3FC	R/W

28.12 Registers

The addresses in this section are relative to the base address of the SAR ADC Controller in Table 3 in Chapter 3 *System and Memory [to be added later]*.

Register 28.1. ADC_CTRL0_REG (0x000)



ADC_XPD_SAR1_FORCE Configures SAR1 power force control.

0–1: Auto control (on when not IDLE, off in IDLE)

2: Force off

3: Force on

(R/W)

ADC_SAR1_CONTINUE_MODE_EN Configures whether to enable continuous sampling mode for SAR1.

0: Disable

1: Enable

(R/W)

ADC_SAR1_CLK_POS_SEL Configures the sample data clock phase for SAR1.

0: Clock falling edge

1: Clock rising edge

(R/W)

ADC_SAR1_PATT_TYPE Configures when the SAR1 pattern pointer updates.

0: Update when measurement completes

1: Update when measurement starts

(R/W)

ADC_SAR1_PATT_P_CLEAR Write 1 to clear the SAR1 pattern pointer. (WT)

ADC_SAR1_PATT_LEN Configures the SAR1 pattern table length.

0–15: Lengths 1–16

(R/W)

ADC_SAR1_TRIGGER_MODE Configures the trigger mode for SAR1.

0: OFF

1: SCAN

2: SW

3: ETM

(R/W)

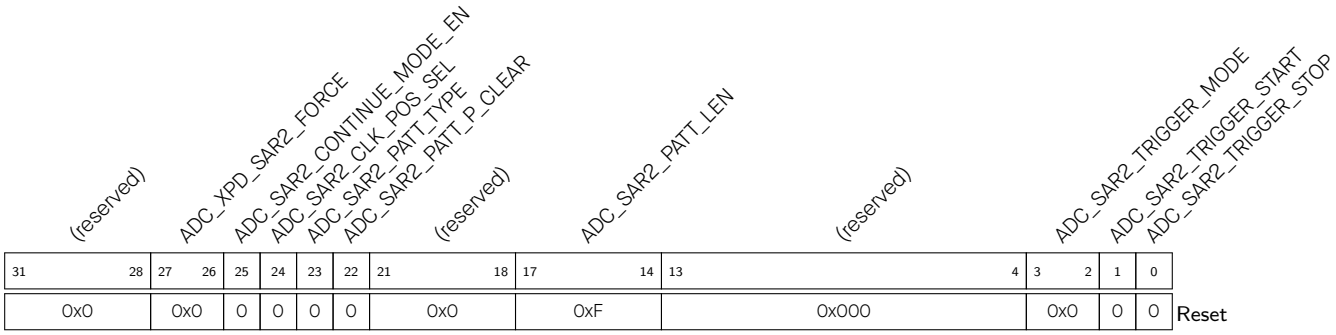
ADC_SAR1_TRIGGER_START Write 1 to trigger one SAR1 conversion.

Valid only when ADC_SAR1_TRIGGER_MODE is 2.

(WT)

ADC_SAR1_TRIGGER_STOP Write 1 to stop SAR1 conversion and return the FSM to IDLE. (WT)

Register 28.2. ADC_CTRL1_REG (0x004)



ADC_XPD_SAR2_FORCE Configures SAR2 power force control.

0–1: Auto control (on when not IDLE, off in IDLE)

2: Force off

3: Force on

(R/W)

ADC_SAR2_CONTINUE_MODE_EN Configures whether to enable continuous sampling mode for SAR2.

0: Disable

1: Enable

(R/W)

ADC_SAR2_CLK_POS_SEL Configures the sample data clock phase for SAR2.

0: Clock falling edge

1: Clock rising edge

(R/W)

ADC_SAR2_PATT_TYPE Configures when the SAR2 pattern pointer updates.

0: Update when measurement completes

1: Update when measurement starts

(R/W)

ADC_SAR2_PATT_P_CLEAR Write 1 to clear the SAR2 pattern pointer. (WT)

ADC_SAR2_PATT_LEN Configures the SAR2 pattern table length.

0–15: Lengths 1–16

(R/W)

ADC_SAR2_TRIGGER_MODE Configures the trigger mode for SAR2.

0: OFF

1: SCAN

2: SW

3: ETM

(R/W)

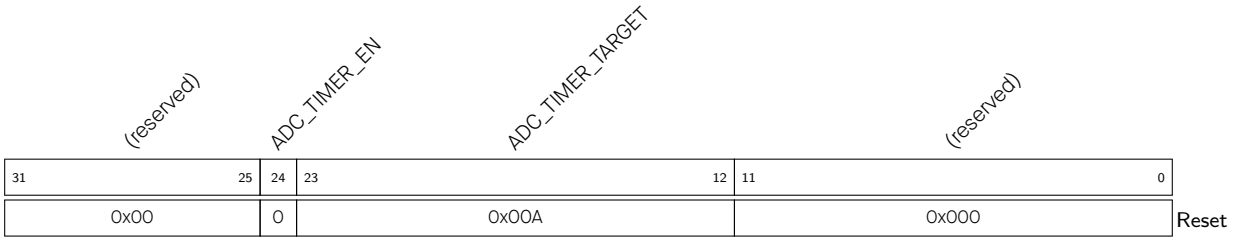
ADC_SAR2_TRIGGER_START Write 1 to trigger one SAR2 conversion.

Valid only when ADC_SAR2_TRIGGER_MODE is 2.

(WT)

ADC_SAR2_TRIGGER_STOP Write 1 to stop SAR2 conversion and return the FSM to IDLE. (WT)

Register 28.3. ADC_CTRL2_REG (0x008)



ADC_TIMER_EN Configures whether to enable the timer.

0: Disable

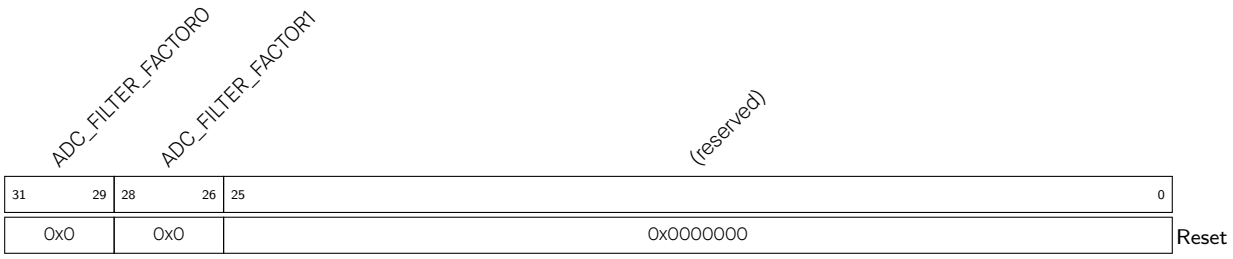
1: Enable

(R/W)

ADC_TIMER_TARGET Configures the half-period count for one sampling trigger pulse from the timer. Measurement unit: ADC working clock cycles.

(R/W)

Register 28.4. ADC_FILTER_CTRL1_REG (0x00C)



ADC_FILTER_FACTOR0 Configures the filter factor for filter channel 0.

0: k=1 (no filtering)

1: k=2

2: k=4

3: k=8

4: k=16

5: k=32

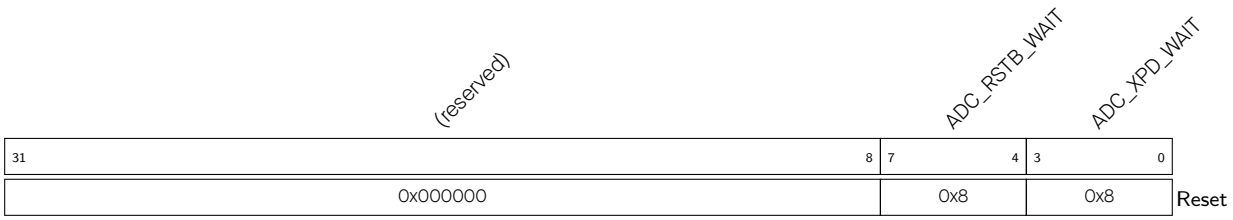
6: k=64

7: reserved

(R/W)

ADC_FILTER_FACTOR1 Configures the filter factor for filter channel 1. Encoding is the same as ADC_FILTER_FACTOR0. (R/W)

Register 28.5. ADC_FSM_WAIT_REG (0x010)



ADC_RSTB_WAIT Configures the SAR channel reset wait time.

Measurement unit: completed SAR conversions (each corresponds to one T_{conv} = 24 ADC working clocks).

When set to 0, the state lasts a single ADC working clock.

(R/W)

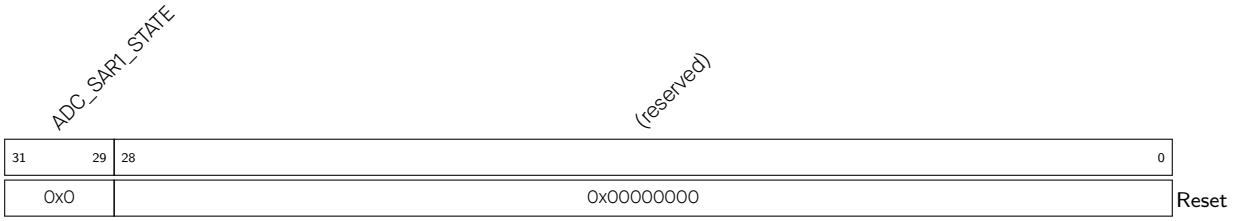
ADC_XPD_WAIT Configures the SAR power-up wait time.

Measurement unit: completed SAR conversions (each corresponds to one T_{conv} = 24 ADC working clocks).

When set to 0, the state lasts a single ADC working clock.

(R/W)

Register 28.6. ADC_SAR1_STATUS_REG (0x014)



ADC_SAR1_STATE Represents the current SAR1 FSM state.

0: IDLE

1: XPD

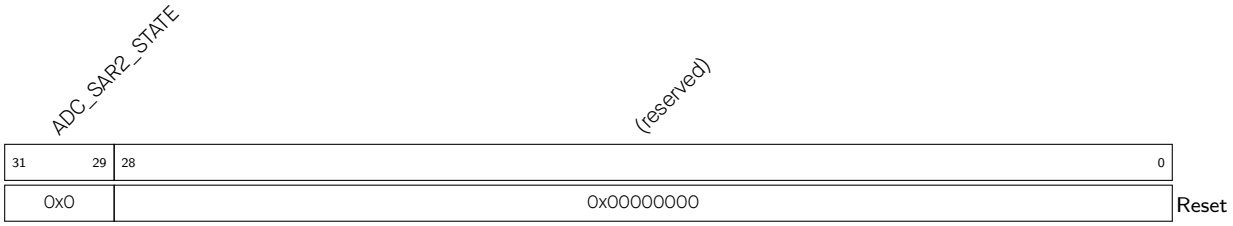
2: RSTB

3: READ

4–7: reserved

(RO)

Register 28.7. ADC_SAR2_STATUS_REG (0x018)



ADC_SAR2_STATE Represents the current SAR2 FSM state.

0: IDLE

1: XPD

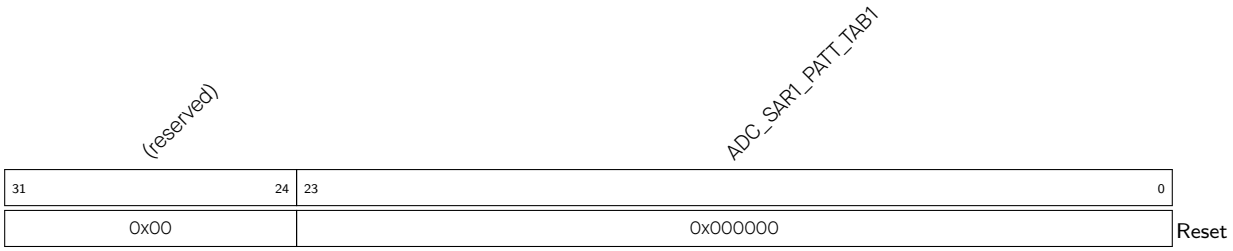
2: RSTB

3: READ

4–7: reserved

(RO)

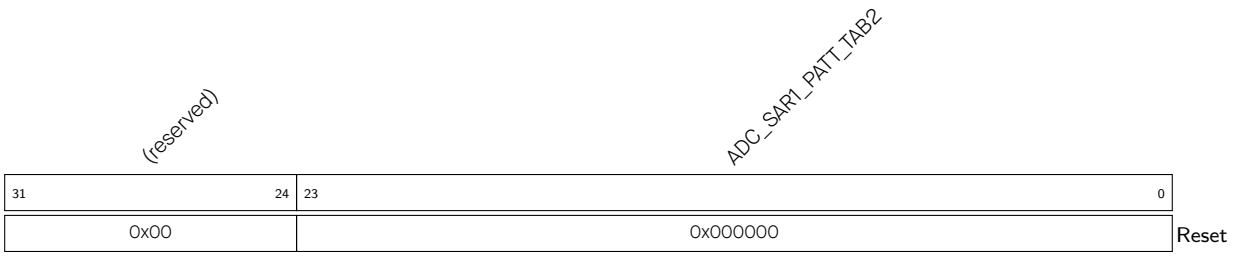
Register 28.8. ADC_SAR1_PATT_TAB1_REG (0x01C)



ADC_SAR1_PATT_TAB1 Configures SAR1 pattern table patterns 0–3. Each pattern is 6 bits. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

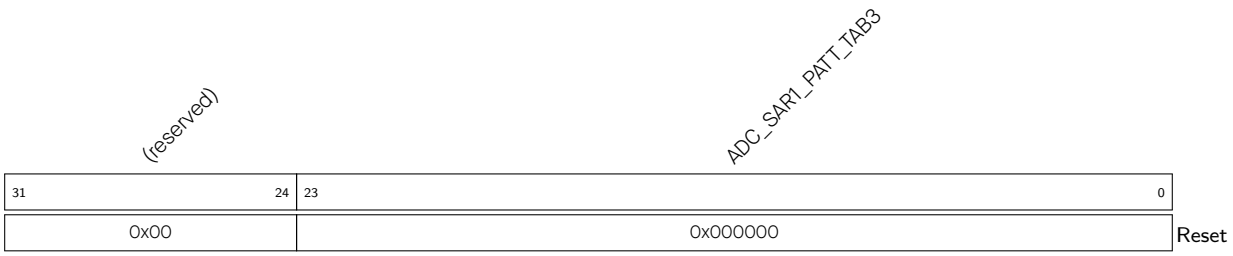
Register 28.9. ADC_SAR1_PATT_TAB2_REG (0x020)



ADC_SAR1_PATT_TAB2 Configures SAR1 pattern table patterns 4–7. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

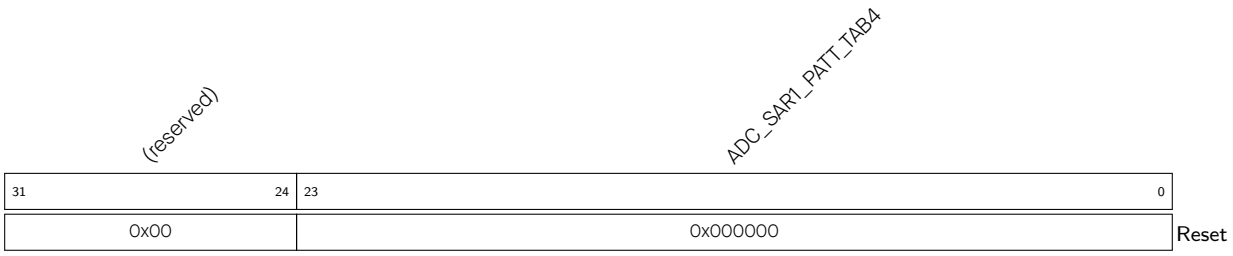
Register 28.10. ADC_SAR1_PATT_TAB3_REG (0x024)



ADC_SAR1_PATT_TAB3 Configures SAR1 pattern table patterns 8–11. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

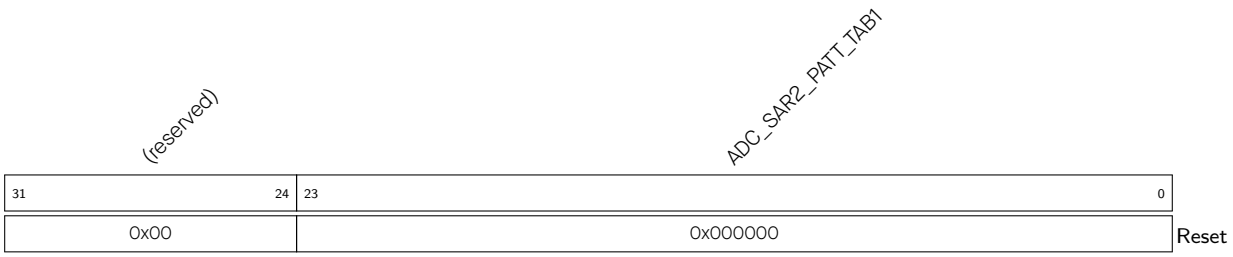
Register 28.11. ADC_SAR1_PATT_TAB4_REG (0x028)



ADC_SAR1_PATT_TAB4 Configures SAR1 pattern table patterns 12–15. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

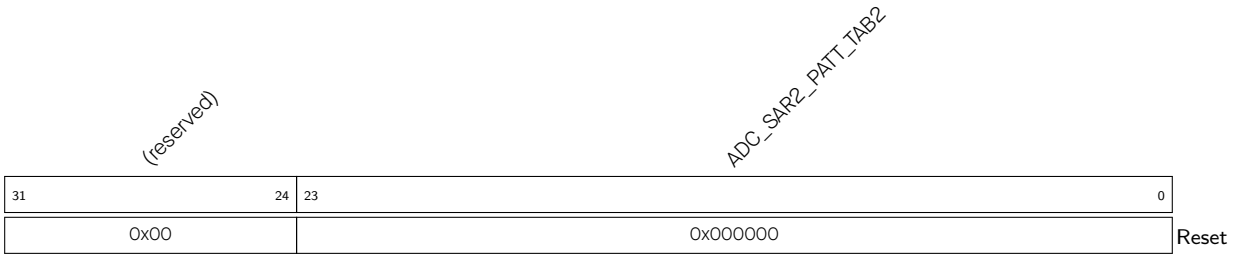
Register 28.12. ADC_SAR2_PATT_TAB1_REG (0x02C)



ADC_SAR2_PATT_TAB1 Configures SAR2 pattern table patterns 0–3. Each pattern is 6 bits. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

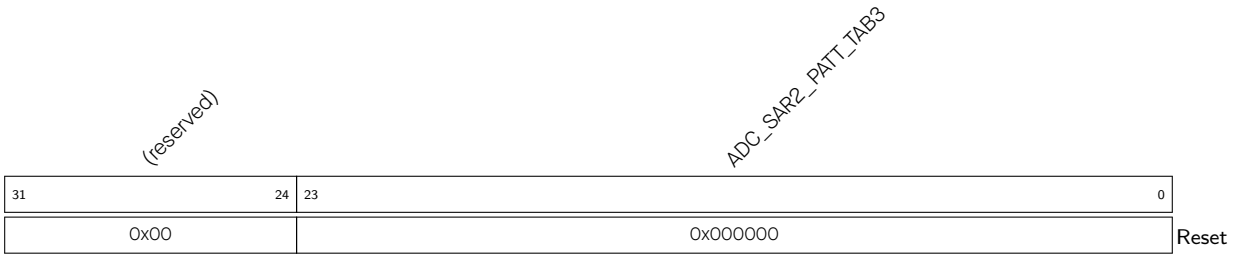
Register 28.13. ADC_SAR2_PATT_TAB2_REG (0x030)



ADC_SAR2_PATT_TAB2 Configures SAR2 pattern table patterns 4–7. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

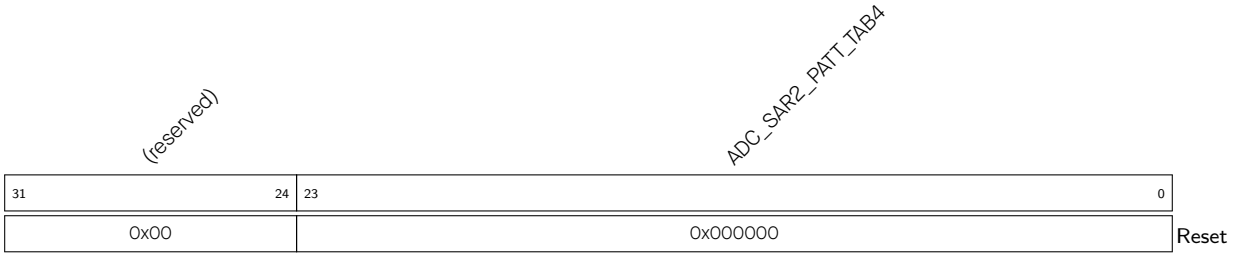
Register 28.14. ADC_SAR2_PATT_TAB3_REG (0x034)



ADC_SAR2_PATT_TAB3 Configures SAR2 pattern table patterns 8–11. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

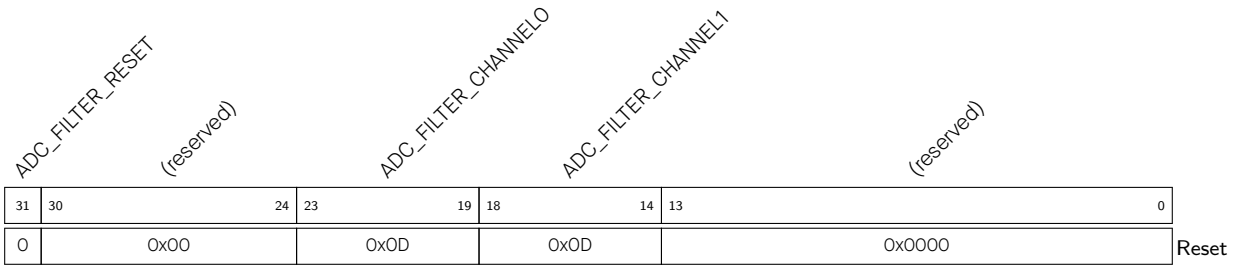
Register 28.15. ADC_SAR2_PATT_TAB4_REG (0x038)



ADC_SAR2_PATT_TAB4 Configures SAR2 pattern table patterns 12–15. See Section 28.4.2.1 *Pattern Table Structure*.

(R/W)

Register 28.16. ADC_FILTER_CTRL0_REG (0x03C)



ADC_FILTER_RESET Write 1 to clear the internal state of both filter channels. (WT)

ADC_FILTER_CHANNEL0 Configures the channel number for filter channel 0.

bit[4]: Distinguishes SAR1/SAR2

0: SAR1

1: SAR2

bit[3:0]: Channel

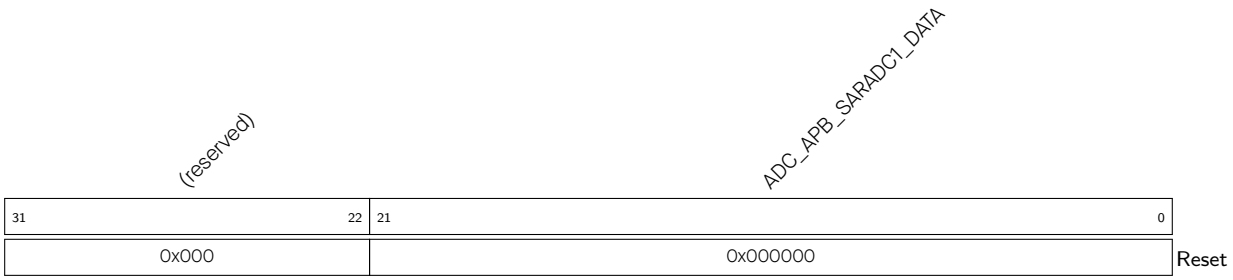
(R/W)

ADC_FILTER_CHANNEL1 Configures the channel number for filter channel 1.

Encoding for bit[4] and bit[3:0] is the same as ADC_FILTER_CHANNEL0.

(R/W)

Register 28.17. ADC_SAR1_DATA_STATUS_REG (0x040)



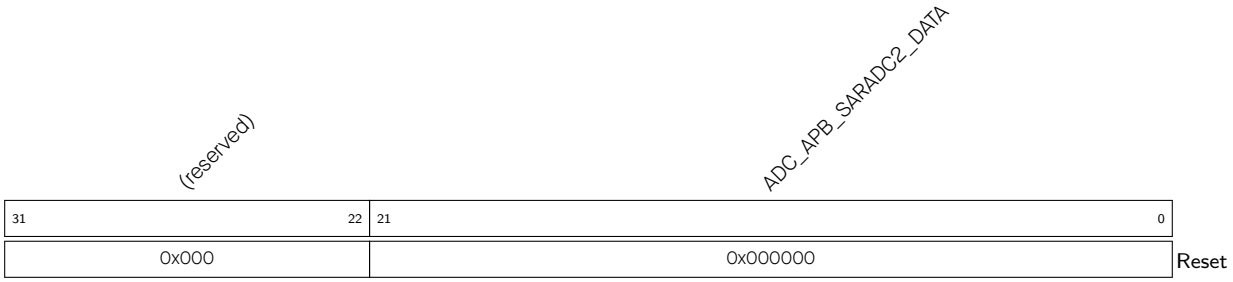
ADC_APB_SARADC1_DATA Represents the most recent SAR1 conversion data.

bit[21:17]: Channel number

bit[16:0]: 17-bit ADC data

(RO)

Register 28.18. ADC_SAR2_DATA_STATUS_REG (0x044)



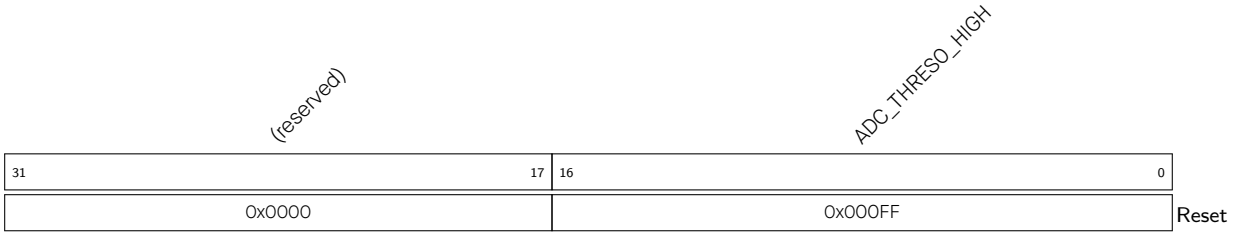
ADC_APB_SARADC2_DATA Represents the most recent SAR2 conversion data.

bit[21:17]: Channel number

bit[16:0]: 17-bit ADC data

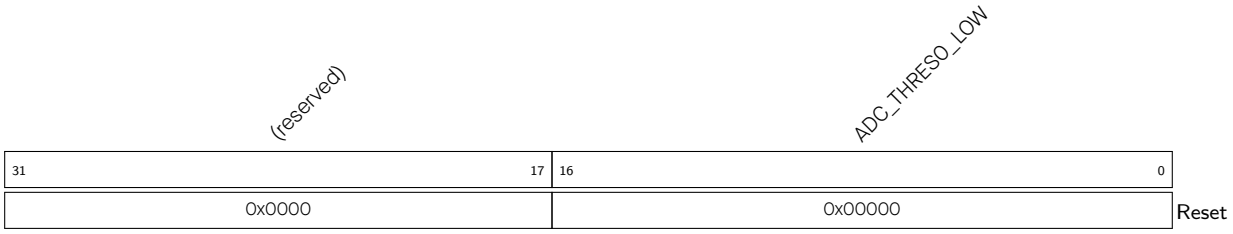
(RO)

Register 28.19. ADC_THRES0_HIGH_THRES_REG (0x048)



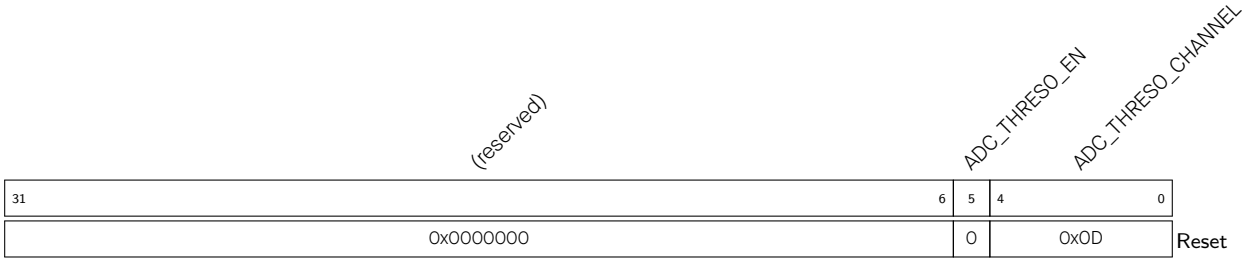
ADC_THRES0_HIGH Configures the upper threshold for threshold 0. (R/W)

Register 28.20. ADC_THRES0_LOW_THRES_REG (0x04C)



ADC_THRES0_LOW Configures the lower threshold for threshold 0. (R/W)

Register 28.21. ADC_THRES0_CTRL_REG (0x050)



ADC_THRES0_EN Configures whether to enable the threshold 0 comparator.

0: Disable

1: Enable

(R/W)

ADC_THRES0_CHANNEL Configures the channel monitored by threshold 0.

bit[4]: Distinguishes SAR1/SAR2

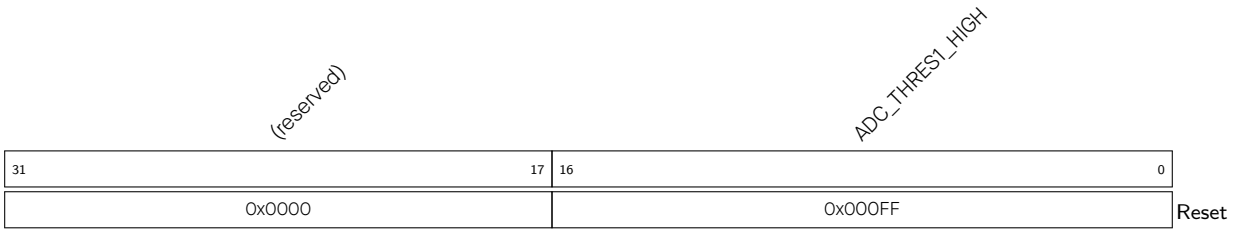
0: SAR1

1: SAR2

bit[3:0]: Channel

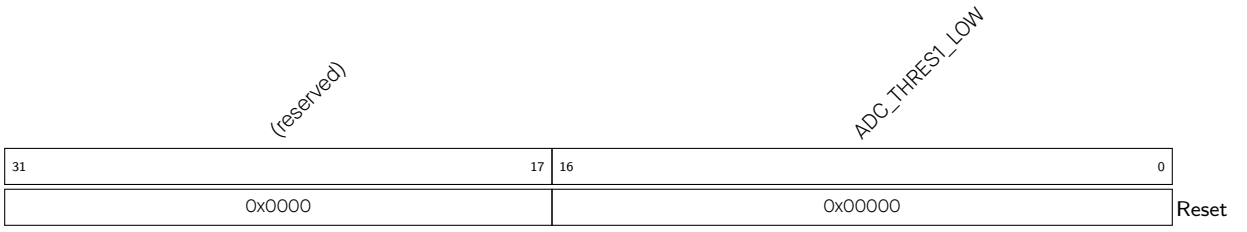
(R/W)

Register 28.22. ADC_THRES1_HIGH_THRES_REG (0x054)



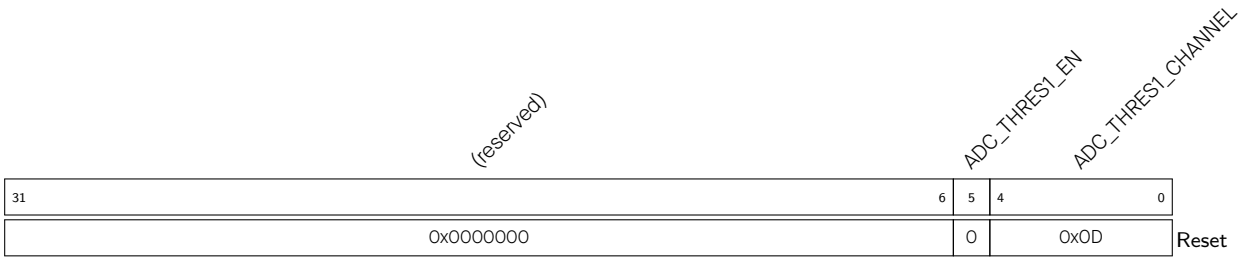
ADC_THRES1_HIGH Configures the upper threshold for threshold 1. (R/W)

Register 28.23. ADC_THRES1_LOW_THRES_REG (0x058)



ADC_THRES1_LOW Configures the lower threshold for threshold 1. (R/W)

Register 28.24. ADC_THRES1_CTRL_REG (0x05C)



ADC_THRES1_EN Configures whether to enable the threshold 1 comparator.

0: Disable

1: Enable

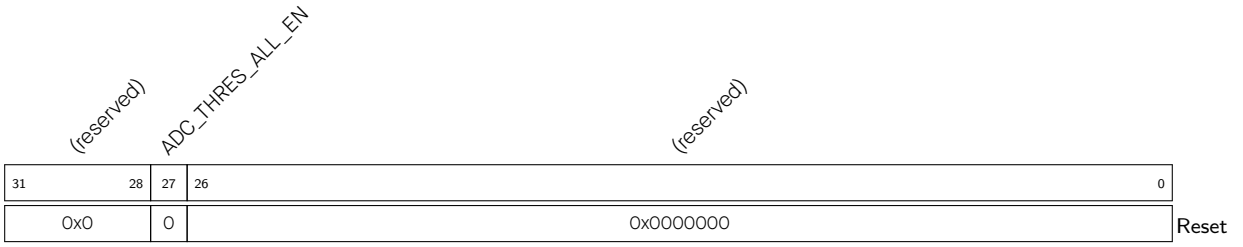
(R/W)

ADC_THRES1_CHANNEL Configures the channel monitored by threshold 1.

Encoding for bit[4] and bit[3:0] is the same as ADC_THRESO_CHANNEL.

(R/W)

Register 28.25. ADC_THRES_CTRL_REG (0x060)



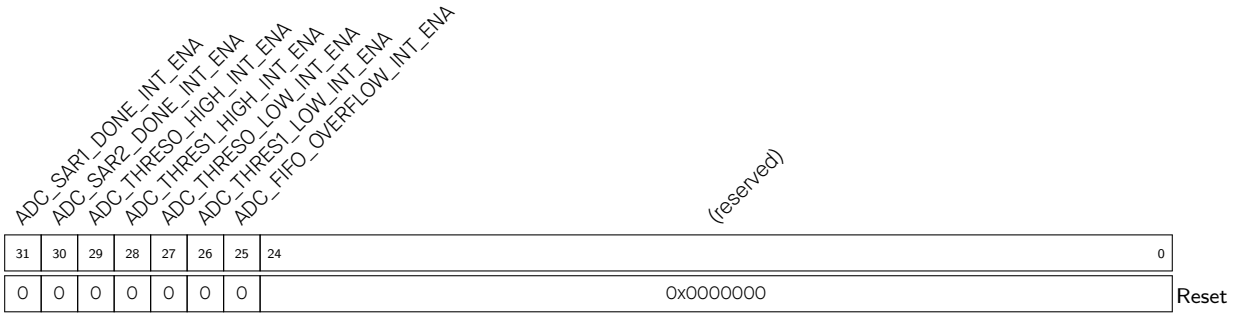
ADC_THRES_ALL_EN Configures whether to enable all-channel monitoring for threshold 0.

0: Monitor only the channel selected by ADC_THRESO_CHANNEL

1: Monitor all channels (threshold 1 is unaffected)

(R/W)

Register 28.26. ADC_INT_ENA_REG (0x064)



ADC_SAR1_DONE_INT_ENA Write 1 to enable the ADC_SAR1_DONE_INT interrupt. (R/W)

ADC_SAR2_DONE_INT_ENA Write 1 to enable the ADC_SAR2_DONE_INT interrupt. (R/W)

ADC_THRES0_HIGH_INT_ENA Write 1 to enable the ADC_THRES0_HIGH_INT interrupt. (R/W)

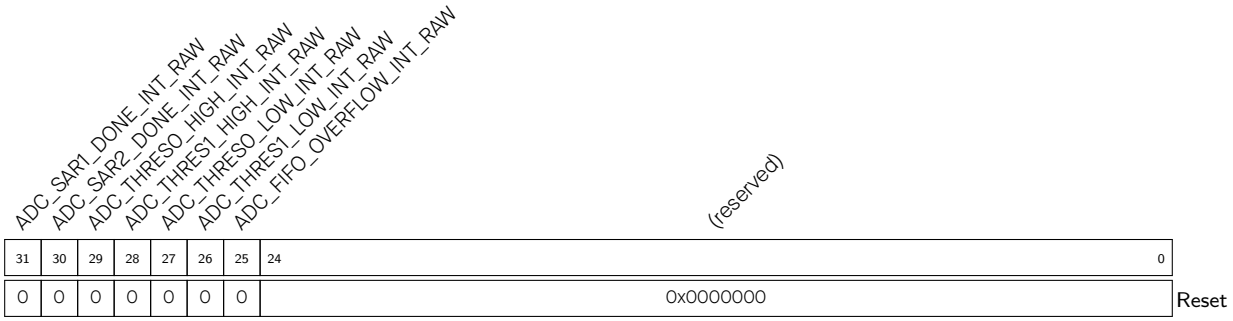
ADC_THRES1_HIGH_INT_ENA Write 1 to enable the ADC_THRES1_HIGH_INT interrupt. (R/W)

ADC_THRES0_LOW_INT_ENA Write 1 to enable the ADC_THRES0_LOW_INT interrupt. (R/W)

ADC_THRES1_LOW_INT_ENA Write 1 to enable the ADC_THRES1_LOW_INT interrupt. (R/W)

ADC_FIFO_OVERFLOW_INT_ENA Write 1 to enable the ADC_FIFO_OVERFLOW_INT interrupt. (R/W)

Register 28.27. ADC_INT_RAW_REG (0x068)



ADC_SAR1_DONE_INT_RAW The raw interrupt status of the ADC_SAR1_DONE_INT interrupt. (R/WTC/SS)

ADC_SAR2_DONE_INT_RAW The raw interrupt status of the ADC_SAR2_DONE_INT interrupt. (R/WTC/SS)

ADC_THRES0_HIGH_INT_RAW The raw interrupt status of the ADC_THRES0_HIGH_INT interrupt. (R/WTC/SS)

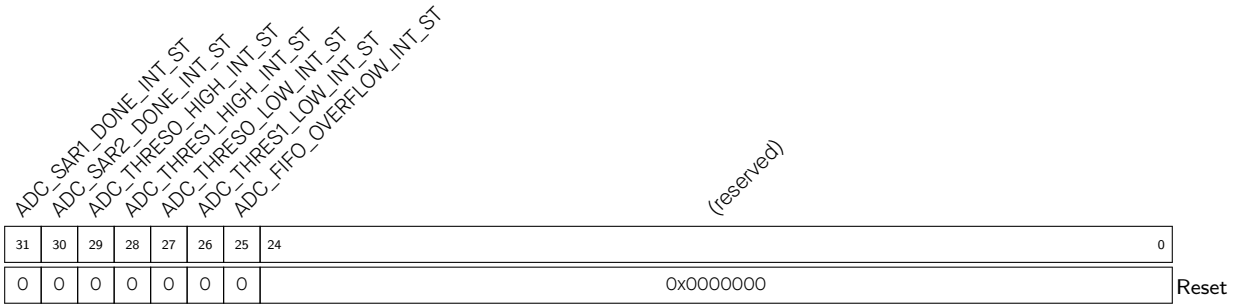
ADC_THRES1_HIGH_INT_RAW The raw interrupt status of the ADC_THRES1_HIGH_INT interrupt. (R/WTC/SS)

ADC_THRES0_LOW_INT_RAW The raw interrupt status of the ADC_THRES0_LOW_INT interrupt. (R/WTC/SS)

ADC_THRES1_LOW_INT_RAW The raw interrupt status of the ADC_THRES1_LOW_INT interrupt. (R/WTC/SS)

ADC_FIFO_OVERFLOW_INT_RAW The raw interrupt status of the ADC_FIFO_OVERFLOW_INT interrupt. (R/WTC/SS)

Register 28.28. ADC_INT_ST_REG (0x06C)



ADC_SAR1_DONE_INT_ST The masked interrupt status of the ADC_SAR1_DONE_INT interrupt. (RO)

ADC_SAR2_DONE_INT_ST The masked interrupt status of the ADC_SAR2_DONE_INT interrupt. (RO)

ADC_THRES0_HIGH_INT_ST The masked interrupt status of the ADC_THRES0_HIGH_INT interrupt. (RO)

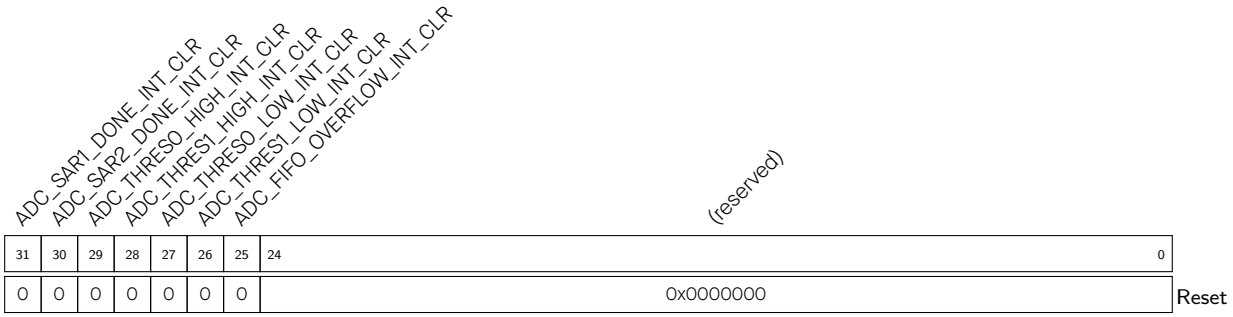
ADC_THRES1_HIGH_INT_ST The masked interrupt status of the ADC_THRES1_HIGH_INT interrupt. (RO)

ADC_THRES0_LOW_INT_ST The masked interrupt status of the ADC_THRES0_LOW_INT interrupt. (RO)

ADC_THRES1_LOW_INT_ST The masked interrupt status of the ADC_THRES1_LOW_INT interrupt. (RO)

ADC_FIFO_OVERFLOW_INT_ST The masked interrupt status of the ADC_FIFO_OVERFLOW_INT interrupt. (RO)

Register 28.29. ADC_INT_CLR_REG (0x070)



ADC_SAR1_DONE_INT_CLR Write 1 to clear the ADC_SAR1_DONE_INT interrupt. (WT)

ADC_SAR2_DONE_INT_CLR Write 1 to clear the ADC_SAR2_DONE_INT interrupt. (WT)

ADC_THRES0_HIGH_INT_CLR Write 1 to clear the ADC_THRES0_HIGH_INT interrupt. (WT)

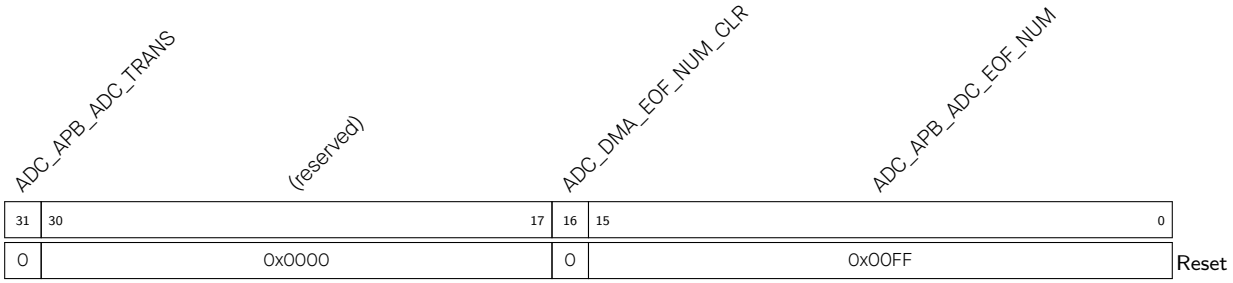
ADC_THRES1_HIGH_INT_CLR Write 1 to clear the ADC_THRES1_HIGH_INT interrupt. (WT)

ADC_THRES0_LOW_INT_CLR Write 1 to clear the ADC_THRES0_LOW_INT interrupt. (WT)

ADC_THRES1_LOW_INT_CLR Write 1 to clear the ADC_THRES1_LOW_INT interrupt. (WT)

ADC_FIFO_OVERFLOW_INT_CLR Write 1 to clear the ADC_FIFO_OVERFLOW_INT interrupt. (WT)

Register 28.30. ADC_DMA_CONF_REG (0x074)



ADC_APB_ADC_TRANS Configures whether to enable DMA transfer.

0: Disable

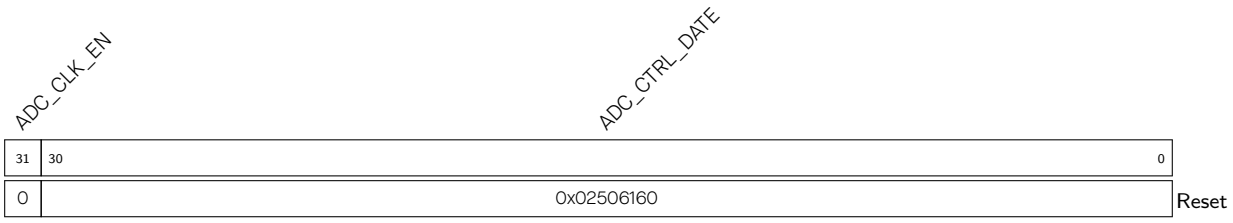
1: Enable

(R/W)

ADC_DMA_EOF_NUM_CLR Write 1 to clear the DMA EOF counter. (WT)

ADC_APB_ADC_EOF_NUM Configures the DMA frame length (number of samples). (R/W)

Register 28.31. ADC_CTRL_DATE_REG (0x3FC)



ADC_CLK_EN Configures register clock gating.

0: Automatic clock gating

1: Clock always on

(R/W)

ADC_CTRL_DATE Version control register. (R/W)

Chapter 29

Analog Voltage Comparator

29.1 Introduction

ESP32-S31 integrates an analog voltage comparator that monitors voltage changes by comparing the input voltage with a reference voltage. The comparator provides multiple chip pads (PADs) for voltage signal input and supports multi-channel comparison. Comparison results can generate ETM (Event Task Matrix) events or trigger interrupts.

29.2 Feature List

The analog voltage comparator has the following main features:

- Provides four chip pad (PAD) interfaces for voltage signal input
- Supports internal or external reference voltage, with hysteresis function available for internal reference voltage
- Supports multi-channel voltage comparison:
  - Supports up to three channels
  - Compares the input voltages from different PADs with the reference voltage sequentially through polling scan
  - Scan can be initiated by software or triggered by ETM tasks
- Each channel provides the following functions:
  - Glitch filtering
  - Status changes can generate ETM events or trigger interrupts
  - Channel timer for recording interrupt intervals

29.3 Architectural Overview

Figure 29.3-1 shows the structure of the analog voltage comparator.

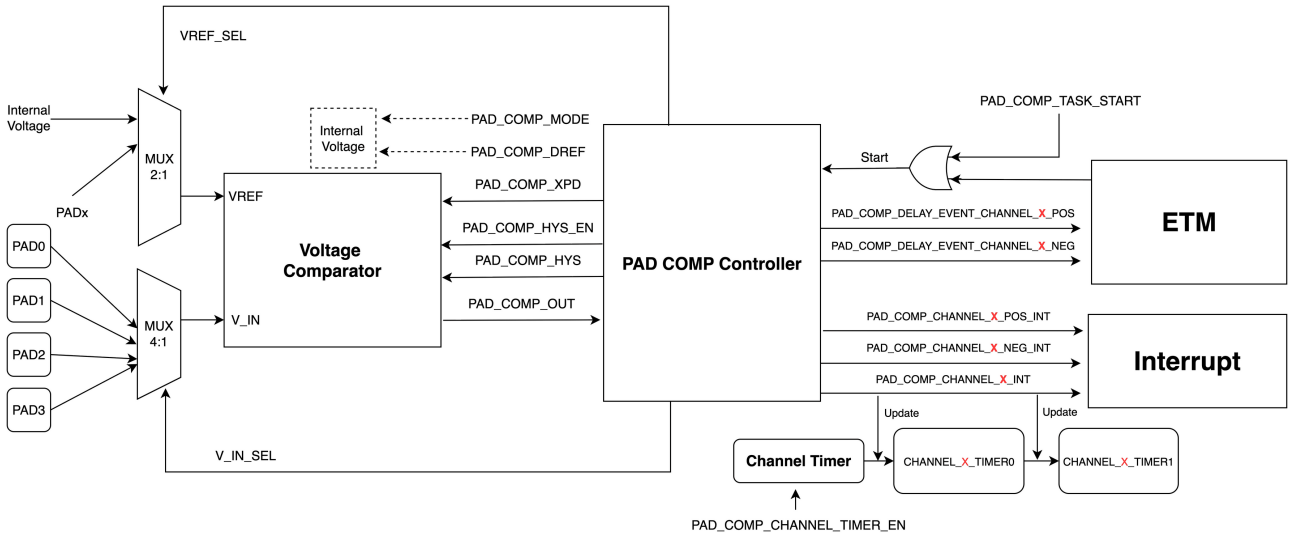


Figure 29.3-1. Analog Voltage Comparator Architecture

As shown in the figure above, the structure of the analog voltage comparator is as follows:

- Four PAD interfaces: PAD0~PAD3, used for voltage signal input, which can serve as channel inputs or external reference voltage source.
- Voltage Comparator: Compares the input voltage (V_IN) with the reference voltage (VREF), and outputs the comparison result through PAD_COMP_OUT. The reference voltage can be generated internally or provided by an external PAD.
- PAD COMP Controller (Analog Voltage Comparator Controller): Controls channel scanning, processes comparison results, and generates interrupts and ETM events.
- Channel Timer: Provides timing reference for interrupt intervals of each channel.

Table 29.3-1 lists the mapping between the comparator PADs and GPIO pins.

Table 29.3-1. Mapping Between PADs and GPIO Pins

	PAD0	PAD1	PAD2	PAD3
GPIO Pin	GPIO37	GPIO38	GPIO39	GPIO40

29.4 Functional Description

29.4.1 Reference Voltage Configuration

The reference voltage of the analog voltage comparator can be either internal or external. According to the reference voltage source, the comparison modes are divided into the following two types:

- **Internal Reference Voltage Mode**: Set PAD_COMP_MODE = 0 to compare the input voltage with the internal reference voltage.

  In this mode, all four PADs can be assigned to channels as inputs, but since the hardware only supports three channels, a maximum of three PADs can be used.

  The internal reference voltage is configured by PAD_COMP_DREF:

- Formula: VREF = PAD_COMP_DREF × 0.1 × VDDPST_3 (V)
- Range: 0 ~ (0.7 × VDDPST_3) V
- Step size: (0.1 × VDDPST_3) V

VDDPST_3 is the supply voltage of the digital HP IO power domain that powers the comparator PADs. It is supplied via the VDDPST_3 pin and is typically 3.3 V.

The internal reference voltage supports hysteresis function, which can improve the stability of the comparator in the critical region. Hysteresis is enabled when PAD_COMP_HYS_EN = 1. PAD_COMP_HYS configures the hysteresis voltage V_HYS.

When PAD_COMP_OUT is low, the internal reference voltage is VREF + V_HYS; when PAD_COMP_OUT is high, the internal reference voltage is VREF. Table 29.4-1 lists the correspondence between V_HYS and PAD_COMP_HYS.

Table 29.4-1. V_HYS Values

PAD_COMP_HYS	V_HYS
0	0 V
1	0.04 V
2	0.08 V
4	0.12 V
3, 5, 6, 7	Reserved

- **External Reference Voltage Mode**: Set PAD_COMP_MODE = 1 to compare the input voltage with the external reference voltage.

  In this mode, one of the four PADs needs to be selected as the external reference voltage source, and the remaining three PADs can be assigned to the three channels as inputs.

  Select the external reference voltage source PAD through PAD_COMP_VREF_CHANNEL_SEL. This field is a 4-bit bitmap, corresponding from high to low to PAD3, PAD2, PAD1, and PAD0, with only one bit allowed to be 1.

  Note:
  The PAD used as the external reference voltage source should not be assigned to any channel as an input PAD simultaneously.

29.4.2 PAD COMP Controller

The controller (PAD COMP Controller) supports polling scan, scanning enabled channels sequentially in channel number order (1 → 2 → 3).

- **Input PAD Assignment**: Select the input PAD for each channel through PAD_COMP_CHANNEL_X_SEL (X = 1, 2, 3).

  This field is a 4-bit bitmap, corresponding from high to low to PAD3, PAD2, PAD1, and PAD0, with only one bit allowed to be 1.

> **Note:**
> Each channel must select a different PAD. In external reference voltage mode, the PAD used as the external reference voltage source cannot be assigned to any channel simultaneously.

- **Enable Channel**: Select which channels participate in scanning through PAD_COMP_POLL_MASK.

  This field is a 3-bit bitmap, corresponding from high to low to channel 3, channel 2, and channel 1. The corresponding bit being 1 indicates that the channel is enabled, and unenabled channels will be skipped during the scanning process.

- **Scan Trigger and Mode**: Scanning can be triggered by software via PAD_COMP_START, or by the ETM task PAD_COMP_TASK_START. The scanning behavior is controlled by PAD_COMP_POLL_MODE:
  - 0: Full scan mode. Upon each trigger, scan all enabled channels sequentially.
  - 1: Single-step polling mode. Upon each trigger, scan only the next enabled channel.
- **Channel Switching Wait Time**: Since the Voltage Comparator is a multiplexed module, a certain stabilization time is required when switching input channels, controlled by PAD_COMP_POLL_PERIOD, in units of PAD_COMP_CLK clock cycles, typically 1 μs.
- **Channel Status and Glitch Filtering**: To avoid glitches when the input channel is in the critical region of the reference voltage, the module supports glitch filtering. After the stabilization time ends, glitch filtering is performed on PAD_COMP_OUT according to the channel status PAD_COMP_CHANNEL_X_STATUS.

  If the current status is high, PAD_COMP_LIMIT_CNT consecutive low levels need to be sampled before switching to low. Conversely, if the current status is low, PAD_COMP_LIMIT_CNT consecutive high levels need to be sampled before the channel status switches to high.

  After PAD_COMP_OUT transitions once and triggers an interrupt, PAD_COMP_FILTER_CNT configures a debounce time in PAD_COMP_CLK clock cycles, during which no new interrupt is triggered. This suppresses duplicate interrupts caused by the input voltage jittering around the reference voltage. PAD_COMP_LIMIT_CNT and PAD_COMP_FILTER_CNT work together: the former filters glitches during sampling, and the latter suppresses duplicate interrupts after one has been triggered.
- **Channel Interrupt**: Interrupts are triggered by changes in channel status:
  - When PAD_COMP_CHANNEL_X_STATUS changes from low to high, PAD_COMP_CHANNEL_X_INT and PAD_COMP_CHANNEL_X_POS_INT are triggered.
  - When PAD_COMP_CHANNEL_X_STATUS changes from high to low, PAD_COMP_CHANNEL_X_INT and PAD_COMP_CHANNEL_X_NEG_INT are triggered.
- **Channel ETM Event**: When PAD_COMP_CHANNEL_X_EVENT_TIMER_EN = 1, ETM events are generated with a delay of PAD_COMP_DELAY_EVENT_TIME PAD_COMP_CLK clock cycles after the interrupt is triggered:
  - PAD_COMP_DELAY_EVENT_CHANNEL_X_POS: Corresponds to PAD_COMP_CHANNEL_X_POS_INT
  - PAD_COMP_DELAY_EVENT_CHANNEL_X_NEG: Corresponds to PAD_COMP_CHANNEL_X_NEG_INT
- **Channel Timer**: When PAD_COMP_CHANNEL_TIMER_EN = 1, the channel timer continuously counts according to the PAD_COMP_CLK. Whenever PAD_COMP_CHANNEL_X_INT is generated, the current

count value is saved to PAD_COMP_CHANNEL_X_TIMER0, while the previous TIMER0 value is saved to PAD_COMP_CHANNEL_X_TIMER1, thereby recording the time interval between two interrupts.

Figure 29.4-1 shows the multi-channel scanning timing in full scan mode. In this example, the configuration is: PAD_COMP_POLL_MODE = 0, PAD_COMP_POLL_MASK = 3'b111 (all three channels enabled), PAD_COMP_POLL_PERIOD = 4, PAD_COMP_LIMIT_CNT = 5, PAD_COMP_CHANNEL_1_SEL = 4'b0001, PAD_COMP_CHANNEL_2_SEL = 4'b0010, PAD_COMP_CHANNEL_3_SEL = 4'b0100.

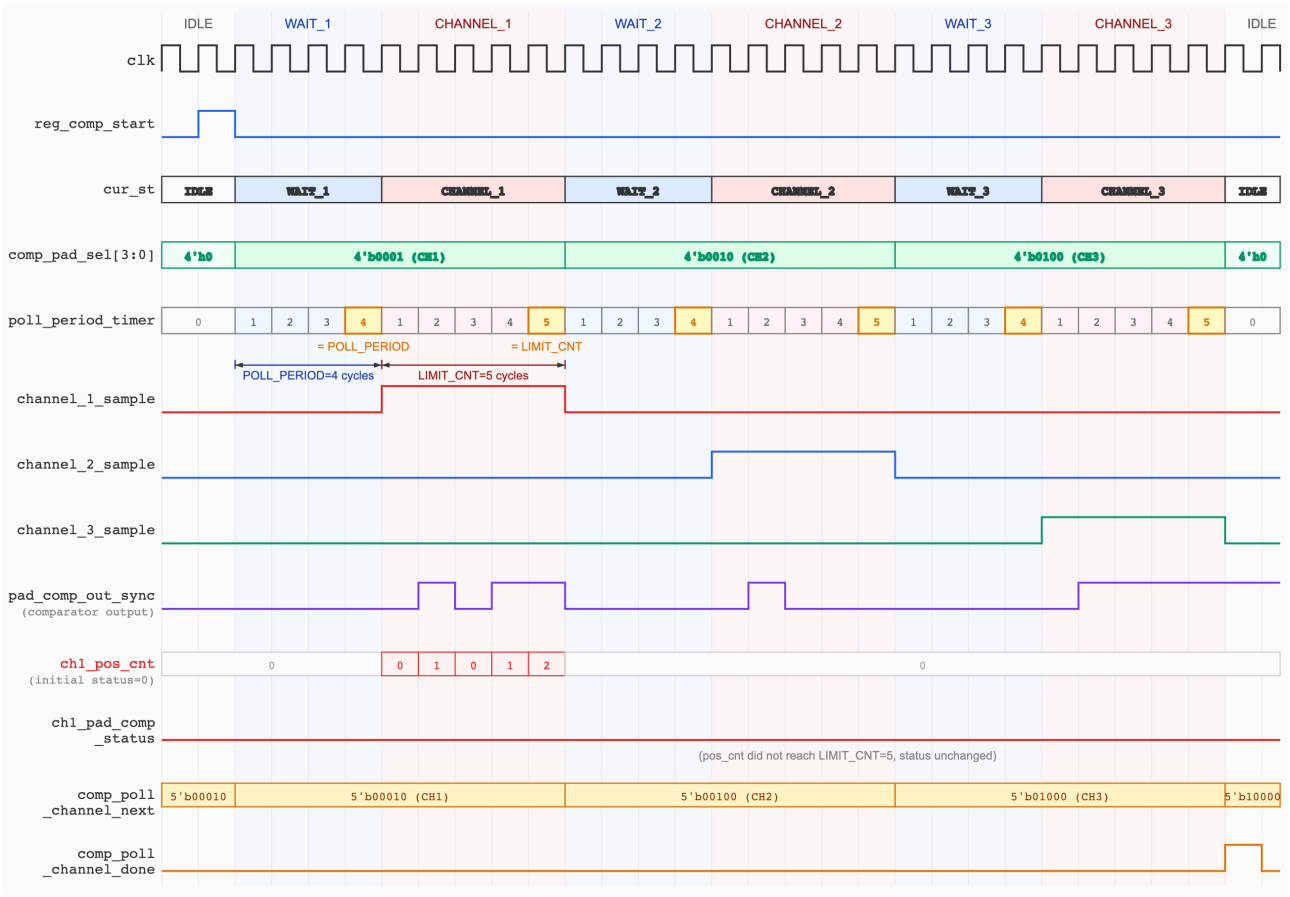


Figure 29.4-1. Analog Voltage Comparator Full Scan Mode Timing Diagram

- The blue-shaded WAIT_x phases indicate that the channel has been switched and the comparator output needs PAD_COMP_POLL_PERIOD cycles to stabilize.
- The red-shaded CHANNEL_x phases indicate that the comparator output is sampled for PAD_COMP_LIMIT_CNT cycles, with independent counting for positive and negative directions.
- The yellow highlighted values indicate timer match values that trigger state transitions.

In full scan mode, the state machine iterates through all channels enabled by PAD_COMP_POLL_MASK (WAIT → CHANNEL) and returns to IDLE after all channels are completed. In single-step polling mode (PAD_COMP_POLL_MODE = 1), each trigger completes only one channel before returning to IDLE, and the next trigger continues polling the next enabled channel.

29.4.3 Clock

The analog voltage comparator has two types of clocks:

- APB_CLK: the bus clock used for register access, enabled by HP_SYS_CLKRST_PAD_COMP_APB_CLK_EN
- PAD_COMP_CLK: the functional clock used by the comparator, enabled by HP_SYS_CLKRST_PAD_COMP_CLK_EN

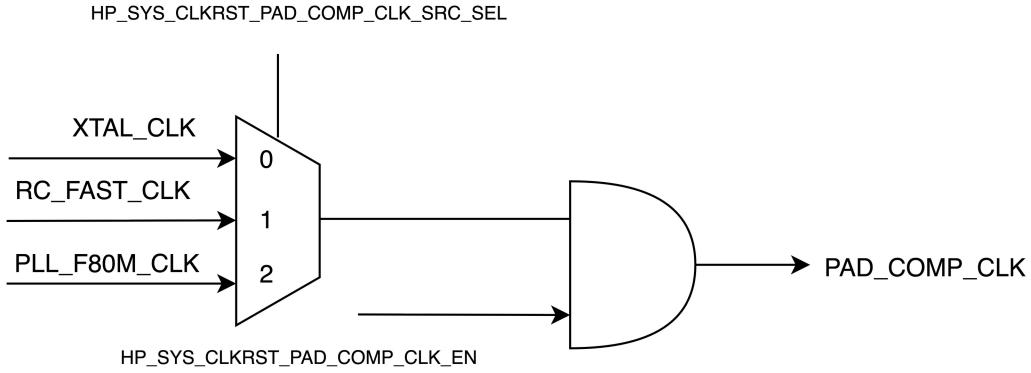


Figure 29.4-2. Analog Voltage Comparator Clock Diagram

As shown in Figure 29.4-2 *Analog Voltage Comparator Clock Diagram*, the clock source of PAD_COMP_CLK is selected through HP_SYS_CLKRST_PAD_COMP_CLK_SRC_SEL, and the available clock sources include:

- XTAL_CLK
- RC_FAST_CLK
- PLL_F80M_CLK

For more information about clocks, please refer to Chapter 7 *Reset and Clock [to be added later]*.

29.4.4 Reset

The analog voltage comparator has two reset controls. Write 1 to the corresponding field to initiate the reset and 0 to release the reset:

- APB reset: HP_SYS_CLKRST_PAD_COMP_APB_RST_EN
- Core reset: HP_SYS_CLKRST_PAD_COMP_CORE_RST_EN

For more information about reset, please refer to Chapter 7 *Reset and Clock [to be added later]*.

29.5 Event Task Matrix

In ESP32-S31, the analog voltage comparator supports ETM feature. The module can generate ETM events to trigger ETM tasks of other peripherals. For more details about ETM, please refer to Chapter 10 Event Task Matrix (ETM) [to be added later]. This section only describes ETM events and tasks related to the analog voltage comparator.

The ETM events generated by the analog voltage comparator include:

- PAD_COMP_DELAY_EVENT_CHANNEL_X_POS: The PAD_COMP_OUT of channel X (after glitch filtering) changes from low to high, i.e., the input voltage changes from below the reference voltage to above the

reference voltage.

- PAD_COMP_DELAY_EVENT_CHANNEL_X_NEG: The PAD_COMP_OUT of channel X (after glitch filtering) changes from high to low, i.e., the input voltage changes from above the reference voltage to below the reference voltage.

The ETM tasks that can trigger the analog voltage comparator include:

- PAD_COMP_TASK_START: Triggers channel scanning. According to PAD_COMP_POLL_MODE configuration: when it is 0, each trigger completes one scan of all enabled channels; when it is 1, each trigger completes one scan of only the next enabled channel.

29.6 Interrupts

The analog voltage comparator in ESP32-S31 can generate the PAD_COMP_INT interrupt signal and send it to the Interrupt Matrix.

There are several internal interrupt sources from analog voltage comparator that can generate the above interrupt signal. Table 29.6-1 lists the internal interrupt sources, trigger conditions, and generated interrupt signals of the analog voltage comparator.

Table 29.6-1. Internal Interrupt Sources of the Analog Voltage Comparator

Internal Interrupt Source	Trigger Condition	Interrupt Signal
PAD_COMP_CHANNEL_X_INT	The filtered PAD_COMP_OUT signal transitions	PAD_COMP_INT
PAD_COMP_CHANNEL_X_NEG_INT	The filtered PAD_COMP_OUT signal transitions from high to low	PAD_COMP_INT
PAD_COMP_CHANNEL_X_POS_INT	The filtered PAD_COMP_OUT signal transitions from low to high	PAD_COMP_INT

Note:
For explanations of terms such as interrupt, interrupt source, and interrupt signal, and their relationships, please refer to Chapter 9 *Interrupt Matrix [to be added later]* > 9.1 *Interrupt Terminology*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. For specific register names, please refer to Section 29.8 *Register Summary*.

29.7 Programming Procedures

The programming procedure for the analog voltage comparator is as follows:

1. Set HP_SYS_CLKRST_PAD_COMP_APB_CLK_EN to 1 to enable the bus clock APB_CLK.
2. Set HP_SYS_CLKRST_PAD_COMP_CLK_EN to 1 to enable the functional clock PAD_COMP_CLK.
3. Set PAD_COMP_XPD to 1 to power up and enable the voltage comparator.

4. Configure the reference voltage: Select internal or external mode and set the corresponding parameters (refer to 29.4.1 *Reference Voltage Configuration*).
5. If external reference voltage mode is selected, the digital functions of the PAD used as the external reference voltage source need to be disabled, including IE, OE, WPU, and WPD. For specific configuration methods, please refer to the subsection 34.1 *Analog Functions [to be added later]* in Chapter 6 *GPIO Matrix and IO MUX [to be added later]*.
6. Configure the controller: Assign channel input PADs, enable channels, set scanning mode, etc. (refer to 29.4.2 *PAD COMP Controller*).
7. Trigger scanning via PAD_COMP_START or PAD_COMP_TASK_START, and obtain comparison results.

29.8 Register Summary

All addresses in this section are address offsets relative to the analog voltage comparator base address. For the specific base address, please refer to Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

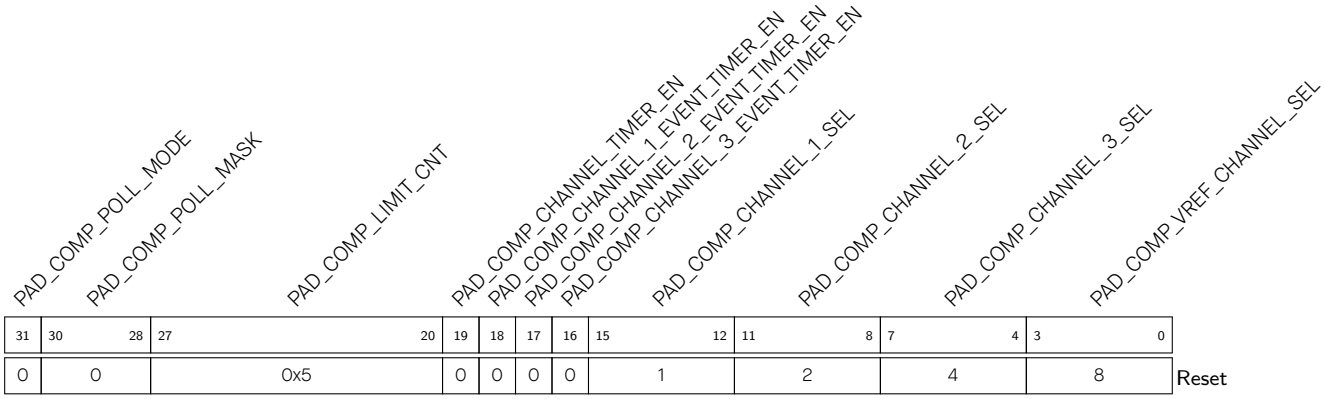
Name	Description	Address	Access
Configuration Registers			
PAD_COMP_CONF_REG	Channel and scan configuration register	0x0000	R/W
PAD_COMP_FILTER_CNT_REG	Debounce time register	0x0004	R/W
PAD_COMP_POLL_PERIOD_REG	Channel switch period register	0x0008	R/W
PAD_COMP_DELAY_EVENT_TIME_REG	ETM event delay time register	0x000C	R/W
PAD_COMP_INT_ENA_REG	Interrupt enable register	0x0010	R/W
PAD_COMP_INT_RAW_REG	Raw interrupt status register	0x0014	R/WTC/SS
PAD_COMP_INT_CLR_REG	Interrupt clear register	0x0018	WT
PAD_COMP_INT_ST_REG	Masked interrupt status register	0x001C	RO
PAD_COMP_CHANNEL_1_TIMER0_REG	Channel 1 timer 0 register	0x0020	RO
PAD_COMP_CHANNEL_1_TIMER1_REG	Channel 1 timer 1 register	0x0024	RO
PAD_COMP_CHANNEL_2_TIMER0_REG	Channel 2 timer 0 register	0x0028	RO
PAD_COMP_CHANNEL_2_TIMER1_REG	Channel 2 timer 1 register	0x002C	RO
PAD_COMP_CHANNEL_3_TIMER0_REG	Channel 3 timer 0 register	0x0030	RO
PAD_COMP_CHANNEL_3_TIMER1_REG	Channel 3 timer 1 register	0x0034	RO
PAD_COMP_CHANNEL_STATUS_REG	Channel status register	0x0038	RO
PAD_COMP_CFG_REG	Comparator configuration register	0x003C	R/W
PAD_COMP_START_REG	PAD COMP start configuration register	0x0040	WT
Version Register			
PAD_COMP_DATE_REG	Version register	0x03FC	R/W

29.9 Registers

All addresses in this section are address offsets relative to the analog voltage comparator base address. For the specific base address, please refer to Table 3 in Chapter 3 *System and Memory [to be added later]*.

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

Register 29.1. PAD_COMP_CONF_REG (0x0000)



PAD_COMP_VREF_CHANNEL_SEL Configures the PAD used as the external reference voltage source.

4-bit bitmap, corresponding from high to low to PAD3~PAD0, with only one bit allowed to be 1. Valid only when PAD_COMP_MODE = 1.

(R/W)

PAD_COMP_CHANNEL_3_SEL Configures the input PAD for channel 3.

4-bit bitmap, corresponding from high to low to PAD3~PAD0, with only one bit allowed to be 1.

Note: Each channel must select a different PAD and cannot select the PAD assigned to PAD_COMP_VREF_CHANNEL_SEL.

(R/W)

PAD_COMP_CHANNEL_2_SEL Configures the input PAD for channel 2.

4-bit bitmap, corresponding from high to low to PAD3~PAD0, with only one bit allowed to be 1.

Note: Each channel must select a different PAD and cannot select the PAD assigned to PAD_COMP_VREF_CHANNEL_SEL.

(R/W)

PAD_COMP_CHANNEL_1_SEL Configures the input PAD for channel 1.

4-bit bitmap, corresponding from high to low to PAD3~PAD0, with only one bit allowed to be 1.

Note: Each channel must select a different PAD and cannot select the PAD assigned to PAD_COMP_VREF_CHANNEL_SEL.

(R/W)

Continued on next page...

Register 29.1. PAD_COMP_CONF_REG (0x0000)

Continued from previous page...

PAD_COMP_CHANNEL_3_EVENT_TIMER_EN Configures whether to enable the ETM event delay function for channel 3.
0: Disable
1: Enable
(R/W)

PAD_COMP_CHANNEL_2_EVENT_TIMER_EN Configures whether to enable the ETM event delay function for channel 2.
0: Disable
1: Enable
(R/W)

PAD_COMP_CHANNEL_1_EVENT_TIMER_EN Configures whether to enable the ETM event delay function for channel 1.
0: Disable
1: Enable
(R/W)

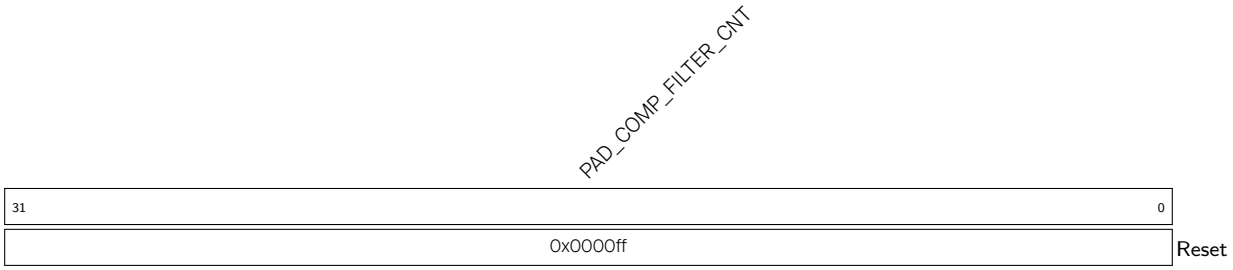
PAD_COMP_CHANNEL_TIMER_EN Configures whether to enable the channel timer.
0: Disable
1: Enable
(R/W)

PAD_COMP_LIMIT_CNT Configures the number of consecutive samples required for glitch filtering.
(R/W)

PAD_COMP_POLL_MASK Configures the channels participating in scanning.
3-bit bitmap, corresponding from high to low to channel 3~1. The corresponding bit being 1 indicates that the channel is enabled.
(R/W)

PAD_COMP_POLL_MODE Configures the channel scanning mode.
0: Full scan mode
1: Single-step polling mode
(R/W)

Register 29.2. PAD_COMP_FILTER_CNT_REG (0x0004)



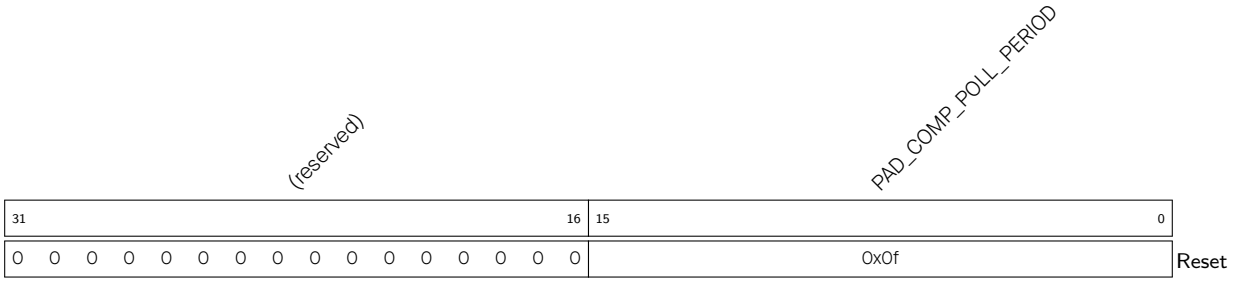
PAD_COMP_FILTER_CNT Configures the debounce time after an interrupt is triggered. No new interrupt is triggered during this time.

Unit: PAD_COMP_CLK clock cycle.

This field must be greater than or equal to 1.

(R/W)

Register 29.3. PAD_COMP_POLL_PERIOD_REG (0x0008)

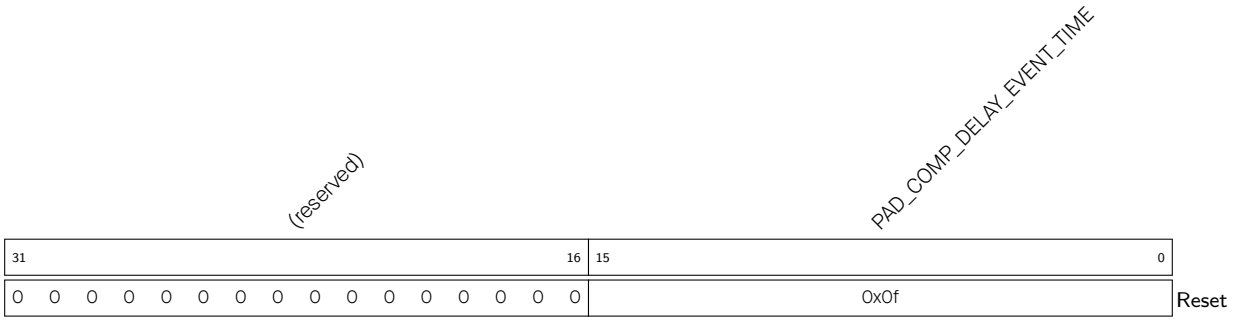


PAD_COMP_POLL_PERIOD Configures the channel switching stabilization time.

Unit: PAD_COMP_CLK clock cycle.

(R/W)

Register 29.4. PAD_COMP_DELAY_EVENT_TIME_REG (0x000C)



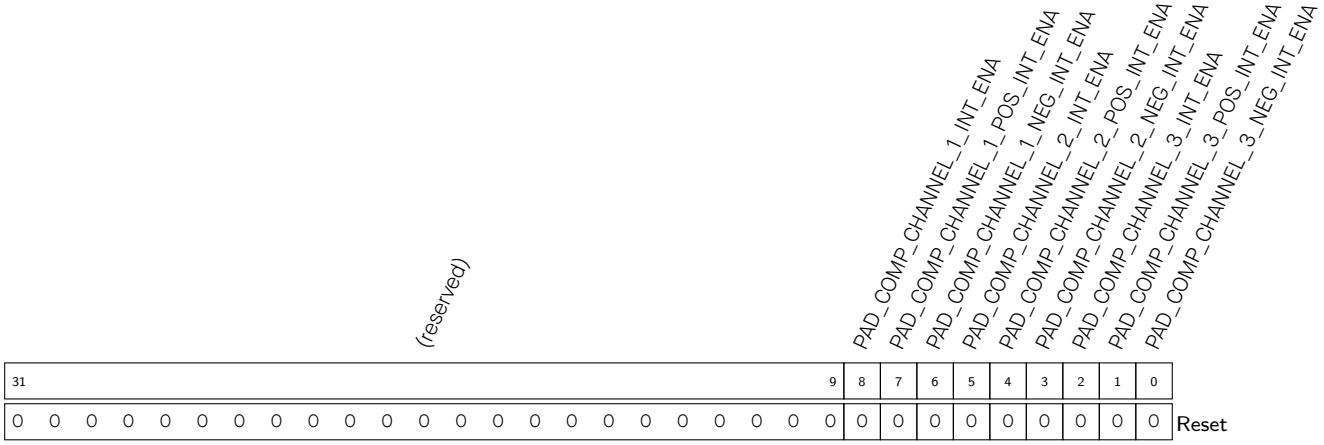
PAD_COMP_DELAY_EVENT_TIME Configures the ETM event delay time.

Unit: PAD_COMP_CLK clock cycle.

Valid only when PAD_COMP_CHANNEL_X_EVENT_TIMER_EN = 1.

(R/W)

Register 29.5. PAD_COMP_INT_ENA_REG (0x0010)



PAD_COMP_CHANNEL_3_NEG_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_3_NEG_INT. (R/W)

PAD_COMP_CHANNEL_3_POS_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_3_POS_INT. (R/W)

PAD_COMP_CHANNEL_3_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_3_INT. (R/W)

PAD_COMP_CHANNEL_2_NEG_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_2_NEG_INT. (R/W)

PAD_COMP_CHANNEL_2_POS_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_2_POS_INT. (R/W)

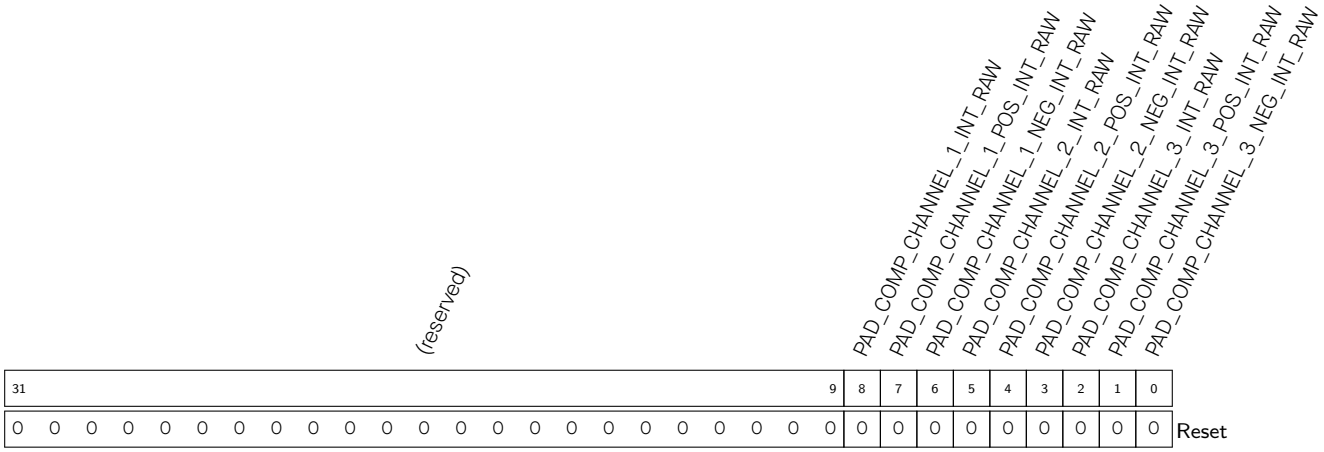
PAD_COMP_CHANNEL_2_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_2_INT. (R/W)

PAD_COMP_CHANNEL_1_NEG_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_1_NEG_INT. (R/W)

PAD_COMP_CHANNEL_1_POS_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_1_POS_INT. (R/W)

PAD_COMP_CHANNEL_1_INT_ENA Write 1 to enable PAD_COMP_CHANNEL_1_INT. (R/W)

Register 29.6. PAD_COMP_INT_RAW_REG (0x0014)



PAD_COMP_CHANNEL_3_NEG_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_3_NEG_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_3_POS_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_3_POS_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_3_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_3_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_2_NEG_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_2_NEG_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_2_POS_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_2_POS_INT. (R/WTC/SS)

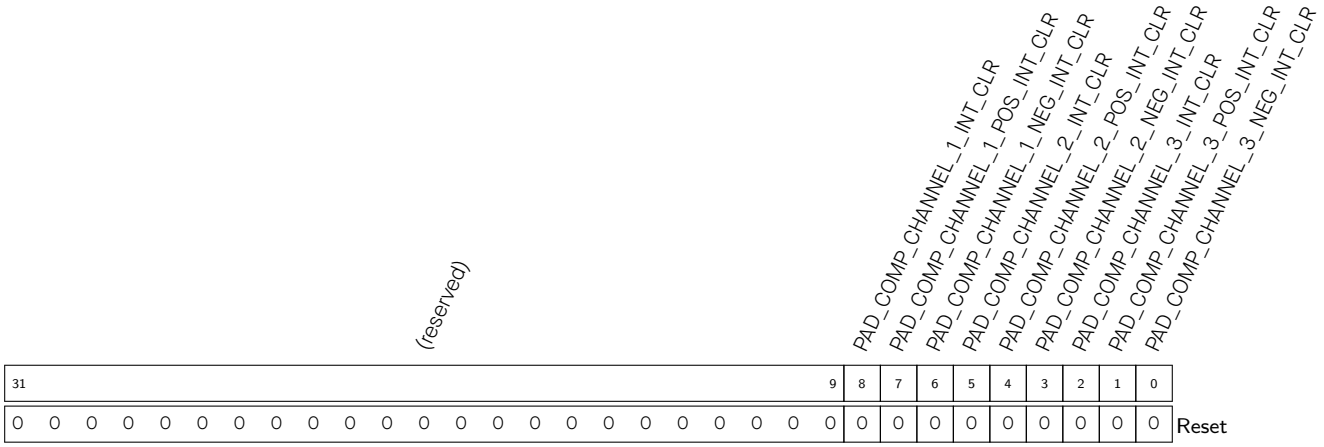
PAD_COMP_CHANNEL_2_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_2_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_1_NEG_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_1_NEG_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_1_POS_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_1_POS_INT. (R/WTC/SS)

PAD_COMP_CHANNEL_1_INT_RAW Raw interrupt status of PAD_COMP_CHANNEL_1_INT. (R/WTC/SS)

Register 29.7. PAD_COMP_INT_CLR_REG (0x0018)



PAD_COMP_CHANNEL_3_NEG_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_3_NEG_INT. (WT)

PAD_COMP_CHANNEL_3_POS_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_3_POS_INT. (WT)

PAD_COMP_CHANNEL_3_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_3_INT. (WT)

PAD_COMP_CHANNEL_2_NEG_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_2_NEG_INT. (WT)

PAD_COMP_CHANNEL_2_POS_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_2_POS_INT. (WT)

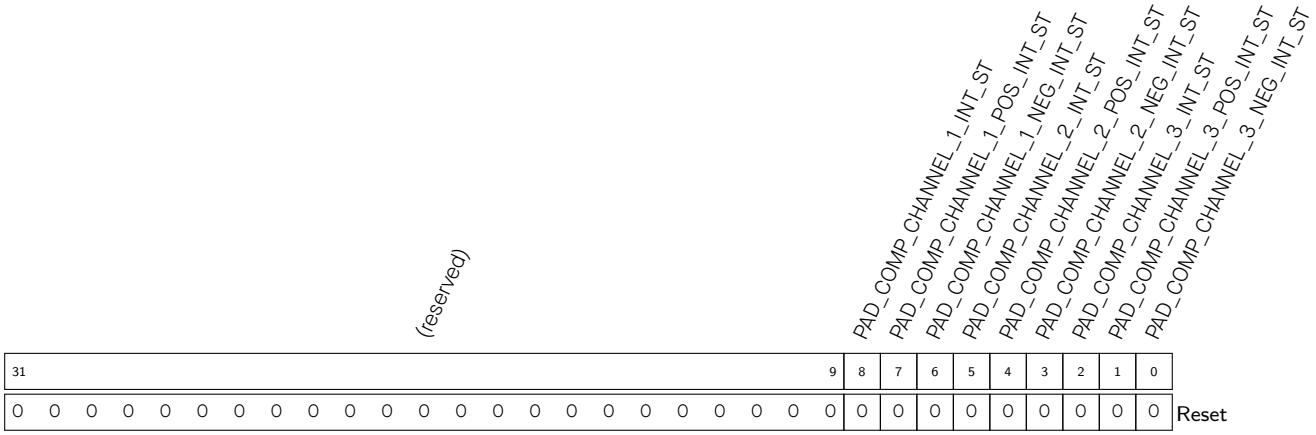
PAD_COMP_CHANNEL_2_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_2_INT. (WT)

PAD_COMP_CHANNEL_1_NEG_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_1_NEG_INT. (WT)

PAD_COMP_CHANNEL_1_POS_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_1_POS_INT. (WT)

PAD_COMP_CHANNEL_1_INT_CLR Write 1 to clear PAD_COMP_CHANNEL_1_INT. (WT)

Register 29.8. PAD_COMP_INT_ST_REG (0x001C)



PAD_COMP_CHANNEL_3_NEG_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_3_NEG_INT. (RO)

PAD_COMP_CHANNEL_3_POS_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_3_POS_INT. (RO)

PAD_COMP_CHANNEL_3_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_3_INT. (RO)

PAD_COMP_CHANNEL_2_NEG_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_2_NEG_INT. (RO)

PAD_COMP_CHANNEL_2_POS_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_2_POS_INT. (RO)

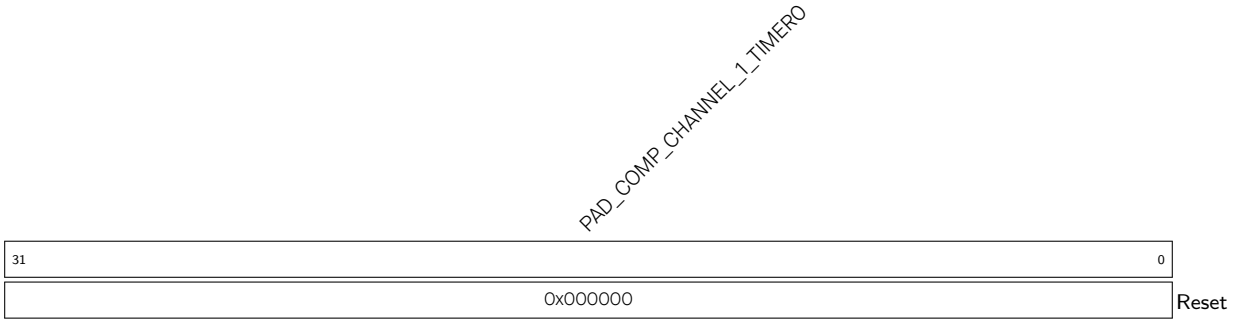
PAD_COMP_CHANNEL_2_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_2_INT. (RO)

PAD_COMP_CHANNEL_1_NEG_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_1_NEG_INT. (RO)

PAD_COMP_CHANNEL_1_POS_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_1_POS_INT. (RO)

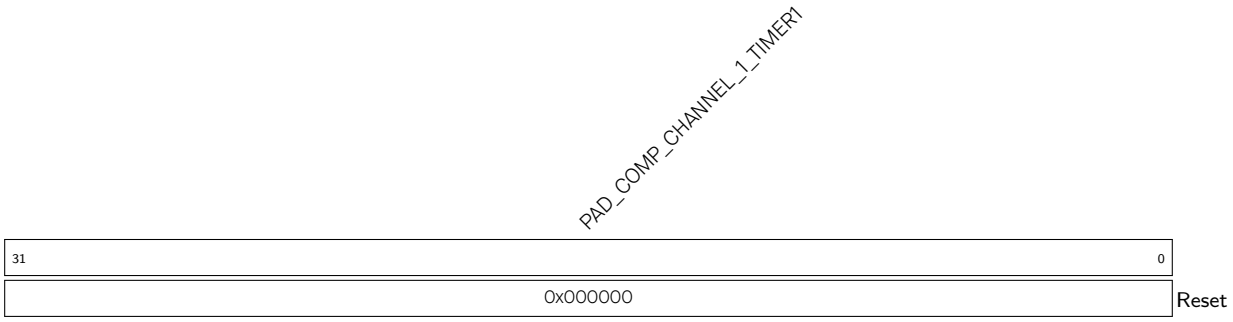
PAD_COMP_CHANNEL_1_INT_ST Masked interrupt status of PAD_COMP_CHANNEL_1_INT. (RO)

Register 29.9. PAD_COMP_CHANNEL_1_TIMER0_REG (0x0020)



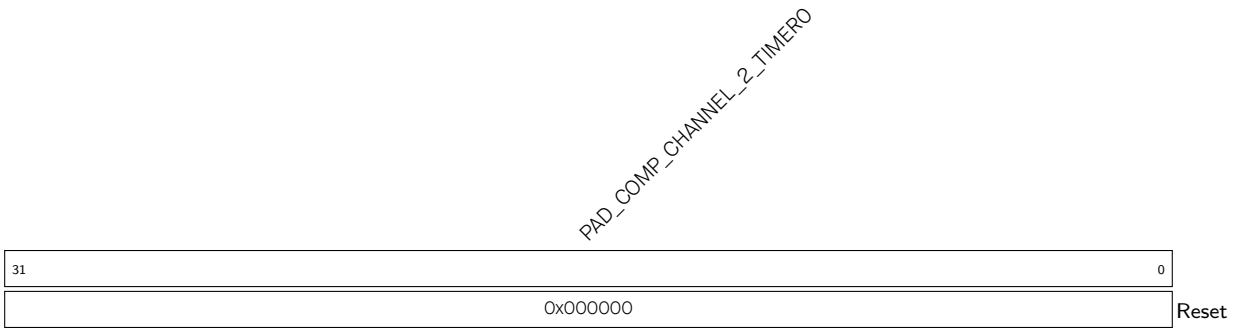
PAD_COMP_CHANNEL_1_TIMER0 Represents the timer value when the current interrupt of channel 1 occurred. (RO)

Register 29.10. PAD_COMP_CHANNEL_1_TIMER1_REG (0x0024)



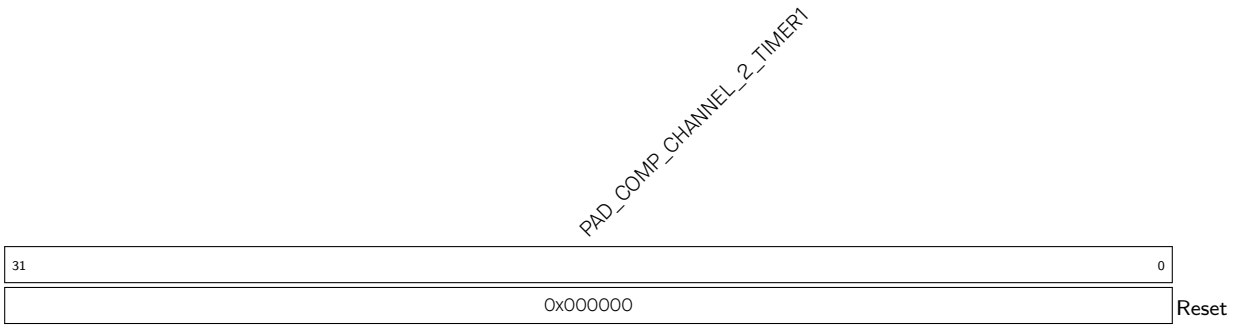
PAD_COMP_CHANNEL_1_TIMER1 Represents the timer value when the previous interrupt of channel 1 occurred. (RO)

Register 29.11. PAD_COMP_CHANNEL_2_TIMER0_REG (0x0028)



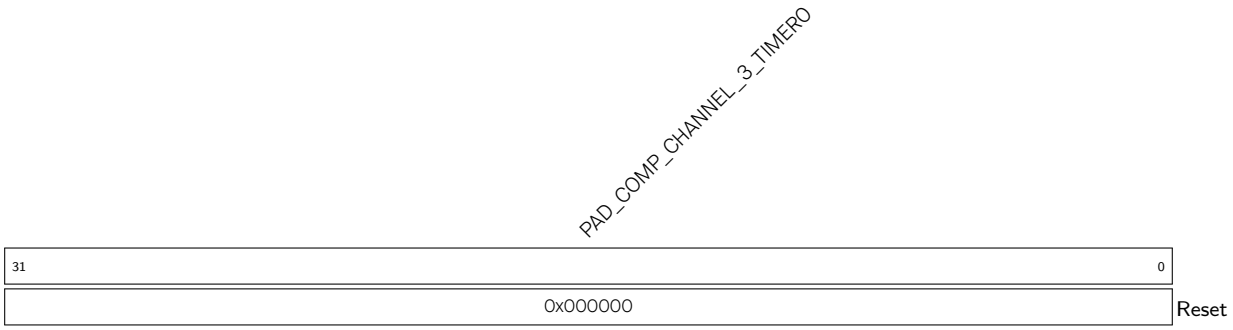
PAD_COMP_CHANNEL_2_TIMER0 Represents the timer value when the current interrupt of channel 2 occurred. (RO)

Register 29.12. PAD_COMP_CHANNEL_2_TIMER1_REG (0x002C)



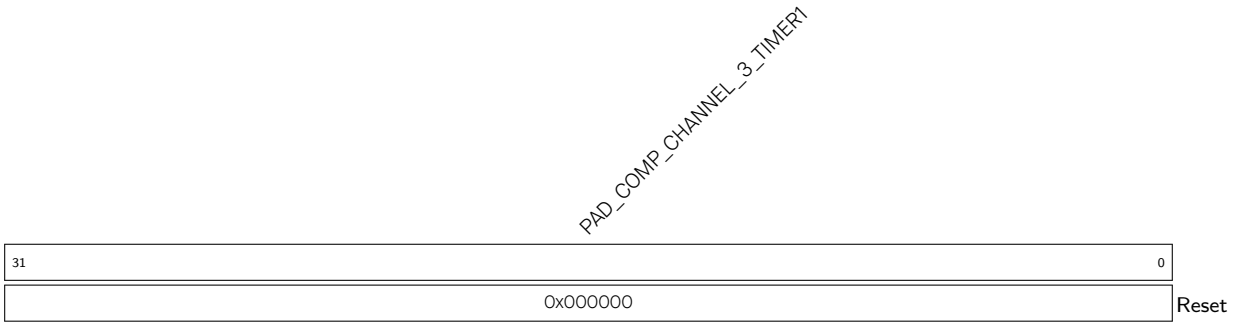
PAD_COMP_CHANNEL_2_TIMER1 Represents the timer value when the previous interrupt of channel 2 occurred. (RO)

Register 29.13. PAD_COMP_CHANNEL_3_TIMER0_REG (0x0030)



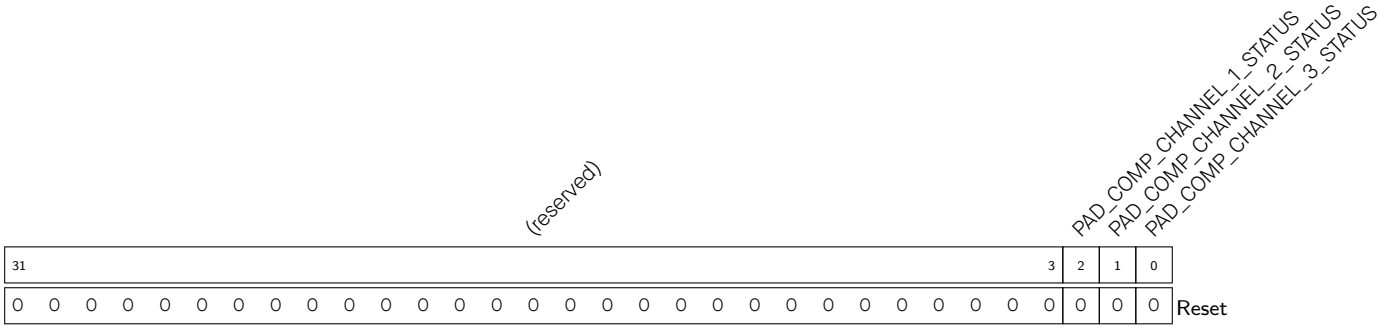
PAD_COMP_CHANNEL_3_TIMER0 Represents the timer value when the current interrupt of channel 3 occurred. (RO)

Register 29.14. PAD_COMP_CHANNEL_3_TIMER1_REG (0x0034)



PAD_COMP_CHANNEL_3_TIMER1 Represents the timer value when the previous interrupt of channel 3 occurred. (RO)

Register 29.15. PAD_COMP_CHANNEL_STATUS_REG (0x0038)



PAD_COMP_CHANNEL_3_STATUS Represents the comparator output status of channel 3.

0: Input voltage < Reference voltage

1: Input voltage > Reference voltage

(RO)

PAD_COMP_CHANNEL_2_STATUS Represents the comparator output status of channel 2.

0: Input voltage < Reference voltage

1: Input voltage > Reference voltage

(RO)

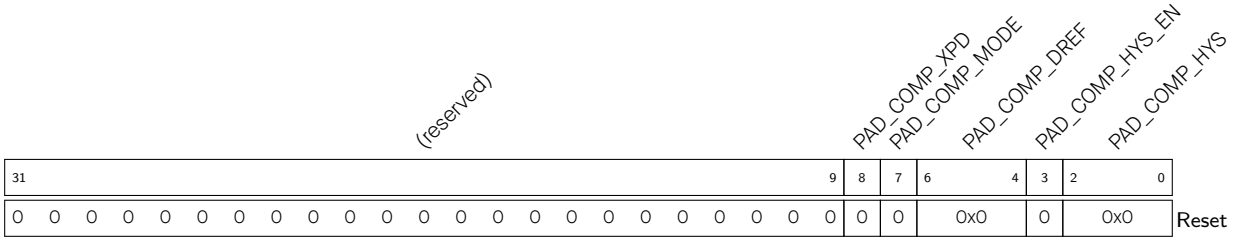
PAD_COMP_CHANNEL_1_STATUS Represents the comparator output status of channel 1.

0: Input voltage < Reference voltage

1: Input voltage > Reference voltage

(RO)

Register 29.16. PAD_COMP_CFG_REG (0x003C)



PAD_COMP_HYS Configures the hysteresis voltage.

0: 0 V

1: 0.04 V

2: 0.08 V

4: 0.12 V

3, 5, 6, 7: Reserved

(R/W)

PAD_COMP_HYS_EN Configures whether to enable the hysteresis function.

0: Disable

1: Enable

Valid only when PAD_COMP_MODE = 0.

(R/W)

PAD_COMP_DREF Configures the internal reference voltage.

The internal reference voltage VREF = PAD_COMP_DREF $\times$ 0.1 $\times$ VDDPST_3 (V).

Valid only when PAD_COMP_MODE = 0.

(R/W)

PAD_COMP_MODE Configures the comparator mode.

0: Internal reference voltage mode

1: External reference voltage mode

(R/W)

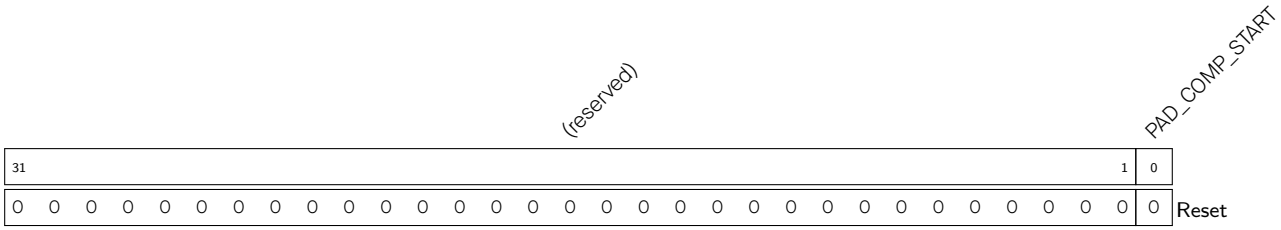
PAD_COMP_XPD Configures the comparator power.

0: Power down

1: Power up

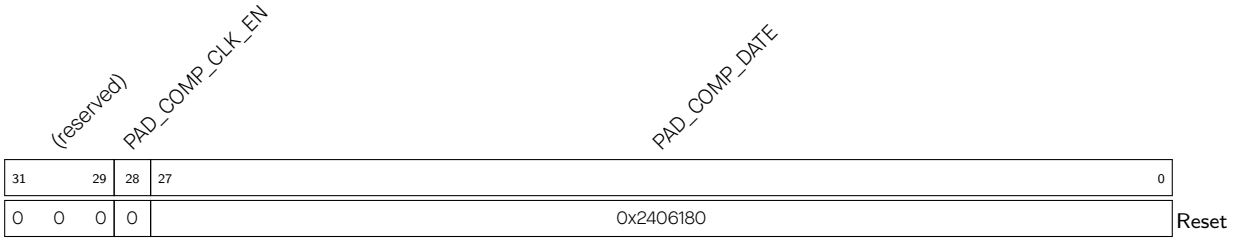
(R/W)

Register 29.17. PAD_COMP_START_REG (0x0040)



PAD_COMP_START Trigger PAD COMP task by software. (WT)

Register 29.18. PAD_COMP_DATE_REG (0x03FC)



PAD_COMP_DATE Version control register. (R/W)

PAD_COMP_CLK_EN Configures whether to enable the register access clock APB_CLK. (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-S31 Series Datasheet* – Specifications of the ESP32-S31 hardware.
- *ESP32-S31 Hardware Design Guidelines* – Guidelines on how to integrate the ESP32-S31 into your hardware product.
- *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-S31 Product/Process Change Notifications (PCN)*
  https://documentation.espressif.com/en/documentList?t=PCN&s=ESP32-S31
- *ESP32-S31 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://documentation.espressif.com/en/documentList?t=Advisory&s=ESP32-S31
- *Documentation Updates and Update Notification Subscription*
  https://documentation.espressif.com/en/documentList?eol=false

Developer Zone

- ESP-IDF Programming Guide for ESP32-S31 – 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-S31 Series SoCs* – Browse through all ESP32-S31 SoCs.
  https://espressif.com/en/products/socs?id=ESP32-S31
- *ESP32-S31 Series Modules* – Browse through all ESP32-S31-based modules.
  https://espressif.com/en/products/modules?id=ESP32-S31
- *ESP32-S31 Series DevKits* – Browse through all ESP32-S31-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32-S31
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector?language=en

Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.
  https://espressif.com/en/contact-us/sales-questions

Glossary

Abbreviations for Peripherals

2D-DMA	2D-DMA Controller
ADC	ADC Controller
AES	AES (Advanced Encryption Standard) Accelerator
ASRC	Audio Sample Rate Converter
CORDIC	CORDIC Accelerator
DAC	DAC Controller
DMA	DMA (Direct Memory Access) Controller
ECC	ECC (Elliptic Curve Cryptography) Accelerator
ECDSA_DS	ECDSA Digital Signature Peripheral
eFuse	eFuse Controller
EMAC	Ethernet Media Access Controller
ETM	Event Task Matrix
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LCD_CAM	LCD and Camera Controller
LEDC	LED PWM (Pulse Width Modulation) Controller
MCPWM	Motor Control PWM (Pulse Width Modulation)
PARLIO	Parallel IO Controller
PCNT	Pulse Count Controller
PMS	Permission Control
PPA	Pixel-Processing Accelerator
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
RSA_DS	RSA Digital Signature Peripheral
SDC	Secure Debug Controller
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
TIMG	Timer Group
TOUCH	Touch Sensor
TRACE	RISC-V Trace Encoder
TSENS	Temperature Sensor
TWAI	Two-Wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
USB OTG	USB On-The-Go
USB_SERIAL_JTAG	USB Serial/JTAG Controller
WDT	Watchdog Timers
XTS_AES	External Memory Encryption and Decryption

Abbreviations Related to Registers

REG **Register**.

SYSREG **System registers** are a group of registers that control system reset, memory, clocks, software interrupts, power management, clock gating, etc.

ISO **Isolation**. If a peripheral or other chip component is powered down, the pins, if any, to which its output signals are routed will go into a floating state. ISO registers isolate such pins and keep them at a certain determined value, so that the other non-powered-down peripherals/devices attached to these pins are not affected.

NMI **Non-maskable interrupt** is a hardware interrupt that cannot be disabled or ignored by the CPU instructions. Such interrupts exist to signal the occurrence of a critical error.

W1TS Abbreviation added to names of registers/fields to indicate that such register/field should be used to set a field in a corresponding register with a similar name. For example, the register `GPIO_ENABLE_W1TS_REG` should be used to set the corresponding fields in the register `GPIO_ENABLE_REG`.

W1TC Same as *W1TS*, but used to clear a field in a corresponding register.

Access Types for Registers

Sections *Register Summary* and *Register Description* in TRM chapters specify access types for registers and their fields.

Most frequently used access types and their combinations are as follows:

- RO
- WO
- WT
- R/W
- R/W1
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

Descriptions of all access types are provided below.

R **Read.** User application can read from this register/field; usually combined with other access types.

RO **Read only.** User application can only read from this register/field.

HRO **Hardware Read Only.** Only hardware can read from this register/field; used for storing default settings for variable parameters.

W **Write.** User application can write to this register/field; usually combined with other access types.

WO **Write only.** User application can only write to this register/field.

W1 **Write Once.** User application can write to this register/field only once; only allowed to write 1; writing 0 is invalid.

SS **Self set.** On a specified event, hardware automatically writes 1 to this register/field; used with 1-bit fields.

SC **Self clear.** On a specified event, hardware automatically writes 0 to this register/field; used with 1-bit and multi-bit fields.

SM **Self modify.** On a specified event, hardware automatically writes a specified value to this register/field; used with multi-bit fields.

SU **Self update.** On a specified event, hardware automatically updates this register/field; used with multi-bit fields.

RS **Read to set.** If user application reads from this register/field, hardware automatically writes 1 to it.

RC **Read to clear.** If user application reads from this register/field, hardware automatically writes 0 to it.

RF **Read from FIFO.** If user application writes new data to FIFO, the register/field automatically reads it.

WF **Write to FIFO.** If user application writes new data to this register/field, it automatically passes the data to FIFO via APB bus.

WS **Write any value to set.** If user application writes to this register/field, hardware automatically sets this register/field.

W1S **Write 1 to set.** If user application writes 1 to this register/field, hardware automatically sets this register/field.

W0S **Write 0 to set.** If user application writes 0 to this register/field, hardware automatically sets this register/field.

WC **Write any value to clear.** If user application writes to this register/field, hardware automatically clears this register/field.

W1C **Write 1 to clear.** If user application writes 1 to this register/field, hardware automatically clears this register/field.

W0C **Write 0 to clear.** If user application writes 0 to this register/field, hardware automatically clears this register/field.

WT **Write 1 to trigger an event.** If user application writes 1 to this field, this action triggers an event (pulse in the APB bus) or clears a corresponding WTC field (see WTC).

WTC **Write to clear.** Hardware automatically clears this field if user application writes 1 to the corresponding WT field (see WT).

W1T **Write 1 to toggle.** If user application writes 1 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

W0T **Write 0 to toggle.** If user application writes 0 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

WL **Write if a lock is deactivated.** If the lock is deactivated, user application can write to this register/field.

varies **The access type varies.** Different fields of this register might have different access types.

Programming Reserved Register Field

Introduction

A field in a register is reserved if the field is not open to users, or produces unpredictable results if configured to values other than defaults.

Programming Reserved Register Field

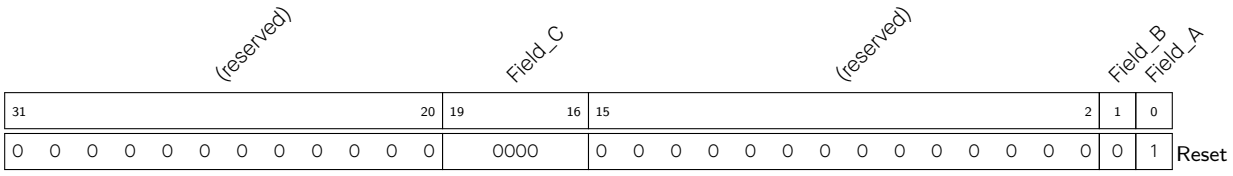
The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

1. Read the value of the register, modify only the fields you want to configure and then write back the value so that reserved fields are untouched.

   OR

2. Modify only the fields you want to configure and write back the default value of the reserved fields. The default value of a field is provided in the "Reset" line of a register diagram. For example, the default value of Field_A in Register X is 1.

Register 29.19. 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 29.9-4.
- **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 29.9-4. 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
2026-09-21	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