ESP32-P4

Technical Reference Manual Pre-release v0.1

PRELIMINARY
CONFIDENTIAL

ESPRESSIF

About This Document

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

Navigation in This Document

Here are some tips on navigation through this extensive document:

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

Release Status at a Glance

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

No.	Chapter	Progress
Part I. Microprocessor and Master		
1	High-Performance CPU [to be added later]	19%
2	RISC-V Trace Encoder (TRACE)	Published
3	Processor Instruction Extensions [to be added later]	19%
4	Low-Power CPU	Published
Part II. System DMA		
5	GDMA Controller (GDMA-AHB, GDMA-AXI)	Published
6	DW-GDMA Controller [to be added later]	48%
7	2D-DMA Controller (2D-DMA)	Published
Part III. Memory Organization		
8	System and Memory	Published
9	eFuse Controller	Published
Part IV. System Component		
10	GPIO Matrix and IO MUX	Published
11	Reset and Clock	Published
12	Chip Boot Control	Published
13	Interrupt Matrix	Published
14	Event Task Matrix (ETM)	Published
15	Low-Power Management [to be added later]	0%
16	System Timer	Published
17	Timer Group (TIMG)	Published
18	Watchdog Timers (WDT)	Published
19	RTC Timer	Published
20	Permission Control (PMS)	Published
21	System Registers [to be added later]	19%
22	Debug Assistant	Published
23	LP Mailbox	Published
24	Brown-out Detector	Published
Part V. Cryptography/Security Component		
25	AES Accelerator (AES)	Published
26	ECC Accelerator (ECC)	Published
27	HMAC Accelerator (HMAC)	Published
28	RSA Accelerator (RSA)	Published
29	SHA Accelerator (SHA)	Published
30	Digital Signature Algorithm (DSA)	Published
31	Elliptic Curve Digital Signature Algorithm (ECDSA)	Published
32	External Memory Encryption and Decryption [to be added later]	80%
33	Random Number Generator (RNG)	Published
34	Key Manager [to be added later]	48%
Part VI. Image and Voice Processing		

No.	Chapter	Progress
35	JPEG Codec	Published
36	Image Signal Processor [to be added later]	48%
37	Pixel-Processing Accelerator (PPA)	Published
38	LCD and Camera Controller (LCD_CAM)	Published
39	H264 Encoder [to be added later]	48%
40	MIPI CSI [to be added later]	24%
41	MIPI DSI [to be added later]	19%
42	Voice Activity Detection [to be added later]	30%
Part VII. Connectivity Interface		
43	UART Controller (UART)	Published
44	SPI Controller [to be added later]	27%
45	I2C Controller (I2C)	Published
46	Analog I2C Controller	Published
47	I3C Controller [to be added later]	19%
48	I2S Controller (I2S)	Published
49	LP I2S Controller [to be added later]	30%
50	Pulse Count Controller (PCNT)	Published
51	USB 2.0 OTG High-Speed [to be added later]	27%
52	USB 2.0 OTG Full-Speed [to be added later]	36%
53	USB Serial/JTAG Controller (USB_SERIAL_JTAG)	Published
54	Ethernet Media Access Controller (EMAC)	Published
55	Two-Wire Automotive Interface (TWAI)	Published
56	SD/MMC Host Controller (SDHOST)	Published
57	LED PWM Controller (LEDC)	Published
58	Motor Control PWM (MCPWM)	Published
59	Remote Control Peripheral (RMT)	Published
60	Parallel IO Controller (PARLIO)	Published
61	Bit-scrambler [to be added later]	48%
Part VIII. Analog Signal Processing		
62	Touch Sensor [to be added later]	65%
63	Temperature Sensor (TSENS)	Published
64	ADC Controller (ADC)	Published
65	Analog Voltage Comparator	Published

Note:

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

Contents

I Microprocessor and Master 43

1 RISC-V Trace Encoder (TRACE) 44
1.1 Terminology 44
1.2 Introduction 45
1.3 Features 46
1.4 Architectural Overview 47
1.5 Functional Description 47
1.5.1 Synchronization 47
1.5.2 Address Mode 48
1.5.3 Optional Sideband Signals 48
1.5.4 Filtering 49
1.5.5 Anchor Tag 49
1.5.6 Memory Writing Mode 50
1.5.7 Automatic Restart 50
1.6 Encoder Output Packets 50
1.6.1 Header 50
1.6.2 Index 51
1.6.3 Payload 51
1.6.3.1 Format 3 Packets 51
1.6.3.2 Format 2 Packets 53
1.6.3.3 Format 1 Packets 54
1.7 Interrupt 55
1.8 Programming Procedures 56
1.8.1 Encoder Option Configuration 56
1.8.2 Filter Configuration 57
1.8.3 Enable Encoder 58
1.8.4 Disable Encoder 59
1.8.5 Notify 59
1.8.6 Decode Data Packets 60
1.8.7 AHB Configuration 60
1.8.8 Software Retention 61
1.9 Register Summary 62
1.10 Registers 63

2 Low-Power CPU 74
2.1 Overview 74
2.2 Features 75
2.3 Configuration and Status Registers (CSRs) 75
2.3.1 Register Summary 75

2.3.2 Registers 76
2.4 Interrupts and Exceptions 83
2.4.1 Interrupts 84
2.4.2 Interrupt Handling 84
2.4.3 Exceptions 85
2.5 Debugging 85
2.5.1 Features 85
2.5.2 Functional Description 86
2.5.3 Register Summary 86
2.5.4 Registers 86
2.6 Hardware Trigger 88
2.6.1 Features 88
2.6.2 Functional Description 88
2.6.3 Trigger Execution Flow 89
2.6.4 Register Summary 89
2.6.5 Registers 89
2.7 Performance Counter 92
2.8 System Access 93
2.8.1 Memory Access 93
2.8.2 Peripheral Access 93
2.9 Event Task Matrix Feature 93
2.10 Sleep and Wake-Up Process 94
2.10.1 Features 94
2.10.2 Process 94
2.10.3 Wake-Up Sources 96
2.10.4 Sleep Rejection 96

II System DMA 97

3 GDMA Controller (GDMA-AHB, GDMA-AXI) 98
3.1 Overview 98
3.2 Features 99
3.3 Architecture 100
3.4 Functional Description 101
3.4.1 Linked List 101
3.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 102
3.4.3 Memory-to-Memory Data Transfer 103
3.4.4 Enabling GDMA 103
3.4.5 Linked List Reading Process 104
3.4.6 EOF 105
3.4.7 Accessing Memory 105
3.4.8 Arbitration 106
3.4.9 CRC Calculation 106
3.5 Event Task Matrix Feature 108
3.6 Interrupts 109
3.7 Programming Procedures 110

3.7.1 Programming Procedures for GDMA's Transmit Channel 110
3.7.2 Programming Procedures for GDMA's Receive Channel 111
3.7.3 Programming Procedures for Memory-to-Memory Transfer 111
3.7.4 Programming Procedures for Channel Priority and Weight 111
3.7.5 Programming Procedures for CRC Calculation 112
3.8 Register Summary 113
3.8.1 GDMA-AHB Register Summary 113
3.8.2 GDMA-AXI Register Summary 120
3.9 Registers 129
3.9.1 GDMA-AHB Registers 129
3.9.2 GDMA-AXI Registers 163

4 2D-DMA Controller (2D-DMA) 206

4.1 Overview 206
4.2 Features 206
4.3 Architecture 206
4.4 Functional Description 207
4.4.1 Transfer Mode 207
4.4.2 Linked List 208
4.4.3 Padding in 2D-MOD1 Mode and DSCR-PORT Mode 211
4.4.4 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer 212
4.4.5 Memory-to-Memory Data Transfer 213
4.4.6 Macroblock Reordering 213
4.4.6.1 Macroblock Reordering in TX Direction 213
4.4.6.2 Macroblock Reordering in RX Direction 214
4.4.7 Color Space Conversion 215
4.4.8 Enabling 2D-DMA 218
4.4.9 Linked List Reading Process 219
4.4.10 EOF 219
4.4.11 Accessing Memory 220
4.4.12 Arbitration 220
4.5 Event Task Matrix Feature 221
4.6 Interrupts 222
4.7 Programming Procedures 224
4.7.1 General Configurations for 2D-DMA 224
4.7.2 Mode-Specific Configurations for 2D-DMA 224
4.7.3 Configurations for 2D-DMA's Transmit Channel 224
4.7.4 Configurations for 2D-DMA's Receive Channel 224
4.7.5 Configurations for Memory-to-Memory Transfer 225
4.7.6 Configurations for Channel Priority and Weight 225
4.7.7 Resetting 2D-DMA While it Runs 226
4.8 Register Summary 227
4.9 Registers 233

III Memory Organization 277

5 System and Memory 278
5.1 Overview 278
5.2 Features 278
5.3 Functional Description 279
5.3.1 Address Mapping 279
5.3.2 Internal Memory 282
5.3.3 External Memory 284
5.3.3.1 External Memory Address Mapping 284
5.3.3.2 Cache 285
5.3.3.3 Cache Operations 286
5.3.4 DMA Address Space 287
5.3.5 Modules/Peripherals Address Mapping 288

6 eFuse Controller 292
6.1 Overview 292
6.2 Features 292
6.3 Functional Description 292
6.3.1 Structure 292
6.3.1.1 EFUSE_WR_DIS 301
6.3.1.2 EFUSE_RD_DIS 301
6.3.1.3 Data Storage 301
6.3.2 Programming of Parameters 303
6.3.3 Reading of Parameters by Users 304
6.3.4 eFuse VDDQ Timing 306
6.3.5 Parameters Used by Hardware Modules 306
6.4 Interrupts 306
6.5 Register Summary 308
6.6 Registers 312

IV System Component 344

7 GPIO Matrix and IO MUX 345
7.1 Overview 345
7.2 Features 345
7.2.1 HP GPIO Matrix and HP IO MUX 345
7.2.2 LP GPIO Matrix and LP IO MUX 346
7.3 Architectural Overview 346
7.4 Peripheral Input via GPIO Matrix 348
7.4.1 Overview 349
7.4.2 Signal Synchronization 349
7.4.3 GPIO Filter 350
7.4.4 Glitch Filter 350
7.4.5 Simple GPIO Input 351
7.4.6 GPIO Wakeup 351
7.4.6.1 HP GPIO Wakeup 351
7.4.6.2 LP GPIO Wakeup 351

7.4.7 Programming Procedure 352
7.4.7.1 HP GPIO Matrix 352
7.4.7.2 LP GPIO Matrix 353
7.5 Peripheral Output via GPIO Matrix 354
7.5.1 Overview 354
7.5.2 Simple GPIO Output 354
7.5.3 Sigma Delta Modulated Output (SDM) 355
7.5.3.1 Functional Description 355
7.5.3.2 SDM Configuration 355
7.5.4 Programming Procedure 356
7.5.4.1 HP GPIO Matrix 356
7.5.4.2 LP GPIO Matrix 357
7.6 Direct Input and Output via IO MUX 357
7.6.1 Overview 357
7.6.2 Functional Description 357
7.6.2.1 HP IO MUX 357
7.6.2.2 LP IO MUX 358
7.7 Analog Functions 358
7.7.1 Overview 358
7.7.2 Analog Functions 358
7.8 Pin Functions in Light-sleep 359
7.9 Pin Hold Feature 359
7.10 Hysteresis Characteristics of GPIO Pins 360
7.11 Power Supplies and Management of GPIO Pins 361
7.11.1 Power Supplies of GPIO Pins 361
7.11.2 Power Supply Management 361
7.12 HP Peripheral Signal List 361
7.13 LP Peripheral Signal List 373
7.14 HP IO MUX Functions List 375
7.15 LP IO MUX Functions List 378
7.16 GPIO Pin Analog Functions List 379
7.17 Event Task Matrix Function 380
7.18 Interrupts 381
7.19 Register Summary 382
7.19.1 HP GPIO Matrix Register Summary 382
7.19.2 HP IO MUX Register Summary 384
7.19.3 GPIO EXT Register Summary 386
7.19.4 LP GPIO Matrix Register Summary 387
7.19.5 LP IO MUX Register Summary 389
7.20 Registers 389
7.20.1 HP GPIO Matrix Registers 390
7.20.2 HP IO MUX Registers 412
7.20.3 GPIO EXT Registers 415
7.20.4 LP GPIO Matrix Registers 431
7.20.5 LP IO MUX Registers 440

8 Reset and Clock 445
8.1 Reset 445
8.1.1 Overview 445
8.1.2 Architectural Overview 445
8.1.3 Features 445
8.1.4 Functional Description 446
8.1.5 Peripheral Reset 448
8.2 Clock 448
8.2.1 Overview 448
8.2.2 Architectural Overview 449
8.2.3 Features 449
8.2.4 Functional Description 450
8.2.4.1 HP System Clock 450
8.2.4.2 LP System Clock 451
8.2.4.3 Peripheral Clocks 452
8.3 Programming Procedures 456
8.3.1 HP System Clock Configuration 456
8.3.2 LP System Clock Configuration 456
8.3.3 Peripheral Clock Reset and Configuration 456
8.4 Register Summary 459
8.4.1 Reset and Clock (HP_SYS_CLKRST) Register Summary 459
8.4.2 LP Always on Clock and Reset (LP_CLKRST) Register Summary 460
8.4.3 LP Peripheral Clock and Reset (LPPERI) Register Summary 461
8.5 Registers 462
8.5.1 Reset and Clock (HP_SYS_CLKRST) Registers 462
8.5.2 LP Always on Clock and Reset (LP_CLKRST) Registers 551
8.5.3 LP Peripheral Clock and Reset (LPPERI) Registers 571

9 Chip Boot Control 581
9.1 Overview 581
9.2 Functional Description 581
9.2.1 Default Configuration 581
9.2.2 Boot Mode Control 582
9.2.3 ROM Messages Printing Control 584
9.2.4 JTAG Signal Source Control 585

10 Interrupt Matrix 586
10.1 Overview 586
10.2 Interrupt Terminology in ESP32-P4 586
10.2.1 Interrupt 586
10.2.2 Interrupt Signal/interrupt Source 586
10.2.3 Interrupt Flow in ESP32-P4 587
10.3 Features 587
10.4 Functional Description 588
10.4.1 Peripheral Interrupt Sources 588
10.4.2 Assign Peripheral Interrupt Source to HP CPU Interrupt 594

10.4.2.1 Assign One Peripheral Interrupt Source (Source_*Y*) to HP CPU*x* 594
10.4.2.2 Assign Multiple Peripheral Interrupt Sources (Source_*Y*) to HP CPU*x* 594
10.4.2.3 Disable HP CPU*x* Peripheral Interrupt Source (Source_*x*) 594
10.4.3 Query Current Interrupt Status of HP CPU*x* Peripheral Interrupt Source 594
10.5 Register Summary 595
10.5.1 HP CPU0 Interrupt Matrix Register Summary 595
10.5.2 HP CPU1 Interrupt Matrix Register Summary 604
10.6 Registers 615
10.6.1 HP CPU0 Interrupt Matrix Registers 615
10.6.2 HP CPU1 Interrupt Matrix Registers 620

11 Event Task Matrix (ETM) 627
11.1 Overview 627
11.2 Features 627
11.3 Functional Description 627
11.3.1 Architecture 627
11.3.2 Events 628
11.3.3 Tasks 633
11.3.4 Event and Task Status 638
11.3.5 Timing Considerations 638
11.3.6 Channel Control 640
11.4 Register Summary 641
11.5 Registers 645

12 System Timer 784
12.1 Overview 784
12.2 Features 784
12.3 System Timer Structure 785
12.4 Clock Source Selection 785
12.5 Functional Description 786
12.5.1 Counter 786
12.5.2 Comparator and Alarm 787
12.5.3 Event Task Matrix 788
12.5.4 Synchronization Operation 788
12.6 Interrupts 789
12.7 Programming Procedure 789
12.7.1 Read Current Count Value 789
12.7.2 Configure a One-Time Alarm in Target Mode 790
12.7.3 Configure Periodic Alarms in Period Mode 790
12.7.4 Update After Light-sleep 790
12.8 Register Summary 792
12.9 Registers 794

13 Timer Group (TIMG) 809
13.1 Overview 809
13.2 Features 809

13.3 Functional Description 810
13.3.1 16-bit Prescaler and Clock Selection 810
13.3.2 54-bit Time-base Counter 811
13.3.3 Alarm Generation 811
13.3.4 Timer Reload 812
13.3.5 Event Task Matrix Feature 813
13.3.6 RTC_SLOW_CLK Frequency Calculation 814
13.3.7 Interrupts 814
13.4 Configuration and Usage 814
13.4.1 Timer as a Simple Clock 814
13.4.2 Timer as One-shot Alarm 815
13.4.3 Timer as Periodic Alarm by APB 815
13.4.4 Timer as Periodic Alarm by ETM 816
13.4.5 RTC_SLOW_CLK Frequency Calculation 816
13.5 Register Summary 818
13.6 Registers 820

14 Watchdog Timers (WDT) 834
14.1 Overview 834
14.2 Digital Watchdog Timers 835
14.2.1 Features 835
14.2.2 Functional Description 836
14.2.2.1 Clock Source and 32-Bit Counter 836
14.2.2.2 Stages and Timeout Actions 837
14.2.2.3 Write Protection 837
14.2.2.4 Flash Boot Protection 838
14.3 Super Watchdog 838
14.3.1 Features 838
14.3.2 Super Watchdog Controller 838
14.3.2.1 Structure 839
14.3.2.2 Workflow 839
14.4 Interrupts 839
14.5 Register Summary 840
14.6 Registers 840

15 RTC Timer 849
15.1 Introduction 849
15.2 Feature List 849
15.3 Functional Description 849
15.4 Event Task Matrix Feature 851
15.5 Interrupts 851
15.6 Register Summary 853
15.7 Registers 854

16 Permission Control (PMS) 863
16.1 Overview 863

16.2 Features 865
16.3 Functional Description 865
16.3.1 Architecture 865
16.3.2 Address Ranges and Permissions Management 866
16.3.2.1 Address Ranges Managed by HP APM and LP APM 866
16.3.2.2 Address Ranges Managed by DMA APM 867
16.4 Programming Procedure 867
16.4.1 Configuring Access Permissions for HP CPU0/1 868
16.4.2 Configuring Access Permissions for LP CPU 868
16.4.3 Configuring Access Permission for DMA Masters 869
16.5 Illegal Access and Interrupts 869
16.6 Register Summary 870
16.6.1 HP_DMA_PMS_REG 870
16.6.2 HP_PERI_PMS_REG 876
16.6.3 LP2HP_PERI_PMS_REG 877
16.6.4 LP_PERI_PMS_REG 877
16.6.5 HP2LP_PERI_PMS_REG 878
16.7 Registers 879
16.7.1 HP_DMA_PMS_REG 879
16.7.2 HP_PERI_PMS_REG 906
16.7.3 LP2HP_PERI_PMS_REG 933
16.7.4 LP_PERI_PMS_REG 946
16.7.5 HP2LP_PERI_PMS_REG 953

17 Debug Assistant 962
17.1 Overview 962
17.2 Features 962
17.3 Functional Description 962
17.3.1 Region Read/Write Monitoring 962
17.3.2 SP Monitoring 963
17.3.3 PC Logging 963
17.3.4 CPU/DMA Bus Access Logging 963
17.4 Interrupts 963
17.5 Recommended Operation 964
17.5.1 Region Monitoring and SP Monitoring Configuration 964
17.5.2 PC Logging Configuration 967
17.5.3 CPU/DMA Bus Access Logging Configuration 967
17.5.3.1 HP CPU0/1 Bus Access Logging Configuration 967
17.5.3.2 DMA Bus Access Logging Configuration 970
17.6 Register Summary 974
17.6.1 HP CPU Bus Logging Configuration Register Summary 974
17.6.2 DMA Bus Logging Configuration Register Summary 974
17.6.3 Summary of Other Registers 975
17.7 Registers 979
17.7.1 HP CPU Bus Logging Configuration Registers 979
17.7.2 DMA Bus Logging Configuration Registers 986

17.7.3 Other Registers 993

18 LP Mailbox 1022
18.1 Overview 1022
18.2 Features 1022
18.3 Functional Description 1023
18.3.1 Message Registers 1023
18.3.2 Interrupts 1023
18.3.3 Inter-Core Communication 1024
18.4 Register Summary 1025
18.5 Registers 1025

19 Brown-out Detector 1039
19.1 Introduction 1039
19.2 Feature List 1039
19.3 Architectural Overview 1040
19.4 Functional Description 1040
19.5 Interrupts 1042
19.6 Programming Procedures 1043
19.7 Register Summary 1044
19.8 Registers 1045

V Cryptography/Security Component 1053

20 AES Accelerator (AES) 1054
20.1 Introduction 1054
20.2 Features 1054
20.3 Clock and Reset 1054
20.4 AES Working Modes 1055
20.5 Typical AES Working Mode 1056
20.5.1 Key, Plaintext, and Ciphertext 1056
20.5.2 Endianness 1056
20.5.3 Operation Process 1059
20.6 DMA-AES Working Mode 1060
20.6.1 Key, Plaintext, and Ciphertext 1060
20.6.2 Endianness 1061
20.6.3 Standard Incrementing Function 1062
20.6.4 Block Number 1062
20.6.5 Initialization Vector 1062
20.6.6 Block Operation Process 1062
20.6.7 GCM Operation Process 1063
20.7 GCM Algorithm 1064
20.7.1 Hash Subkey 1065
20.7.2 J_0 1065
20.7.3 Authentication Tag 1065
20.7.4 AAD Block Number 1066

20.7.5 Number of Effective Bits of Incomplete Blocks 1066
20.8 Interrupts 1066
20.9 Memory Summary 1067
20.10 Register Summary 1068
20.11 Registers 1069

21 ECC Accelerator (ECC) 1075
21.1 Introduction 1075
21.2 Features 1075
21.3 ECC Basics 1075
21.3.1 Elliptic Curve and Points on the Curves 1075
21.3.2 Affine Coordinates and Jacobian Coordinates 1075
21.3.3 Memory Blocks 1076
21.3.4 Data and Data Block 1076
21.3.5 Writing Data 1076
21.3.6 Reading Data 1077
21.3.7 Standard Calculation and Jacobian Calculation 1077
21.4 Function Description 1077
21.4.1 Key Size 1077
21.4.2 Working Modes 1078
21.4.2.1 Affine Point Multiplication (Affine Point Multi) 1078
21.4.2.2 Affine Point Verification (Affine Point Verif) 1078
21.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi) 1079
21.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi) 1079
21.4.2.5 Point Addition (Point Add) 1079
21.4.2.6 Jacobian Point Verification (Jacobian Point Verif) 1080
21.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi) 1080
21.4.2.8 Mod Addition (Mod Add) 1080
21.4.2.9 Mod Subtraction (Mod Sub) 1080
21.4.2.10 Mod Multiplication (Mod Multi) 1081
21.4.2.11 Mod Division (Mod Div) 1081
21.5 Clock and Reset 1081
21.6 Interrupts 1082
21.7 Programming Procedures 1082
21.8 Register Summary 1083
21.9 Registers 1084

22 HMAC Accelerator (HMAC) 1088
22.1 Main Features 1088
22.2 Functional Description 1088
22.2.1 Upstream Mode 1088
22.2.2 Downstream Mode - JTAG Enable Feature 1089
22.2.3 Downstream Mode - Digital Signature Algorithm and Key Derivation Feature 1089
22.2.4 HMAC eFuse Configuration 1090
22.2.5 HMAC Process (Detailed) 1091

22.2.5.1 Enable HMAC Module 1091
22.2.5.2 Configure HMAC Keys and Key Purposes 1091
22.2.5.3 Downstream Mode Process 1091
22.2.5.4 Upstream Mode Process 1092
22.3 HMAC Algorithm Details 1093
22.3.1 Padding Bits 1093
22.3.2 HMAC Algorithm Structure 1093
22.4 Register Summary 1095
22.5 Registers 1097

23 RSA Accelerator (RSA) 1104
23.1 Introduction 1104
23.2 Features 1104
23.3 Functional Description 1104
23.3.1 Definitions and Representations 1105
23.3.2 Large-Number Modular Exponentiation 1105
23.3.3 Large-Number Modular Multiplication 1106
23.3.4 Large-Number Multiplication 1107
23.3.5 Options for Additional Acceleration 1107
23.4 Interrupts 1109
23.5 Memory Summary 1110
23.6 Register Summary 1110
23.7 Registers 1111

24 SHA Accelerator (SHA) 1115
24.1 Introduction 1115
24.2 Features 1115
24.3 Working Modes 1115
24.4 Function Description 1116
24.4.1 Preprocessing 1116
24.4.1.1 Padding the Message 1117
24.4.1.2 Parsing the Message 1117
24.4.1.3 Setting the Initial Hash Value 1118
24.4.2 Hash Operation 1119
24.4.2.1 Typical SHA Mode Process 1119
24.4.2.2 DMA-SHA Mode Process 1121
24.4.3 Message Digest 1122
24.5 Interrupt 1123
24.6 Register Summary 1125
24.7 Registers 1127

25 Digital Signature Algorithm (DSA) 1131
25.1 Overview 1131
25.2 Features 1131
25.3 Functional Description 1131
25.3.1 Overview 1131

25.3.2 Private Key Operands 1132
25.3.3 Software Prerequisites 1132
25.3.4 DSA Operation at the Hardware Level 1134
25.3.5 DSA Operation at the Software Level 1134
25.4 Memory Summary 1137
25.5 Register Summary 1138
25.6 Registers 1139

26 Elliptic Curve Digital Signature Algorithm (ECDSA) 1142
26.1 Introduction 1142
26.2 Features 1142
26.3 ECDSA Basics 1142
26.3.1 Domain Parameters 1142
26.3.2 Key Generation 1143
26.3.3 Signature Generation 1143
26.3.4 Signature Verification 1144
26.4 Functional Description 1144
26.4.1 ECDSA Working Modes 1144
26.4.2 Data and Data Block 1145
26.4.2.1 Writing Data 1146
26.4.2.2 Reading Data 1146
26.4.2.3 Padding the Message 1146
26.4.2.4 Parsing the Message 1147
26.4.3 Security Features 1147
26.4.3.1 Dynamic Access Permission 1147
26.4.3.2 Hardware Occupation 1147
26.5 Programming Procedures 1148
26.5.1 ECDSA Process 1148
26.5.1.1 IDLE Stage 1148
26.5.1.2 PREP Stage 1149
26.5.1.3 LOAD Stage 1149
26.5.1.4 PROC Stage 1149
26.5.1.5 GAIN Stage 1150
26.5.1.6 POST Stage 1150
26.5.1.7 ECDSA SHA Interface 1150
26.5.2 Clocks and Resets 1151
26.5.3 Interrupts 1151
26.6 Memory Blocks 1153
26.7 Register Summary 1154
26.8 Registers 1155

27 Random Number Generator (RNG) 1162
27.1 Introduction 1162
27.2 Feature List 1162
27.3 Functional Description 1162
27.4 Programming Procedure 1163

27.5 Register Summary 1164
27.6 Registers 1165

VI Image and Voice Processing 1166

28 JPEG Codec 1167
28.1 Terminology 1167
28.2 Introduction 1168
28.3 Features 1169
28.4 Architectural Overview 1170
28.5 Functional Description 1171
28.5.1 JPEG Encoder 1172
28.5.1.1 Pause 1172
28.5.1.2 Color Space Conversion 1172
28.5.1.3 Configurable Quantization Coefficient Table 1172
28.5.1.4 Stuffed Zero Byte 1174
28.5.1.5 EOI Marker 1174
28.5.2 JPEG Decoder 1175
28.5.2.1 Multiple Chrominance Components 1175
28.5.2.2 Parsing RST Marker 1176
28.5.2.3 Configurable Quantization Coefficient Table 1176
28.5.2.4 Configurable Huffman Table 1176
28.5.2.5 Timeout Detection 1178
28.6 Interrupts 1179
28.7 Programming Procedures 1180
28.7.1 JPEG Encoder 1180
28.7.2 JPEG Decoder 1183
28.7.3 Reset 1187
28.8 Register Summary 1188
28.9 Registers 1189

29 Pixel-Processing Accelerator (PPA) 1210
29.1 Overview 1210
29.2 Terminology 1210
29.3 Features 1210
29.4 Architectural Overview 1211
29.5 Functional Description 1212
29.5.1 PPA Color Spaces 1212
29.5.1.1 SRM Color Space 1213
29.5.1.2 BLEND Color Space 1215
29.5.2 PPA 2D-DMA Linked List Configuration 1216
29.5.2.1 SRM Special Configuration 1217
29.5.3 Scaling - Rotation - Mirroring (SRM) 1217
29.5.3.1 Basic Functionality 1217
29.5.3.2 Pixel Block Rearrangement 1218
29.5.4 Layer Blending (BLEND) 1219

29.5.4.1 Basic Functionality 1219
29.5.4.2 Filled Image Output 1220
29.6 Interrupts 1220
29.7 Programming Procedures 1221
29.7.1 PPA Clock Reset Configuration 1221
29.7.2 SRM Configuration 1221
29.7.3 BLEND CLUT Configuration 1222
29.7.4 BLEND Configuration 1223
29.7.5 BLEND Image Filling Configuration 1223
29.7.6 Error Handling 1224
29.8 Register Summary 1225
29.9 Registers 1227

30 LCD and Camera Controller (LCD_CAM) 1249
30.1 Overview 1249
30.2 Features 1249
30.3 Functional Description 1250
30.3.1 Block Diagram 1250
30.3.2 Signal Description 1250
30.3.3 LCD_CAM Module Clocks 1251
30.3.3.1 LCD Clock 1251
30.3.3.2 Camera Clock 1253
30.3.4 LCD_CAM Reset 1254
30.3.5 LCD_CAM Data Format Control 1254
30.3.5.1 LCD Data Format Control 1254
30.3.5.2 Camera Data Format Control 1257
30.3.6 YUV-RGB Data Format Conversion 1258
30.3.6.1 YUV Formats 1258
30.3.6.2 Format Conversion Configuration 1259
30.3.7 LCD_CAM Timing 1261
30.3.7.1 LCD Timing (RGB Format) 1261
30.3.7.2 LCD Timing (I8080/MOTO6800 Format) 1262
30.4 Interrupts 1262
30.5 Software Configuration Process 1263
30.5.1 Configure LCD (RGB Format) as TX Mode 1263
30.5.2 Configure LCD (I8080/MOTO6800 Format) as TX Mode 1263
30.5.3 Configure Camera as RX Mode 1265
30.6 Register Summary 1266
30.7 Registers 1267

VII Connectivity Interface 1285

31 UART Controller (UART) 1286
31.1 Overview 1286
31.2 Features 1286
31.3 UART Structure 1287

31.4 Functional Description 1288
31.4.1 Clock and Reset 1288
31.4.2 UART FIFO 1289
31.4.3 Baud Rate Generation and Detection 1289
31.4.3.1 Baud Rate Generation 1289
31.4.3.2 Baud Rate Detection 1290
31.4.4 UART Data Frame 1291
31.4.5 AT_CMD Character Structure 1292
31.4.6 RS485 1292
31.4.6.1 Driver Control 1292
31.4.6.2 Turnaround Delay 1293
31.4.6.3 Bus Snooping 1293
31.4.7 IrDA 1293
31.4.8 Wakeup 1294
31.4.9 Flow Control 1295
31.4.9.1 Hardware Flow Control 1296
31.4.9.2 Software Flow Control 1297
31.4.10 GDMA Mode 1298
31.4.11 Interrupts 1299
31.5 Programming Procedures 1300
31.5.1 Register Type 1300
31.5.2 Detailed Steps 1301
31.5.2.1 Initializing UART*n* 1301
31.5.2.2 Configuring UART*n* Communication 1301
31.5.2.3 Enabling UART*n* 1302
31.6 Register Summary 1303
31.6.1 UART Register Summary 1303
31.6.2 LP UART Register Summary 1304
31.6.3 UHCI Register Summary 1305
31.7 Registers 1307
31.7.1 UART Registers 1307
31.7.2 LP UART Registers 1330
31.7.3 UHCI Registers 1350

32 I2C Controller (I2C) 1372
32.1 Overview 1372
32.2 Features 1372
32.3 I2C Architecture 1373
32.4 Functional Description 1375
32.4.1 Clock Configuration 1375
32.4.2 SCL and SDA Noise Filtering 1376
32.4.3 SCL Clock Stretching 1376
32.4.4 Generating SCL Pulses in Idle State 1377
32.4.5 Synchronization 1377
32.4.6 Open-Drain Output 1378
32.4.7 Timing Parameter Configuration 1379

32.4.8 Timeout Control 1381
32.4.9 Command Configuration 1381
32.4.10 TX/RX RAM Data Storage 1382
32.4.11 Data Conversion 1383
32.4.12 Addressing Mode 1383
32.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode 1384
32.4.14 To Start the I2C Controller 1384
32.5 Functional Differences Between LP_I2C and I2C 1384
32.6 Programming Example 1385
32.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence 1385
32.6.1.1 Introduction 1385
32.6.1.2 Configuration Example 1386
32.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence 1387
32.6.2.1 Introduction 1387
32.6.2.2 Configuration Example 1387
32.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 1388
32.6.3.1 Introduction 1389
32.6.3.2 Configuration Example 1389
32.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 1390
32.6.4.1 Introduction 1391
32.6.4.2 Configuration Example 1392
32.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence 1393
32.6.5.1 Introduction 1393
32.6.5.2 Configuration Example 1394
32.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence 1395
32.6.6.1 Introduction 1396
32.6.6.2 Configuration Example 1396
32.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence 1398
32.6.7.1 Introduction 1398
32.6.7.2 Configuration Example 1399
32.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences 1400
32.6.8.1 Introduction 1401
32.6.8.2 Configuration Example 1402
32.7 Interrupts 1404
32.8 Register Summary 1406
32.8.1 I2C Register Summary 1406
32.8.2 LP_I2C Register Summary 1407
32.9 Registers 1409
32.9.1 I2C Registers 1409
32.9.2 LP_I2C Registers 1432

33 Analog I2C Controller 1454
33.1 Introduction 1454
33.2 Feature List 1454
33.3 Architectural Overview 1455
33.4 Functional Description 1455

33.5 Programming Procedures 1456
33.6 Register Summary 1457
33.7 Registers 1458

34 I2S Controller (I2S) 1462
34.1 Overview 1462
34.2 Terminology 1462
34.3 Features 1463
34.4 System Architecture 1464
34.5 Supported Audio Standards 1466
34.5.1 TDM Philips Standard 1467
34.5.2 TDM MSB Alignment Standard 1467
34.5.3 TDM PCM Standard 1468
34.5.4 PDM Standard 1468
34.6 I2S TX/RX Clock 1469
34.7 I2S Reset 1471
34.8 I2S Master/Slave Mode 1471
34.8.1 Master/Slave TX Mode 1471
34.8.2 Master/Slave RX Mode 1472
34.9 Transmitting Data 1472
34.9.1 Data Format Control 1473
34.9.1.1 Bit Width Control of Channel Valid Data 1473
34.9.1.2 Endian Control of Channel Valid Data 1473
34.9.1.3 A-law/μ-law Compression and Decompression 1474
34.9.1.4 Bit Width Control of Channel TX Data 1474
34.9.1.5 Bit Order Control of Channel Data 1474
34.9.2 Channel Mode Control 1475
34.9.2.1 TDM TX Mode 1475
34.9.2.2 PDM TX Mode 1476
34.9.3 Synchronous Counter 1479
34.10 Receiving Data 1480
34.10.1 Channel Mode Control 1480
34.10.1.1 TDM RX Mode 1480
34.10.1.2 PDM RX Mode 1480
34.10.2 Data Format Control 1481
34.10.2.1 Bit Order Control of Channel Data 1481
34.10.2.2 Bit Width Control of Channel Storage (Valid) Data 1482
34.10.2.3 Bit Width Control of Channel RX Data 1482
34.10.2.4 Endian Control of Channel Storage Data 1482
34.10.2.5 A-law/μ-law Compression and Decompression 1482
34.11 Event Task Matrix Feature 1483
34.12 I2S Interrupts 1483
34.13 Software Configuration Process 1484
34.13.1 Configure I2S as TX Mode 1484
34.13.2 Configure I2S as RX Mode 1485
34.14 Register Summary 1486

34.15 Registers 1488

35 Pulse Count Controller (PCNT) 1508
35.1 Introduction 1508
35.2 Feature List 1508
35.3 Architectural Overview 1509
35.4 Functional Description 1510
35.5 Interrupts 1511
35.6 Programming Procedures 1512
35.6.1 Channel 0 Incrementing Independently 1512
35.6.2 Channel 0 Decrementing Independently 1513
35.6.3 Channel 0 and Channel 1 Incrementing Together 1513
35.7 Register Summary 1515
35.8 Registers 1516

36 USB Serial/JTAG Controller (USB_SERIAL_JTAG) 1524
36.1 Overview 1524
36.2 Features 1524
36.3 Functional Description 1526
36.3.1 CDC-ACM USB Interface Functional Description 1526
36.3.2 CDC-ACM Firmware Interface Functional Description 1527
36.3.3 USB-to-JTAG Interface: JTAG Command Processor 1528
36.3.4 USB-to-JTAG Interface: CMD_REP Usage Example 1529
36.3.5 USB-to-JTAG Interface: Response Capture Unit 1530
36.3.6 USB-to-JTAG Interface: Control Transfer Requests 1530
36.4 Recommended Operation 1531
36.5 Interrupts 1532
36.6 Register Summary 1534
36.7 Registers 1536

37 Ethernet Media Access Controller (EMAC) 1560
37.1 Overview 1560
37.2 Features 1560
37.3 Ethernet MAC Architecture 1562
37.4 Functional Description 1562
37.4.1 EMAC_CORE 1562
37.4.1.1 Transmission 1562
37.4.1.2 Reception 1564
37.4.2 EMAC_MTL (MAC Transaction Layer) 1565
37.4.3 EMAC_DMA 1566
37.4.3.1 Transmit Descriptors 1566
37.4.3.2 Receive Descriptors 1571
37.4.4 PHY Interface 1577
37.4.4.1 Media Independent Interface: MII 1577
37.4.4.2 Reduced Media Independent Interface: RMII 1579
37.4.4.3 Station Management Agent (SMA) Interface 1581

37.4.5 MAC Address Filtering 1582
37.4.5.1 Unicast Destination Address Filtering 1582
37.4.5.2 Multicast Destination Address Filtering 1582
37.4.5.3 Broadcast Address Filter 1582
37.4.5.4 Unicast Source Address Filter 1582
37.4.5.5 Inverse Filtering Operation 1583
37.4.6 Energy Efficient Ethernet (EEE) 1584
37.4.6.1 Transmission 1584
37.4.6.2 Reception 1584
37.4.7 Source Address, VLAN, and CRC Control 1584
37.4.7.1 Source Address Control 1585
37.4.7.2 VLAN Control 1585
37.4.7.3 CRC Control 1585
37.4.8 Time Stamp 1585
37.4.8.1 Transmit Path 1585
37.4.8.2 Receive Path 1585
37.4.9 Remote Wakeup 1586
37.4.10 Good Transmitted and Received Frames 1586
37.5 Programming Procedures 1586
37.5.1 MAC System Layer Configuration 1586
37.5.2 EMAC Initial Configuration 1588
37.5.3 Starting Transmission 1588
37.5.4 Starting Reception 1588
37.5.5 TX Entering and Exiting the LPI State 1589
37.5.6 RX Entering and Exiting the LPI State 1589
37.6 Interrupts 1589
37.7 Register Summary 1591
37.8 Registers 1594

38 Two-Wire Automotive Interface (TWAI) 1641
38.1 Features 1641
38.2 Protocol Overview 1642
38.2.1 TWAI Properties 1642
38.2.2 TWAI Messages 1643
38.2.2.1 Data Frames and Remote Frames 1643
38.2.2.2 Error and Overload Frames 1645
38.2.2.3 Interframe Space 1647
38.2.3 TWAI Errors 1647
38.2.3.1 Error Types 1647
38.2.3.2 Error States 1648
38.2.3.3 Error State Transition 1648
38.2.3.4 Error Counter Rules 1648
38.2.4 TWAI Bit Timing 1649
38.2.4.1 Nominal Bit 1649
38.2.4.2 Hard Synchronization and Resynchronization 1650
38.3 Architectural Overview 1651

38.3.1 Bit Timing Logic 1652
38.3.2 Bit Stream Processor 1652
38.3.3 Acceptance Filter 1652
38.3.4 Receive FIFO 1652
38.3.5 Error Management Logic 1652
38.3.6 Registers Block 1652
38.4 Functional Description 1653
38.4.1 Modes 1653
38.4.1.1 Reset Mode 1653
38.4.1.2 Operation Mode 1654
38.4.2 Bit Timing 1654
38.4.3 Transmit and Receive Buffers 1655
38.4.3.1 Overview of Buffers 1655
38.4.3.2 Frame Information 1656
38.4.3.3 Frame Identifier 1656
38.4.3.4 Frame Data 1657
38.4.4 Receive FIFO and Data Overruns 1657
38.4.5 Acceptance Filter 1658
38.4.5.1 Single-Filter Mode 1659
38.4.5.2 Dual-Filter Mode 1659
38.4.6 Error Management 1661
38.4.6.1 Error Warning Limit 1662
38.4.6.2 Error Passive 1662
38.4.6.3 Bus-Off and Bus-Off Recovery 1662
38.4.7 Error Code Capture 1663
38.4.8 Arbitration Lost Capture 1664
38.4.9 Transceiver Auto-Standby 1665
38.5 Interrupts 1666
38.6 Register Summary 1670
38.7 Registers 1672

39 SD/MMC Host Controller (SDHOST) 1689
39.1 Overview 1689
39.2 Features 1689
39.3 SD/MMC External Interface Signals 1690
39.4 Functional Description 1690
39.4.1 SD/MMC Host Controller Architecture 1690
39.4.1.1 Bus Interface Unit (BIU) 1691
39.4.1.2 Card Interface Unit (CIU) 1691
39.4.2 Command Path 1692
39.4.3 Data Path 1692
39.4.3.1 Data Transmit 1692
39.4.3.2 Data Receive 1693
39.5 Software Restrictions for CIU Operations 1694
39.6 RAM for Receiving and Transmitting Data 1695
39.6.1 TX RAM Module 1695

39.6.2 RX RAM Module 1695
39.7 DMA Linked List 1695
39.8 DMA Descriptor Format 1696
39.9 Programming Procedures 1698
39.9.1 Initializing Registers 1698
39.9.2 Sending Commands 1699
39.9.3 Initializing DMA 1699
39.9.4 Initializing DMA Transmission 1700
39.9.5 Initializing DMA Reception 1700
39.10 Clock Phase Selection 1701
39.11 Interrupt 1701
39.12 Register Summary 1703
39.13 Registers 1705

40 LED PWM Controller (LEDC) 1733
40.1 Overview 1733
40.2 Features 1733
40.3 Architectural Overview 1733
40.4 Functional Description 1734
40.4.1 Timers 1734
40.4.1.1 Clock Source 1735
40.4.1.2 Clock Divider Configuration 1735
40.4.1.3 20-Bit Counter 1736
40.4.2 PWM Generators 1737
40.4.3 Duty Cycle Fading 1738
40.4.3.1 Linear Duty Cycle Fading 1739
40.4.3.2 Gamma Curve Fading 1739
40.4.3.3 Suspend and Resume Duty Cycle Fading 1740
40.4.4 Event Task Matrix Feature 1741
40.5 Interrupts 1742
40.6 Programming Procedures 1742
40.7 Memory Blocks 1744
40.8 Register Summary 1745
40.9 Registers 1747

41 Motor Control PWM (MCPWM) 1759
41.1 Overview 1759
41.2 Features 1759
41.3 Modules 1762
41.3.1 Overview 1762
41.3.1.1 Timer Module 1762
41.3.1.2 Operator Module 1762
41.3.1.3 Capture Module 1763
41.3.1.4 ETM Module 1763
41.3.2 PWM Timer Module 1764
41.3.2.1 Configurations of the PWM Timer Module 1764

41.3.2.2 PWM Timer's Working Modes and Timing Event Generation 1764
41.3.2.3 Shadow Register of PWM Timer 1770
41.3.2.4 PWM Timer Synchronization and Phase Locking 1770
41.3.3 PWM Operator Module 1770
41.3.3.1 PWM Generator Module 1772
41.3.3.2 Dead Time Generator Module 1784
41.3.3.3 PWM Carrier Module 1789
41.3.3.4 Fault Detection Module 1791
41.3.4 Capture Module 1792
41.3.4.1 Introduction 1792
41.3.4.2 Capture Timer 1793
41.3.4.3 Capture Channel 1793
41.3.5 ETM Module 1793
41.3.5.1 Overview 1793
41.3.5.2 MCPWM-Related ETM Events 1793
41.3.5.3 MCPWM-Related ETM Tasks 1794
41.4 Interrupts 1795
41.5 Register Summary 1796
41.6 Registers 1799

42 Remote Control Peripheral (RMT) 1846
42.1 Overview 1846
42.2 Features 1846
42.3 Functional Description 1847
42.3.1 Architecture 1847
42.3.2 RAM 1848
42.3.2.1 Structure of RAM 1848
42.3.2.2 Use of RAM 1848
42.3.2.3 RAM Access 1849
42.3.3 Clock 1850
42.3.4 Transmitter 1850
42.3.4.1 Normal TX Mode 1851
42.3.4.2 Wrap TX Mode 1851
42.3.4.3 TX Modulation 1851
42.3.4.4 Continuous TX Mode 1852
42.3.4.5 Simultaneous TX Mode 1852
42.3.5 Receiver 1852
42.3.5.1 Normal RX Mode 1852
42.3.5.2 Wrap RX Mode 1853
42.3.5.3 RX Filtering 1853
42.3.5.4 RX Demodulation 1853
42.3.6 Configuration Update 1853
42.4 Interrupts 1854
42.5 Register Summary 1856
42.6 Registers 1858

43 Parallel IO Controller (PARLIO) 1874
43.1 Introduction 1874
43.2 Glossary 1874
43.3 Features 1874
43.4 Architectural Overview 1876
43.5 Functional Description 1876
43.5.1 Clock Generator 1876
43.5.2 Clock & Reset Restriction 1877
43.5.3 Master-Slave Mode 1879
43.5.4 Receive Modes of the RX Unit 1880
43.5.4.1 Level Enable Mode 1880
43.5.4.2 Pulse Enable Mode 1880
43.5.4.3 Software Enable Mode 1881
43.5.5 RX Unit GDMA SUC EOF Generation 1882
43.5.6 TX Unit EOF Generation 1882
43.5.7 RX Unit Timeout 1882
43.5.8 Output Clock Gating of TX Unit 1882
43.5.9 Valid Signal Output of TX Unit 1883
43.5.10 Bus Idle Value of TX Unit 1883
43.5.11 Data Transfer in a Single Frame 1883
43.5.12 Bit Reversal in One Byte 1884
43.6 Interrupts 1884
43.7 Programming Procedures 1885
43.7.1 Data Receiving Operation Process 1885
43.7.2 Data Transmitting Operation Process 1885
43.8 Application Examples 1886
43.8.1 Co-working with SPI 1886
43.8.2 Co-working with I2S 1887
43.8.3 Co-working with LCD 1888
43.9 Register Summary 1890
43.10 Registers 1891

VIII Analog Signal Processing 1901

44 Temperature Sensor (TSENS) 1902
44.1 Overview 1902
44.2 Features 1902
44.3 Architecture 1902
44.4 Functional Description 1903
44.4.1 Temperature Sensor Power Up 1903
44.4.2 Temperature Sensor Clock 1903
44.4.3 Wake-Up Modes for Automatic Temperature Monitoring 1903
44.4.4 Temperature Measurement Range and Offset 1904
44.4.5 Data Conversion 1904
44.5 Event Task Matrix Feature 1904

44.6 Interrupts 1905
44.7 Programming Procedure 1905
44.8 Register Summary 1906
44.9 Registers 1907

45 ADC Controller (ADC) 1913
45.1 Overview 1913
45.2 Terminology 1914
45.3 Features 1914
45.4 Architecture 1915
45.5 Functional Description 1916
45.5.1 SAR ADC Power Up 1916
45.5.2 SAR ADC Channels 1916
45.5.3 SAR ADC Clock 1917
45.5.4 SAR ADC Conversion and Attenuation 1917
45.5.5 HP ADC FSM 1918
45.5.6 HP ADC Pattern Table 1919
45.5.7 HP ADC Pattern Configuration Example for Multi-channel Sampling 1920
45.5.8 Dual HP ADC Sampling Control 1921
45.5.9 HP ADC Filters 1921
45.5.10 HP ADC Threshold Monitors 1921
45.5.11 HP ADC GDMA Support 1922
45.5.12 LP ADC Wake-Up Modes for Automatic Monitoring 1922
45.6 Event Task Matrix Feature 1922
45.7 Interrupts 1923
45.8 Programming Procedure 1924
45.8.1 HP ADC Multi-Channel Sampling Mode 1924
45.8.2 LP ADC One-shot Sampling Mode 1924
45.8.3 LP ADC Automatic Monitoring 1924
45.9 Register Summary 1926
45.9.1 HP ADC Register Summary 1926
45.9.2 LP ADC Register Summary 1926
45.10 Registers 1928
45.10.1 HP ADC Registers 1928
45.10.2 LP ADC Registers 1938

46 Analog Voltage Comparator 1955
46.1 Introduction 1955
46.2 Feature List 1955
46.3 Architectural Overview 1955
46.4 Functional Description 1956
46.5 Event Task Matrix Feature 1957
46.6 Interrupts 1957
46.7 Programming Procedures 1958

IX Appendix 1959

Related Documentation and Resources 1960

Glossary 1961
Abbreviations for Peripherals 1961
Abbreviations Related to Registers 1961
Access Types for Registers 1962

Programming Reserved Register Field 1964
Introduction 1964
Programming Reserved Register Field 1964

Interrupt Configuration Registers 1965

Revision History 1966

List of Tables

1.3-1 Trace Encoder Parameters 46
1.5-1 Optional sideband signals 48
1.6-1 Header Format 51
1.6-2 Index Format 51
1.6-3 Packet format 3 subformat 0 51
1.6-4 Packet format 3 subformat 1 52
1.6-5 Packet format 3 subformat 3 52
1.6-6 Packet format 2 53
1.6-7 Packet format 1 with address 54
1.6-8 Packet format 1 without address 55
1.7-1 CORE*n*_TRACE's Internal Interrupt Sources 56
1.8-1 Debug module trigger support (the action field of the mcontrol register) 57
1.8-2 Trace Status 59

2.4-1 LP CPU Interrupt Mapping 84
2.4-2 LP CPU Exception Causes 85
2.7-1 Performance Counter 92
2.10-1 Wake Sources 96

3.4-1 GDMA-AHB Selecting Peripherals via Register Configuration 102
3.4-2 GDMA-AXI Selecting Peripherals via Register Configuration 103

4.4-1 Descriptor Field Description 209
4.4-2 Selecting Peripherals via Register Configuration in TX Direction 213
4.4-3 Selecting Peripherals via Register Configuration in RX Direction 213
4.4-4 Recommended Configurations for Macroblock Reordering in TX Direction 214
4.4-5 Recommended Configuration Table for Macroblock Reordering in 2D-DMA Receive Direction (Without Color Format Conversion) 214
4.4-6 Parameter Configuration for Color Space Conversion in TX Direction 216
4.4-7 Parameter Configuration for Color Space Conversion in RX Direction 217
4.4-8 RGBB888 Storage Format 218
4.4-9 YUV444 Storage Format 218
4.4-10 YUV422 Storage Format 218

5.3-1 Memory Address Mapping 281
5.3-2 Module/Peripheral Address Mapping 288

6.3-1 Parameters in eFuse BLOCK0 294
6.3-2 Secure Key Purpose Values 299
6.3-3 Parameters in BLOCK1 to BLOCK10 300
6.3-4 Registers Information 305
6.3-5 Configuration of Default VDDQ Timing Parameters 306

6.4-1 eFuse's Internal Interrupt Sources 307

7.4-1 HP GPIO Wakeup Signal Trigger and Clear Conditions 351
7.4-2 LP GPIO Wakeup Signal Trigger and Clear Conditions 352
7.8-1 Bit Used to Control IO MUX Functions in Light-sleep Mode 359
7.12-1 Peripheral Signals via HP GPIO Matrix 363
7.13-1 Peripheral Signals via LP GPIO Matrix 374
7.14-1 HP IO MUX Pin Functions 375
7.15-1 LP IO MUX Pin Functions 378
7.16-1 GPIO Pin Analog Functions 379
7.18-1 HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources 381
7.18-2 LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources 382

8.1-1 HP CPU Reset Source 446
8.1-2 LP CPU Reset Source 447
8.2-1 ROOT_CLK Clock Source 450
8.2-2 LP_SLOW_CLK Clock Source Selection 451
8.2-3 LP_FAST_CLK Clock Source Selection 452
8.2-4 Derived HP Clock sources 453
8.2-5 HP Peripheral Clocks 453
8.2-6 Derived LP Clock Sources 455
8.2-7 LP Peripheral Clocks 455

9.2-1 Default Configuration of Strapping Pins 582
9.2-2 Boot Mode Control 582
9.2-3 ROM Message Printing Control 584
9.2-4 JTAG Signal Source Control 585

10.4-1 CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources 590

11.3-1 Selectable Events for ETM Channel*n* 628
11.3-2 Mappable Tasks for ETM Channel*n* 634

12.5-1 UNIT*n* Configuration Bits 786
12.5-2 Trigger Point 788
12.5-3 Synchronization Operation for Configuration Registers 788
12.6-1 System Timer's Internal Interrupt Sources 789

13.3-1 Alarm Generation When Up-Down Counter Increments 812
13.3-2 Alarm Generation When Up-Down Counter Decrements 812
13.3-3 TIMG*n*'s Internal Interrupt Sources 814

14.2-1 Timeout Actions 837

15.3-1 Configure RTC Timer to Log the Occurrence Time of Specific Events 850
15.3-2 Target times Configuration 850
15.5-1 RTC Timer's Internal Interrupt Sources 851

16.1-1 Management Areas by PMP and APM 863

16.5-1 APM Internal Interrupt Sources 869

17.5-1 HP CPU0/1 Packet Format 969
17.5-2 HP CPU LOST Packet Format 969
17.5-3 DMA_2 Packet Format 972
17.5-4 DMA_3 Packet Format 972
17.5-5 DMA LOST Packet Format 972

18.3-1 LP Mailbox's Internal Interrupt Sources 1024

20.4-1 AES Accelerator Working Mode 1055
20.4-2 Key Length and Encryption/Decryption 1055
20.5-1 Working Status under Typical AES Working Mode 1056
20.5-2 Text Endianness Type for Typical AES 1056
20.5-2 Text Endianness Type for Typical AES 1057
20.5-3 Key Endianness Type for AES-128 Encryption and Decryption 1058
20.5-4 Key Endianness Type for AES-256 Encryption and Decryption 1058
20.6-1 Block Cipher Mode 1060
20.6-2 Working Status under DMA-AES Working mode 1060
20.6-3 TEXT-PADDING 1061
20.6-4 Text Endianness for DMA-AES 1061
20.8-1 AES's Internal Interrupt Sources 1066

21.3-1 ECC Accelerator Memory Blocks 1076
21.4-1 ECC Accelerator Key Size Selection 1077
21.4-2 Working Modes of ECC Accelerator 1078

22.2-1 HMAC Purposes and Configuration Value 1090

23.3-1 Acceleration Performance 1109
23.4-1 RSA's Internal Interrupt Source 1109
23.5-1 RSA Accelerator Memory Blocks 1110

24.3-1 SHA Accelerator Working Mode 1116
24.3-2 SHA Hash Algorithm Selection 1116
24.4-4 The Storage and Length of Message Digest from Different Algorithms 1123
24.5-1 SHA's Internal Interrupt Sources 1123

26.4-1 ECDSA Working Mode 1145
26.4-2 ECDSA Elliptic Curves Selection 1145
26.4-3 ECDSA SHA Algorithm 1145
26.4-4 ECDSA Working Status 1145
26.6-1 ECDSA Memory Blocks 1153

28.1-2 Marker Code Assignments 1168
28.2-1 The Number of Data Units in the MCU of Different Image Format 1168
28.5-1 Supported Image Formats for Color Space Conversion 1172
28.5-2 Position Number of Coefficients in a Quantization Coefficient Table (Example) 1173
28.5-3 Recommended 8-bit Precision Quantization Coefficient Table for the Luminance Component 1173

28.5-4 Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component 1174
28.5-5 Sampling Factor and Image Format 1175
28.7-1 JPEG Encoder hb and vb Configuration 1181
28.7-2 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* Configuration 1182
28.7-3 JPEG Decoder JPEG_HA and JPEG_VA Configuration 1184
28.7-4 JPEG Decoder hb and vb Configuration 1184
28.7-5 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration 1185
28.7-6 2D DMA Color Space Conversion Configuration 1186

29.5-1 Pixel Storage Order 1212
29.5-2 PPA SRM Alpha Configuration 1213
29.5-3 BLEND CLUT Address 1215
29.5-4 PPA Linked List Configuration 1216
29.5-5 Number of Pixel Block Rearrangements in SRM Output 1219
29.6-1 SRM Parameter Error Type 1220

30.3-1 LCD_CAM Input and Output Signal Descriptions 1251
30.3-2 LCD Data Format Control – Stage One 1256
30.3-3 LCD Output Data Reordering 1257
30.3-4 Camera Data Format Control 1258
30.3-5 Conversion Mode Control in 24-bit Mode 1260
30.3-6 Conversion Mode Control in Other Modes 1260

31.2-1 UART and LP UART Feautre Comparison 1286
31.4-1 UART_CHAR_WAKEUP Mode Configuration 1295

32.3-1 I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1) 1375
32.4-1 I2C Synchronous Registers 1378

34.4-1 I2S*n* Signal Description 1466
34.9-1 Bit Width of Channel Valid Data 1473
34.9-2 Endian of Channel Valid Data 1473
34.9-3 Data-Fetching Control in PDM Mode 1476
34.9-4 I2S*n* Channel Control for Raw PDM Data 1477
34.9-5 I2S*O* PCM-to-PDM Data Output 1477
34.10-1 PDM-to-PCM Data Input 1481
34.10-2 Channel Storage Data Width 1482
34.10-3 Channel Storage Data Endian 1482

35.4-1 Counter Mode. Rising edge of Input Pulse Signal. Control Signal in Low State 1510
35.4-2 Counter Mode. Rising edge of Input Pulse Signal. Control Signal in High State 1510
35.4-3 Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in Low State 1510
35.4-4 Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in High State 1511
35.5-1 PCNT's Internal Interrupt Sources 1512

36.3-1 Standard CDC-ACM Control Requests 1526
36.3-2 CDC-ACM Settings with RTS and DTR 1527
36.3-3 Commands of a Nibble 1529

36.3-4 USB-to-JTAG Control Requests 1530
36.3-5 JTAG Capability Descriptors 1531
36.4-1 Reset SoC into Download Mode 1532
36.4-2 Reset SoC into Booting from flash 1532

37.4-1 Transmit Descriptor 0 (TDES0) 1566
37.4-2 Transmit Descriptor 1 (TDES1) 1570
37.4-3 Transmit Descriptor 2 (TDES2) 1570
37.4-4 Transmit Descriptor 3 (TDES3) 1570
37.4-5 Transmit Descriptor 6 (TDES6) 1571
37.4-6 Transmit Descriptor 7 (TDES7) 1571
37.4-7 Receive Descriptor 0 (RDES0) 1571
37.4-8 Receive Descriptor 1 (RDES1) 1574
37.4-9 Receive Descriptor 2 (RDES2) 1575
37.4-10 Receive Descriptor 3 (RDES3) 1575
37.4-11 Receive Descriptor 4 (RDES4) 1575
37.4-12 Receive Descriptor 6 (RDES6) 1577
37.4-13 Receive Descriptor 7 (RDES7) 1577
37.4-14 Destination Address Filtering 1583
37.4-15 Source Address Filtering 1584

38.2-1 Data Frames and Remote Frames in SFF and EFF 1644
38.2-2 Error Frame 1646
38.2-3 Overload Frame 1646
38.2-4 Interframe Space 1647
38.2-5 Segments of a Nominal Bit Time 1650
38.4-1 Bit Information of TWAI_BUS_TIMING_0_REG (0x18) 1654
38.4-2 Bit Information of TWAI_BUS_TIMING_1_REG (0x1c) 1655
38.4-3 Buffer Layout for Standard Frame Format and Extended Frame Format 1655
38.4-4 TX/RX Frame Information (SFF/EFF); TWAI Address 0x40 1656
38.4-5 TX/RX Identifier 1 (SFF); TWAI Address 0x44 1656
38.4-6 TX/RX Identifier 2 (SFF); TWAI Address 0x48 1656
38.4-7 TX/RX Identifier 1 (EFF); TWAI Address 0x44 1657
38.4-8 TX/RX Identifier 2 (EFF); TWAI Address 0x48 1657
38.4-9 TX/RX Identifier 3 (EFF); TWAI Address 0x4c 1657
38.4-10 TX/RX Identifier 4 (EFF); TWAI Address 0x50 1657
38.4-11 Bit Information of TWAI_ERR_CODE_CAP_REG (0x30) 1663
38.4-12 Bit Information of Bits SEG.4 - SEG.0 1663
38.4-13 Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c) 1664
38.5-1 TWAI's Internal Interrupt Sources 1666

39.3-1 SD/MMC Signal Description 1690
39.8-1 DES0 Descriptor Field 1696
39.8-2 DES1 Descriptor Field 1697
39.8-3 DES2 Descriptor Field 1698
39.8-4 DES3 Descriptor Field 1698

40.4-1 Commonly-used Frequencies and Resolutions 1737
40.7-1 LEDC Memory Blocks 1744

41.3-1 Functions of Each Submodule in the Operator Module 1763
41.3-2 Timing Events Used in PWM Generator 1773
41.3-3 Timing Events Priority When PWM Timer Increments 1773
41.3-4 Timing Events Priority when PWM Timer Decrements 1774
41.3-5 Dead Time Generator Switches Control Fields 1785
41.3-6 Typical Dead Time Generator Operating Modes 1785
41.3-7 MCPWM-Related ETM Events 1793
41.3-8 ETM Related Tasks 1794

42.3-1 Configuration Update 1854

43.5-1 Operations to Reset AXI Clock Domain with Clock Restrictions 1878
43.5-2 Requirements for TX Unit Operating as Slave with Clock Restrictions 1879
43.5-3 Requirements for RX Unit Operating as Slave with Clock Restrictions 1880
43.6-1 PARLIO's Internal Interrupt Sources 1884

44.4-1 Temperature Measurement Range and Offset 1904

45.5-1 SAR ADC Channels and Chip Pins 1916

46.4-1 Mapping Between PAD and GPIO 1956
46.6-1 Analog Voltage Comparator's Internal Interrupt Sources 1958

46.7-4 Configuration of ENA/RAW/ST Registers 1965

List of Figures

1.0-1 Trace Encoder Overview 44
1.2-1 Trace Flow Overview 45
1.6-1 Trace packet Format 50

2.1-1 LP CPU Overview 74
2.10-1 Wake-Up and Sleep Flow of LP CPU 94

3.1-1 Modules that Share GDMA-AHB Channels 98
3.1-2 Modules that Share GDMA-AXI Channels 99
3.3-1 GDMA controller Architecture 100
3.4-1 Structure of a Linked List 101
3.4-2 Relationship among Linked Lists 104
3.4-3 CRC Calculation Matrix 108

4.3-1 2D-DMA Architecture 207
4.4-1 Structure of a Linked List 208
4.4-2 Padding Illustration (Overall) 212
4.4-3 Padding Illustration (Detail) 212
4.4-4 Macroblock Layout of JPEG Decoder Output Data 215
4.4-5 Color Space Conversion 216

5.3-1 System Structure and Address Mapping 281
5.3-2 Cache Structure 286

6.3-1 Data Flow in eFuse 293
6.3-2 Shift Register Circuit (first 32 output) 302
6.3-3 Shift Register Circuit (last 12 output) 302

7.3-1 Architecture of HP GPIO Matrix, HP IO MUX, LP GPIO Matrix, and LP IO MUX 347
7.3-2 Internal Structure of a Pad 348
7.4-1 GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock 350
7.4-2 GPIO Filter Timing of GPIO Input Signals 350
7.4-3 Glitch Filter Timing Example 353
7.10-1 Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled 360
7.10-2 Example of Level Flip on the Chip Pad — Hysteresis Function Enabled 361

8.1-1 Reset Types 445
8.2-1 HP and LP System Clock 449
8.3-1 Clock Configuration Example 458

9.2-1 Chip Boot Flow 583

10.2-1 Interrupt Flow in ESP32-P4 587
10.3-1 Interrupt Matrix Structure 588

11.3-1 ETM Channel*n* Architecture 628
11.3-2 Event Task Matrix Clock Architecture 639

12.3-1 System Timer Structure 785
12.5-1 System Timer Alarm Generation 786

13.1-1 Timer Group Overview 809
13.3-1 Timer Group Architecture 810

14.1-1 Watchdog Timers Overview 835
14.2-1 Digital Watchdog Timers in ESP32-P4 836
14.3-1 Super Watchdog Controller Structure 839

16.1-1 PMP-APM Management Relation 864
16.3-1 APM Architecture 866

18.3-1 LP Mailbox Block Diagram 1023

19.3-1 Brown-out Detector Overview 1040
19.4-1 Structure of Brown-out Detector 1042

20.7-1 GCM Encryption Process 1064

22.3-1 HMAC SHA-256 Padding Diagram 1093
22.3-2 HMAC Structure Schematic Diagram 1094

25.3-1 Software Preparations and Hardware Working Process 1133

26.5-1 ECDSA Process 1148

27.3-1 Noise Source 1162

28.1-1 JPEG Bitstream Format 1168
28.2-1 JPEG codec system connection block diagram 1169
28.4-1 JPEG encoder architecture 1170
28.4-2 JPEG decoder architecture 1171

29.1-1 PPA 2D-DMA Connection 1210
29.4-1 PPA Architecture 1211
29.5-1 PPA SRM Input Pixel Processing 1213
29.5-2 PPA SRM Alpha Configuration Logic 1213
29.5-3 2D-DMA Linked List 1216
29.5-4 SRM Procedure 1218

30.3-1 LCD_CAM Block Diagram 1250
30.3-2 LCD Clock 1252
30.3-3 Camera Clock 1253
30.3-4 LCD Frame Structure 1261
30.3-5 LCD Timing (RGB Format) 1262
30.3-6 LCD Timing (I8080/MOTO6800 Format) 1262

31.3-1 UART Structure 1287
31.4-1 UART Controllers Division 1290
31.4-2 The Timing Diagram of Weak UART Signals Along Negative Edges 1290
31.4-3 Structure of UART Data Frame 1291
31.4-4 AT_CMD Character Structure 1292
31.4-5 Driver Control Diagram in RS485 Mode 1293
31.4-6 The Timing Diagram of Encoding and Decoding in SIR mode 1294
31.4-7 IrDA Encoding and Decoding Diagram 1294
31.4-8 Hardware Flow Control Diagram 1296
31.4-9 Connection between Hardware Flow Control Signals 1297
31.4-10 Data Transfer in GDMA Mode 1298
31.5-1 UART Programming Procedures 1301

32.3-1 I2C Master Architecture 1373
32.3-2 I2C Slave Architecture 1373
32.3-3 I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1) 1374
32.4-1 I2C Timing Diagram 1379
32.4-2 Structure of I2C Command Registers 1381
32.6-1 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address 1385
32.6-2 $I2C_{master}$ Writing to a Slave with a 10-bit Address 1387
32.6-3 $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses 1389
32.6-4 $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences 1391
32.6-5 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address 1393
32.6-6 $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address 1396
32.6-7 $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with a 7-bit Address 1398
32.6-8 $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments 1401

33.3-1 Analog I2C Controller Architecture 1455
33.4-1 Signal Phase 1456

34.4-1 ESP32-P4 I2S System Diagram 1465
34.5-1 TDM Philips Standard Timing Diagram 1467
34.5-2 TDM MSB Alignment Standard Timing Diagram 1468
34.5-3 TDM PCM Standard Timing Diagram 1468
34.5-4 PDM Standard Timing Diagram 1469
34.6-1 I2S*n* Clock Generator 1469
34.9-1 TX Data Format Control 1475
34.9-2 TDM Channel Control 1476
34.9-3 PDM Channel Control Example 1479

35.1-1 PCNT Overview 1508
35.3-1 PCNT Architectural Overview 1509
35.6-1 Channel 0 Up Counting Diagram 1512
35.6-2 Channel 0 Down Counting Diagram 1513
35.6-3 Two Channels Up Counting Diagram 1513

36.2-1 USB Serial/JTAG High Level Diagram 1525
36.2-2 USB Serial/JTAG Block Diagram 1526

37.1-1 Ethernet MAC Functionality Overview 1560
37.3-1 Ethernet MAC Block Diagram 1562
37.4-1 Transmit Descriptor Linked List 1566
37.4-2 Receive Descriptor Linked List 1571
37.4-3 MII Interface 1578
37.4-4 MII Clock 1579
37.4-5 RMII Interface 1580
37.4-6 RMII Clock 1581

38.2-1 Bit Fields in Data Frames and Remote Frames 1644
38.2-2 Fields of an Error Frame 1646
38.2-3 Fields of an Overload Frame 1646
38.2-4 The Fields within an Interframe Space (*Error Passive) 1647
38.2-5 Layout of a Bit 1650
38.3-1 TWAI Overview Diagram 1651
38.4-1 Acceptance Filter 1659
38.4-2 Single-Filter Mode 1660
38.4-3 Dual-Filter Mode 1661
38.4-4 Error State Transition 1662
38.4-5 Positions of Arbitration Lost Bits 1664

39.2-1 SD/MMC Controller Topology 1689
39.3-1 SD/MMC Controller External Interface Signals 1690
39.4-1 SDIO Host Block Diagram 1691
39.4-2 Command Path State Machine 1692
39.4-3 Data Transmit State Machine 1693
39.4-4 Data Receive State Machine 1693
39.7-1 Linked List Structure 1696
39.8-1 Descriptor Format 1696

40.3-1 LED PWM Architecture 1734
40.3-2 Timer and PWM Generator Block Diagram 1734
40.4-1 Frequency Division When LEDC_CLK_DIV is a Non-Integer Value 1736
40.4-2 Relationship Between Counter And Resolution 1736
40.4-3 LED PWM Output Signal Diagram 1738
40.4-4 Output Signal of Linear Duty Cycle Fading 1739
40.4-5 Output Signal of Gamma Curve Fading 1740

41.2-1 MCPWM Module Overview 1760
41.3-1 Timer Module 1762
41.3-2 Operator Module 1762
41.3-3 Capture Module 1763
41.3-4 ETM Module 1763
41.3-5 Count-Up Mode Waveform 1765
41.3-6 Count-Down Mode Waveforms 1765
41.3-7 Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event 1765
41.3-8 Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event 1766

41.3-9 UTEP and UTEZ Generation in Count-Up Mode 1767
41.3-10 DTEP and DTEZ Generation in Count-Down Mode 1768
41.3-11 DTEP and UTEZ Generation in Count-Up-Down Mode 1768
41.3-12 Block Diagram of A PWM Operator 1771
41.3-13 Symmetrical Waveform in Count-Up-Down Mode 1775
41.3-14 Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 1776
41.3-15 Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A 1777
41.3-16 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 1778
41.3-17 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary 1779
41.3-18 Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B 1780
41.3-19 Example of an NCI Software-Force Event on PWM*x*A 1782
41.3-20 Example of a CNTU Software-Force Event on PWM*x*B 1783
41.3-21 Options for Setting up the Dead Time Generator Module 1785
41.3-22 Active High Complementary (AHC) Dead Time Waveforms 1786
41.3-23 Active Low Complementary (ALC) Dead Time Waveforms 1787
41.3-24 Active High (AH) Dead Time Waveforms 1787
41.3-25 Active Low (AL) Dead Time Waveforms 1788
41.3-26 Example of Waveforms Showing PWM Carrier Action 1789
41.3-27 Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule 1790
41.3-28 Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule 1791

42.3-1 RMT Architecture 1847
42.3-2 Format of Pulse Code in RAM 1848

43.4-1 PARLIO Architecture 1876
43.5-1 PARLIO Clock Generation 1877
43.5-2 Positive Waveform 1879
43.5-3 Negative Waveform 1879
43.5-4 Level Enable Mode for RX Unit 1880
43.5-5 Sub-Modes of Pulse Enable Mode for RX Unit 1881
43.5-6 Software Enable Mode for RX Unit 1881

44.3-1 Temperature Sensor Architecture 1903

45.1-1 SAR ADC Simplified Architecture 1913
45.4-1 SAR ADC Architecture 1915
45.5-1 HP ADC FSM Block Diagram 1918
45.5-2 ADC_SAR*x*_PATT_TAB1_REG Contains Patterns 0 - 3 1919
45.5-3 ADC_SAR*x*_PATT_TAB2_REG Contains Patterns 4 - 7 1919
45.5-4 ADC_SAR*x*_PATT_TAB3_REG Contains Patterns 8 - 11 1919
45.5-5 ADC_SAR*x*_PATT_TAB4_REG Contains Patterns 12 - 15 1919
45.5-6 Pattern Structure 1920
45.5-7 cmd0 configuration 1920

45.5-8 cmd1 Configuration 1920
45.5-9 GDMA Data Format 1922

46.3-1 Analog Voltage Comparator Architecture 1956

Part I

Microprocessor and Master

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

Chapter 1

RISC-V Trace Encoder (TRACE)

The high-performance CPU (HP CPU) of ESP32-P4 supports the instruction trace interface through the trace encoder. The trace encoder connects to HP CPU's instruction trace interface, compresses the information into smaller packets, and then stores the packets into internal SRAM (see Chapter 5 *System and Memory*).

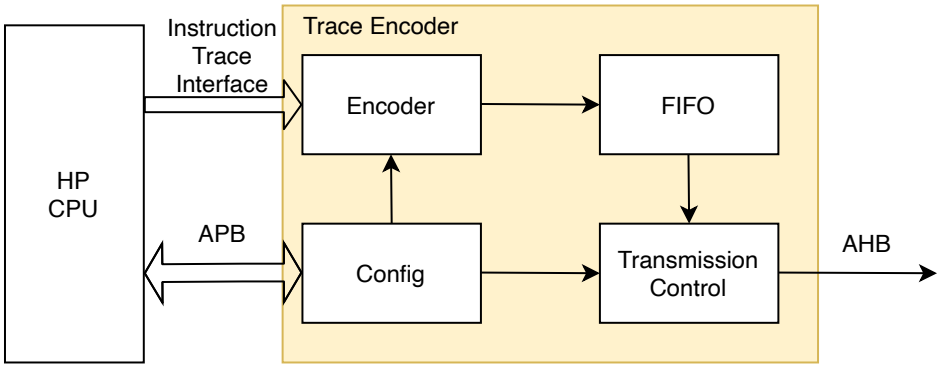


Figure 1.0-1. Trace Encoder Overview

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.

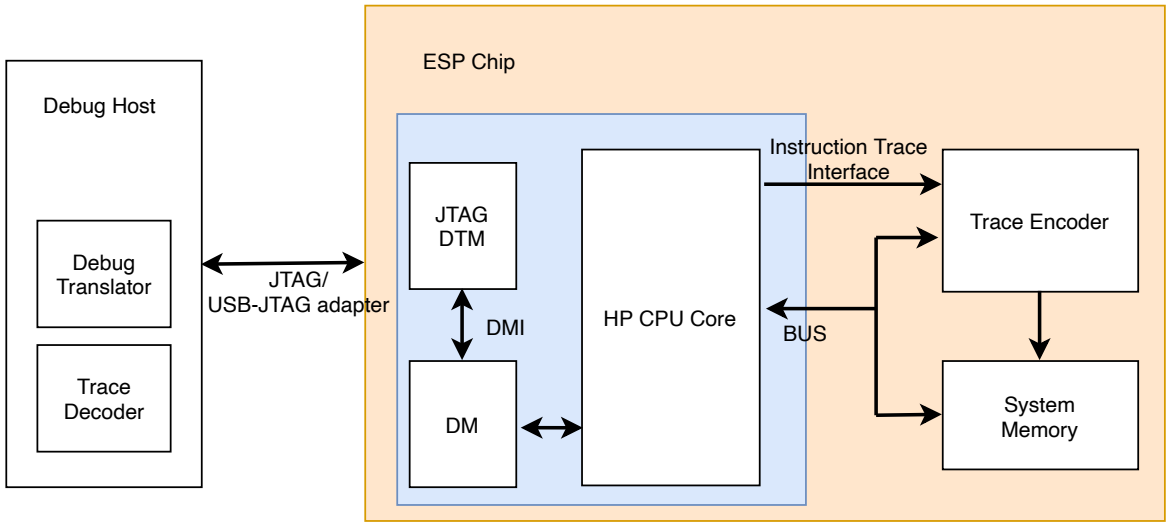


Figure 1.2-1. Trace Flow Overview

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 the instruction information executed by the HP CPU. Such information includes instruction address, instruction type, etc. For more details about ESP32-P4 HP CPU's instruction trace interface, please refer to Chapter 1 *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
- Arbitrary address range of the trace memory size
- Configurable synchronization modes:
  - Synchronization counter counts by packet
  - Synchronization counter counts by cycle
  - Synchronization counter can be disabled
- Trace lost status to indicate packet loss
- Automatic restart after packet loss
- Memory writing in the loop or non-loop mode
- Support two interrupts:
  - Triggered when the packet size exceeds the configured memory space
  - Triggered when a packet is lost
- FIFO (128 × 8 bits) to buffer packets
- AHB burst transmission with configurable burst length

Table 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	1	Only machine and user mode are supported
retires_p	1	Maximum number of instructions that can be retired per block
return_stack_size_p	0	Implicit return mode is not supported
sijump_p	0	Sequentially inferable jump mode is not supported
taken_branches_p	1	Only one instruction retired per cycle
impdef_width_p	0	Not implemented

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

1.4 Architectural Overview

This chapter mainly introduces the implementation details of ESP32-P4'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-P4 supports two address modes: delta address mode and full address mode.

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

1.5.3 Optional Sideband Signals

The Efficient Trace for RISC-V Version 2.0 > Chapter Optional Sideband Signals has provided some optional sideband signals for encoder control. ESP32-P4 has implemented the following functions:

Table 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 signal unit if required, and offers:

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

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

- 1 run-time selectable comparator units 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-P4 encoder inserts zero bytes between data packets:

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

1.5.6 Memory Writing Mode

When writing the trace memory, the size of the trace packets might exceed the capacity of the memory. In this case, you can choose whether to wrap around the trace memory or not by configuring the memory writing mode:

- Loop mode: When the size of the trace packets exceeds the capacity of the trace memory (namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG), the trace memory is wrapped around, so that the encoder loops back to the memory's starting address TRACE_MEM_START_ADDR_REG, and old data in the memory will be overwritten by new data.
- Non-loop mode: When the size of the trace packets exceeds the capacity of the trace memory, the trace memory is not wrapped around. The encoder stops at TRACE_MEM_END_ADDR_REG, and old data will be retained.

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-P4 trace encoder output packet format.

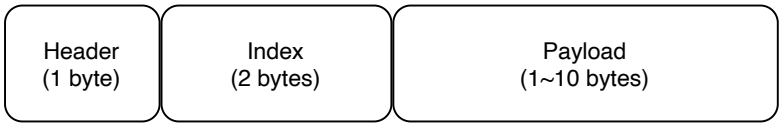


Figure 1.6-1. Trace packet Format

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

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	0~65536

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-P4 only supports 3 of them.

Format 3 Subformat 0 - Synchronization

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

Table 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	1	The privilege level of the reported instruction
address	31	Full instruction address. The address must be left shifted 1 bit in order to recreate the original byte address.
sign_extend	3	Reserved

Format 3 Subformat 1 - Exception

This packet also contains all the information the decoder needs to fully identify an instruction. It is sent following an exception or interrupt, and includes the cause, the 'trap value' (for exceptions), and the address

of the trap handler or of the exception itself. The length is variable: if the interrupt field is 1, the tvalepc field is omitted and the length is 6 bytes, otherwise the length is 10 bytes.

Table 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
branch	1	Set to 0 if the address points to a branch instruction, and the branch was taken. Set to 1 if the instruction is not a branch or if the branch is not taken.
privilege	1	The privilege level of the reported instruction
ecause	6	Exception cause
interrupt	1	Interrupt
theaddr	1	When set to 1, the address field points to the trap handler address. When set to 0, the address field points to the EPC (exception program counter) for an exception at the target for an updiscon, and is undefined for other exceptions and interrupts.
address	31	Full instruction address. The value of this field must be left shifted 1 bit in order to recreate the original byte address.
tvalepc	32	Value from appropriate utval/stval/mtval CSR (control/status register). Omitted if the interrupt is 1
sign_extend	3	Reserved

Format 3 Subformat 3 - Support

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

Table 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.

Field name	Bits	Description
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 when either the branch information must be reported (for example because the branch map is full), or when the address of instruction must be reported, and there has been at least one branch since the previous packet.

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

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

Format 1 - address, branch_map

The length is variable.

Table 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.

Field name	Bits	Description
branch_map	Variable	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken The field Bits is variable and determined by the branches field.
notify	1	If the value of this bit is different from the MSB of address, it indicates that this packet is reporting an instruction that is not the target of an uninferable discontinuity because a notification was requested via trigger[2] or a filter matched.
updiscon	1	If the value of this bit is different from notify, it indicates that this packet is reporting the instruction following an uninferable discontinuity and is also the instruction before an exception, privilege change or resync.
zero_ext	Variable	The length of this field is determined by branches as follows: • 1: 6 bits • 2 ~ 3: 4 bits • 4 ~ 31: 0 bits
address	Variable	Delta instruction address. These bits can be 8/16/24/32

Format 1 - no address, branch_map

The length is 5 bytes.

Table 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. The length of branch_map is 31 bits. Only 0 valid.
branch_map	31	An array of bits indicating whether branches are taken or not. Bit 0 represents the oldest branch instruction executed. For each bit: • 0: branch taken • 1: branch not taken
sign_extend	2	Reserved

1.7 Interrupt

ESP32-P4's CORE*n*_TRACE (i.e., the trace encoder of core 0 or core 1) can generate the CORE*n*_TRACE_INTR interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from CORE*n*_TRACE that can generate the above interrupt signal. The interrupt sources from CORE*n*_TRACE are listed with their trigger conditions and the resulted interrupt signals in Table 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 trace memory, namely when TRACE_MEM_CURRENT_ADDR_REG reaches the value of TRACE_MEM_END_ADDR_REG. If necessary, this interrupt can be enabled to notify the HP CPU for processing, such as applying for a new memory space again	CORE*n*_TRACE_INTR
TRACE_FIFO_OVERFLOW_INTR	Triggered when the internal FIFO overflows and one or more packets have been lost.	CORE*n*_TRACE_INTR

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 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 will automatically stop if TRACE_RESTART_ENA is set. When the FIFO is empty, the encoder will restart to report packets.
- Stall CPU while FIFO is almost full
  - If TRACE_STALL_ENA is set, when the FIFO is almost full, a stall request will be aserted to the CPU, and the CPU will be stalled until the request is deasserted.
- Stop tracing when the CPU is halted

  If TRACE_HALT_ENA is set, when the CPU is halted, the encoder will output a packet to report the address of the last instruction before halting, followed by a support packet to indicate that tracing has stopped. Upon the deassertion of the halted signal, the encoder will start tracing again, commencing with a synchronization packet.
- Stop tracing when the CPU is in reset

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

- Using trigger outputs from the Debug Module

  The debug module of the RISC-V hart has a trigger unit. This defines a match control register (mcontrol) containing a 4-bit action field, and reserves codes 2 ~ 4 of this field for trace use. ESP32-P4 has implemented this feature. The values of the action field are listed in Table 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 1 *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: neither the primary comparator nor secondary comparator match $!(P\&\&S)$
    * 3: start filtering when primary matches up until secondary matches

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

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

2. Set TRACE_FILTER_EN to enable filter unit.

1.8.3 Enable Encoder

- Configure the address space for the trace memory via TRACE_MEM_START_ADDR_REG and TRACE_MEM_END_ADDR_REG. The encoder can access L2 MEM and PSRAM:
  - L2MEM: 0x4FF00000 ~ 0x4FFBFFFF
  - PSRAM: 0x48000000 ~ 0x4BFFFFFF
- Update the value of TRACE_MEM_CURRENT_ADDR_REG to the value of TRACE_MEM_START_ADDR_REG by setting TRACE_MEM_CURRENT_ADDR_UPDATE
- (Optional) Configure the memory writing mode via the TRACE_MEM_LOOP bit of TRACE_TRIGGER_REG
  - 0: Non-loop mode
  - 1: Loop mode (default)
- Configure the synchronization mode via the TRACE_RESYNC_MODE bit of TRACE_RESYNC_PROLONGED_REG
  - 0: Disable the synchronization counter
  - 1: Invalid. No effect
  - 2: Synchronization counter counts by packet
  - 3: Synchronization counter counts by cycle
- (Optional) Configures the threshold for the synchronization counter (default value is 128) via TRACE_RESYNC_PROLONGED_REG
- (Optional) Enable Interrupt
  - Set the corresponding bit of TRACE_INTR_ENA_REG to enable the corresponding interrupt
  - Set the corresponding bit of TRACE_INTR_CLR_REG to clear the corresponding interrupt
  - Read TRACE_INTR_RAW_REG to know which interrupt occurs

- (Optional) Enable automatic restart by setting the TRACE_RESTART_ENA bit of TRACE_TRIGGER_REG. This function is enabled by default
- There are 2 ways to enable the trace encoder:
  - Setting the TRACE_TRIGGER_ON bit of TRACE_TRIGGER_REG
  - Through the Debug module Trace-on to start the encoder. When the trigger is matched and the action is 2, it will start the encoder. For trigger configurations, please refer to the RISC-V Debug Specification.

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

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	If 1 indicates that the FIFO is empty, all data have been written to memory
TRACE_WORK_STATUS	Encoder's work status: • 0: idle state, the encoder is not started • 1: the encoder is normally working that output packets • 2: the encoder is not outputting packets, the CPU is halted or in reset • 3: the encoder is not outputting packets, it occurs while a packet is lost, and is waiting for FIFO empty to restart.

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 trace memory is full
    * if read 0, read the trace packets from TRACE_MEM_START_ADDR_REG
    * if read 1, and the loop mode is enabled, then the old trace packets are overwritten. In this case, read the TRACE_MEM_CURRENT_ADDR_REG to know the last writing address, and use this address as the first address to decode
- Use the decoder to decode data packets
  - The decoder reads all data packets starting from the first address, and reconstructs the data stream with the binary file
  - As mentioned in 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-P4 encoder supports the AHB bus. There are some optional configurations to control the transmission bandwidth of the encoder.

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

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

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

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

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 5.3-2 in Chapter 5 *System and Memory*.

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

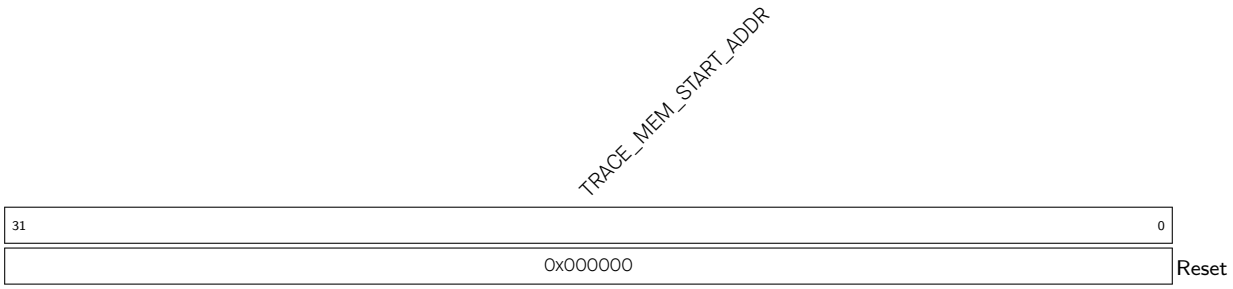
Name	Description	Address	Access
Memory configuration registers			
TRACE_MEM_START_ADDR_REG	Memory start address	0x0000	R/W
TRACE_MEM_END_ADDR_REG	Memory end address	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	Memory current address	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	Memory address update	0x000C	WT
FIFO status register			
TRACE_FIFO_STATUS_REG	FIFO status register	0x0010	RO
Interrupt registers			
TRACE_INTR_ENA_REG	Interrupt enable register	0x0014	R/W
TRACE_INTR_RAW_REG	Interrupt raw status register	0x0018	RO
TRACE_INTR_CLR_REG	Interrupt clear register	0x001C	WT
Trace configuration registers			
TRACE_TRIGGER_REG	Trace trigger register	0x0020	varies
TRACE_CONFIG_REG	Trace configuration register	0x0024	R/W
TRACE_FILTER_CONTROL_REG	Filter control register	0x0028	R/W
TRACE_FILTER_MATCH_CONTROL_REG	Filter match control register	0x002C	R/W
TRACE_FILTER_COMPARATOR_CONTROL_REG	Filter comparator match control register	0x0030	R/W
TRACE_FILTER_P_COMPARATOR_MATCH_REG	Primary comparator match value register	0x0034	R/W
TRACE_FILTER_S_COMPARATOR_MATCH_REG	Secondary comparator match value register	0x0038	R/W
TRACE_RESYNC_PROLONGED_REG	Resynchronization configuration register	0x003C	R/W
TRACE_AHB_CONFIG_REG	AHB configuration register	0x0040	R/W
Clock gating control and configuration register			
TRACE_CLOCK_GATE_REG	Clock gating control register	0x0044	R/W
Version register			
TRACE_DATE_REG	Version control register	0x03FC	R/W

1.10 Registers

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

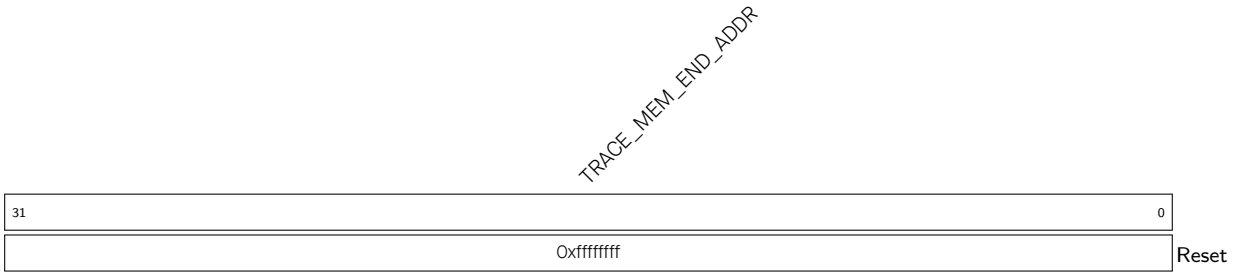
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 1.1. TRACE_MEM_START_ADDR_REG (0x0000)



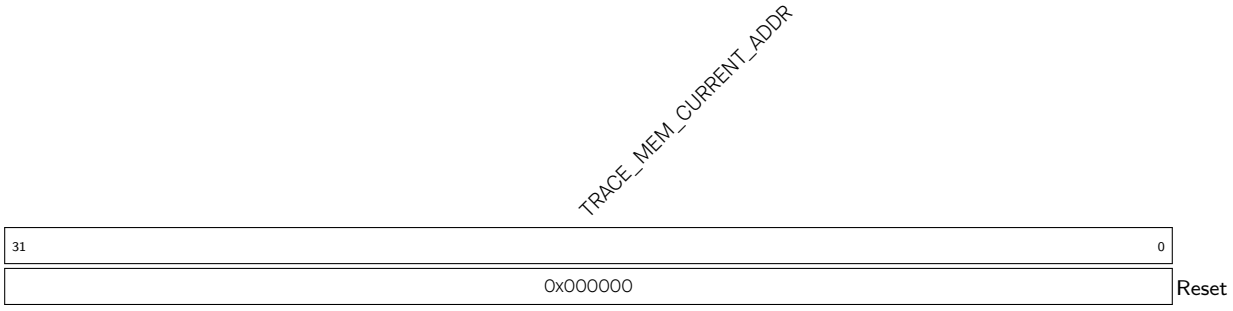
TRACE_MEM_START_ADDR Configures the start address of the trace memory. (R/W)

Register 1.2. TRACE_MEM_END_ADDR_REG (0x0004)



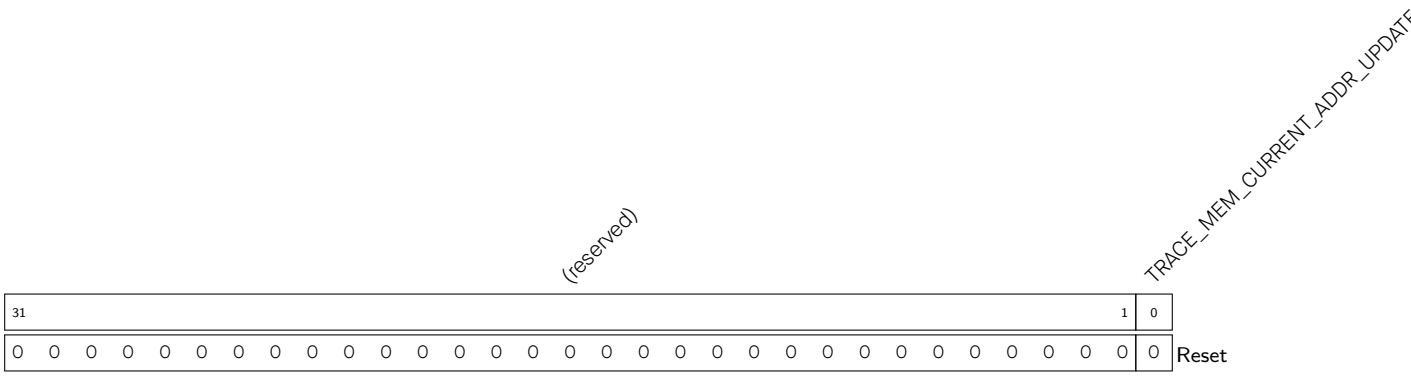
TRACE_MEM_END_ADDR Configures the end address of the trace memory. (R/W)

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)



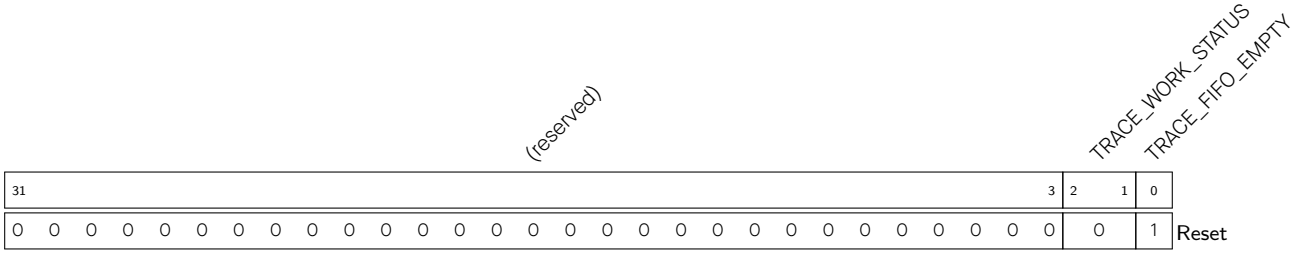
TRACE_MEM_CURRENT_ADDR_UPDATE Configures whether to update the value of TRACE_MEM_CURRENT_ADDR to TRACE_MEM_START_ADDR.

0: Not update

1: Update

(WT)

Register 1.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY Represents whether the FIFO is empty.

1: Empty

0: Not empty (RO)

TRACE_WORK_STATUS Represents the state of the encoder:

0: Idle state

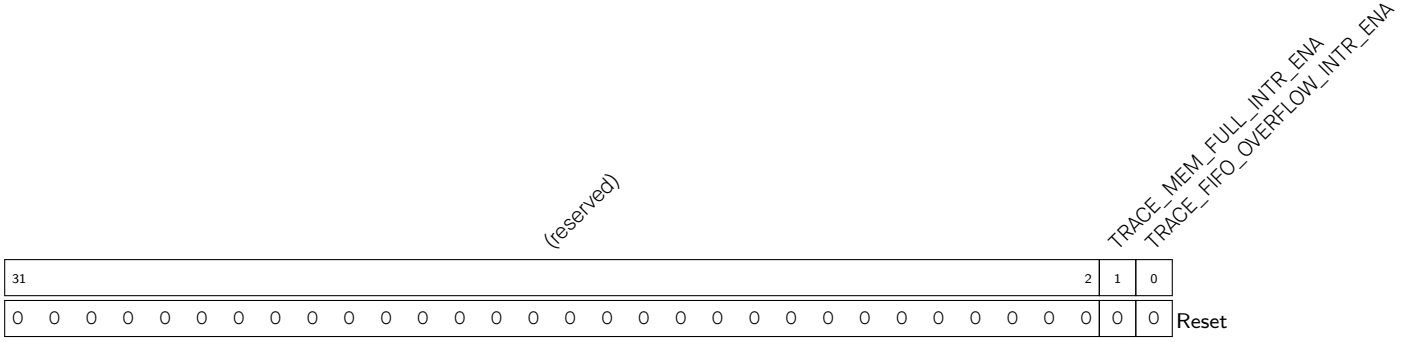
1: Working state

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

3: Lost state

(RO)

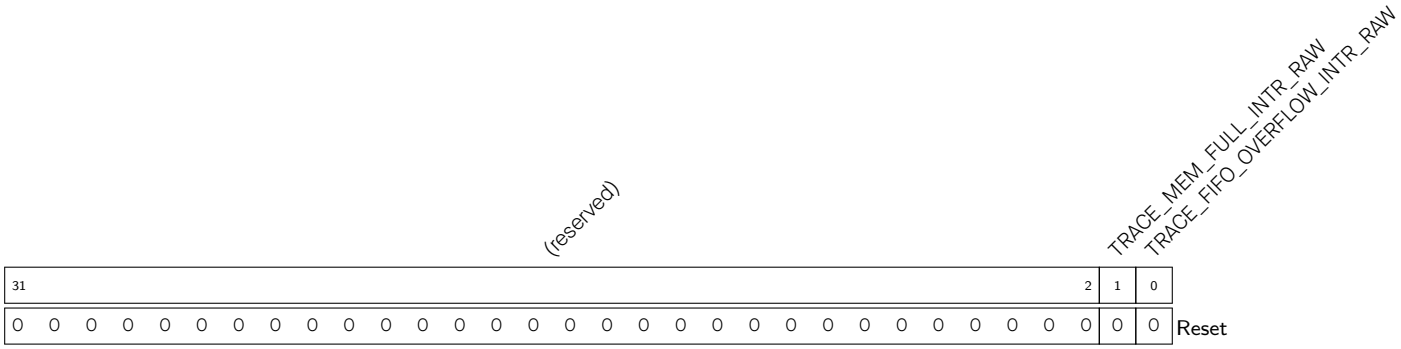
Register 1.6. TRACE_INTR_ENA_REG (0x0014)



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)

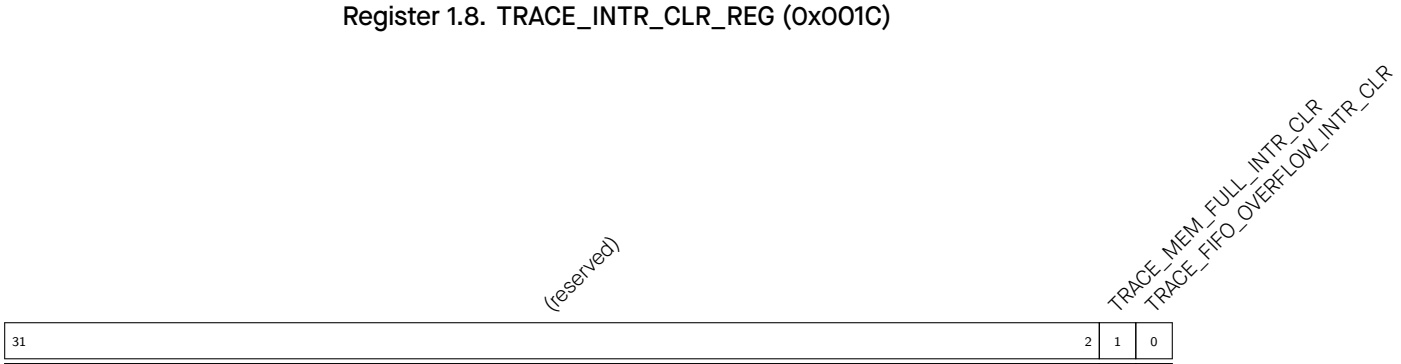
Register 1.7. TRACE_INTR_RAW_REG (0x0018)



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)

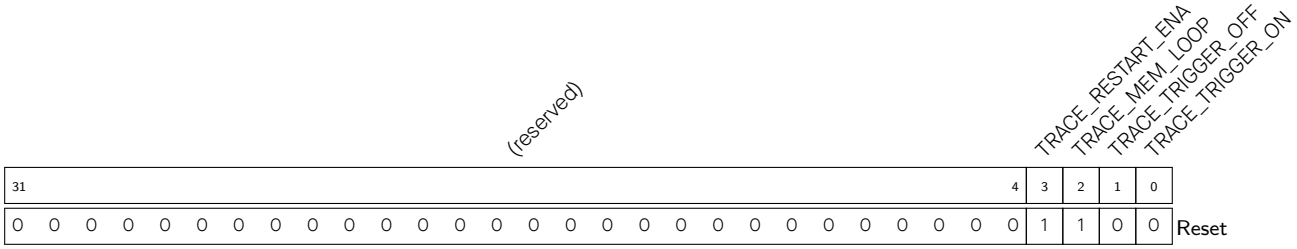
Register 1.8. TRACE_INTR_CLR_REG (0x001C)



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

1: Enable

(WT)

TRACE_TRIGGER_OFF Configures whether to disable the encoder.

0: Invalid

1: Disable

(WT)

TRACE_MEM_LOOP Configures the memory writing mode.

0: Non-loop mode

1: Loop mode

(R/W)

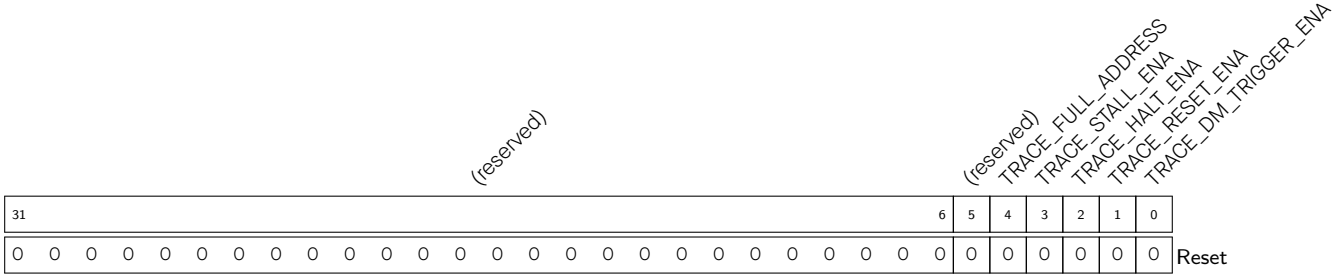
TRACE_RESTART_ENA Configures whether to enable automatic restart.

0: Disable

1: Enable

(R/W)

Register 1.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA Configure whether to enable the trigger signal.

0: Disable

1: Enable

(R/W)

TRACE_RESET_ENA Configures whether to enable the reset signal.

0: Disable

1: Enable

(R/W)

TRACE_HALT_ENA Configures whether to enable the halted signal.

0: Disable

1: Enable

(R/W)

TRACE_STALL_ENA Configures whether to enable the stall signal.

0: Disable

1: Enable

(R/W)

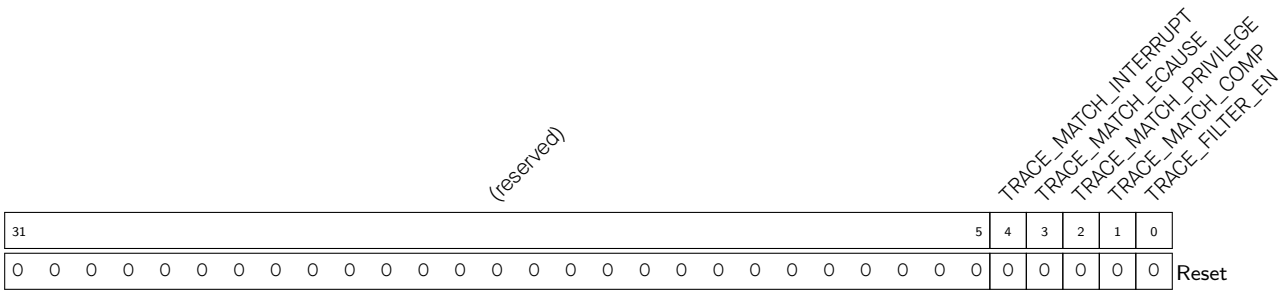
TRACE_FULL_ADDRESS Configure the address mode.

0: Delta address mode

1: Full address mode

(R/W)

Register 1.11. TRACE_FILTER_CONTROL_REG (0x0028)



TRACE_FILTER_EN Configure whether to enable filtering.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_COMP Configures whether to enable the comparator match mode.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_PRIVILEGE Configures whether to enable the privilege match mode.

0: Disable

1: Enable

(R/W)

TRACE_MATCH_ECAUSE Configures whether to enable the ecause match mode.

0: Disable

1: Enable

(R/W)

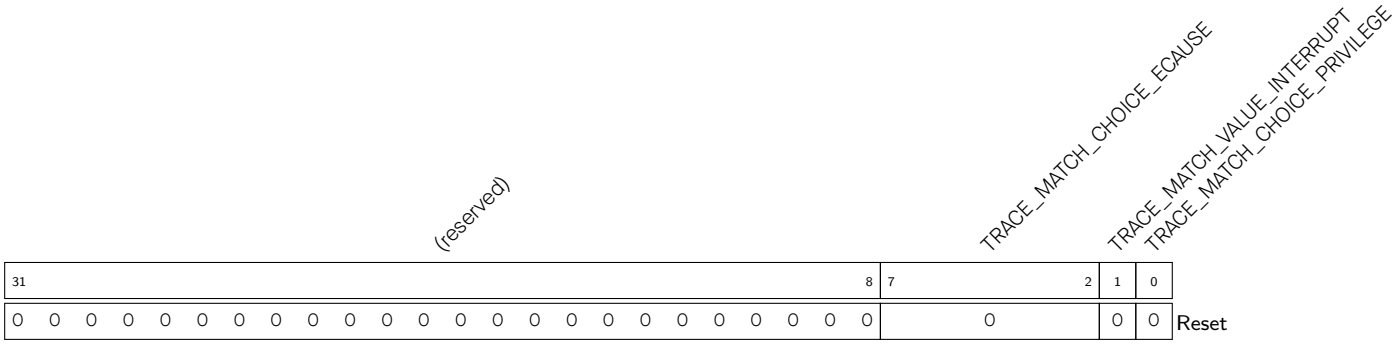
TRACE_MATCH_INTERRUPT Configures whether to enable the interrupt match mode.

0: Disable

1: Enable

(R/W)

Register 1.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)



TRACE_MATCH_CHOICE_PRIVILEGE Configures the privilege level for matching.

0: User mode

1: Machine mode

(R/W)

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

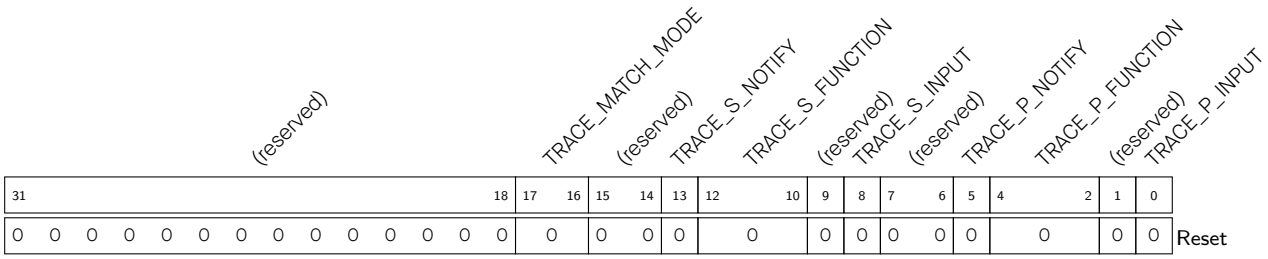
0: itype=1

1: itype=2

(R/W)

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

Register 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 of for secondary comparator.

0: Equal

1: Not equal

2: Less than

3: Less than or equal

4: Greater than

5: Greater than or equal

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

Others: Always match

(R/W)

Continued on the next page...

Register 1.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

Continued from the previous page...

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

0: Not report

1: Report

(R/W)

TRACE_MATCH_MODE Configures the comparator match condition.

0: Only the primary comparator matches

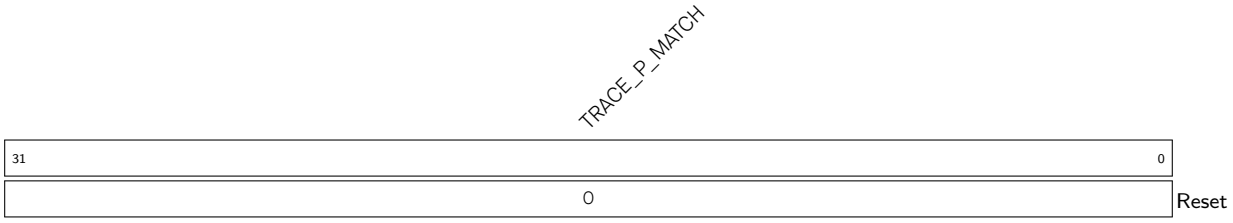
1: Both primary and secondary comparators match (P&&S)

2: Neither primary nor secondary comparator match (!P&&S)

3: Start filtering when the primary comparator matches and stop filtering when the secondary comparator matches

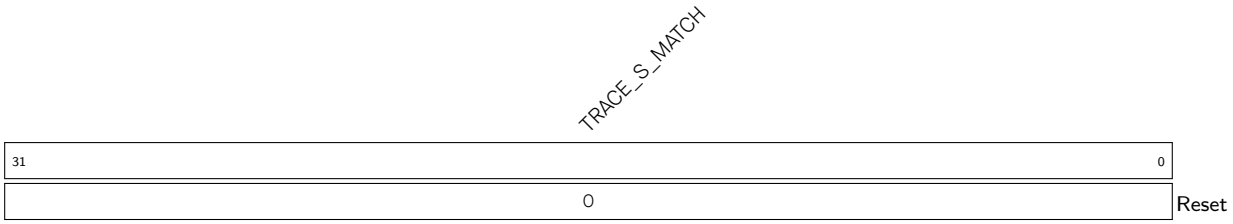
(R/W)

Register 1.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



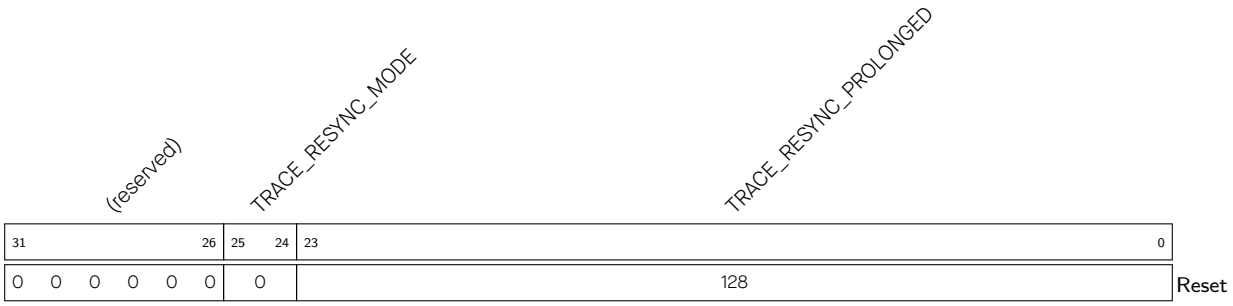
TRACE_P_MATCH Configures the match value for the primary comparator. (R/W)

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

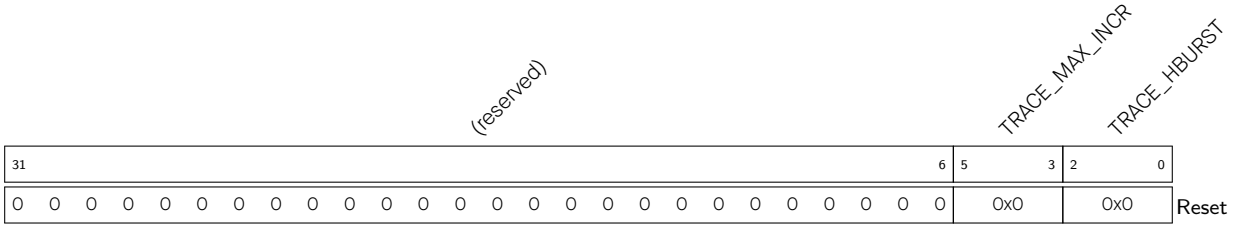
1: Invalid

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

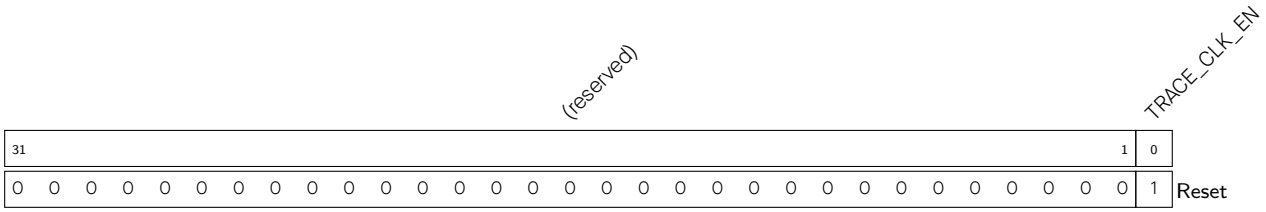
4: INCR8

Others: Invalid

(R/W)

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

Register 1.18. TRACE_CLOCK_GATE_REG (0x0044)



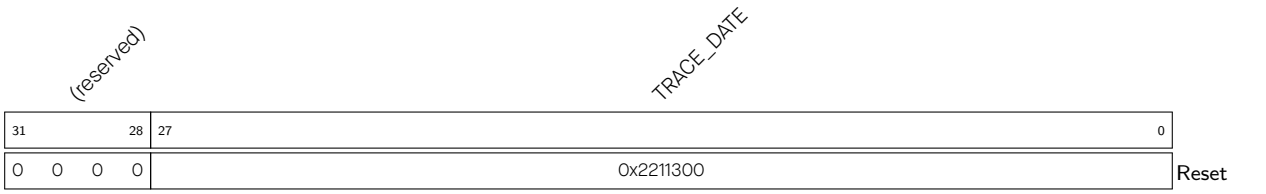
TRACE_CLK_EN Configures register clock gating. 0: Support clock only when the application writes registers to save power

1: Always force the clock on for registers

This bit doesn't 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-P4 Low-Power CPU (LP CPU) is a 32-bit processor based upon RISC-V ISA comprising integer (I), multiplication/division (M), atomic (A), and compressed (C) standard extensions. It features ultra-low power consumption and has a 2-stage, in-order, and scalar pipeline. The LP CPU core complex has an interrupt controller (INTC), a debug module (DM), and system bus (SYS BUS) interfaces for memory and peripheral access.

The LP CPU is in sleep mode by default (see Section 2.10). It can stay powered on when the chip enters Deep-sleep mode (see Chapter 5 *Low-Power Management [to be added later]* for details) and can access most peripherals and memories (see Chapter 5 *System and Memory* for details). It has two application scenarios:

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

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

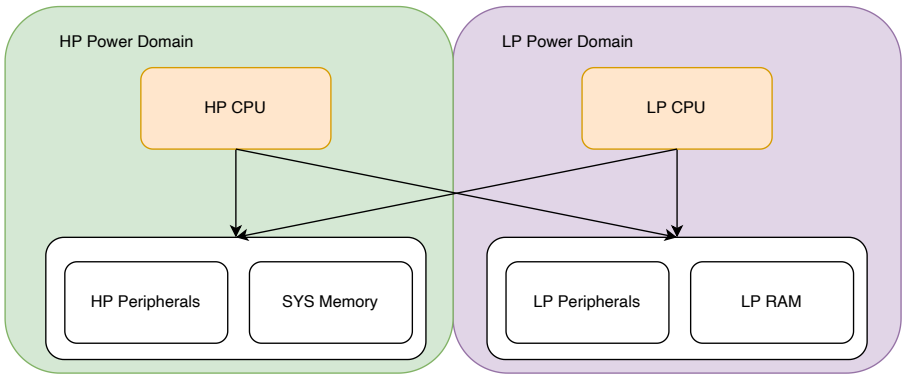


Figure 2.1-1. LP CPU Overview

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 of the other domain cannot access the resources of that domain. In other words, only when the domain is powered on can its resources be accessed. For more information about power, please refer to the chapter 5 *Low-Power Management [to be added later]*.

2.2 Features

The LP CPU has the following features:

- Operating clock frequency of up to 40 MHz
- 18 vector interrupts
- Debug module compliant with RISC-V External Debug Support Version 0.13 with external debugger support over an industry-standard JTAG/USB port
- Hardware trigger compliant with RISC-V External Debug Support Version 0.13 with up to 2 breakpoints/watchpoints
- Core performance metric events
- Wake-up interrupt for HP CPU
- Access to HP memory and LP memory
- Access to the entire peripheral address space

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
mcause [3]	Machine Trap Cause	0x342	R/W

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

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

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

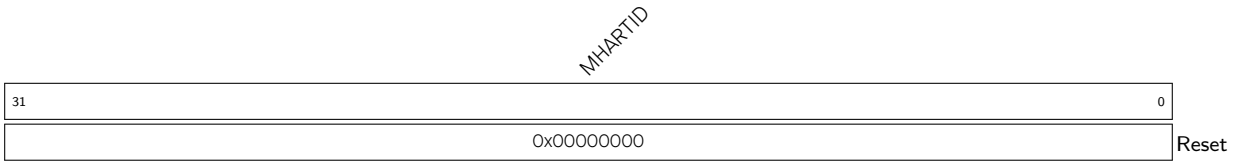
Name	Description	Address	Access
mtval	Machine Trap Value	0x343	R/W
mip	Machine Interrupt Pending	0x344	R/W
Trigger Module CSRs (shared with Debug Mode)			
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
Debug Mode CSRs			
dcsr	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

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

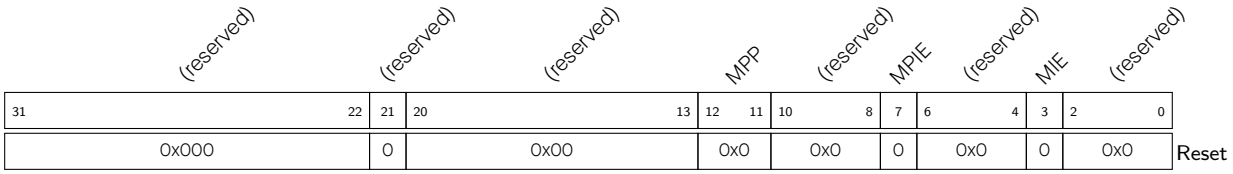
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 2.1. mhartid (0xF14)



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

Register 2.2. mstatus (0x300)



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

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

MPP Configures machine previous privilege mode (before trap).

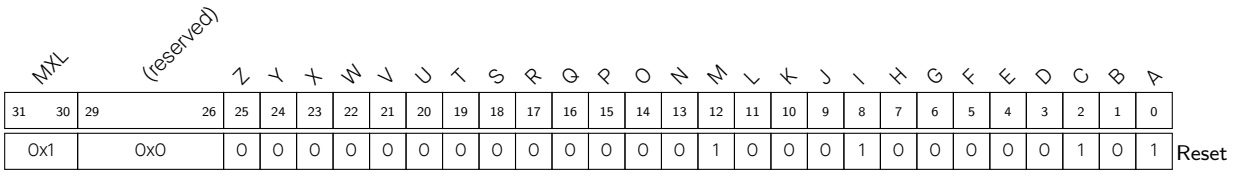
0x3: Machine mode

Other values: Invalid

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

(R/W)

Register 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 = 0. (RO)

T Reserved = 0. (RO)

S Supervisor mode implemented = 0. (RO)

R Reserved = 0. (RO)

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

P Reserved = 0. (RO)

O Reserved = 0. (RO)

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

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

L Reserved = 0. (RO)

K Reserved = 0. (RO)

J Reserved = 0. (RO)

I RV32I base ISA = 1. (RO)

H Hypervisor extension = 0. (RO)

G Additional standard extensions present = 0. (RO)

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

E RV32E base ISA = 0. (RO)

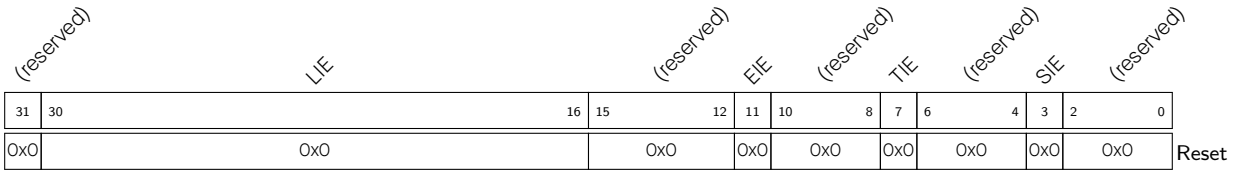
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

A Atomic Standard Extension = 1. (RO)

Register 2.4. mie (0x304)



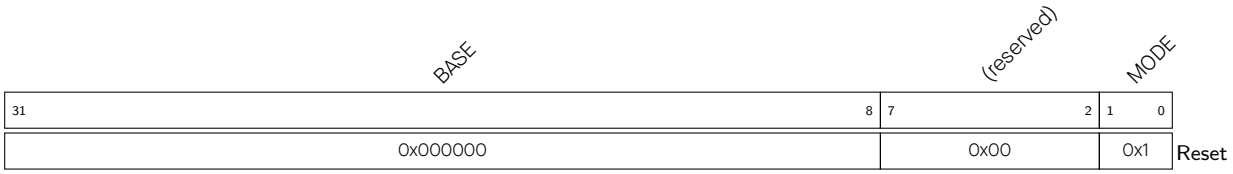
SIE Write 1 to enable software interrupts. (R/W)

TIE Write 1 to enable timer interrupts. (R/W)

EIE Write 1 to enable external interrupts. (R/W)

LIE Write 1 to enable the corresponding local interrupt. (R/W)

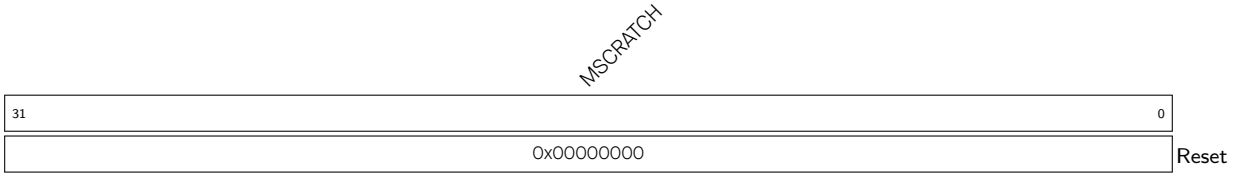
Register 2.5. mtvec (0x305)



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

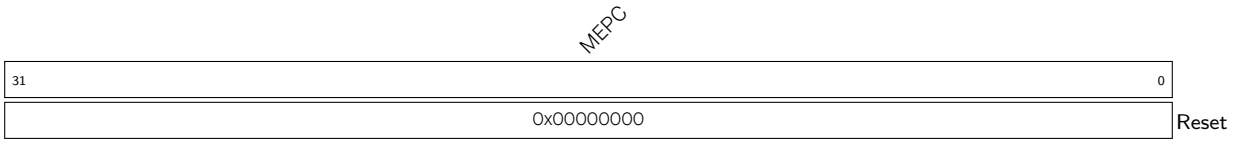
BASE Configures the higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 2.6. mscratch (0x340)



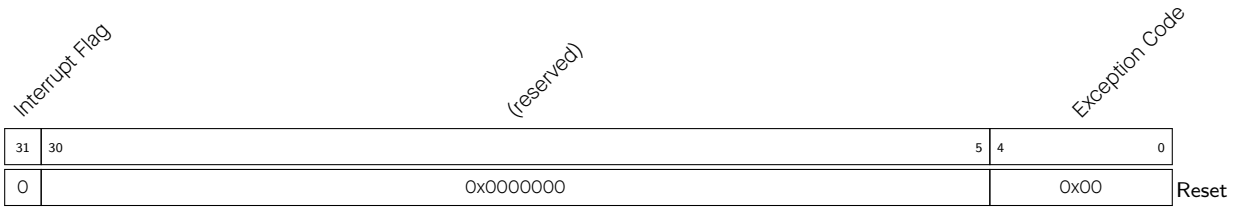
MSCRATCH Contains machine scratch information for custom use. (R/W)

Register 2.7. mepc (0x341)



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

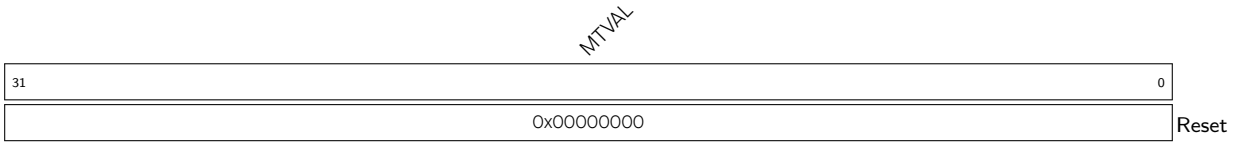
Register 2.8. mcause (0x342)



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

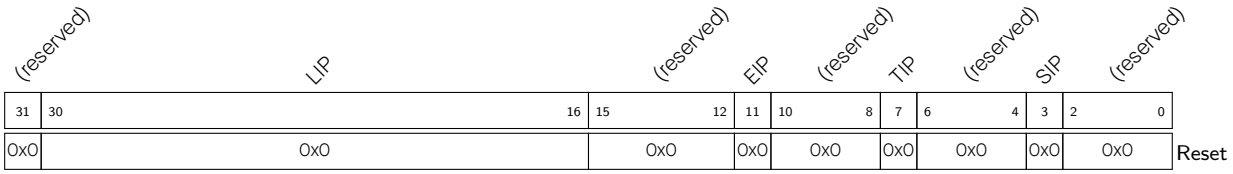
Interrupt Flag This flag is automatically updated when CPU enters trap. If this is found to be set, it indicates that the latest trap occurred due to an interrupt. For exceptions it remains unset. (R/W)

Register 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)



SIP Configures the pending status of the software interrupt.

0: Not pending

1: Pending

(R/W)

TIP Configures the pending status of the timer interrupt.

0: Not pending

1: Pending

(R/W)

EIP Configures the pending status of the external interrupts.

0: Not pending

1: Pending

(R/W)

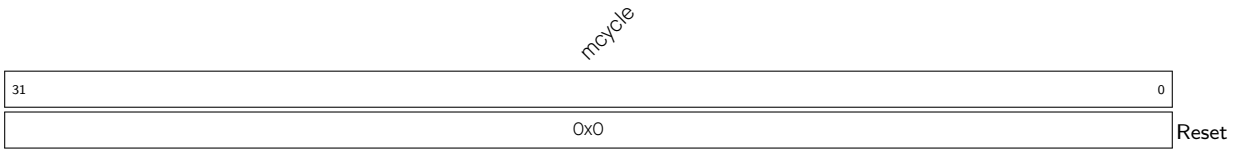
LIP Configures the pending status of the local interrupts.

0: Not pending

1: Pending

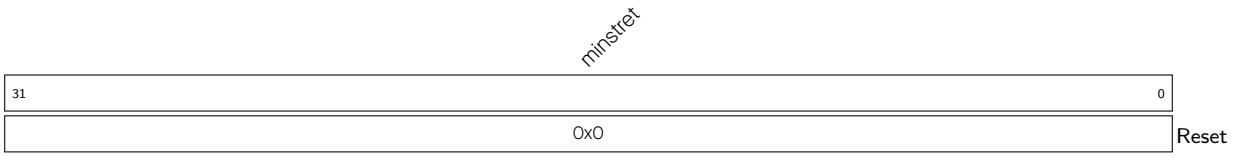
(R/W)

Register 2.11. mcycle (0xB00)



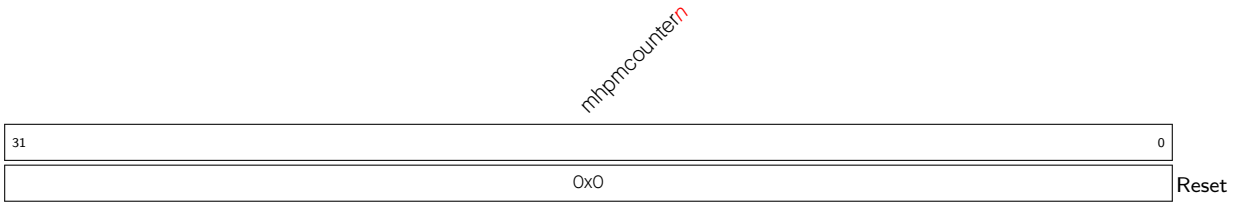
MCYCLE Configures the lower 32 bits of the clock cycle counter. (R/W)

Register 2.12. minstret (0xB02)



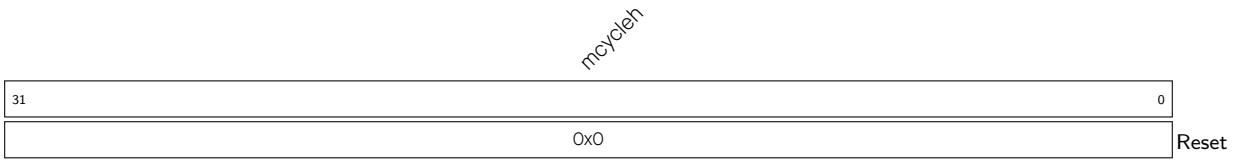
MINSTRET Configures the lower 32 bits of the instruction counter. (R/W)

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)



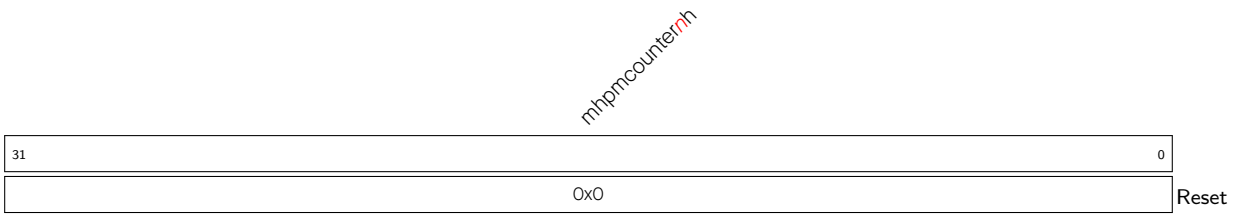
MCYCLEH Configures the higher 32 bits of the clock cycle counter. (R/W)

Register 2.15. minstreth (0xB82)



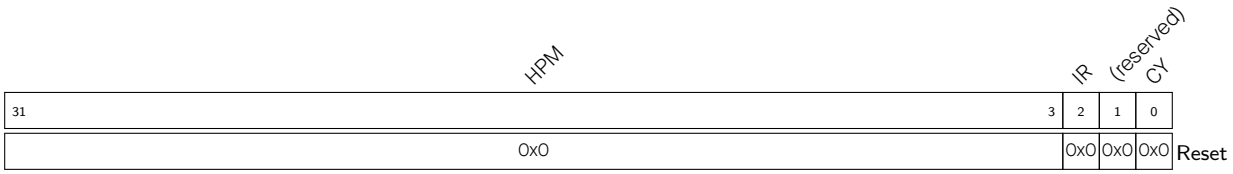
MINSTRETH Configures the higher 32 bits of the instruction counter. (R/W)

Register 2.16. mhpmcounter*n*h (*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 Configure whether the instruction counter increments.
0: The counter does not count
1: The counter increments
(R/W)

CY Configure whether the clock cycle counter increments.
0: The counter does not count
1: The counter increments
(R/W)

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 $mtvec + 4 * ID$, 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 starts up, the base address of the vector table is initialized to the boot address 0x50000000. After startup, the base address can be changed by writing to the mtvec CSR. For more information about CSRs, see Section 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. The default value for this register is 0x50100000, which corresponds to the starting address of the LP ROM.

2.4.1 Interrupts

The LP CPU in the ESP32-P4 supports 18 interrupt inputs. Each interrupt corresponds to an ID, which is denoted as n (where n = 3, 7, 11, 16 to 30), and has a specific entry address of $mtvec + 4 * n$. These interrupts are associated with specific peripheral sources. For the mapping between peripheral sources and interrupt IDs, please refer to 2.4-1.

Table 2.4-1. LP CPU Interrupt Mapping

Interrupt ID	Peripheral Source
3	LP_SW_INTR
7	LP_UART_INTR
11	LP_SPI_INTR
16	LP_TRNG_INTR
17	LP_I2C_INTR
18	LP_GPIO_INTR
19	LP_ADC_INTR
20	LP_TOUCH_INTR
21	LP_TSENS_INTR
22	LP_EFUSE_INTR
23	LP_SYSREG_INTR
24	LP_ANAPERI_INTR
25	PMU_REG_0_INTR, PMU_REG_1_INTR
26	MB_HP_INTR, MB_LP_INTR
27	P_TIMER_REG_0_INTR, LP_TIMER_REG_1_INTR
28	LP_WDT_INTR
29	LP_RTC_INTR
30	HP_INTR

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

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 $mtvec + 4 * n$ 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
2	Illegal instructions
3	Breakpoints (EBREAK)
6	Misaligned atomic instructions

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-P4 system debugging overview, please refer to Chapter 1 *High-Performance CPU [to be added later]* > Figure 1.1 *Debug System Overview [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) to communicate with Debug Transport Hardware (e.g., ESP-Prog adapter). The Debug Transport Hardware connects the Debug Host to the CPU's Debug Transport Module (DTM) through a standard JTAG interface. The DTM provides access to the debug module (DM) using the Debug Module Interface (DMI).

ESP32-P4 features two DTMs interconnected in a daisy-chain configuration. Specifically, TAP0 connects to the HP CPU, while TAP1 is dedicated to the LP CPU.

The HP DM supports multicore debugging according to the specification RISC-V External Debug Support Version 0.13. It can simultaneously control both cores of the HP CPU. In contrast, the LP DM is designed for debugging the single-core LP CPU.

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

2.5.1 Features

The Low-Power CPU debug module has the following debugging features:

- Access to necessary information about the implementation
- CPU core halt and resume
- Read and write of CPU core registers (including CSRs)

- CPU core reset
- CPU halted on software breakpoint (planted breakpoint instruction)
- Hardware single-stepping
- Two hardware triggers (which can be used as breakpoints/watchpoints). See Section 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.

According to the specification, a hart has the following states: nonexistent, unavail, running, and halted. By default, the LP CPU is in the unavail state. To connect the LP CPU for debugging, users need to clear that state by configuring the LPPERI_CPU_REG register.

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

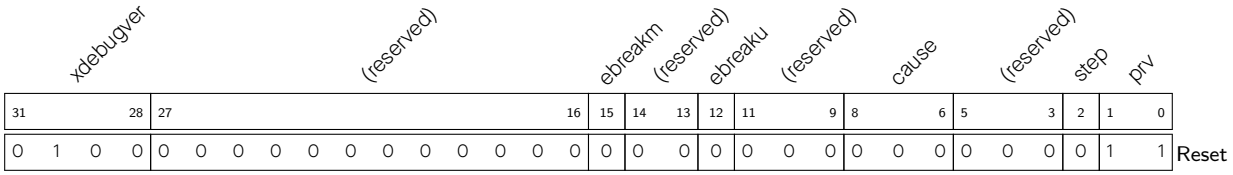
All debug module registers are implemented in accordance with the specification RISC-V External Debug Support Version 0.13. For more information, refer to the specification.

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 privileged mode

1: Enter debug mode

(R/W)

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

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

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

3: haltreq was set. (priority 2)

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

Other values: reserved for future use

(RO)

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

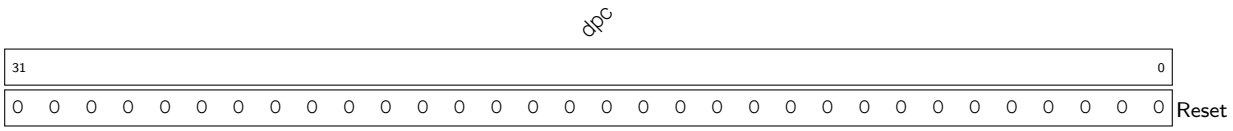
If the instruction does not complete due to an exception, the core will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set.

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

(R/W)

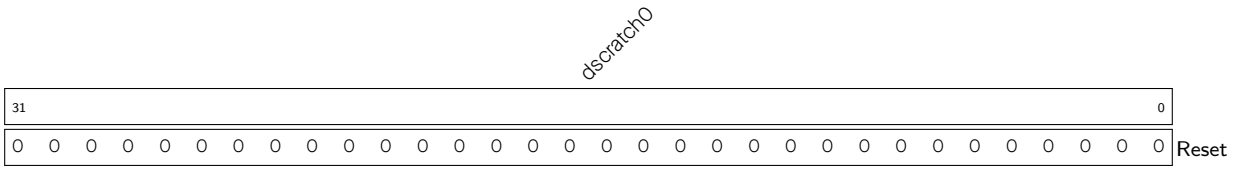
prv Contains the privilege level the core is operating in when debug mode is entered. A debugger can change this value to change the core's privilege level when exiting debug mode. Only **0x3** (machine mode) and **0x0** (user mode) are supported. (RO)

Register 2.19. dpc (0x7B1)



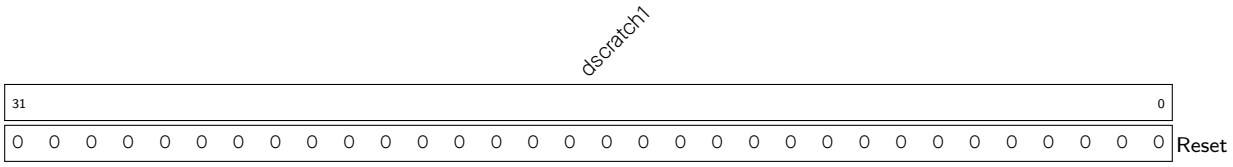
dpc Upon entry to debug mode, dpc is written with the address of the next instruction that will be executed. When resuming, the CPU core's PC is updated to the address stored in dpc. In debug mode, dpc can be modified. This field can be accessed in debug mode. (R/W)

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

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 three CSRs. See Section 2.6 for details. Among them, tdata1 and tdata2 are abstract CSRs, which means they are shadow registers for accessing internal registers in the trigger units, one at a time.

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

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

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

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

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

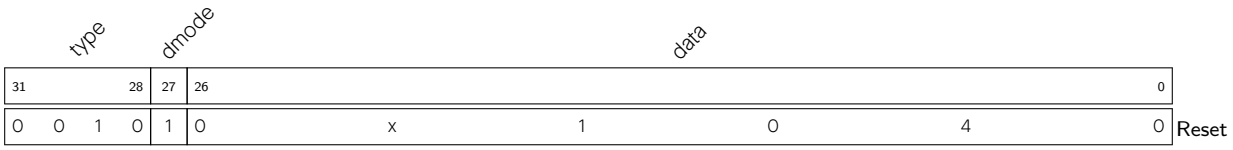
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

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)

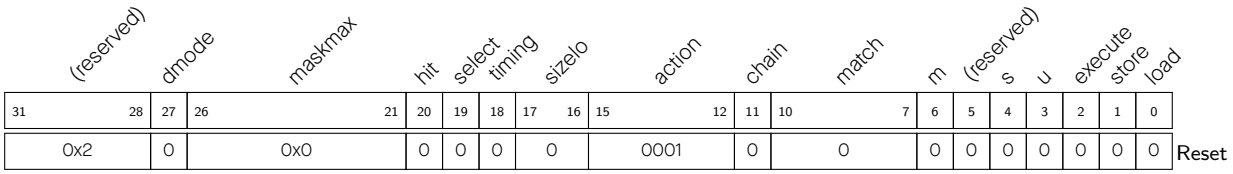
data Configures the abstract tdata1 content. This will always be interpreted as fields of mcontrol since only match type (0x2) triggers are supported. (R/W)

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...

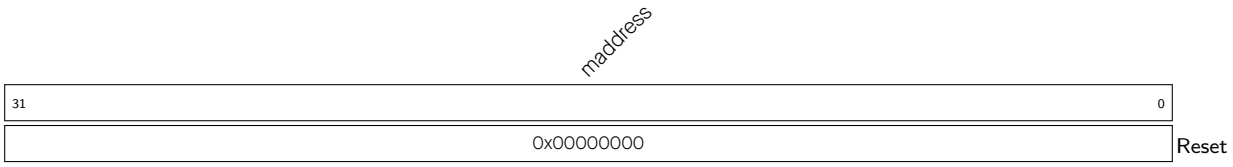
u Set this field to make the selected trigger operate in user mode. Operation in user mode is not supported. This field is always 0. (RO)

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

2.8.1 Memory Access

The ESP32-P4 LP CPU can access LP ROM, LP SRAM, and L2 SRAM. For more information, please refer to Section 5 *System and Memory*.

- LP ROM: 16 KB region starting from 0x5010_0000 to 0x5010_3FFF. It is used for instruction fetch, data read, etc.
- LP SRAM: 32 KB region starting from 0x5010_8000 ~ 0x5010_FFFF. It is used for instruction fetch, data read, data write, etc.
- L2 SRAM: 768 KB region starting from 0x4FF0_0000 to 0x4FFB_FFFF. It is used for instruction fetch, data read, data write, etc.

Note:
The LP CPU experiences significant latency when accessing L2 SRAM, with each access taking approximately 20 LP CPU clock cycles. However, it can access LP ROM and LP SRAM with no latency.

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

2.8.2 Peripheral Access

Table 5.3-2 in Chapter 5 *System and Memory* lists the peripherals accessible by the LP CPU and their base addresses.

2.9 Event Task Matrix Feature

The LP CPU on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows LP CPU's ETM tasks to be triggered by any peripherals' ETM events, or LP CPU's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the LP CPU. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

LP CPU can receive the following ETM task:

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

LP CPU can generate the following ETM events:

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

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.

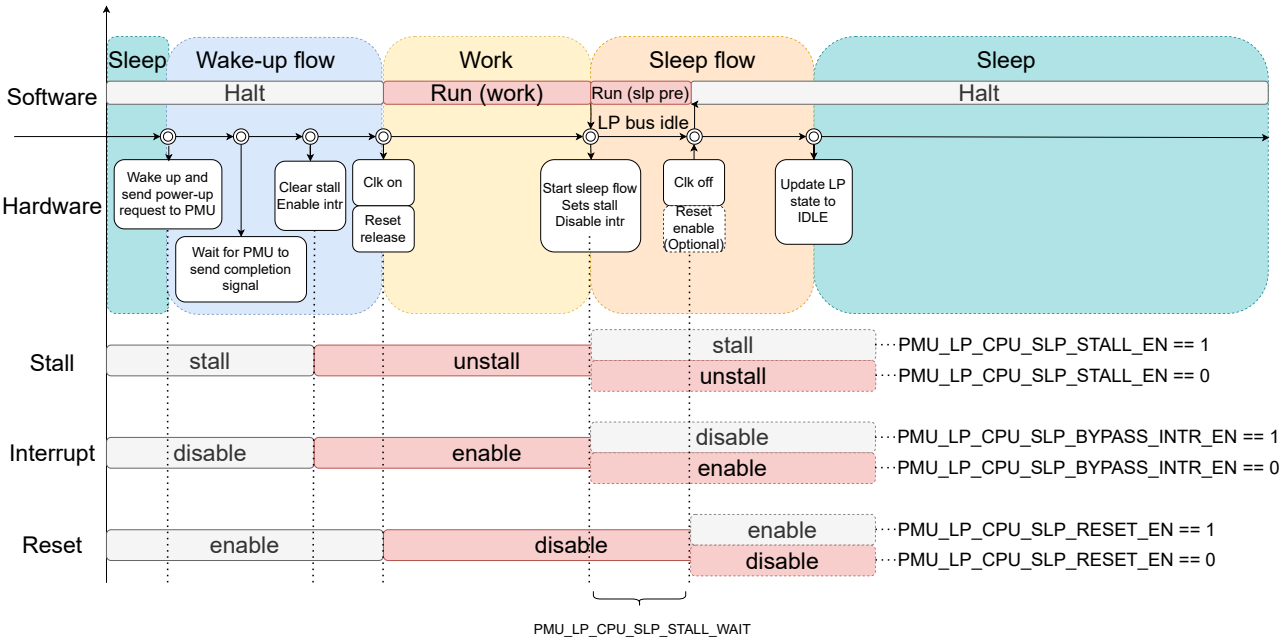


Figure 2.10-1. Wake-Up and Sleep Flow of LP CPU

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 0x400000
    * 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(9)	LP IO	The LP CPU can be waken up by the LP IO interrupt status register signal. For more information, please refer to Chapter 7 *GPIO Matrix and IO MUX*.
BIT(10)	LP UART	The LP CPU can be waken up when the LP UART receives a certain number of RX pulses. REG_UART_WAKEUP_EN needs to be enabled. For more information, please refer to Chapter 31 *UART Controller (UART)*.
BIT(13)	RTC timer	The LP CPU can be waken up by the RTC timer target 0 timeout interrupt. For more information, please refer to Chapter 5 *Low-Power Management [to be added later]*.
BIT(14)	Brownout detection	The LP CPU can be waken up when a brownout occurs. For more information, please refer to Chapter 19 *Brown-out Detector*.
BIT(17)	ETM	The LP CPU can be waken up by the ETM task. For more information, please refer to Chapter 11 *Event Task Matrix (ETM)*.
BIT(18)	RTC timer	The LP CPU can be waken up by the RTC timer target 0 timeout interrupt. For more information, please refer to Chapter 5 *Low-Power Management [to be added later]*.
BIT(19)	LP I2S audio	The LP CPU can be waken up when the predefined audio is received via I2S. For more details, please refer to Chapter 34 *I2S Controller (I2S)*.
BIT(22)	PMU_HP_TRIGGER_LP register	The HP CPU sets the register PMU_HP_TRIGGER_LP to wake up the LP CPU, and sets REG_HP_SW_TRIGGER_INT_CLR to clear this wake-up source.

[1] Value of the PMU_LP_CPU_WAKEUP_EN register

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.

Part II

System DMA

Focused on managing and transferring data within the system, this part explores controllers for Direct Memory Access (DMA), providing guidelines for efficient data movement supporting various peripherals and devices.

Chapter 3

GDMA Controller (GDMA-AHB, GDMA-AXI)

3.1 Overview

General Direct Memory Access (GDMA) is a feature that allows peripheral-to-memory, memory-to-peripheral, and memory-to-memory data transfer at high speed. The CPU is not involved in the GDMA transfer and therefore is more efficient with less workload.

ESP32-P4 has two types of general-purpose DMA controllers, namely GDMA-AHB and GDMA-AXI, to directly access the AHB bus or the AXI bus respectively. Both GDMA-AHB and GDMA-AXI have six independent channels, i.e., three transmit channels and three receive channels.

The GDMA-AHB channels are shared and can be assigned to I3C, UHCI, I2S, ADC, or RMT to access internal memory.

The GDMA-AXI channels are also shared and can be assigned to LCD, CAM, two general-purpose SPIs, PARLIO, AES, or SHA to access both internal and external memory.

GDMA-AHB and GDMA-AXI use configurable priority and weight arbitration schemes to manage peripherals' needs for bandwidth.

Unless otherwise specified, the term "GDMA controller" refers to both GDMA-AHB and GDMA-AXI in the following text.

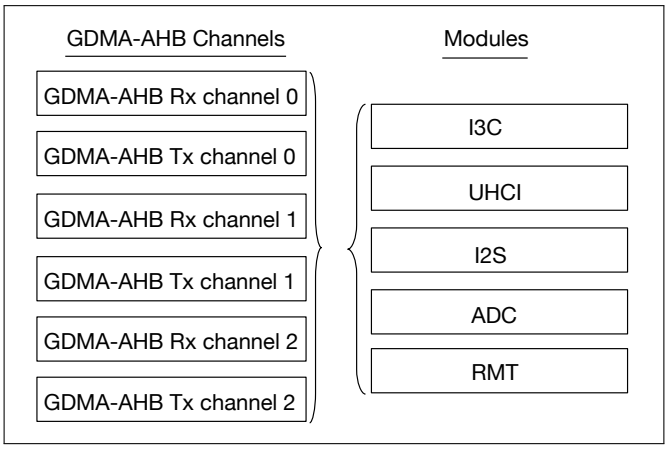


Figure 3.1-1. Modules that Share GDMA-AHB Channels

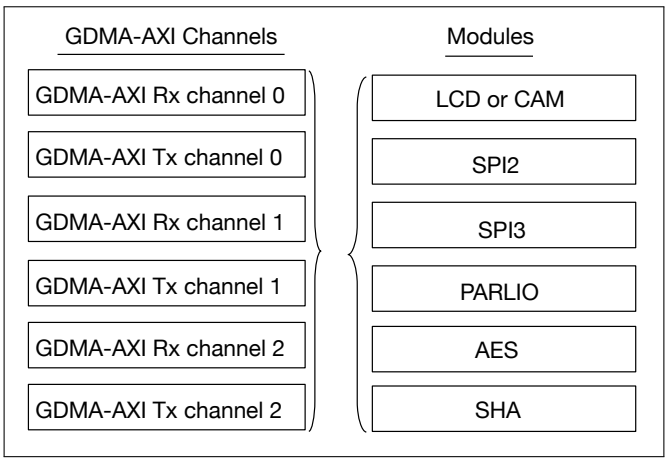


Figure 3.1-2. Modules that Share GDMA-AXI Channels

3.2 Features

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

- Architecture:
  - GDMA-AHB: AHB bus architecture
  - GDMA-AXI: AXI bus architecture, which gives the possibility to complete up to 8 transactions out of order and up to 8 outstanding transactions
- Programmable length of data to be transferred in bytes
- Access via any address and size
- Alignment:
  - GDMA-AHB:
    - Descriptor address: 1-word aligned
    - Data address and length (only internal memory is accessible): no requirements
  - GDMA-AXI:
    - Descriptor address: 2-word aligned
    - Data address and length:
      - Internal memory and non-encrypted external memory address space: no requirements
      - Encrypted external memory address space: 16-byte aligned
- Linked list of descriptors
- INCR burst transfer when accessing memory
- Three transmit channels and three receive channels for each controller
- Software-configurable selection of peripheral requesting its service
- Configurable channel priority and weight arbitration
- Support for memory transfer

- CRC calculation of data

3.3 Architecture

In ESP32-P4, all modules that need high-speed data transfer support GDMA. The GDMA controller and CPU data bus have access to the same address space in memory. Figure 3.3-1 shows the basic architecture of the GDMA controller.

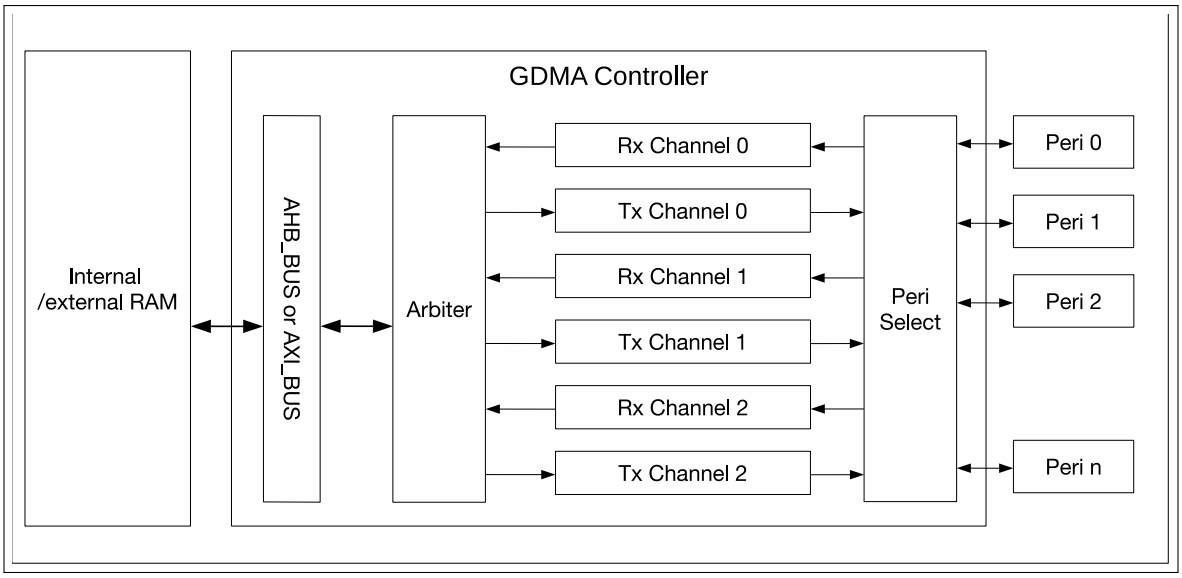


Figure 3.3-1. GDMA controller Architecture

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

GDMA-AHB and GDMA-AXI read data from or write data to internal memory or internal and external memory via AHB_BUS or AXI_BUS respectively. Before this, GDMA uses configurable arbitration schemes for channels requesting read or write access. For the available address range of internal and external memory, please see Chapter 5 *System and Memory*.

Software can use the GDMA through linked lists. GDMA-AHB linked lists are stored in internal memory, while GDMA-AXI linked lists are stored in internal or external memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number (ranging from 0 to 2). The GDMA reads an outlink*n* (i.e., a linked list of transmit descriptors) from memory and transmit data in corresponding memory according to the outlink*n*, or read an inlink*n* (i.e., a linked list of receive descriptors) and store received data into specific address space in memory according to the inlink*n*.

3.4 Functional Description

3.4.1 Linked List

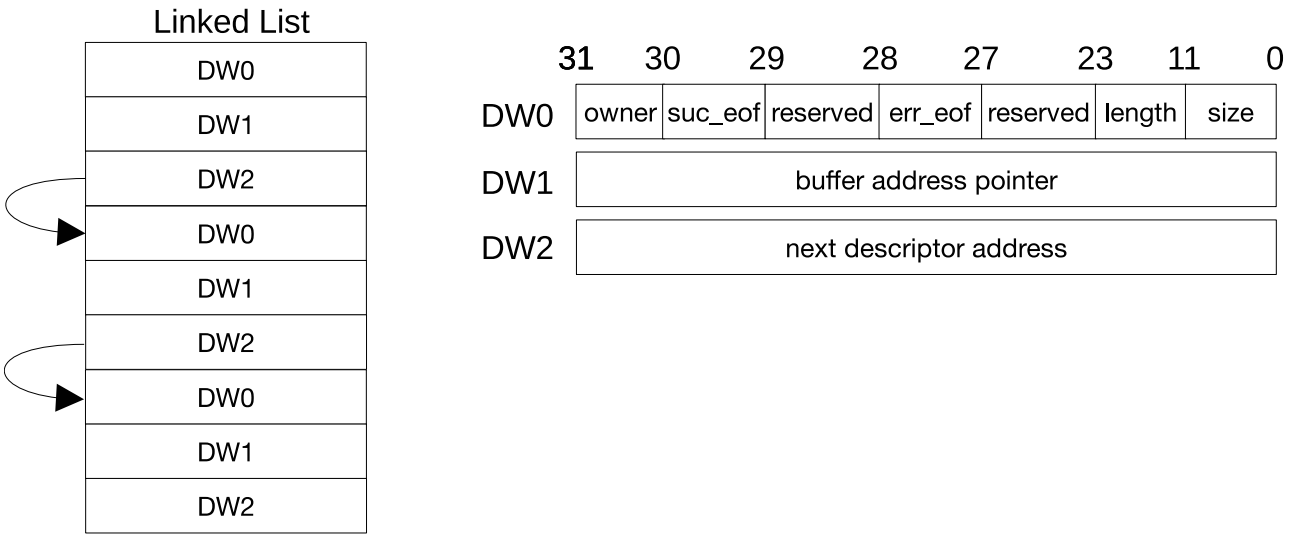


Figure 3.4-1. Structure of a Linked List

Figure 3.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of three words. Linked lists should be stored in the memory for the GDMA to be able to use them. The meanings of a descriptor's fields are as follows:

- owner (DW0) [31]: Specifies who is allowed to access the buffer that this descriptor points to.
  0: CPU can access the buffer.
  1: The GDMA controller can access the buffer.
  When the GDMA controller stops using the buffer, this bit in a receive descriptor is automatically cleared by hardware, while this bit in a transmit descriptor can only be automatically cleared by hardware if AHB/AXI_DMA_OUT_AUTO_WRBACK_CH*n* is set to 1. Software can disable automatic clearing by hardware by setting the AHB/AXI_DMA_OUT_LOOP_TEST_CH*n* or AHB/AXI_DMA_IN_LOOP_TEST_CH*n* bit. When software loads a linked list, this bit should be set to 1.
  Note: AHB/AXI_DMA_OUT is the prefix of transmit channel registers, and AHB/AXI_DMA_IN is the prefix of receive channel registers.
- suc_eof (DW0) [30]: Specifies whether the AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT or AHB/AXI_DMA_OUT_EOF_CH*n*_INT interrupt will be triggered when the data corresponding to this descriptor has been received or transmitted.
  0: No interrupt will be triggered after the current descriptor's successful transfer;
  1: An interrupt will be triggered after the current descriptor's successful transfer.
  For receive descriptors, software needs to clear this bit to 0, and hardware will set it to 1 after receiving data containing the EOF flag.
  For transmit descriptors, software needs to set this bit to 1 as needed.
  If software configures this bit to 1 in a descriptor, the GDMA will include the EOF flag in the data sent to the corresponding peripheral, indicating to the peripheral that this data segment marks the end of one

transfer phase.

- reserved (DW0) [29]: Reserved. The value of this bit does not matter.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  0: The received data does not have errors.
  1: The received data has errors.
  This bit is used only when UHCI or PARLIO uses the GDMA to receive data. When an error is detected in the received data segment corresponding to a descriptor, this bit in the receive descriptor is set to 1 by hardware.
- reserved (DW0) [27:24]: Reserved.
- length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored in the buffer.
- size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
- buffer address pointer (DW1): Address of the buffer.
- next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value is 0. For GDMA-AHB, this field can only point to the internal memory space; for GDMA-AXI, this field can point to the internal or external memory space.

> **Note:**
> Please note that addresses of GDMA-AHB descriptors should be 4-byte aligned, and addresses of GDMA-AXI descriptors should be 8-byte aligned.

If the length of data received is smaller than the size of the buffer, the GDMA controller will not use the available space of the buffer in the next transaction.

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

The GDMA controller can transfer data from memory to peripheral (transmit) and from peripheral to memory (receive). A transmit channel transfers data in the specified memory location to a peripheral's transmitter via an outlink*n*, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlink*n*.

Every transmit and receive channel can be connected to any peripheral with the GDMA feature. Table 3.4-1 and Table 3.4-2 illustrate how to select the peripheral to be connected via registers. "Dummy-*n*" corresponds to register values for memory-to-memory data transfer. When a channel is connected to a peripheral, the rest of the channels cannot be connected to that peripheral.

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

AHB_DMA_PERI_IN_SEL_CH*n* AHB_DMA_PERI_OUT_SEL_CH*n*	Peripheral
0	I3C
1	Dummy-1

2	UHCI
3	I2S0
4	I2S1
5	I2S2
6 ~ 7	Dummy-6 ~ 7
8	ADC
9	Dummy-9
10	RMT
11 ~ 15	Dummy-11 ~ 15
16 ~ 63	Invalid

Table 3.4-2. GDMA-AXI Selecting Peripherals via Register Configuration

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

3.4.3 Memory-to-Memory Data Transfer

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

To access external memory, please use GDMA-AXI.

3.4.4 Enabling GDMA

The software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures the AHB/AXI_DMA_INLINK_ADDR_CH*n* field with the address of the first receive descriptor, and sets the AHB/AXI_DMA_INLINK_START_CH*n* bit to enable GDMA. When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures the AHB/AXI_DMA_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets the AHB/AXI_DMA_OUTLINK_START_CH*n* bit to enable GDMA. The AHB/AXI_DMA_INLINK_START_CH*n* bit and AHB/AXI_DMA_OUTLINK_START_CH*n* bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a DMA transfer that is already started. Naively, it would seem to be possible to do this by clearing the EOF bit of the final descriptor in the existing list and

setting its next descriptor address pointer field (DW2) to the first descriptor of the to-be-added list. However, this strategy fails if the existing DMA transfer is almost or entirely finished. Instead, the GDMA controller has specialized logic to make sure a DMA transfer can be continued or restarted: if the transfer is ongoing, the controller will make sure to take the appended descriptors into account; if the transfer has already finished, the controller will restart with the new descriptors. This is implemented by the Restart function.

When using the Restart function, software needs to rewrite the address of the first descriptor in the new list to DW2 of the last descriptor in the loaded list, and set AHB/AXI_DMA_INLINK_RESTART_CH*n* bit or AHB/AXI_DMA_OUTLINK_RESTART_CH*n* bit (these two bits are cleared automatically by hardware). As shown in Figure 3.4-2, by doing so hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

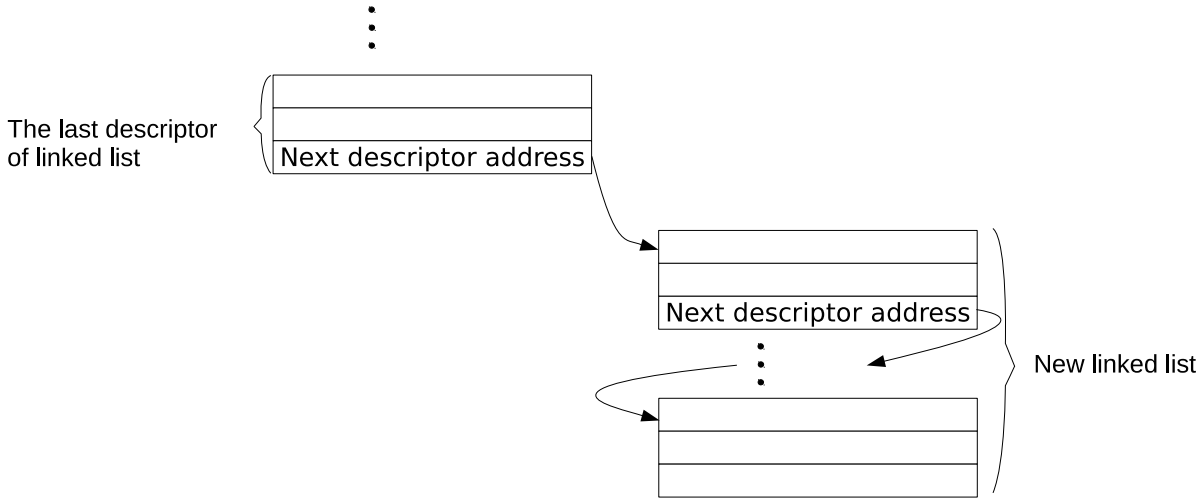


Figure 3.4-2. Relationship among Linked Lists

3.4.5 Linked List Reading Process

Once configured and enabled by software, the GDMA controller starts to read the linked list from memory. The GDMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, the corresponding GDMA channel will start data transfer. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT or AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the buffer is accessed by the CPU. In this case, the owner bit fails the check. The owner bit will not be checked if AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* or AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* is 0.
- Descriptor address check, which checks if the descriptor address is located in configured internal memory space for GDMA-AHB, and configured internal or external memory space for GDMA-AXI. If a GDMA-AHB descriptor points to AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR (the value should be located in internal memory), it passes the check. If a GDMA-AXI descriptor points to AXI_DMA_ACCESS_INTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_INTR_MEM_END_ADDR (the value should be located in internal memory), or AXI_DMA_ACCESS_EXTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_EXTR_MEM_END_ADDR (the value

should be located in external memory), it passes the check. For details, please refer to Section 3.4.7.

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

3.4.6 EOF

Note: In this chapter, EOF of transmit descriptors refers to suc_eof (i.e., bit 30 of DW0), while EOF of receive descriptors refers to both suc_eof and err_eof (i.e., bit 28 of DW0).

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

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

- AHB/AXI_DMA_OUT_EOF_CH*n*_INT, generated when the suc_eof bit of any descriptor in the linked list is set, and the data corresponding to this descriptor has been transmitted. This interrupt is enabled by setting the AHB/AXI_DMA_OUT_EOF_CH*n*_INT_ENA bit.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT, generated when the suc_eof bit of the last descriptor in the linked list is set, and the data corresponding to the last descriptor has been transmitted. This interrupt is enabled by setting the AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA bit.

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

- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT, generated when a data segment with an EOF flag has been received. This interrupt is enabled by setting the AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA bit.
- (UHCI and PARLIO only) AHB/AXI_DMA_IN_ERR_CH*n*_EOF_INT, generated when a data segment corresponding to a descriptor has been received with errors. This interrupt is enabled by setting the AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA bit, and is valid only when the channel is connected to UHCI or PARLIO.

When detecting an AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT or AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt, software can read the value of the AHB/AXI_DMA_OUT_EOF_DES_ADDR_CH*n* or AHB/AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* field, which stores the address of the finished descriptor. Therefore, the software can tell which descriptors have been used and reclaim them as needed. For RX, the address of the descriptor can also be stored to the AHB/AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* field when an AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT interrupt is triggered.

3.4.7 Accessing Memory

Any transmit and receive channels of GDMA-AHB and GDMA-AXI can access the internal memory address space configured by AHB/AXI_DMA_ACCESS_INTR_MEM_START_ADDR and AHB/AXI_DMA_ACCESS_INTR_MEM_END_ADDR. Moreover, the transmit and receive channels of GDMA-AXI can access the external memory address space configured by AXI_DMA_ACCESS_EXTR_MEM_START_ADDR and AXI_DMA_ACCESS_EXTR_MEM_END_ADDR.

To improve data transfer efficiency, GDMA can send data in burst mode. For GDMA-AHB, this mode is disabled by default, and can be enabled for receive channels by setting AHB_DMA_IN_DATA_BURST_EN_CH*n*, and enabled for transmit channels by setting AHB_DMA_OUT_DATA_BURST_EN_CH*n*. For GDMA-AXI, this mode is enabled by default, and cannot be disabled. Users can configure the number of data bytes in a single AXI

burst (burst length) to 8, 16, 32, 64, or 128 bytes via AXI_DMA_IN_BURST_SIZE_SEL_CH*n* and AXI_DMA_OUT_BURST_SIZE_SEL_CH*n*.

When GDMA-AHB accesses the configured internal memory space, there are no requirements for descriptor field alignment. When GDMA-AXI accesses the configured internal memory and non-encrypted external memory space, there are no requirements for descriptor field alignment; when it accesses the encrypted external memory space (see Chapter 7 *External Memory Encryption and Decryption [to be added later]*), buffer address pointer and data should be 16-byte aligned, and other fields are not required to be aligned.

3.4.8 Arbitration

To ensure timely response to peripherals running at a high speed with low latency (such as SPI), the GDMA controller implements two arbitration schemes, based on channel priority and channel weight respectively.

- Priority arbitration: Each channel can be assigned a priority from 0 ~ 5 (in total 6 levels). The larger the number, the higher the priority. When several channels perform data transfers at the same time, the GDMA would respond to the transfer requests according to priority levels, and the channel with higher priority would get a response more timely.
- Weight arbitration:
  - Each channel is assigned a weight (i.e., the number of tokens) from 0 ~ 15. The GDMA divides the AHB or AXI bus clock period into multiple time slots, and in each time slot, the number of transfers performed by a channel is determined by the number of its tokens. Every time a channel performs a data transfer, one of its tokens is spent. If all of its tokens have been spent in a time slot, then this channel's transfer request will no longer be responded to until the time slot expires, or until tokens of all channels have been spent and the new time slot starts.
  - If the number of tokens assigned to a channel is not zero, the GDMA waits for this channel's tokens to be spent even if it does not have data transfer requests, and will not exit from this time slot ahead of expiration. This leads to a waste of bandwidth. Therefore, the GDMA provides weight arbitration optimization. When this feature is enabled, a channel without data transfer requests will not be involved in the arbitration and its tokens will be ignored. When all channels no longer have transfer requests, the GDMA arbiter exits from the current time slot before expiration. This way, the arbitration response is accelerated and the transfer efficiency is improved.

Note:

- If channels have the same weight but different priorities, on condition that the number of bytes to be transferred is the same and not large, channels with higher priorities would finish data transfers first.
- If channels have different priorities and weights, on condition that the number of bytes to be transferred is the same, channels with higher weights would be allocated more bandwidth and finish data transfers first.

3.4.9 CRC Calculation

The GDMA has a CRC calculation module for both TX and RX. This module supports arbitrary polynomials and 8-bit parallel computation, namely 8 bits at a time. The CRC result is up to 32-bit wide for GDMA-AHB, and up

to 16-bit for GDMA-AXI. It is calculated for each linked list descriptor by descriptor and becomes invalid when the Restart function is triggered during transfer.

For easier understanding of register configuration rules, assume:

- The 8-bit data to calculate each time is data[7:0].
- The required CRC result is 5-bit wide, and the polynomial used is $crc[4:0] = x \oplus 5 + x \oplus 2 + 1$;
- The initial CRC value is 0x1F, and the intermediate result calculated in each clock cycle is crc_tmp[4:0].

Taking as an example the GDMA reading data from memory and transmitting the data to peripherals, the following fields are provided by hardware for CRC computation:

- AHB/AXI_DMA_OUT_CRC_INIT_DATA_CH*n*: Configures the initial CRC value, i.e., 0x1F in the above example. AHB_DMA_OUT_CRC_INIT_DATA_CH*n* can be configured with a 32-bit value at the maximum, while AXI_DMA_OUT_CRC_INIT_DATA_CH*n* can be configured with a 16-bit value at the maximum, meaning that the CRC result could be 32 bits or 16 bits at most.
- AHB/AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*: Stores the computation result of all data in a DMA transfer (until data pointed by the last descriptor in the linked list has been transferred). The computation result could be 32 bits at most.
- AHB/AXI_DMA_TX_CRC_WIDTH_CH*n*: Configures the CRC result length.
- AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n*, which latches the CRC result.
- AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*: Indicates which bits in crc_tmp[4:0] are used to compute crc[x] (x is configured via AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n*). If AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* is configured as 0xF, it means that only crc_tmp[3], crc_tmp[2], crc_tmp[1], and crc_tmp[0] are used for computation, and other bits are not used (in this case crc_tmp[4]).
- AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*: Indicates which bits in data[7:0] are used to compute crc[x] (x is configured via AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*). GDMA calculates the CRC in the unit of byte, so only the lower 8 bits of this field needs to be configured. If AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* is configured as 0xF, it means that only data[3], data[2], data[1], and data[0] are used for computation.

CRC calculation is achieved in the GDMA by configuring a parallel calculation matrix derived from the polynomial. If the expected CRC result is n bits, then the matrix has n rows and $n + 8$ columns.

Based on the assumptions at the beginning of this section, the calculation matrix would be 5 rows and 13 columns, and the actual hardware CRC calculation formulas would be as follows ($\oplus$represents the XOR operation):

$$crc[0] = crc\_tmp[0] \oplus crc\_tmp[2] \oplus crc\_tmp[3] \oplus data[0] \oplus data[3] \oplus data[5] \oplus data[6]$$
$$crc[1] = crc\_tmp[1] \oplus crc\_tmp[3] \oplus crc\_tmp[4] \oplus data[1] \oplus data[4] \oplus data[6] \oplus data[7]$$
$$crc[2] = crc\_tmp[0] \oplus crc\_tmp[3] \oplus crc\_tmp[4] \oplus data[0] \oplus data[2] \oplus data[3] \oplus data[6] \oplus data[7]$$
$$crc[3] = crc\_tmp[0] \oplus crc\_tmp[1] \oplus crc\_tmp[4] \oplus data[1] \oplus data[4] \oplus data[3] \oplus data[7]$$
$$crc[4] = crc\_tmp[1] \oplus crc\_tmp[2] \oplus data[2] \oplus data[4] \oplus data[5]$$

The above calculation formula can be abstracted as a switch matrix shown in Figure 3.4-3:

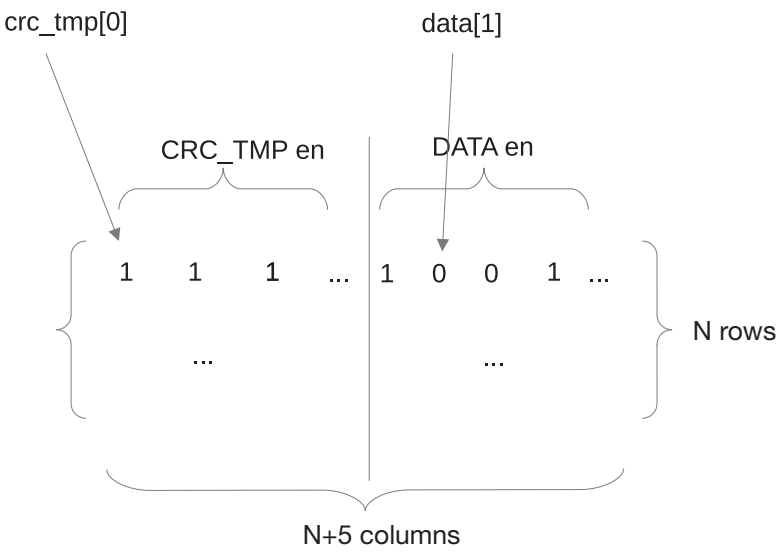


Figure 3.4-3. CRC Calculation Matrix

Note:
The CRC calculation described above is a Linear Feedback Shift Register (LFSR) implementation of the bytewise CRC algorithm. The formulas can be generated with some online tools according to the polynomial and the data width for parallel computation (always 8 bits for ESP32-P4).

3.5 Event Task Matrix Feature

The GDMA controller on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows GDMA's ETM tasks to be triggered by any peripherals' ETM events, or GDMA's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to GDMA. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

GDMA can receive the following ETM tasks:

- GDMA_AHB/AXI_TASK_IN_START_CH*n*: Enables the corresponding RX channel *n* for data transfer.
- GDMA_AHB/AXI_TASK_OUT_START_CH*n*: Enables the corresponding TX channel *n* for data transfer.

Note:
Above ETM tasks can achieve the same functions as CPU configuring AHB/AXI_DMA_INLINK_START_CH*n* and AHB/AXI_DMA_OUTLINK_START_CH*n*. When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB/AXI_DMA_OUT_ETM_EN_CH*n* is 1, only ETM tasks can be used to configure the transfer direction and enable the corresponding GDMA channel. When AHB/AXI_DMA_IN_ETM_EN_CH*n* or AHB_DMA_OUT_ETM_EN_CH*n* is 0, only CPU can be used to enable the corresponding GDMA channel.

GDMA can generate the following ETM events:

- GDMA_AHB/AXI_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.

- GDMA_AHB/AXI_EVT_IN_SUC_EOF_CH*n*: Indicates that the data corresponding to a receive descriptor has been received via channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_IN_FIFO_EMPTY_CH*n*: Indicates that the RX FIFO has become empty.
- GDMA_AHB/AXI_EVT_IN_FIFO_FULL_CH*n*: Indicates that the RX FIFO has become full.
- GDMA_AHB/AXI_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- GDMA_AHB/AXI_EVT_OUT_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that the data corresponding to the last transmit descriptors has been sent via transmit channel *n* and the EOF bit of this descriptor is 1.
- GDMA_AHB/AXI_EVT_OUT_FIFO_EMPTY_CH*n*: Indicates that the TX FIFO has become empty.
- GDMA_AHB/AXI_EVT_OUT_FIFO_FULL_CH*n*: Indicates that the TX FIFO has become full.

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

3.6 Interrupts

ESP32-P4's GDMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- AHB/AXI_PDMA_IN_CH0_INTR
- AHB/AXI_PDMA_IN_CH1_INTR
- AHB/AXI_PDMA_IN_CH2_INTR
- AHB/AXI_PDMA_OUT_CH0_INTR
- AHB/AXI_PDMA_OUT_CH1_INTR
- AHB/AXI_PDMA_OUT_CH2_INTR

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

The AHB/AXI_PDMA_IN_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- AHB/AXI_DMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.
- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT: Triggered when the suc_eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received via receive channel *n*.
- AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the data segment corresponding to a descriptor received via receive channel *n*. This interrupt is used only for UHCI (UART0 or UART1) or PARLIO.
- AHB/AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel *n*.

- AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT: Triggered when a receive descriptor on receive channel *n* fails any of the two descriptor checks.
- AHB_DMA_INFIFO_OVF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the RX FIFO of GDMA-AHB or the L1/L2/L3 RX FIFO of GDMA-AXI overflows.
- AHB_DMA_INFIFO_UDF_CH*n*_INT or AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the RX FIFO of GDMA-AHB or the L1/L2/L3 RX FIFO of GDMA-AXI underflows.

The AHB/AXI_PDMA_OUT_CH*n*_INTR (*n* is 0 ~ 2) signals are generated by the following interrupt sources:

- AHB/AXI_DMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- AHB/AXI_DMA_OUT_EOF_CH*n*_INT: Triggered when the suc_eof bit in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If AHB/AXI_DMA_OUT_EOF_MODE_CH*n* is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters GDMA's transmit channel; if AHB/AXI_DMA_OUT_EOF_MODE_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.
- AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when a transmit descriptor on transmit channel *n* fails any of the two descriptor checks.
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel *n*.
- AHB_DMA_OUTFIFO_OVF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT: Triggered when the TX FIFO of GDMA-AHB or the L1/L2/L3 TX FIFO of GDMA-AXI overflows.
- AHB_DMA_OUTFIFO_UDF_CH*n*_INT or AXI_DMA_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT: Triggered when the TX FIFO of GDMA-AHB or the L1/L2/L3 TX FIFO of GDMA-AXI underflows.

3.7 Programming Procedures

The clock gating for GDMA can be configured via HP_SYS_CLKRST_AHB/AXI_PDMA_SYS_CLK_EN, and is enabled by default. GDMA can be reset by configuring HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA.

3.7.1 Programming Procedures for GDMA's Transmit Channel

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

1. Set AHB/AXI_DMA_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer.
2. Load an outlink, and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
3. Configure AHB/AXI_DMA_PERI_OUT_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 3.4-1 and Table 3.4-2.
4. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.

5. Configure and enable the corresponding peripheral (SPI2, UHCI (UART0 or UART1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals.
6. Wait for the AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

3.7.2 Programming Procedures for GDMA's Receive Channel

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

1. Set AHB/AXI_DMA_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer.
2. Load an inlink, and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
3. Configure AHB/AXI_DMA_PERI_IN_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 3.4-1 and Table 3.4-2.
4. Set AHB/AXI_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
5. Configure and enable the corresponding peripheral (SPI2, UHCI (UART0 or UART1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals.

3.7.3 Programming Procedures for Memory-to-Memory Transfer

To transfer data from one memory location to another, GDMA should be configured by software as follows:

1. Set AHB/AXI_DMA_OUT_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer.
2. Set AHB/AXI_DMA_IN_RST_CH*n* first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer.
3. Load an outlink, and configure AHB/AXI_DMA_OUTLINK_ADDR_CH*n* with address of the first transmit descriptor.
4. Load an inlink, and configure AHB/AXI_DMA_INLINK_ADDR_CH*n* with address of the first receive descriptor.
5. Set AHB/AXI_DMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer.
6. Set AHB/AXI_DMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer.
7. Set AHB/AXI_DMA_INLINK_START_CH*n* to enable GDMA's receive channel for data transfer.
8. If the suc_eof bit is set in a transmit descriptor, an AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT interrupt will be triggered when the data segment corresponding to this descriptor has been transmitted.

3.7.4 Programming Procedures for Channel Priority and Weight

The priority arbitration can be configured as follows:

1. Configure the channel priority for TX and RX respectively via AHB/AXI_DMA_TX_PRI_CH*n* and AHB/AXI_DMA_RX_PRI_CH*n*.

The weight arbitration can be configured as follows:

1. Configure the time slot for TX and RX respectively via AHB/AXI_DMA_ARB_TIMEOUT_TX and AHB/AXI_DMA_ARB_TIMEOUT_RX.
2. Configure the number of tokens for TX and RX respectively via AHB/AXI_DMA_TX_CH_ARB_WEIGH_CH*n* and AHB/AXI_DMA_RX_CH_ARB_WEIGH_CH*n*.
3. Enable weight arbitration optimization for TX and RX respectively by clearing AHB/AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* and AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*.
4. Enable the arbitration for TX and RX respectively by setting AHB/AXI_DMA_WEIGHT_EN_TX and AHB/AXI_DMA_WEIGHT_EN_RX.

3.7.5 Programming Procedures for CRC Calculation

The CRC calculation registers can be configured as follows:

1. Configure which bits in crc_tmp and data are used in the matrix via AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n*, AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*.
2. Latch the values of the above four fields by first writing 1 and then 0 to AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n*. Then the configuration for a row is complete.
3. Repeat Step 1 and Step 2 until all rows in the matrix are set up, and then start DMA transfer.

3.8 Register Summary

3.8.1 GDMA-AHB Register Summary

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

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

Name	Description	Address	Access
Interrupt Registers			
AHB_DMA_IN_INT_RAW_CH0_REG	RX channel 0 raw interrupt status register	0x0000	R/WTC/SS
AHB_DMA_IN_INT_ST_CH0_REG	RX channel 0 masked interrupt status register	0x0004	RO
AHB_DMA_IN_INT_ENA_CH0_REG	RX channel 0 interrupt enable register	0x0008	R/W
AHB_DMA_IN_INT_CLR_CH0_REG	RX channel 0 interrupt clear register	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX channel 1 raw interrupt status register	0x0010	R/WTC/SS
AHB_DMA_IN_INT_ST_CH1_REG	RX channel 1 masked interrupt status register	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX channel 1 interrupt enable register	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX channel 1 interrupt clear register	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX channel 2 raw interrupt status register	0x0020	R/WTC/SS
AHB_DMA_IN_INT_ST_CH2_REG	RX channel 2 masked interrupt status register	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX channel 2 interrupt enable register	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX channel 2 interrupt clear register	0x002C	WT
AHB_DMA_OUT_INT_RAW_CH0_REG	TX channel 0 raw interrupt status register	0x0030	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH0_REG	TX channel 0 masked interrupt status register	0x0034	RO
AHB_DMA_OUT_INT_ENA_CH0_REG	TX channel 0 interrupt enable register	0x0038	R/W
AHB_DMA_OUT_INT_CLR_CH0_REG	TX channel 0 interrupt clear register	0x003C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX channel 1 raw interrupt status register	0x0040	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX channel 1 masked interrupt status register	0x0044	RO
AHB_DMA_OUT_INT_ENA_CH1_REG	TX channel 1 interrupt enable register	0x0048	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX channel 1 interrupt clear register	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX channel 2 raw interrupt status register	0x0050	R/WTC/SS

GoBack

Name	Description	Address	Access
AHB_DMA_OUT_INT_ST_CH2_REG	TX channel 2 masked interrupt status register	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX channel 2 interrupt enable register	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX channel 2 interrupt clear register	0x005C	WT
Debug Registers			
AHB_DMA_AHB_TEST_REG	Reserved	0x0060	R/W
Configuration Registers			
AHB_DMA_MISC_CONF_REG	Miscellaneous register	0x0064	R/W
AHB_DMA_IN_CONF0_CH0_REG	Configuration register 0 of RX channel 0	0x0070	R/W
AHB_DMA_IN_CONF1_CH0_REG	Configuration register 1 of Rx channel 0	0x0074	R/W
AHB_DMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x007C	varies
AHB_DMA_IN_LINK_CH0_REG	Linked list descriptor configuration and control register of RX channel 0	0x0080	varies
AHB_DMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x00D0	R/W
AHB_DMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x00D4	R/W
AHB_DMA_OUT_PUSH_CH0_REG	Push control register of TX channel 0	0x00DC	varies
AHB_DMA_OUT_LINK_CH0_REG	Linked list descriptor configuration and control register of TX channel 0	0x00E0	varies
AHB_DMA_IN_CONF0_CH1_REG	Configuration register 0 of RX channel 1	0x0130	R/W
AHB_DMA_IN_CONF1_CH1_REG	Configuration register 1 of RX channel 1	0x0134	R/W
AHB_DMA_IN_POP_CH1_REG	Pop control register of RX channel 1	0x013C	varies
AHB_DMA_IN_LINK_CH1_REG	Linked list descriptor configuration and control register of RX channel 1	0x0140	varies
AHB_DMA_OUT_CONF0_CH1_REG	Configuration register 0 of TX channel 1	0x0190	R/W
AHB_DMA_OUT_CONF1_CH1_REG	Configuration register 1 of TX channel 1	0x0194	R/W
AHB_DMA_OUT_PUSH_CH1_REG	Push control register of TX channel 1	0x019C	varies
AHB_DMA_OUT_LINK_CH1_REG	Linked list descriptor configuration and control register of TX channel 1	0x01A0	varies
AHB_DMA_IN_CONF0_CH2_REG	Configuration register 0 of RX channel 2	0x01F0	R/W
AHB_DMA_IN_CONF1_CH2_REG	Configuration register 1 of RX channel 2	0x01F4	R/W

Name	Description	Address	Access
AHB_DMA_IN_POP_CH2_REG	Pop control register of RX channel 2	0x01FC	varies
AHB_DMA_IN_LINK_CH2_REG	Linked list descriptor configuration and control register of RX channel 2	0x0200	varies
AHB_DMA_OUT_CONF0_CH2_REG	Configuration register 0 of TX channel 2	0x0250	R/W
AHB_DMA_OUT_CONF1_CH2_REG	Configuration register 1 of TX channel 2	0x0254	R/W
AHB_DMA_OUT_PUSH_CH2_REG	Push control register of TX channel 2	0x025C	varies
AHB_DMA_OUT_LINK_CH2_REG	Linked list descriptor configuration and control register of TX channel 2	0x0260	varies
AHB_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX channel 0 CRC initial value configuration register	0x02BC	R/W
AHB_DMA_TX_CRC_WIDTH_CH0_REG	TX channel 0 CRC result width configuration register	0x02C0	R/W
AHB_DMA_OUT_CRC_CLEAR_CH0_REG	TX channel 0 CRC result clear register	0x02C4	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX channel 0 CRC result register	0x02C8	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX channel 0 CRC intermediate result mask register	0x02CC	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x02D0	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX channel 0 CRC data input mask register	0x02D4	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX channel 0 CRC data input mask target register	0x02D8	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX channel 0 arbitration weight configuration register	0x02DC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX channel 0 weight arbitration optimization enable register	0x02E0	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX channel 1 CRC initial value configuration register	0x02E4	R/W
AHB_DMA_TX_CRC_WIDTH_CH1_REG	TX channel 1 CRC result width configuration register	R/W	
AHB_DMA_OUT_CRC_CLEAR_CH1_REG	TX channel 1 CRC result clear register	0x02EC	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX channel 1 CRC result register	0x02F0	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH1_REG	TX channel 1 CRC intermediate result mask register	0x02F4	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH1_REG	TX channel 1 CRC intermediate result mask target register	0x02F8	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX channel 1 CRC data input mask register	0x02FC	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX channel 1 CRC data input mask target register	0x0300	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX channel 1 arbitration weight configuration register	0x0304	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX channel 1 weight arbitration optimization enable register	0x0308	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX channel 2 CRC initial value configuration register	0x030C	R/W

Name	Description	Address	Access
AHB_DMA_TX_CRC_WIDTH_CH2_REG	TX channel 2 CRC result width configuration register	0x0310	R/W
AHB_DMA_OUT_CRC_CLEAR_CH2_REG	TX channel 2 CRC result clear register	0x0314	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX channel 2 CRC result register	0x0318	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH2_REG	TX channel 2 CRC intermediate result mask register	0x031C	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH2_REG	TX channel 2 CRC intermediate result mask target register	0x0320	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX channel 2 CRC data input mask register	0x0324	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX channel 2 CRC data input mask target register	0x0328	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH2_REG	TX channel 2 arbitration weight configuration register	0x032C	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH2_REG	TX channel 2 weight arbitration optimization enable register	0x0330	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH0_REG	RX channel 0 CRC initial value configuration register	0x0334	R/W
AHB_DMA_RX_CRC_WIDTH_CH0_REG	RX channel 0 CRC result width configuration register	0x0338	R/W
AHB_DMA_IN_CRC_CLEAR_CH0_REG	RX channel 0 CRC result clear register	0x033C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX channel 0 CRC result register	0x0340	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH0_REG	RX channel 0 CRC intermediate result mask register	0x0344	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0348	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	RX channel 0 CRC data input mask register	0x034C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0350	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX channel 0 arbitration weight configuration register	0x0354	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX channel 0 weight arbitration optimization enable register	0x0358	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH1_REG	RX channel 1 CRC initial value configuration register	0x035C	R/W
AHB_DMA_RX_CRC_WIDTH_CH1_REG	RX channel 1 CRC result width configuration register	0x0360	R/W
AHB_DMA_IN_CRC_CLEAR_CH1_REG	RX channel 1 CRC result clear register	0x0364	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX channel 1 CRC result register	0x0368	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH1_REG	RX channel 1 CRC intermediate result mask register	0x036C	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x0370	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	RX channel 1 CRC data input mask register	0x0374	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x0378	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX channel 1 arbitration weight configuration register	0x037C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX channel 1 weight arbitration optimization enable register	0x0380	R/W

Name	Description	Address	Access
AHB_DMA_IN_CRC_INIT_DATA_CH2_REG	RX channel 2 CRC initial value configuration register	0x0384	R/W
AHB_DMA_RX_CRC_WIDTH_CH2_REG	RX channel 2 CRC result width configuration register	0x0388	R/W
AHB_DMA_IN_CRC_CLEAR_CH2_REG	RX channel 2 CRC result clear register	0x038C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX channel 2 CRC result register	0x0390	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH2_REG	RX channel 2 CRC intermediate result mask register	0x0394	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x0398	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	RX channel 2 CRC data input mask register	0x039C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x03A0	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH2_REG	RX channel 2 arbitration weight configuration register	0x03A4	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH2_REG	RX channel 2 weight arbitration optimization enable register	0x03A8	R/W
AHB_DMA_IN_LINK_ADDR_CH0_REG	Linked list descriptor configuration register of RX channel 0	0x03AC	R/W
AHB_DMA_IN_LINK_ADDR_CH1_REG	Linked list descriptor configuration register of RX channel 1	0x03B0	R/W
AHB_DMA_IN_LINK_ADDR_CH2_REG	Linked list descriptor configuration register of RX channel 2	0x03B4	R/W
AHB_DMA_OUT_LINK_ADDR_CH0_REG	Linked list descriptor configuration register of TX channel 0	0x03B8	R/W
AHB_DMA_OUT_LINK_ADDR_CH1_REG	Linked list descriptor configuration register of TX channel 1	0x03BC	R/W
AHB_DMA_OUT_LINK_ADDR_CH2_REG	Linked list descriptor configuration register of TX channel 2	0x03C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	Accessible address space start address configuration register	0x03C4	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	Accessible address space end address configuration register	0x03C8	R/W
AHB_DMA_ARB_TIMEOUT_TX_REG	TX Arbitration timeout configuration register	0x03CC	R/W
AHB_DMA_ARB_TIMEOUT_RX_REG	RX Arbitration timeout configuration register	0x03D0	R/W
AHB_DMA_WEIGHT_EN_TX_REG	TX weight arbitration enable register	0x03D4	R/W
AHB_DMA_WEIGHT_EN_RX_REG	RX weight arbitration enable register	0x03D8	R/W
Version Register			
AHB_DMA_DATE_REG	Version control register	0x0068	R/W
Status Registers			
AHB_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status	0x0078	RO
AHB_DMA_IN_STATE_CH0_REG	RX channel 0 status	0x0084	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Receive descriptor address when EOF occurs on RX channel 0	0x0088	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Receive descriptor address when errors occur on RX channel 0	0x008C	RO

Name	Description	Address	Access
AHB_DMA_IN_DSCR_CH0_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0090	RO
AHB_DMA_IN_DSCR_BF0_CH0_REG	Address of the current pre-read receive descriptor on RX channel 0	0x0094	RO
AHB_DMA_IN_DSCR_BF1_CH0_REG	Address of the previous pre-read receive descriptor on RX channel 0	0x0098	RO
AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status	0x00D8	RO
AHB_DMA_OUT_STATE_CH0_REG	TX channel 0 status	0x00E4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	Transmit descriptor address when EOF occurs on TX channel 0	0x00E8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last transmit descriptor address when EOF occurs on TX channel 0	0x00EC	RO
AHB_DMA_OUT_DSCR_CH0_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x00F0	RO
AHB_DMA_OUT_DSCR_BF0_CH0_REG	Address of the current pre-read transmit descriptor on TX channel 0	0x00F4	RO
AHB_DMA_OUT_DSCR_BF1_CH0_REG	Address of the previous pre-read transmit descriptor on TX channel 0	0x00F8	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status	0x0138	RO
AHB_DMA_IN_STATE_CH1_REG	RX channel 1 status	0x0144	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Receive descriptor address when EOF occurs on RX channel 1	0x0148	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Receive descriptor address when errors occur on RX channel 1	0x014C	RO
AHB_DMA_IN_DSCR_CH1_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x0150	RO
AHB_DMA_IN_DSCR_BF0_CH1_REG	Address of the current pre-read receive descriptor on RX channel 1	0x0154	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	Address of the previous pre-read receive descriptor on RX channel 1	0x0158	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 1 FIFO status	0x0198	RO
AHB_DMA_OUT_STATE_CH1_REG	TX channel 1 status	0x01A4	RO

Name	Description	Address	Access
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	Transmit descriptor address when EOF occurs on TX channel 1	0x01A8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last transmit descriptor address when EOF occurs on TX channel 1	0x01AC	RO
AHB_DMA_OUT_DSCR_CH1_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x01B0	RO
AHB_DMA_OUT_DSCR_BF0_CH1_REG	Address of the current pre-read transmit descriptor on TX channel 1	0x01B4	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	Address of the previous pre-read transmit descriptor on TX channel 1	0x01B8	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status	0x01F8	RO
AHB_DMA_IN_STATE_CH2_REG	RX channel 2 status	0x0204	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Receive descriptor address when EOF occurs on RX channel 2	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Receive descriptor address when errors occur on RX channel 2	0x020C	RO
AHB_DMA_IN_DSCR_CH2_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 2	0x0210	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	Address of the current pre-read receive descriptor on RX channel 2	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	Address of the previous pre-read receive descriptor on RX channel 2	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX channel 2 status	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	Transmit descriptor address when EOF occurs on TX channel 2	0x0268	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last transmit descriptor address when EOF occurs on TX channel 2	0x026C	RO
AHB_DMA_OUT_DSCR_CH2_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x0270	RO
AHB_DMA_OUT_DSCR_BF0_CH2_REG	Address of the current pre-read transmit descriptor on TX channel 2	0x0274	RO

Name	Description	Address	Access
AHB_DMA_OUT_DSCR_BF1_CH2_REG	Address of the previous pre-read transmit descriptor on TX channel 2	0x0278	RO
Priority Registers			
AHB_DMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x009C	R/W
AHB_DMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x00FC	R/W
AHB_DMA_IN_PRI_CH1_REG	Priority register of RX channel 1	0x015C	R/W
AHB_DMA_OUT_PRI_CH1_REG	Priority register of TX channel 1	0x01BC	R/W
AHB_DMA_IN_PRI_CH2_REG	Priority register of RX channel 2	0x021C	R/W
AHB_DMA_OUT_PRI_CH2_REG	Priority register of TX channel 2	0x027C	R/W
Peripheral Select Registers			
AHB_DMA_IN_PERI_SEL_CH0_REG	Peripheral selection register of RX channel 0	0x00A0	R/W
AHB_DMA_OUT_PERI_SEL_CH0_REG	Peripheral selection register of TX channel 0	0x0100	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	Peripheral selection register of RX channel 1	0x0160	R/W
AHB_DMA_OUT_PERI_SEL_CH1_REG	Peripheral selection register of TX channel 1	0x01C0	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	Peripheral selection register of RX channel 2	0x0220	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	Peripheral selection register of TX channel 2	0x0280	R/W

3.8.2 GDMA-AXI Register Summary

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

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

Name	Description	Address	Access
Interrupt Registers			
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

GoBack

Name	Description	Address	Access
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	Configuration register 0 of RX channel 0	0x0010	R/W
AXI_DMA_IN_CONF1_CH0_REG	Configuration register 1 of RX channel 0	0x0014	R/W
AXI_DMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x001C	varies
AXI_DMA_IN_LINK1_CH0_REG	Linked list descriptor configuration and control register 1 of RX channel 0	0x0020	varies
AXI_DMA_IN_LINK2_CH0_REG	Linked list descriptor configuration and control register 2 of RX channel 0	0x0024	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH0_REG	RX channel 0 CRC initial value configuration register	0x0048	R/W

Name	Description	Address	Access
AXI_DMA_RX_CRC_WIDTH_CH0_REG	RX channel 0 CRC result width configuration register	0x004C	R/W
AXI_DMA_IN_CRC_CLEAR_CH0_REG	RX channel 0 CRC result clear register	0x0050	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX channel 0 CRC result register	0x0054	RO
AXI_DMA_RX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x005C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	CRC RX channel 0 CRC intermediate result mask register	0x0060	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX channel 0 CRC data input mask target register	0x0064	R/W
AXI_DMA_IN_CONF0_CH1_REG	Configuration register 0 of RX channel 1	0x0078	R/W
AXI_DMA_IN_CONF1_CH1_REG	Configuration register 1 of RX channel 1	0x007C	R/W
AXI_DMA_IN_POP_CH1_REG	Pop control register of RX channel 1	0x0084	varies
AXI_DMA_IN_LINK1_CH1_REG	Linked list descriptor configuration and control register 1 of RX channel 1	0x0088	varies
AXI_DMA_IN_LINK2_CH1_REG	Linked list descriptor configuration and control register 2 of RX channel 1	0x008C	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH1_REG	RX channel 1 CRC initial value configuration register	0x00B0	R/W
AXI_DMA_RX_CRC_WIDTH_CH1_REG	RX channel 1 CRC result width configuration register	0x00B4	R/W
AXI_DMA_IN_CRC_CLEAR_CH1_REG	RX channel 1 CRC result clear register	0x00B8	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX channel 1 CRC result register	0x00BC	RO
AXI_DMA_RX_CRC_EN_ADDR_CH1_REG	RX channel 1 CRC intermediate result mask target register	0x00C4	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	CRC RX channel 1 CRC intermediate result mask register	0x00C8	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX channel 1 CRC data input mask target register	0x00CC	R/W
AXI_DMA_IN_CONF0_CH2_REG	Configuration register 0 of RX channel 2	0x00E0	R/W
AXI_DMA_IN_CONF1_CH2_REG	Configuration register 1 of RX channel 2	0x00E4	R/W
AXI_DMA_IN_POP_CH2_REG	Pop control register of RX channel 2	0x00EC	varies
AXI_DMA_IN_LINK1_CH2_REG	Linked list descriptor configuration and control register 1 of RX channel 2	0x00F0	varies
AXI_DMA_IN_LINK2_CH2_REG	Linked list descriptor configuration and control register 2 of RX channel 2	0x00F4	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH2_REG	RX channel 2 CRC initial value configuration register	0x0118	R/W
AXI_DMA_RX_CRC_WIDTH_CH2_REG	RX channel 2 CRC result width configuration register	0x011C	R/W

Name	Description	Address	Access
AXI_DMA_IN_CRC_CLEAR_CH2_REG	RX channel 2 CRC result clear register	0x0120	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX channel 2 CRC result register	0x0124	RO
AXI_DMA_RX_CRC_EN_ADDR_CH2_REG	RX channel 2 CRC intermediate result mask target register	0x012C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	CRC RX channel 2 CRC intermediate result mask register	0x0130	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX channel 2 CRC data input mask target register	0x0134	R/W
AXI_DMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x0148	R/W
AXI_DMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x014C	R/W
AXI_DMA_OUT_PUSH_CH0_REG	Push control register of TX channel 0	0x0154	varies
AXI_DMA_OUT_LINK1_CH0_REG	Linked list descriptor configuration and control register 1 of TX channel 0	0x0158	varies
AXI_DMA_OUT_LINK2_CH0_REG	Linked list descriptor configuration and control register 2 of TX channel 0	0x015C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX channel 0 CRC initial value configuration register	0x0180	R/W
AXI_DMA_TX_CRC_WIDTH_CH0_REG	TX channel 0 CRC result width configuration register	0x0184	R/W
AXI_DMA_OUT_CRC_CLEAR_CH0_REG	TX channel 0 CRC result clear register	0x0188	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX channel 0 CRC result register	0x018C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX channel 0 CRC intermediate result mask register	0x0190	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH0_REG	TX channel 0 CRC intermediate result mask target register	0x0194	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX channel 0 CRC data input mask register	0x0198	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX channel 0 CRC data input mask target register	0x019C	R/W
AXI_DMA_OUT_CONF0_CH1_REG	Configuration register 0 of TX channel 1	0x01B0	R/W
AXI_DMA_OUT_CONF1_CH1_REG	Configuration register 1 of TX channel 1	0x01B4	R/W
AXI_DMA_OUT_PUSH_CH1_REG	Push control register of TX channel 1	0x01BC	varies
AXI_DMA_OUT_LINK1_CH1_REG	Linked list descriptor configuration and control register 1 of TX channel 1	0x01C0	varies
AXI_DMA_OUT_LINK2_CH1_REG	Linked list descriptor configuration and control register 2 of TX channel 1	0x01C4	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX channel 1 CRC initial value configuration register	0x01E8	R/W
AXI_DMA_TX_CRC_WIDTH_CH1_REG	TX channel 1 CRC result width configuration register	0x01EC	R/W

Name	Description	Address	Access
AXI_DMA_OUT_CRC_CLEAR_CH1_REG	TX channel 1 CRC result clear register	0x01F0	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX channel 1 CRC result register	0x01F4	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH1_REG	CRC TX channel 1 CRC intermediate result mask register	0x01F8	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH1_REG	TX channel 1 CRC intermediate result mask target register	0x01FC	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX channel 1 CRC data input mask target register	0x0200	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX channel 1 CRC data input mask target register	0x0204	R/W
AXI_DMA_OUT_CONF0_CH2_REG	Configuration register 0 of TX channel 2	0x0218	R/W
AXI_DMA_OUT_CONF1_CH2_REG	Configuration register 1 of TX channel 2	0x021C	R/W
AXI_DMA_OUT_PUSH_CH2_REG	Push control register of TX channel 2	0x0224	varies
AXI_DMA_OUT_LINK1_CH2_REG	Linked list descriptor configuration and control register of TX channel 2	0x0228	varies
AXI_DMA_OUT_LINK2_CH2_REG	Linked list descriptor configuration and control register 2 1of TX channel 2	0x022C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX channel 2 CRC initial value configuration register	0x0250	R/W
AXI_DMA_TX_CRC_WIDTH_CH2_REG	TX channel 2 CRC result width configuration register	0x0254	R/W
AXI_DMA_OUT_CRC_CLEAR_CH2_REG	TX channel 2 CRC result clear register	0x0258	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX channel 2 CRC result register	0x025C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH2_REG	CRC TX channel 2 CRC intermediate result mask register	0x0260	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH2_REG	TX channel 2 CRC intermediate result mask target register	0x0264	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX channel 2 CRC data input mask target register	0x0268	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX channel 2 CRC data input mask target register	0x026C	R/W
AXI_DMA_ARB_TIMEOUT_REG	Arbitration timeout configuration register	0x0270	R/W
AXI_DMA_WEIGHT_EN_REG	Weight arbitration enable register	0x0274	R/W
AXI_DMA_IN_MEM_CONF_REG	Internal memory power configuration register for RX channel	0x0278	R/W
AXI_DMA_INTR_MEM_START_ADDR_REG	Accessible internal memory start address configuration register	0x027C	R/W
AXI_DMA_INTR_MEM_END_ADDR_REG	Accessible internal memory end address configuration register	0x0280	R/W
AXI_DMA_EXTR_MEM_START_ADDR_REG	Accessible external memory start address configuration register	0x0284	R/W
AXI_DMA_EXTR_MEM_END_ADDR_REG	Accessible external memory end address configuration register	0x0288	R/W
AXI_DMA_IN_RESET_AVAIL_CH0_REG	RX channel 0 reset status register	0x028C	RO

GoBack

Name	Description	Address	Access
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 register	0x02A8	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH0_REG	CRC RX channel 0 CRC intermediate result mask register	0x0058	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH1_REG	CRC RX channel 1 CRC intermediate result mask register	0x00C0	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH2_REG	CRC RX channel 2 CRC intermediate result mask register	0x0128	R/W
Status Registers			
AXI_DMA_INFIFO_STATUS_CH0_REG	RX channel 0 FIFO status register	0x0018	RO
AXI_DMA_IN_STATE_CH0_REG	RX channel 0 status register	0x0028	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Receive descriptor address when EOF occurs on RX channel 0	0x002C	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Receive descriptor address when errors occur on RX channel 0	0x0030	RO
AXI_DMA_IN_DSCR_CH0_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0034	RO
AXI_DMA_IN_DSCR_BF0_CH0_REG	Address of the current pre-read receive descriptor on RX channel 0	0x0038	RO
AXI_DMA_IN_DSCR_BF1_CH0_REG	Address of the previous pre-read receive descriptor on RX channel 0	0x003C	RO
AXI_DMA_INFIFO_STATUS_CH1_REG	RX channel 1 FIFO status register	0x0080	RO
AXI_DMA_IN_STATE_CH1_REG	RX channel 1 status register	0x0090	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Receive descriptor address when EOF occurs on RX channel 1	0x0094	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Receive descriptor address when errors occur on RX channel 1	0x0098	RO
AXI_DMA_IN_DSCR_CH1_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 1	0x009C	RO
AXI_DMA_IN_DSCR_BF0_CH1_REG	Address of the current pre-read receive descriptor on RX channel 1	0x00A0	RO

GoBack

Name	Description	Address	Access
AXI_DMA_IN_DSCR_BF1_CH1_REG	Address of the previous pre-read receive descriptor on RX channel 1	0x00A4	RO
AXI_DMA_INFIFO_STATUS_CH2_REG	RX channel 2 FIFO status register	0x00E8	RO
AXI_DMA_IN_STATE_CH2_REG	RX channel 2 status register	0x00F8	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Receive descriptor address when EOF occurs on RX channel 2	0x00FC	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Receive descriptor address when errors occur on RX channel 2	0x0100	RO
AXI_DMA_IN_DSCR_CH2_REG	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 2	0x0104	RO
AXI_DMA_IN_DSCR_BF0_CH2_REG	Address of the current pre-read receive descriptor on RX channel 2	0x0108	RO
AXI_DMA_IN_DSCR_BF1_CH2_REG	Address of the previous pre-read receive descriptor on RX channel 2	0x010C	RO
AXI_DMA_OUTFIFO_STATUS_CH0_REG	TX channel 0 FIFO status	0x0150	RO
AXI_DMA_OUT_STATE_CH0_REG	TX channel 0 status	0x0160	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH0_REG	Transmit descriptor address when EOF occurs on TX channel 0	0x0164	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last transmit descriptor address when EOF occurs on TX channel 0	0x0168	RO
AXI_DMA_OUT_DSCR_CH0_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x016C	RO
AXI_DMA_OUT_DSCR_BF0_CH0_REG	Address of the current pre-read transmit descriptor on TX channel 0	0x0170	RO
AXI_DMA_OUT_DSCR_BF1_CH0_REG	Address of the previous pre-read transmit descriptor on TX channel 0	0x0174	RO
AXI_DMA_OUTFIFO_STATUS_CH1_REG	TX channel 0 FIFO status	0x01B8	RO
AXI_DMA_OUT_STATE_CH1_REG	TX channel 1 status	0x01C8	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH1_REG	Transmit descriptor address when EOF occurs on TX channel 1	0x01CC	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last transmit descriptor address when EOF occurs on TX channel 1	0x01D0	RO

Name	Description	Address	Access
AXI_DMA_OUT_DSCR_CH1_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 1	0x01D4	RO
AXI_DMA_OUT_DSCR_BF0_CH1_REG	Address of the current pre-read transmit descriptor on TX channel 1	0x01D8	RO
AXI_DMA_OUT_DSCR_BF1_CH1_REG	Address of the previous pre-read transmit descriptor on TX channel 1	0x01DC	RO
AXI_DMA_OUTFIFO_STATUS_CH2_REG	TX channel 2 FIFO status	0x0220	RO
AXI_DMA_OUT_STATE_CH2_REG	TX channel 2 status	0x0230	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH2_REG	Transmit descriptor address when EOF occurs on TX channel 2	0x0234	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last transmit descriptor address when EOF occurs on TX channel 2	0x0238	RO
AXI_DMA_OUT_DSCR_CH2_REG	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 2	0x023C	RO
AXI_DMA_OUT_DSCR_BF0_CH2_REG	Address of the current pre-read transmit descriptor on TX channel 2	0x0240	RO
AXI_DMA_OUT_DSCR_BF1_CH2_REG	Address of the previous pre-read transmit descriptor on TX channel 2	0x0244	RO
Priority Registers			
AXI_DMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x0040	R/W
AXI_DMA_IN_PRI_CH1_REG	Priority register of RX channel 1	0x00A8	R/W
AXI_DMA_IN_PRI_CH2_REG	Priority register of RX channel 2	0x0110	R/W
AXI_DMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x0178	R/W
AXI_DMA_OUT_PRI_CH1_REG	Priority register of TX channel 1	0x01E0	R/W
AXI_DMA_OUT_PRI_CH2_REG	Priority register of TX channel 2	0x0248	R/W
Peripheral Select Registers			
AXI_DMA_IN_PERI_SEL_CH0_REG	Peripheral selection register of RX channel 0	0x0044	R/W
AXI_DMA_IN_PERI_SEL_CH1_REG	Peripheral selection register of RX channel 1	0x00AC	R/W
AXI_DMA_IN_PERI_SEL_CH2_REG	Peripheral selection register of RX channel 2	0x0114	R/W
AXI_DMA_OUT_PERI_SEL_CH0_REG	Peripheral selection register of TX channel 0	0x017C	R/W

GoBack

Name	Description	Address	Access
AXI_DMA_OUT_PERI_SEL_CH1_REG	Peripheral selection register of TX channel 1	0x01E4	R/W
AXI_DMA_OUT_PERI_SEL_CH2_REG	Peripheral selection register of TX channel 2	0x024C	R/W
Version Register			
AXI_DMA_DATE_REG	Version control register	0x02D8	R/W

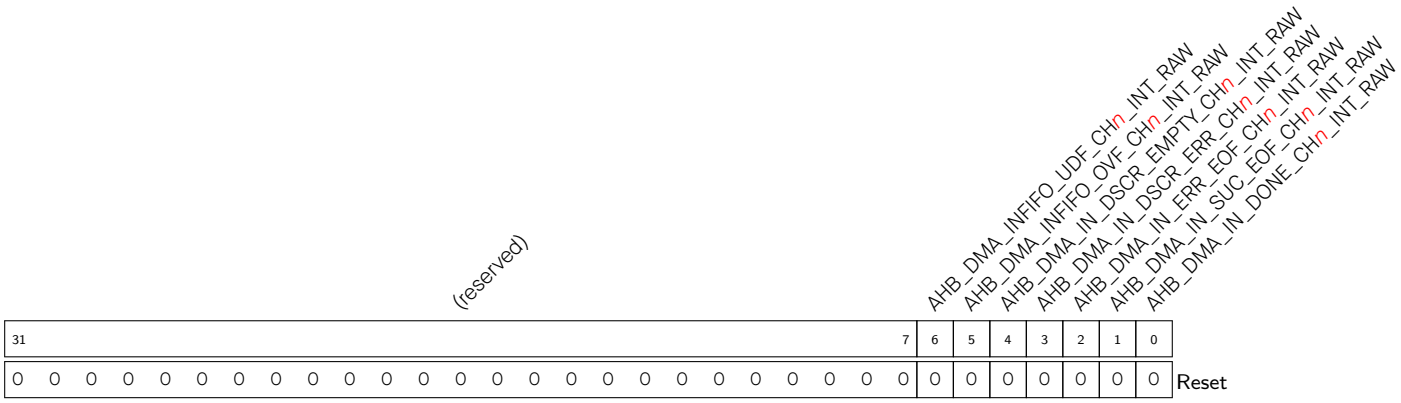
3.9 Registers

3.9.1 GDMA-AHB Registers

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

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

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



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

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

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

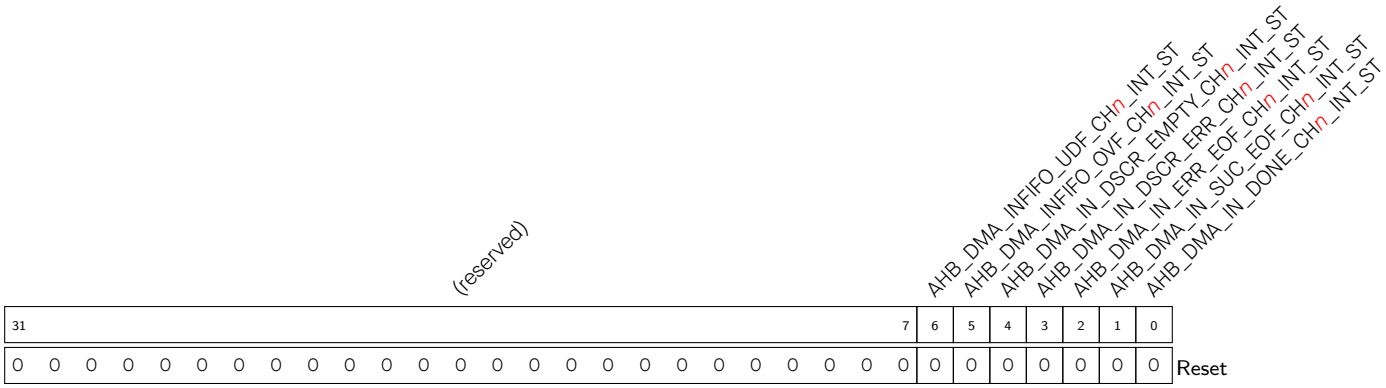
AHB_DMA_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

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

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

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

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



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)

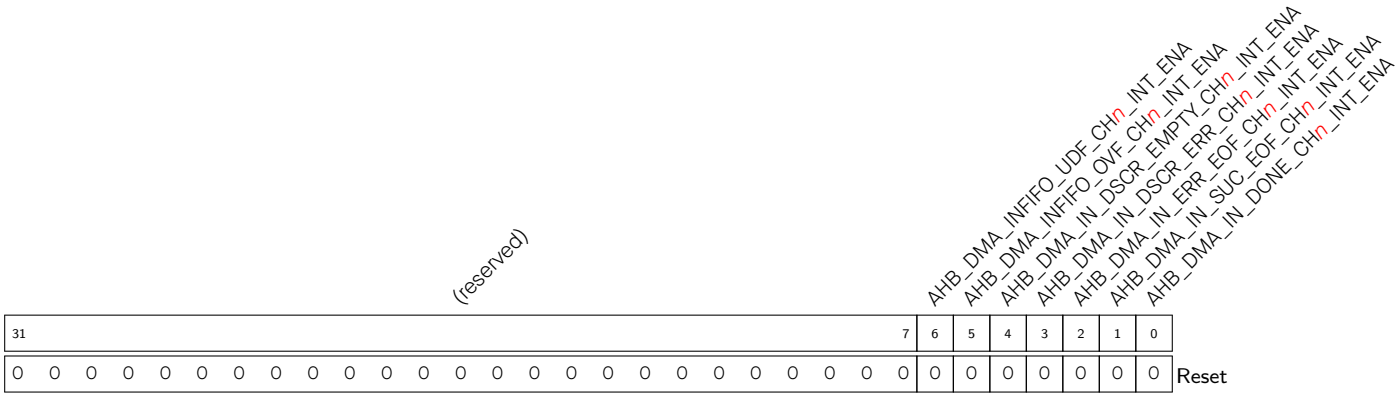
AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (RO)

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)

Register 3.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10n*)**



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

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

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

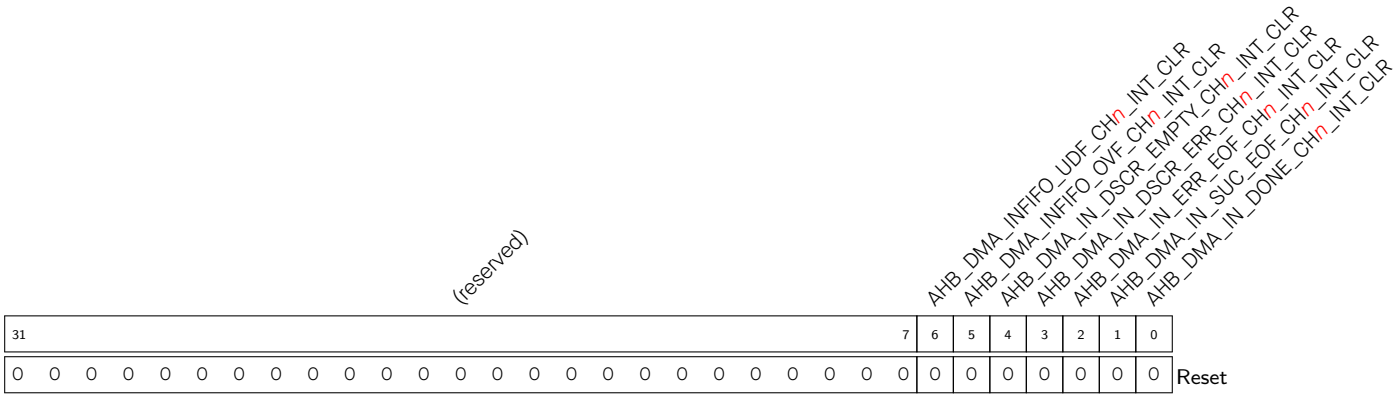
AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (R/W)

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

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

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

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



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

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

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

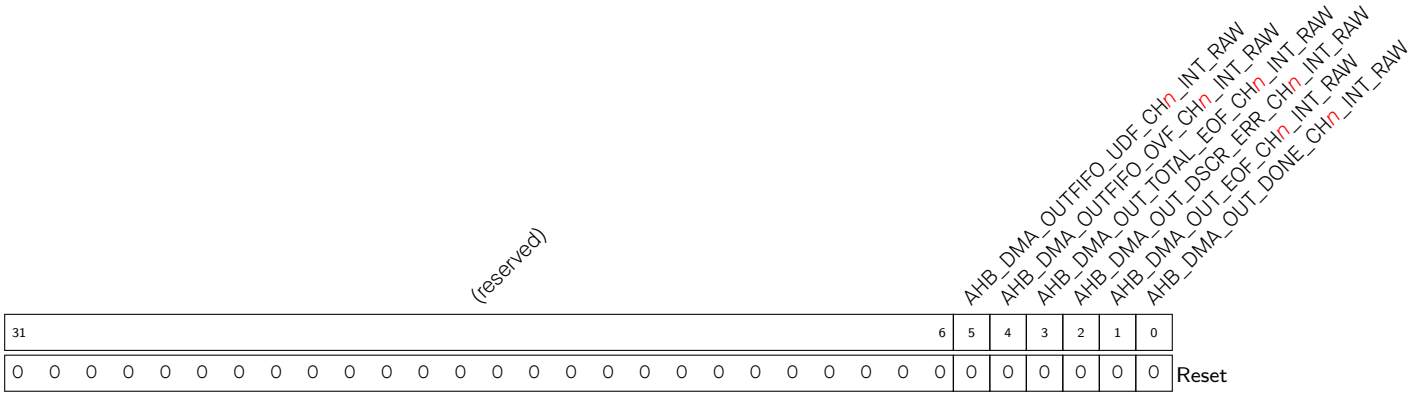
AHB_DMA_IN_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_IN_DSCR_ERR_CH*n*_INT. (WT)

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

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

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

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



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

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

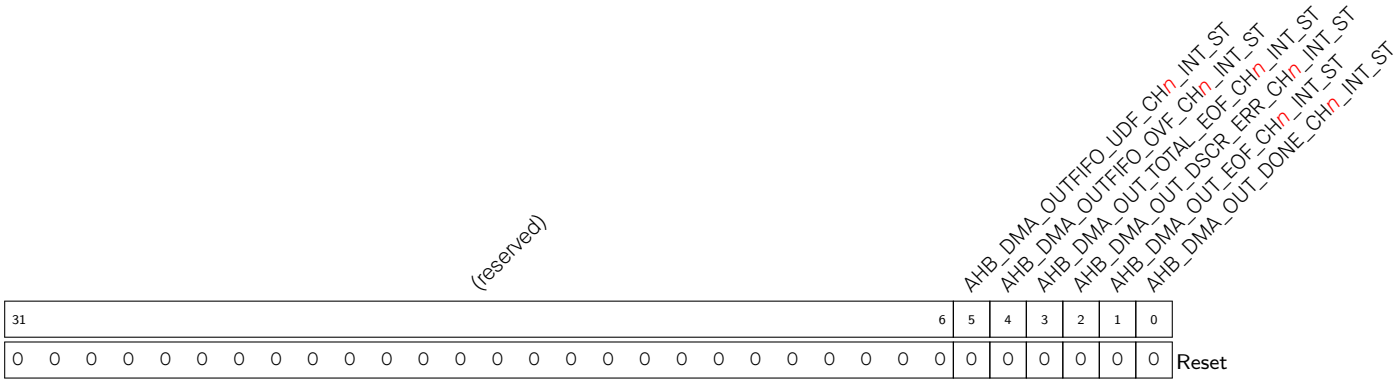
AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/WTC/SS)

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

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

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

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



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

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

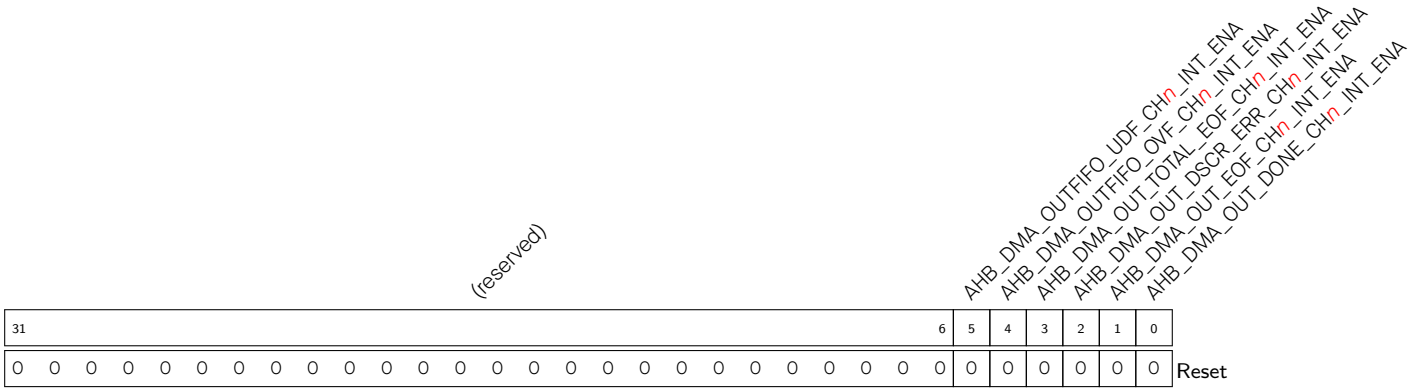
AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ST The masked interrupt status of AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (RO)

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

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

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

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



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

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

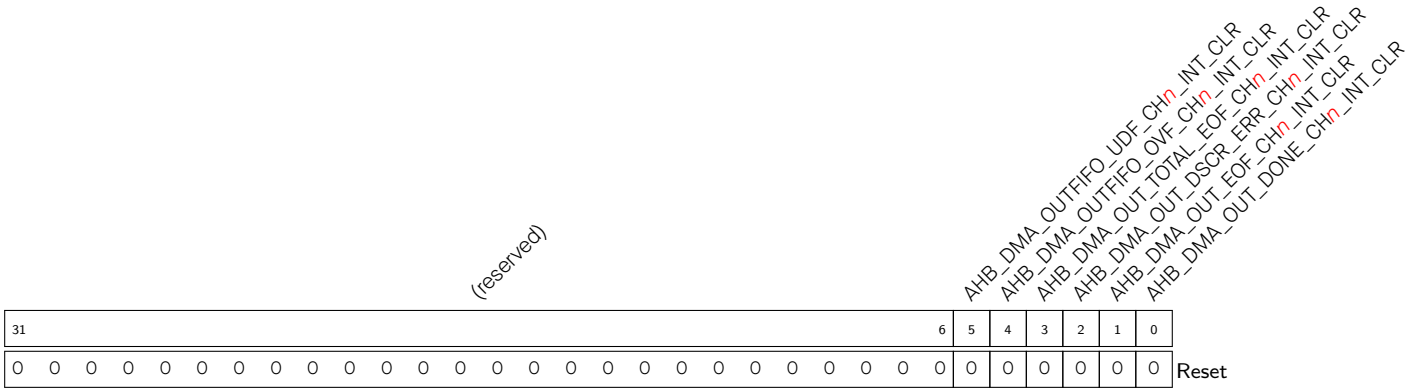
AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA Write 1 to enable AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (R/W)

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

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

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

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



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

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

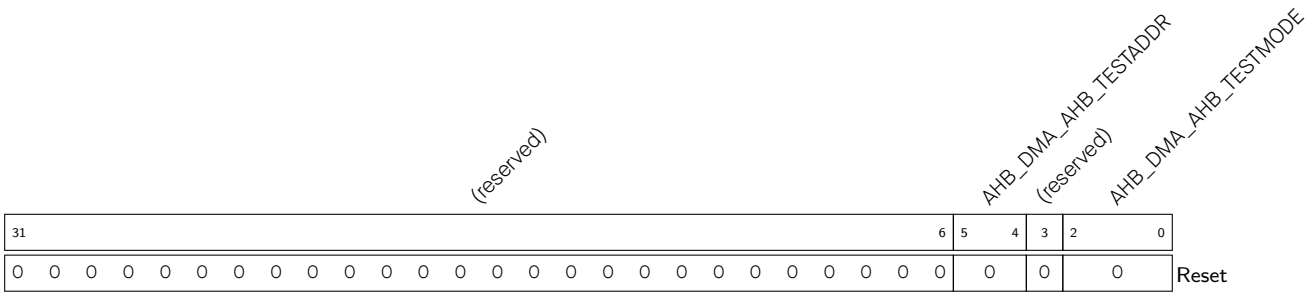
AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR Write 1 to clear AHB_DMA_OUT_DSCR_ERR_CH*n*_INT. (WT)

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

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

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

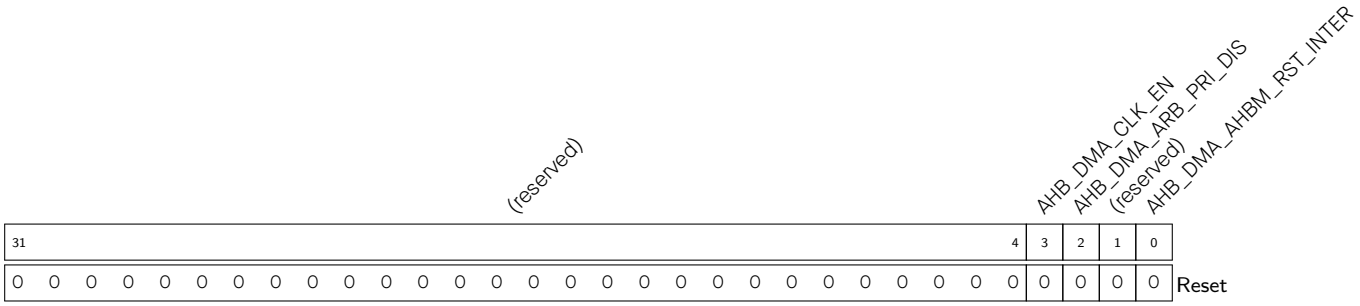
Register 3.9. AHB_DMA_AHB_TEST_REG (0x0060)



AHB_DMA_AHB_TESTMODE Reserved. (R/W)

AHB_DMA_AHB_TESTADDR Reserved. (R/W)

Register 3.10. AHB_DMA_MISC_CONF_REG (0x0064)



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

AHB_DMA_ARB_PRI_DIS Configures whether to disable the priority arbitration.

0: Enable

1: Disable

(R/W)

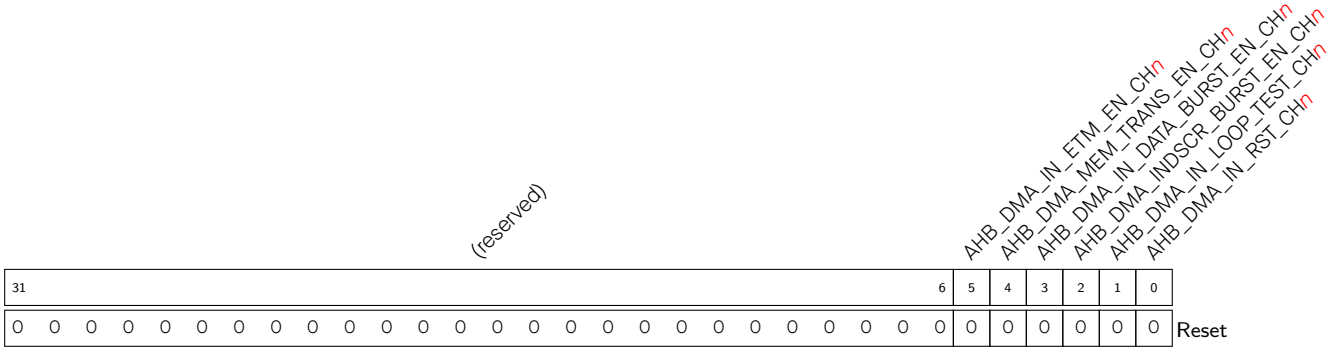
AHB_DMA_CLK_EN Configures AHB DMA clock gating.

0: Support clock only when the application writes registers

1: Always force the clock on for registers

(R/W)

Register 3.11. AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0*n*)



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

AHB_DMA_IN_LOOP_TEST_CH*n* Reserved. (R/W)

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

0: Disable

1: Enable

(R/W)

AHB_DMA_IN_DATA_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n*.

0: Disable

1: Enable

(R/W)

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

0: Disable

1: Enable

(R/W)

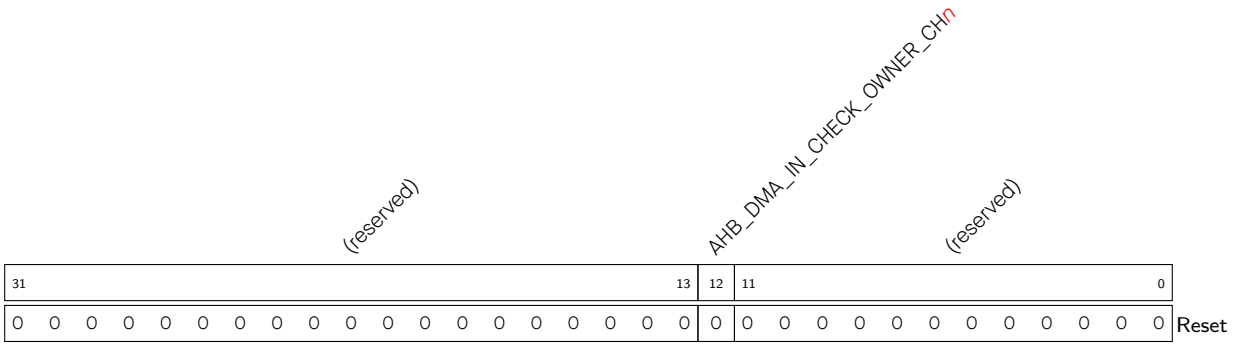
AHB_DMA_IN_ETM_EN_CH*n* Configures whether to enable ETM control for RX channel*n*.

0: Disable

1: Enable

(R/W)

Register 3.12. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0*n*)



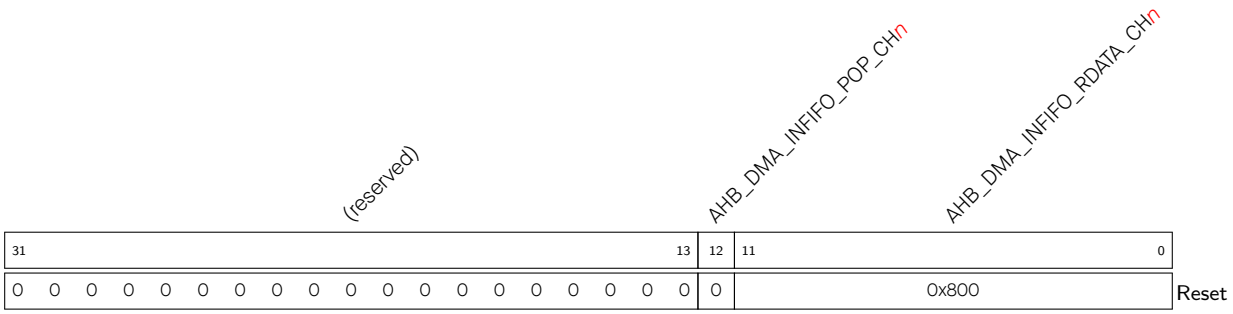
AHB_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

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



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

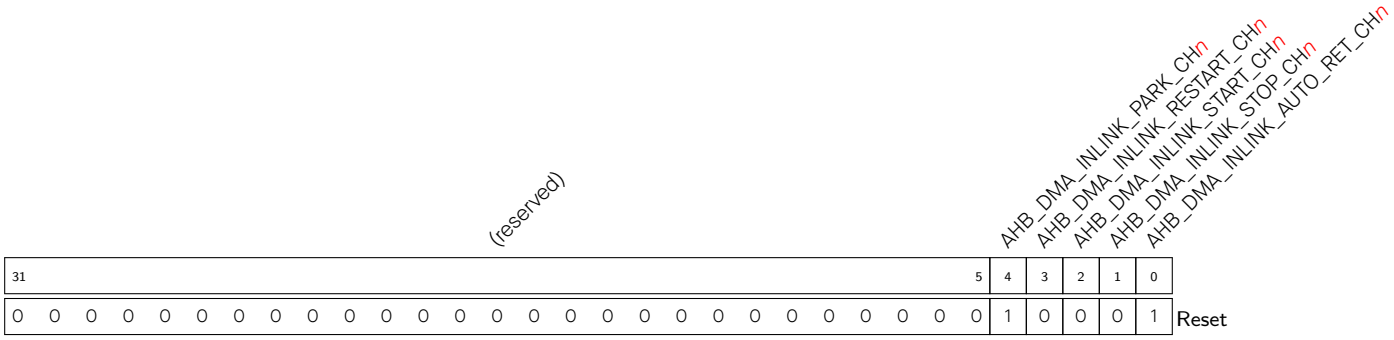
AHB_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AHB DMA RX FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 3.14. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+0xC0n*)**



AHB_DMA_INLINK_AUTO_RET_CH*n* Configures whether to return to the current receive descriptor's address when there are some errors in current receiving data.

0: Not return

1: Return

(R/W)

AHB_DMA_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AHB_DMA_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for AHB DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

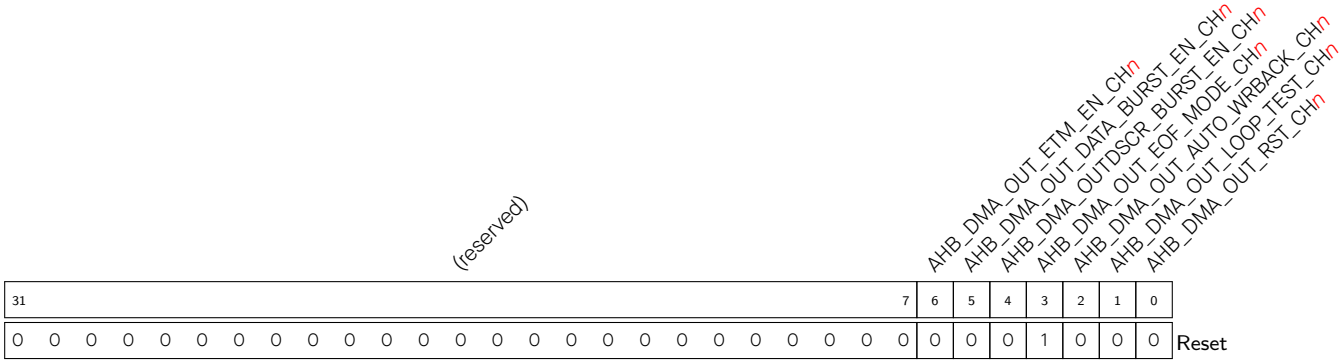
AHB_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

(RO)

Register 3.15. AHB_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0n*)**



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

0: Release reset

1: Reset

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH*n* Reserved. (R/W)

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

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_EOF_MODE_CH*n* Configures when to generate EOF flag.

0: EOF flag for TX channel *n* is generated when data to be transmitted has been pushed into FIFO in AHB DMA.

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

(R/W)

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

0: Disable

1: Enable

(R/W)

AHB_DMA_OUT_DATA_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n*.

0: Disable

1: Enable

(R/W)

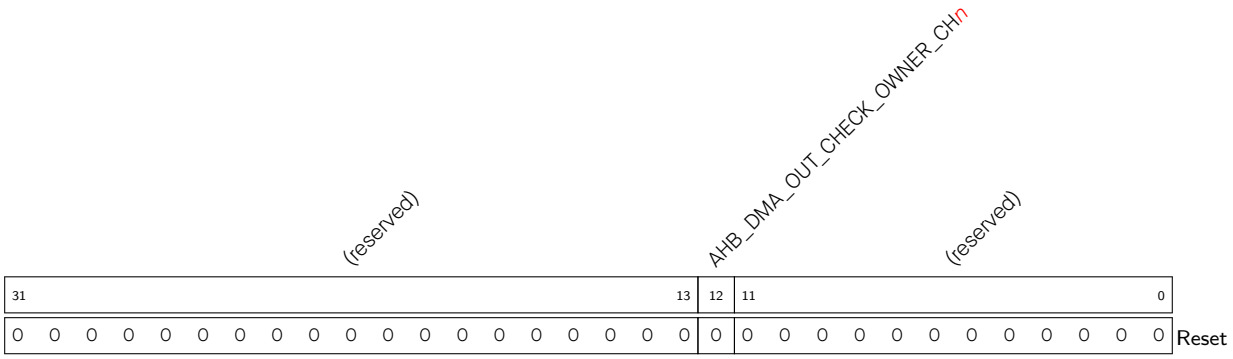
AHB_DMA_OUT_ETM_EN_CH*n* Configures whether to enable ETM control for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.16. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0**n*)



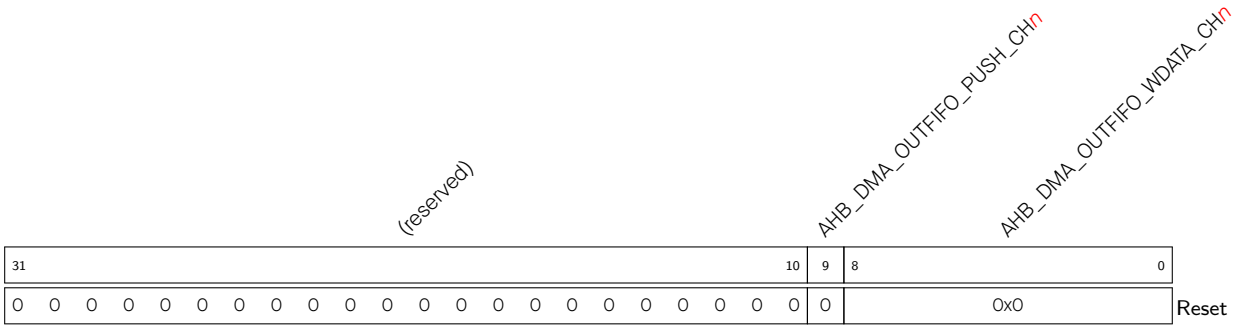
AHB_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

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



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

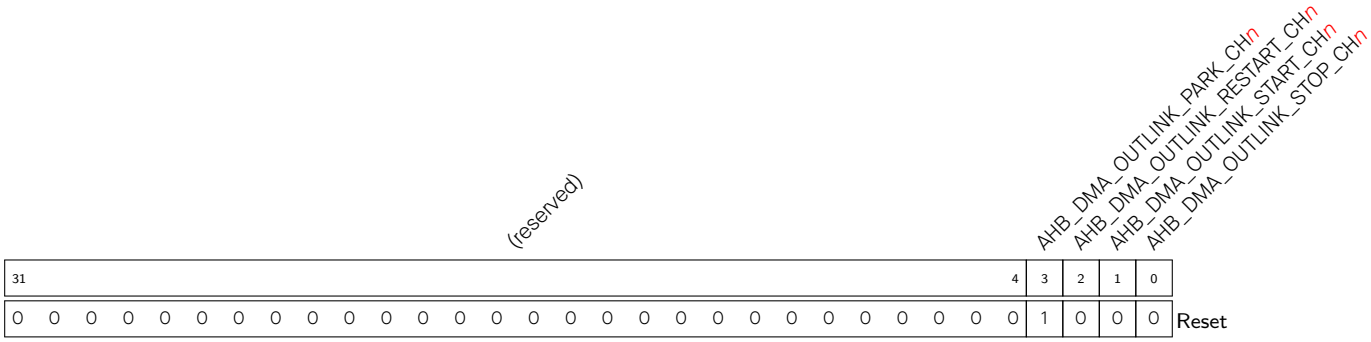
AHB_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AHB DMA TX FIFO.

0: Invalid. No effect

1: Push

(WT)

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



AHB_DMA_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(WT)

AHB_DMA_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AHB_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AHB DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

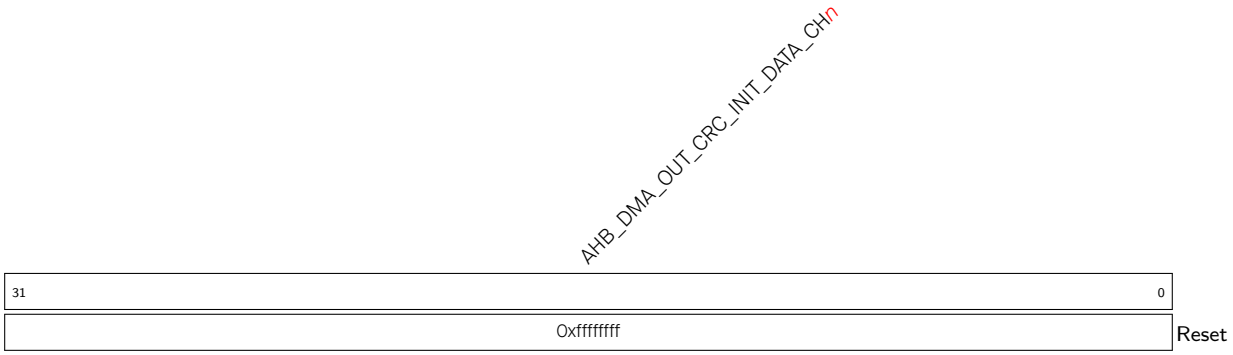
AHB_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

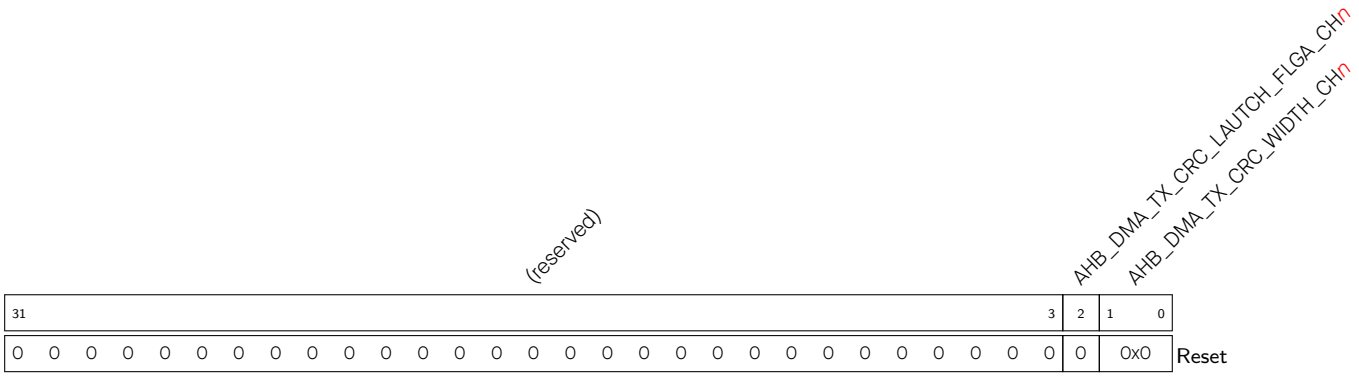
(RO)

Register 3.19. AHB_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x02BC+0x28**n*)



AHB_DMA_OUT_CRC_INIT_DATA_CH*n* Configures the CRC initial value for TX channel *n*. (R/W)

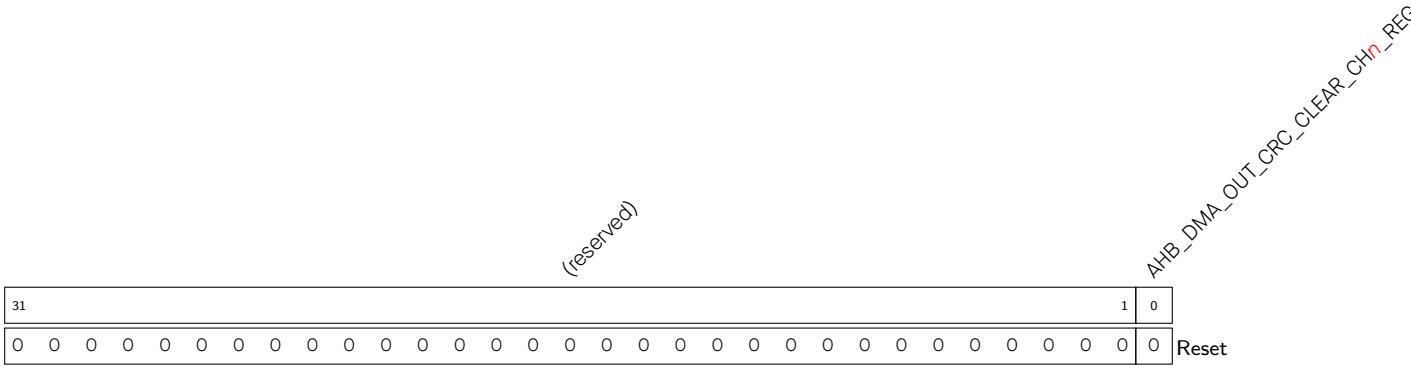
Register 3.20. AHB_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x02C0+0x28**n*)



AHB_DMA_TX_CRC_WIDTH_CH*n* Configures the CRC result width for TX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

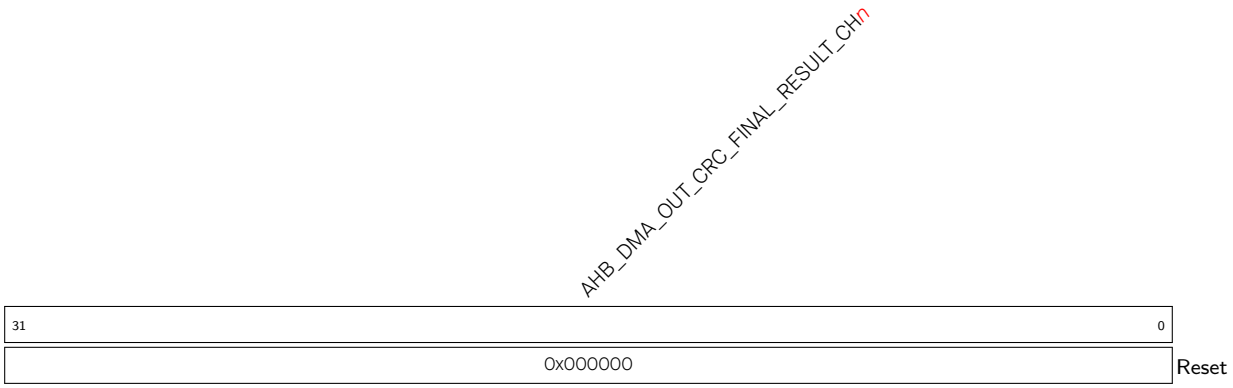
AHB_DMA_TX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AHB_DMA_TX_CRC_EN_ADDR_CH*n*, AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 3.21. AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x02C4+0x28**n*)



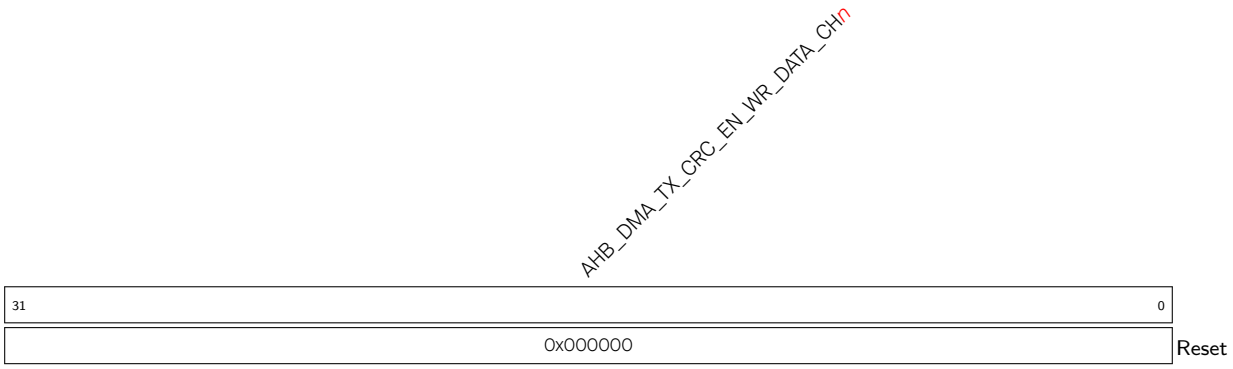
AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG Reserved. (R/W)

Register 3.22. AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x02C8+0x28*n)



AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n* Represents the CRC result for TX channel *n*. (RO)

Register 3.23. AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02CC+0x28*n)



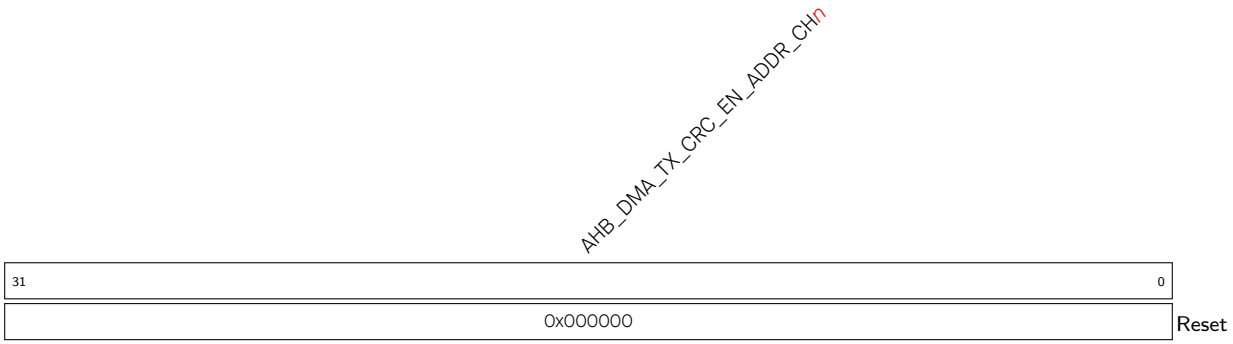
AHB_DMA_TX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

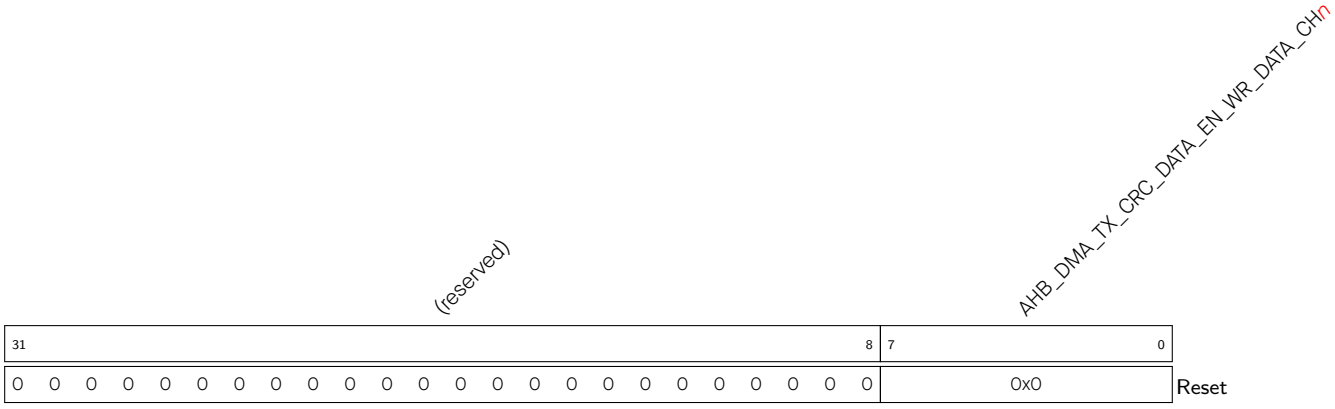
(R/W)

Register 3.24. AHB_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D0+0x28n*)**



AHB_DMA_TX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 3.25. AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02D4+0x28n*)**



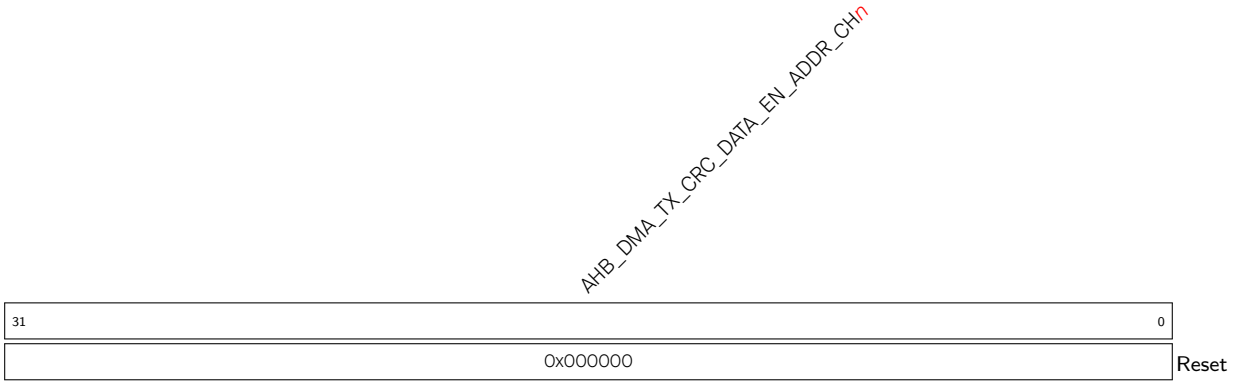
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

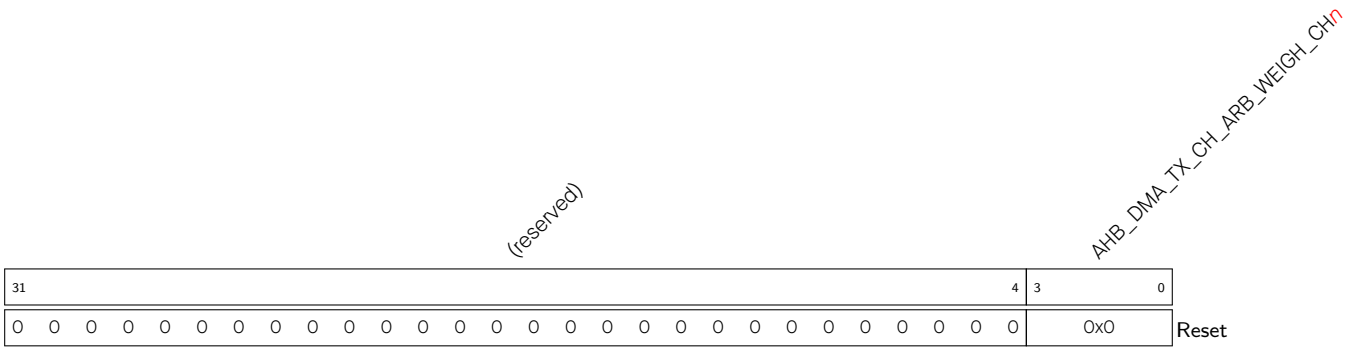
(R/W)

Register 3.26. AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D8+0x28**n*)



AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

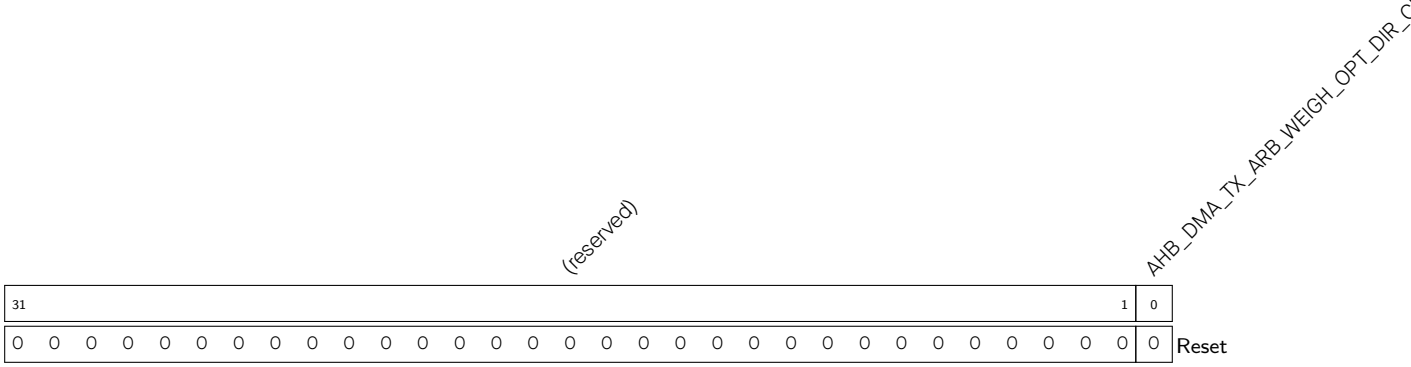
Register 3.27. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x02DC+0x28**n*)



AHB_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

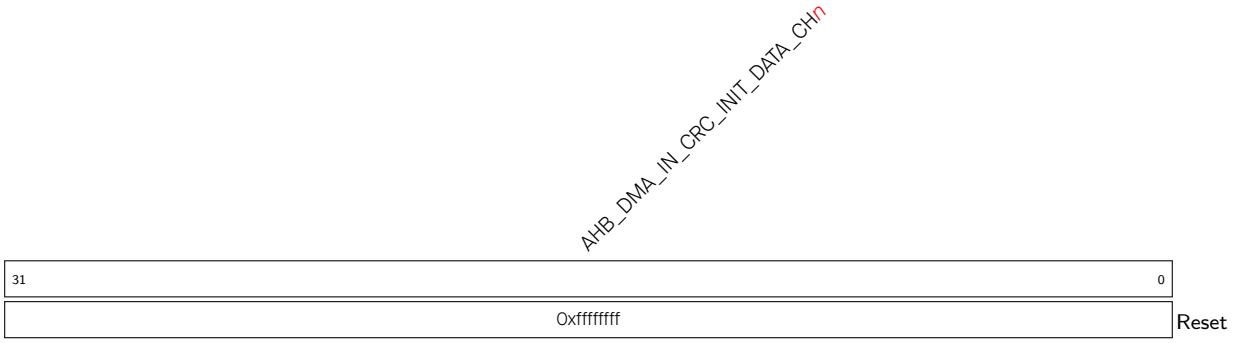
Value range: 0 ~ 15. (R/W)

Register 3.28. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x02E0+0x28**n*)



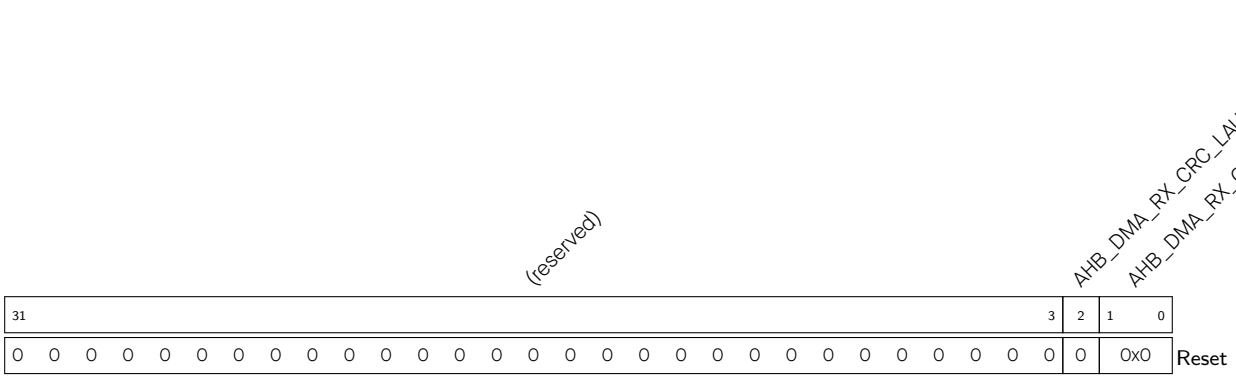
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Reserved. (R/W)

Register 3.29. AHB_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0334+0x28**n*)



AHB_DMA_IN_CRC_INIT_DATA_CH*n* Configures the CRC initial value for RX channel *n*. (R/W)

Register 3.30. AHB_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x0338+0x28**n*)



AHB_DMA_RX_CRC_WIDTH_CH*n* Configures the CRC result width for RX channel *n*.

0: $\leq$ 8 bits

1: $\leq$ 16 bits

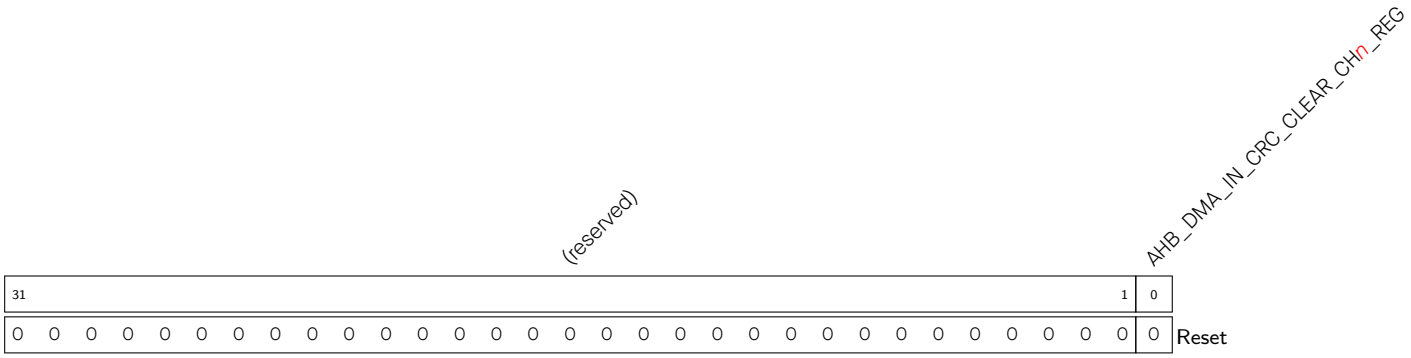
2: $\leq$ 24 bits

3: $\leq$ 32 bits

(R/W)

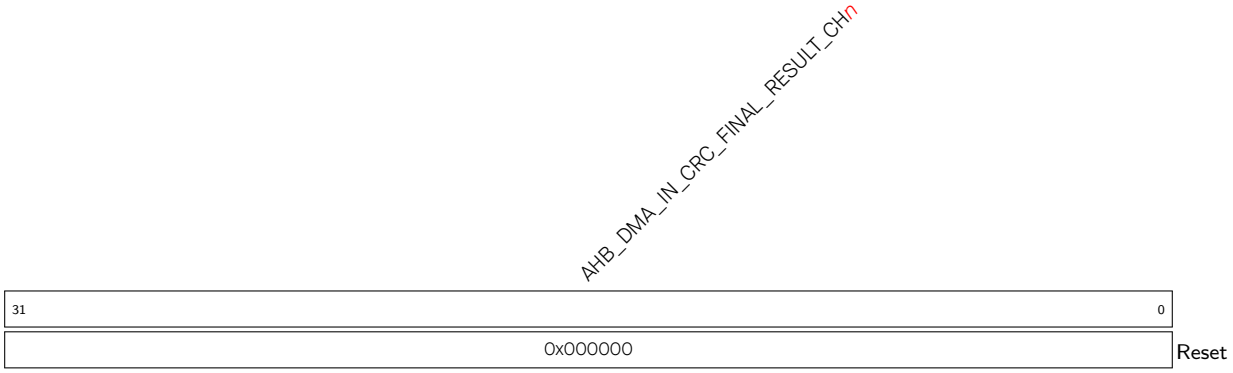
AHB_DMA_RX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AHB_DMA_RX_CRC_EN_ADDR_CH*n*, AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*, AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*, and AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 3.31. AHB_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x033C+0x28**n*)



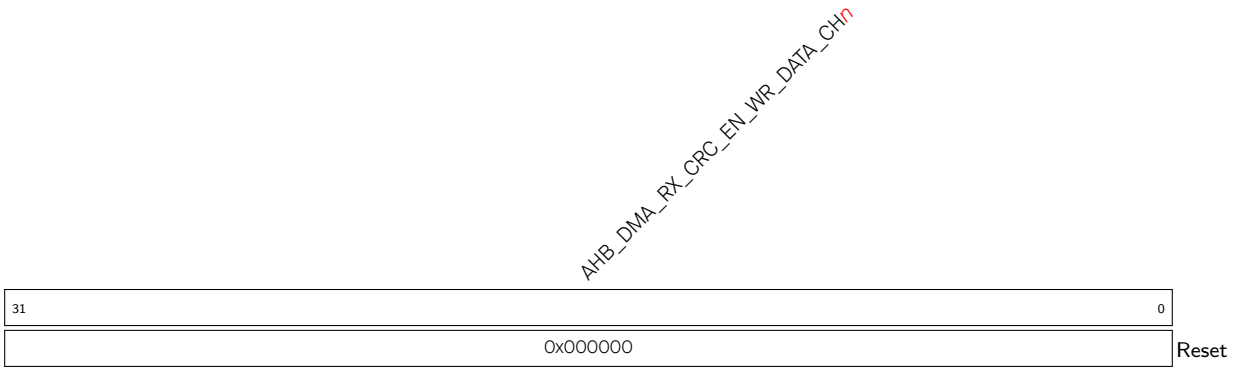
AHB_DMA_IN_CRC_CLEAR_CH*n*_REG Reserved. (R/W)

Register 3.32. AHB_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x0340+0x28**n*)



AHB_DMA_IN_CRC_FINAL_RESULT_CH*n* Represents the CRC result for RX channel *n*. (RO)

Register 3.33. AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0344+0x28**n*)



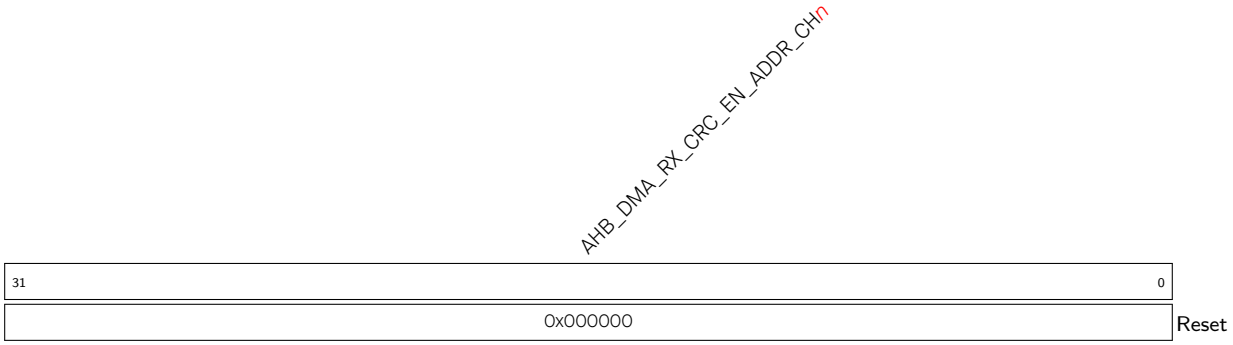
AHB_DMA_RX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

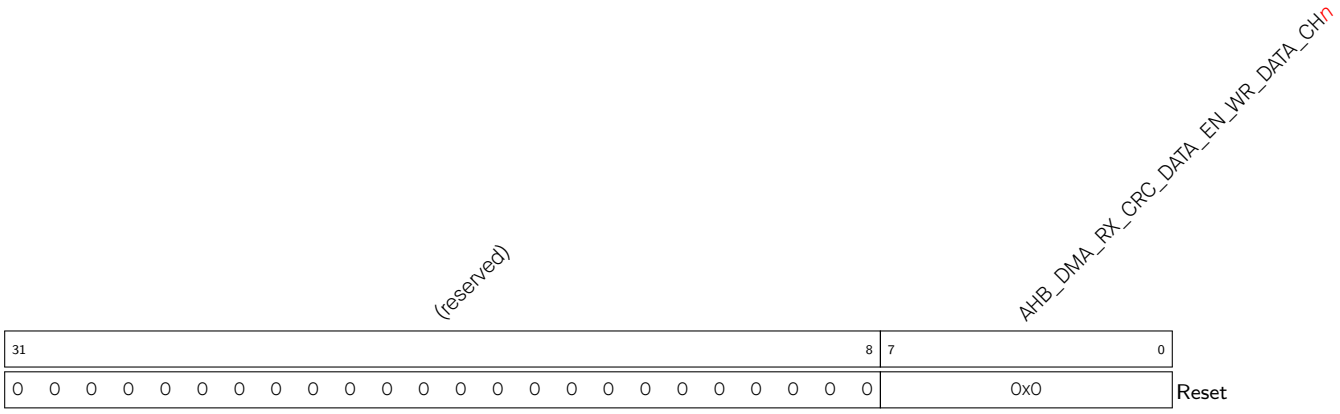
(R/W)

Register 3.34. AHB_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0348+0x28n*)**



AHB_DMA_RX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG register targets for RX channel *n*. (R/W)

Register 3.35. AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x034C+0x28n*)**



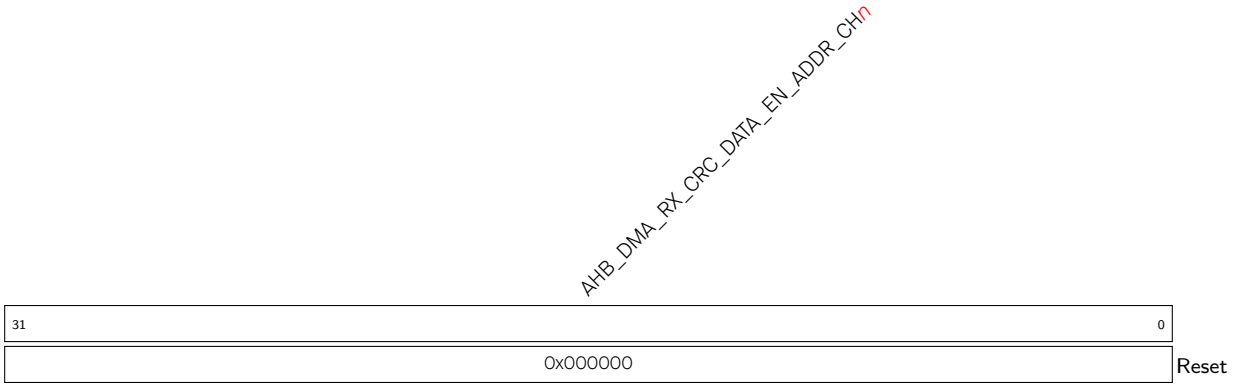
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

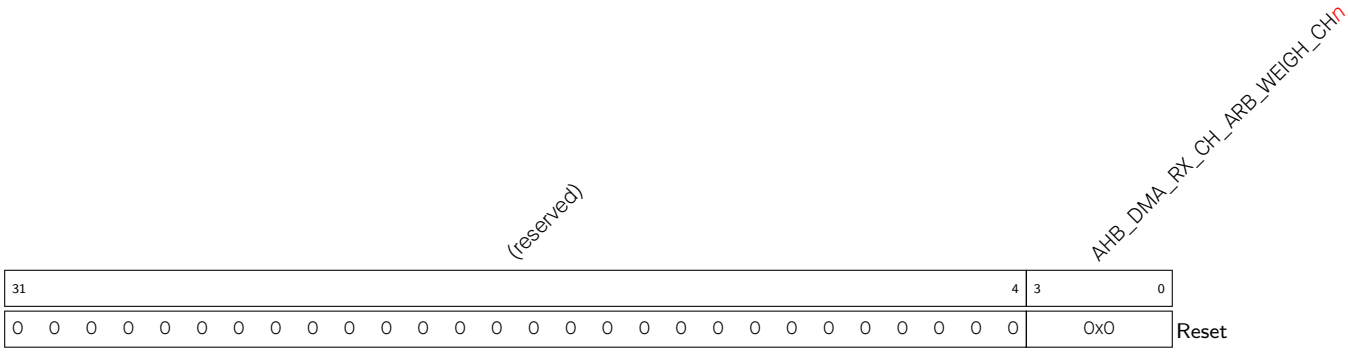
(R/W)

Register 3.36. AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0350+0x28*n)



AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for RX channel *n*. (R/W)

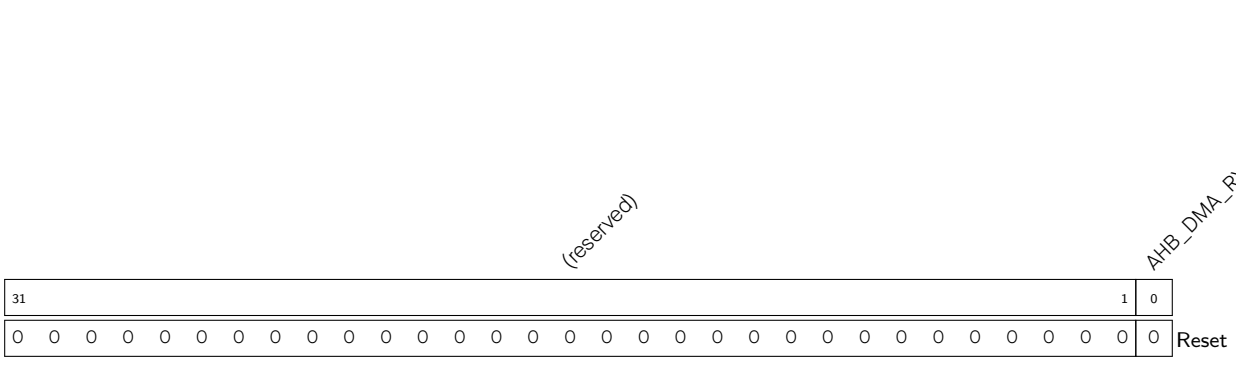
Register 3.37. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x0354+0x28*n)



AHB_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

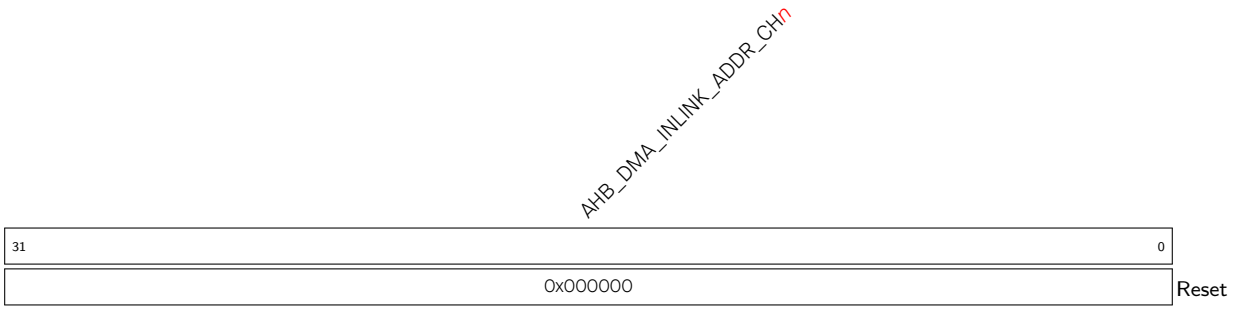
Value range: 0 ~ 15. (R/W)

Register 3.38. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x0358+0x28*n)



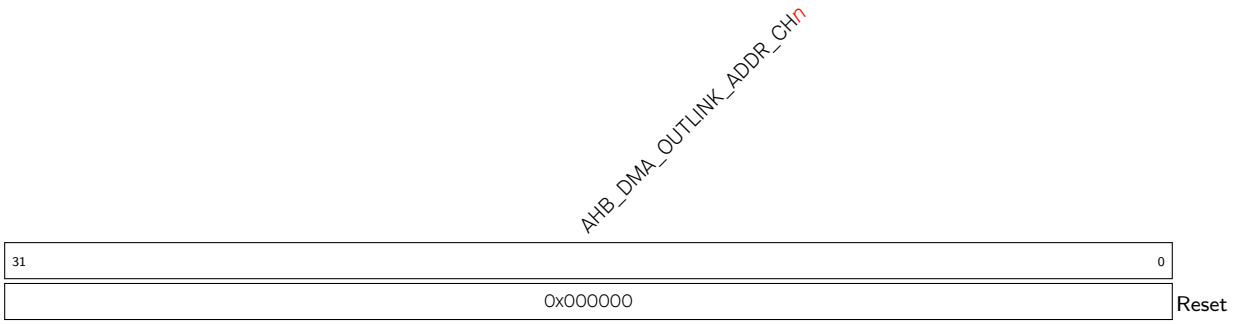
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Reserved. (R/W)

Register 3.39. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03AC+0x4**n*)



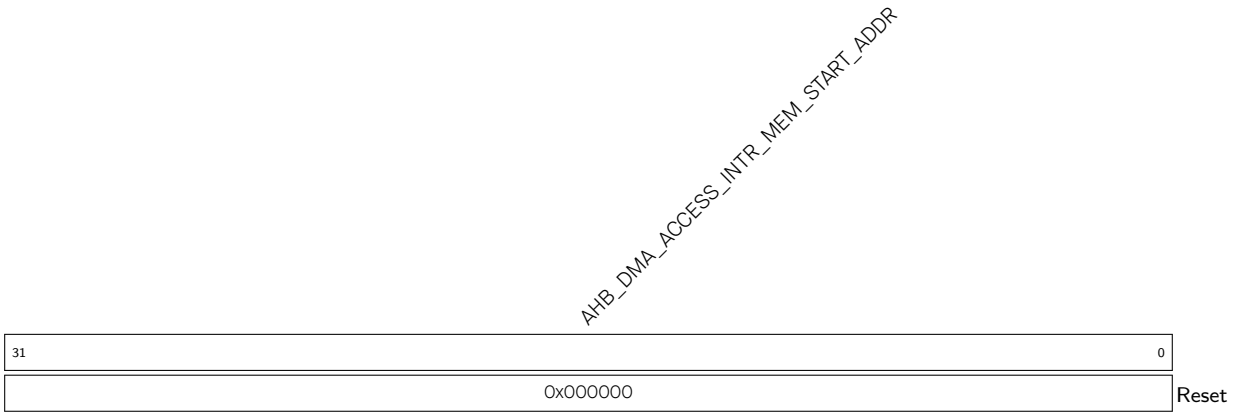
AHB_DMA_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 3.40. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03B8+0x4**n*)



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

Register 3.41. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)



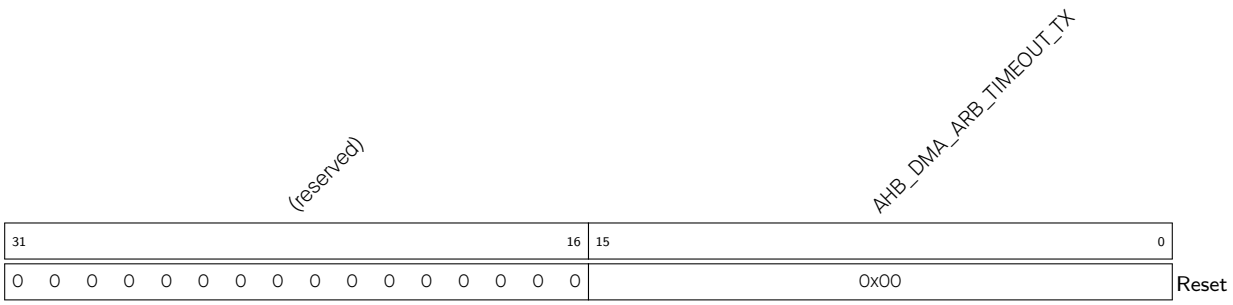
AHB_DMA_ACCESS_INTR_MEM_START_ADDR Configures the start address of accessible address space. (R/W)

Register 3.42. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)



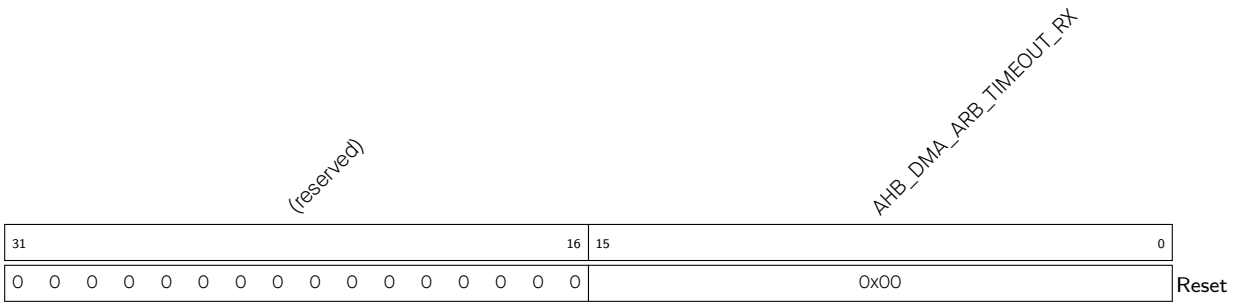
AHB_DMA_ACCESS_INTR_MEM_END_ADDR Configures the end address of accessible address space. (R/W)

Register 3.43. AHB_DMA_ARB_TIMEOUT_TX_REG (0x03CC)



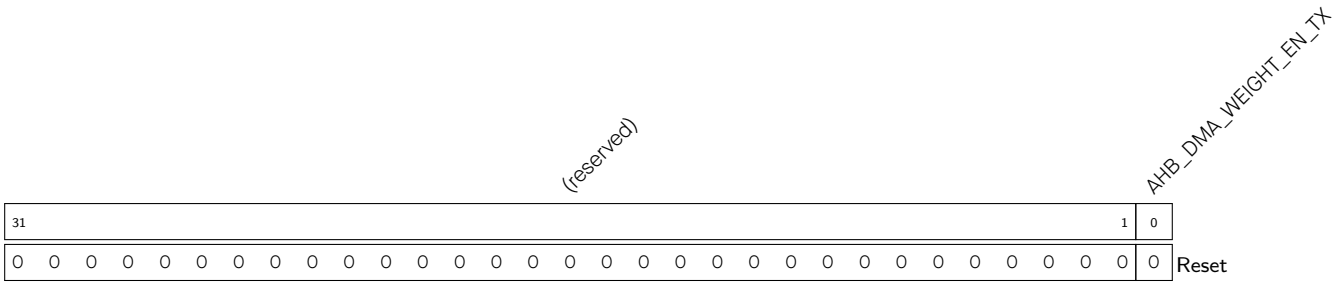
AHB_DMA_ARB_TIMEOUT_TX Configures the time slot for TX. Measurement unit: AHB bus clock cycle. (R/W)

Register 3.44. AHB_DMA_ARB_TIMEOUT_RX_REG (0x03D0)



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

Register 3.45. AHB_DMA_WEIGHT_EN_TX_REG (0x03D4)



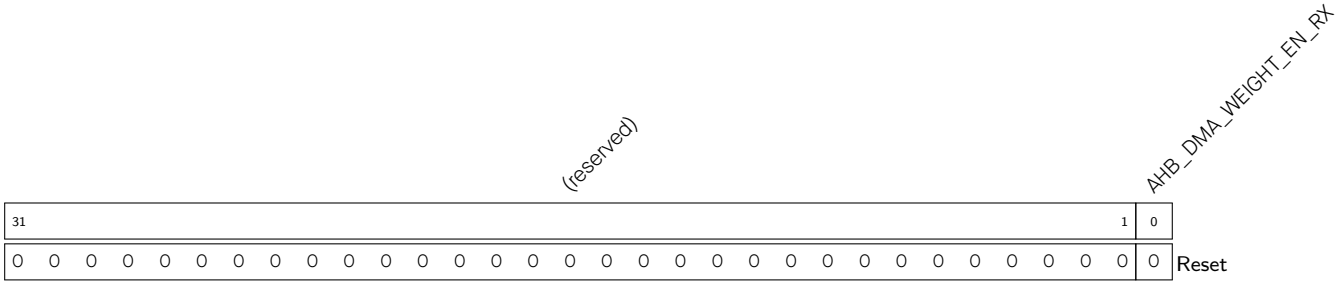
AHB_DMA_WEIGHT_EN_TX Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

Register 3.46. AHB_DMA_WEIGHT_EN_RX_REG (0x03D8)



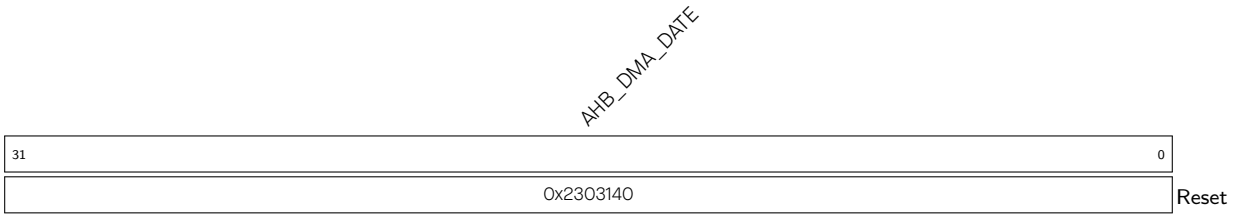
AHB_DMA_WEIGHT_EN_RX Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

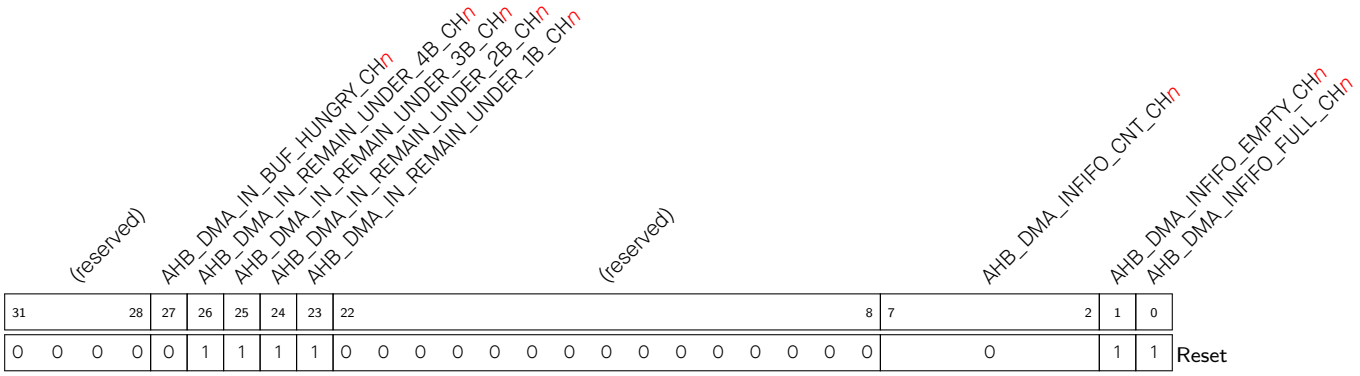
(R/W)

Register 3.47. AHB_DMA_DATE_REG (0x0068)



AHB_DMA_DATE Version control register. (R/W)

Register 3.48. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0n*)**



AHB_DMA_INFIFO_FULL_CH*n* Represents whether L1 RX FIFO is full.

0: Not Full

1: Full

(RO)

AHB_DMA_INFIFO_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AHB_DMA_INFIFO_CNT_CH*n* Represents the number of data bytes in L1 RX FIFO for RX channel *n*.

(RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

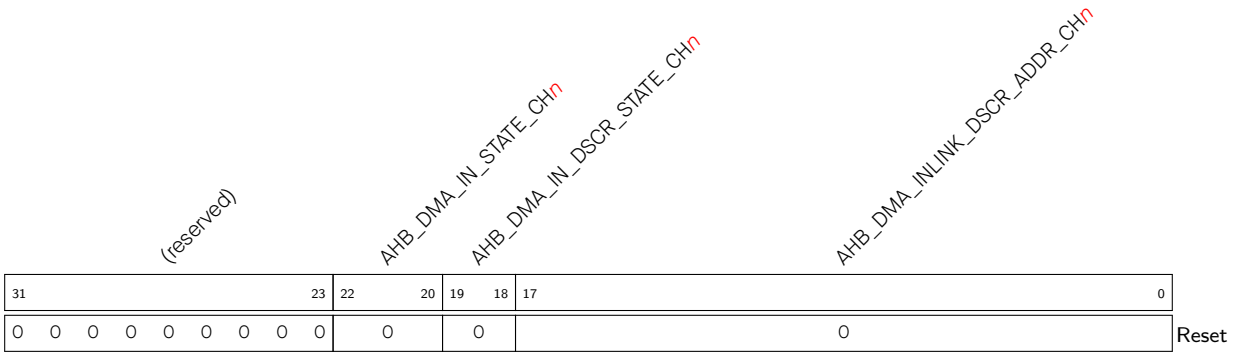
AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

AHB_DMA_IN_BUF_HUNGRY_CH*n* Reserved. (RO)

Register 3.49. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n*)

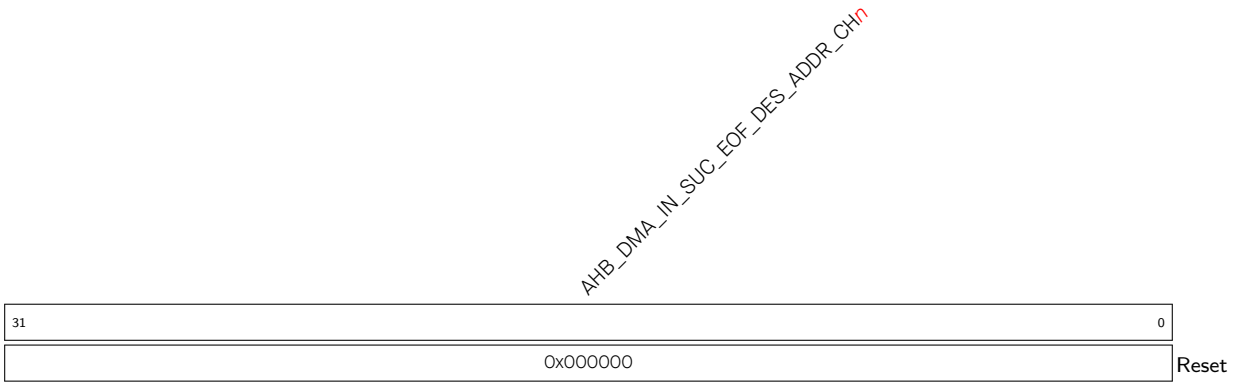


AHB_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

AHB_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

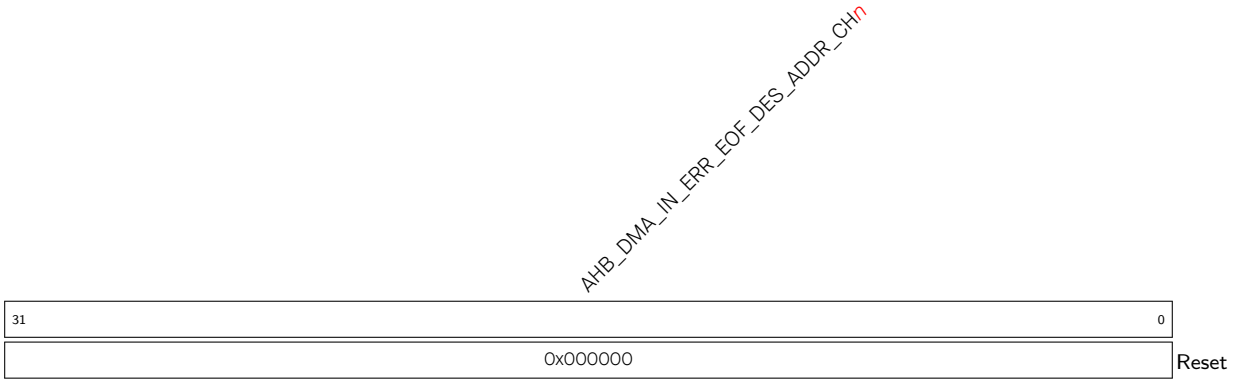
AHB_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.50. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n*)



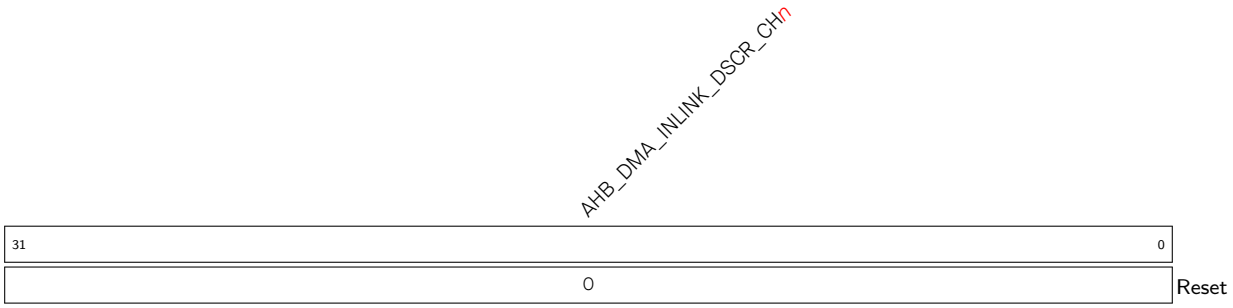
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.51. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n*)



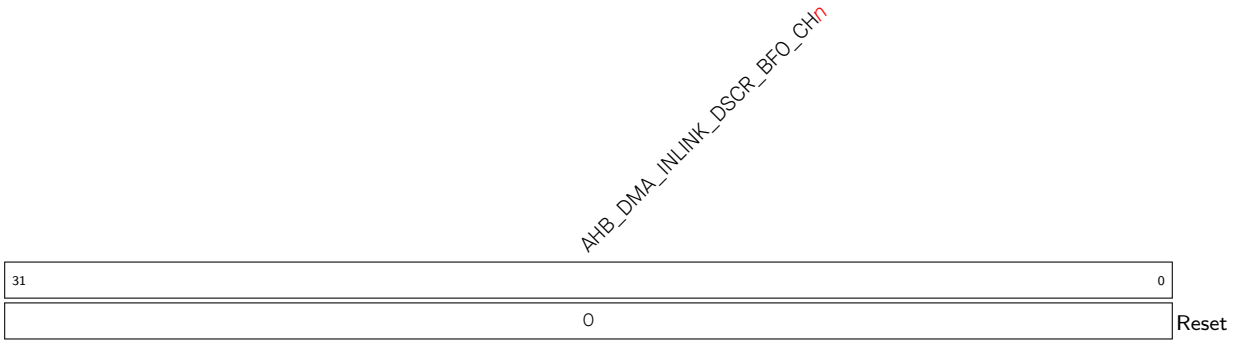
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for UHCI. (RO)

Register 3.52. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n*)



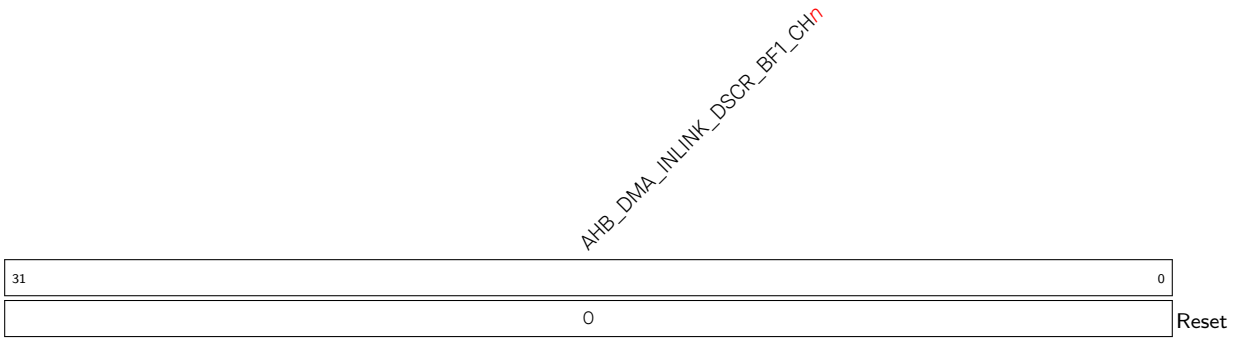
AHB_DMA_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 3.53. AHB_DMA_IN_DSCR_BFO_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n*)



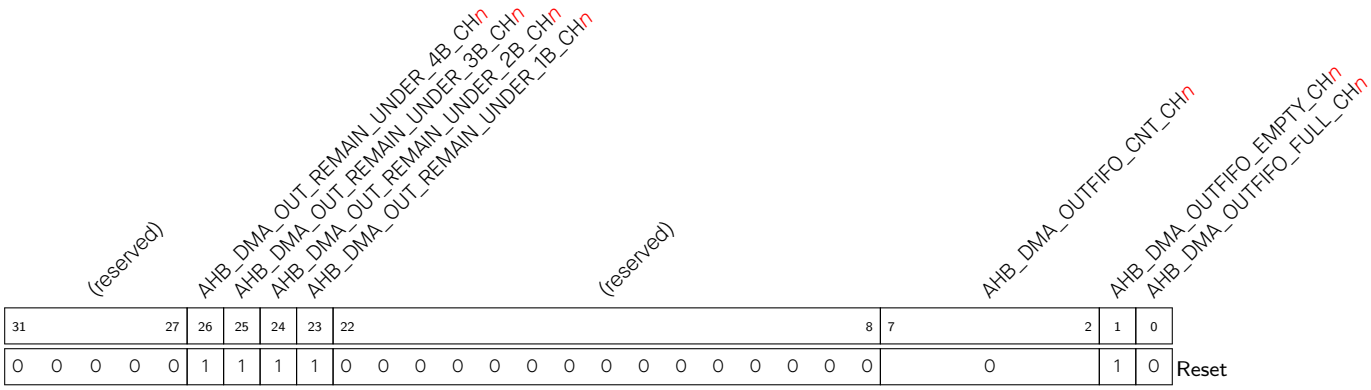
AHB_DMA_INLINK_DSCR_BFO_CH*n* Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 3.54. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n*)



AHB_DMA_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 3.55. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n*)



AHB_DMA_OUTFIFO_FULL_CH*n* Represents whether L1 TX FIFO is full.

0: Not Full

1: Full

(RO)

AHB_DMA_OUTFIFO_EMPTY_CH*n* Represents whether L1 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

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

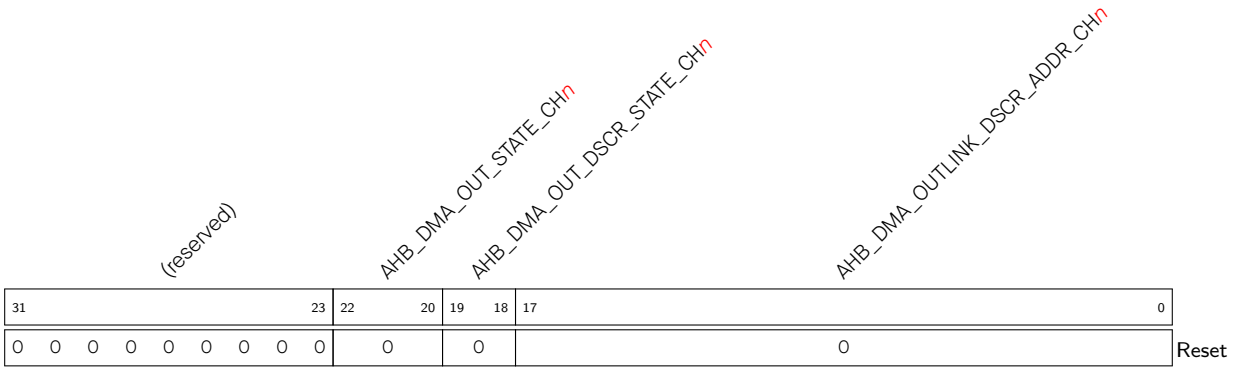
AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

Register 3.56. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0*n*)

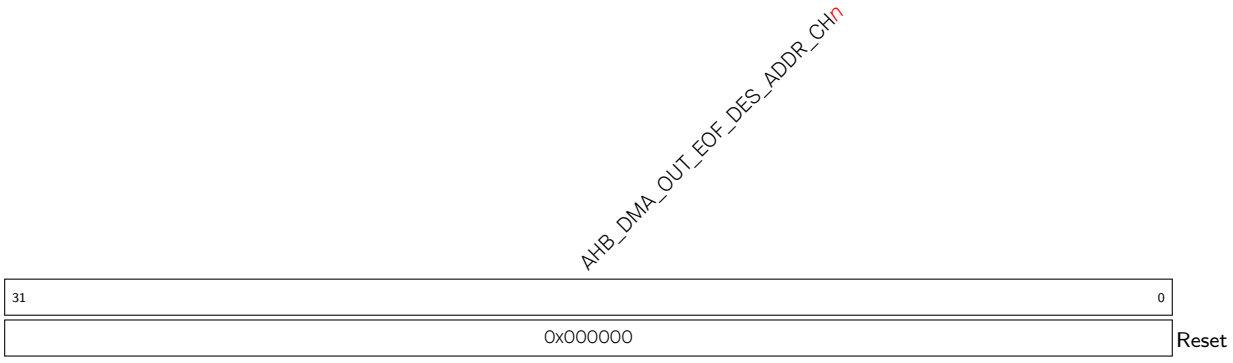


AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

AHB_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

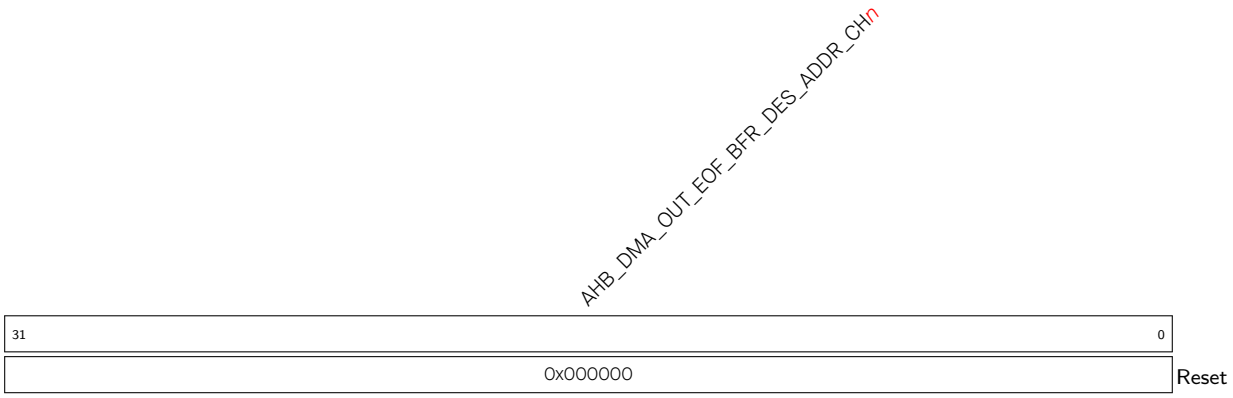
AHB_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.57. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0*n*)



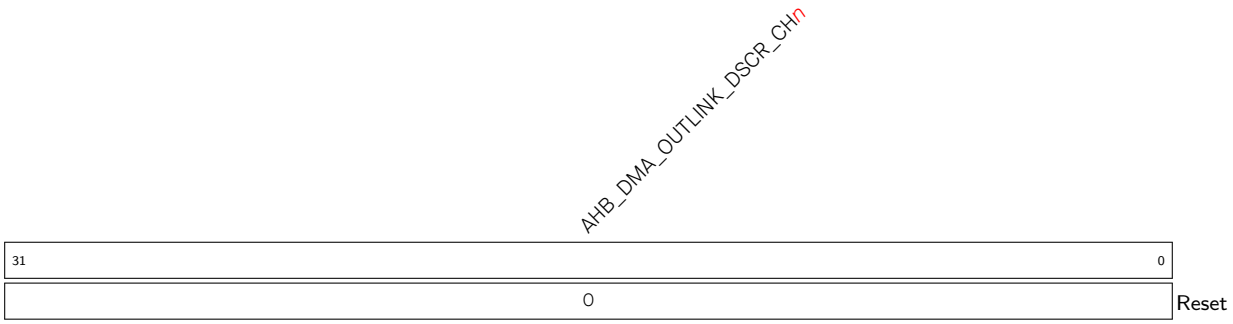
AHB_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.58. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0*n*)



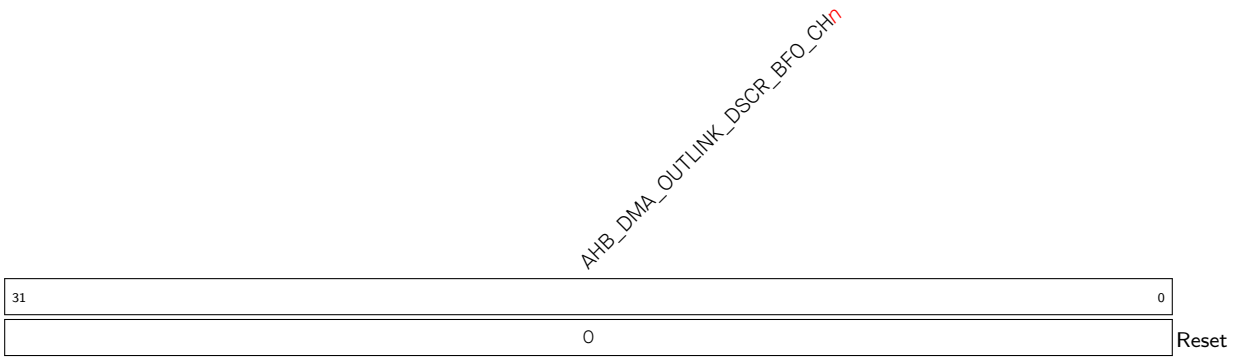
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 3.59. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0*n*)



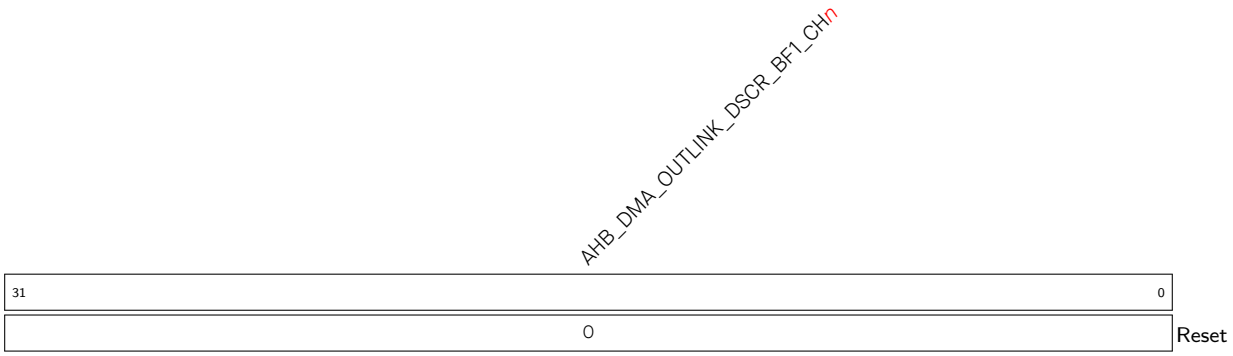
AHB_DMA_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 3.60. AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0*n*)



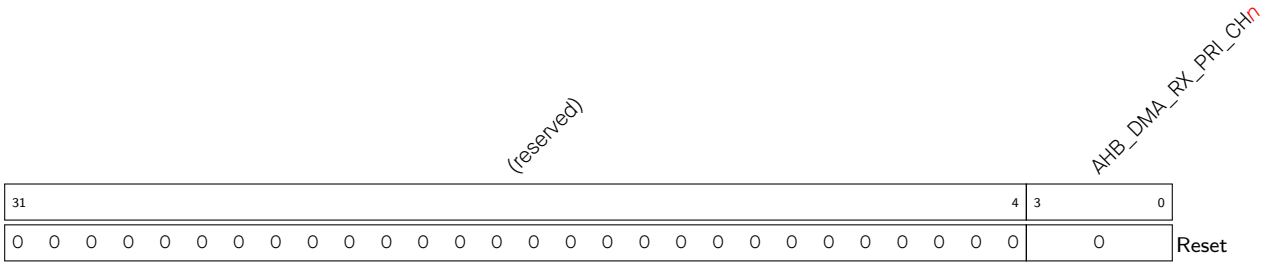
AHB_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 3.61. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



AHB_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

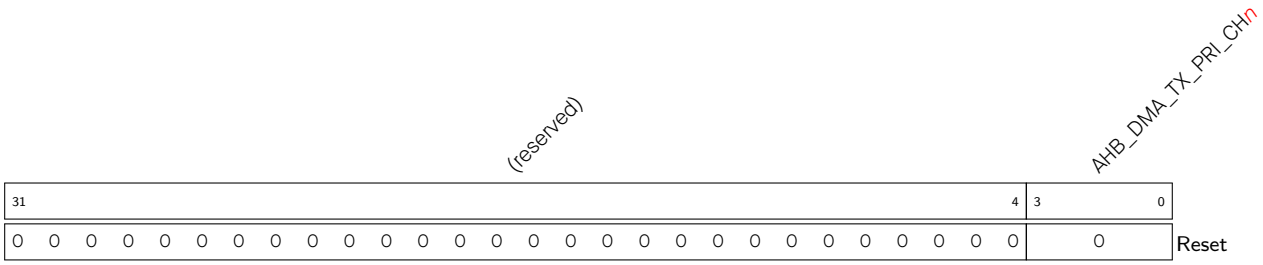
Register 3.62. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)



AHB_DMA_RX_PRI_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5 (R/W)

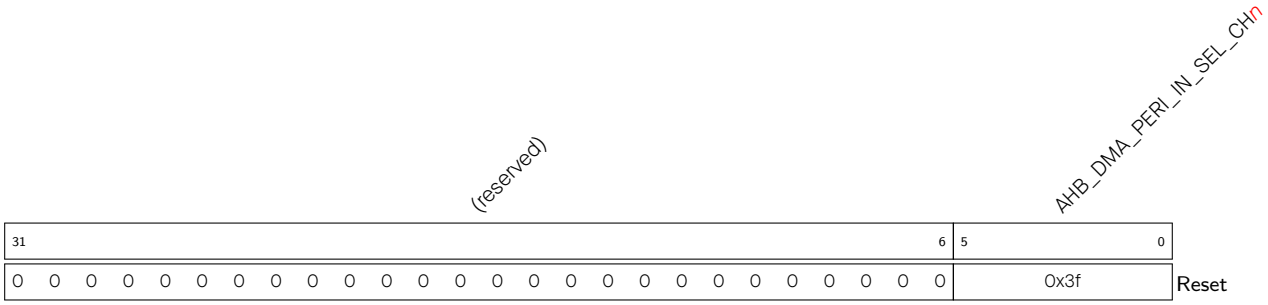
Register 3.63. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)



AHB_DMA_TX_PRI_CH*n* Configures the priority of TX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5 (R/W)

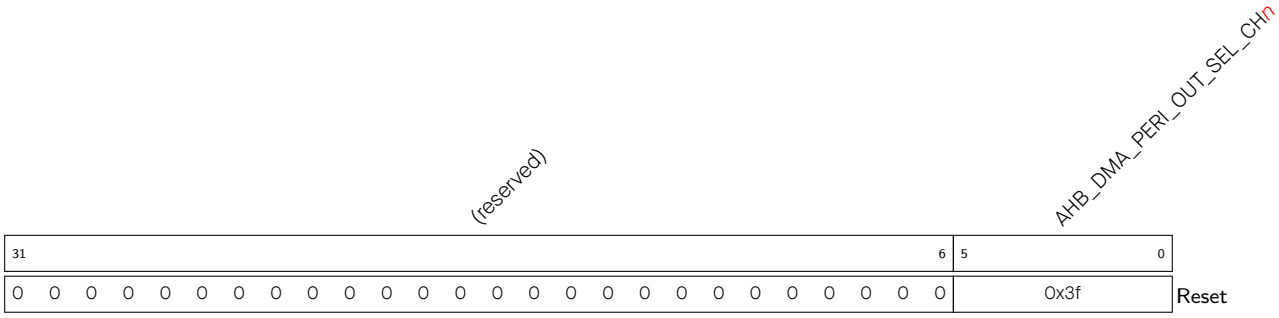
Register 3.64. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0n*)**



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

0: I3C
1: Dummy-1
2: UHCI
3: I2S0
4: I2S1
5: I2S2
6: Dummy-6
7: Dummy-7
8: ADC
9: Dummy-9
10: RMT
11 ~ 15: Dummy-11 ~ Dummy-15
16 ~ 63: Invalid
(R/W)

Register 3.65. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0**n*)



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

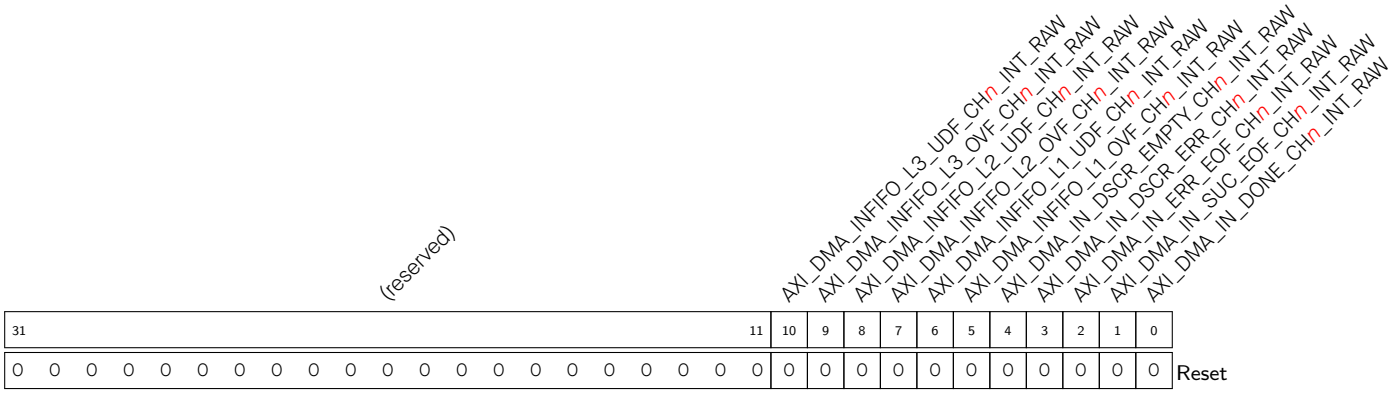
0: I3C
1: Dummy-1
2: UHCI
3: I2S0
4: I2S1
5: I2S2
6: Dummy-6
7: Dummy-7
8: ADC
9: Dummy-9
10: RMT
11 ~ 15: Dummy-11 ~ Dummy-15
16 ~ 63: Invalid
(R/W)

3.9.2 GDMA-AXI Registers

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

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

Register 3.66. AXI_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x68**n*)



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

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

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

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

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

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

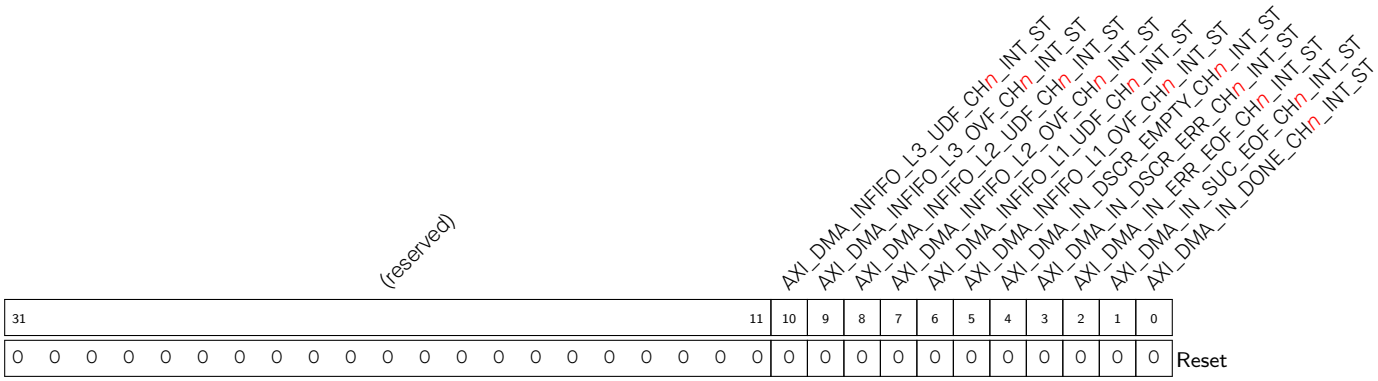
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

Register 3.67. AXI_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x68**n*)



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

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

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

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

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

AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (RO)

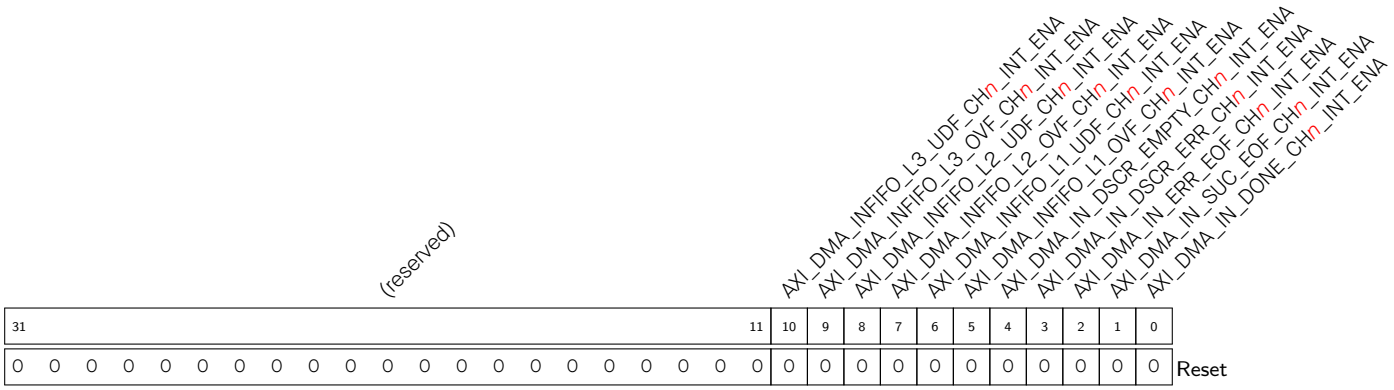
AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (RO)

Register 3.68. 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)

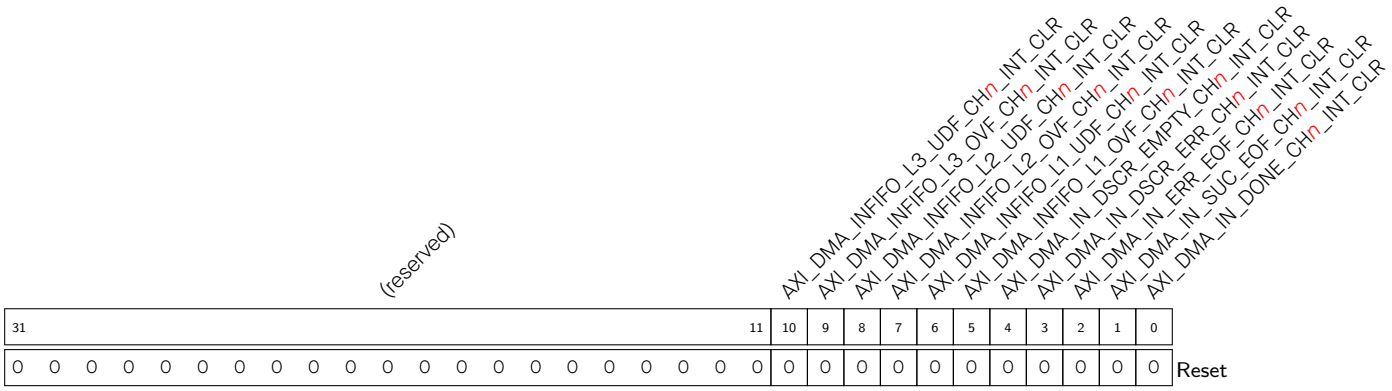
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (R/W)

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 3.69. AXI_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x68**n*)



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

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

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

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

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

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L1_UDF_CH*n*_INT. (WT)

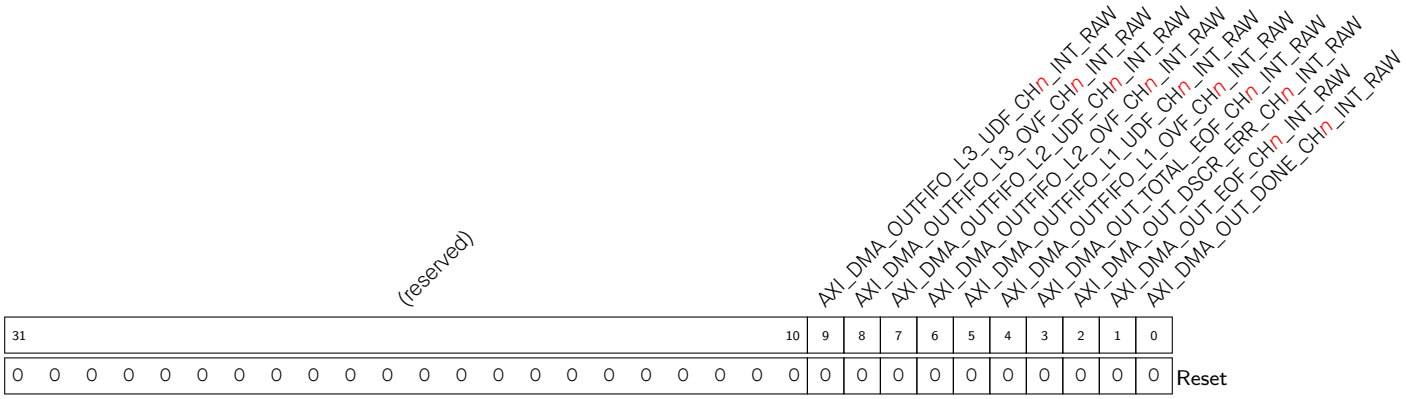
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L2_UDF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_OVF_CH*n*_INT. (WT)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_INFIFO_L3_UDF_CH*n*_INT. (WT)

Register 3.70. AXI_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0138+0x68n*)**



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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (R/WTC/SS)

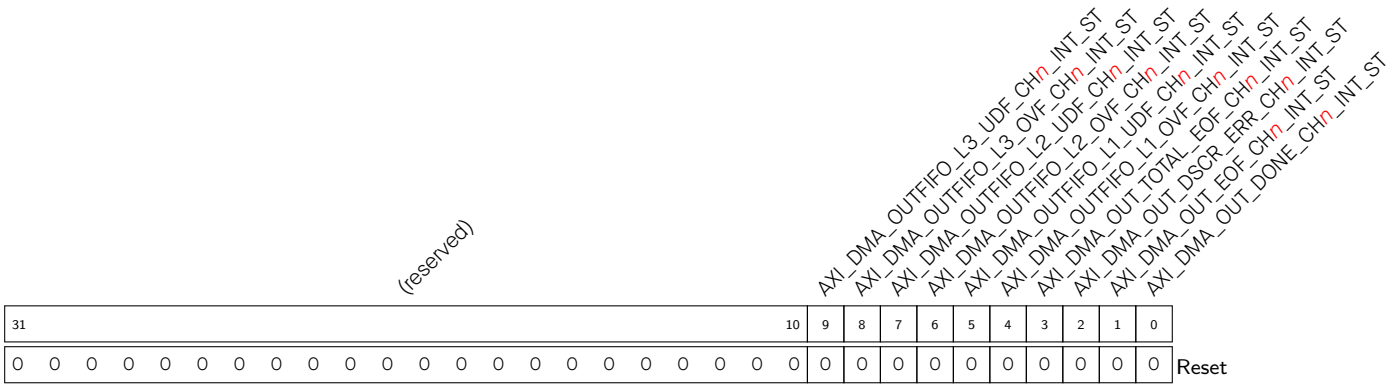
AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (R/WTC/SS)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_RAW The raw interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (R/WTC/SS)

Register 3.71. AXI_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x013C+0x68**n*)



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

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

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

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

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT. (RO)

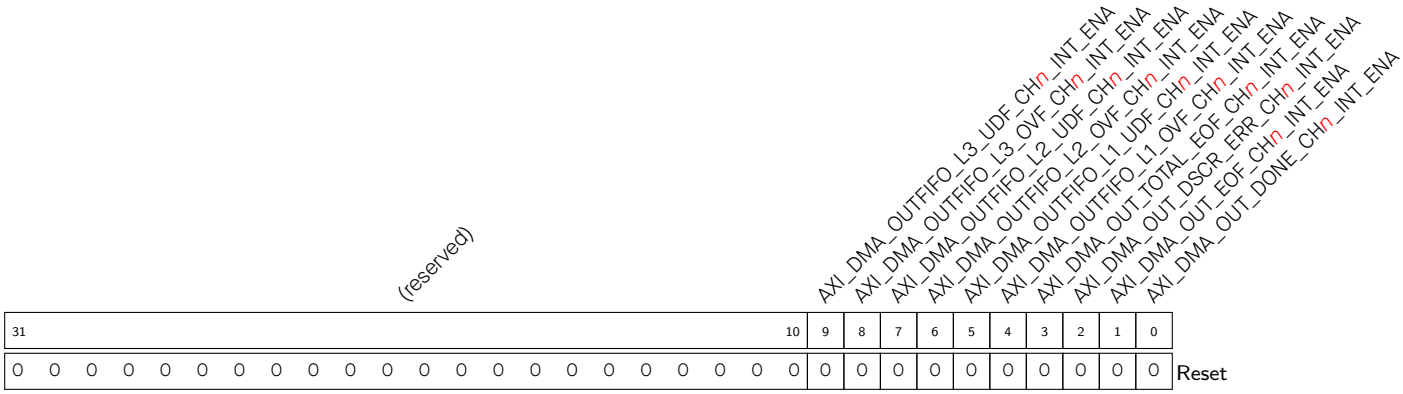
AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ST The masked interrupt status of AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT. (RO)

Register 3.72. 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)

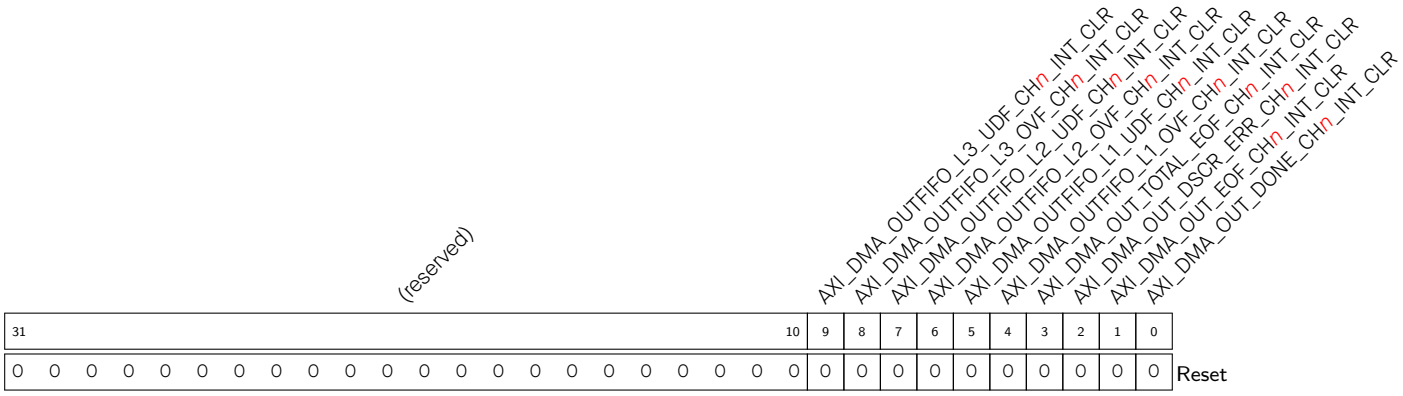
AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ENA Write 1 to enable AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (R/W)

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)

Register 3.73. AXI_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x0144+0x68n*)**



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)

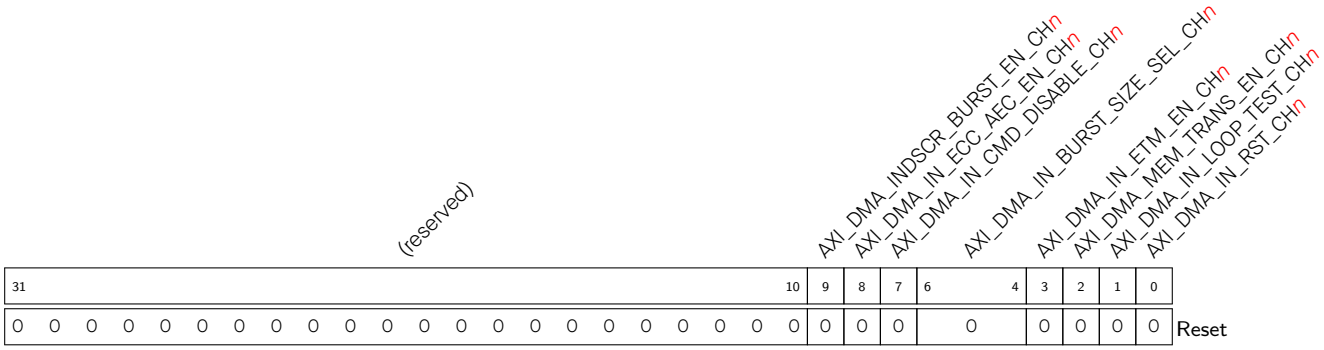
AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_CLR Write 1 to clear AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT. (WT)

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)

Register 3.74. AXI_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0010+0x68n*)**



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

AXI_DMA_IN_LOOP_TEST_CH*n* Reserved. (R/W)

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

0: Disable

1: Enable

(R/W)

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

0: Disable

1: Enable

(R/W)

AXI_DMA_IN_BURST_SIZE_SEL_CH*n* Configures the burst length for RX channel *n*.

0: 8 bytes

1: 16 bytes

2: 32 bytes

3: 64 bytes

4: 128 bytes

5 ~ 7: Invalid

(R/W)

AXI_DMA_IN_CMD_DISABLE_CH*n* Configures whether to disable command on RX channel *n*.

0: Enable

1: Disable

(R/W)

AXI_DMA_IN_ECC_AEC_EN_CH*n* Configures whether AXI DMA can access external memory space for ECC and AES via RX channel *n*.

0: Not access

1: Access

(R/W)

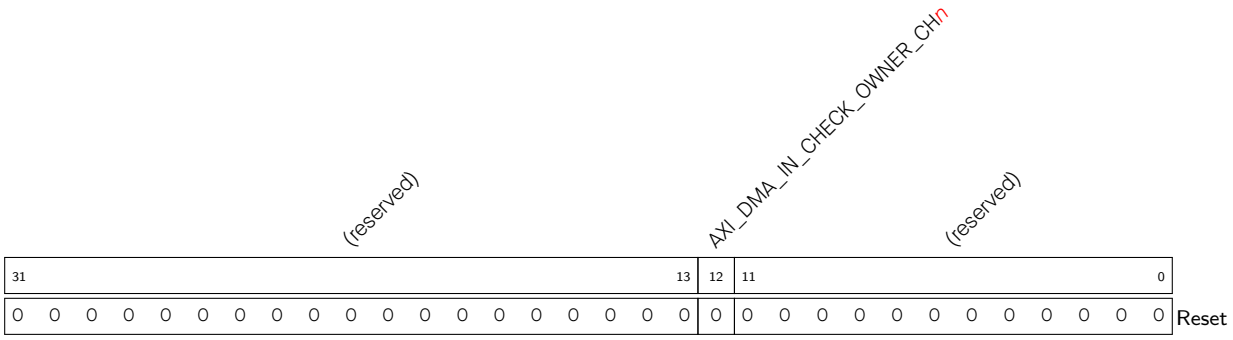
AXI_DMA_INDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for RX channel *n* to read descriptors when accessing internal memory.

0: Disable

1: Enable

(R/W)

Register 3.75. AXI_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0014+0x68n*)**



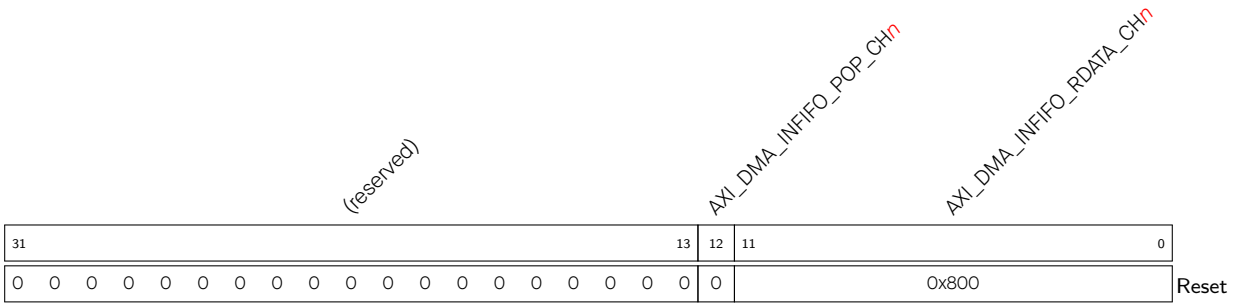
AXI_DMA_IN_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.76. AXI_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x001C+0x68n*)**



AXI_DMA_INFIFO_RDATA_CH*n* Represents the data popped from AXI DMA RX FIFO. (RO)

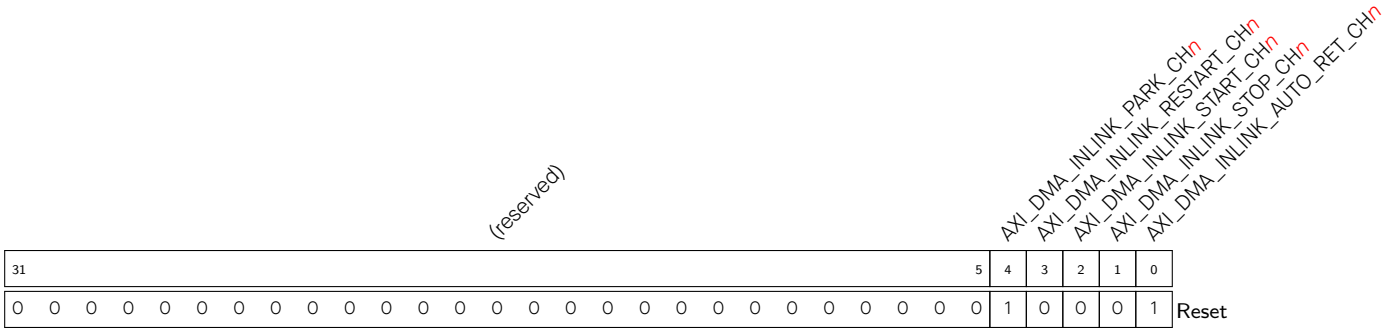
AXI_DMA_INFIFO_POP_CH*n* Configures whether to pop data from AXI DMA RX FIFO.

0: Invalid. No effect

1: Pop

(WT)

Register 3.77. 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's address when there are some errors in current receiving data.

0: Not return

1: Return

(R/W)

AXI_DMA_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AXI_DMA_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

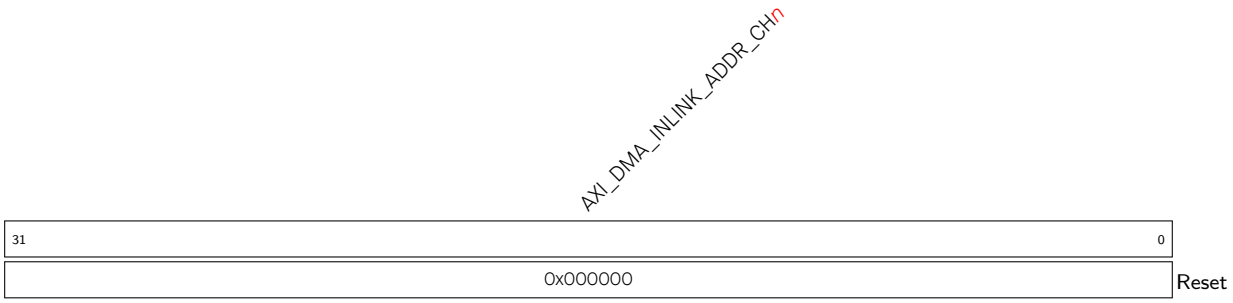
AXI_DMA_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

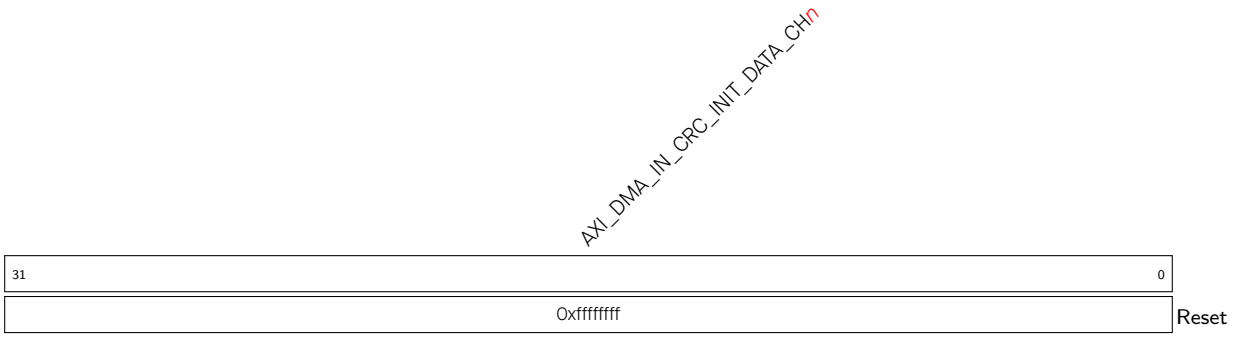
(RO)

Register 3.78. AXI_DMA_IN_LINK2_CH*n*_REG (*n*: 0-2) (0x0024+0x68n*)**



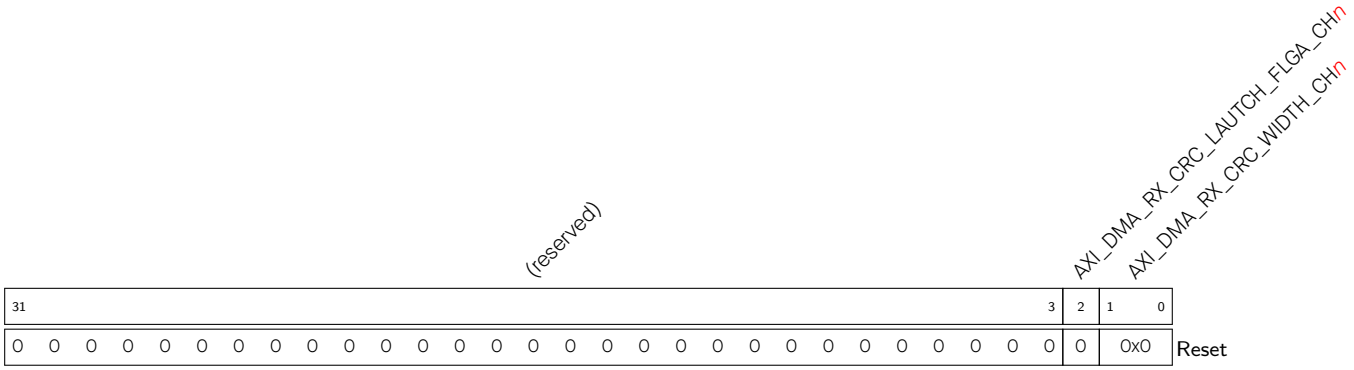
AXI_DMA_INLINK_ADDR_CH*n* Represents the first receive descriptor's address. (R/W)

Register 3.79. AXI_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0048+0x68n*)**



AXI_DMA_IN_CRC_INIT_DATA_CH*n* Configures the CRC initial value for RX channel *n*. (R/W)

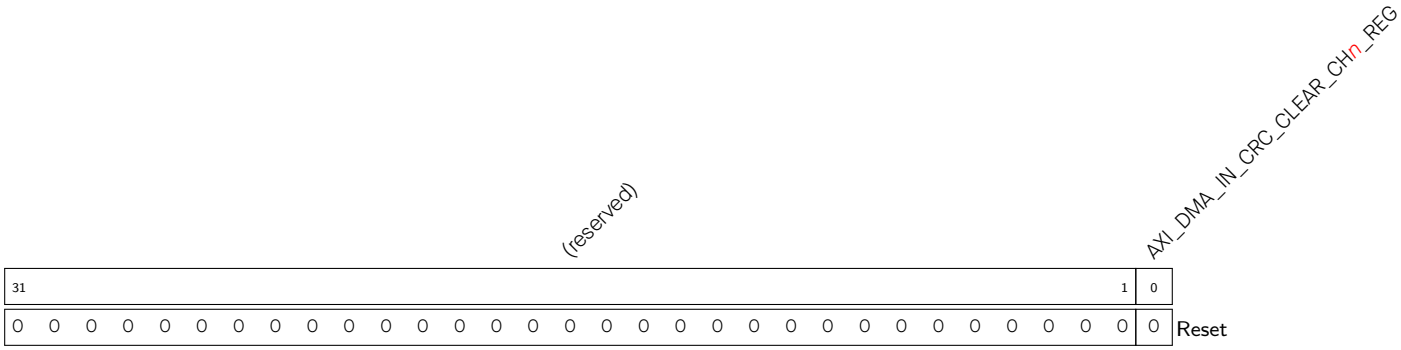
Register 3.80. AXI_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x004C+0x68n*)**



AXI_DMA_RX_CRC_WIDTH_CH*n* Configures the CRC result width for RX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

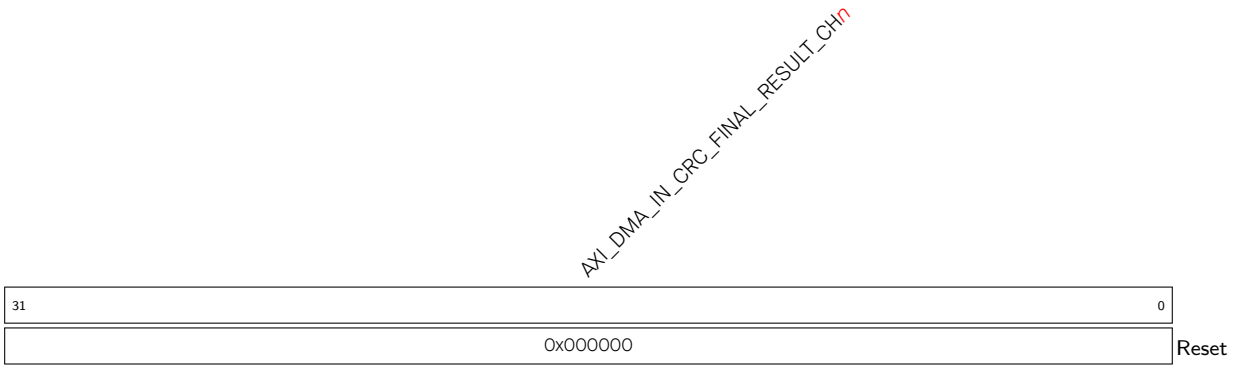
AXI_DMA_RX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AXI_DMA_RX_CRC_EN_ADDR_CH*n*, AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*, AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*, and AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 3.81. AXI_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x0050+0x68n*)**



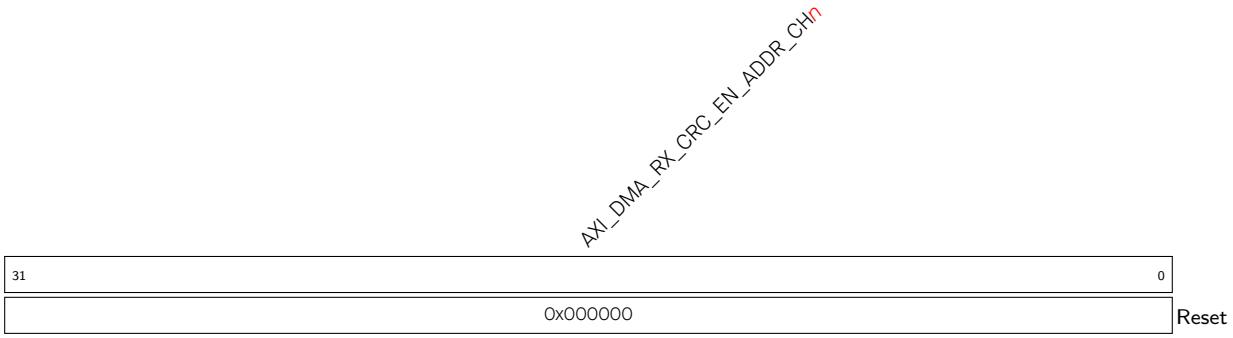
AXI_DMA_IN_CRC_CLEAR_CH*n*_REG Configures whether to clear the CRC result for RX channel *n*.
0: Not clear
1: Clear
(R/W)

Register 3.82. AXI_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x0054+0x68n*)**



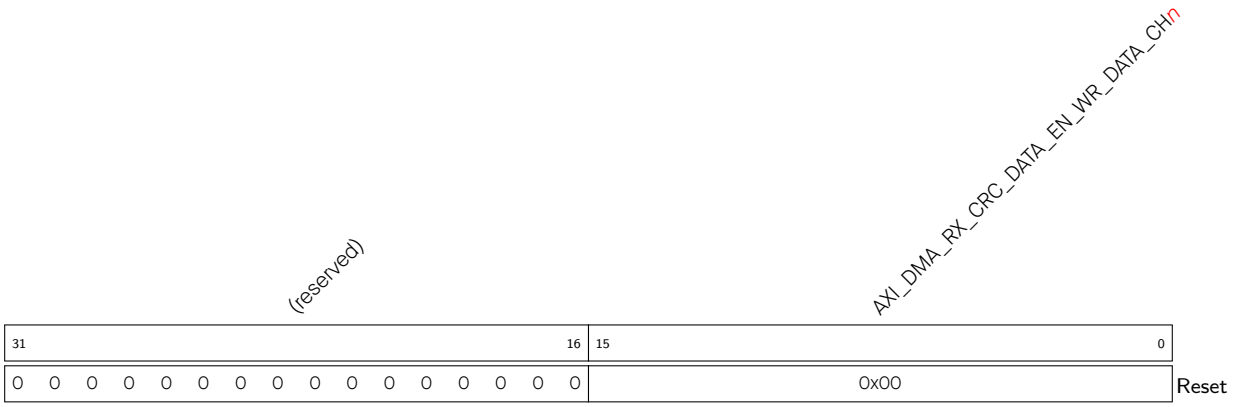
AXI_DMA_IN_CRC_FINAL_RESULT_CH*n* Represents the CRC result for RX channel *n*. (RO)

Register 3.83. AXI_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x005C+0x68n*)**



AXI_DMA_RX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 3.84. AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0060+0x68n*)**



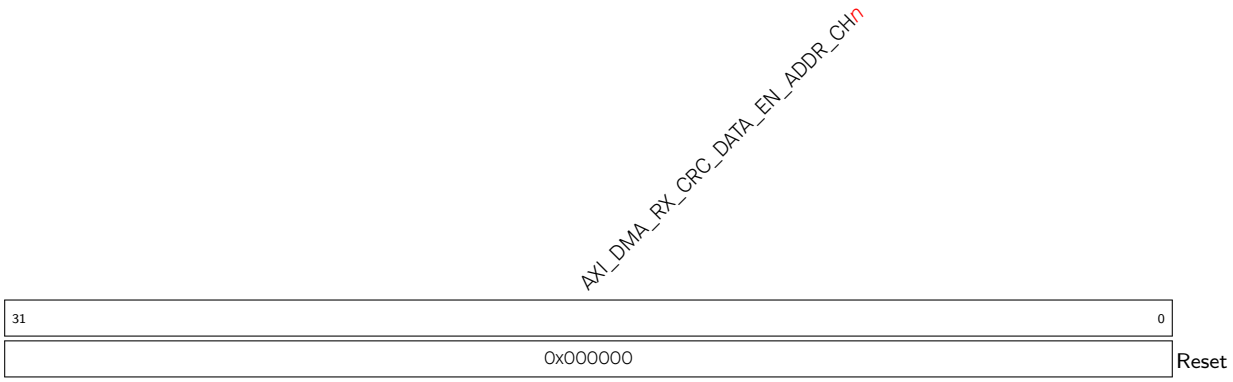
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

(R/W)

Register 3.85. AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0064+0x68n*)**



AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 3.86. AXI_DMA_OUT_CONF0_CHn_REG(n: 0-2) (0x0148+0x68*n)

(reserved)	AXI_DMA_OUTDSCR_BURST_EN_CHn	AXI_DMA_OUT_ECC_AEC_EN_CHn	AXI_DMA_OUT_CMD_DISABLE_CHn	AXI_DMA_OUT_BURST_SIZE_SEL_CHn	AXI_DMA_OUT_ETM_EN_CHn	AXI_DMA_OUT_EOF_MODE_CHn	AXI_DMA_OUT_AUTO_WRBACK_CHn	AXI_DMA_OUT_LOOP_TEST_CHn	AXI_DMA_OUT_RST_CHn	
31 ... 11	10	9	8	7 ... 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	1	0	0	0	Reset

AXI_DMA_OUT_RST_CHn Configures the reset state of TX channel n FSM and TX FIFO pointer.
- 0: Release reset
- 1: Reset
- (R/W)

AXI_DMA_OUT_LOOP_TEST_CHn Reserved. (R/W)

AXI_DMA_OUT_AUTO_WRBACK_CHn Configures whether to enable automatic outlink write-back when all the data in TX FIFO has been transmitted.
- 0: Disable
- 1: Enable
- (R/W)

AXI_DMA_OUT_EOF_MODE_CHn Configures when to generate EOF flag.
- 0: EOF flag for TX channel n is generated when data to be transmitted has been pushed into FIFO in AXI DMA.
- 1: EOF flag for TX channel n is generated when data to be transmitted has been popped from FIFO in AXI DMA.
- (R/W)

AXI_DMA_OUT_ETM_EN_CHn Configures whether to enable ETM control for TX channel n.
- 0: Disable
- 1: Enable
- ((R/W)

AXI_DMA_OUT_BURST_SIZE_SEL_CHn Configures the burst length for TX channel n.
- 0: 8 bytes
- 1: 16 bytes
- 2: 32 bytes
- 3: 64 bytes
- 4: 128 bytes
- 5 ~ 7: Invalid
- (R/W)

AXI_DMA_OUT_CMD_DISABLE_CHn Configures whether to disable command on TX channel n.
- 0: Enable
- 1: Disable
- (R/W)

Continued on the next page...

Register 3.86. AXI_DMA_OUT_CONF0_CH*n*_REG(*n*: 0-2) (0x0148+0x68**n*)

Continued from the previous page...

AXI_DMA_OUT_ECC_AEC_EN_CH*n* Configures whether AXI DMA can access external memory space for ECC and AES via TX channel *n*.

0: Not access

1: Access

(R/W)

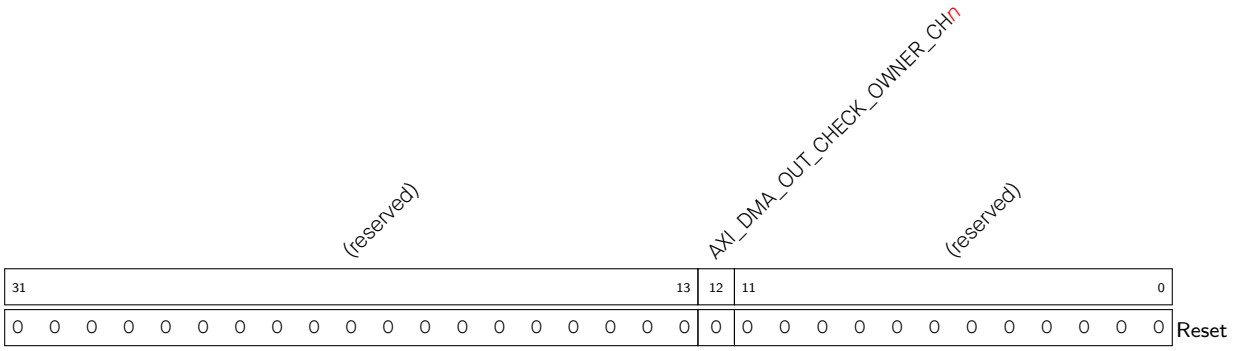
AXI_DMA_OUTDSCR_BURST_EN_CH*n* Configures whether to enable INCR burst transfer for TX channel *n* reading descriptors when accessing internal memory.

0: Disable

1: Enable

(R/W)

Register 3.87. AXI_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x014C+0x68**n*)



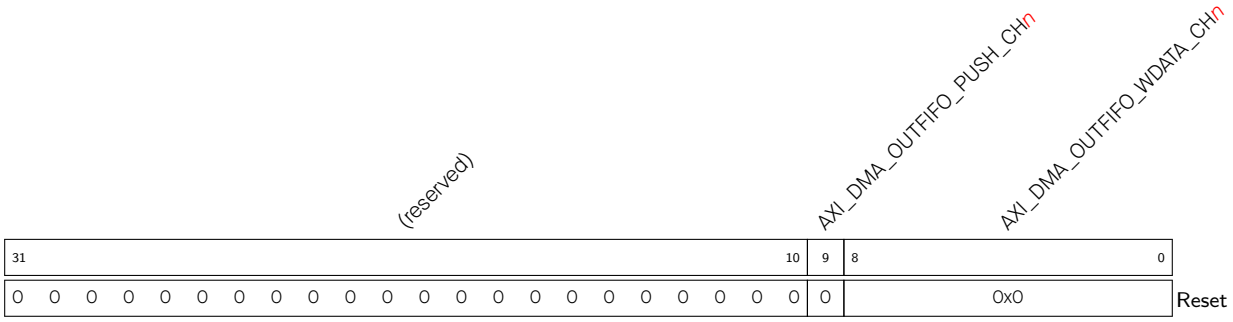
AXI_DMA_OUT_CHECK_OWNER_CH*n* Configures whether to enable owner bit check for TX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.88. AXI_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x0154+0x68n*)**



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

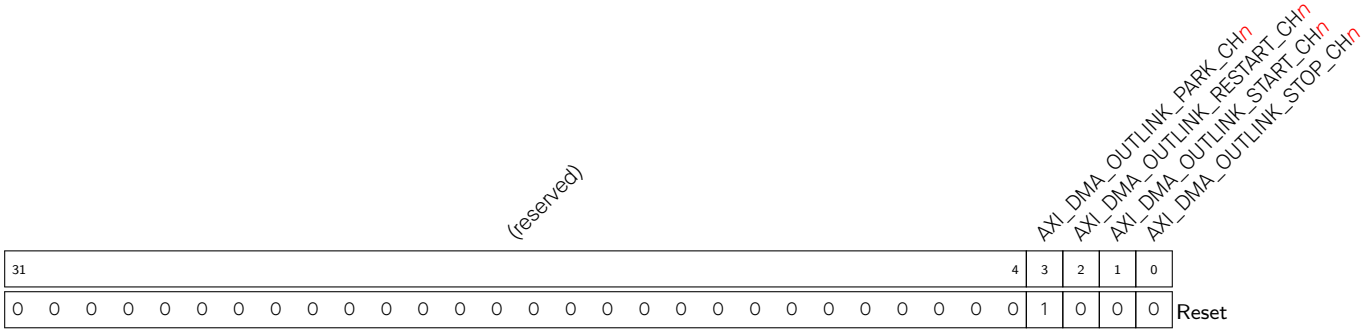
AXI_DMA_OUTFIFO_PUSH_CH*n* Configures whether to push data into AXI DMA TX FIFO.

0: Invalid. No effect

1: Push

(WT)

Register 3.89. AXI_DMA_OUT_LINK1_CH*n*_REG (*n*: 0-2) (0x0158+0x68**n*)



AXI_DMA_OUTLINK_STOP_CH*n* Configures whether to stop TX channel *n* from transmitting data.

0: Invalid. No effect

1: Stop

(WT)

AXI_DMA_OUTLINK_START_CH*n* Configures whether to enable TX channel *n* for data transfer.

0: Disable

1: Enable

(WT)

AXI_DMA_OUTLINK_RESTART_CH*n* Configures whether to restart TX channel *n* for AXI DMA transfer.

0: Invalid. No effect

1: Restart

(WT)

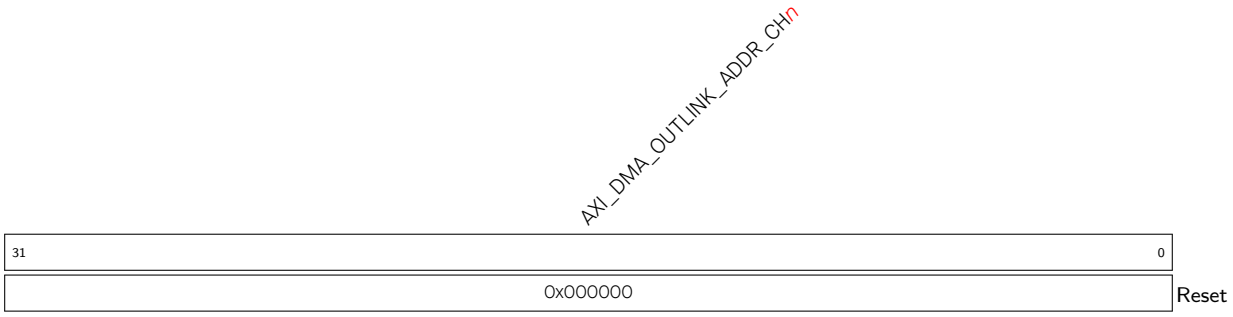
AXI_DMA_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

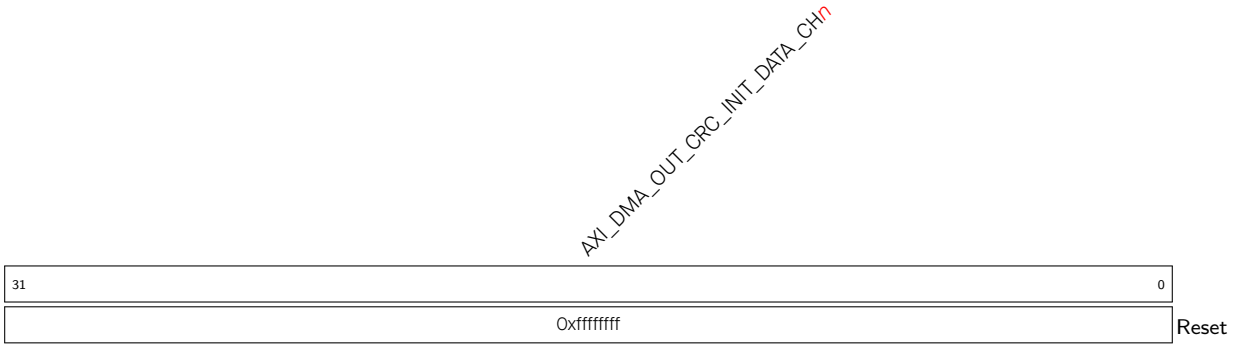
(RO)

Register 3.90. AXI_DMA_OUT_LINK2_CH*n*_REG (*n*: 0-2) (0x015C+0x68**n*)



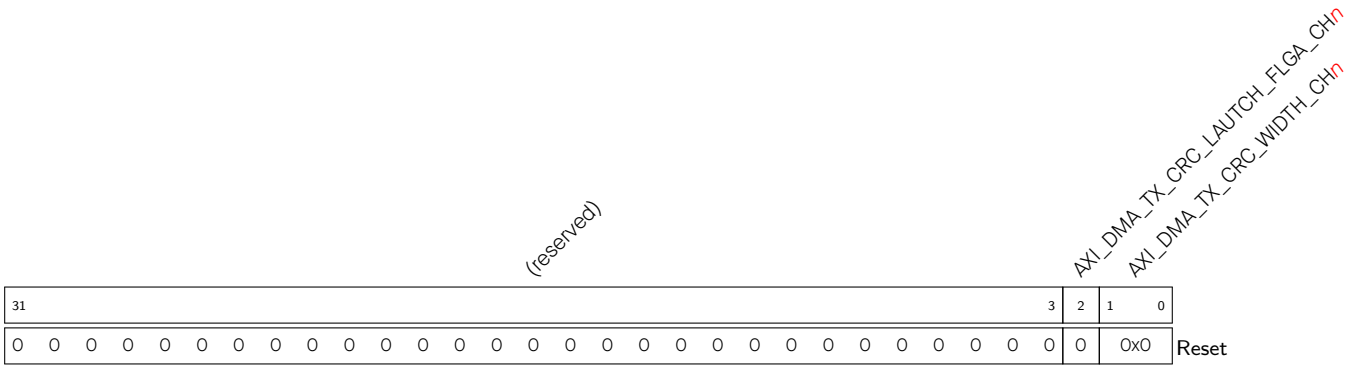
AXI_DMA_OUTLINK_ADDR_CH*n* Represents the first transmit descriptor's address. (R/W)

Register 3.91. AXI_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0180+0x68n*)**



AXI_DMA_OUT_CRC_INIT_DATA_CH*n* Configures the CRC initial value for TX channel *n*. (R/W)

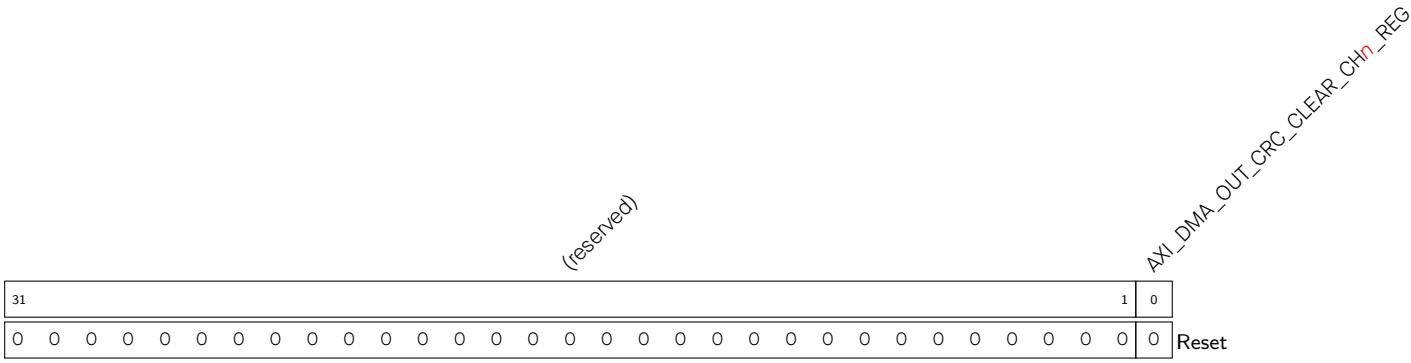
Register 3.92. AXI_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x0184+0x68n*)**



AXI_DMA_TX_CRC_WIDTH_CH*n* Configures the CRC result width for TX channel *n*. 0: $\leq$ 8 bits
1: $\leq$ 16 bits
2: $\leq$ 24 bits
3: $\leq$ 32 bits
(R/W)

AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n* Write 1 and then 0 to latch the values of AXI_DMA_TX_CRC_EN_ADDR_CH*n*, AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*, AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*, and AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*. (R/W)

Register 3.93. AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x0188+0x68**n*)



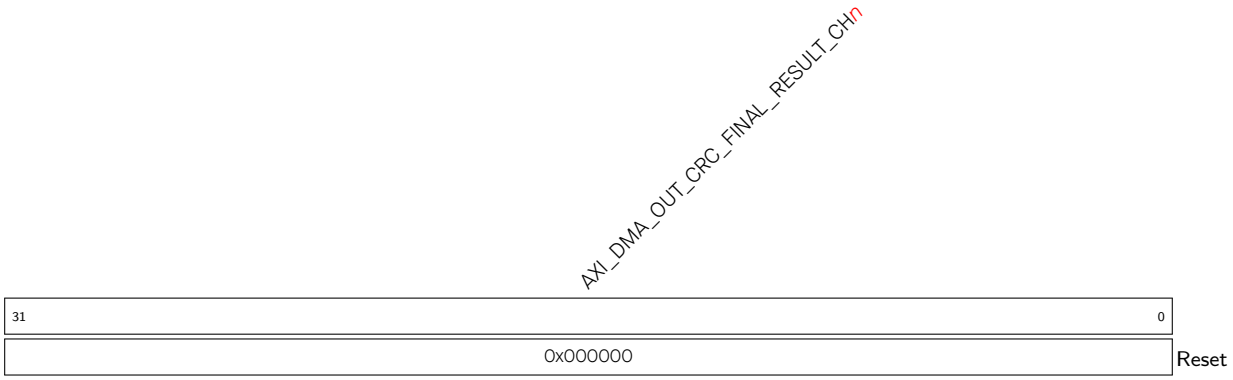
AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG Configures whether to clear the CRC result for TX channel *n*.

0: Not clear

1: Clear

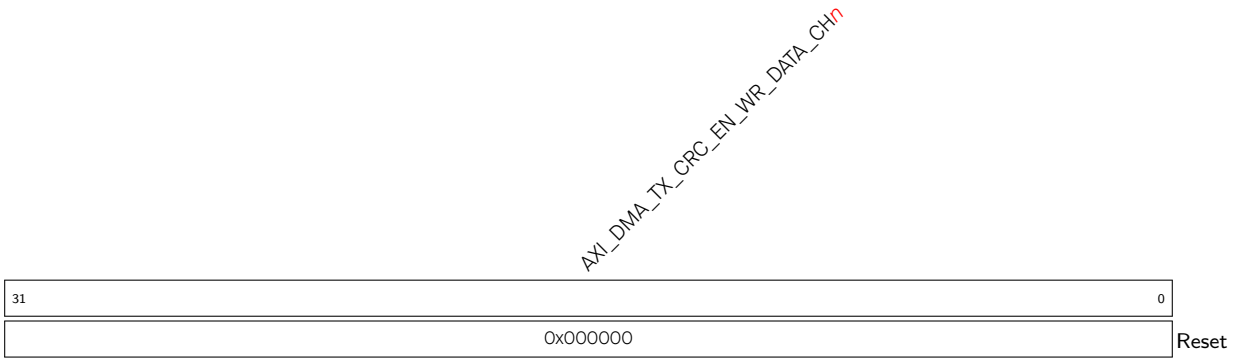
(R/W)

Register 3.94. AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x018C+0x68**n*)



AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n* Represents the CRC result for TX channel *n*. (RO)

Register 3.95. AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0190+0x68**n*)



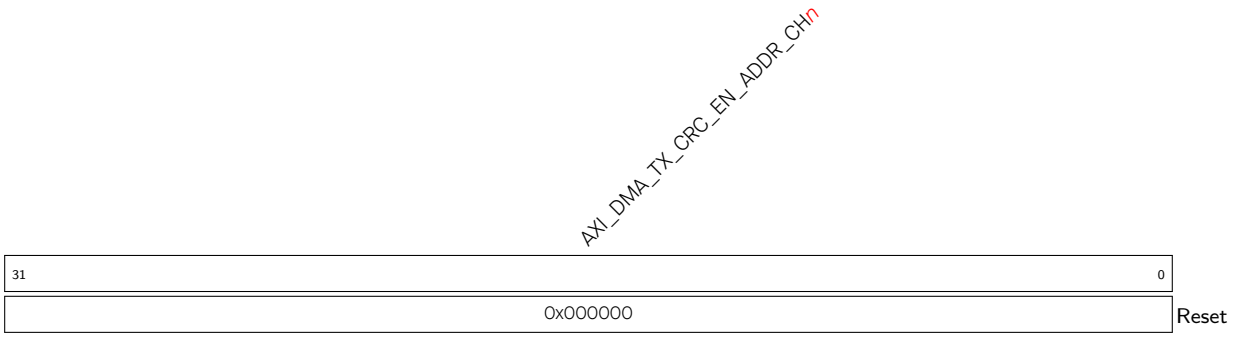
AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

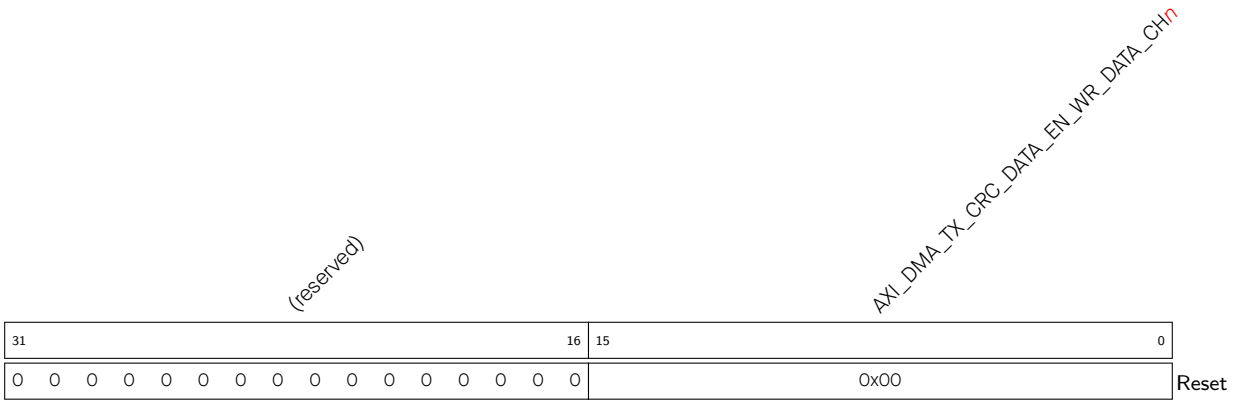
(R/W)

Register 3.96. AXI_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0194+0x68**n*)



AXI_DMA_TX_CRC_EN_ADDR_CH*n* Configures at which bit of the CRC result the AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*. (R/W)

Register 3.97. AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0198+0x68n*)**



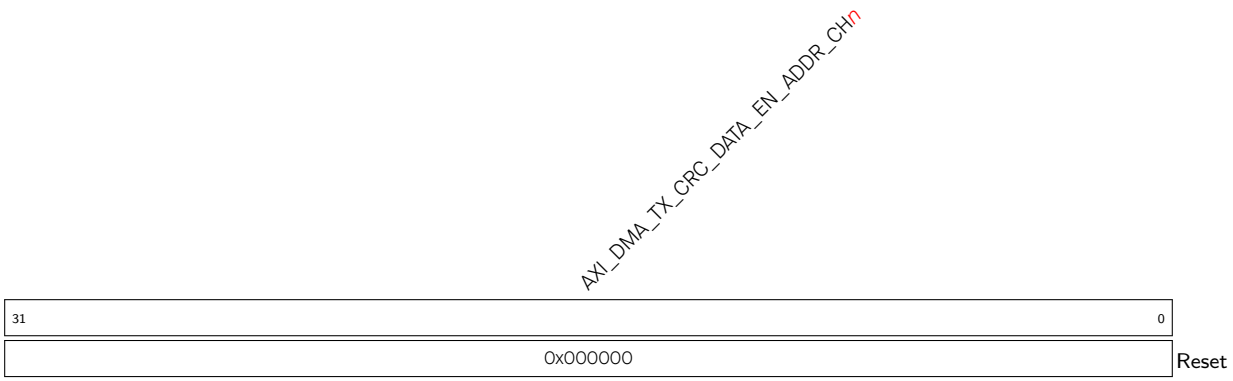
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* Configures whether to include each bit of the input data in the CRC calculation matrix for TX channel *n*.

0: Not include

1: Include

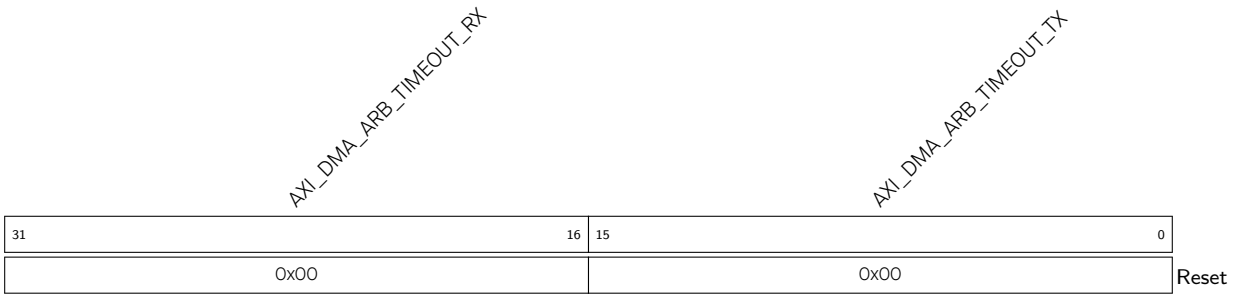
(R/W)

Register 3.98. AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x019C+0x68n*)**



AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n* Configures at which bit of the CRC result the AAXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG register targets for TX channel *n*.(R/W)

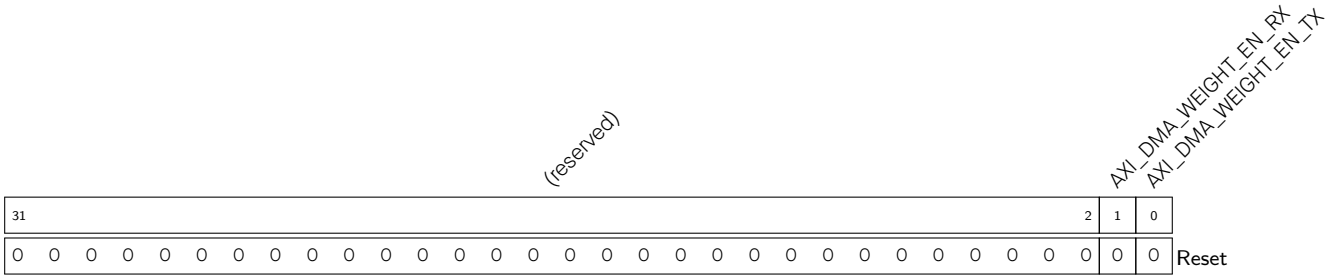
Register 3.99. AXI_DMA_ARB_TIMEOUT_REG (0x0270)



AXI_DMA_ARB_TIMEOUT_TX Configures the time slot for TX. Measurement unit: AXI bus clock cycle. (R/W)

AXI_DMA_ARB_TIMEOUT_RX Configures the time slot for RX. Measurement unit: AXI bus clock cycle. (R/W)

Register 3.100. AXI_DMA_WEIGHT_EN_REG (0x0274)



AXI_DMA_WEIGHT_EN_TX Configures whether to enable weight arbitration for TX.

0: Disable

1: Enable

(R/W)

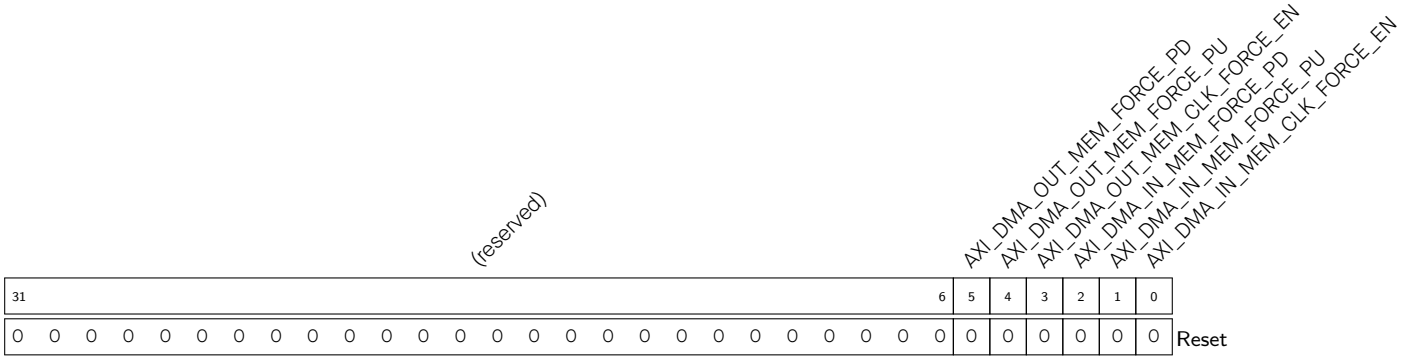
AXI_DMA_WEIGHT_EN_RX Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

(R/W)

Register 3.101. AXI_DMA_IN_MEM_CONF_REG (0x0278)



AXI_DMA_IN_MEM_CLK_FORCE_EN Configures memory clock gating for RX.

0: Support clock only when AXI DMA accesses memory

1: Always force the clock on

(R/W)

AXI_DMA_IN_MEM_FORCE_PU Configures whether to force power up memory for RX.

0: Not force power up

1: Force power up

(R/W)

AXI_DMA_IN_MEM_FORCE_PD Configures whether to force power down memory for RX.

0: Not force power down

1: Force power down

(R/W)

AXI_DMA_OUT_MEM_CLK_FORCE_EN Configures memory clock gating for TX.

0: Support clock only when AXI DMA accesses memory

1: Always force the clock on

(R/W)

AXI_DMA_OUT_MEM_FORCE_PU Configures whether to force power up memory for TX.

0: Not force power up

1: Force power up

(R/W)

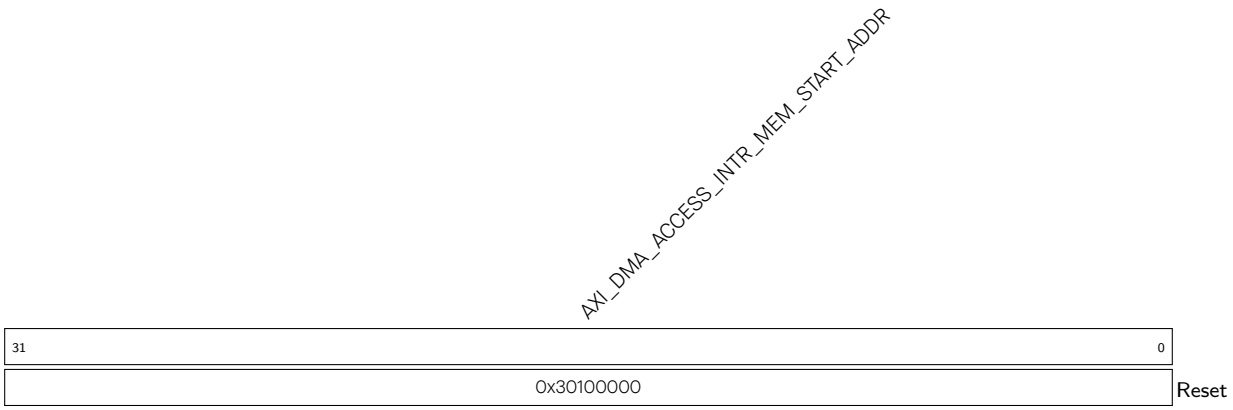
AXI_DMA_OUT_MEM_FORCE_PD Configures whether to force power down memory for TX.

0: Not force power down

1: Force power down

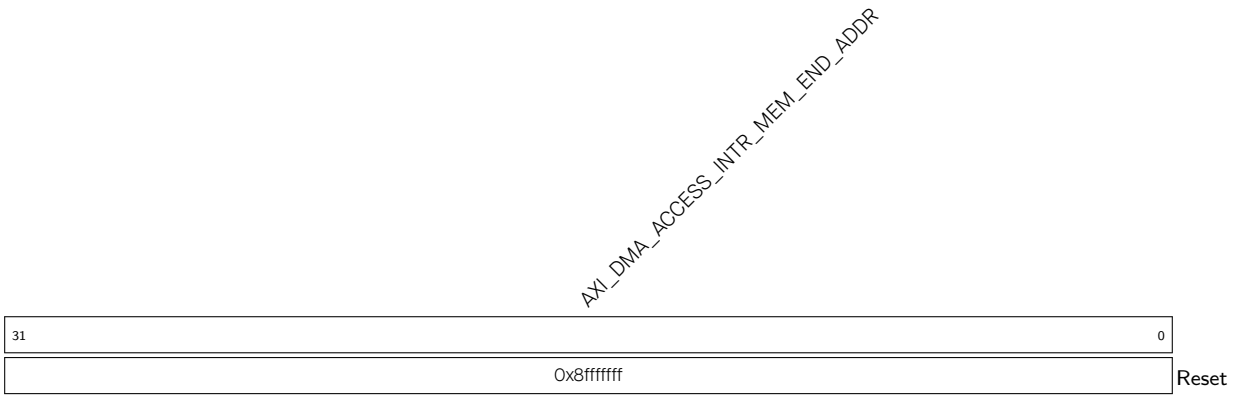
(R/W)

Register 3.102. AXI_DMA_INTR_MEM_START_ADDR_REG (0x027C)



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

Register 3.103. AXI_DMA_INTR_MEM_END_ADDR_REG (0x0280)



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

Register 3.104. AXI_DMA_EXTR_MEM_START_ADDR_REG (0x0284)



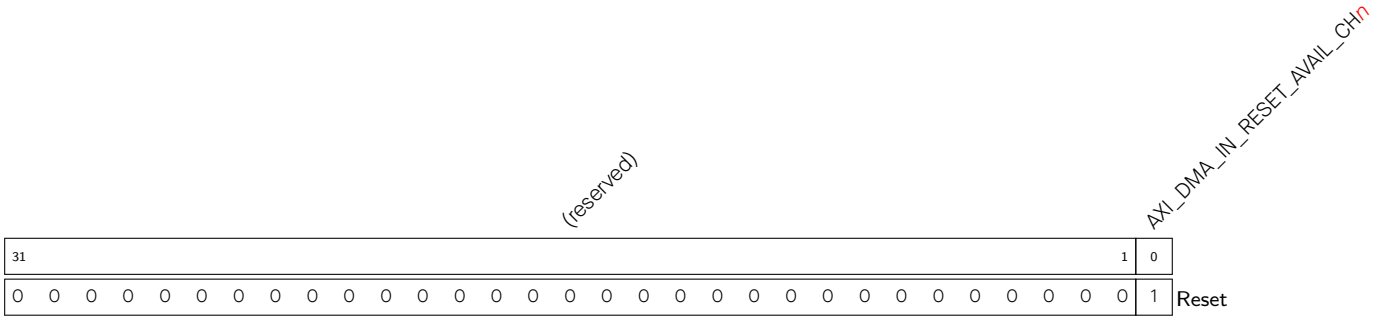
AXI_DMA_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible external memory address space. (R/W)

Register 3.105. AXI_DMA_EXTR_MEM_END_ADDR_REG (0x0288)



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

Register 3.106. AXI_DMA_IN_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x028C+0x4n*)**



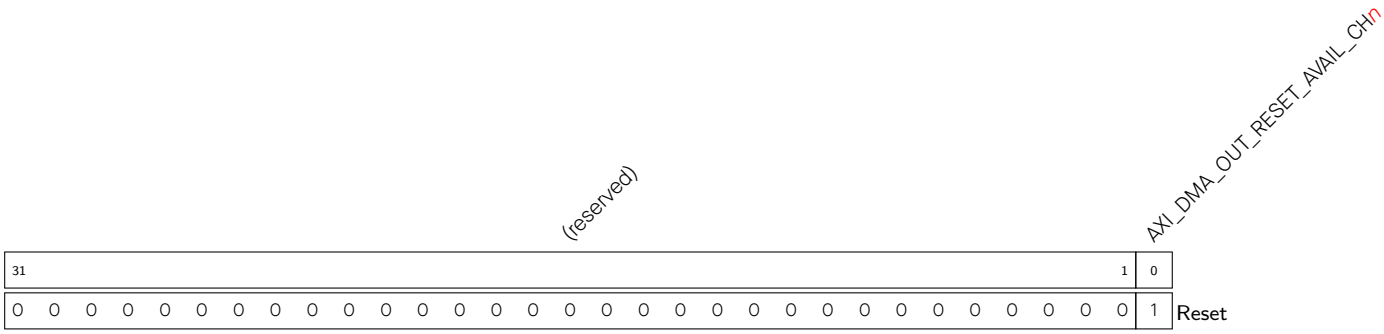
AXI_DMA_IN_RESET_AVAIL_CH*n* Represents whether RX channel *n* can be reset.

0: Cannot be reset by software

1: Can be reset

(RO)

Register 3.107. AXI_DMA_OUT_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x0298+0x4n*)**



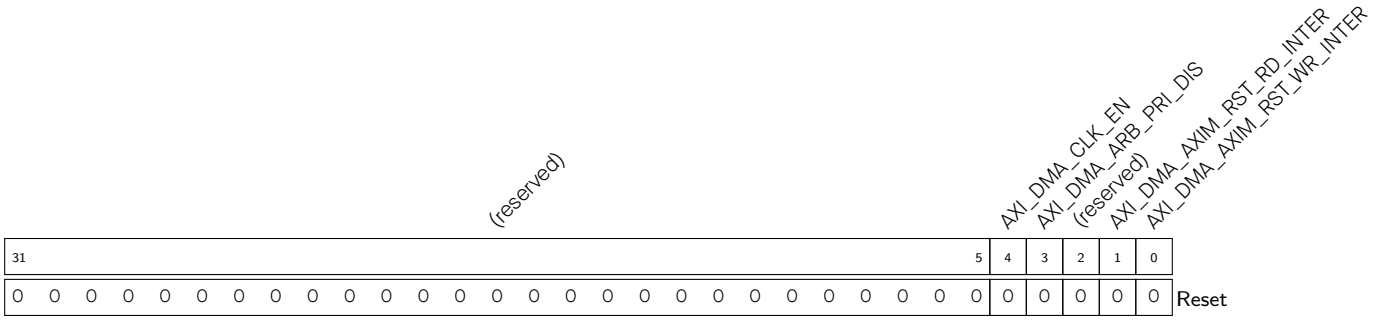
AXI_DMA_OUT_RESET_AVAIL_CH*n* Represents whether TX channel *n* can be reset.

0: Cannot be reset by software

1: Can be reset

. (RO)

Register 3.108. AXI_DMA_MISC_CONF_REG (0x02A8)



AXI_DMA_AXIM_RST_WR_INTER Write 1 and then 0 to reset the internal AXI write FSM. (R/W)

AXI_DMA_AXIM_RST_RD_INTER Write 1 and then 0 to reset the internal AXI read FSM. (R/W)

AXI_DMA_ARB_PRI_DIS Configures whether to disable the priority arbitration.

0: Enable

1: Disable

(R/W)

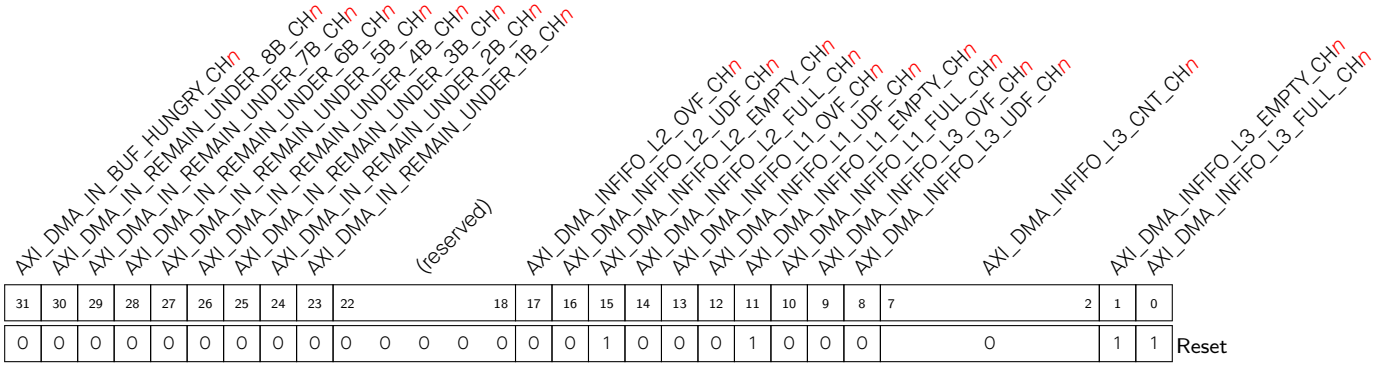
AXI_DMA_CLK_EN Configures AXI DMA clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 3.109. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68n*)**



AXI_DMA_INFIFO_L3_FULL_CH*n* Represents whether L3 RX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_INFIFO_L3_EMPTY_CH*n* Represents whether L3 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_INFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 RX FIFO for RX channel *n*. (RO)

AXI_DMA_INFIFO_L3_UDF_CH*n* Represents whether L3 RX FIFO underflows.

0: No underflow

1: Underflow

(RO)

AXI_DMA_INFIFO_L3_OVF_CH*n* Represents whether L3 RX FIFO overflows.

0: No overflow

1: Overflow

(RO)

AXI_DMA_INFIFO_L1_FULL_CH*n* Represents whether L1 RX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_INFIFO_L1_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_INFIFO_L1_UDF_CH*n* Represents whether L1 RX FIFO underflows.

0: No underflow

1: Underflow

(RO)

Continued on the next page...

Register 3.109. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0018+0x68**n*)

Continued from the previous page...

AXI_DMA_INFIFO_L1_OVF_CH*n* Represents whether L1 RX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_INFIFO_L2_FULL_CH*n* Represents whether L2 RX FIFO is full.
0: Not Full
1: Full
(RO)

AXI_DMA_INFIFO_L2_EMPTY_CH*n* Represents whether L1 RX FIFO is empty.
0: Not empty
1: Empty
(RO)

AXI_DMA_INFIFO_L2_UDF_CH*n* Represents whether L2 RX FIFO underflows.
0: No underflow
1: Underflow
(RO)

AXI_DMA_INFIFO_L2_OVF_CH*n* Represents whether L2 RX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_IN_REMAIN_UNDER_1B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_2B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_3B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_4B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_5B_CH*n* Reserved.(RO)

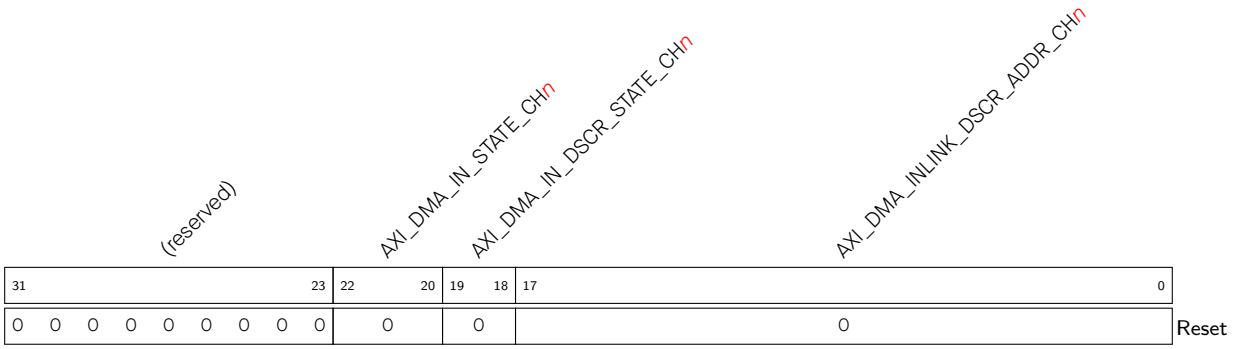
AXI_DMA_IN_REMAIN_UNDER_6B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_7B_CH*n* Reserved.(RO)

AXI_DMA_IN_REMAIN_UNDER_8B_CH*n* Reserved.(RO)

AXI_DMA_IN_BUF_HUNGRY_CH*n* Reserved.(RO)

Register 3.110. AXI_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0028+0x68**n*)

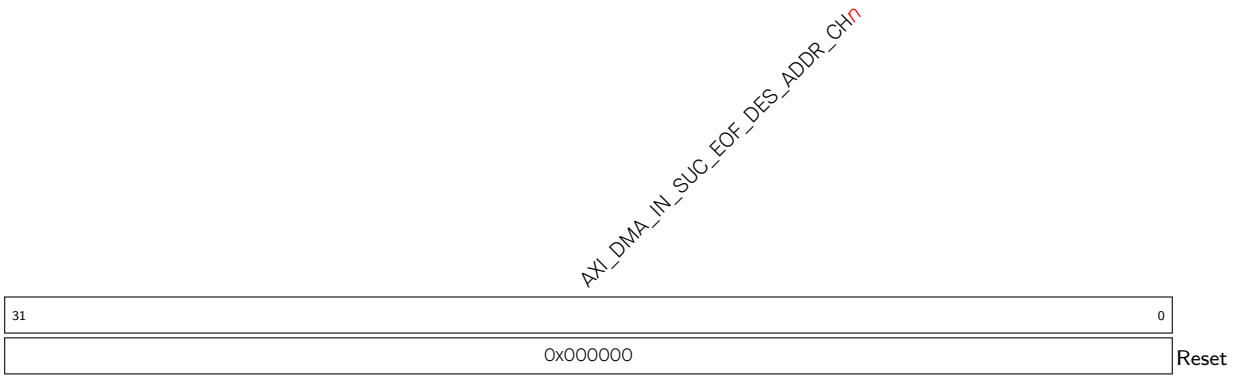


AXI_DMA_INLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

AXI_DMA_IN_DSCR_STATE_CH*n* Reserved. (RO)

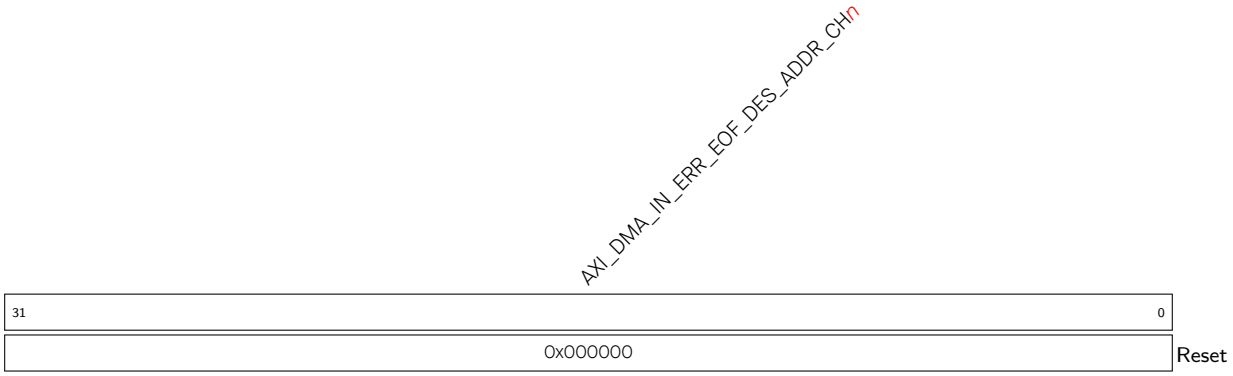
AXI_DMA_IN_STATE_CH*n* Reserved. (RO)

Register 3.111. AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x002C+0x68**n*)



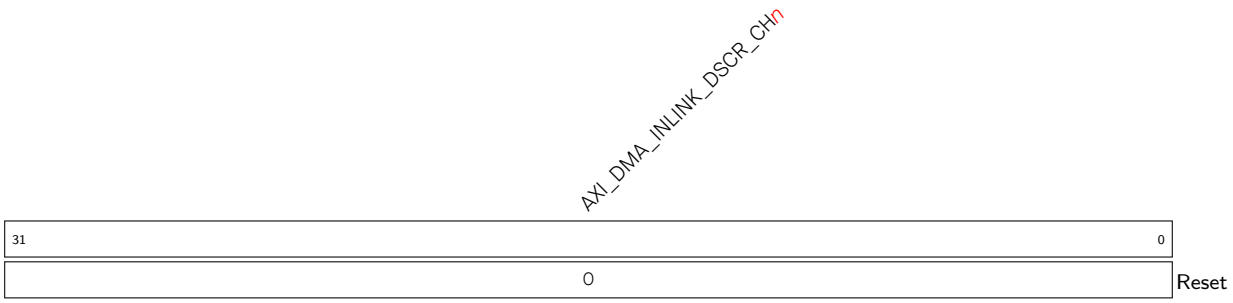
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.112. AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0030+0x68**n*)



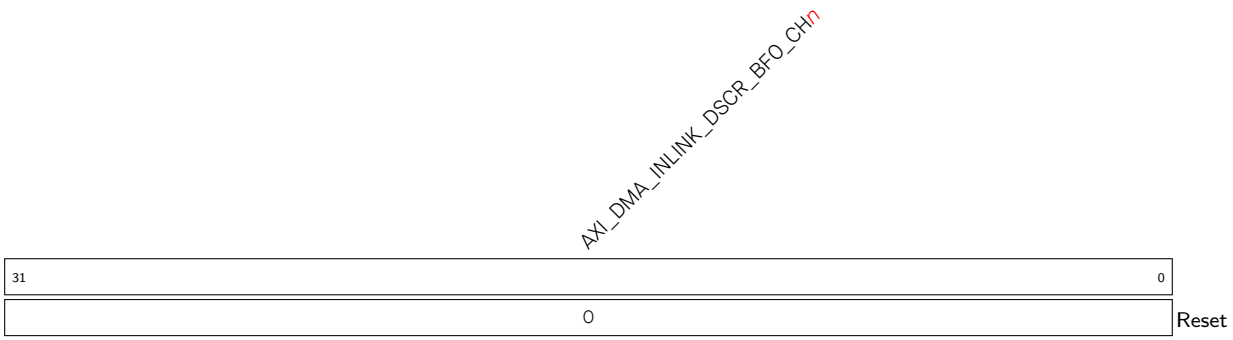
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for UHCI. (RO)

Register 3.113. AXI_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0034+0x68**n*)



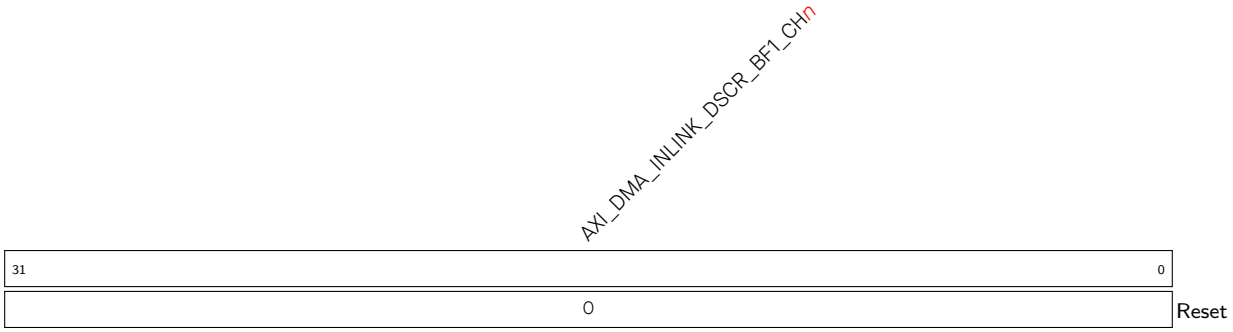
AXI_DMA_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 3.114. AXI_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0038+0x68**n*)



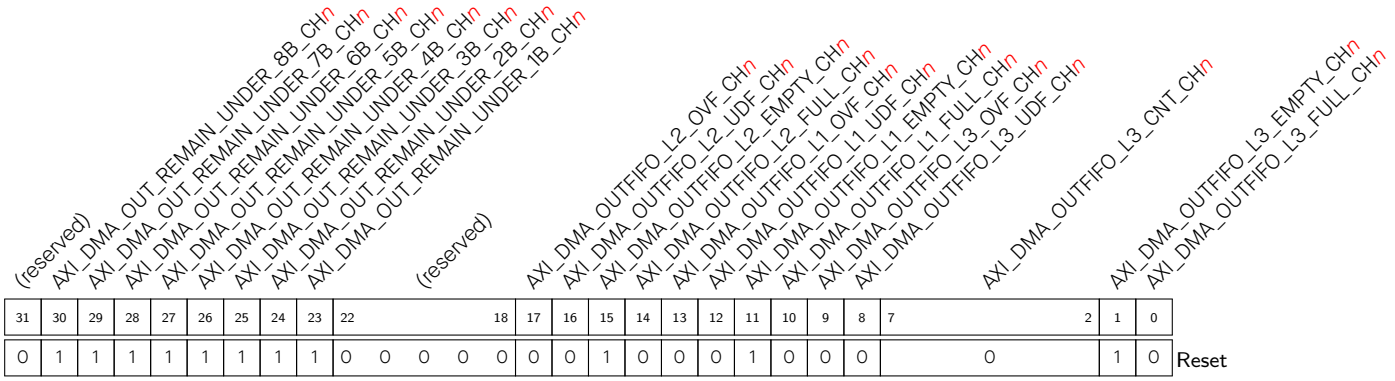
AXI_DMA_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 3.115. AXI_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x003C+0x68*n*)**



AXI_DMA_INLINK_DSCR_BF1_CH*n* Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 3.116. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68**n*)



AXI_DMA_OUTFIFO_L3_FULL_CH*n* Represents whether L3 TX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_OUTFIFO_L3_EMPTY_CH*n* Represents whether L3 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_OUTFIFO_L3_CNT_CH*n* Represents the number of data bytes in L3 TX FIFO for TX channel *n*. (RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n* Represents whether L3 TX FIFO underflows.

0: No underflow

1: Underflow

(RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n* Represents whether L3 TX FIFO overflows.

0: No overflow

1: Overflow

(RO)

AXI_DMA_OUTFIFO_L1_FULL_CH*n* Represents whether L1 TX FIFO is full.

0: Not Full

1: Full

(RO)

AXI_DMA_OUTFIFO_L1_EMPTY_CH*n* Represents whether L1 TX FIFO is empty.

0: Not empty

1: Empty

(RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n* Represents whether L1 TX FIFO underflows.

0: No underflow

1: Underflow

(RO)

Continued on the next page...

Register 3.116. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0150+0x68**n*)

Continued from the previous page...

AXI_DMA_OUTFIFO_L1_OVF_CH*n* Represents whether L1 TX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_OUTFIFO_L2_FULL_CH*n* Represents whether L2 TX FIFO is full.
0: Not Full
1: Full
(RO)

AXI_DMA_OUTFIFO_L2_EMPTY_CH*n* Represents whether L2 TX FIFO is empty.
0: Not empty
1: Empty
(RO)

AXI_DMA_OUTFIFO_L2_UDF_CH*n* Represents whether L2 TX FIFO underflows.
0: No underflow
1: Underflow
(RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n* Represents whether L2 TX FIFO overflows.
0: No overflow
1: Overflow
(RO)

AXI_DMA_OUT_REMAIN_UNDER_1B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_2B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_3B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_4B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_5B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_6B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_7B_CH*n* Reserved.(RO)

AXI_DMA_OUT_REMAIN_UNDER_8B_CH*n* Reserved.(RO)

Register 3.117. AXI_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x0160+0x68**n*)

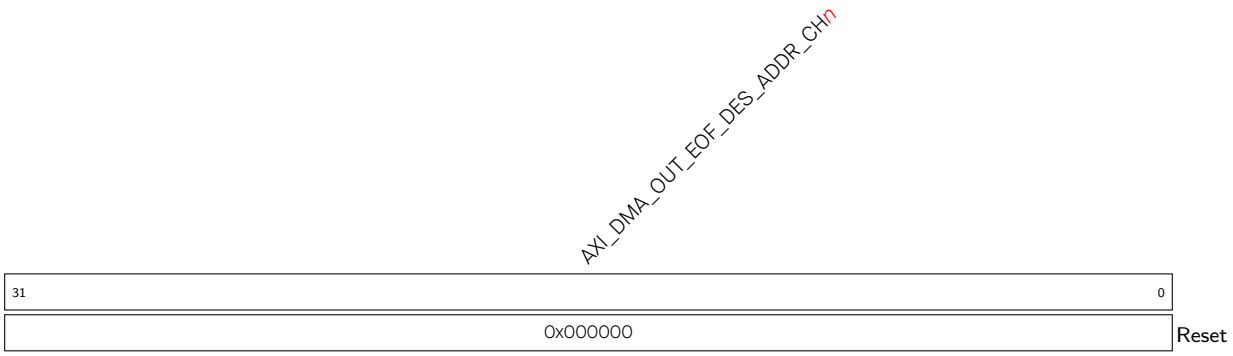


AXI_DMA_OUTLINK_DSCR_ADDR_CH*n* Represents the lower 18 bits of the next transmit descriptor address that is pre-read (but not processed yet). If the current transmit descriptor is the last descriptor, then this field represents the address of the current transmit descriptor. (RO)

AXI_DMA_OUT_DSCR_STATE_CH*n* Reserved. (RO)

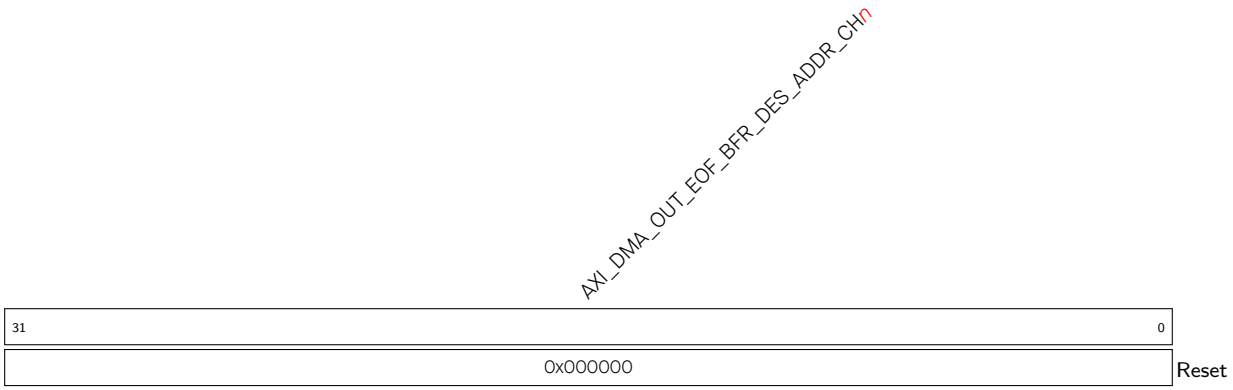
AXI_DMA_OUT_STATE_CH*n* Reserved. (RO)

Register 3.118. AXI_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0164+0x68**n*)



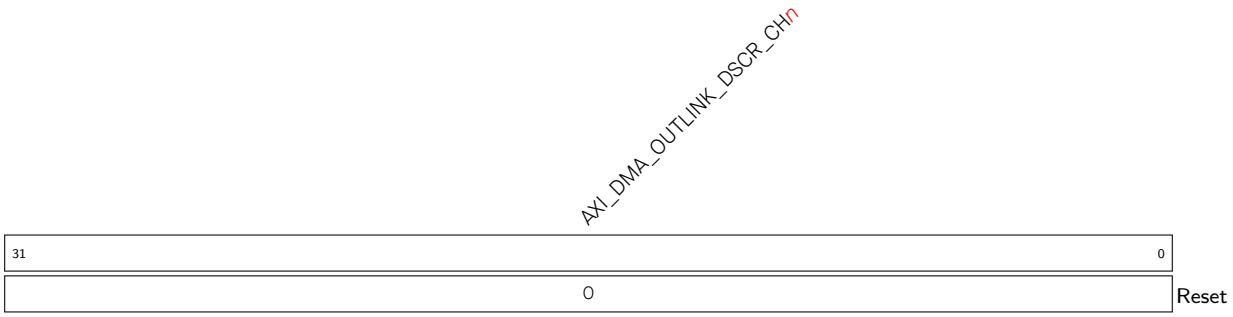
AXI_DMA_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 3.119. AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0168+0x68**n*)



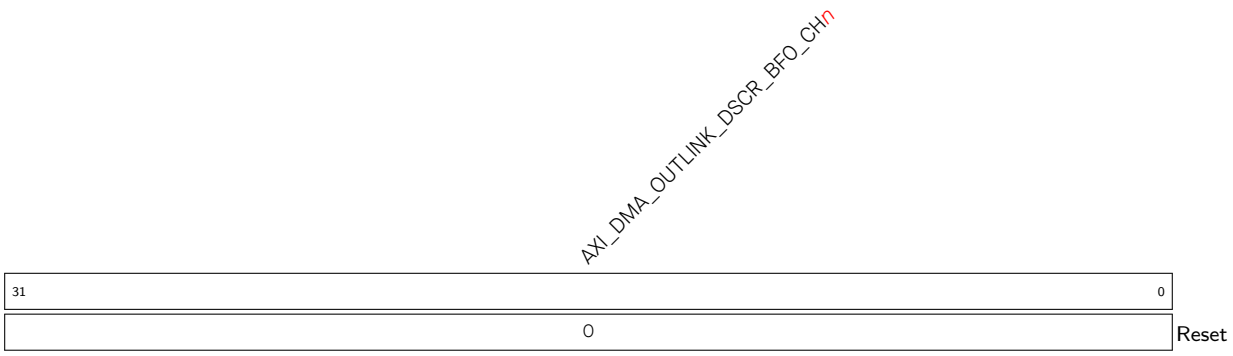
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* Represents the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 3.120. AXI_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x016C+0x68**n*)



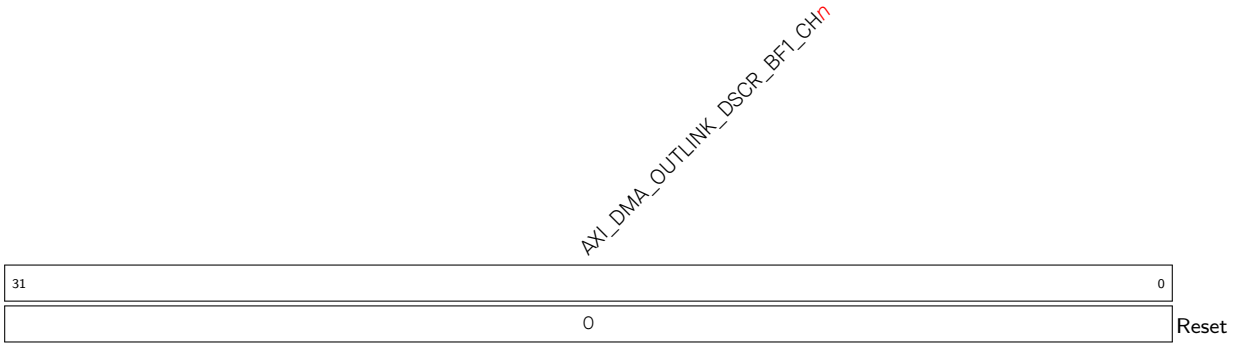
AXI_DMA_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 3.121. AXI_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0170+0x68**n*)



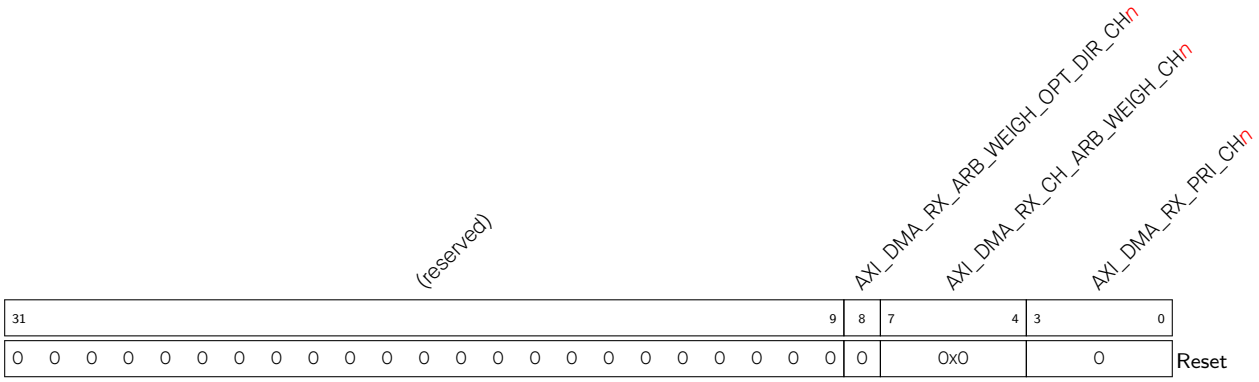
AXI_DMA_OUTLINK_DSCR_BF0_CH*n* Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 3.122. AXI_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0174+0x68**n*)



AXI_DMA_OUTLINK_DSCR_BF1_CH*n* Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

Register 3.123. 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*. The larger the value, the higher the priority.

Value range: 0 ~ 5. (R/W)

AXI_DMA_RX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

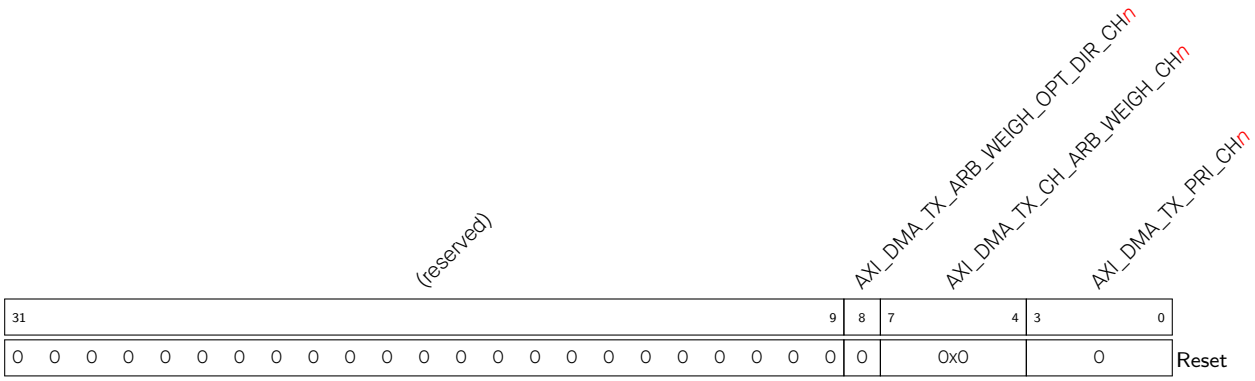
AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.124. AXI_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x0178+0x68*n*)**



AXI_DMA_TX_PRI_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 5. (R/W)

AXI_DMA_TX_CH_ARB_WEIGH_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

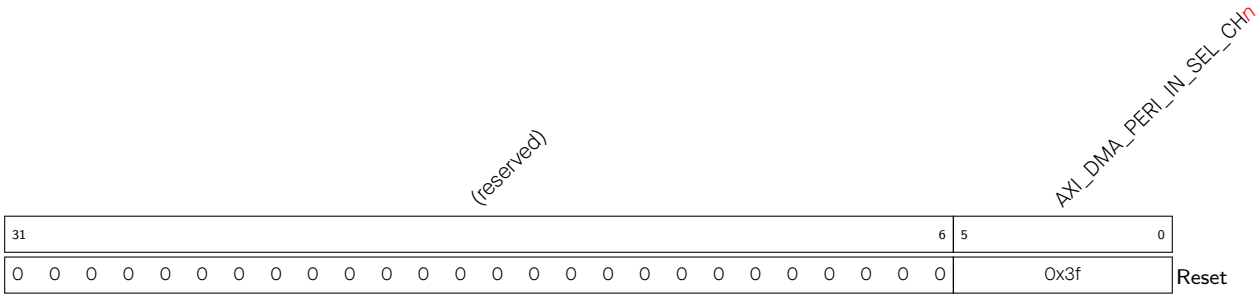
AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* Configures whether to enable weight optimization for RX channel *n*.

0: Disable

1: Enable

(R/W)

Register 3.125. AXI_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0044+0x68*n*)**



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

0: LCD_CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

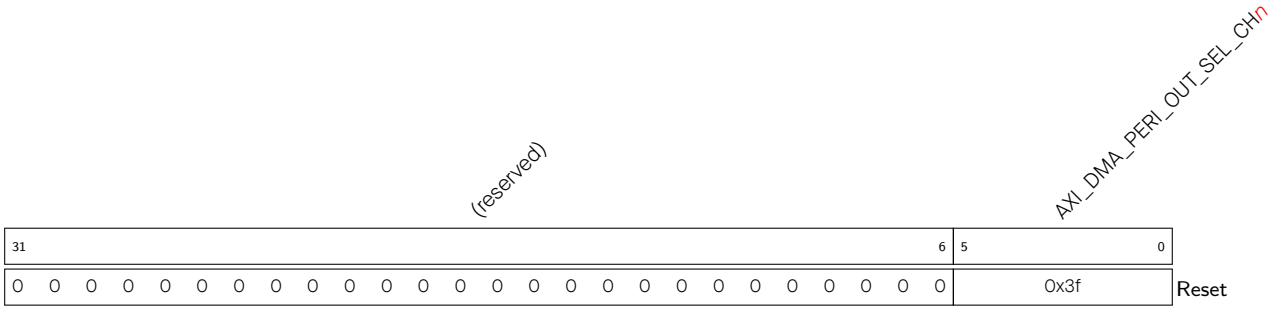
5: SHA

6 ~ 15: Dummy-6 ~ Dummy-15

16 ~ 63: Invalid

(R/W)

Register 3.126. AXI_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x017C+0x68n*)**



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

0: LCD_CAM

1: SPI2

2: SPI3

3: PARLIO

4: AES

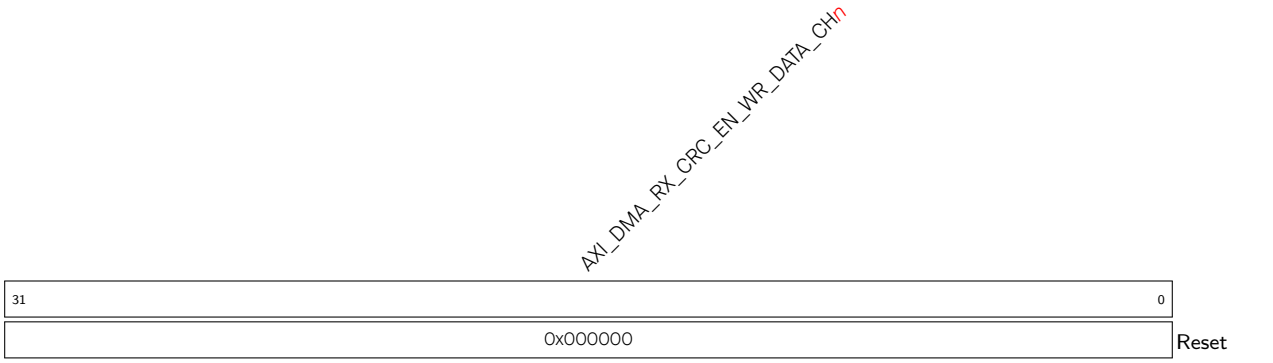
5: SHA

6 ~ 15: Dummy-6 ~ Dummy-15

16 ~ 63: Invalid

(R/W)

Register 3.127. AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0058+0x68n*)**



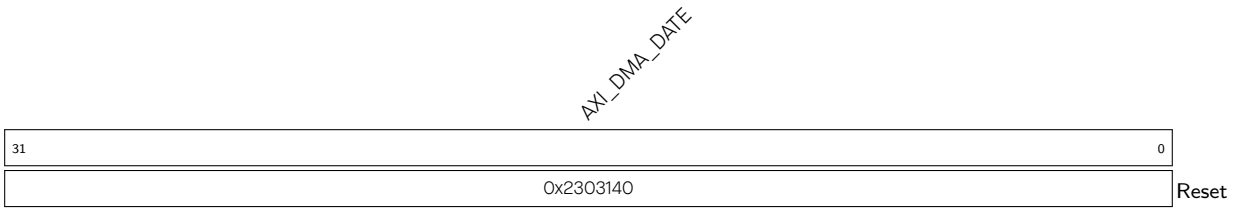
AXI_DMA_RX_CRC_EN_WR_DATA_CH*n* Configures whether to include each bit of the intermediate result in the CRC calculation matrix for RX channel *n*.

0: Not include

1: Include

(R/W)

Register 3.128. 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
- Three memory-to-peripheral channels, and two 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-P4, only JPEG Codec and PPA support 2D-DMA. The 2D-DMA controller and CPU data bus have access to the same address space in memory. Figure 4.3-1 shows the basic architecture of the 2D-DMA controller.

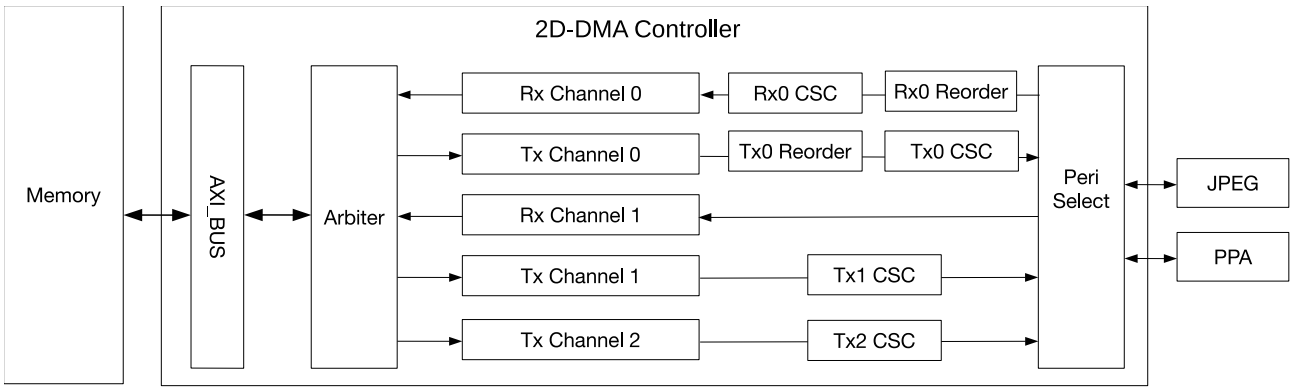


Figure 4.3-1. 2D-DMA Architecture

The 2D-DMA has five independent channels: three transmit channels and two 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 five channels, transmit channel 0 and receive channel 0 support macroblock reordering, while transmit channel 0 ~ 2 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 *5 System and Memory*.

The software can use the 2D-DMA through linked lists stored in internal or external memory. These linked lists consist of outlink*n* and inlink*n*, where *n* indicates the channel number. For outlink*n*, *n* ranges from 0 to 2; for inlink*n*, *n* is 0 or 1. 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 28 *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 29 *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

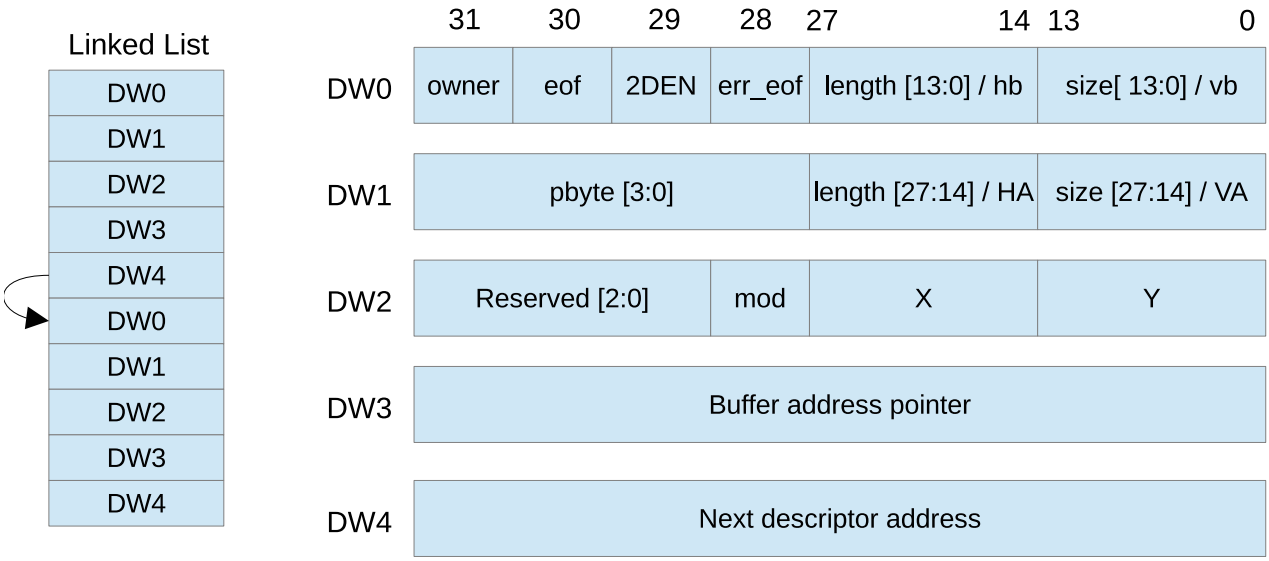


Figure 4.4-1. Structure of a Linked List

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 8x8, 8x16, or 16x16 (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 20x20 (YUV420) or 18x18. 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 8x8, 8x16, or 16x16 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.

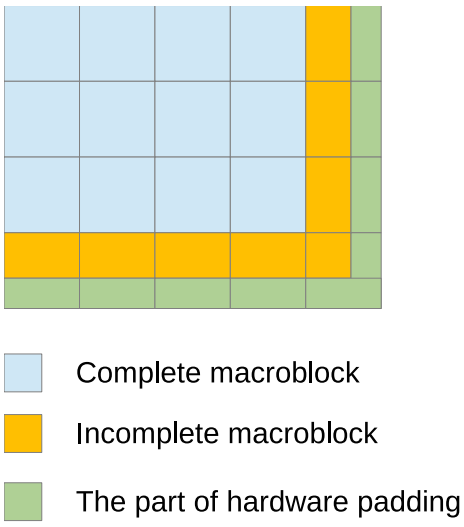


Figure 4.4-2. Padding Illustration (Overall)

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 BLEND module
3 ~ 7	Dummy-3 ~ 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 BLEND0 channel
3	PPA's BLEND1 channel
4 ~ 7	Dummy-4 ~ 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 28 *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	16x16	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.

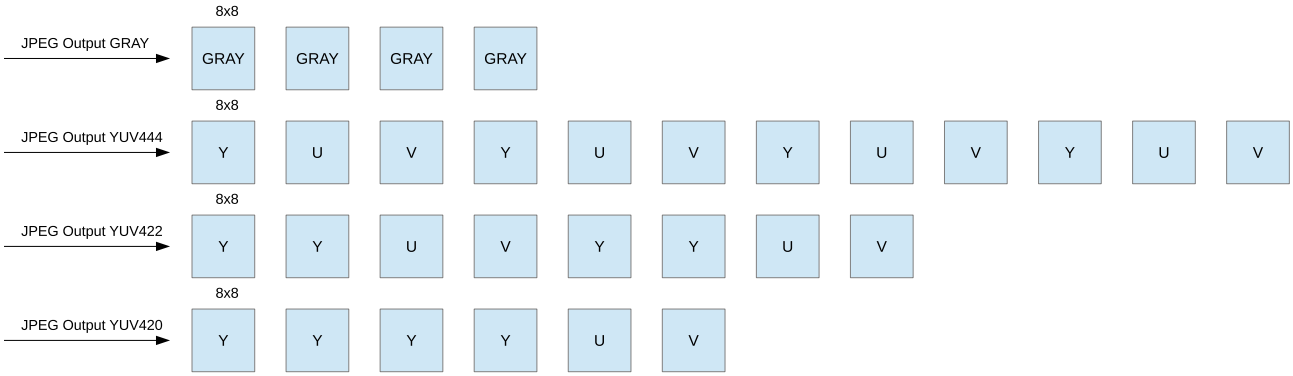


Figure 4.4-4. Macroblock Layout of JPEG Decoder Output Data

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 to YUV444 to RGB888/RGB565
- 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.

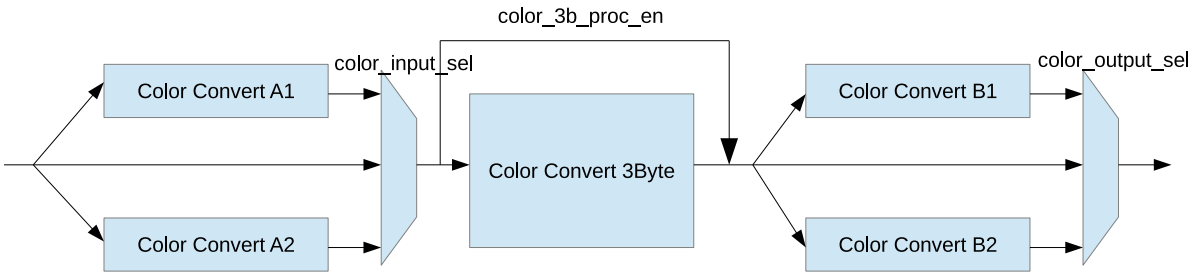


Figure 4.4-5. Color Space Conversion

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
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
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. Ususally, 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 7 *External Memory Encryption and Decryption [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 ~ 3 (in total 4 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-P4 supports the Event Task Matrix (ETM) function, which allows 2D-DMA's ETM tasks to be triggered by any peripherals' ETM events, or 2D-DMA's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to 2D-DMA. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

2D-DMA can receive the following ETM tasks:

- DMA2D_TASK_IN_START_CH*n*: Enables the corresponding receive channel *n* for data transfer.
- DMA2D_TASK_OUT_START_CH*n*: Enables the corresponding transmit channel *n* for data transfer.

Note:
Above ETM tasks can achieve the same functions as CPU configuring DMA2D_INLINK_START_CH*n* and DMA2D_OUTLINK_START_CH*n*. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 1, only ETM tasks can be used to configure the transfer direction and enable the corresponding 2D-DMA channel. When DMA2D_IN_ETM_EN_CH*n* or DMA2D_OUT_ETM_EN_CH*n* is 0, only CPU can be used to enable the corresponding 2D-DMA channel.

- DMA2D_TASK_IN_DSCR_READY_CH*n*: Allows receive channel *n* to process the next descriptor.
- DMA2D_TASK_OUT_DSCR_READY_CH*n*: Allows transmit channel *n* to process the next descriptor.

Note:
In normal conditions, after finishing with the current descriptor if the next descriptor is available (i.e., the next descriptor address field is not zero), the 2D-DMA would process the next descriptor directly and reads/writes to the memory according to the descriptor's configuration until all descriptors are processed. When DMA2D_IN_ETM_LOOP_EN_CH*n* or DMA2D_OUT_ETM_LOOP_EN_CH*n* is set, after finishing with the current descriptor, the 2D-DMA would not start processing the next descriptor until receiving the DMA2D_TASK_IN_DSCR_READY_CH*n* or

DMA2D_TASK_OUT_DSCR_READY_CH*n* task.
The DMA2D_TASK_IN_DSCR_READY_CH*n* or DMA2D_TASK_OUT_DSCR_READY_CH*n* task can be stored in the task buffer, which can cache at most four tasks. When the 2D-DMA receives this task but the current descriptor is not yet processed, it can cache the task. The depth of the task buffer is determined by DMA2D_IN_DSCR_TASK_MAX_CH*n* or DMA2D_OUT_DSCR_TASK_MAX_CH*n*.
When the task buffer exceeds its maximum capacity, it will trigger an DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT or DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT interrupt.

2D-DMA can generate the following ETM events:

- DMA2D_EVT_IN_DONE_CH*n*: Indicates that the data has been received according to the receive descriptor via channel *n*.
- DMA2D_EVT_IN_SUC_EOF_CH*n*: Indicates that the data corresponding to a receive descriptor has been received via channel *n* and the Eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_DONE_CH*n*: Indicates that the data has been transmitted according to the transmit descriptor via channel *n*.
- DMA2D_EVT_OUT_SUC_EOF_CH*n*: Indicates that the data corresponding to a transmit descriptor has been transmitted or received via channel *n* and the eof bit of this descriptor is 1.
- DMA2D_EVT_OUT_TOTAL_EOF_CH*n*: Indicates that the data corresponding to the last transmit descriptors has been sent via transmit channel *n* and the eof bit of this descriptor is 1.

In practical applications, 2D-DMA's ETM events can trigger its own ETM tasks. For example, the DMA2D_EVT_OUT_TOTAL_EOF_CH0 event can trigger the DMA2D_TASK_IN_START_CH1 task, and in this way trigger a new round of 2D-DMA operations.

4.6 Interrupts

ESP32-P4's 2D-DMA module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- DMA2D_IN_CH0_INTR
- DMA2D_IN_CH1_INTR
- DMA2D_OUT_CH0_INTR
- DMA2D_OUT_CH1_INTR
- DMA2D_OUT_CH2_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 ~ 1) 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 ~ 2) 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 28 *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 29 *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 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
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

GoBack

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_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

Name	Description	Address	Access
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
DMA2D_IN_ETM_CONF_CH1_REG	Configures the ETM of RX channel 1	0x0648	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

Name	Description	Address	Access
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_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
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_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

4.9 Registers

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

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

Register 4.1. DMA2D_OUT_CONF0_CHn_REG (n: 0-2) (0x0000+0x100*n)

Bits	Field	Reset
31:27	(reserved)	0 0 0 0 0
26	DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CHn	0
25	DMA2D_OUT_CMD_DISABLE_CHn	0
24	DMA2D_OUT_RST_CHn	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_CHn	0
10:9	DMA2D_OUT_MACRO_BLOCK_SIZE_CHn	0x0
8:6	DMA2D_OUT_MEM_BURST_LENGTH_CHn	0x0
5	(reserved)	0
4	DMA2D_OUT_CHECK_OWNER_CHn	0
3	DMA2D_OUT_ECC_AES_EN_CHn	0
2	DMA2D_OUTDSCR_BURST_EN_CHn	0
1	DMA2D_OUT_EOF_MODE_CHn	1
0	DMA2D_OUT_AUTO_WRBACK_CHn	0

DMA2D_OUT_AUTO_WRBACK_CHn 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_CHn 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_CHn Configures whether to enable INCR burst transfer for TX channel *n* reading descriptors.

0: Disable

1: Enable

(R/W)

DMA2D_OUT_ECC_AES_EN_CHn 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*: 0-2) (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*: 0-2) (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)

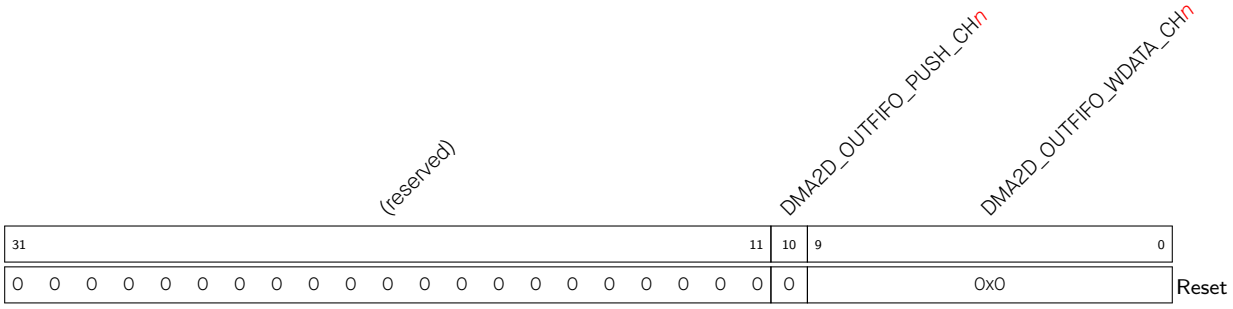
DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH*n* Configures whether to enable weight optimization for TX channel *n*.

0: Enable

1: Disable

(R/W)

Register 4.2. DMA2D_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x0018+0x100**n*)



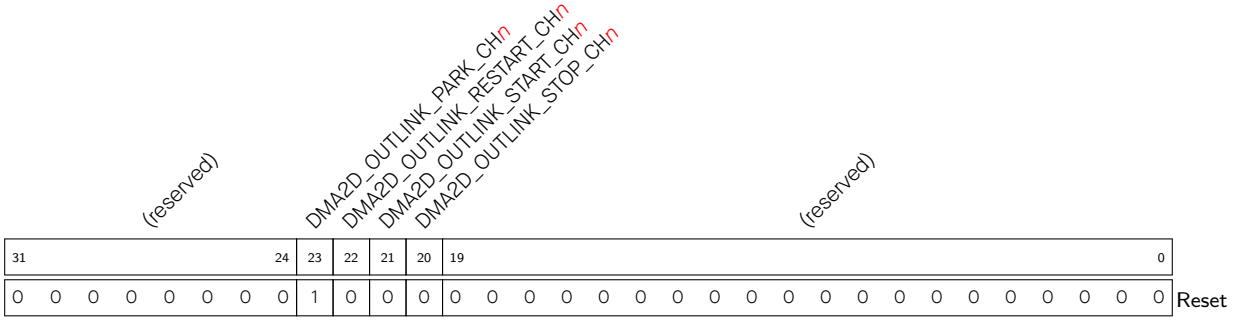
DMA2D_OUTFIFO_WDATA_CH*n* Configures the data to be pushed into the 2D-DMA TX FIFO. (R/W)

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*: 0-2) (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)

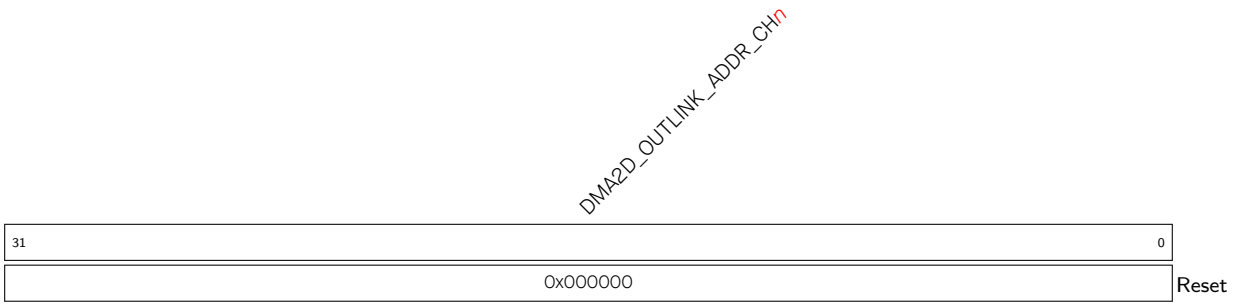
DMA2D_OUTLINK_PARK_CH*n* Represents the status of the transmit descriptor's FSM.

0: Running

1: Idle

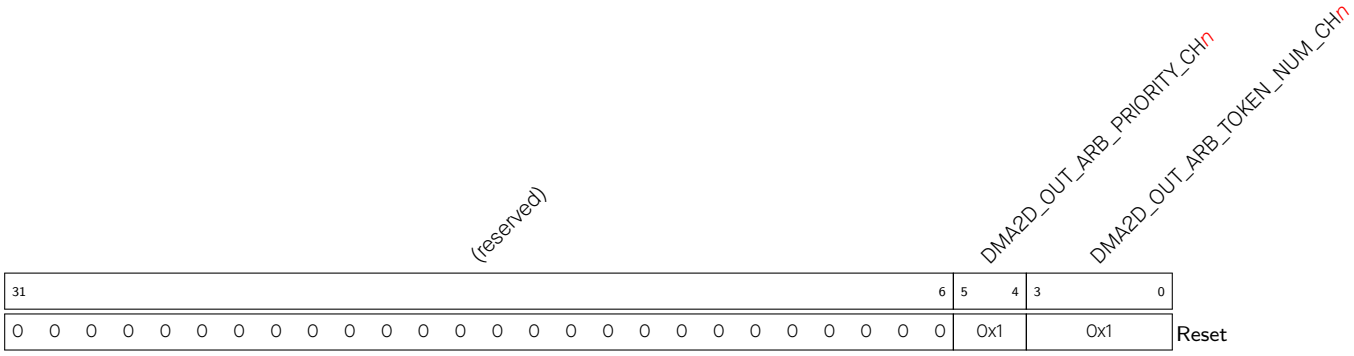
(RO)

Register 4.4. DMA2D_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-2) (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*: 0-2) (0x003C+0x100**n*)



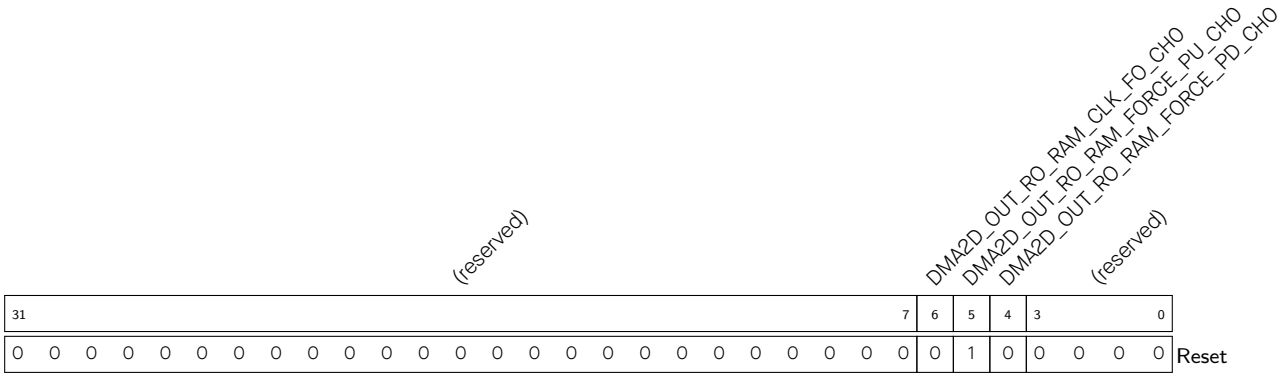
DMA2D_OUT_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of TX channel *n*.

Value range: 0 ~ 15. (R/W)

DMA2D_OUT_ARB_PRIORITY_CH*n* Configures the priority of TX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 3. (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)

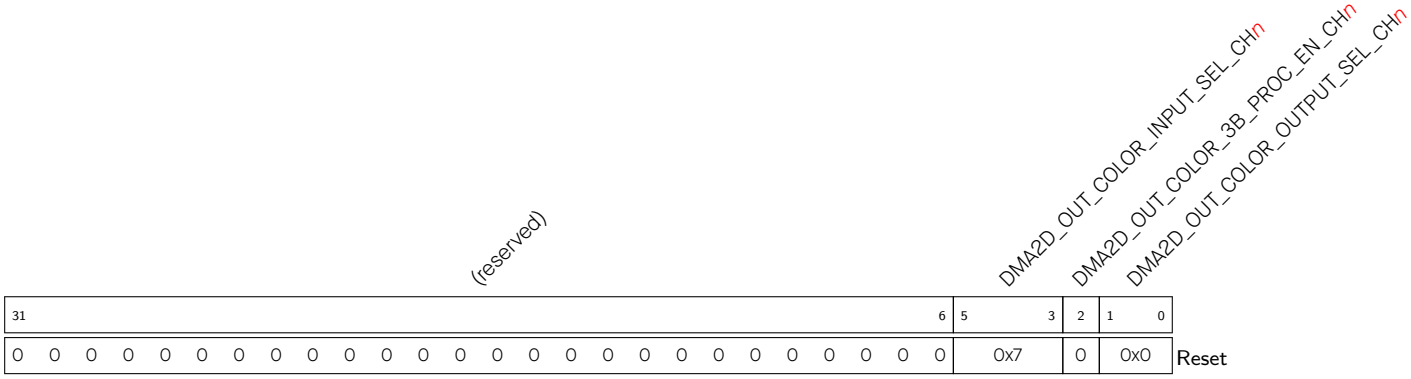
DMA2D_OUT_RO_RAM_CLK_FO_CH0 Configures the clock gating when TX channel 0 accessing the 2D-DMA RAM.

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*: 0-2) (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

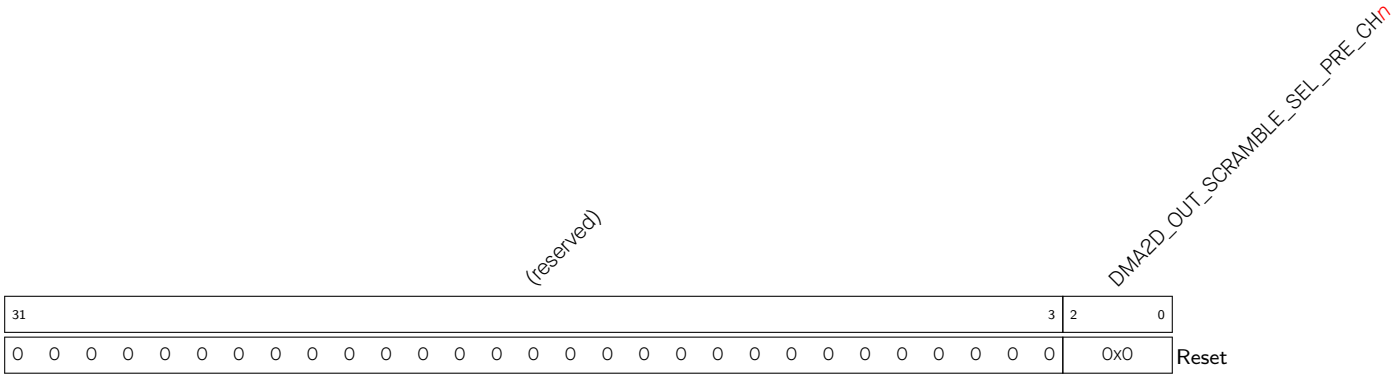
3: Other 3 bytes/pixel color format

7: Disable color space conversion

Others: Invalid

(R/W)

Register 4.8. DMA2D_OUT_SCRAMBLE_CH*n*_REG (*n*: 0-2) (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

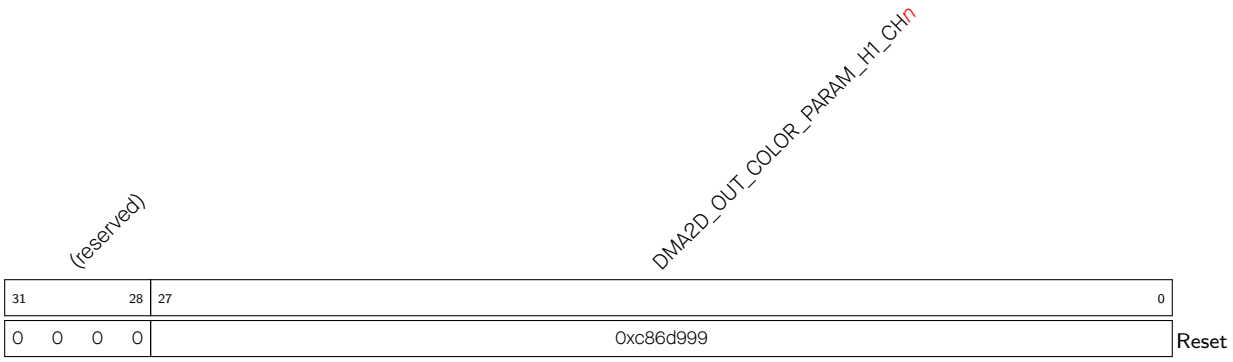
(R/W)

Register 4.9. DMA2D_OUT_COLOR_PARAM0_CH*n*_REG (*n*: 0-2) (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*: 0-2) (0x0054+0x100**n*)



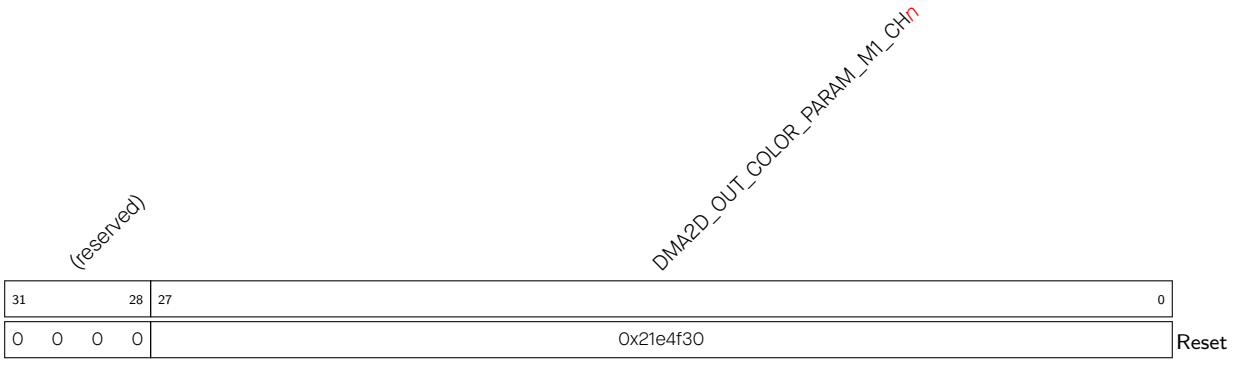
DMA2D_OUT_COLOR_PARAM_H1_CH*n* Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.11. DMA2D_OUT_COLOR_PARAM2_CH*n*_REG (*n*: 0-2) (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*: 0-2) (0x005C+0x100**n*)



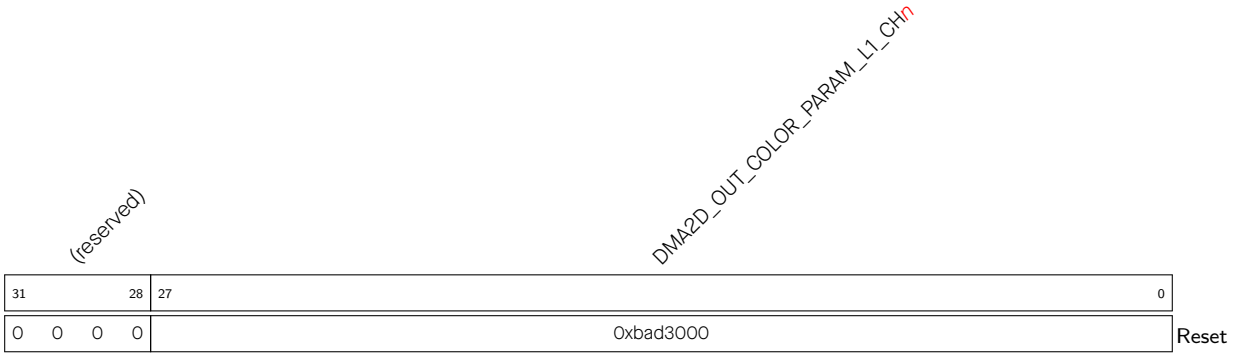
DMA2D_OUT_COLOR_PARAM_M1_CH*n* Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.13. DMA2D_OUT_COLOR_PARAM4_CH*n*_REG (*n*: 0-2) (0x0060+0x100n*)**



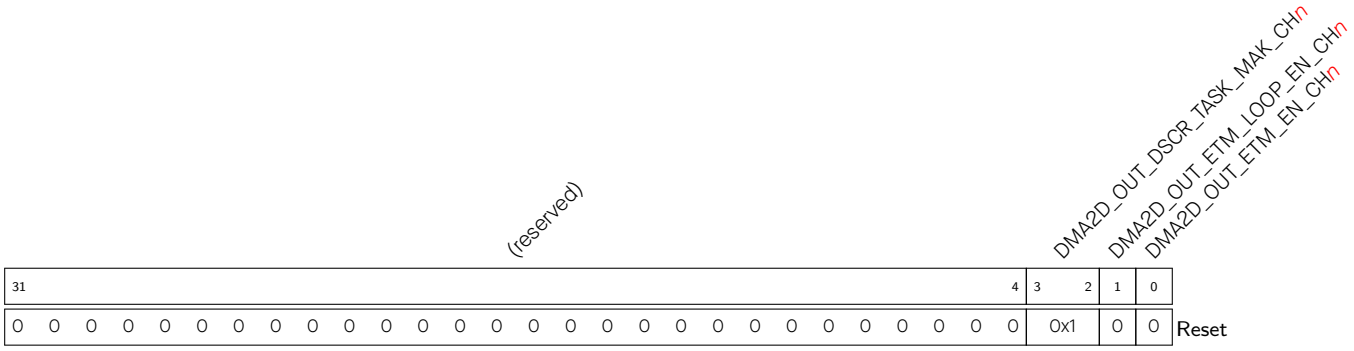
DMA2D_OUT_COLOR_PARAM_LO_CH*n* Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for TX channel *n*. (R/W)

Register 4.14. DMA2D_OUT_COLOR_PARAM5_CH*n*_REG (*n*: 0-2) (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*: 0-2) (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*.

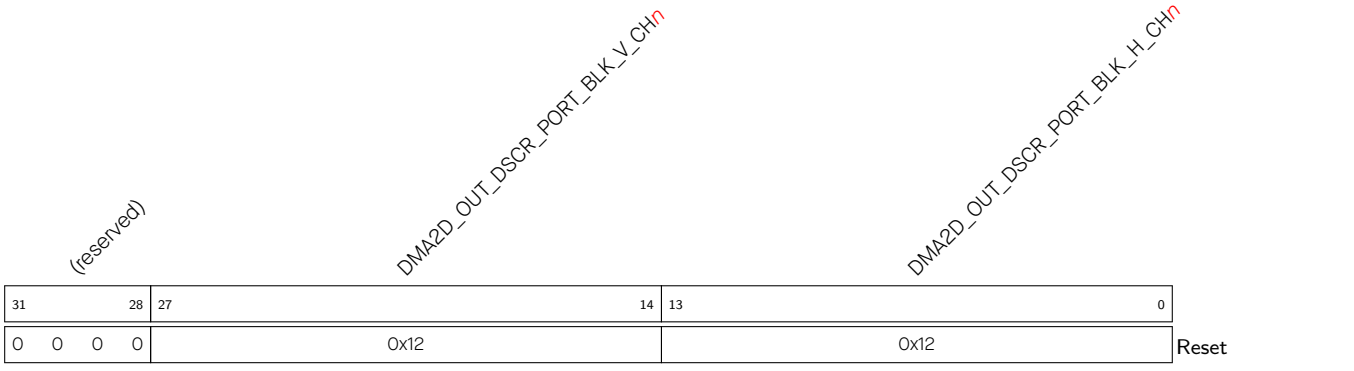
0: Not use ETM task

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*: 0-2) (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_CHn_REG (n: 0-1) (0x0500+0x100*n)

Bits	Field	Reset
31:27	(reserved)	0 0 0 0 0
26	DMA2D_IN_ARB_WEIGHT_OPT_DIS_CHn	0
25	DMA2D_IN_CMD_DISABLE_CHn	0
24	DMA2D_IN_RST_CHn	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_CHn	0
10:9	DMA2D_IN_MACRO_BLOCK_SIZE_CHn	0x0
8:6	DMA2D_IN_MEM_BURST_LENGTH_CHn	0x0
5	reserved	0
4	DMA2D_IN_CHECK_OWNER_CHn	0
3	DMA2D_IN_ECC_AES_EN_CHn	0
2	DMA2D_INDSCR_BURST_EN_CHn	0
1	(reserved)	0
0	DMA2D_IN_MEM_TRANS_EN_CHn	0

DMA2D_IN_MEM_TRANS_EN_CHn 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_CHn 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_CHn 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_CHn Configures whether to enable the owner bit check for RX channel n.
0: Disable
1: Enable
(R/W)

DMA2D_IN_MEM_BURST_LENGTH_CHn 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*: 0-1) (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)

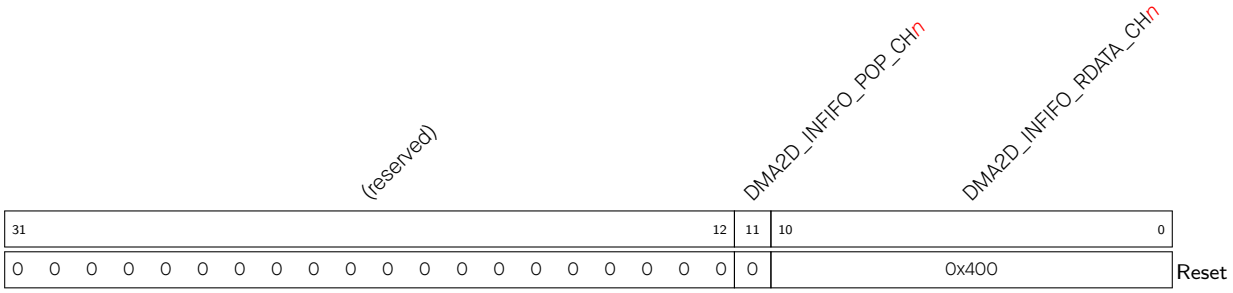
DMA2D_IN_REORDER_EN_CH0 Configures whether to enable macroblock reordering for RX channel 0.
0: Disable
1: Enable
(R/W)

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*: 0-1) (0x0518+0x100n*)**



DMA2D_INFIFO_RDATA_CH*n* Represents the data popped from 2D-DMA RX FIFO. (RO)

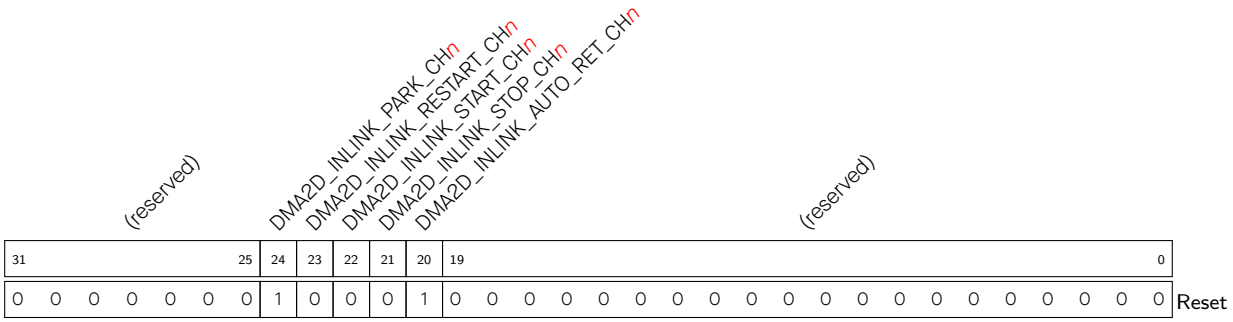
DMA2D_INFIFO_POP_CH*n* Configures whether to pop data from 2D-DMA RX FIFO.

0: Not pop

1: Pop

(R/W/SC)

Register 4.19. DMA2D_IN_LINK_CONF_CH*n*_REG (*n*: 0-1) (0x051C+0x100**n*)



DMA2D_INLINK_AUTO_RET_CH*n* Configure the value of the owner bit written back to the receive descriptor.

0: Write back 0

1: Write back 1

(R/W)

DMA2D_INLINK_STOP_CH*n* Configures whether to stop RX channel *n* from receiving data.

0: Invalid. No effect

1: Stop

(R/W/SC)

DMA2D_INLINK_START_CH*n* Configures whether to enable RX channel *n* for data transfer.

0: Disable

1: Enable

(R/W/SC)

DMA2D_INLINK_RESTART_CH*n* Configures whether to restart RX channel *n* for 2D-DMA transfer.

0: Invalid. No effect

1: Restart

(R/W/SC)

DMA2D_INLINK_PARK_CH*n* Represents the status of the receive descriptor's FSM.

0: Running

1: Idle

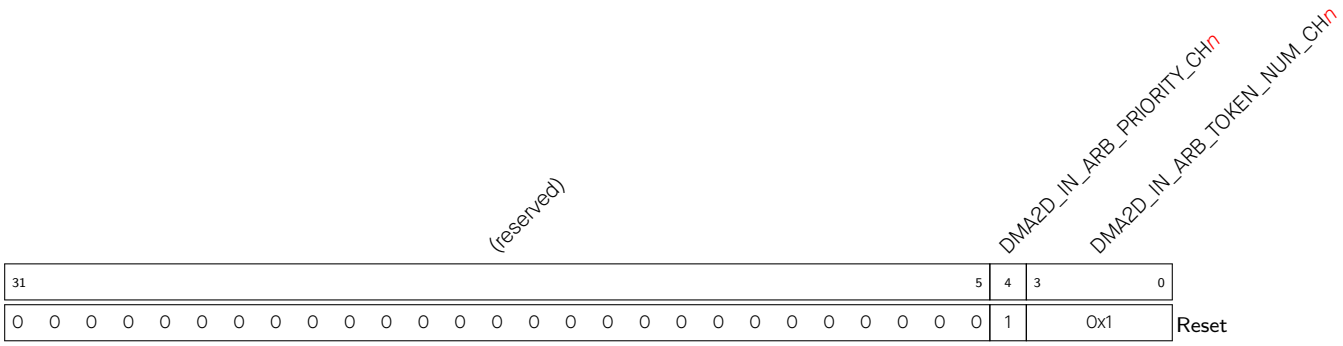
(RO)

Register 4.20. DMA2D_IN_LINK_ADDR_CH*n*_REG (*n*: 0-1) (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*: 0-1) (0x0540+0x100n*)**



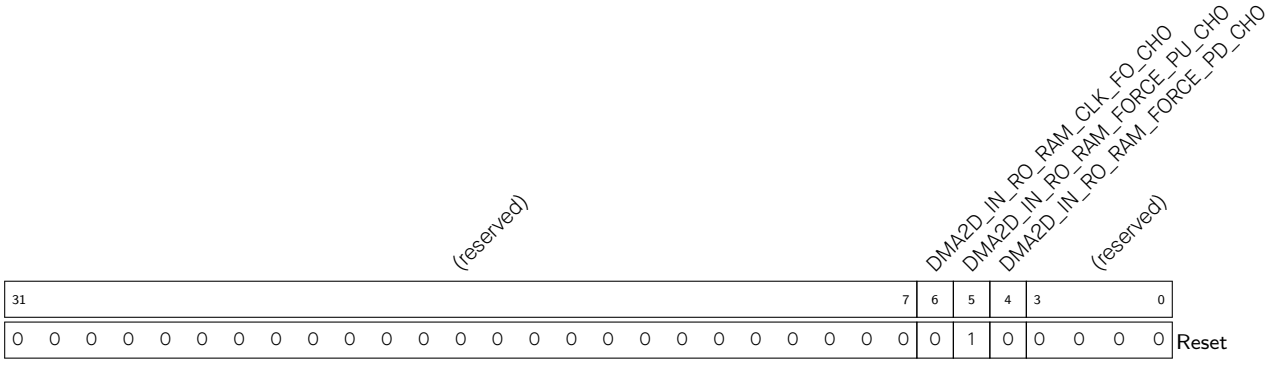
DMA2D_IN_ARB_TOKEN_NUM_CH*n* Configures the weight (i.e the number of tokens) of RX channel *n*.

Value range: 0 ~ 15. (R/W)

DMA2D_IN_ARB_PRIORITY_CH*n* Configures the priority of RX channel *n*. The larger the value, the higher the priority.

Value range: 0 ~ 3. (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)

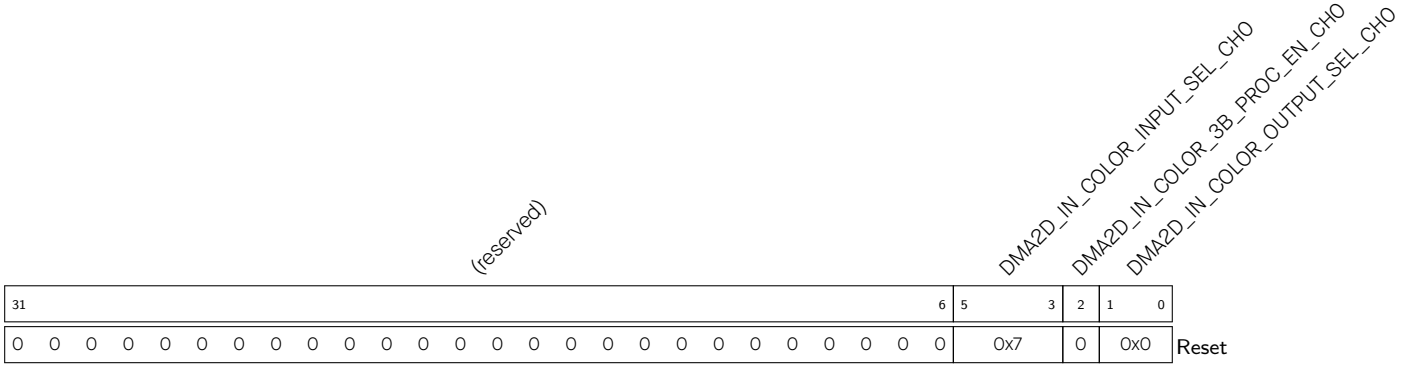
DMA2D_IN_RO_RAM_CLK_FO_CH0 Configures the clock gating when RX channel 0 accessing the 2D-DMA RAM.

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

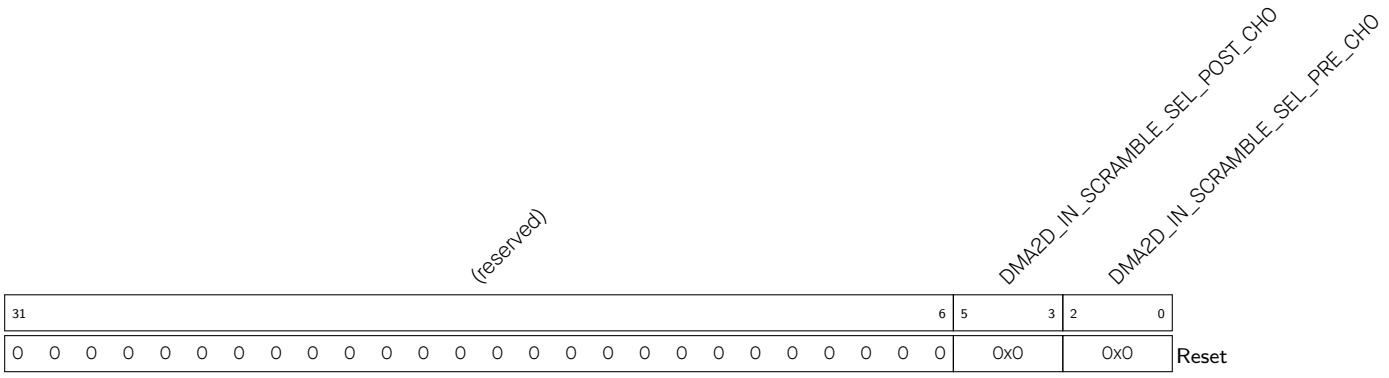
2: YUV444/420

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

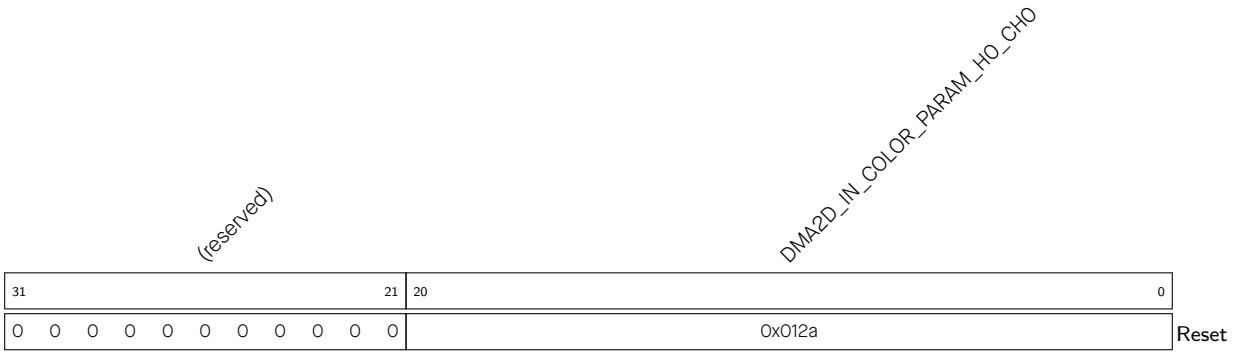
4: BYTE0-2-1

5: BYTE0-1-2

Others: Invalid

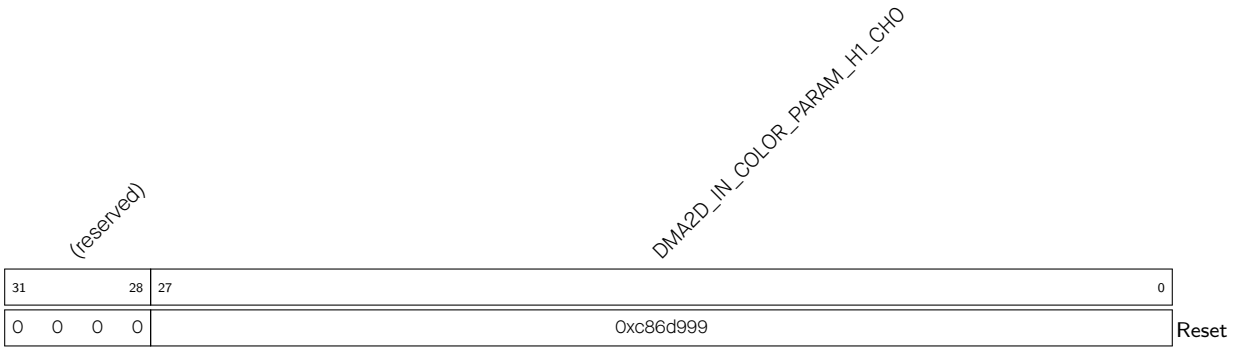
(R/W)

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)



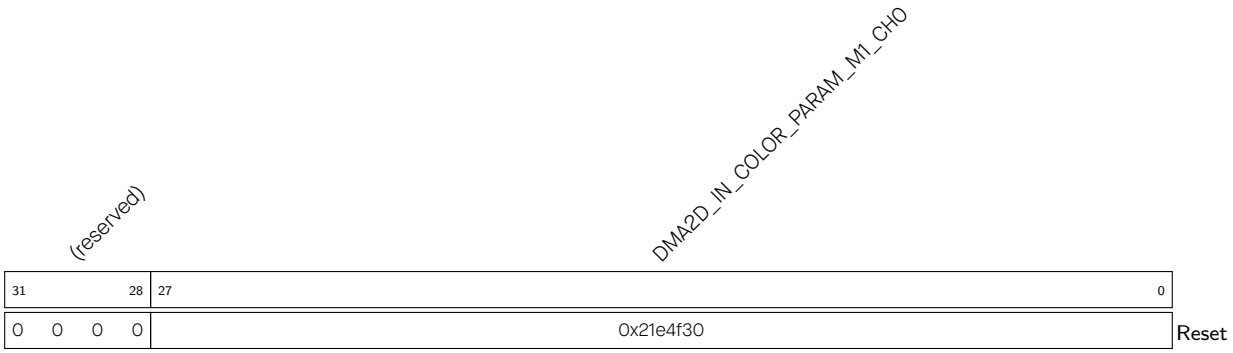
DMA2D_IN_COLOR_PARAM_H1_CH0 Configures the coefficient C and D for the most significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.27. DMA2D_IN_COLOR_PARAM2_CH0_REG (0x055C)



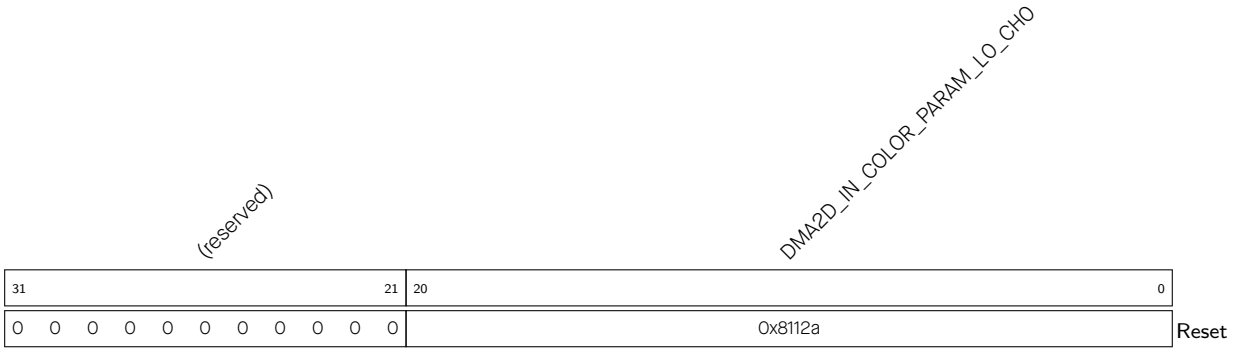
DMA2D_IN_COLOR_PARAM_M0_CH0 Configures the coefficient A and B for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.28. DMA2D_IN_COLOR_PARAM3_CH0_REG (0x0560)



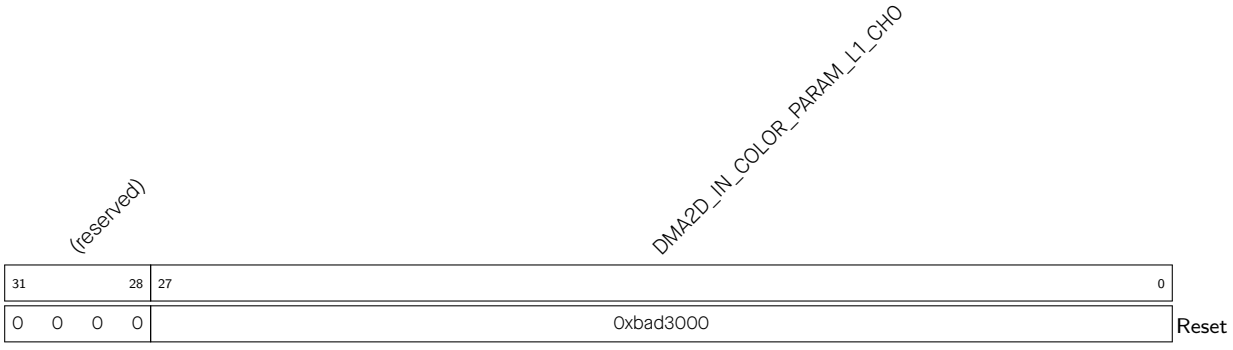
DMA2D_IN_COLOR_PARAM_M1_CH0 Configures the coefficient C and D for the middle byte of the input of color space conversion's second stage for RX channel 0. (R/W)

Register 4.29. DMA2D_IN_COLOR_PARAM4_CH0_REG (0x0564)



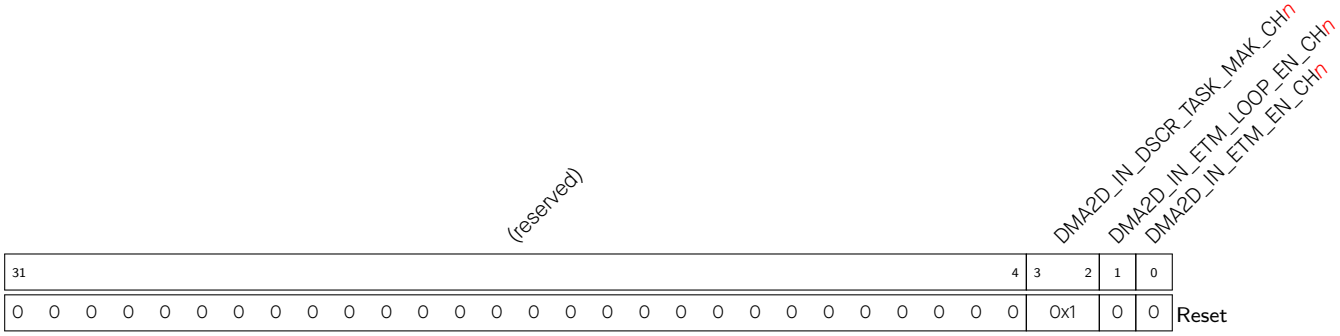
DMA2D_IN_COLOR_PARAM_L0_CH0 Configures the coefficient A and B for the least significant byte of the input of color space conversion's second stage for RX channel 0. (R/W)

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*: 0-1) (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*.

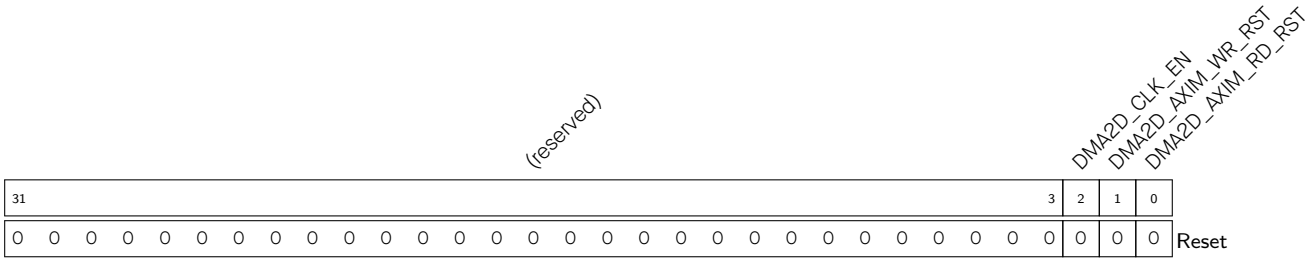
0: Not use ETM task

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

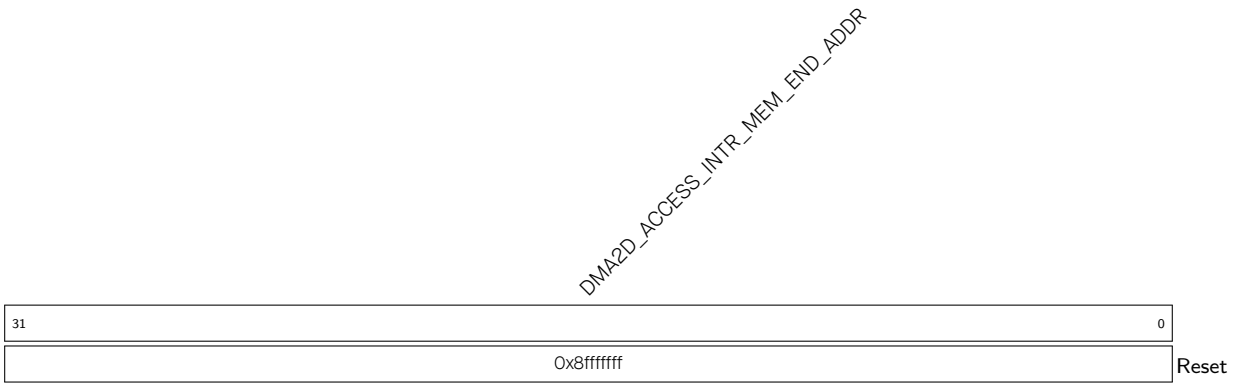
(R/W)

Register 4.33. DMA2D_INTR_MEM_START_ADDR_REG (0x0A08)



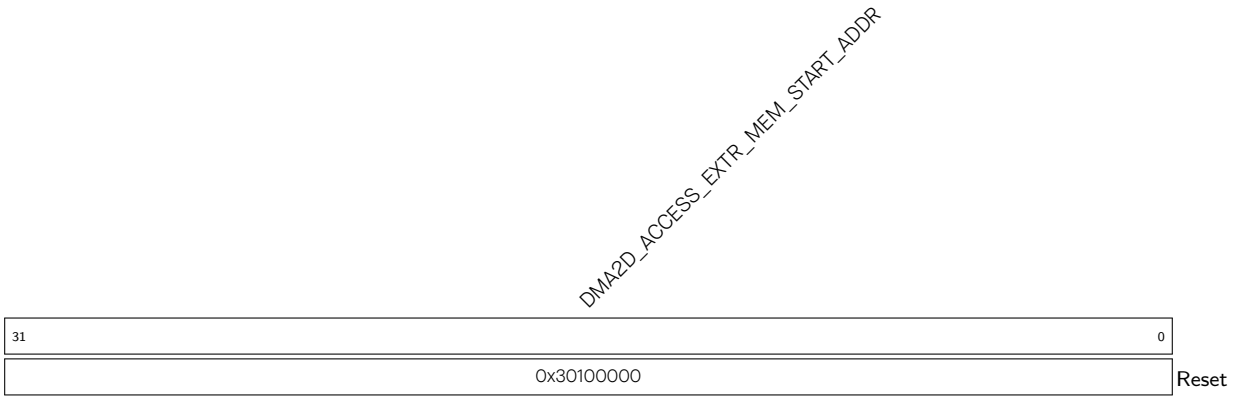
DMA2D_ACCESS_INTR_MEM_START_ADDR Configures the start address of the accessible internal address space. (R/W)

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)



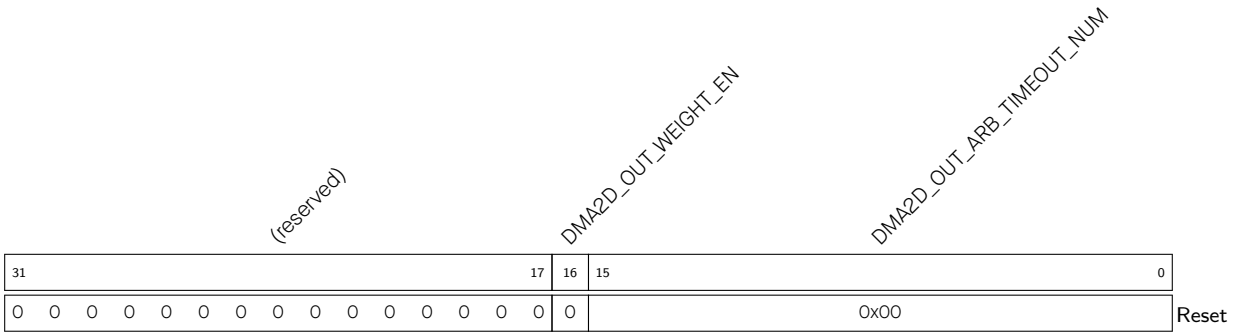
DMA2D_ACCESS_EXTR_MEM_START_ADDR Configures the start address of the accessible external address space. (R/W)

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)

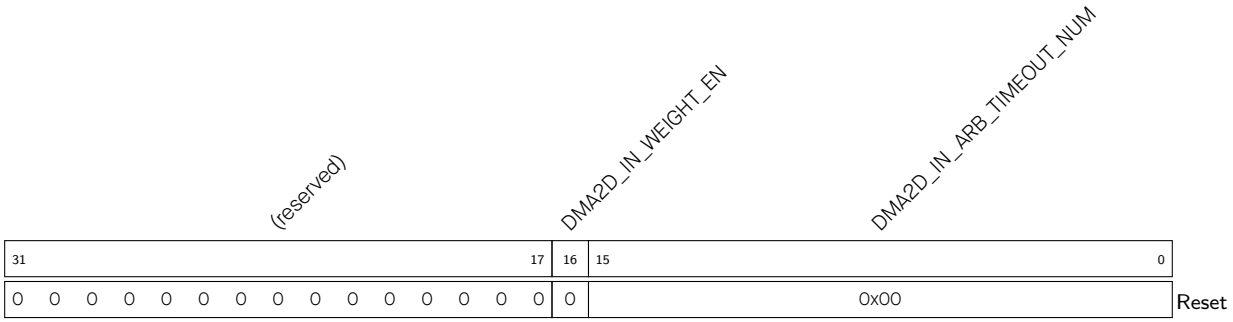
DMA2D_OUT_WEIGHT_EN Configures whether to enable weight arbitration for TX.

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)

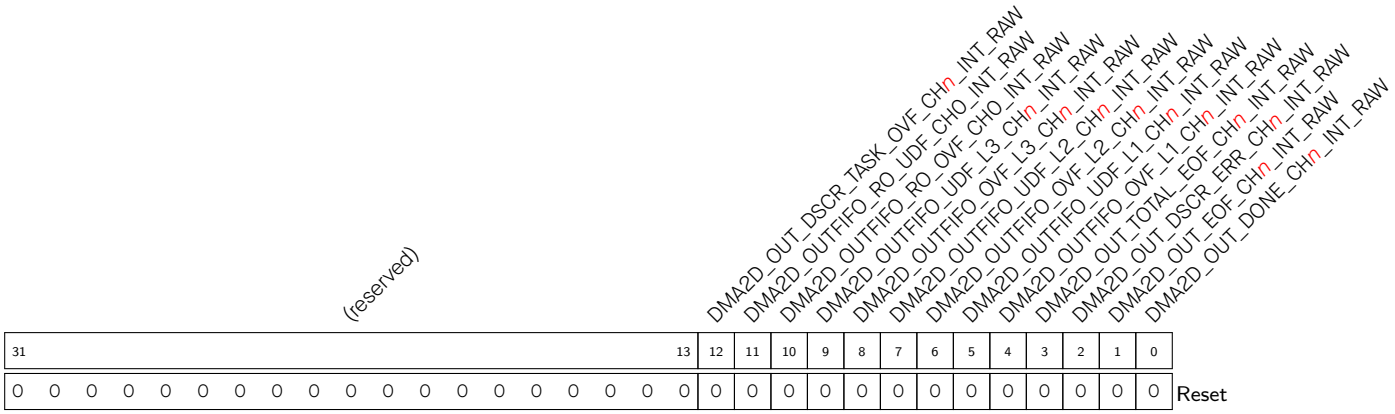
DMA2D_IN_WEIGHT_EN Configures whether to enable weight arbitration for RX.

0: Disable

1: Enable

(R/W)

Register 4.39. DMA2D_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (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)

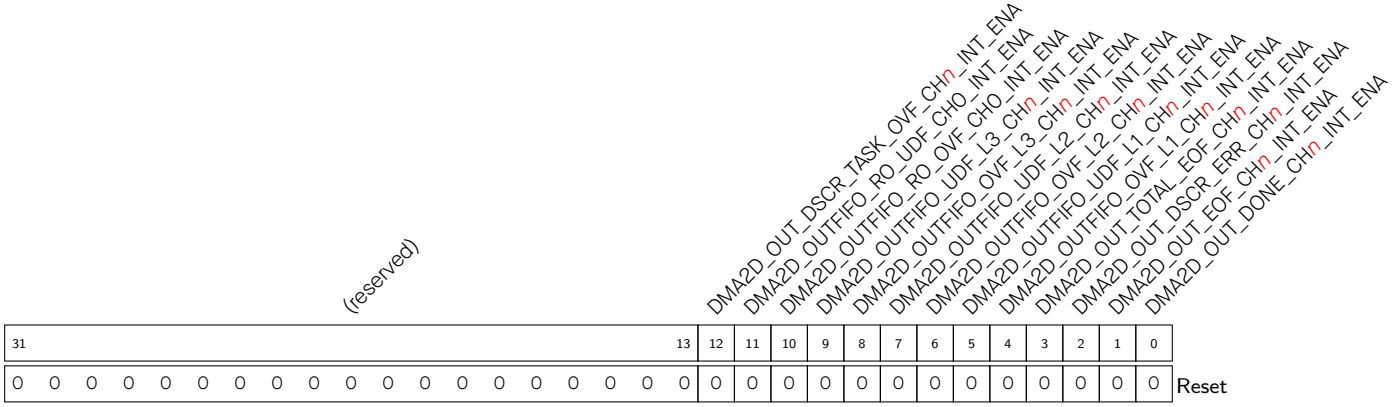
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_OVF_CH0_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_OVF_CH0_INT. (R/WTC/SS)

DMA2D_OUTFIFO_RO_UDF_CH0_INT_RAW The raw interrupt status of DMA2D_OUTFIFO_RO_UDF_CH0_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*: 0-2) (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)

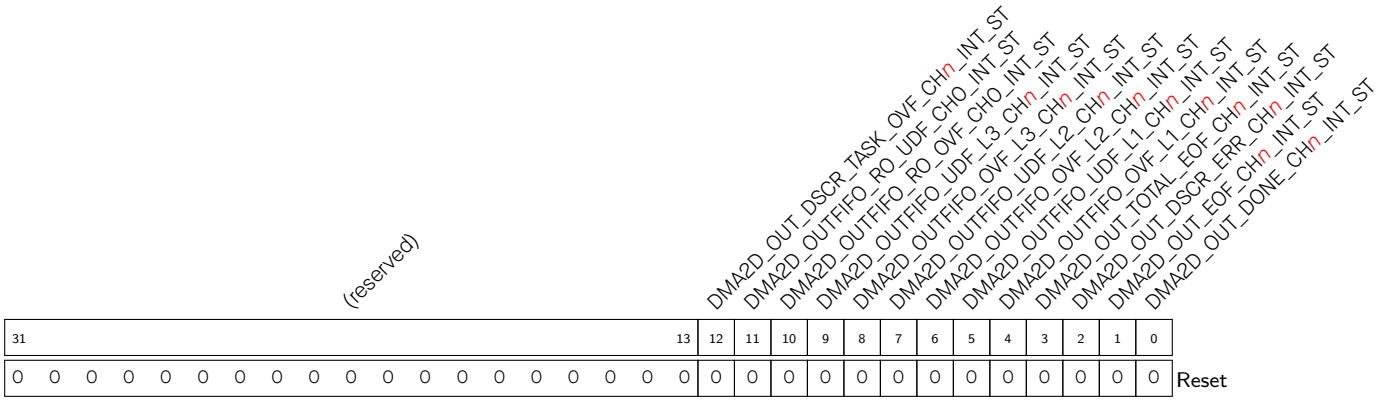
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ENA Write 1 to enable the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (R/W)

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*: 0-2) (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)

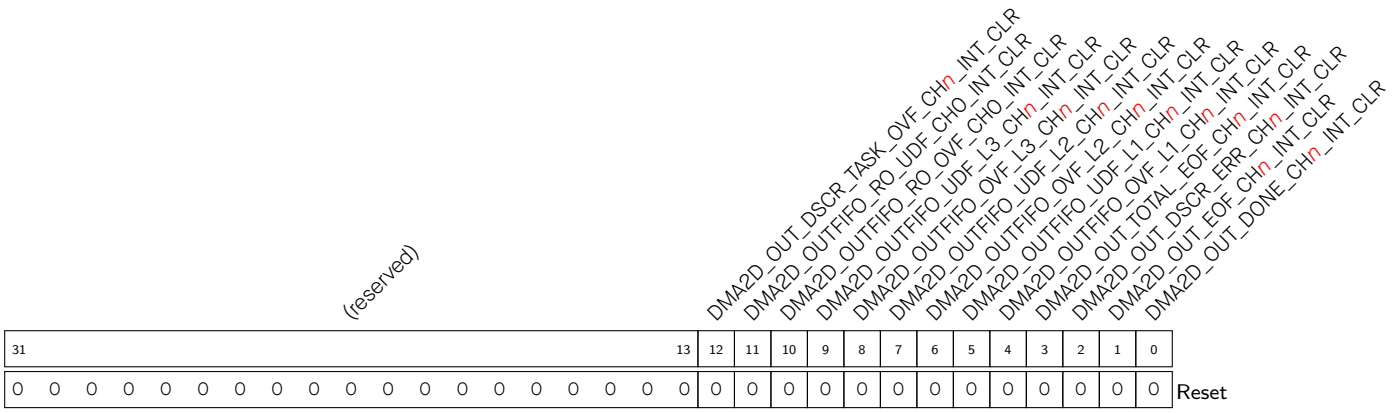
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ST The masked interrupt status of DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (RO)

DMA2D_OUTFIFO_RO_OVF_CH0_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_OVF_CH0_INT. (RO)

DMA2D_OUTFIFO_RO_UDF_CH0_INT_ST The masked interrupt status of DMA2D_OUTFIFO_RO_UDF_CH0_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*: 0-2) (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)

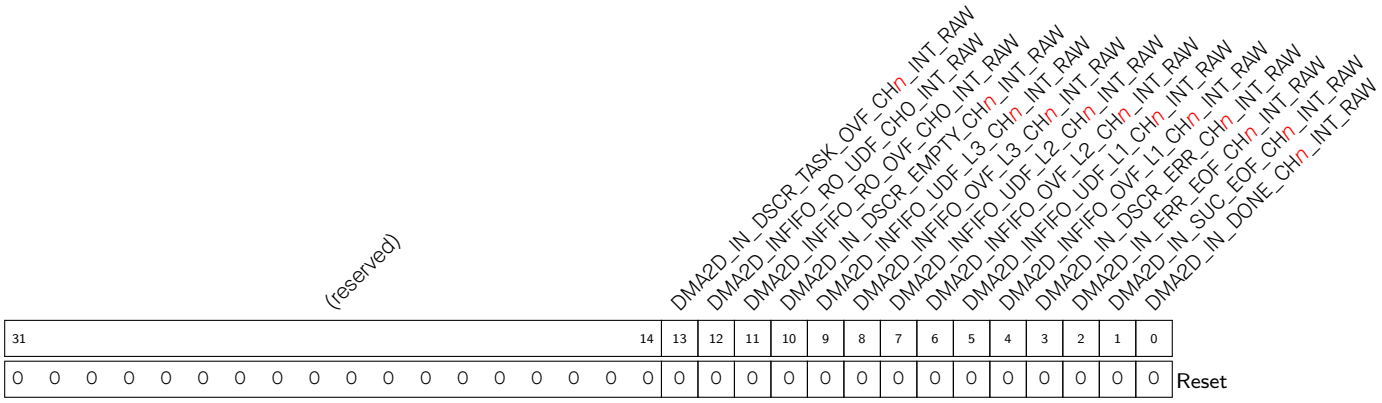
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_CLR Write 1 to clear the DMA2D_OUTFIFO_UDF_L3_CH*n*_INT. (WT)

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*: 0-1) (0x0504+0x100**n*)



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)

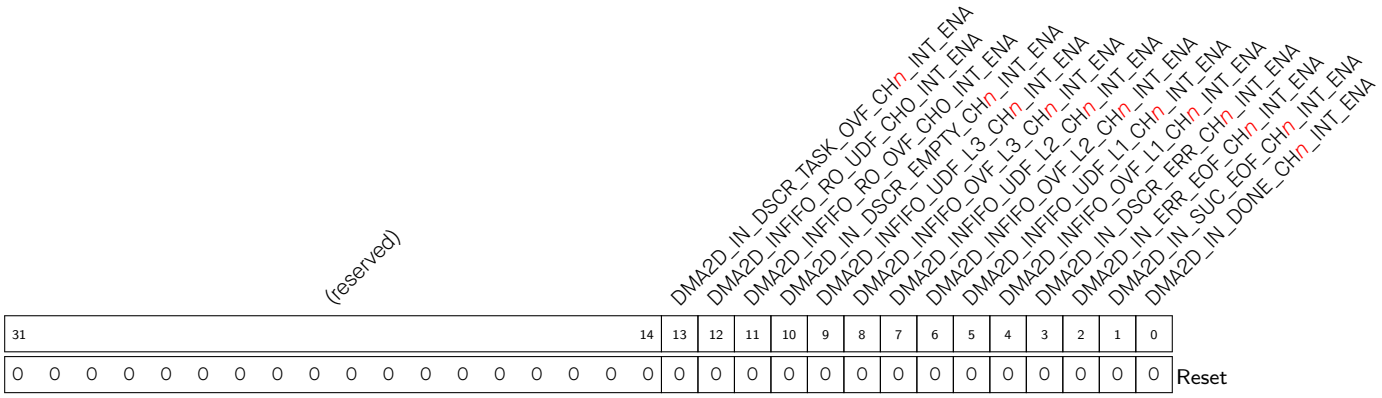
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_RAW The raw interrupt status of DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_OVF_CH0_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_OVF_CH0_INT. (R/WTC/SS)

DMA2D_INFIFO_RO_UDF_CH0_INT_RAW The raw interrupt status of DMA2D_INFIFO_RO_UDF_CH0_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*: 0-1) (0x0508+0x100n*)**



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)

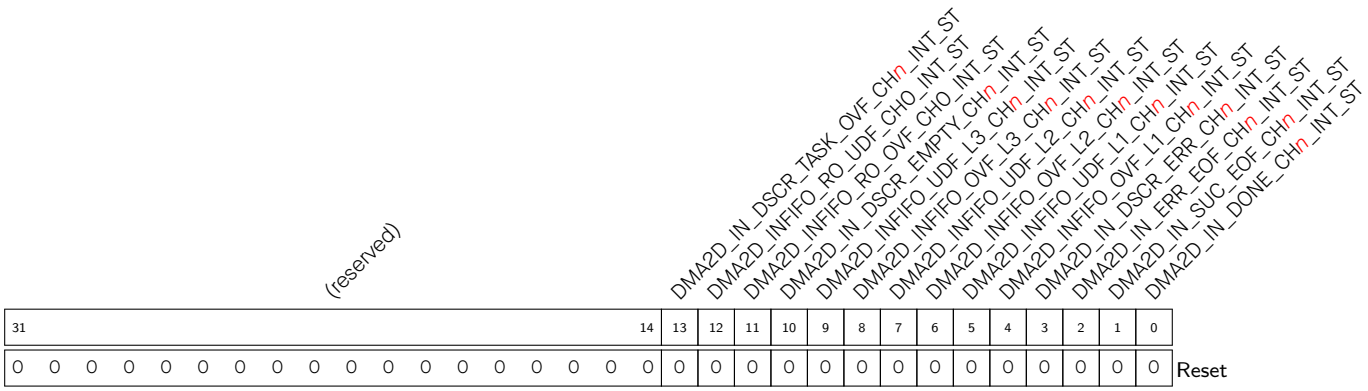
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ENA Write 1 to enable the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (R/W)

DMA2D_INFIFO_RO_OVF_CHO_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_OVF_CHO_INT. (R/W)

DMA2D_INFIFO_RO_UDF_CHO_INT_ENA Write 1 to enable the DMA2D_INFIFO_RO_UDF_CHO_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*: 0-1) (0x050C+0x100n*)**



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)

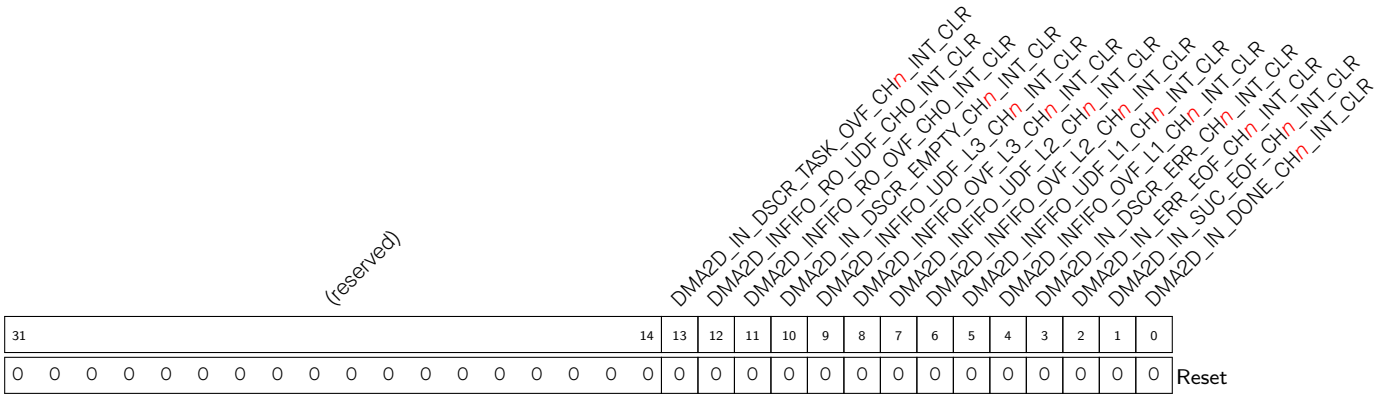
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ST The masked interrupt status of the DMA2D_IN_DSCR_EMPTY_CH*n*_INT. (RO)

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_CHn_REG (n: 0-1) (0x0510+0x100*n)



DMA2D_IN_DONE_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DONE_CHn_INT. (WT)

DMA2D_IN_SUC_EOF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_SUC_EOF_CHn_INT. (WT)

DMA2D_IN_ERR_EOF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_ERR_EOF_CHn_INT. (WT)

DMA2D_IN_DSCR_ERR_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_ERR_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L1_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L1_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L1_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L1_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L2_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L2_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L2_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L2_CHn_INT. (WT)

DMA2D_INFIFO_OVF_L3_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_OVF_L3_CHn_INT. (WT)

DMA2D_INFIFO_UDF_L3_CHn_INT_CLR Write 1 to clear the DMA2D_INFIFO_UDF_L3_CHn_INT. (WT)

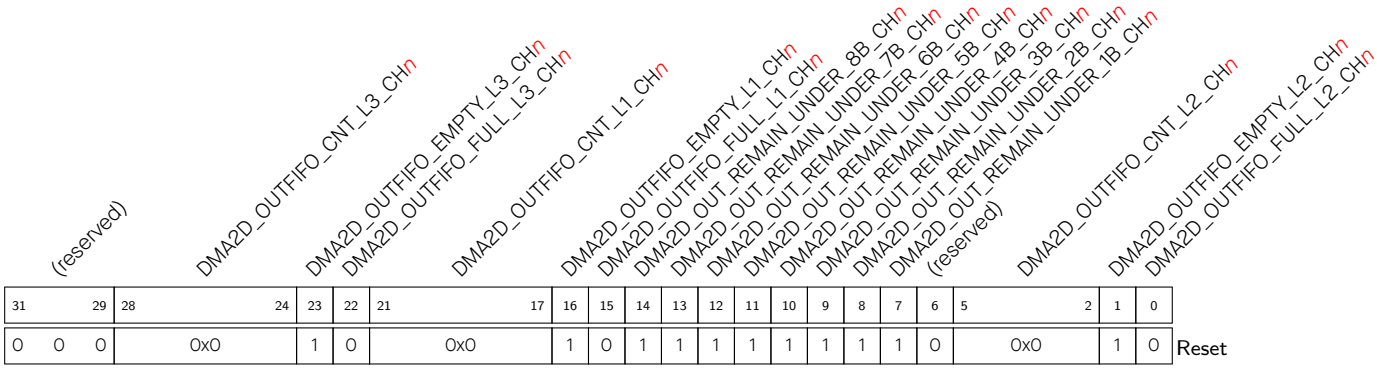
DMA2D_IN_DSCR_EMPTY_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_EMPTY_CHn_INT. (WT)

DMA2D_INFIFO_RO_OVF_CHO_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_OVF_CHO_INT. (WT)

DMA2D_INFIFO_RO_UDF_CHO_INT_CLR Write 1 to clear the DMA2D_INFIFO_RO_UDF_CHO_INT. (WT)

DMA2D_IN_DSCR_TASK_OVF_CHn_INT_CLR Write 1 to clear the DMA2D_IN_DSCR_TASK_OVF_CHn_INT. (WT)

Register 4.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0014+0x100**n*)



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*: 0-2) (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.

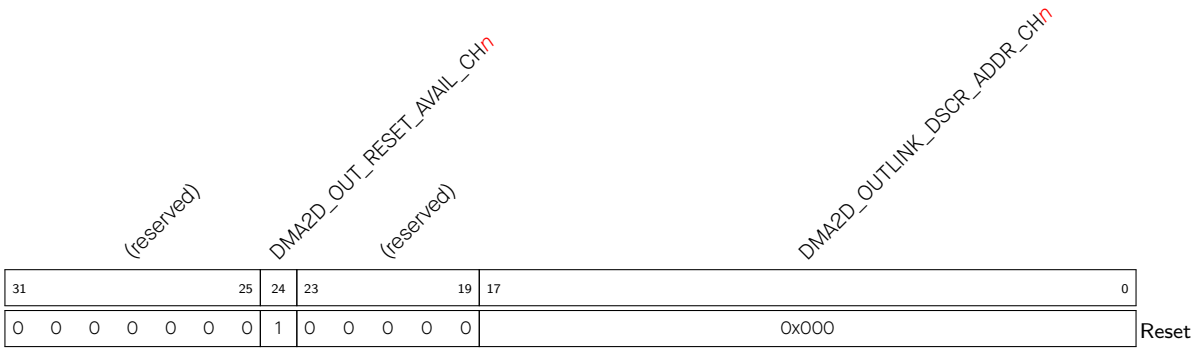
0: Not empty

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*: 0-2) (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)

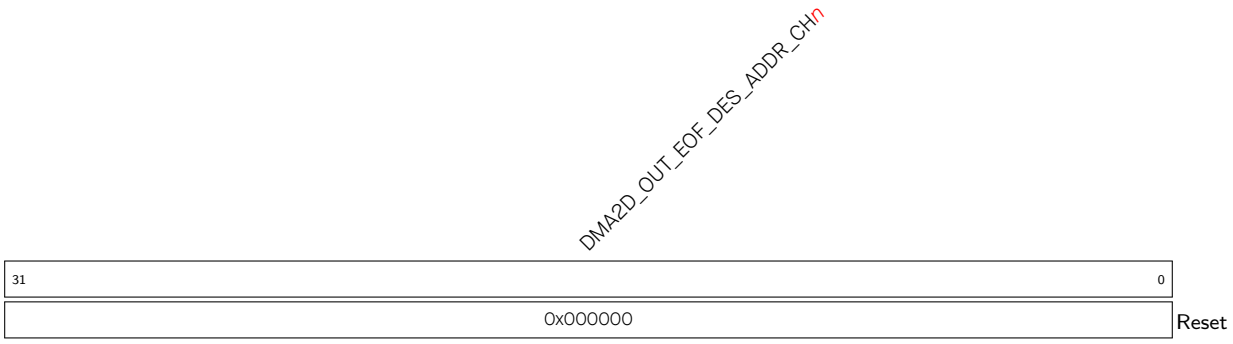
DMA2D_OUT_RESET_AVAIL_CH*n* Represents whether it is safe to reset TX channel *n*.

0: Not safe

1: Safe

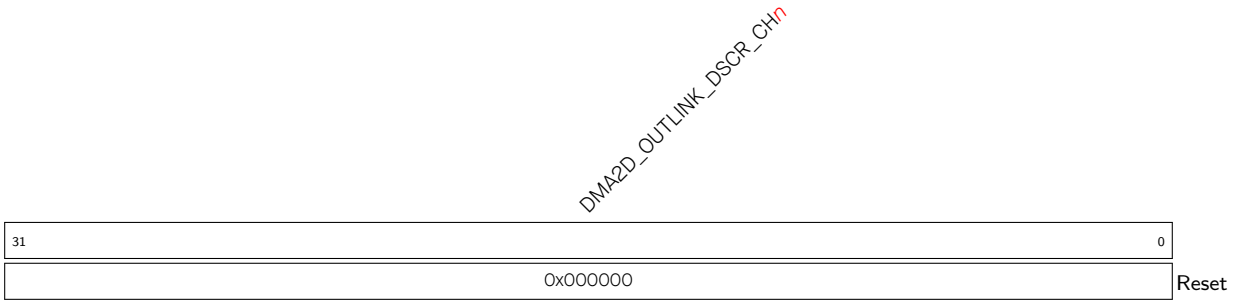
(RO)

Register 4.49. DMA2D_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0028+0x100**n*)



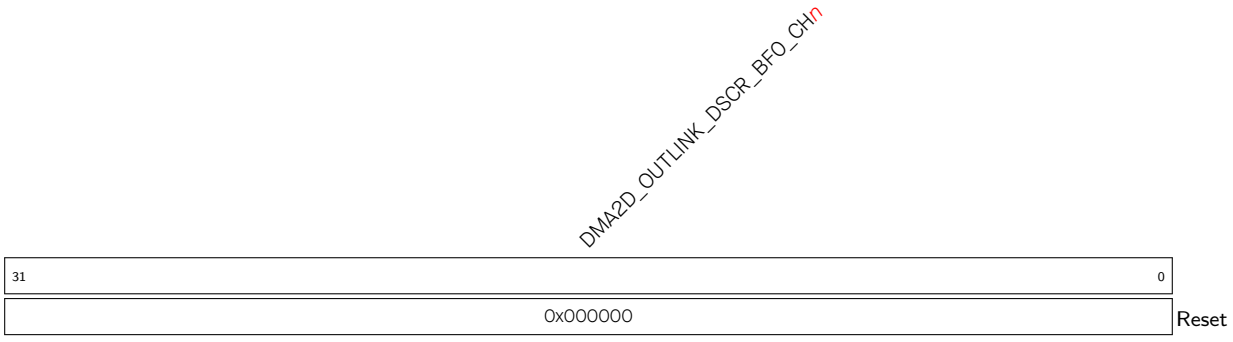
DMA2D_OUT_EOF_DES_ADDR_CH*n* Represents the address of the transmit descriptor when the eof bit in this descriptor is 1. (RO)

Register 4.50. DMA2D_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x002C+0x100**n*)



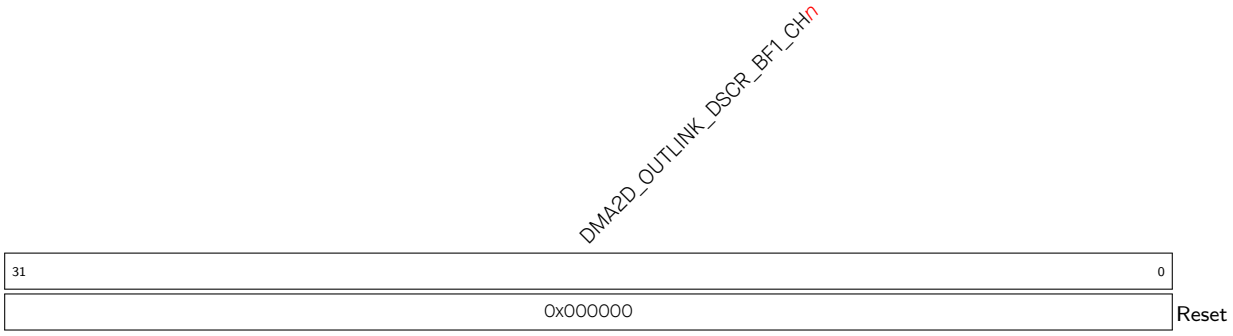
DMA2D_OUTLINK_DSCR_CH*n* Represents the address of the next transmit descriptor pointed by the current transmit descriptor that is pre-read. (RO)

Register 4.51. DMA2D_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (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*: 0-2) (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*: 0-1) (0x0514+0x100**n*)



DMA2D_INFIFO_FULL_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L2_CH*n* Represents whether the L2 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L2_CH*n* Represents the number of data bytes in the L2 RX FIFO for RX channel *n*. (RO)

DMA2D_IN_REMAIN_UNDER_1B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_2B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_3B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_4B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_5B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_6B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_7B_CH*n* Reserved. (RO)

DMA2D_IN_REMAIN_UNDER_8B_CH*n* Reserved. (RO)

DMA2D_INFIFO_FULL_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is full.

0: Not full

1: Full

(RO)

DMA2D_INFIFO_EMPTY_L1_CH*n* Represents whether the L1 RX FIFO for RX channel *n* is empty.

0: Not empty

1: Empty

(RO)

DMA2D_INFIFO_CNT_L1_CH*n* Represents the number of data bytes in the L1 RX FIFO for RX channel *n*. (RO)

Continued on the next page...

Register 4.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-1) (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.

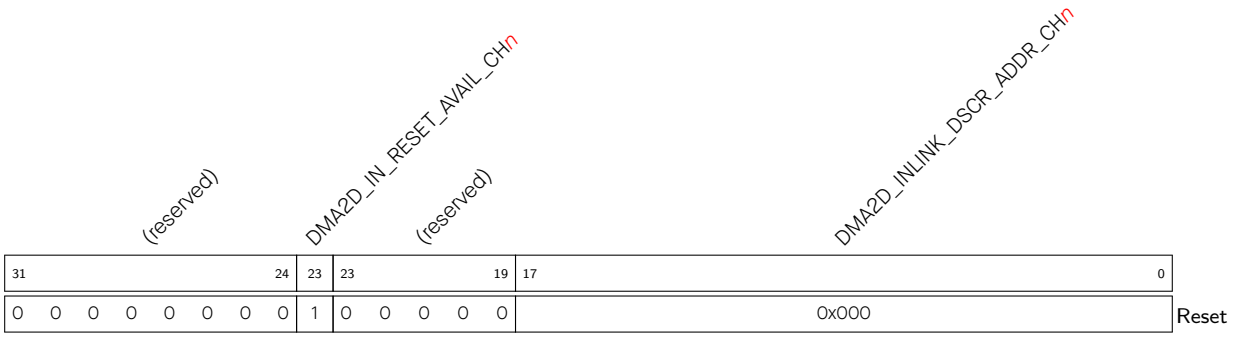
0: Not empty

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*: 0-1) (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)

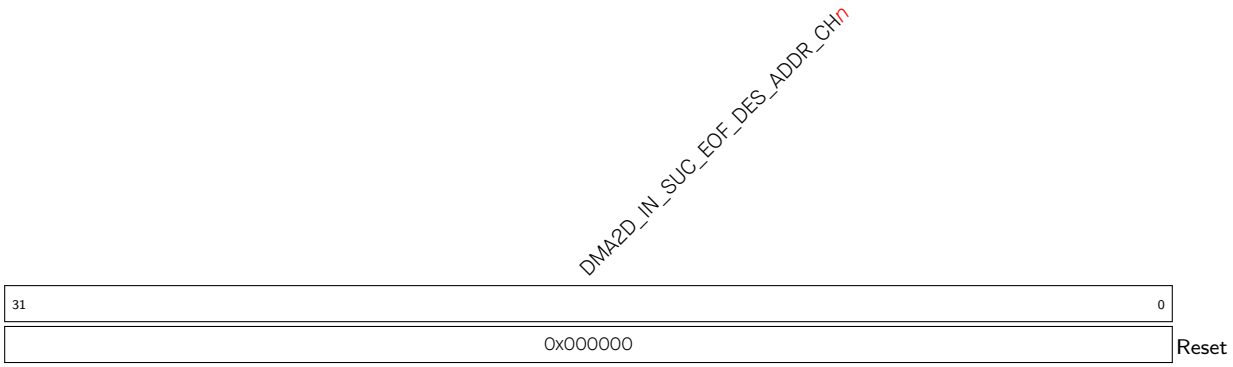
DMA2D_IN_RESET_AVAIL_CH*n* Represents whether it is safe to reset RX channel *n*.

0: Not safe

1: Safe

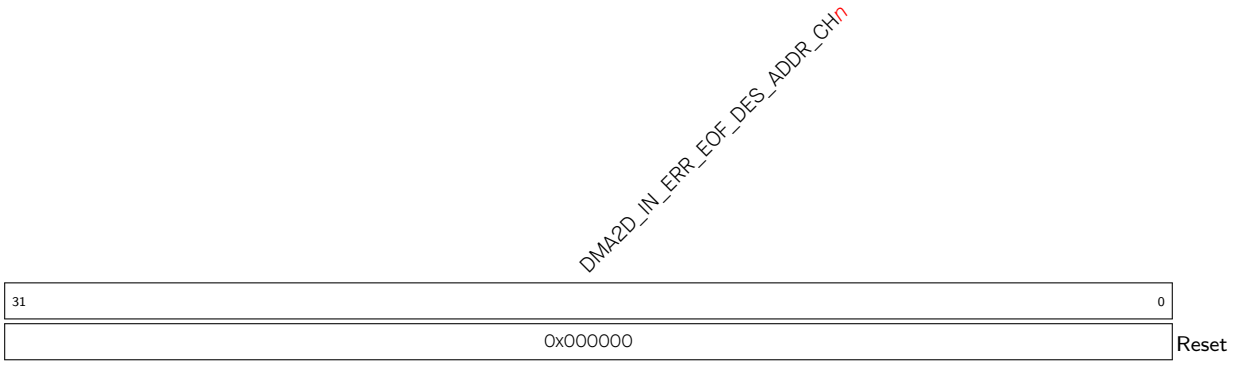
(RO)

Register 4.55. DMA2D_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x0528+0x100**n*)



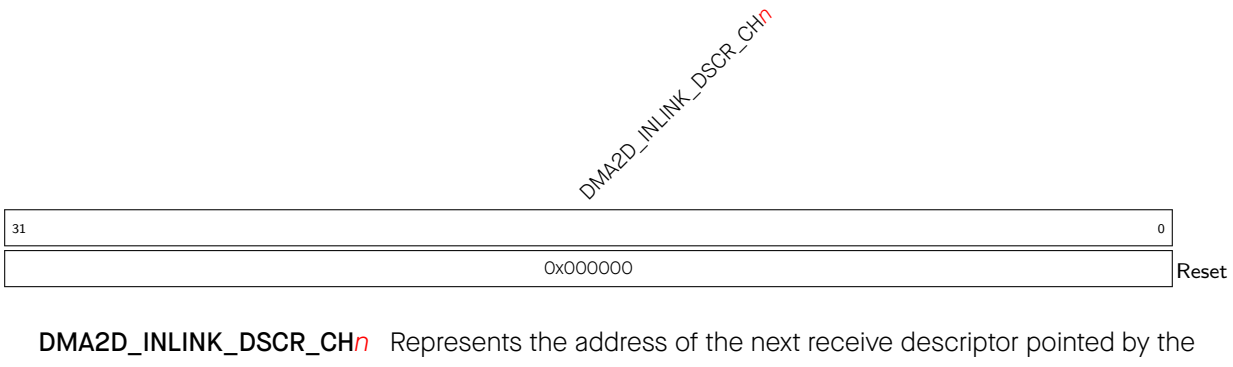
DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when the eof bit in this descriptor is 1. (RO)

Register 4.56. DMA2D_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-1) (0x052C+0x100**n*)



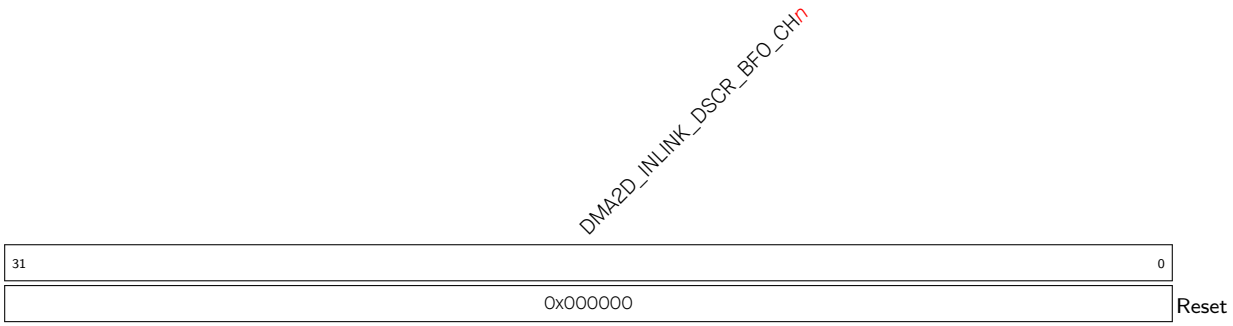
DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* Represents the address of the receive descriptor when there are some errors in the currently received data. Valid only for JPEG. (RO)

Register 4.57. DMA2D_IN_DSCR_CH*n*_REG (*n*: 0-1) (0x0530+0x100**n*)



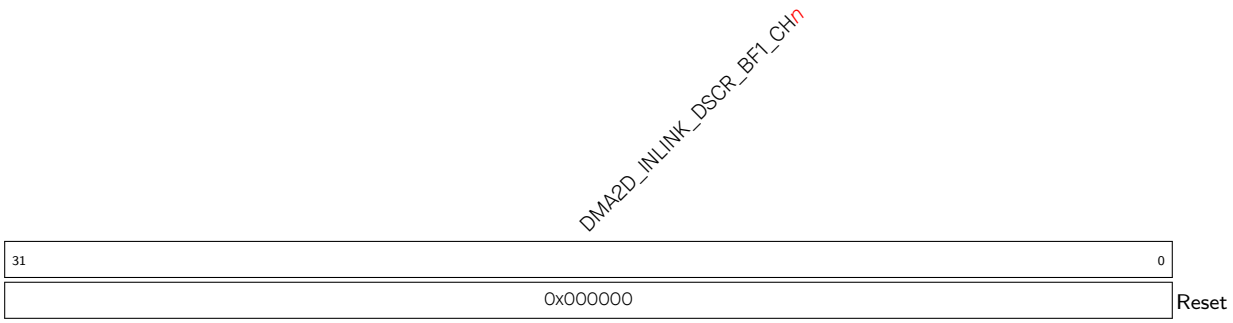
DMA2D_INLINK_DSCR_CH*n* Represents the address of the next receive descriptor pointed by the current receive descriptor that is pre-read. (RO)

Register 4.58. DMA2D_IN_DSCR_BF0_CH*n*_REG (*n*: 0-1) (0x0534+0x100**n*)



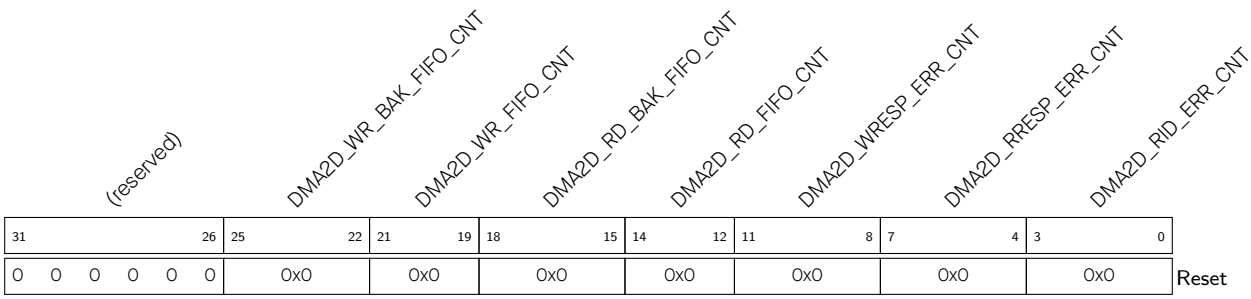
DMA2D_INLINK_DSCR_BF0_CH*n* Represents the address of the current receive descriptor that is pre-read. (RO)

Register 4.59. DMA2D_IN_DSCR_BF1_CH*n*_REG (*n*: 0-1) (0x0538+0x100**n*)



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)

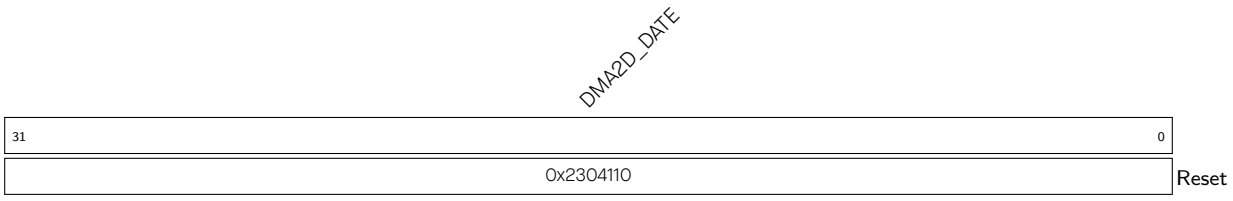
DMA2D_RD_FIFO_CNT Represents the number of remaining commands in the AXI read command FIFO. (RO)

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)

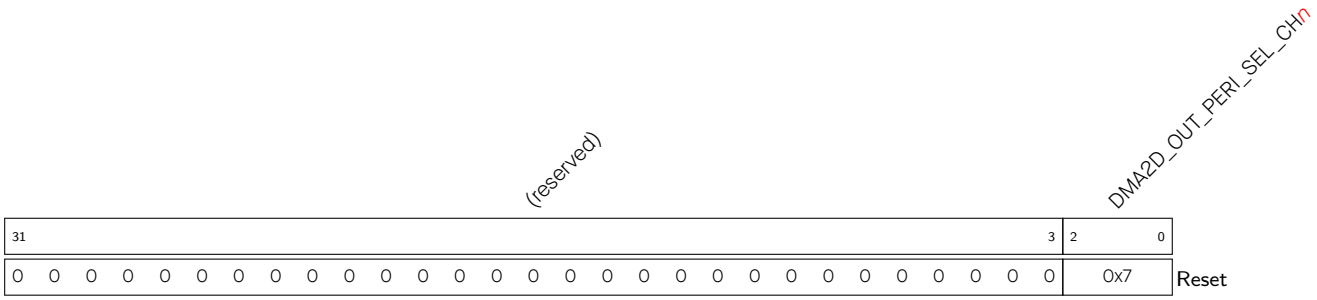
DMA2D_WR_BAK_FIFO_CNT Represents the number of remaining commands in the AXI write backup command FIFO. (RO)

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*: 0-2) (0x0038+0x100n*)**



DMA2D_OUT_PERI_SEL_CH*n* Configures the peripheral connected to TX channel *n*.

0: JPEG

1: SRM module of PPA

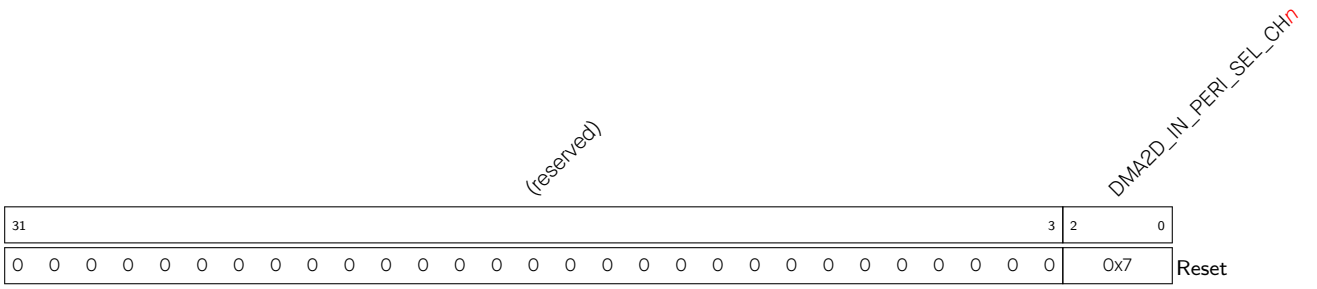
2: BLEND0 channel of PPA

3: BLEND1 channel of PPA

4 ~ 7: Dummy 4 ~ 7

(R/W)

Register 4.63. DMA2D_IN_PERI_SEL_CH*n*_REG (*n*: 0-1) (0x053C+0x100n*)**



DMA2D_IN_PERI_SEL_CH*n* Configures the peripheral connected to RX channel *n*.

0: JPEG

1: SRM module of PPA

2: BLEND module of PPA

3 ~ 7: Dummy 3 ~ 7

(R/W)

Part III

Memory Organization

This part provides insights into the system's memory structure, discussing the organization and mapping of RAM, ROM, eFuse, and external memories, offering a framework for understanding memory-related subsystems.

Chapter 5

System and Memory

5.1 Overview

ESP32-P4 integrates two processors:

- a high-performance 32-bit RISC-V dual-core processor (HP CPU), five-stage pipeline, clock frequency up to 400 MHz.
- a low-power 32-bit RISC-V single-core processor (LP CPU), two-stage pipeline, clock frequency up to 40 MHz.

All internal memory, external memory, and peripherals are located on the HP CPU and LP CPU buses.

5.2 Features

- **Address Space**
  - 896 KB of HP internal memory address space, accessible by the instruction bus or data bus of HP CPU via cache
  - 8 KB of HP internal memory address space, accessible by the instruction bus or data bus of HP CPU
  - 48 KB of LP internal memory address space, accessible by the instruction bus or data bus of LP CPU with zero latency
  - 1256 KB of peripheral address space
  - 64 MB of external flash virtual address space, accessible by the instruction bus or data bus
  - 64 MB of external RAM virtual address space, accessible by the instruction bus or data bus
  - 768 KB of internal DMA address space
  - 128 MB of external DMA address space
- **Internal Memory**
  - 128 KB of HP ROM
  - 768 KB of HP L2MEM
  - 8 KB of HP TCM (Tightly Coupled Memories)
  - 16 KB of LP ROM
  - 32 KB of LP SRAM
- **External Memory**

  - Supports up to 64 MB of external flash and RAM

- **Peripheral Space**
  - 96 modules/peripherals in total
- **GDMA**
  - 7 GDMA-AHB supported modules/peripherals
  - 6 GDMA-AXI supported modules/peripherals

5.3 Functional Description

5.3.1 Address Mapping

Figure 5.3-1 illustrates the system structure and address mapping. All the non-reserved addresses are accessible via both the instruction bus and the data bus, meaning that the instruction bus and the data bus share the same address space.

Both the data bus and instruction bus of the HP CPU and LP CPU are little-endian. However, the HP CPU's data bus (DBUS) has a 128-bit data width, while other buses maintain a 32-bit data width.

The HP CPU can access data via the data bus using single-byte, double-byte, 4-byte alignment, and, in case of AI Instruction, up to 16-byte alignment. The LP CPU can access data via the data bus using single-byte, double-byte, and 4-byte alignment.

The HP CPU has the following access capabilities:

- Direct access to HP TCM via both the data bus and instruction bus.
- Access to internal memory and external memory mapped into the address space via cache, including:
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB external flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB external RAM (0x4800_0000 ~ 0x4BFF_FFFF)

  Note:
  Software can use the addresses starting with "0x4" in two ways: cache-able access or noncache-able access, depending on the control of CPU Physical Memory Attribution (PMA). Cache-able access is done via cache. Noncache-able access is done without cache, which is similar to the direct access to the addresses starting with "0x8", however, such direct access to "0x8" addresses is usually for debugging.

- Direct access to internal memory and external memory mapped into the following address space without cache. This access is slower than that via cache.
  - 128 KB HP ROM (0x8FC0_0000 ~ 0x8FC1_FFFF)
  - 768 KB HP L2MEM (0x8FF0_0000 ~ 0x8FFB_FFFF)
  - 32 KB LP SRAM (0x5010_8000 ~ 0x5010_FFFF)

  - 64 MB external flash (0x8000_0000 ~ 0x83FF_FFFF)
  - 64 MB external RAM (0x8800_0000 ~ 0x8BFF_FFFF)
- Direct access to modules/peripherals via the data bus, including:
  - HP CPU Peripherals
  - HP Peripherals
  - LP Peripherals

The LP CPU's access capabilities include:

- Direct access to LP SRAM and LP ROM via both the data bus and instruction bus.
- Access to internal memory and external memory, including:
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB external flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB external RAM (0x4800_0000 ~ 0x4BFF_FFFF)
- Direct access to modules/peripherals via the data bus, including:
  - HP CPU Peripherals
  - HP Peripherals
  - LP Peripherals

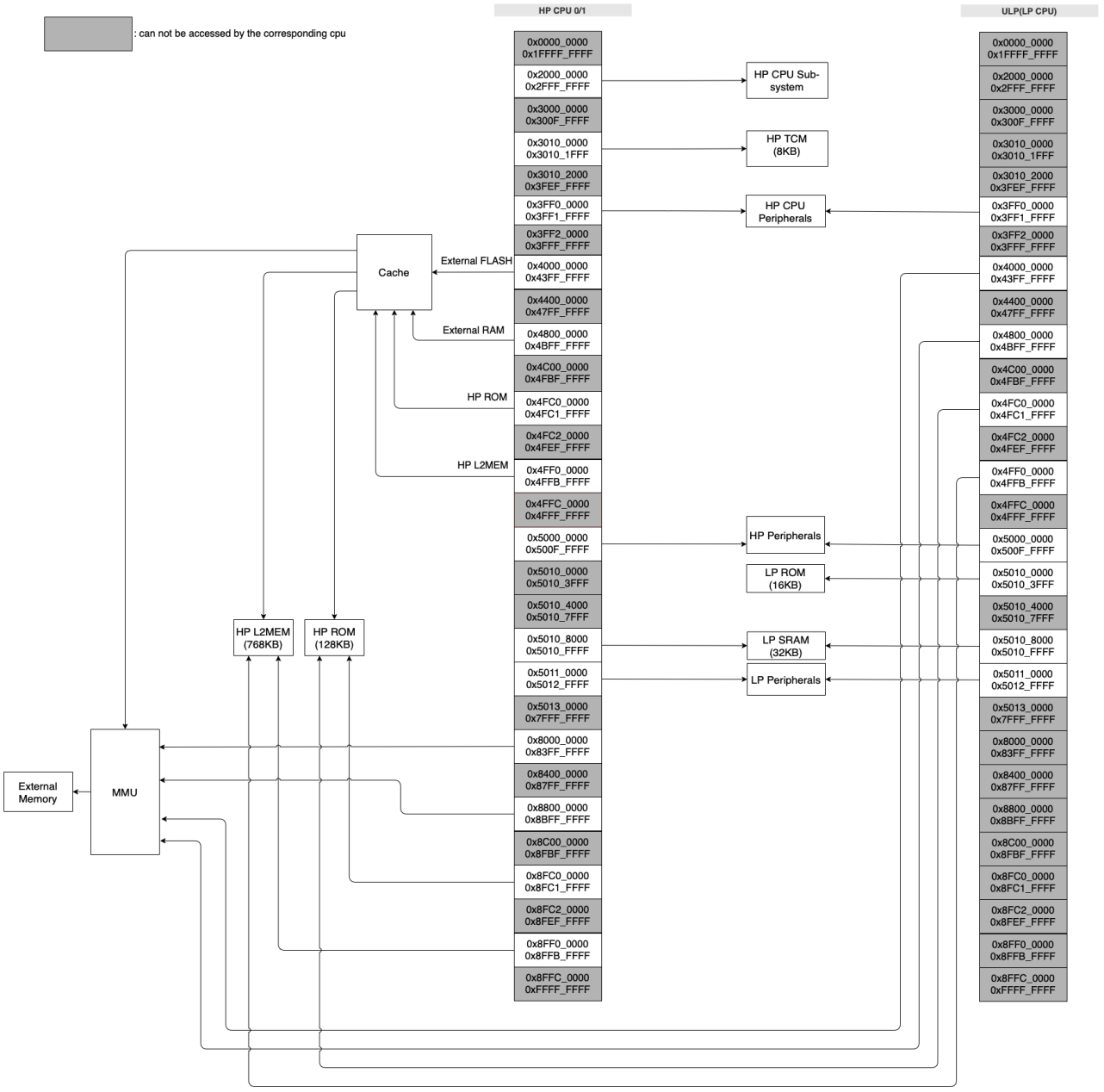


Figure 5.3-1. System Structure and Address Mapping

- The range of addresses available in the address space may be larger than the actual available memory of a particular type.
- For HP CPU sub-system, please refer to Chapter 1 *High-Performance CPU [to be added later]*.

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

Table 5.3-1. Memory Address Mapping

Bus Type	Boundary Address		Size	Target*
	Low Address	High Address		
	0x0000_0000	0x300F_FFFF		Reserved

Cont'd on next page

Table 5.3-1 – cont'd from previous page

Bus Type	Boundary Address		Size	Target*
	Low Address	High Address		
Data/Instruction bus	0x3010_0000	0x3010_1FFF	8 KB	HP TCM
	0x3010_2000	0x3FEF_FFFF		Reserved
Data/Instruction bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	HP CPU peripherals
	0x3FF2_0000	0x3FFF_FFFF		Reserved
Data/Instruction bus	0x4000_0000	0x43FF_FFFF	64 MB	External flash
	0x4400_0000	0x47FF_FFFF		Reserved
Data/Instruction bus	0x4800_0000	0x4BFF_FFFF	64 MB	External RAM
	0x4C00_000	0x4FBF_FFFF		Reserved
Data/Instruction bus	0x4FC0_0000	0x4FC1_FFFF	128 KB	HP ROM
	0x4FC2_0000	0x4FEF_FFFF		Reserved
Data/Instruction bus	0x4FF0_0000	0x4FFB_FFFF	768 KB	HP L2MEM
	0x4FFC_0000	0x4FFF_FFFF		Reserved
Data/Instruction bus	0x5000_0000	0x500F_FFFF	1 MB	HP peripherals
Data/Instruction bus	0x5010_0000	0x5010_3FFF	16 KB	LP ROM
	0x5010_4000	0x5010_7FFF		Reserved
Data/Instruction bus	0x5010_8000	0x5010_FFFF	32 KB	LP SRAM
Data/Instruction bus	0x5011_0000	0x5012_FFFF	128 KB	LP peripherals
	0x5013_0000	0x7FFF_FFFF		Reserved
Data/Instruction bus	0x8000_0000	0x83FF_FFFF	64 MB	External flash (direct access by CPU)
	0x8400_0000	0x87FF_FFFF		Reserved
Data/Instruction bus	0x8800_0000	0x8BFF_FFFF	64 MB	External RAM (direct access by CPU)
	0x8C00_000	0x8FBF_FFFF		Reserved
Data/Instruction bus	0x8FC0_0000	0x8FC1_FFFF	128 KB	HP ROM (direct access by CPU)
	0x8FC2_0000	0x8FEF_FFFF		Reserved
Data/Instruction bus	0x8FF0_0000	0x8FFB_FFFF	768 KB	HP L2MEM (direct access by CPU)
	0x8FFC_0000	0xFFFF_FFFF		Reserved

* Address starting with 0x4xxx_xxxx can be configured as noncache-able or cache-able, depending on CPU PMA, whereas address starting with 0x8xxx_xxxx is accessed directly by CPU.

* For the targets accessible by HP CPU or LP CPU, see Figure 5.3-1.

5.3.2 Internal Memory

ESP32-P4 has various types of internal memory:

- **HP ROM (128 KB):** This read-only memory is dedicated to the HP system and is not programmable. It contains the ROM code and read-only data of some low-level system software. The HP ROM is accessed by the HP CPU through the l2 cache at half the frequency of HP CPU.
- **HP L2MEM (768 KB):** A volatile memory accessible by the HP CPU, LP CPU, or DMA peripherals at half the frequency of the HP CPU. HP L2MEM can be configured to retain power during Light-sleep mode, making it suitable for data retention or register backup.
- **LP ROM (16 KB):** This read-only memory serves the LP system, containing the code for booting the LP CPU and some basic system functions. The LP ROM is accessed by the LP CPU with zero latency.

- **LP SRAM (32 KB):** A volatile memory accessible by both the HP CPU and LP CPU, operating at the same frequency as the LP CPU.
- **HP TCM (8 KB):** A volatile memory accessed by the HP CPU, finishing one access in two cycles.

1. Details for HP ROM

The 128 KB HP ROM is a read-only memory accessed by the HP CPU through the instruction bus or data bus via the following addresses as shown in Table 5.3-1:

- 0x4FC0_0000 (cache-able or noncache-able, configuration-dependent) ~ 0x4FC1_FFFF (cache-able or noncache-able, configuration-dependent).
- 0x8FC0_0000 (direct access by CPU) ~ 0x8FC1_FFFF (direct access by CPU).

2. Details for HP L2MEM

This 768 KB HP L2MEM is a read-and-write memory accessed by the HP CPU or LP CPU through the instruction bus or data bus, or by DMA master through AHB matrix or AXI matrix via the following addresses as shown in Table 5.3-1:

- 0x4FF0_0000 (cache-able or noncache-able, configuration-dependent) ~ 0x4FFB_FFFF (cache-able or noncache-able, configuration-dependent).
- 0x8FF0_0000 (direct access by CPU) ~ 0x8FFB_FFFF (direct access by CPU).

HP L2MEM can be accessed by various peripherals:

- by USB 2.0 High-speed, USB 2.0 Full-speed, GMAC, SDMMC, TRACE, GDMA-AHB, and SYSMON via AHB matrix.
- by GDMA, GDMA-AXI, 2D-DMA, and H264 Encoder via AXI matrix.

The HP L2MEM supports Error Correction Code (ECC) check to correct one-bit memory flip error. In the event of a one-bit flip error, the data in HP L2MEM still remains valid. Before ECC check, HP L2MEM itself must be initialized first.

HP L2MEM consists of six memory units, each occupying 128 KB. The initialization process allows individual configuration of each unit for power considerations. It is recommended to initialize memory units sequentially rather than simultaneously. This sequential approach helps prevent excessive power consumption and avoids triggering a brownout condition when initializing the entire 768 KB HP L2MEM.

Detailed Configuration Steps:
Take unit0 as an example, other units follow the same configuration steps.

- Set HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN and clear HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET to start initializing HP L2MEM.
- Read the value of HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE.
- Once HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE reaches a value of 1, set HP_SYSTEM_L2_RAM_UNIT0_ECC_EN to start ECC check.
- (Recommended) clear HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN and set HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET after HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE reaches 1 to complete the initialization process.

During the initialization process, avoid any access to HP L2MEM unit being initialized.

3. Details for LP ROM

This 16 KB LP ROM is a read-only memory accessed by LP CPU through the instruction bus or data bus via their shared address 0x5010_0000 ~ 0x5010_3FFF as shown in Table 5.3-1.

4. Details for LP SRAM

This 32 KB LP SRAM is a read-and-write memory accessed by the HP CPU via AHB matrix and by the LP CPU through the instruction bus or data bus via their shared address 0x5010_8000 ~ 0x5010_FFFF as shown in Table 5.3-1. LP SRAM supports atomic operation.

5. Details for HP TCM

This 8 KB HP TCM is a read-and-write memory accessed by the HP CPU through the instruction bus or data bus via their shared address 0x3010_0000 ~ 0x3010_1FFF as shown in Table 5.3-1. HP TCM operates at the same frequency as the HP CPU, offering the fastest internal memory access in ESP32-P4. Additionally, HP TCM supports atomic operation of the HP CPU.

HP TCM supports parity check to detect one-bit memory flip error. Before conducting the parity check, HP TCM itself must be initialized first.

Detailed Configuration Steps:

- Set HP_SYSTEM_TCM_INIT_EN and clear HP_SYSTEM_TCM_INIT_CNT_RESET to start initializing HP TCM.
- Read the value of HP_SYSTEM_TCM_INIT_DONE.
- Once HP_SYSTEM_TCM_INIT_DONE reaches a value of 1, set HP_SYSTEM_HP_TCM_PARITY_CHECK_EN to start parity check.
- (Recommended) clear HP_SYSTEM_TCM_INIT_EN and set HP_SYSTEM_TCM_INIT_CNT_RESET after HP_SYSTEM_TCM_INIT_CNT_RESET reaches 1 to complete the initialization process.

During the above process, avoid any access to HP TCM.

5.3.3 External Memory

ESP32-P4 supports SPI, dual SPI, quad SPI and QPI interfaces for connecting to external flash, as well as OPI and HPI interfaces for connecting to external RAM.

ESP32-P4 provides the following security features to protect users' data in external flash and RAM:

- hardware manual encryption and automatic decryption based on XTS-AES algorithm to protect users' programs and data stored in the external flash.
- automatic encryption and decryption based on XTS-AES algorithm to protect users' data in the external RAM.

5.3.3.1 External Memory Address Mapping

HP CPU can access external flash or RAM via 0x4000_0000 ~ 0x43FF_FFFF or via 0x4800_0000 ~ 0x4BFF_FFFF using cache-able or noncache-able methods. HP CPU can also access external flash or RAM directly via 0x8000_0000 ~ 0x83FF_FFFF or via 0x8800_0000 ~ 0x8BFF_FFFF, which is slower than the access via cache and is therefore commonly used for debugging.
When the HP CPU accesses external memory, it maps the addresses to physical addresses based on the information in the Memory Management Unit (MMU).

- Range (0x4000_0000 ~ 0x43FF_FFFF) is mapped to the physical address of the external flash accessed by HP CPU via cache.
- Range (0x4800_0000 ~ 0x4BFF_FFFF) is mapped to the physical address of the external RAM accessed by HP CPU via cache.
- Range (0x8000_0000 ~ 0x83FF_FFFF) is mapped to the physical address of the external flash accessed by HP CPU directly.
- Range (0x8800_0000 ~ 0x8BFF_FFFF) is mapped to the physical address of the external RAM accessed by HP CPU directly.

This mapping allows ESP32-P4 to address up to 64 MB of external flash and 64 MB of external RAM. **Note that** the instruction bus shares the same address space (64 MB) with the data bus for accessing external memory.

5.3.3.2 Cache

As shown in Figure 5.3-2, ESP32-P4 has a two-level cache system, with the following features:

- L1 cache, including:
  - 16 KB of instruction cache (icache) with a 64 B block size, four-way set associative
  - 64 KB of data cache (dcache) with a 64 B block size, two-way set associative, supporting two writing strategies: write-through and write-back
- L2 cache: 128 KB/256 KB/512 KB with 64 B/128 B block sizes, eight-way set associative
- Cache-able and noncache-able access, depending on CPU PMA
- Preload operation
- Lock operation
- Critical word first and early restart

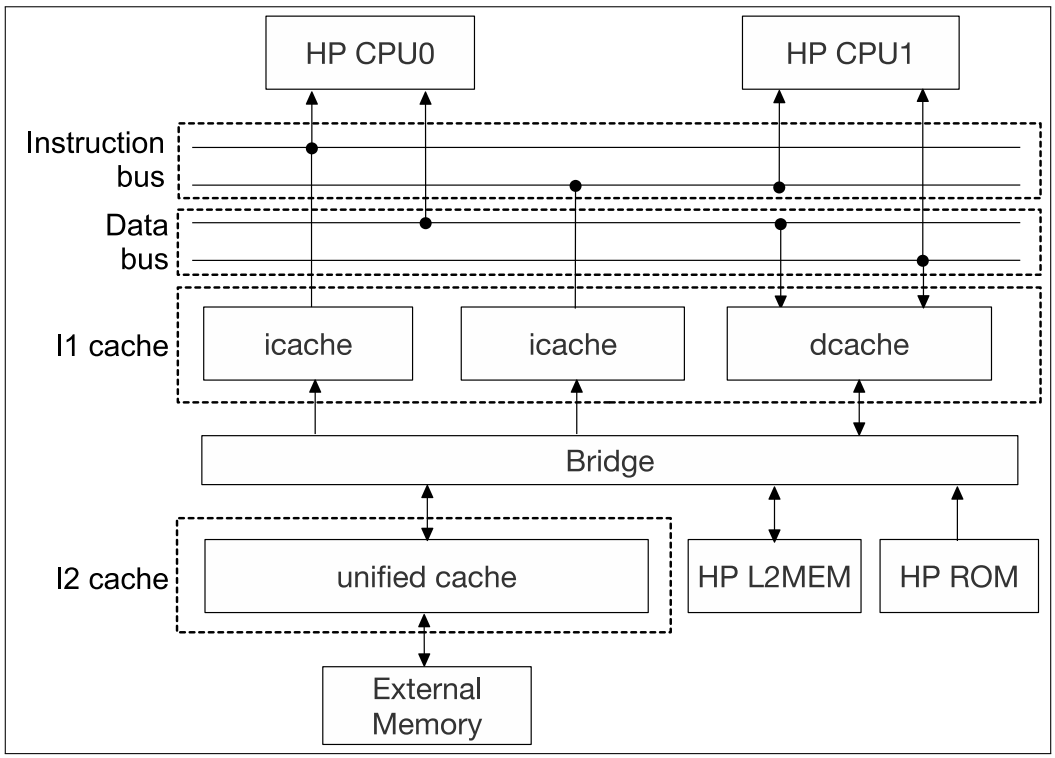


Figure 5.3-2. Cache Structure

5.3.3.3 Cache Operations

ESP32-P4 I1 cache and I2 cache supports the following operations:

1. **Write-back** (for dcache and I2 cache only):
   - Clears dirty bits in the tag memory and updates new data to external memory.
   - After the write-back operation, the new data is updated to external memory.
   - Users can choose whether to invalidate the data in the cache.
   - If the data is not invalidated, CPU will read/write data directly from/to the cache where data missing will not occur.
2. **Clean** (for dcache and I2 cache only):
   - Clears dirty bits in the tag memory without updating data to external memory.
   - After the clean operation, old data remains in external memory, while the cache holds the new one (unaware of it).
   - CPU can then read/write the data directly from/to the cache, where data missing will not occur.
3. **Invalidate**:
   - Removes valid data from the cache.
   - Once this operation is done, the deleted data is stored only in external memory.
   - HP CPU needs to access external memory if it wants to access the data again.
   - Two types of invalidate operation:

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

4. **Preload**:
   - Loads instructions and data into the cache in advance.
   - Minimum unit of preload-operation is one block.
   - Two types of preload operation:
     - Manual-Preload: involves the hardware prefetching continuous data based on the virtual address specified by the software.
     - Auto-Preload: involves the hardware prefetching continuous data based on the current address where the cache hits or misses (configuration-dependent).
5. **Lock/Unlock**:
   - Lock operation prevents easily replacing data in the cache.
   - Two types of lock:
     - Prelock: locks data in the specified area when filling missing data to cache memory, leaving the data outside the specified area unlocked.
     - Manual Lock: checks data in the cache memory and locks data only if it falls in the specified area, leaving data outside the specified area unlocked.
   - When there is missing data, the cache replaces the data in the unlocked way first, ensuring that the data in the locked way remains in the cache and will not be replaced.
   - When all ways within the cache are locked, the cache will replace data, as if it was not locked.
   - Unlocking is the reverse of locking and can only be done manually.
   - Note: Manual-Invalidate operation only works on the unlocked data. If you plan to perform this operation on the locked data, please unlock them first.

5.3.4 DMA Address Space

ESP32-P4 includes AHB General Direct Memory Access (GDMA-AHB), AXI General Direct Memory Access (GDMA-AXI), and DW General Direct Memory Access (DW-GDMA), all providing general direct memory access services.

In ESP32-P4, masters like USB 2.0 High-speed, USB 2.0 Full-speed, GDMAC, Trace0/1, DW-GDMA, 2D-DMA, and H264 Encoder have built-in DMA function modules, providing direct memory access services.

All DMA-enabled master devices mentioned above can access:

- HP L2MEM (0x4FF0_0000~0x4FFB_FFFF).
- external flash (0x4000_0000 ~ 0x43FF_FFFF).
- external RAM (0x4800_0000 ~ 0x4BFF_FFFF).

Additionally, DW-GDMA Master 1 can also access:

- MIPI CSI internal memory (0x5010_4000 ~ 0x5010_4FFF).

- MIPI DSI internal memory (0x5010_5000 ~ 0x5010_5FFF).

Note:
When accessing a memory via DMA, a corresponding access permission is needed, otherwise this access may fail. For more information about permission control, please refer to Chapter 16 *Permission Control (PMS)*.

5.3.5 Modules/Peripherals Address Mapping

Table 5.3-2 lists all the modules/peripherals and their respective address ranges. Note that the address space of specific modules/peripherals is defined by "Boundary Address" (including both Low Address and High Address).

Table 5.3-2. Module/Peripheral Address Mapping

Target	Boundary Address		Size (KB)
	Low Address	High Address	
HP CPU Peripherals (HP CPU PERI)			
Reserved	0x3FF0_0000	0x3FF0_3FFF	
RISC-V Trace Encoder 0	0x3FF0_4000	0x3FF0_4FFF	4
RISC-V Trace Encoder 1	0x3FF0_5000	0x3FF0_5FFF	4
Bus Monitor	0x3FF0_6000	0x3FF0_6FFF	4
Reserved	0x3FF0_7000	0x3FF0_DFFF	
L2MEM Monitor	0x3FF0_E000	0x3FF0_EFFF	4
TCM Monitor	0x3FF0_F000	0x3FF0_FFFF	4
Cache	0x3FF1_0000	0x3FF1_FFFF	64
HP Peripherals 0 (HP PERI0)			
USB 2.0 OTG High-Speed	0x5000_0000	0x5003_FFFF	256
USB 2.0 OTG Full-Speed	0x5004_0000	0x5007_FFFF	256
USB 2.0 OTG Full-Speed PHY	0x5008_0000	0x5008_0FFF	4
DW-GDMA Controller	0x5008_1000	0x5008_1FFF	4
Reserved	0x5008_2000	0x5008_2FFF	
SD/MMC Host Controller	0x5008_3000	0x5008_3FFF	4
H264 Encoder	0x5008_4000	0x5008_4FFF	4
GDMA-AHB	0x5008_5000	0x5008_5FFF	4
JPEG Codec	0x5008_6000	0x5008_6FFF	4
Pixel-Processing Accelerator (PPA)	0x5008_7000	0x5008_7FFF	4
2D-DMA Controller	0x5008_6000	0x5008_8FFF	4
Key Management	0x5008_6000	0x5008_9FFF	4
GDMA-AXI	0x5008_A000	0x5008_BFFF	8
AES Accelerator (AES)	0x5009_0000	0x5009_0FFF	4
SHA Accelerator (SHA)	0x5009_1000	0x5009_1FFF	4
RSA Accelerator (RSA)	0x5009_2000	0x5009_2FFF	4
Digital Signature (DS)	0x5009_3000	0x5009_3FFF	4
HMAC Accelerator (HMAC)	0x5009_4000	0x5009_4FFF	4

Cont'd on next page

Table 5.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
ECC Accelerator (ECC)	0x5009_5000	0x5009_5FFF	4
ECDSA	0x5009_6000	0x5009_6FFF	4
Reserved	0x5009_7000	0x5009_7FFF	
GMAC	0x5009_8000	0x5009_BFFF	16
USB 2.0 OTG High-Speed PHY	0x5009_C000	0x5009_CFFF	4
Reserved	0x5009_D000	0x5009_DFFF	
CSI Host	0x5009_F000	0x500F_FFFF	4
DSI Host	0x500A_0000	0x500A_0FFF	4
Image Signal Processor (ISP)	0x500A_1000	0x500A_1FFF	4
Remote Control Peripheral (RMT)	0x500A_2000	0x500A_2FFF	4
Bit-scrambler	0x500A_3000	0x500A_3FFF	4
AXI ICM	0x500A_4000	0x500A_4FFF	4
HP Peripheral Permission (HP_PERI_PMS)	0x500A_5000	0x5009_57FF	2
LP2HP Peripheral Permission (LP2HP_PERI_PMS)	0x500A_5800	0x500A_5FFF	2
HP DMA Permission (HP_DMA_PMS)	0x500A_6000	0x500A_6FFF	4
H264 DMA	0x500A_7000	0x500A_7FFF	4
Reserved	0x500A_8000	0x500B_FFFF	
HP Peripherals 1 (HP PERI1)			
Motor Control PWM0 (MCPWM0)	0x500C_0000	0x500C_0FFF	4
Motor Control PWM1 (MCPWM1)	0x500C_1000	0x500C_1FFF	4
Timer Group 0 (TIMG0)	0x500C_2000	0x500C_2FFF	4
Timer Group 1 (TIMG1)	0x500C_3000	0x500C_3FFF	4
I2C Controller0 (I2C0)	0x500C_4000	0x500C_4FFF	4
I2C Controller1 (I2C1)	0x500C_5000	0x500C_5FFF	4
I2S Controller0 (I2S0)	0x500C_6000	0x500C_6FFF	4
I2S Controller1 (I2S1)	0x500C_7000	0x500C_7FFF	4
I2S Controller0 (I2S2)	0x500C_8000	0x500C_8FFF	4
Pulse Count Controller (PCNT)	0x500C_9000	0x500C_9FFF	4
UART Controller 0 (UART0)	0x500C_A000	0x500C_AFFF	4
UART Controller 1 (UART1)	0x500C_B000	0x500C_BFFF	4
UART Controller 2 (UART2)	0x500C_C000	0x500C_CFFF	4
UART Controller 3 (UART3)	0x500C_D000	0x500C_DFFF	4
UART Controller 4 (UART4)	0x500C_E000	0x500C_EFFF	4
Parallel IO Controller (PARL_IO)	0x500C_F000	0x500C_FFFF	4
General Purpose SPI2 (GP-SPI2)	0x500D_0000	0x500D_0FFF	4
General Purpose SPI3 (GP-SPI3)	0x500D_1000	0x500D_1FFF	4
USB Serial/JTAG Controller	0x500D_2000	0x500D_2FFF	4
LED PWM Controller (LEDC)	0x500D_3000	0x500D_3FFF	4
Reserved	0x500D_4000	0x500D_4FFF	
Event Task Matrix (SOC_ETM)	0x500D_5000	0x500D_5FFF	4

Cont'd on next page

Table 5.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Interrupt Matrix (INTMTX)	0x500D_6000	0x500D_6FFF	4
Two-wire Automotive Interface 0 (TWAI0)	0x500D_7000	0x500D_7FFF	4
Two-wire Automotive Interface 1 (TWAI1)	0x500D_8000	0x500D_8FFF	4
Two-wire Automotive Interface 2 (TWAI2)	0x500D_9000	0x500D_9FFF	4
I3C Master Controller	0x500D_A000	0x500D_AFFF	4
I3C Slave Controller	0x500D_B000	0x500D_BFFF	4
LCD and Camera Controller (LCD_CAM)	0x500D_C000	0x500D_CFFF	4
Reserved	0x500D_D000	0x500D_DFFF	
ADC Controller	0x500D_E000	0x500D_EFFF	4
UHCI Controller (UHCI)	0x500D_F000	0x500D_FFFF	4
GPIO Matrix	0x500E_0000	0x500E_0FFF	4
IO MUX	0x500E_1000	0x500E_1FFF	4
System Timer (SYSTIMER)	0x500E_2000	0x500E_2FFF	4
Reserved	0x500E_3000	0x500E_4FFF	
HP System Register (SYSREG)	0x500E_5000	0x500D_5FFF	4
Reset and Clock	0x500E_6000	0x500E_6FFF	4
Reserved	0x500E_7000	0x500F_FFFF	
MIPI Camera and LCD Memory			
MIPI Camera Memory	0x5010_4000	0x5010_4FFF	4
MIPI LCD Memory	0x5010_5000	0x5010_5FFF	4
LP Always On Peripherals (LP AON PERI)			
LP System Register	0x5011_0000	0x5011_0FFF	4
LP Always-on Clock and Reset	0x5011_1000	0x5011_1FFF	4
LP Timer	0x5011_2000	0x5011_2FFF	4
LP Analog peripherals (LP ANAPERI)	0x5011_3000	0x5011_3FFF	4
LP HUK	0x5011_4000	0x5011_4FFF	4
Power Management Unit (PMU)	0x5011_5000	0x5011_5FFF	4
LP Watch Dog Timer (LP WDT)	0x5011_6000	0x5011_6FFF	4
Reserved	0x5011_7000	0x5011_7FFF	
LP Maibox (LP MB)	0x5011_8000	0x5011_8FFF	4
Reserved	0x5011_A000	0x5011_FFFF	
LP Peripherals (LP PERI)			
LP Peripheral Clock and Reset (LP PERI-CLKRST)	0x5012_0000	0x5012_0FFF	4
LP UART	0x5012_1000	0x5012_1FFF	4
LP I2C	0x5012_2000	0x5012_2FFF	4
LP SPI	0x5012_3000	0x5012_3FFF	4
LP Analog I2C	0x5012_4000	0x5012_4FFF	4
LP I2S	0x5012_5000	0x5012_5FFF	4
Reserved	0x5012_6000	0x5012_6FFF	4
LP ADC	0x5012_7000	0x5012_7FFF	4

Cont'd on next page

Table 5.3-2 – cont'd from previous page

Target	Boundary Address		Size (KB)
	Low Address	High Address	
Reserved	0x5012_8000	0x5012_9FFF	
LP GPIO Matrix	0x5012_A000	0x5012_AFFF	4
LP IO MUX	0x5012_B000	0x5012_BFFF	4
LP Interrupt (LP INTR)	0x5012_C000	0x5012_CFFF	4
LP eFuse	0x5012_D000	0x5012_DFFF	4
LP Peripheral Permission (LP_PERI_PMS)	0x5012_E000	0x5012_E7FF	2
HP2LP Peripheral Permission (HP2LP_PERI_PMS)	0x5012_E800	0x5012_EFFF	2
LP Temperature Sensor	0x5012_F000	0x5012_FFFF	4

Note:

As shown in Figure 5.3-1, HP CPU and LP CPU can access HP CPU PERI, HP PERI0, HP PERI1, LP AON PERI and LP PERI listed in Table 5.3-2.

Chapter 6

eFuse Controller

6.1 Overview

ESP32-P4 contains a 4096-bit eFuse memory to store parameters and user data. The parameters include control parameters for some hardware modules, system data parameters and keys used for the encryption/decryption module. Once an eFuse bit is programmed to 1, it can never be reverted to 0. The eFuse controller programs individual bits of parameters in eFuse according to user configurations. From outside the chip, eFuse data can only be read via the eFuse controller. For some data, such as some keys stored in eFuse for internal use by hardware cryptography modules (e.g., digital signature, HMAC), if read protection is not enabled, the data can be read from outside the chip; if read protection is enabled, the data cannot be read from outside the chip.

6.2 Features

- 4096-bit one-time programmable memory (including up to 1792 bits reserved for custom use depending on whether the EFUSE_KEY_PURPOSE_*n* parameters are set to 0. For more information, see Table 6.3-2.)
- Configurable write protection
- Configurable read protection
- Various hardware encoding schemes against data corruption

6.3 Functional Description

6.3.1 Structure

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

Users can program bits in the eFuse memory via the eFuse controller by writing the data to be programmed to the programming register and executing the programming instruction. For detailed programming steps, please refer to Section 6.3.2.

Users cannot directly read the data programmed in the eFuse memory, so they need to read the programmed data into the Reading Data Register of the corresponding address segment through the eFuse controller. During the reading process, if the data is inconsistent with that in the eFuse memory, the eFuse controller can automatically correct it through the hardware encoding mechanism (see Section 6.3.1.3 for details), and send

the error message to the error report register. For detailed steps to read parameters, please refer to the Section 6.3.3.

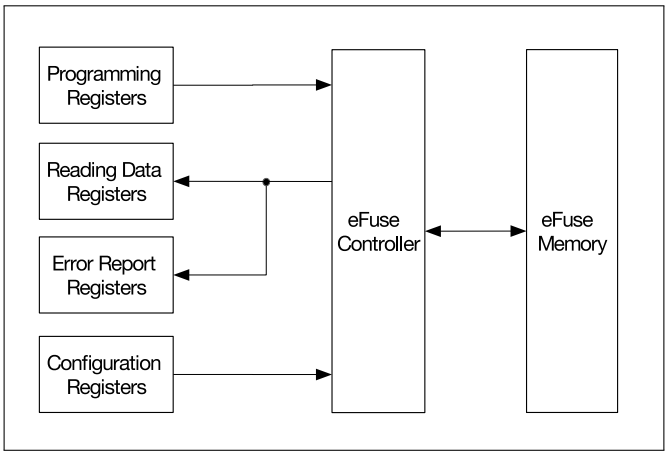


Figure 6.3-1. Data Flow in eFuse

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

BLOCK0 holds most parameters for software and hardware uses.

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

The **EFUSE_WR_DIS** parameter is used to disable write protection of other parameters. **EFUSE_RD_DIS** is used to disable read protection of BLOCK4 ~ BLOCK10. For more information on these two parameters, please see Section 6.3.1.1 and Section 6.3.1.2.

Table 6.3-1. Parameters in eFuse BLOCK0

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_WR_DIS	32	Y	N/A	Represents whether writing of eFuse bits by eFuse controller is disabled.
EFUSE_RD_DIS	7	Y	0	Represents whether reading data from BLOCK4 ~ 10 in eFuse memory by users is disabled.
EFUSE_USB_DEVICE_EXCHG_PINS	1	Y	30	Represents whether the USB Serial/JTAG pins D+ and D- are exchanged.
EFUSE_USB_OTG11_EXCHG_PINS	1	Y	30	Represents whether the USB FS pins D+ and D- are exchanged.
EFUSE_DIS_USB_JTAG	1	Y	2	Represents whether the USB-to-JTAG function in USB Serial/JTAG module is disabled.
EFUSE_POWERGLITCH_EN	1	Y	2	Represents whether power glitch detection is enabled.
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	Represents whether the function to force the chip into Download mode is disabled.
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	2	Represents accessing MSPI flash/MSPI RAM by SYS AXI matrix is disabled during boot_mode_download.
EFUSE_DIS_TWAI	1	Y	2	Represents whether the TWAI controller is disabled.
EFUSE_JTAG_SEL_ENABLE	1	Y	2	Represents whether the selection of a JTAG signal source through the strapping value is enabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0.
EFUSE_SOFT_DIS_JTAG	3	Y	31	Represents whether PAD JTAG is disabled in the soft way.
EFUSE_DIS_PAD_JTAG	1	Y	2	Represents whether PAD JTAG is disabled in the hard way (permanently).
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	Represents whether flash encryption is disabled (except in SPI boot mode).

Cont'd on next page

GoBack

Table 6.3-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_USB_PHY_SEL	1	Y	30	Represents the connection between USB Serial/JTAG Controller, OTG_FS and FS_PHY1, FS_PHY2.
EFUSE_KM_HUK_GEN_STATE_LOW	6	Y	19	Represents the lower bits that determine the validity of the Hardware Unique Key (HUK) generation mode. It is combined with the EFUSE_KM_HUK_GEN_STATE_HIGH field to indicate whether the HUK generation mode is valid.
EFUSE_KM_HUK_GEN_STATE_HIGH	3	Y	19	Represents the higher bits that determine the validity of the Hardware Unique Key (HUK) generation mode. It is combined with the EFUSE_KM_HUK_GEN_STATE_LOW field to represent whether the HUK generation mode is valid.
EFUSE_KM_RND_SWITCH_CYCLE	2	Y	1	Represents the cycle at which the Key Manager (KM) switches random numbers.
EFUSE_KM_DEPLOY_ONLY_ONCE	4	Y	1	Represents whether the corresponding key can be deployed only once.
EFUSE_FORCE_USE_KEY_MANAGER_KEY	4	Y	1	Represents whether the corresponding key must come from the Key Manager.
EFUSE_FORCE_DISABLE_SW_INIT_KEY	1	Y	1	Represents whether to disable the use of the initialization key written by software and instead force use efuse_init_key.
EFUSE_XTS_KEY_LENGTH_256	1	N	1	Represents which key is used for flash encryption.
EFUSE_WDT_DELAY_SEL	2	Y	2	Represents RTC watchdog timeout threshold.
EFUSE_SPI_BOOT_CRYPT_CNT	3	Y	4	Represents whether SPI boot encryption/decryption is enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE0	1	N	5	Represents whether revoking the first Secure Boot key is enabled.

Cont'd on next page

Table 6.3-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	Represents whether revoking the second Secure Boot key is enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	Represents whether revoking the third Secure Boot key is enabled.
EFUSE_KEY_PURPOSE_0	4	Y	8	Represents Key0 purpose. See Table 6.3-2.
EFUSE_KEY_PURPOSE_1	4	Y	9	Represents Key1 purpose. See Table 6.3-2.
EFUSE_KEY_PURPOSE_2	4	Y	10	Represents Key2 purpose. See Table 6.3-2.
EFUSE_KEY_PURPOSE_3	4	Y	11	Represents Key3 purpose. See Table 6.3-2.
EFUSE_KEY_PURPOSE_4	4	Y	12	Represents Key4 purpose. See Table 6.3-2.
EFUSE_KEY_PURPOSE_5	4	Y	13	Represents Key5 purpose. See Table 6.3-2.
EFUSE_SEC_DPA_LEVEL	2	Y	14	Represents the security level of anti-DPA (differential power analysis) attack.
EFUSE_CRYPT_DPA_ENABLE	1	Y	14	Represents whether defense against DPA attack is enabled.
EFUSE_SECURE_BOOT_EN	1	N	15	Represents whether Secure Boot is enabled.
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	Represents whether aggressive revocation of Secure Boot is enabled.
EFUSE_FLASH_TYPE	1	N	18	Represents the type of the interfaced flash.
EFUSE_FLASH_PAGE_SIZE	2	N	18	Represents flash page size.
EFUSE_FLASH_ECC_EN	1	N	18	Represents whether ECC is enabled during flash boot.
EFUSE_DIS_USB_OTG_DOWNLOAD_MODE	1	N	18	Represents whether download via USB-OTG is disabled.
EFUSE_FLASH_TPUW	4	N	18	Represents the flash waiting time after power-up.
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	Represents whether all Download modes are disabled.
EFUSE_DIS_DIRECT_BOOT	1	N	18	Represents whether direct boot mode is disabled.
EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT	1	N	18	Represents whether print from USB-Serial-JTAG during ROM boot is disabled.

Cont'd on next page

GoBack

Table 6.3-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_LOCK_KM_KEY	1	N	1	Represents whether the keys in the Key Manager are locked after deployment.
EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE	1	N	18	Represents whether the USB-Serial-JTAG download function is disabled.
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	18	Represents whether security download is enabled.
EFUSE_UART_PRINT_CONTROL	2	N	18	Represents the type of UART printing.
EFUSE_FORCE_SEND_RESUME	1	N	18	Represents whether ROM code is forced to send a resume command during SPI boot.
EFUSE_SECURE_VERSION	16	N	18	Represents the version used by ESP-IDF anti-rollback feature.
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	19	Represents whether FAST VERIFY ON WAKE is disabled or enabled when Secure Boot is enabled.
EFUSE_HYS_EN_PAD	1	Y	2	Represents whether the hysteresis function of PAD0 – PAD27 is enabled.
EFUSE_DCDC_VSET	5	Y	N/A	Represents the default DCDC voltage.
EFUSE_OPXA_TIEH_SEL0	2	Y	2	Represents which output source is used for Extern Power (VO1).
EFUSE_OPXA_TIEH_SEL1	2	Y	2	Represents which output source is used for Extern Power (VO2).
EFUSE_OPXA_TIEH_SEL2	2	Y	2	Represents which output source is used for Extern Power (VO3).
EFUSE_OPXA_TIEH_SEL3	2	Y	2	Represents which output source is used for Extern Power (VO4).
EFUSE_KM_DISABLE_DEPLOY_MODE	4	Y	1	Represents whether the deploy mode of the Key Manager is disabled.
EFUSE_HP_PWR_SRC_SEL	1	Y	3	Represents the HP system power source.

Cont'd on next page

GoBack

Table 6.3-1 – cont'd from previous page

Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_DCDC_VSET_EN	1	Y	N/A	Represents whether to use the default voltage configured by EFUSE_DCDC_VSET.
EFUSE_DIS_WDT	1	Y	2	Represents whether to disable the watchdog.
EFUSE_DIS_SWD	1	Y	2	Represents whether to disable the super watchdog.

Table 6.3-2 lists all key purposes and their values. Setting the eFuse parameter EFUSE_KEY_PURPOSE_*n* declares the purpose of KEY*n* (*n*: 0 ~ 5).

Table 6.3-2. Secure Key Purpose Values

Key Purpose Values	Purposes
0	User purposes
1	ECDSA_KEY
2	XTS_AES_256_KEY_1
3	XTS_AES_256_KEY_2
4	XTS_AES_128_KEY (flash/SRAM encryption and decryption)
5	HMAC Downstream mode (both JTAG and DS)
6	JTAG in HMAC Downstream mode
7	Digital Signature peripheral in HMAC Downstream mode
8	HMAC Upstream mode
9	SECURE_BOOT_DIGEST0 (secure boot key digest)
10	SECURE_BOOT_DIGEST1 (secure boot key digest)
11	SECURE_BOOT_DIGEST2 (secure boot key digest)
12	KM_INIT_KEY

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

Table 6.3-3. Parameters in BLOCK1 to BLOCK10

BLOCK	Parameters	Bit Width	Accessible by Hardware	Write Protection by EFUSE_WR_DIS Bit Number	Read Protection by EFUSE_RD_DIS Bit Number	Description
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC address
	EFUSE_MAC_EXT	16	N	20	N/A	Extended MAC address
	EFUSE_MAC_RESERVED	50	N	20	N/A	Reserved MAC address
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	System data
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	System data
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	User data
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 or user data
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 or user data
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 or user data
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 or user data
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 or user data
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 or user data
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	System data

Among these blocks, BLOCK4 ~ 9 can be used to store KEY0 ~ 5. Up to six 256-bit keys can be written into eFuse. Whenever a key is written, its purpose value should also be written (see table 6.3-2). For example, when a key for the JTAG function in HMAC Downstream mode is written to KEY3 (i.e., BLOCK7), its key purpose value 6 should also be written to EFUSE_KEY_PURPOSE_3.

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

BLOCK1 ~ BLOCK10 use the RS coding scheme, so there are some limitations on writing to these parameters. For more detailed information, please refer to Section 6.3.1.3 and Section 6.3.2.

6.3.1.1 EFUSE_WR_DIS

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

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

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

When the write protection bit of a parameter is set to 1, it means that this parameter is write-protected and none of its bits can be modified, with non-programmed bits always remaining 0 and programmed bits always remaining 1. That is to say, if a parameter is write-protected, it will always remain in this state and cannot be changed.

6.3.1.2 EFUSE_RD_DIS

Only the parameters in BLOCK4 ~ BLOCK10 can be set to be read-protected from users, as shown in column "Read Protection by EFUSE_RD_DIS Bit Number" of Table 6.3-3. After EFUSE_RD_DIS has been programmed, execute an eFuse read operation so the new values would take effect.

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

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

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

6.3.1.3 Data Storage

Internally, eFuse uses the hardware encoding scheme to protect data from corruption. The scheme and the encoding process are invisible to users.

All BLOCK0 parameters except for EFUSE_WR_DIS are stored with four backups, meaning each bit is stored four times. This backup scheme is not visible to users.

In BLOCK0, EFUSE_WR_DIS occupies 32 bits, and other parameters takes 152 bits each. So, the eFuse memory space occupied by BLOCK0 is 32 + 152 * 4 = 640 bits.

BLOCK1 ~ BLOCK10 use RS (44, 32) coding scheme that supports up to 6 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

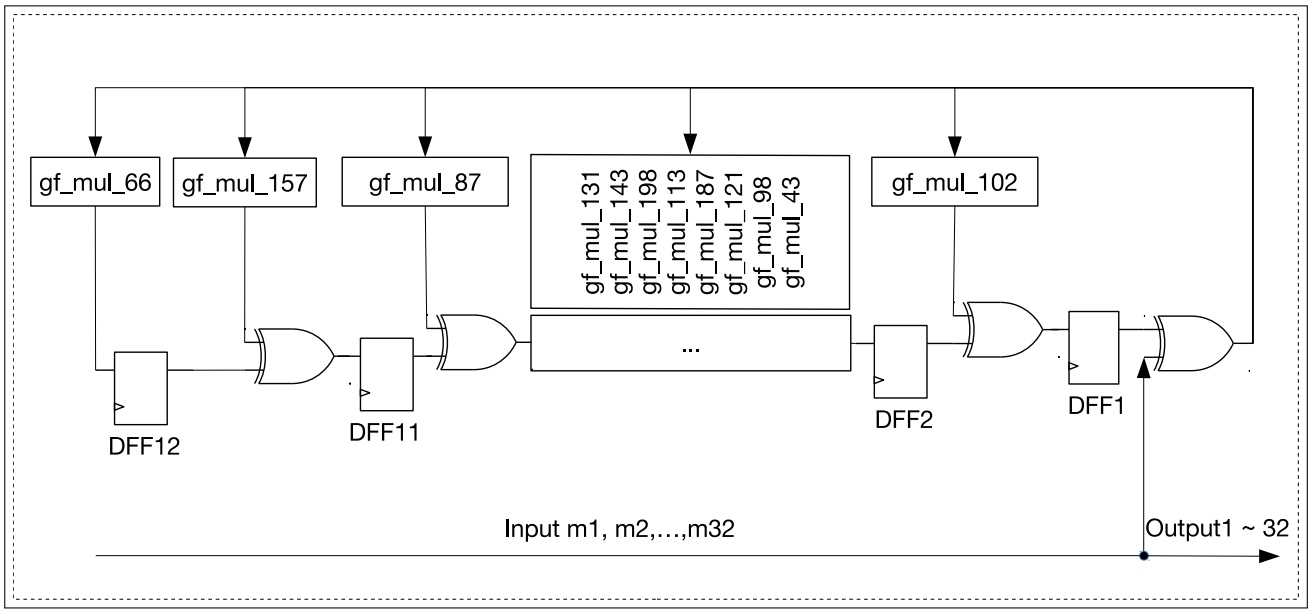


Figure 6.3-2. Shift Register Circuit (first 32 output)

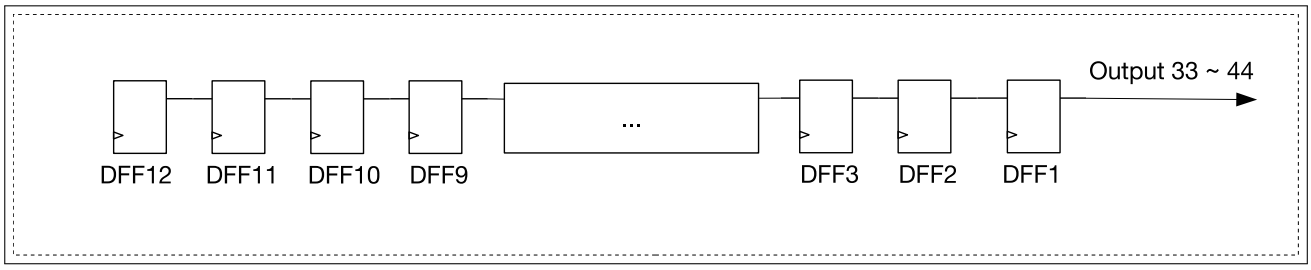


Figure 6.3-3. Shift Register Circuit (last 12 output)

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

- Bytes [0:31] are the data bytes itself
- Bytes [32:43] are the encoded parity bytes stored in 8-bit flip-flops DFF1, DFF2, ..., DFF12 (gf_mul_n is the result of multiplying a byte of data in $GF(2^8)$ by α^n, where n is an integer).

After that, the hardware programs into eFuse the 44-byte codeword consisting of the data bytes and the parity bytes. When the eFuse block is read, the eFuse controller automatically decodes the codeword and applies error correction if needed.

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

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

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

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

6.3.2 Programming of Parameters

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

Since there is a one-to-one correspondence between the reading data registers and the programming data registers (see table 6.3-4 for details), users can find out where the data to be programmed is located in programming registers by checking the parameter description and the parameter location in the corresponding read registers.

For example, if the user wants to program the parameter EFUSE_USB_DEVICE_EXCHG_PINS in BLOCK0 to 1, they can first search the reading data registers EFUSE_RD_REPEAT_DATA0 ~ 4_REG in BLOCK0 for where the parameter is located, namely, the 8th bit in EFUSE_RD_REPEAT_DATA0_REG. So, the user can set the 8th bit of EFUSE_PGM_DATA1_REG to 1 and follow the programming steps below. After the steps are completed, the corresponding bit in the eFuse memory will be programmed to 1.

Programming preparation

- Programming BLOCK0
  1. Set EFUSE_BLK_NUM to 0.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG the data to be programmed to BLOCK0.
     The data in EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG does not affect the programming of BLOCK0.
- BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.
- Programming BLOCK2 ~ 10
  1. Set EFUSE_BLK_NUM to the block number.
  2. Write into EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG the data to be programmed. Write into EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG the corresponding RS code.

Programming process

The process of programming parameters is as follows:

1. Configure the value of parameter EFUSE_BLK_NUM to determine the block to be programmed.
2. Write parameters to be programmed to registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
3. Make sure the eFuse programming voltage VDDQ is configured correctly as described in Section 6.3.4.
4. Configure the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5A5A.

5. Configure the field EFUSE_PGM_CMD of register EFUSE_CMD_REG to 1.
6. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a PGM_DONE interrupt. For more information on how to identify a PGM_DONE or READ_DONE interrupt, please see the end of Section 6.3.3.
7. Clear the parameters in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
8. Trigger an eFuse read operation (see Section 6.3.3) to update eFuse registers with the new values.
9. Check error record registers. If the values read in error record registers are not 0, the programming process should be performed again following above steps 1 ~ 7. Please check the following error record registers for different eFuse blocks:
    - BLOCK0: EFUSE_RD_REPEAT_ERR0_REG ~ EFUSE_RD_REPEAT_ERR4_REG
    - BLOCK1: EFUSE_MAC_SYS_ERR_NUM, EFUSE_MAC_SYS_FAIL
    - BLOCK2: EFUSE_SYS_PART1_ERR_NUM, EFUSE_SYS_PART1_FAIL
    - BLOCK3: EFUSE_USR_DATA_ERR_NUM, EFUSE_USR_DATA_FAIL
    - BLOCK4: EFUSE_KEY0_ERR_NUM, EFUSE_KEY0_FAIL
    - BLOCK5: EFUSE_KEY1_ERR_NUM, EFUSE_KEY1_FAIL
    - BLOCK6: EFUSE_KEY2_ERR_NUM, EFUSE_KEY2_FAIL
    - BLOCK7: EFUSE_KEY3_ERR_NUM, EFUSE_KEY3_FAIL
    - BLOCK8: EFUSE_KEY4_ERR_NUM, EFUSE_KEY4_FAIL
    - BLOCK9: EFUSE_KEY5_ERR_NUM, EFUSE_KEY5_FAIL
    - BLOCK10: EFUSE_SYS_PART2_ERR_NUM, EFUSE_SYS_PART2_FAIL

Limitations

In BLOCK0, each bit can be programmed separately. However, we recommend to minimize programming cycles and program all the bits of a parameter in one programming action. In addition, after all parameters controlled by a certain bit of EFUSE_WR_DIS are programmed, that bit should be immediately programmed. The programming of parameters controlled by a certain bit of EFUSE_WR_DIS, and the programming of the bit itself can even be completed at the same time in one programming action.

BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.

BLOCK2 ~ 10 can only be programmed once. Repeated programming is not allowed.

6.3.3 Reading of Parameters by Users

Users cannot read eFuse bits directly. The eFuse controller hardware reads all eFuse bits and stores the results to their corresponding registers in its memory space. Then, users can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in Table 6.3-4.

Table 6.3-4. Registers Information

BLOCK	Read Registers	Registers When Programming This Block
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_PART1_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_PART2_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating reading data registers

The eFuse controller reads eFuse memory to update corresponding registers. This read operation happens at system reset and can also be triggered manually by users as needed (e.g., if new eFuse values have been programmed). The process of triggering a read operation by users is as follows:

1. Configure the field EFUSE_OP_CODE in register EFUSE_CONF_REG to 0x5AA5.
2. Configure the field EFUSE_READ_CMD in register EFUSE_CMD_REG to 1.
3. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a READ_DONE interrupt. Information on how to identify a PGM_DONE or READ_DONE interrupt is provided below in this section.
4. Read the values of each parameter from eFuse memory.

The eFuse read registers will hold all values until the next read operation.

Error detection

The programming error record registers allows users to check the integrity of parameters stored in the eFuse memory. For instance, they can help detect whether the four-backup parameters are consistent and whether parameters protected by RS encoding are decoded successfully.

Registers EFUSE_RD_REPEAT_ERR0 ~ 3_REG indicate if there are any errors in programming parameters (except EFUSE_WR_DIS) to BLOCK0. The value 1 indicates an error is detected in programming the corresponding bit. The value 0 indicates no error.

Registers EFUSE_RD_RS_ERR0 ~ 1_REG store the number of corrected bytes as well as the result of RS decoding when eFuse controller reads BLOCK1 ~ BLOCK10.

The values of the above registers will be updated every time the reading data registers of eFuse controller have been updated.

Identifying completion of program or read operations

The methods to identify the completion of a program/read operation are described below. Please note that bit 1 corresponds to a program operation, and bit 0 corresponds to a read operation.

- Method one: Poll bit 1/0 in register EFUSE_INT_RAW_REG until it becomes 1, which represents the completion of a program/read operation.
- Method two:
  1. Set bit 1/0 in register EFUSE_INT_ENA_REG to 1 to enable the eFuse controller to post a PGM_DONE or READ_DONE interrupt.

2. Configure the Interrupt Matrix to enable the CPU to respond to eFuse interrupt signals. See Chapter 10 *Interrupt Matrix*.
3. Wait for the PGM_DONE or READ_DONE interrupt.
4. Set bit 1/0 in register EFUSE_INT_CLR_REG to 1 to clear the PGM_DONE or READ_DONE interrupt.

Note

When eFuse controller is updating its registers, it will use EFUSE_PGM_DATA*n*_REG (n=0, 1, ... ,7) again to store data. So please do not write important data into these registers before this updating process is initiated.
During the chip boot process, eFuse controller will automatically update data from eFuse memory into the registers that can be accessed by users. Users can get programmed eFuse data by reading corresponding registers. Thus, there is no need to update the reading data registers in such case.

6.3.4 eFuse VDDQ Timing

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

- EFUSE_DAC_NUM (the rising period of VDDQ): The default value of VDDQ is 2.5 V and the voltage increases by 0.01 V in each clock cycle. The default value of this parameter is 255.
- EFUSE_DAC_CLK_DIV (the clock divisor of VDDQ): The clock period to program VDDQ should be larger than 1 μs.
- EFUSE_PWR_ON_NUM (the power-up time for VDDQ): The programming voltage should be stabilized after this time, which means the value of this parameter should be configured to exceed the result of EFUSE_DAC_CLK_DIV times EFUSE_DAC_NUM.
- EFUSE_PWR_OFF_NUM (the power-out time for VDDQ): The value of this parameter should be larger than 10 μs.

Table 6.3-5. Configuration of Default VDDQ Timing Parameters

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

6.3.5 Parameters Used by Hardware Modules

Some hardware modules are directly connected to the eFuse peripheral in order to use the parameters that are marked with "Y" in columns "Accessible by Hardware" of Table 6.3-1 and Table 6.3-3. Users cannot intervene in this process.

6.4 Interrupts

ESP32-P4's eFuse controller can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- EFUSE_INT

There are several internal interrupt sources from eFuse controller that can generate the above interrupt signal. The interrupt sources from eFuse controller are listed with their trigger conditions and the resulted interrupt signal in Table 6.4-1.

Table 6.4-1. eFuse's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
EFUSE_PGM_DONE_INT	Programming of eFuse completes	EFUSE_INT
EFUSE_READ_DONE_INT	Reading of eFuse completes	

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 6.5 *Register Summary*.

6.5 Register Summary

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

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

Name	Description	Address	Access
Programming Data Registers			
EFUSE_PGM_DATA0_REG	Register 0 that stores data to be programmed	0x0000	R/W
EFUSE_PGM_DATA1_REG	Register 1 that stores data to be programmed	0x0004	R/W
EFUSE_PGM_DATA2_REG	Register 2 that stores data to be programmed	0x0008	R/W
EFUSE_PGM_DATA3_REG	Register 3 that stores data to be programmed	0x000C	R/W
EFUSE_PGM_DATA4_REG	Register 4 that stores data to be programmed	0x0010	R/W
EFUSE_PGM_DATA5_REG	Register 5 that stores data to be programmed	0x0014	R/W
EFUSE_PGM_DATA6_REG	Register 6 that stores data to be programmed	0x0018	R/W
EFUSE_PGM_DATA7_REG	Register 7 that stores data to be programmed	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Register 0 that stores the RS code to be programmed	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Register 1 that stores the RS code to be programmed	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Register 2 that stores the RS code to be programmed	0x0028	R/W
Reading Data Registers for BLOCK0			
EFUSE_RD_WR_DIS_REG	BLOCK0 data register 0 that represents whether programming of individual eFuse memory bit is disabled	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 data register 1	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	BLOCK0 data register 2	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	BLOCK0 data register 3	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	BLOCK0 data register 4	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	BLOCK0 data register 5	0x0040	RO
Reading Data Registers for BLOCK1			
EFUSE_RD_MAC_SYS_0_REG	BLOCK1 data register 0	0x0044	RO
EFUSE_RD_MAC_SYS_1_REG	BLOCK1 data register 1	0x0048	RO
EFUSE_RD_MAC_SYS_2_REG	BLOCK1 data register 2	0x004C	RO
EFUSE_RD_MAC_SYS_3_REG	BLOCK1 data register 3	0x0050	RO
EFUSE_RD_MAC_SYS_4_REG	BLOCK1 data register 4	0x0054	RO
EFUSE_RD_MAC_SYS_5_REG	BLOCK1 data register 5	0x0058	RO
Reading Data Registers for BLOCK2			
EFUSE_RD_SYS_PART1_DATA0_REG	Register 0 for BLOCK2 (system)	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	Register 1 for BLOCK2 (system)	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	Register 2 for BLOCK2 (system)	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	Register 3 for BLOCK2 (system)	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	Register 4 for BLOCK2 (system)	0x006C	RO

Name	Description	Address	Access
EFUSE_RD_SYS_PART1_DATA5_REG	Register 5 for BLOCK2 (system)	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	Register 6 for BLOCK2 (system)	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	Register 7 for BLOCK2 (system)	0x0078	RO
Reading Data Registers for BLOCK3			
EFUSE_RD_USR_DATA0_REG	Register 0 for BLOCK3 (user)	0x007C	RO
EFUSE_RD_USR_DATA1_REG	Register 1 for BLOCK3 (user)	0x0080	RO
EFUSE_RD_USR_DATA2_REG	Register 2 for BLOCK3 (user)	0x0084	RO
EFUSE_RD_USR_DATA3_REG	Register 3 for BLOCK3 (user)	0x0088	RO
EFUSE_RD_USR_DATA4_REG	Register 4 for BLOCK3 (user)	0x008C	RO
EFUSE_RD_USR_DATA5_REG	Register 5 for BLOCK3 (user)	0x0090	RO
EFUSE_RD_USR_DATA6_REG	Register 6 for BLOCK3 (user)	0x0094	RO
EFUSE_RD_USR_DATA7_REG	Register 7 for BLOCK3 (user)	0x0098	RO
Reading Data Registers for BLOCK4			
EFUSE_RD_KEY0_DATA0_REG	Register 0 for BLOCK7 (KEY0)	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	Register 1 for BLOCK7 (KEY0)	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	Register 2 for BLOCK7 (KEY0)	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	Register 3 for BLOCK7 (KEY0)	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	Register 4 for BLOCK7 (KEY0)	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	Register 5 for BLOCK7 (KEY0)	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	Register 6 for BLOCK7 (KEY0)	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	Register 7 for BLOCK7 (KEY0)	0x00B8	RO
Reading Data Registers for BLOCK5			
EFUSE_RD_KEY1_DATA0_REG	Register 0 for BLOCK5 (KEY1)	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	Register 1 for BLOCK5 (KEY1)	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	Register 2 for BLOCK5 (KEY1)	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	Register 3 for BLOCK5 (KEY1)	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	Register 4 for BLOCK5 (KEY1)	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	Register 5 for BLOCK5 (KEY1)	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	Register 6 for BLOCK5 (KEY1)	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	Register 7 for BLOCK5 (KEY1)	0x00D8	RO
Reading Data Registers for BLOCK6			
EFUSE_RD_KEY2_DATA0_REG	Register 0 for BLOCK6 (KEY2)	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	Register 1 for BLOCK6 (KEY2)	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	Register 2 for BLOCK6 (KEY2)	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	Register 3 for BLOCK6 (KEY2)	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	Register 4 for BLOCK6 (KEY2)	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	Register 5 for BLOCK6 (KEY2)	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	Register 6 for BLOCK6 (KEY2)	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	Register 7 for BLOCK6 (KEY2)	0x00F8	RO
Reading Data Registers for BLOCK7			
EFUSE_RD_KEY3_DATA0_REG	Register 0 for BLOCK7 (KEY3)	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	Register 1 for BLOCK7 (KEY3)	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	Register 2 for BLOCK7 (KEY3)	0x0104	RO

Name	Description	Address	Access
EFUSE_RD_KEY3_DATA3_REG	Register 3 for BLOCK7 (KEY3)	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	Register 4 for BLOCK7 (KEY3)	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	Register 5 for BLOCK7 (KEY3)	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	Register 6 for BLOCK7 (KEY3)	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	Register 7 for BLOCK7 (KEY3)	0x0118	RO
Reading Data Registers for BLOCK8			
EFUSE_RD_KEY4_DATA0_REG	Register 0 for BLOCK8 (KEY4)	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	Register 1 for BLOCK8 (KEY4)	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	Register 2 for BLOCK8 (KEY4)	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	Register 3 for BLOCK8 (KEY4)	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	Register 4 for BLOCK8 (KEY4)	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	Register 5 for BLOCK8 (KEY4)	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Register 6 for BLOCK8 (KEY4)	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Register 7 for BLOCK8 (KEY4)	0x0138	RO
Reading Data Registers for BLOCK9			
EFUSE_RD_KEY5_DATA0_REG	Register 0 for BLOCK9 (KEY5)	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	Register 1 for BLOCK9 (KEY5)	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	Register 2 for BLOCK9 (KEY5)	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	Register 3 for BLOCK9 (KEY5)	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	Register 4 for BLOCK9 (KEY5)	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	Register 5 for BLOCK9 (KEY5)	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	Register 6 for BLOCK9 (KEY5)	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	Register 7 for BLOCK9 (KEY5)	0x0158	RO
Reading Data Registers for BLOCK10			
EFUSE_RD_SYS_PART2_DATA0_REG	Register 0 for BLOCK10 (system)	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	Register 1 for BLOCK10 (system)	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	Register 2 for BLOCK10 (system)	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	Register 3 for BLOCK10 (system)	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	Register 4 for BLOCK10 (system)	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	Register 5 for BLOCK10 (system)	0x0170	RO
EFUSE_RD_SYS_PART2_DATA6_REG	Register 6 for BLOCK10 (system)	0x0174	RO
EFUSE_RD_SYS_PART2_DATA7_REG	Register 7 for BLOCK10 (system)	0x0178	RO
Error Report Registers for BLOCK0			
EFUSE_RD_REPEAT_ERR0_REG	Programming error record register 0 for BLOCK0	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	Programming error record register 1 for BLOCK0	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	Programming error record register 2 for BLOCK0	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	Programming error record register 3 for BLOCK0	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	Programming error record register 4 for BLOCK0	0x018C	RO
Error Report Registers for RS Block			
EFUSE_RD_RS_ERR0_REG	Programming error record register 0 for BLOCK1-10	0x01C0	RO

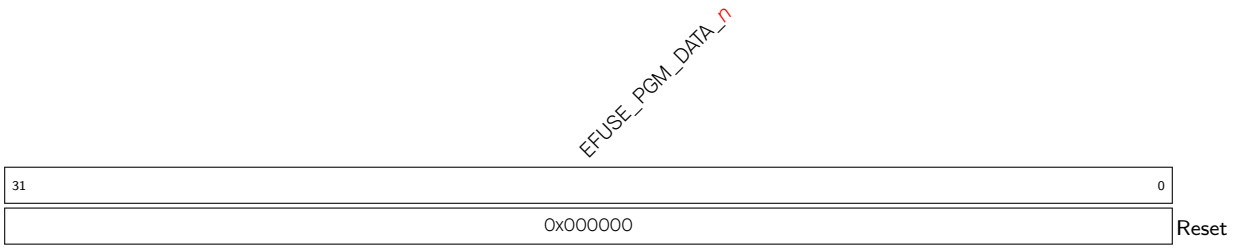
Name	Description	Address	Access
EFUSE_RD_RS_ERR1_REG	Programming error record register 1 for BLOCK1-10	0x01C4	RO
eFuse Clock Register			
EFUSE_CLK_REG	eFuse clock configuration register	0x01C8	R/W
eFuse Configuration Registers			
EFUSE_CONF_REG	Configures eFuse operation mode	0x01CC	R/W
EFUSE_DAC_CONF_REG	Configures the eFuse programming voltage	0x01E8	R/W
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	Configures eFuse programming timing parameters	0x01F0	R/W
EFUSE_WR_TIM_CONF2_REG	Configures eFuse programming timing parameters	0x01F4	R/W
EFUSE_WR_TIM_CONF0_RS_BYPASS_REG	Configures eFuse programming time parameters and RS bypass operation	0x01F8	varies
eFuse Status Register			
EFUSE_STATUS_REG	eFuse status register	0x01D0	RO
eFuse Command Registers			
EFUSE_CMD_REG	eFuse command register	0x01D4	varies
eFuse Interrupt Registers			
EFUSE_INT_RAW_REG	eFuse raw interrupt register	0x01D8	R/SS/WTC
EFUSE_INT_ST_REG	eFuse interrupt status register	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse interrupt enable register	0x01E0	R/W
EFUSE_INT_CLR_REG	eFuse interrupt clear register	0x01E4	WT
eFuse Version Register			
EFUSE_DATE_REG	eFuse version control register	0x01FC	R/W

6.6 Registers

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

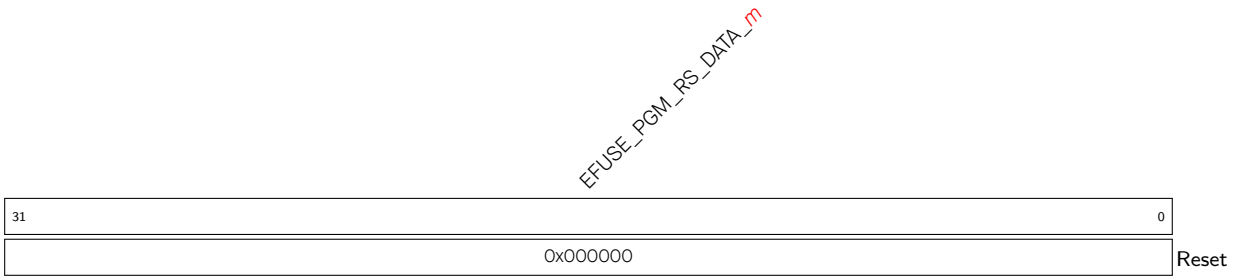
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 6.1. EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+0x4n*)**



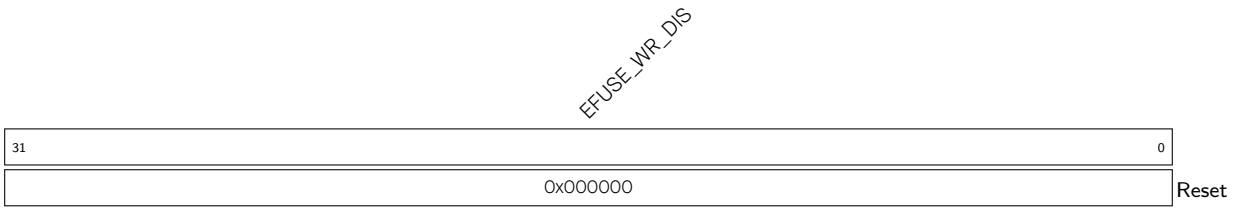
EFUSE_PGM_DATA_*n* Configures the *n*th 32-bit data to be programmed. (R/W)

Register 6.2. EFUSE_PGM_CHECK_VALUE*m*_REG (*m*: 0-2) (0x0020+0x4m*)**



EFUSE_PGM_RS_DATA_*m* Configures the *m*th 32-bit RS code to be programmed. (R/W)

Register 6.3. EFUSE_RD_WR_DIS_REG (0x002C)



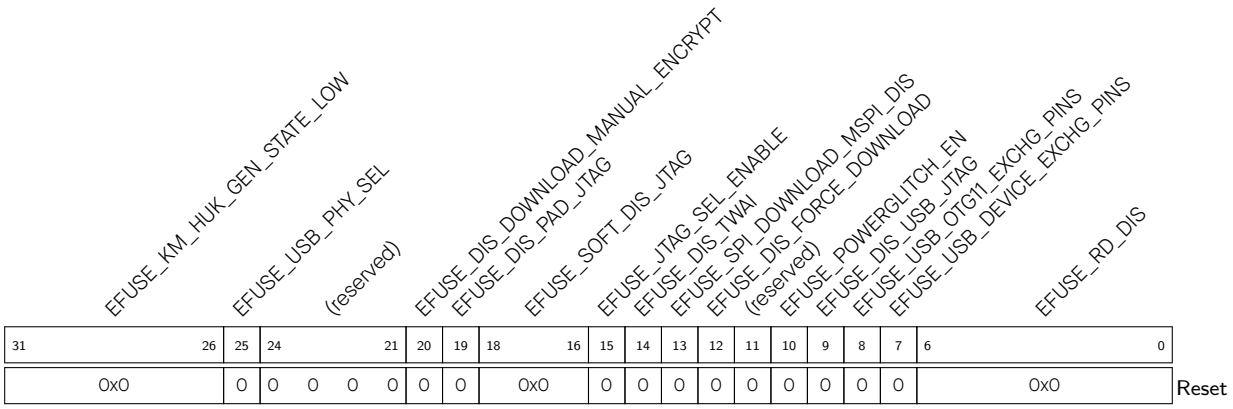
EFUSE_WR_DIS Represents whether programming of individual eFuse memory bit is disabled.

1: Disabled

0: Enabled

(RO)

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS Represents whether reading of individual eFuse block (BLOCK4 ~ BLOCK10) is disabled.
1: Disabled
0: Enabled
(RO)

EFUSE_USB_DEVICE_EXCHG_PINS Represents whether the USB Serial/JTAG pins D+ and D- are exchanged.
0: Not exchanged
1: Exchanged
(RO)

EFUSE_USB_OTG11_EXCHG_PINS Represents whether the USB FS pins D+ and D- are exchanged.
0: Not exchanged
1: Exchanged
(RO)

EFUSE_DIS_USB_JTAG Represents whether the USB-to-JTAG function in USB Serial/JTAG is disabled.
1: Disabled
0: Enabled
(RO)

EFUSE_POWERGLITCH_EN Represents whether power glitch detection is enabled.
0: Disabled
1: Enabled
(RO)

EFUSE_DIS_FORCE_DOWNLOAD Represents whether the function that forces chip into Download mode is disabled.
1: Disabled
0: Enabled
(RO)

Continued on the next page...

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

EFUSE_SPI_DOWNLOAD_MSPI_DIS Represents accessing MSPI flash/MSPI RAM by SYS AXI matrix is disabled during boot_mode_download.
1: Disabled
0: Enabled
(RO)

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

EFUSE_JTAG_SEL_ENABLE Represents whether the selection of a JTAG signal source through the strapping pin value is enabled when both EFUSE_DIS_PAD_JTAG and EFUSE_DIS_USB_JTAG are configured to 0. For more information, please refer to Chapter 9 *Chip Boot Control*.
1: Enabled
0: Disabled
(RO)

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

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

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Represents whether flash encryption is disabled (except in SPI boot mode).
1: Disabled
0: Enabled
(RO)

EFUSE_USB_PHY_SEL Represents the connection between USB Serial/JTAG Controller, OTG_FS and FS_PHY1, FS_PHY2 when LP_SYSREG_SW_HW_USB_PHY_SEL is 0.
0: FS_PHY1 is connected to USB Serial/JTAG Controller and FS_PHY2 is connected to OTG_FS.
1: FS_PHY2 is connected to USB Serial/JTAG Controller and FS_PHY1 is connected to OTG_FS.
(RO)

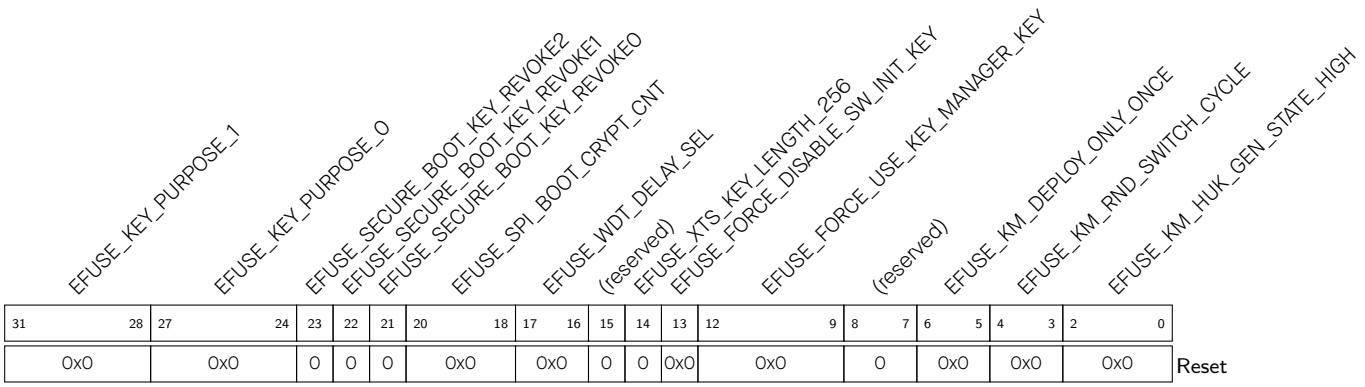
Continued on the next page...

Register 6.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

EFUSE_KM_HUK_GEN_STATE_LOW Represents the lower bits that determine the validity of the Hardware Unique Key (HUK) generation mode. It is combined with the EFUSE_KM_HUK_GEN_STATE_HIGH field to indicate whether the HUK generation mode is valid.
Odd count of bits with a value of 1 in those two fields: Invalid
Even count of bits with a value of 1 in those two fields: Valid
(RO)

Register 6.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_KM_HUK_GEN_STATE_HIGH Represents the higher bits that determine the validity of the Hardware Unique Key (HUK) generation mode. It is combined with the EFUSE_KM_HUK_GEN_STATE_LOW field to represent whether the HUK generation mode is valid.

Odd count of bits with a value of 1 in those two fields: Invalid

Even count of bits with a value of 1 in those two fields: Valid

(RO)

EFUSE_KM_RND_SWITCH_CYCLE Represents the cycle at which the Key Manager switches random numbers.

0: Controlled by the KEYMNG_RND_SWITCH_CYCLE register. For more information, please refer to Chapter 8 *Key Manager [to be added later]*

1: 8 Key Manager clock cycles

2: 16 Key Manager clock cycles

3: 32 Key Manager clock cycles

(RO)

EFUSE_KM_DEPLOY_ONLY_ONCE Represents whether the corresponding key can be deployed only once.

Bit0: Represents whether the ECDSA key can be deployed only once

0: The key can be deployed multiple times

1: The key can be deployed only once

Bit1: Represents whether the XTS key can be deployed only once

0: The key can be deployed multiple times

1: The key can be deployed only once

(RO)

Continued on the next page...

Register 6.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

Continued from the previous page...

EFUSE_FORCE_USE_KEY_MANAGER_KEY Represents whether the corresponding key must come from Key Manager.
Bit0: Represents whether the ECDSA key must come from Key Manager.
0: The key does not need to come from Key Manager
1: The key must come from Key Manager
Bit1: Represents whether the XTS key must come from Key Manager.
0: The key does not need to come from Key Manager
1: The key must come from Key Manager
Bit2 and Bit3: reserved
(RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY Represents whether to disable the use of the initialization key written by software and instead force use efuse_init_key.
0: Enable
1: Disable
(RO)

EFUSE_XTS_KEY_LENGTH_256 Represents which key is used for flash encryption.
0: XTS-256 key. Key length: 512 bits.
1: XTS-128 key. Key length: 256 bits.
(RO)

EFUSE_WDT_DELAY_SEL Represents RTC watchdog timeout threshold.
0: The originally configured STG0 threshold × 2
1: The originally configured STG0 threshold × 4
2: The originally configured STG0 threshold × 8
3: The originally configured STG0 threshold × 16
(RO)

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

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

Continued on the next page...

Register 6.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

Continued from the previous page...

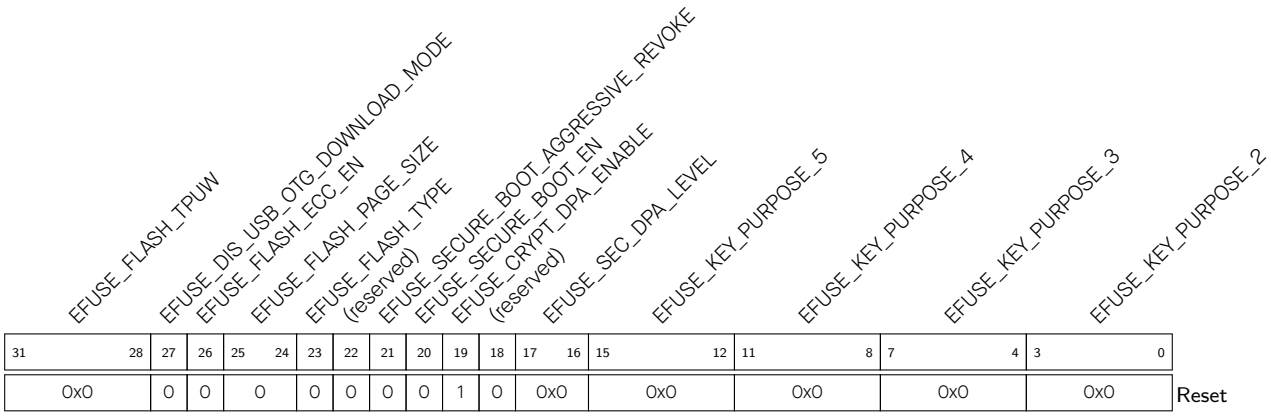
EFUSE_SECURE_BOOT_KEY_REVOKE1 Represents whether revoking Secure Boot key 1 is enabled.
1: Enabled
0: Disabled
(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 Represents whether revoking Secure Boot key 2 is enabled.
1: Enabled
0: Disabled
(RO)

EFUSE_KEY_PURPOSE_0 Represents the purpose of Key0. See Table 6.3-2. (RO)

EFUSE_KEY_PURPOSE_1 Represents the purpose of Key1. See Table 6.3-2. (RO)

Register 6.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 Represents the purpose of Key2. See Table 6.3-2. (RO)

EFUSE_KEY_PURPOSE_3 Represents the purpose of Key3. See Table 6.3-2. (RO)

EFUSE_KEY_PURPOSE_4 Represents the purpose of Key4. See Table 6.3-2. (RO)

EFUSE_KEY_PURPOSE_5 Represents the purpose of Key5. See Table 6.3-2. (RO)

EFUSE_SEC_DPA_LEVEL Represents the security level of anti-DPA attack. The level is adjusted by configuring the clock random frequency division mode.

0: Security level is SEC_DPA_OFF

1: Security level is SEC_DPA_LOW

2: Security level is SEC_DPA_MIDDLE

3: Security level is SEC_DPA_HIGH

For more information, please refer to Chapter 6 *System Registers [to be added later]* > Section 6.1 *Anti-DPA Attack Security Control [to be added later]*.

(RO)

EFUSE_CRYPT_DPA_ENABLE Represents whether defense against DPA attack is enabled.

1: Enabled

0: Disabled

(RO)

EFUSE_SECURE_BOOT_EN Represents whether Secure Boot is enabled.

1: Enabled

0: Disabled

(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE Represents whether aggressive revocation of Secure Boot is enabled.

1: Enabled

0: Disabled

(RO)

Continued on the next page...

Register 6.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

Continued from the previous page...

EFUSE_FLASH_TYPE Represents the type of the interfaced flash.
0: Four data lines
1: eight data lines
(RO)

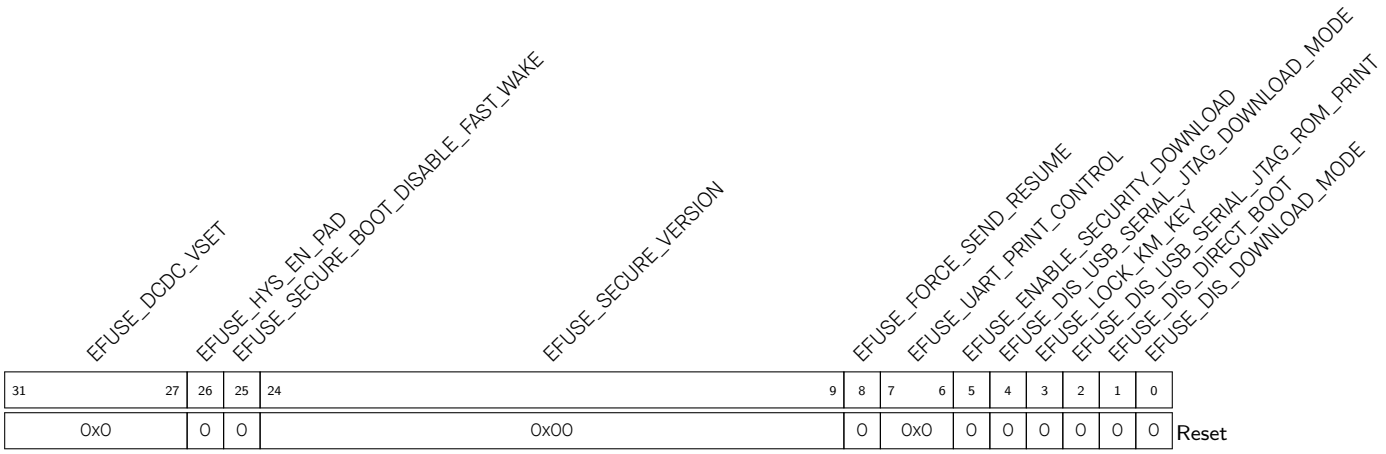
EFUSE_FLASH_PAGE_SIZE Represents flash page size.
0: 256 bytes
1: 512 bytes
2: 1024 bytes
3: 2048 bytes
(RO)

EFUSE_FLASH_ECC_EN Represents whether ECC is enabled during flash boot.
0: Disabled
1: Enabled
(RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE Represents whether download via USB-OTG is disabled.
0: Enabled
1: Disabled
(RO)

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

Register 6.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE Represents whether all Download modes are disabled.

1: Disabled

0: Enabled

(RO)

EFUSE_DIS_DIRECT_BOOT Represents whether direct boot mode is disabled.

1: Disabled

0: Enabled

(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT Represents whether print from USB-Serial-JTAG during ROM boot is disabled.

1: Disabled

0: Enabled

(RO)

EFUSE_LOCK_KM_KEY Represents whether the keys in the Key Manager are locked after deployment.

0: Not locked

1: Locked

(RO)

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

1: Disabled

0: Enabled

(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Represents whether security download is enabled. Only UART is supported for download. Reading/writing RAM or registers is not supported (i.e., stub download is not supported).

1: Enabled

0: Disabled

(RO)

Continued on the next page...

Register 6.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

Continued from the previous page...

EFUSE_UART_PRINT_CONTROL Represents the type of UART printing.

0: Force enable printing.

1: Enable printing when GPIO36 is reset at low level.

2: Enable printing when GPIO36 is reset at high level.

3: Force disable printing.

(RO)

EFUSE_FORCE_SEND_RESUME Represents whether ROM code is forced to send a resume command during SPI boot.

1: Forced.

0: Not forced.

(RO)

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

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE Represents whether FAST VERIFY ON WAKE is disabled when Secure Boot is enabled.

1: Disabled

0: Enabled

(RO)

EFUSE_HYS_EN_PAD Represents whether the hysteresis function of PAD0 – PAD27 is enabled.

1: Enabled

0: Disabled

(RO)

EFUSE_DCDC_VSET Represents the default DCDC voltage. (RO)

Register 6.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_OPXA_TIEH_SEL_0 Represents what controls the power supply for LDO VO0.

0: Controlled by the PMU_LDO_VO_TIEH_0 register field

1: Controlled by the SDMMC0 peripheral

2: The power supply is always high

3: Controlled by the SDMMC1 peripheral

(RO)

EFUSE_OPXA_TIEH_SEL_1 Represents what controls the power supply for LDO VO1.

0: Controlled by the PMU_LDO_VO_TIEH_1 register field

1: Controlled by the SDMMC0 peripheral

2: The power supply is always high

3: Controlled by the SDMMC1 peripheral

(RO)

EFUSE_OPXA_TIEH_SEL_2 Represents what controls the power supply for LDO VO2.

0: Controlled by the PMU_LDO_VO_TIEH_2 register field

1: Controlled by the SDMMC0 peripheral

2: The power supply is always high

3: Controlled by the SDMMC1 peripheral

(RO)

EFUSE_OPXA_TIEH_SEL_3 Represents what controls the power supply for LDO VO3.

0: Controlled by the PMU_LDO_VO_TIEH_3 register field

1: Controlled by the SDMMC0 peripheral

2: The power supply is always high

3: Controlled by the SDMMC1 peripheral

(RO)

Continued on the next page...

Register 6.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)

Continued from the previous page...

EFUSE_KM_DISABLE_DEPLOY_MODE Represents whether specific deploy modes (including random, AES, ECDH0, and ECDH1 modes, but not recovery and export modes) of the corresponding keys are disabled in the Key Manager.
Bit0: Represents whether specific deploy modes of the ECDSA key are disabled.
0: Enabled
1: Disabled
Bit1: Represents whether specific deploy modes of the XTS key are disabled.
0: Enabled
1: Disabled
Bit2: Reserved
Bit3: Reserved
(RO)

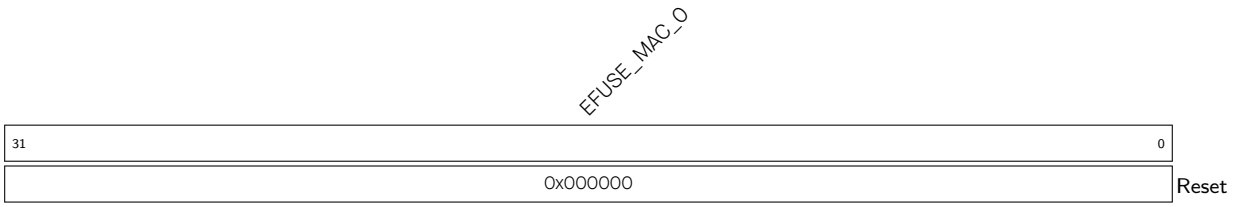
EFUSE_HP_PWR_SRC_SEL Represents the HP system power source.
0: LDO
1: DCDC
(RO)

EFUSE_DCDC_VSET_EN Represents whether to use the default voltage configured by EFUSE_DCDC_VSET.
0: Not use
1: Use
(RO)

EFUSE_DIS_WDT Represents whether to disable the watchdog.
0: Enable
1: Disable
(RO)

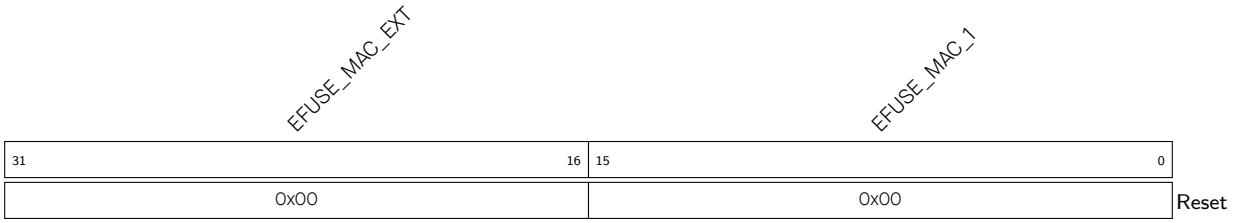
EFUSE_DIS_SWD Represents whether to disable the super watchdog.
0: Enable
1: Disable
(RO)

Register 6.9. EFUSE_RD_MAC_SYS_0_REG (0x0044)



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

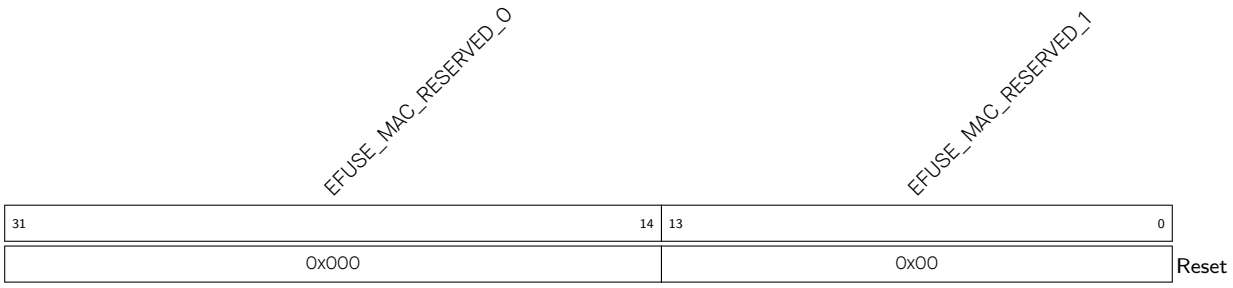
Register 6.10. EFUSE_RD_MAC_SYS_1_REG (0x0048)



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

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

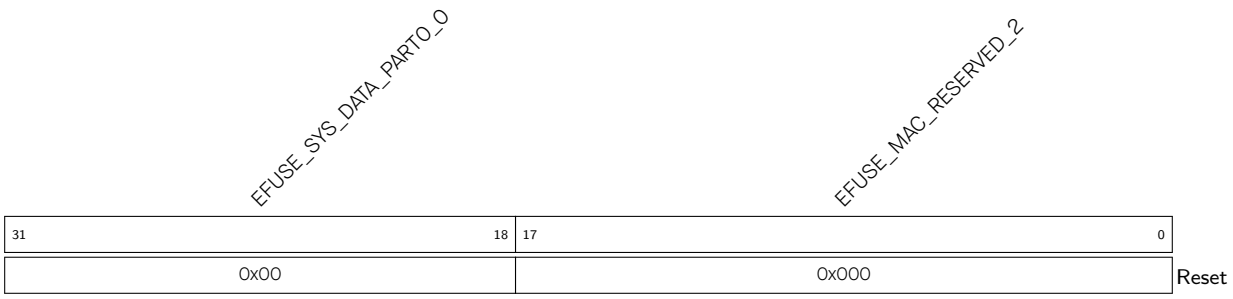
Register 6.11. EFUSE_RD_MAC_SYS_2_REG (0x004C)



EFUSE_MAC_RESERVED_1 Reserved. (RO)

EFUSE_MAC_RESERVED_0 Reserved. (RO)

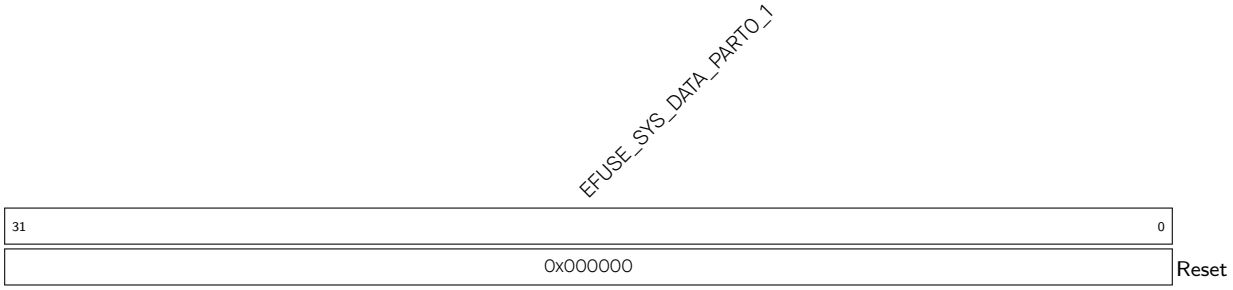
Register 6.12. EFUSE_RD_MAC_SYS_3_REG (0x0050)



EFUSE_MAC_RESERVED_2 Reserved. (RO)

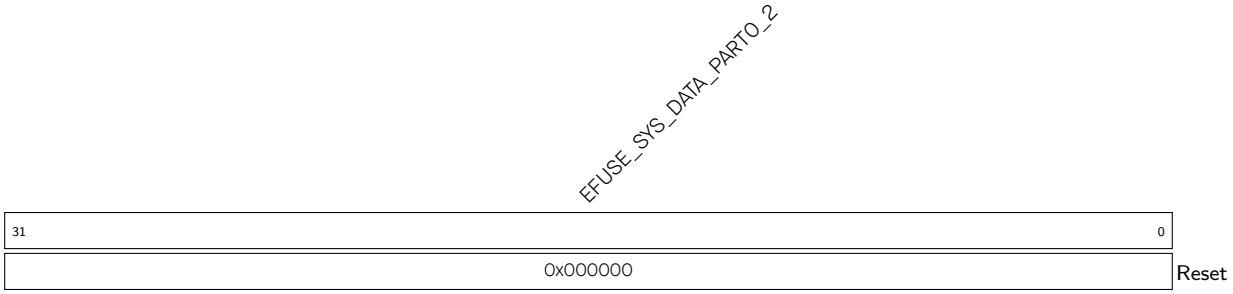
EFUSE_SYS_DATA_PART0_0 Represents the first 14 bits of the zeroth part of system data. (RO)

Register 6.13. EFUSE_RD_MAC_SYS_4_REG (0x0054)



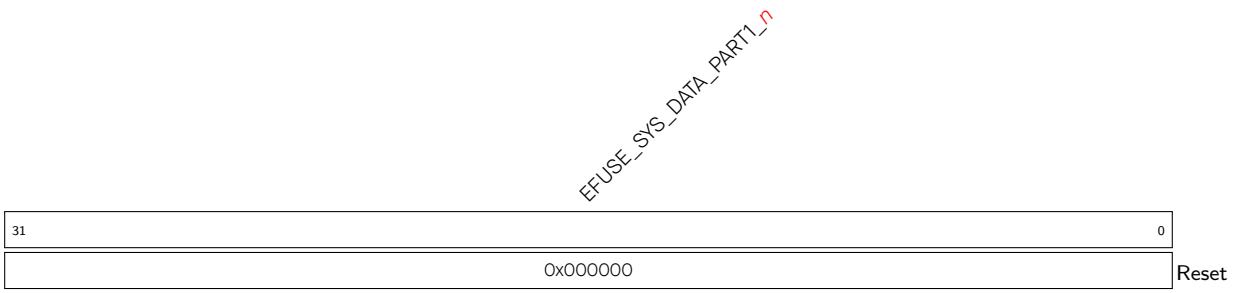
EFUSE_SYS_DATA_PART0_1 Represents the first 32 bits of the zeroth part of system data. (RO)

Register 6.14. EFUSE_RD_MAC_SYS_5_REG (0x0058)



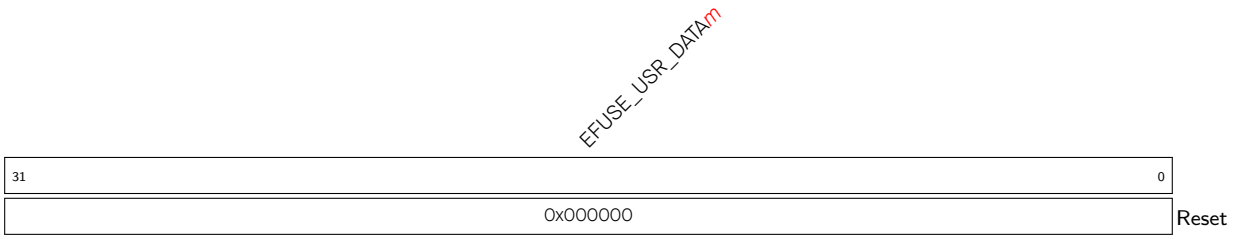
EFUSE_SYS_DATA_PART0_2 Represents the second 32 bits of the zeroth part of system data. (RO)

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



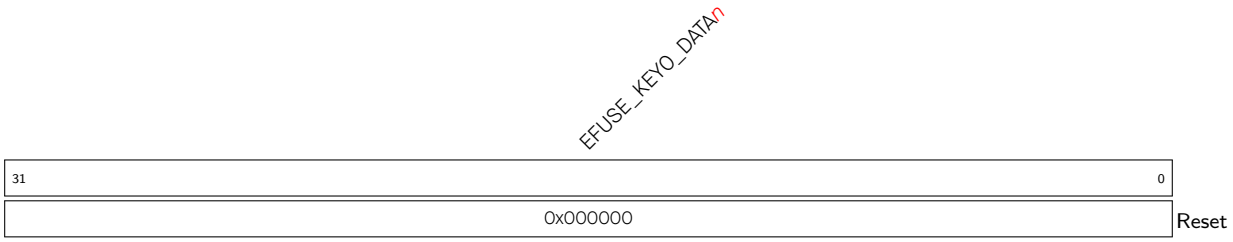
EFUSE_SYS_DATA_PART1_*n* Represents the *n*th 32 bits of the first part of system data. (RO)

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



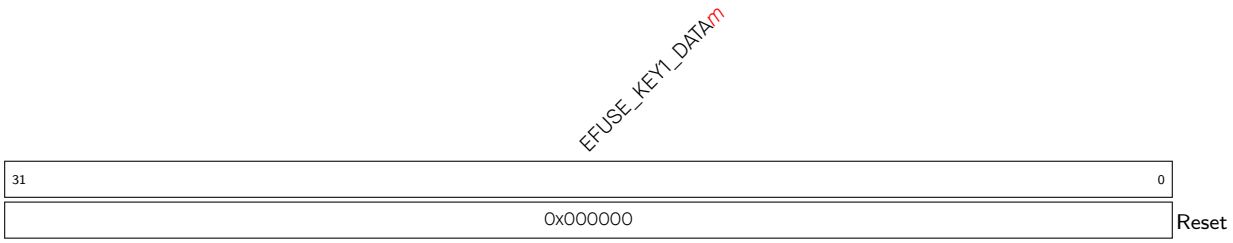
EFUSE_USR_DATA*m* Represents the *m*th 32 bits of BLOCK3 (user). (RO)

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



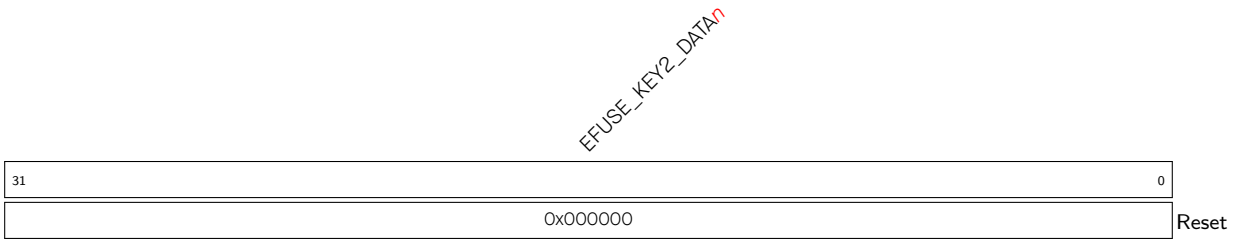
EFUSE_KEY0_DATA*n* Represents the *n*th 32 bits of KEY0. (RO)

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



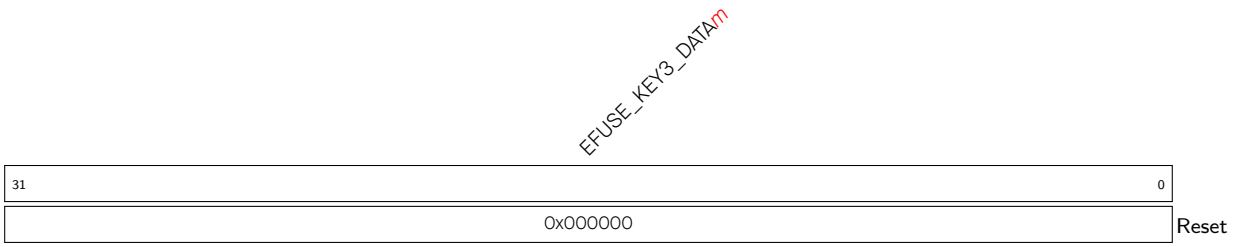
EFUSE_KEY1_DATA*m* Represents the *m*th 32 bits of KEY1. (RO)

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



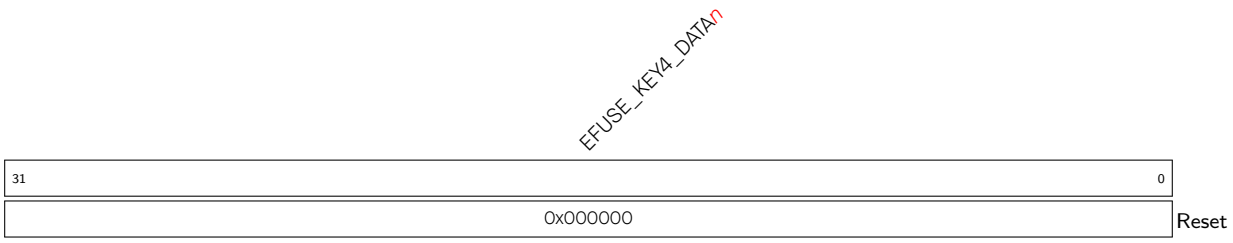
EFUSE_KEY2_DATA*n* Represents the *n*th 32 bits of KEY2. (RO)

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



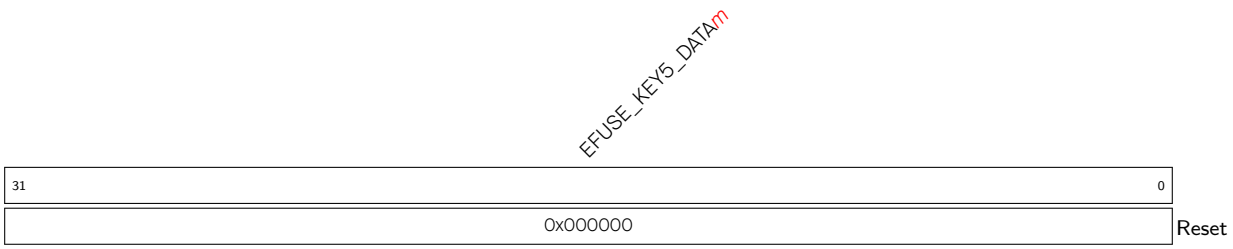
EFUSE_KEY3_DATA*m* Represents the *m*th 32 bits of KEY3. (RO)

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



EFUSE_KEY4_DATA*n* Represents the *n*th 32 bits of KEY4. (RO)

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



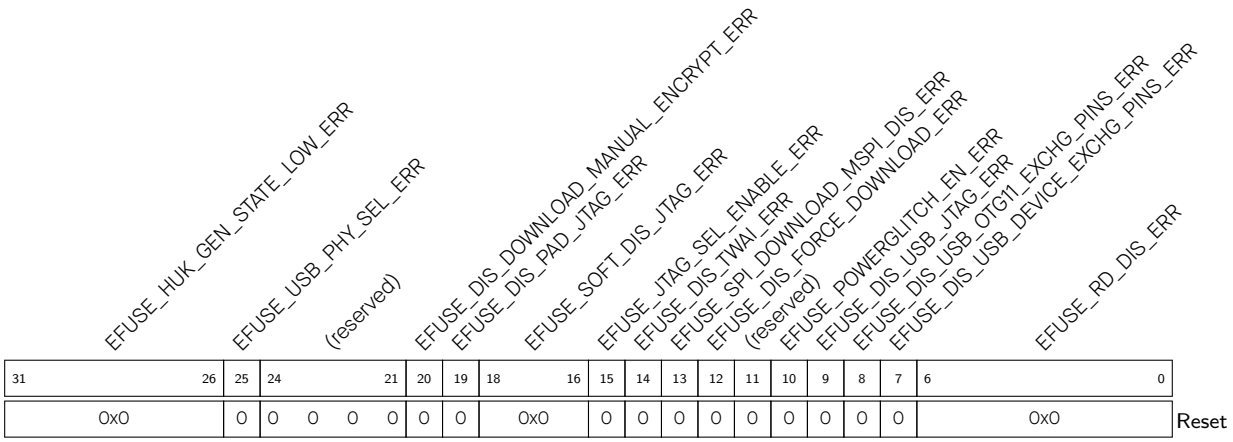
EFUSE_KEY5_DATA*m* Represents the *m*th 32 bits of KEY5. (RO)

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



EFUSE_SYS_DATA_PART2_*n* Represents the *n*th 32 bits of the 2nd part of system data. (RO)

Register 6.24. EFUSE_RD_REPEAT_ERR0_REG (0x017C)



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

EFUSE_DIS_USB_DEVICE_EXCHG_PINS_ERR This bit being 1 represents a programming error of DIS_USB_DEVICE_EXCHG_PINS. (RO)

EFUSE_DIS_USB_OTG11_EXCHG_PINS_ERR This bit being 1 represents a programming error of DIS_USB_OTG11_EXCHG_PINS. (RO)

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

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

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

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

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

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

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

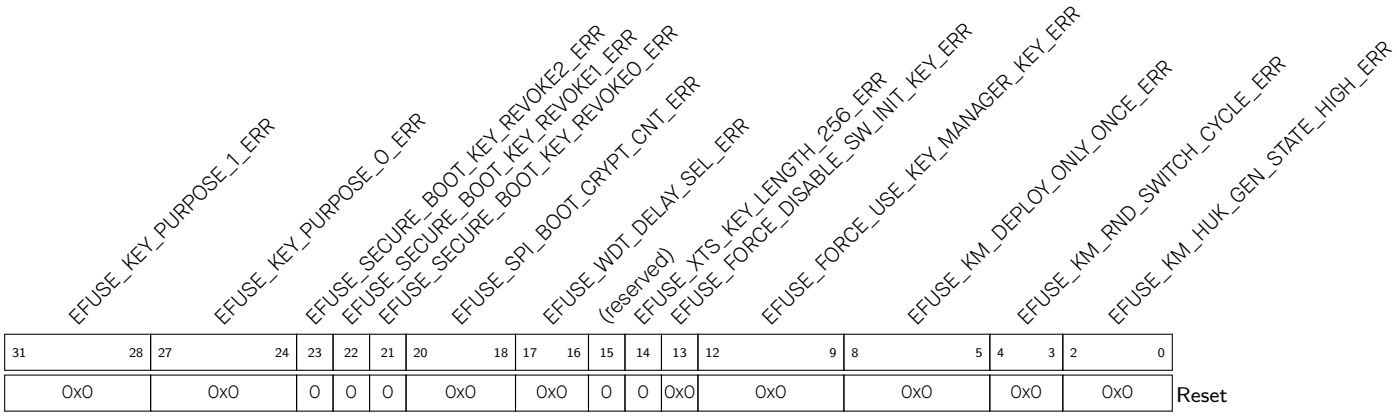
EFUSE_DIS_PAD_JTAG_ERR This bit being 1 represents a programming error of DIS_PAD_JTAG. (RO)

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

EFUSE_USB_PHY_SEL_ERR This bit being 1 represents a programming error of USB_PHY_SEL. (RO)

EFUSE_HUK_GEN_STATE_LOW_ERR Any bit of this field being 1 represents a programming error of HUK_GEN_STATE_LOW. (RO)

Register 6.25. EFUSE_RD_REPEAT_ERR1_REG (0x0180)



EFUSE_KM_HUK_GEN_STATE_HIGH_ERR Any bit of this field being 1 represents a programming error of HUK_GEN_STATE_HIGH. (RO)

EFUSE_KM_RND_SWITCH_CYCLE_ERR Any bit of this field being 1 represents a programming error of KM_RND_SWITCH_CYCLE. (RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_ERR Any bit of this field being 1 represents a programming error of KM_DEPLOY_ONLY_ONCE. (RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_ERR Any bit of this field being 1 represents a programming error of FORCE_USE_KEY_MANAGER_KEY. (RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY_ERR This bit being 1 represents a programming error of FORCE_DISABLE_SW_INIT_KEY. (RO)

EFUSE_XTS_KEY_LENGTH_256_ERR This bit being 1 represents a programming error of XTS_KEY_LENGTH_256. (RO)

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

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

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

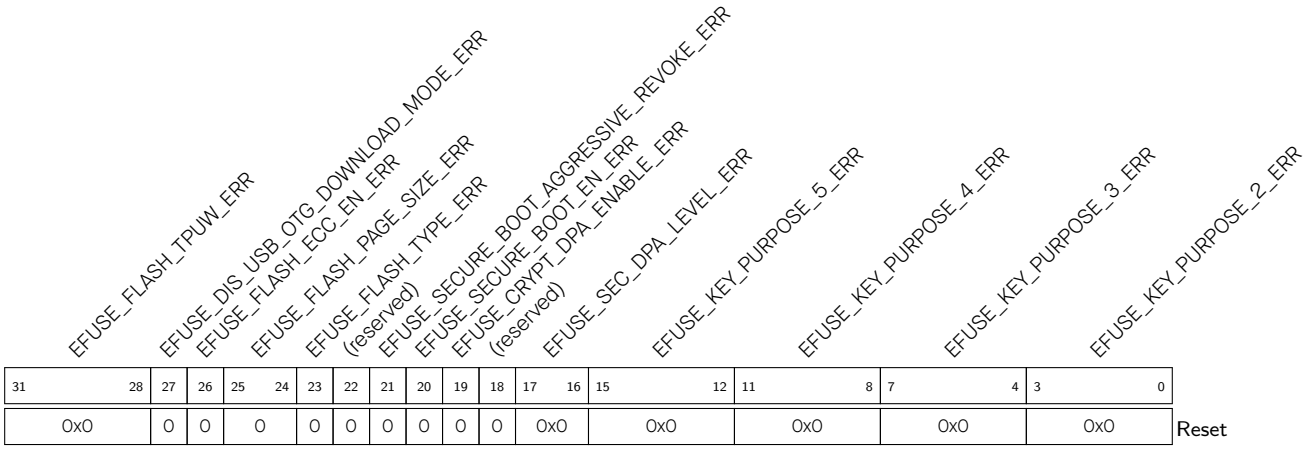
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR This bit being 1 represents a programming error of SECURE_BOOT_KEY_REVOKE1. (RO)

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

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

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

Register 6.26. EFUSE_RD_REPEAT_ERR2_REG (0x0184)



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

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

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

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

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

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

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

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

EFUSE_FLASH_TYPE_ERR This bit being 1 represents a programming error of FLASH_TYPE. (RO)

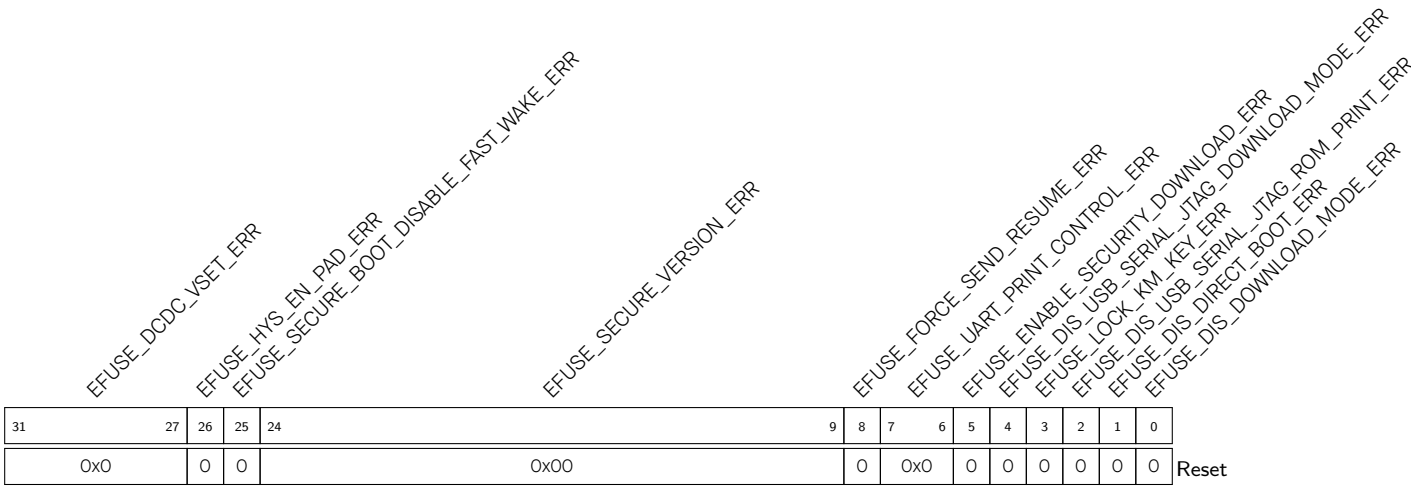
EFUSE_FLASH_PAGE_SIZE_ERR This bit being 1 represents a programming error of FLASH_PAGE_SIZE. (RO)

EFUSE_FLASH_ECC_EN_ERR This bit being 1 represents a programming error of FLASH_ECC_EN. (RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE_ERR This bit being 1 represents a programming error of DIS_USB_OTG_DOWNLOAD_MODE. (RO)

EFUSE_FLASH_TPUW_ERR This bit being 1 represents a programming error of FLASH_TPUW. (RO)

Register 6.27. EFUSE_RD_REPEAT_ERR3_REG (0x0188)



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

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

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR This bit being 1 represents a programming error of DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR. (RO)

EFUSE_LOCK_KM_KEY_ERR This bit being 1 represents a programming error of EFUSE_LOCK_KM_KEY. (RO)

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

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

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

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

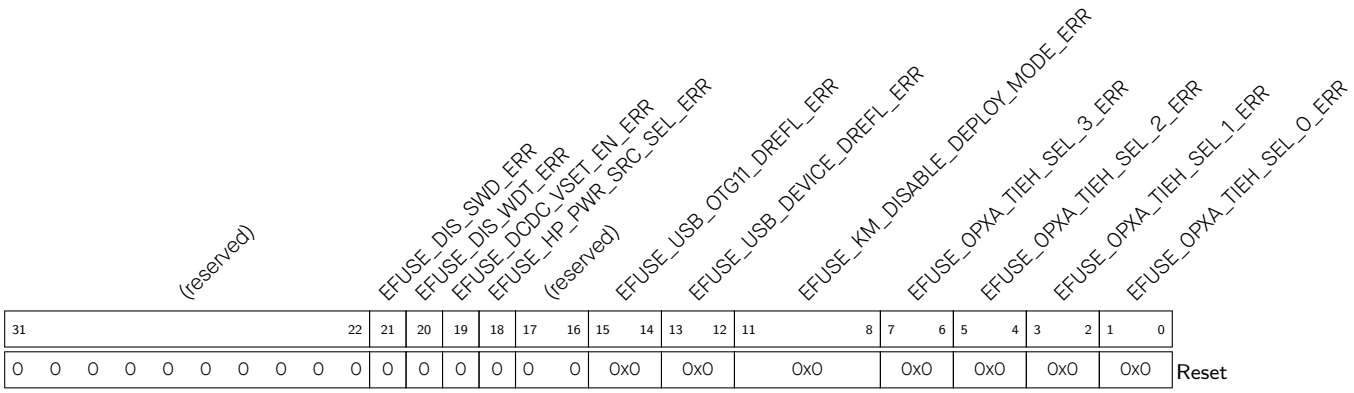
EFUSE_SECURE_VERSION_ERR Any bit of this field being 1 represents a programming error of SECURE_VERSION. (RO)

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

EFUSE_HYS_EN_PAD_ERR This bit being 1 represents a programming error of HYS_EN_PAD. (RO)

EFUSE_DCDC_VSET_ERR Any bit of this field being 1 represents a programming error of DCDC_VSET. (RO)

Register 6.28. EFUSE_RD_REPEAT_ERR4_REG (0x018C)



EFUSE_OPXA_TIEH_SEL_0_ERR Any bit of this field being 1 represents a programming error of OPXA_TIEH_SEL_0. (RO)

EFUSE_OPXA_TIEH_SEL_1_ERR Any bit of this field being 1 represents a programming error of OPXA_TIEH_SEL_1. (RO)

EFUSE_OPXA_TIEH_SEL_2_ERR Any bit of this field being 1 represents a programming error of OPXA_TIEH_SEL_2. (RO)

EFUSE_OPXA_TIEH_SEL_3_ERR Any bit of this field being 1 represents a programming error of OPXA_TIEH_SEL_3. (RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_ERR Any bit of this field being 1 represents a programming error of KM_DISABLE_DEPLOY_MODE_ERR. (RO)

EFUSE_USB_DEVICE_DREFL_ERR Any bit of this field being 1 represents a programming error of USB_DEVICE_DREFL. (RO)

EFUSE_USB_OTG11_DREFL_ERR Any bit of this field being 1 represents a programming error of USB_OTG11_DREFL. (RO)

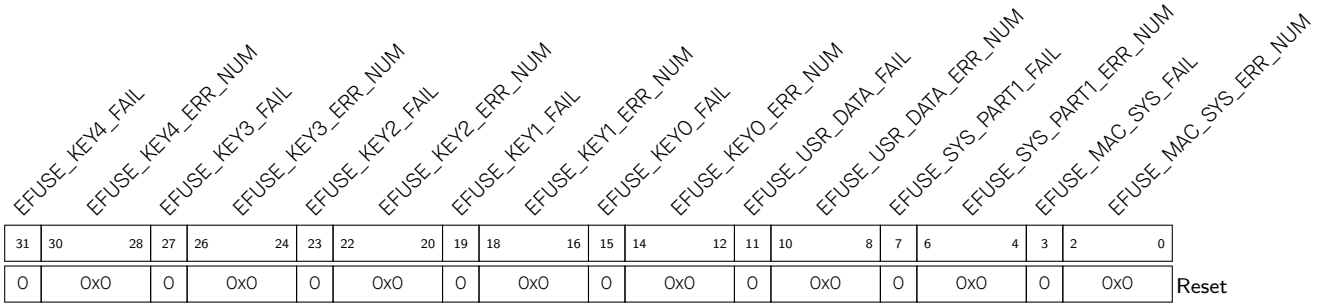
EFUSE_HP_PWR_SRC_SEL_ERR This bit being 1 represents a programming error of HP_PWR_SRC_SEL. (RO)

EFUSE_DCDC_VSET_EN_ERR This bit being 1 represents a programming error of DCDC_VSET_EN. (RO)

EFUSE_DIS_WDT_ERR This bit being 1 represents a programming error of DIS_WDT. (RO)

EFUSE_DIS_SWD_ERR This bit being 1 represents a programming error of DIS_SWD. (RO)

Register 6.29. EFUSE_RD_RS_ERR0_REG (0x01C0)



EFUSE_MAC_SYS_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_MAC_SYS_FAIL Represents whether programming MAC_SYS failed.

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

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

(RO)

EFUSE_SYS_PART1_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_SYS_PART1_FAIL Represents whether programming system part1 data failed.

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

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

(RO)

EFUSE_USR_DATA_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_USR_DATA_FAIL Represents whether programming user data failed.

0: No failure and the user data is reliable.

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

(RO)

EFUSE_KEY0_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY0_FAIL Represents whether programming key0 data failed.

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

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

(RO)

EFUSE_KEY1_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY1_FAIL Represents whether programming key1 data failed.

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

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

(RO)

EFUSE_KEY2_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY2_FAIL Represents whether programming key2 data failed.

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

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

(RO)

Continued on the next page...

Register 6.29. EFUSE_RD_RS_ERR0_REG (0x01C0)

Continued from the previous page...

EFUSE_KEY3_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY3_FAIL Represents whether programming key3 data failed.

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

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

(RO)

EFUSE_KEY4_ERR_NUM Represents the number of error bytes. (RO)

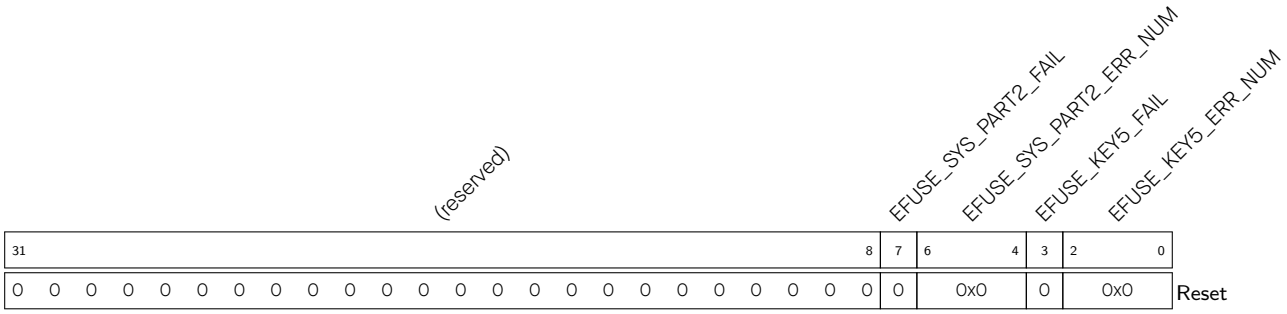
EFUSE_KEY4_FAIL Represents whether programming key4 data failed.

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

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

(RO)

Register 6.30. EFUSE_RD_RS_ERR1_REG (0x01C4)



EFUSE_KEY5_ERR_NUM Represents the number of error bytes. (RO)

EFUSE_KEY5_FAIL Represents whether programming key5 data failed.

0: No failure and the data of key5 is reliable.

1: Programming key5 failed and the number of error bytes is over 6.

(RO)

EFUSE_SYS_PART2_ERR_NUM Represents the number of error bytes. (RO)

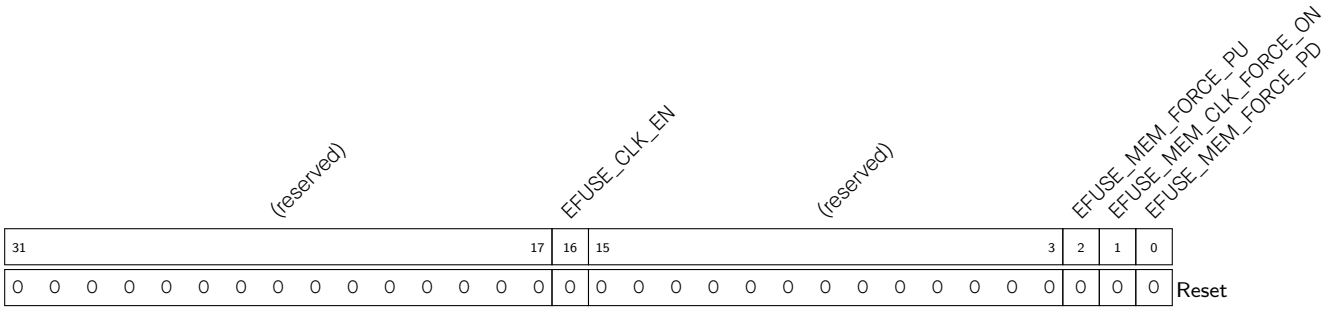
EFUSE_SYS_PART2_FAIL Represents whether programming system part2 data failed.

0: No failure and the data of system part2 is reliable.

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

(RO)

Register 6.31. EFUSE_CLK_REG (0x01C8)



EFUSE_MEM_FORCE_PD Configures whether to force power down eFuse SRAM.

1: Force

0: No effect

(R/W)

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

1: Force activate

0: No effect

(R/W)

EFUSE_MEM_FORCE_PU Configures whether to force power up eFuse SRAM.

1: Force

0: No effect

(R/W)

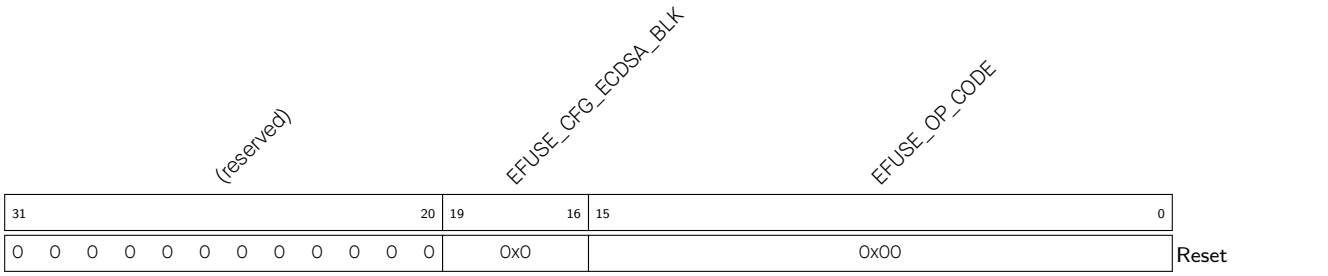
EFUSE_CLK_EN Configures whether to force enable eFuse register configuration clock signal.

1: Force

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

(R/W)

Register 6.32. EFUSE_CONF_REG (0x01CC)



EFUSE_OP_CODE Configures operation command type.

0x5A5A: Program operation command

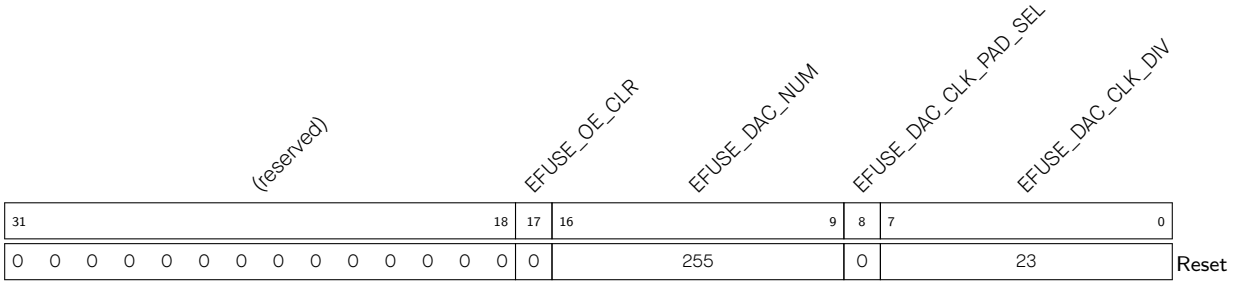
0x5AA5: Read operation command

Other values: No effect

(R/W)

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

Register 6.33. EFUSE_DAC_CONF_REG (0x01E8)



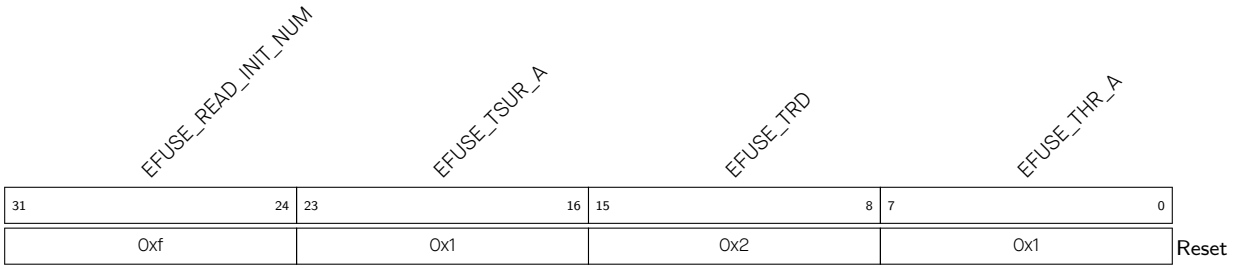
EFUSE_DAC_CLK_DIV Configures the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_CLK_PAD_SEL Don't care. (R/W)

EFUSE_DAC_NUM Configures clock cycles for programming voltage to rise. Measurement unit: a clock cycle divided by EFUSE_DAC_CLK_DIV. (R/W)

EFUSE_OE_CLR Configures whether to reduce the power supply of the programming voltage.
0: Not reduce.
1: Reduce.
(R/W)

Register 6.34. EFUSE_RD_TIM_CONF_REG (0x01EC)



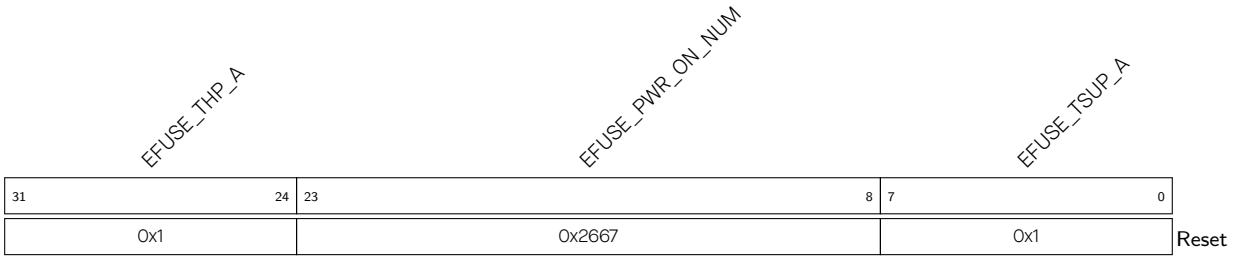
EFUSE_THR_A Configures the read hold time. Measurement unit: One cycle of the eFuse core clock. (R/W)

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

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

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

Register 6.35. EFUSE_WR_TIM_CONF1_REG (0x01F0)

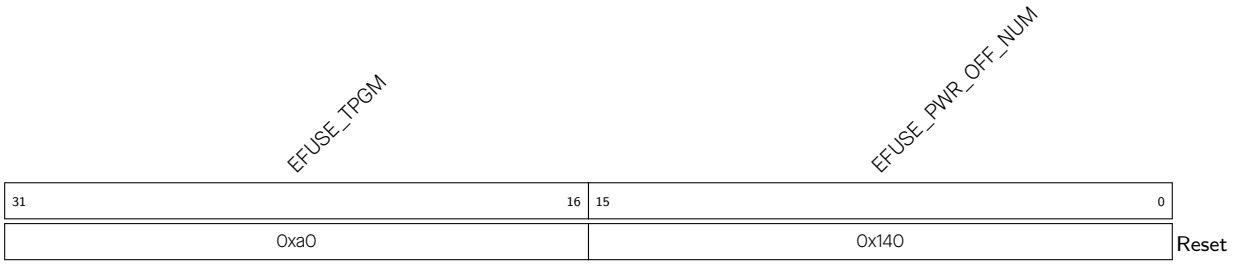


EFUSE_TSUP_A Configures the programming setup time. Measurement unit: One cycle of the eFuse core clock. (R/W)

EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. Measurement unit: One cycle of the eFuse core clock. (R/W)

EFUSE_THP_A Configures the programming hold time. Measurement unit: One cycle of the eFuse core clock. (R/W)

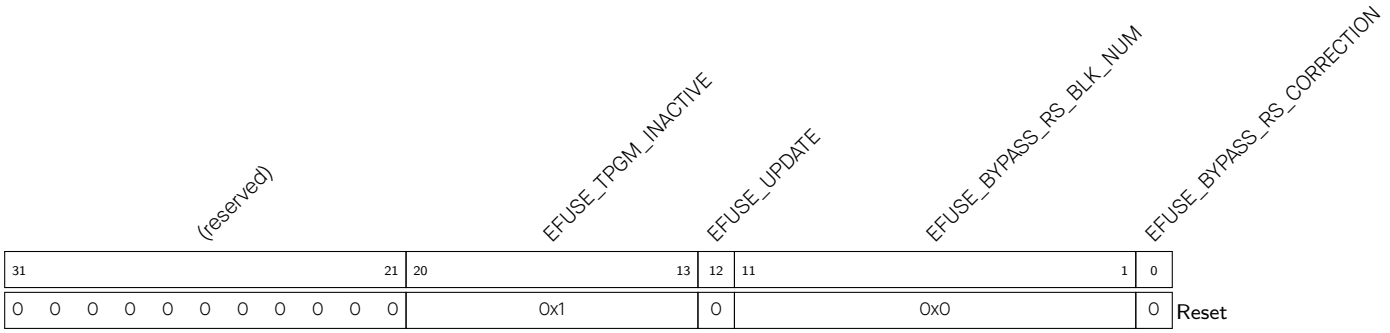
Register 6.36. EFUSE_WR_TIM_CONF2_REG (0x01F4)



EFUSE_PWR_OFF_NUM Configures the power outage time for VDDQ. Measurement unit: One cycle of the eFuse core clock. (R/W)

EFUSE_TPGM Configures the active programming time. Measurement unit: One cycle of the eFuse core clock. (R/W)

Register 6.37. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01F8)



EFUSE_BYPASS_RS_CORRECTION Configures whether to bypass the Reed-Solomon (RS) correction step.

0: Not bypass

1: Bypass

(R/W)

EFUSE_BYPASS_RS_BLK_NUM Configures which block number to bypass the Reed-Solomon (RS) correction step. (R/W)

EFUSE_UPDATE Configures whether to update multi-bit register signals.

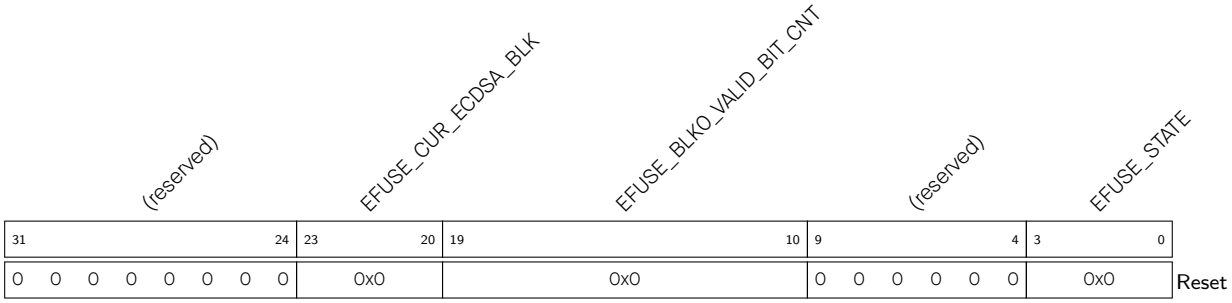
1: Update

0: No effect

(WT)

EFUSE_TPGM_INACTIVE Configures the inactive programming time. Measurement unit: One cycle of the eFuse core clock. (R/W)

Register 6.38. EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE Represents the state of the eFuse state machine.

0: Reset state, the initial state after power-up

1: Idle state

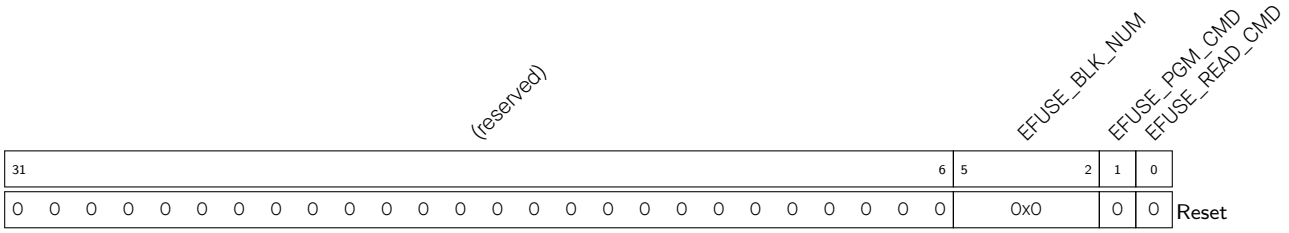
Other values: Non-idle state

(RO)

EFUSE_BLK0_VALID_BIT_CNT Represents the number of block valid bit. (RO)

EFUSE_CUR_ECDSA_BLK Represents which block is used for ECDSA key output. (RO)

Register 6.39. EFUSE_CMD_REG (0x01D4)



EFUSE_READ_CMD Configures whether to send read commands.

1: Send

0: No effect

(R/W/SC)

EFUSE_PGM_CMD Configures whether to send programming commands.

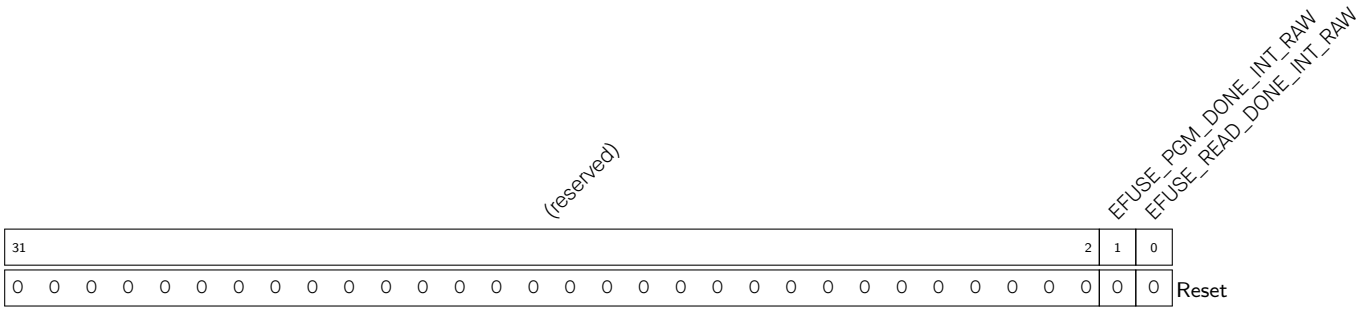
1: Send

0: No effect

(R/W/SC)

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

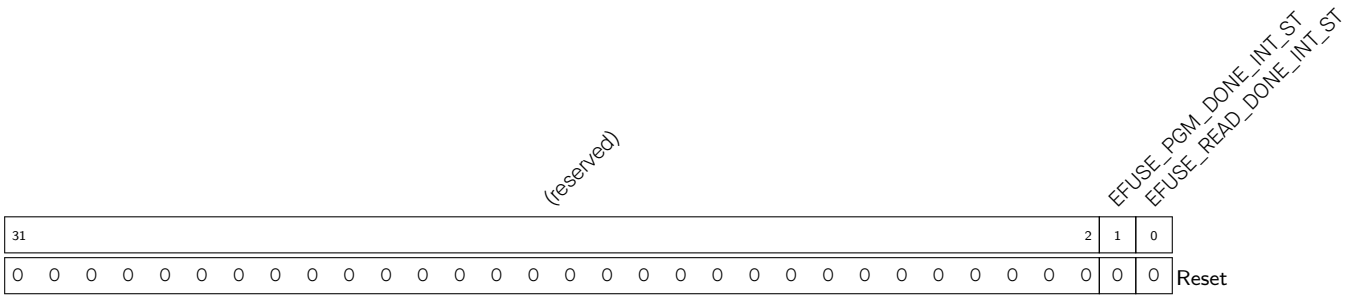
Register 6.40. EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW The raw interrupt status of EFUSE_READ_DONE_INT. (R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW The raw interrupt status of EFUSE_PGM_DONE_INT. (R/SS/WTC)

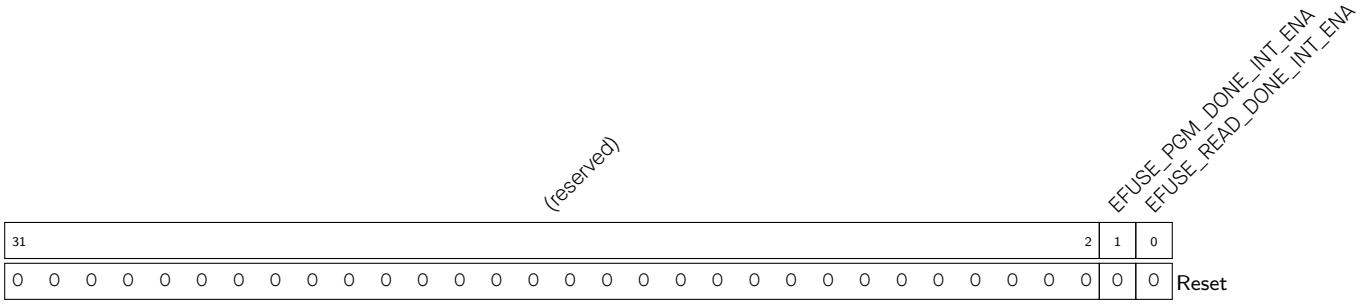
Register 6.41. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST The masked interrupt status of EFUSE_READ_DONE_INT. (RO)

EFUSE_PGM_DONE_INT_ST The masked interrupt status of EFUSE_PGM_DONE_INT. (RO)

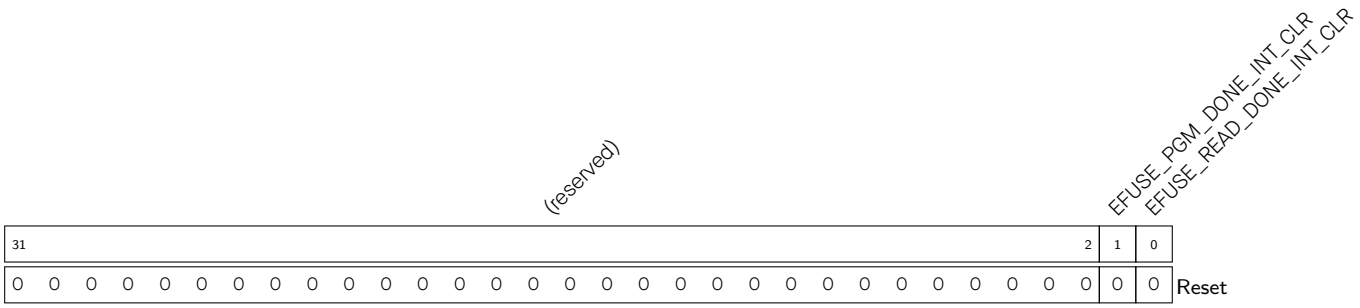
Register 6.42. EFUSE_INT_ENA_REG (0x01E0)



EFUSE_READ_DONE_INT_ENA Write 1 to enable EFUSE_READ_DONE_INT. (R/W)

EFUSE_PGM_DONE_INT_ENA Write 1 to enable EFUSE_PGM_DONE_INT. (R/W)

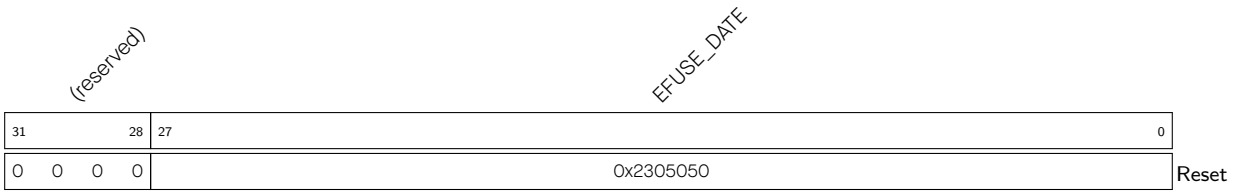
Register 6.43. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR Write 1 to clear EFUSE_READ_DONE_INT. (WT)

EFUSE_PGM_DONE_INT_CLR Write 1 to clear EFUSE_PGM_DONE_INT. (WT)

Register 6.44. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Version control register. (R/W)

Part IV

System Component

Encompassing a range of system-level functionalities, this part describes components related to system boot, clocks, GPIO, timers, watchdogs, debug assistance, event and interrupt handling, low-power management, and various system registers.

Chapter 7

GPIO Matrix and IO MUX

7.1 Overview

GPIO matrix is a hardware structure that allows dynamic remapping between the GPIO pins and peripheral input/output signals to provide greater flexibility. IO MUX (Input/output multiplexing) is a technology that allows the same pin to perform different functions at different times, allowing dynamic switching of pin functions through configuration. ESP32-P4 provides two groups of IO MUX and GPIO matrix:

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

The ESP32-P4 chip features 55 GPIO pins, including 16 low-power (LP) GPIO pins and 39 high-performance (HP) GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal.

- Through HP GPIO matrix and HP IO MUX, HP peripheral input signals can be from any GPIO pins, and HP peripheral output signals can be routed to any GPIO pins.
- Through LP GPIO matrix and LP IO MUX, LP peripheral input signals can be from any LP GPIO pins, and LP peripheral output signals can be routed to any LP GPIO pins.

Together these modules provide highly configurable I/O. The 55 GPIO pins are numbered from GPIO0 to GPIO54.

- LP GPIO pins (GPIO0 ~ GPIO15) can be used by either HP or LP peripherals.
- HP GPIO pins (GPIO16 ~ GPIO54) can be used only by HP peripherals.

The ESP32-P4 chip has dedicated pins for external flash and in-package PSRAM. Such pins can not be used for other purpose. See *ESP32-P4 Datasheet* > Section *Pin Mapping Between Chip and Flash/PSRAM*.

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

7.2 Features

7.2.1 HP GPIO Matrix and HP IO MUX

HP GPIO matrix has the following features:

- A full-switching matrix between HP peripheral input/output signals and the GPIO pins
- 222 HP peripheral input signals sourced from the input of any GPIO pins
- 232 HP peripheral output signals routed to the output of any GPIO pins

- Signal synchronization for HP peripheral inputs based on **HP IO MUX operating clock**
- GPIO Filter hardware for input signal filtering
- Glitch Filter hardware for second-time filtering on input signal
- Sigma delta modulated (SDM) output
- GPIO simple input and output
- HP GPIO Wakeup

HP IO MUX has the following features:

- Control of 55 GPIOs (GPIO0 ~ GPIO54) for HP peripherals.
- A configuration register IO_MUX_GPIO*n*_REG provided for each GPIO pin, to control the pin's input/output, pull-up/pull-down, drive strength, and function selection.
- Better high-frequency digital performance achieved by routing some digital signals (SPI, EMAC) directly from HP IO MUX to peripherals.

7.2.2 LP GPIO Matrix and LP IO MUX

LP GPIO matrix has the following features:

- A full-switching matrix between the LP peripheral input/output signals and the LP GPIO pins
- 14 LP peripheral input signals sourced from the input of any LP GPIO pins
- 14 LP peripheral output signals routed to the output of any LP GPIO pins
- GPIO Filter hardware for input signal filtering
- GPIO simple input and output
- LP GPIO Wakeup

LP IO MUX has the following feature:

- Control of 16 LP GPIO pins (GPIO0 ~ GPIO15) for LP peripherals.
- A configuration register LP_IOMUX_PAD*n*_REG provided for each LP GPIO pin, to control the pin's input/output, pull-up/pull-down, drive strength, function selection, and IO MUX selection.

7.3 Architectural Overview

Figure 7.3-1 shows in details how HP GPIO matrix, HP IO MUX, LP GPIO matrix, and LP IO MUX route signals from pins to peripherals, and from peripherals to pins.

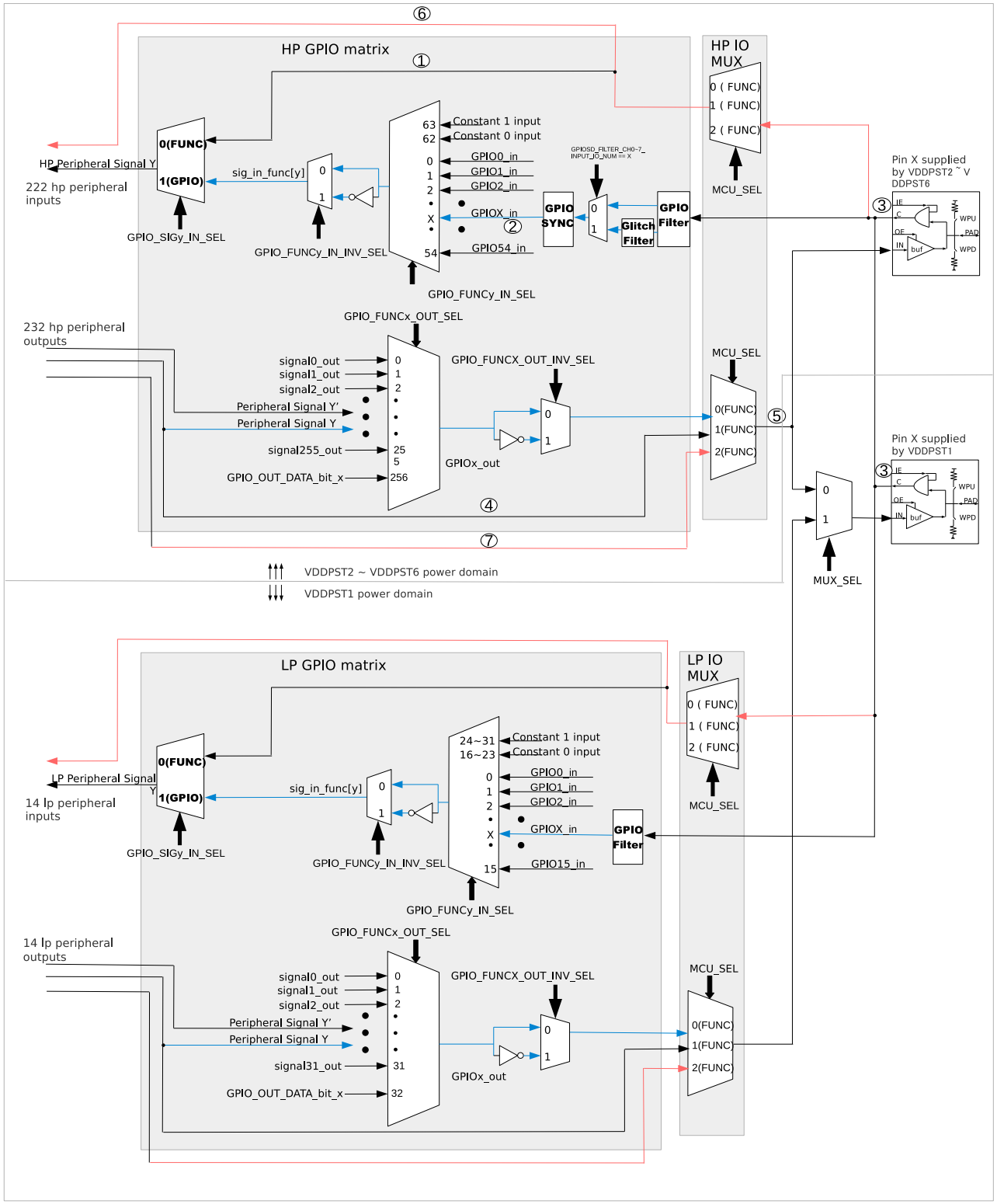


Figure 7.3-1. Architecture of HP GPIO Matrix, HP IO MUX, LP GPIO Matrix, and LP IO MUX

The following points explain the areas marked numerically in the figure above, taking the HP system as an example:

- ① Part of peripheral input signals (marked “yes” in column “Direct input through HP IO MUX” in Table

7.12-1) can be routed to the peripherals via HP IO MUX, or via HP GPIO matrix, while the other input signals can only be routed to the peripherals via HP GPIO matrix.

- ② There are only 55 inputs from GPIO SYNC to HP GPIO matrix, since ESP32-P4 provides 55 GPIO pins in total.
- ③ The pins supplied by VDDPST1 ~ VDDPST6 are controlled by the signals: IE, OE, HYS, WPU, and WPD.
- ④ Part of peripheral outputs (marked "yes" in column "Direct output through HP IO MUX" in Table 7.12-1) can be routed to pins via HP IO MUX or via HP GPIO matrix, while the other output signals can only be routed to pins via HP GPIO matrix.
- ⑤ There are 55 outputs (corresponding to GPIO pin *X*: 0 ~ 54) from HP GPIO matrix to HP IO MUX.
- ⑥ There are peripheral inputs only through HP IO MUX.
- ⑦ There are peripheral outputs only through HP IO MUX.

Figure 7.3-2 shows the internal structure of a pad, which is an electrical interface between the chip logic and the GPIO pin. The structure is applicable to all 55 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals. For the configuration of these signals, see IO_MUX_GPIO*x*_REG orLP_IOMUX_PAD*x*_REG.

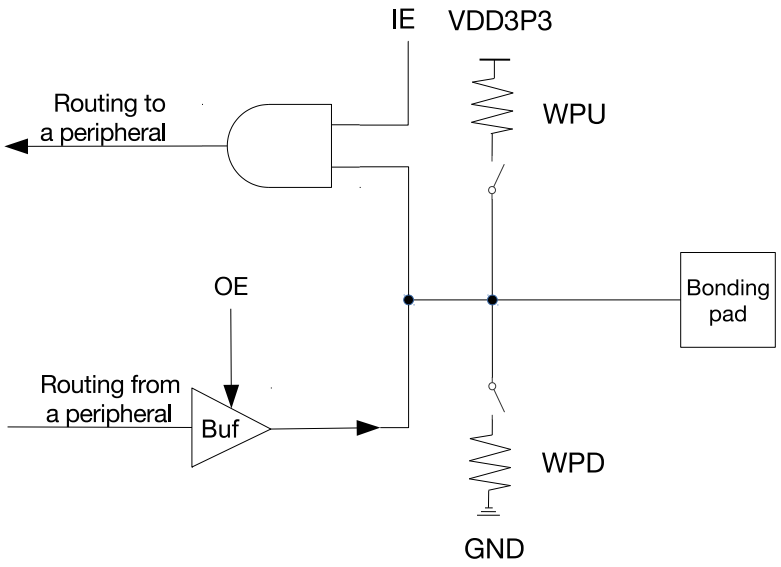


Figure 7.3-2. Internal Structure of a Pad

- IE: input enable
- OE: output enable
- WPU: internal weak pull-up resistor
- WPD: internal weak pull-down resistor
- Bonding pad: a terminal point of the chip logic used to make a physical connection from the chip die to GPIO pin in the chip package

7.4 Peripheral Input via GPIO Matrix

7.4.1 Overview

To receive a peripheral input signal via HP GPIO matrix,

- configure the matrix to source the peripheral input signal from one of the 55 GPIOs (0 ~ 54), see Table 7.12-1.
- configure the peripheral signal to receive input signal via HP GPIO matrix.
- configure the GPIO pin to be controlled by HP IO MUX.

For detailed configuration, see Figure 7.3-1 and Section 7.4.7.

To receive a peripheral input signal via LP GPIO matrix,

- configure the matrix to source the peripheral input signal from one of the 16 GPIOs (0 ~ 15), see Table 7.13-1.
- configure the peripheral signal to receive input signal via LP GPIO matrix.
- configure the LP GPIO pin to be controlled by LP IO MUX.

For detailed configuration, see Figure 7.3-1 and Section 7.4.7.

As shown in Figure 7.3-1, when GPIO matrix is used to input a signal from the pin, all external input signals are sourced from the GPIO pins and then filtered by the GPIO Filter, as shown in Step 2 in Section 7.4.7.

The Glitch Filter is only available in HP GPIO matrix. The Glitch Filter hardware can filter eight of the output signals from the GPIO Filter, and the other unselected signals go directly to the GPIO SYNC hardware, as shown in Step 3 in Section 7.4.7.

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

7.4.2 Signal Synchronization

Only HP GPIO matrix supports this signal synchronization function. Figure 7.4-1 shows the functionality of GPIO SYNC. In the figure, negative sync and positive sync mean GPIO input is synchronized on falling edge and on rising edge of HP IO MUX operating clock respectively.

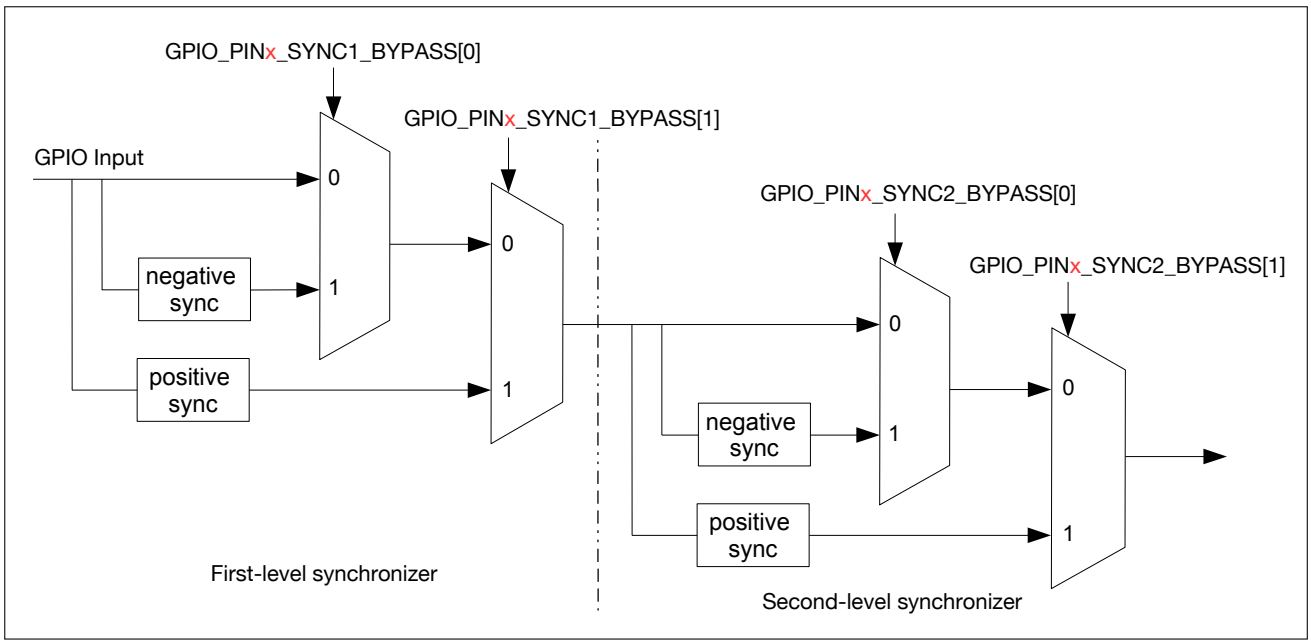


Figure 7.4-1. GPIO Input Synchronized on Rising Edge or on Falling Edge of HP IO MUX Operating Clock

The synchronization function is disabled by default by the synchronizer. But when an asynchronous peripheral signal is connected to the pin, the signal should be synchronized by the two-level synchronizer (i.e., the first-level synchronizer and the second-level synchronizer as shown in Figure 7.4-1) to lower the probability of causing metastability.

7.4.3 GPIO Filter

Both the HP GPIO matrix and LP GPIO matrix support GPIO Filter function. When this function is enabled, only signals with a valid width of more than two clock cycles can be sampled, as illustrated in Figure 7.4-2.

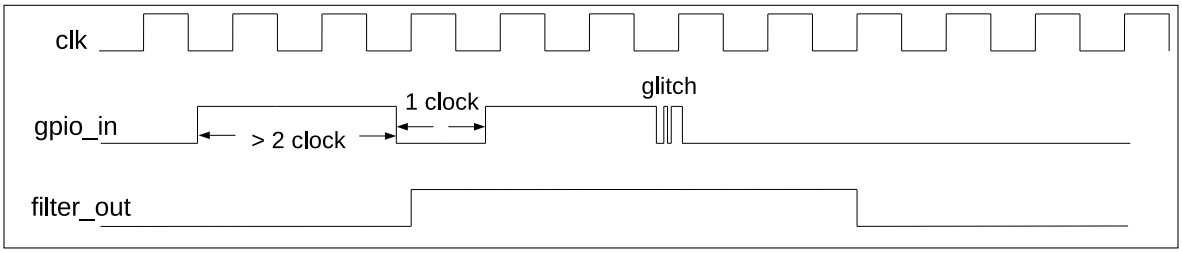


Figure 7.4-2. GPIO Filter Timing of GPIO Input Signals

7.4.4 Glitch Filter

The Glitch Filter function is exclusive to the HP GPIO matrix. The Glitch Filter hardware supports eight channels, each of which selects one signal from the 55 (0 ~ 54) output signals generated by the GPIO Filter hardware. It then conducts a second round of filtering on the selected signal. This Glitch Filter hardware can be used to filter slow-speed signals. For more information, see Step 3 in Section 7.4.7.1.

7.4.5 Simple GPIO Input

Both the HP GPIO matrix and LP GPIO matrix support the Simple GPIO Input function. Enabling this function allows the direct reading of the input value of a GPIO pin at any time, without the need to route the GPIO input to any peripherals.

For HP GPIO matrix, to implement simple GPIO input, follow the steps below:

- Set IO_MUX_GPIO*x*_MCU_IE in IO_MUX_GPIO*x*_REG, to enable pin input.
- Read the GPIO input from GPIO_IN_REG[*x*].

For LP GPIO matrix, to implement simple GPIO input, follow the steps below:

- Set LP_IOMUX_PAD*x*_FUN_IE in LP_IO_MUX_PAD*x*_REG, to enable pin input.
- Read the GPIO input from LP_GPIO_IN_REG[*x*].

7.4.6 GPIO Wakeup

7.4.6.1 HP GPIO Wakeup

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

GPIO0 ~ GPIO54 can be used to generate HP GPIO wakeup by enabling GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0 ~ 54).

HP GPIO wakeup supports both high-voltage sensitive and low-voltage sensitive wakeups, with specific conditions based on the GPIO configuration.

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

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

[1] *x* ranges from 0 to 54.

[2] If the register is configured to any other value, HP GPIO wakeup feature is disabled.

7.4.6.2 LP GPIO Wakeup

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

GPIO0 ~ GPIO15 can be used to generate LP GPIO wakeup by enabling LP_GPIO_PIN*n*_WAKEUP_ENABLE (*n*: 0 ~ 15).

LP GPIO wakeup supports posedge sensitive, negedge sensitive, both posedge and negedge sensitive, high-voltage sensitive and low-voltage sensitive wakeups, with specific conditions based on the GPIO configuration.

Table 7.4-2. LP GPIO Wakeup Signal Trigger and Clear Conditions

LP_GPIO_PIN*x*_INT_TYPE[1] [,2]	Wakeup is generated when	Wakeup is cleared when
1	the input GPIO toggles from low-level to high-level.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
2	the input GPIO toggles from high-level to low-level.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
3	the input GPIO toggles from high-level to low-level, or vice versa.	LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR is set.
4	the input GPIO is low-level.	the input GPIO is high-level.
5	the input GPIO is high-level.	the input GPIO is low-level.

[1] *x* ranges from 0 to 15.

[2] If the register is configured to any other value, LP GPIO wakeup feature is disabled.

7.4.7 Programming Procedure

7.4.7.1 HP GPIO Matrix

To read GPIO pin *X*[1] into HP peripheral signal *Y*, follow the steps below:

1. Configure GPIO_FUNC*y*_IN_SEL_CFG_REG corresponding to HP peripheral signal *Y* in HP GPIO matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via HP GPIO matrix.
   - Set GPIO_FUNC*y*_IN_SEL to the desired GPIO pin, i.e., *X* here.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, these peripherals can only receive input signals via HP GPIO matrix.
2. Optionally enable the GPIO Filter for pin input signals by setting IO_MUX_GPIO*n*_FILTER_EN.
3. Enable Glitch Filter feature as follows:
   - Configure GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM to *m*. *n* (0 ~ 7) represents the channel number. *m* (0 ~ 54) represents the GPIO pin number.
   - Configure GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH to **VALUE1** and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES to **VALUE2**. During **VALUE1** + 1 cycles, if there are **VALUE2** + 1 input signals that do not match the current output signal value, the Glitch Filter hardware inverts the output signal. GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH and GPIO_EXT_FILTER_CH*n*_WINDOW_THRES can be configured to the same value **VALUE3**, then only signals with a width greater than **VALUE3** + 1 clock cycles will be sampled.
   - Set GPIO_EXT_FILTER_CH*n*_EN to enable channel *n*.

   An example is shown in Figure 7.4-3, where GPIO_EXT_FILTER_CH*x*_WINDOW_WIDTH is configured to 3 and GPIO_EXT_FILTER_CH*x*_WINDOW_THRES to 2. The output signal value (signal_out) keeps as "0" in the four clock cycles before T1. The input signal value (signal_in) has been "1" for three clock cycles in the same period, then the output signal is inverted to "1" after T1.

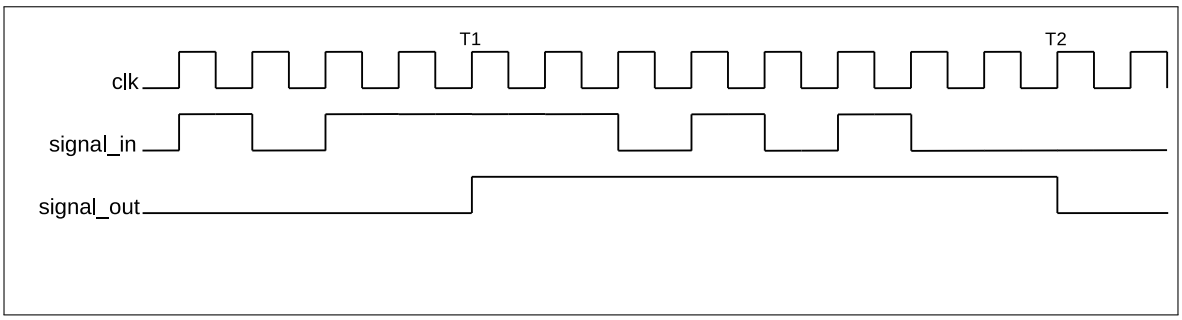


Figure 7.4-3. Glitch Filter Timing Example

4. Synchronize GPIO input signals. To do so, please set GPIO_PINx_REG corresponding to GPIO pin *X* as follows:
   - Set GPIO_PINx_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first-level synchronization, see Figure 7.4-1.
   - Set GPIO_PINx_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second-level synchronization, see Figure 7.4-1.
5. Configure HP IO MUX register to enable pin input. For this end, please set IO_MUX_GPIOx_REG corresponding to GPIO pin *X* as follows:
   - Set IO_MUX_GPIOx_FUN_IE to enable input[2].
   - Set or clear IO_MUX_GPIOx_FUN_WPU and IO_MUX_GPIOx_FUN_WPD as desired to enable or disable pull-up and pull-down resistors.

For example, to connect UART0 TXD input signal[3] (uart0_rxd_pad_in, signal index 10) to GPIO7, please follow the steps below.

1. Set GPIO_SIG10_IN_SEL in GPIO_FUNC10_IN_SEL_CFG_REG to enable peripheral signal input via HP GPIO matrix.
2. Set GPIO_FUNC10_IN_SEL in GPIO_FUNC10_IN_SEL_CFG_REG to 7, i.e., select GPIO7.
3. Set IO_MUX_GPIO7_FUN_IE in IO_MUX_GPIO7_REG to enable pin input.

Note:

1. One input pin can be connected to multiple peripheral input signals.
2. The input signal can be inverted by configuring GPIO_FUNCy_IN_INV_SEL.
3. It is possible to have a HP peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special GPIO_FUNCy_IN_SEL input, instead of a GPIO number:
   - When GPIO_FUNCy_IN_SEL is set to 62, input signal is always 0.
   - When GPIO_FUNCy_IN_SEL is set to 63, input signal is always 1.

7.4.7.2 LP GPIO Matrix

The programming procedure for peripheral input via the LP GPIO matrix resembles that of Section 7.4.7.1. However, it's worth noting that:

- the LP GPIO matrix lacks a Glitch Filter function.
- The control and status registers must use LP GPIO related registers. See Section 7.19.4 and Section 7.19.5.

Note:
It is possible to have a LP peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special LP_GPIO_SIG*y*_IN_SEL input, instead of a GPIO number:

- When LP_GPIO_SIG*y*_IN_SEL is set to 16, input signal is always 0.
- When LP_GPIO_SIG*y*_IN_SEL is set to 24, input signal is always 1.

7.5 Peripheral Output via GPIO Matrix

7.5.1 Overview

To output a signal from a peripheral via GPIO matrix,

- configure the HP GPIO matrix to route HP peripheral output signals (only signals with a name assigned in the column "Output signal" in Table 7.12-1) to one of the 55 GPIOs (0 ~ 54).
- Or configure the LP GPIO matrix to route LP peripherals (only signals with a name assigned in the column "Output signal" in Table 7.13-1) to one of the 16 LP GPIOs (0 ~ 15).
- The output signal is routed from the peripheral into GPIO matrix and then into IO MUX. IO MUX must be configured to set the chosen pin to GPIO function. This enables the GPIO output signal to be connected to the pin.

Note:
There is a range of peripheral output signals (250 ~ 255 in Table 7.12-1) which are not connected to any peripheral, but to the input signals (250 ~ 255) directly.

7.5.2 Simple GPIO Output

GPIO matrix can also be used for simple GPIO output. For this case, one GPIO pin can be configured to directly output the desired value, without routing any peripheral output to this pin.

Follow the steps below to configure HP GPIO matrix for simple GPIO output:

- Set GPIO_FUNC*x*_OUT_SEL with a special peripheral index 256 (0x100).
- Configure the corresponding bit in GPIO_OUT_REG/GPIO_OUT1_REG to the desired GPIO output value.

Note:

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31] correspond to GPIO0 ~ GPIO31, and GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[22] to GPIO32 ~ GPIO54.
- Recommended operation: use GPIO_OUT/OUT1_W1TS and GPIO_OUT/OUT1_W1TC to set or clear GPIO_OUT/OUT1_REG.

Follow the steps below to configure LP GPIO matrix for simple GPIO output:

- Set LP_GPIO_FUNC*x*_OUT_SEL with a special peripheral index 32 (0x20).
- Configure the corresponding bit in LP_GPIO_OUT_REG to the desired GPIO output value.

Note:
Recommended operation: use LP_GPIO_OUT_DATA_W1TS and LP_GPIO_OUT_DATA_W1TC to set or clear LP_GPIO_OUT_REG.

7.5.3 Sigma Delta Modulated Output (SDM)

7.5.3.1 Functional Description

Eight HP peripheral output signals (index: 72 ~ 79 in Table 7.12-1) support 1-bit second-order sigma delta modulation. By default the output is enabled for these eight channels. This Sigma Delta modulator can also output PDM (pulse density modulation) signal with configurable duty cycle. The function is:

$$H(z) = \mathrm{X(z)z^{-1}} + \mathrm{E(z)(1\text{-}z^{-1})^2}$$

E(z) is quantization error and X(z) is the input.
This modulator supports scaling down of IO MUX operating clock by divider 1 ~ 256:

- Set GPIO_EXT_SD_FUNCTION_CLK_EN to enable the modulator clock.
- Configure GPIO_EXT_SD*n*_PRESCALE (*n* = 0 ~ 7 for the eight channels).

After scaling, the clock cycle is equal to one pulse output cycle from the modulator.

GPIO_EXT_SD*n*_IN is a signed number with a range of [-128, 127] and is used to control the duty cycle[1] of PDM output signal.

- GPIO_EXT_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIO_EXT_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIO_EXT_SD*n*_IN = 127, the duty cycle of the output signal is near 100%.

The formula for calculating PDM signal duty cycle is shown as below:

$$Duty\_Cycle = \frac{GPIO\_EXT\_SDn\_IN + 128}{256}$$

Note:
For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period (several pulse cycles, for example, 256 pulse cycles).

7.5.3.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pin via HP GPIO matrix, see Section 7.5.4.1.
- Enable the modulator clock by setting GPIO_EXT_SD_FUNCTION_CLK_EN.

- Configure the divider value by setting GPIO_EXT_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting GPIO_EXT_SD*n*_IN.

7.5.4 Programming Procedure

7.5.4.1 HP GPIO Matrix

The output signals with a name assigned in the column "Output signal" in Table 7.12-1 can be set to go through the HP GPIO matrix into HP IO MUX and then to a pin. Figure 7.3-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pin *X*, follow the steps below:

1. Configure GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pin *X* in HP GPIO matrix. Recommended operation: use corresponding W1TS (write 1 to set) and W1TC (write 1 to clear) registers to set or clear GPIO_ENABLE_REG.
   - Set the GPIO_FUNC*x*_OUT_SEL field in register GPIO_FUNC*x*_OUT_SEL_CFG_REG to the index of the desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OE_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the spi2_dqs_pad_oe in column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" in Table 7.12-1), clear the GPIO_FUNC*x*_OE_SEL bit instead.
   - Set the corresponding bit in register GPIO_ENABLE_W1TC_REG to disable the output from the GPIO pin.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pin *X*.
3. Configure HP IO MUX register to enable output via HP GPIO matrix. Set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set the field IO_MUX_GPIO*x*_MCU_SEL to desired HP IO MUX function corresponding to GPIO pin *X*. This is Function 1 (GPIO function), numeric value 1, for all pins.
   - Set the IO_MUX_GPIO*x*_FUN_DRV field to the desired value for output strength (0 ~ 3). The higher the drive strength, the more current can be sourced/sunk from the pin.
   - If using open drain mode, set/clear the IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD bits to enable/disable the internal pull-up/pull-down resistors.
4. Enable hysteresis function:
   - GPIO0 ~ GPIO15: set LP_IOMUX_LP_GPIO_HYS[*x*] (*x*: 0 ~ 15, corresponding to GPIO0 ~ GPIO15).
   - GPIO16 ~ GPIO47: set HP_SYSTEM_GPIO_0_HYS_LOW[*x*] (*x*: 0 ~ 31, corresponding to GPIO16 ~ GPIO47).
   - GPIO48 ~ GPIO54: set HP_SYSTEM_GPIO_0_HYS_HIGH[*x*] (*x*: 0 ~ 6, corresponding to GPIO48 ~ GPIO54).

Note:

- The output signal from a single peripheral can be sent to multiple pins simultaneously.
- The output signal can be inverted by setting GPIO_FUNC*x*_OUT_INV_SEL.

7.5.4.2 LP GPIO Matrix

The programming procedure for peripheral input via the LP GPIO matrix resembles that of 7.5.4.1. However, it's worth noting that the control and status registers must use LP GPIO matrix registers. See Section 7.19.4 and Section 7.19.5.

7.6 Direct Input and Output via IO MUX

7.6.1 Overview

Some digital signals (SDMMC, EMAC, etc.) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to peripherals. This option is less flexible than routing signals via GPIO matrix, as the IO MUX register for each GPIO pin can only select from a limited number of functions, but high-frequency digital performance can be improved.

ESP32-P4 provides 16 GPIO pins with low power (LP) capabilities. These pins can be controlled by either HP IO MUX or LP IO MUX. If controlled by LP IO MUX, these pins will bypass HP IO MUX and HP GPIO matrix for the use by peripherals in LP system.

When configured as LP GPIOs, the pins can still be controlled by the peripherals in LP system during chip Deep-sleep, and wake up the chip from Deep-sleep.

7.6.2 Functional Description

The pins with LP functions (GPIO0 ~ GPIO15) are controlled by LP_IOMUX_PAD*n*_MUX_SEL (*n*: 0 ~ 15) in register LP_IOMUX_PAD*n*_REG. By default, all bits in these registers are set to 0, routing all input/output signals via HP IO MUX.

If LP_IOMUX_PAD*n*_MUX_SEL is set, then input/output signals are controlled by LP IO MUX. In this mode, LP_IOMUX_PAD*n*_REG is used to control the LP GPIO pins. See 7.15-1 for the LP functions of each LP GPIO pin. Note that LP_IOMUX_PAD*n*_REG applies the LP GPIO pin numbering, not the HP GPIO pin numbering.

7.6.2.1 HP IO MUX

Two fields must be configured in order to bypass HP GPIO matrix for HP peripheral input signals:

1. IO_MUX_GPIO*n*_MCU_SEL for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Table 7.14-1.
2. Clear GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass HP GPIO matrix for HP peripheral output signals, IO_MUX_GPIO*n*_MCU_SEL for the GPIO pin must be set to the required pin function.

7.6.2.2 LP IO MUX

Two fields must be configured in order to bypass LP GPIO matrix for LP peripheral input signals:

1. LP_IOMUX_PAD*n*_MUX_SEL for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Table 7.15-1.
2. Clear LP_GPIO_GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass LP GPIO matrix for LP peripheral output signals, LP_IOMUX_PAD*n*_MUX_SEL for the GPIO pin must be set to the required pin function.

Note:
Not all signals can be directly connected to peripheral via IO MUX. Some input/output signals can only be connected to peripheral via GPIO matrix.

7.7 Analog Functions

7.7.1 Overview

ESP32-P4 provides 34 GPIO pins with analog functions, including 16 LP GPIO pins and 18 HP GPIO pins. See Table 7.16-1.

7.7.2 Analog Functions

When the pin is used for analog purpose, make sure this pin is left floating by configuring registers. By such way, the external analog signal is directly connected to internal analog signal via GPIO pin. The configuration is as follows:

- Clear IO_MUX_GPIO*n*_FUN_IE, IO_MUX_GPIO*n*_FUN_WPU, and IO_MUX_GPIO*n*_FUN_WPD, to set the pin floating.
- Write 1 to the corresponding bit in GPIO_ENABLE_W1TC and GPIO_ENABLE1_W1TC, to clear output enable.

To use these functions listed in Table 7.16-1, please refer to the following related chapters:

- For XTAL_32K related functions, see Chapter 5 *Low-Power Management [to be added later]*.
- For TOUCH related functions, see Chapter 22 *Touch Sensor [to be added later]*.
- For ADC related functions, see Chapter 45 *ADC Controller (ADC)*.
- For COMP related functions, see Chapter 46 *Analog Voltage Comparator*.

Note:
GPIO51 ~ GPIO54 each has two different analog functions, which can be used simultaneously.

7.8 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-P4 is in Light-sleep mode. If IO_MUX_GPIO*n*_SLP_SEL in register IO_MUX_GPIO*n*_REG for a GPIO pin is set to 1, a different set of bits will be used to control the pin when the chip is in Light-sleep mode.

Table 7.8-1. Bit Used to Control IO MUX Functions in Light-sleep Mode

IO MUX Function	Normal Execution OR IO_MUX_GPIO*n*_SLP_SEL = 0	Light-sleep Mode AND IO_MUX_GPIO*n*_SLP_SEL = 1
Output Drive Strength	IO_MUX_GPIO*n*_FUN_DRV	IO_MUX_GPIO*n*_MCU_DRV
Pull-up Resistor	IO_MUX_GPIO*n*_FUN_WPU	IO_MUX_GPIO*n*_MCU_WPU
Pull-down Resistor	IO_MUX_GPIO*n*_FUN_WPD	IO_MUX_GPIO*n*_MCU_WPD
Input Enable	IO_MUX_GPIO*n*_FUN_IE	IO_MUX_GPIO*n*_MCU_IE
Output Enable	OE_SEL from GPIO matrix*	IO_MUX_GPIO*n*_MCU_OE

Note:
If IO_MUX_GPIO*n*_SLP_SEL is set to 0, pin functions remain the same in both normal execution and in Light-sleep mode. Please refer to Section 7.5.4.1 for how to enable output in normal execution.

7.9 Pin Hold Feature

Each GPIO pin has an individual hold function controlled by Power Management Unit (PMU) or registers. When the pin is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO MUX/GPIO configuration is modified. Users can use the hold function for the pins to retain the pin state through a core reset triggered by watchdog time-out or Deep-sleep events.

Digital Pins (GPIO16 ~ GPIO54):

The Hold state of each digital pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- HP_SYSREG_GPIO_0_HOLD_LOW_REG[*n*] (*n* = 0 ~ 31), controls the Hold signal of each pin of GPIO16 ~ GPIO47. HP_SYSREG_GPIO_0_HOLD_HIGH_REG[*n*](*n* = 0 ~ 7), controls the Hold signal of each pin of GPIO48 ~ GPIO54.
- PMU_TIE_HIGH_HP_PAD_HOLD_ALL, controls the global Hold signal of all digital pins.

To use this feature, follow the steps below:

- To maintain the pin's input/output status in Deep-sleep, set HP_SYSREG_GPIO_0_HOLD_LOW_REG[*n*] (where *n* = 0 ~ 31 corresponds to GPIO16 ~ GPIO47) or HP_SYSREG_GPIO_0_HOLD_HIGH_REG[*n*] (where *n* = 0 ~ 7 corresponds to GPIO48 ~ GPIO54) before powering down. To disable the hold function after waking up, clear both registers.
- Alternatively, set PMU_TIE_HIGH_HP_PAD_HOLD_ALL to maintain the input/output status of all digital pins and set PMU_TIE_LOW_HP_PAD_HOLD_ALL to disable the hold function for all digital pins.

- Alternatively, configure the PMU task to hold the HP pins before Deep-sleep. In Deep-sleep, the HP pins would be hold automatically by the PMU.

LP Pins (GPIO0 ~ GPIO15):

The Hold state of each LP pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- LP_IOMUX_LP_PAD_HOLD_REG[*n*], controls the Hold signal of each pin of GPIO0 ~ GPIO15.
- PMU_TIE_HIGH_LP_PAD_HOLD_ALL, controls the global Hold signal of all LP pins.

To use this feature, follow the steps below:

- To maintain the pin's input/output status in Deep-sleep, set LP_IOMUX_LP_PAD_HOLD_REG[*n*] to hold the value of GPIO*n*, or clear LP_IOMUX_LP_PAD_HOLD_REG[*n*] to disable the hold function of GPIO*n*.
- Alternatively, set PMU_TIE_HIGH_LP_PAD_HOLD_ALL to hold the values of all LP pins, and set PMU_TIE_LOW_LP_PAD_HOLD_ALL to disable the hold function of all LP pins.
- Alternatively, configure the PMU task to hold LP pins before Deep-sleep. In Deep-sleep, LP pins will be hold automatically by the PMU.

7.10 Hysteresis Characteristics of GPIO Pins

Each GPIO pin has hysteresis functionality. When hysteresis is not enabled, as shown in Figure 7.10-1, the level flip of the signal (C) input to the chip from the PAD has only one threshold (Vt, about 1.7 V). When the voltage on the PAD is higher than Vt, the level on the C is high. Otherwise, it is low. However, noise on the PAD may affect the signal on C.

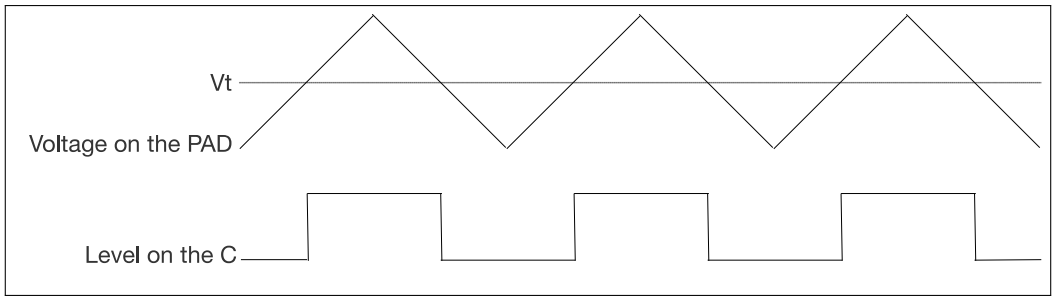


Figure 7.10-1. Example of Level Flip on the Chip Pad — Hysteresis Function Not Enabled

When hysteresis is enabled, as shown in Figure 7.10-2, the level flip of C has two thresholds, high-level threshold (Vth, about 1.7 V) and low-level threshold (Vtl, about 1.4 V). When the voltage of pad goes from low to high, if the voltage is higher than Vth, the level of C is high. When the voltage of PAD goes from high to low, if the voltage is lower than Vtl, the level of C is low. When the voltage of PAD is between Vth and Vtl, the level of C does not change. The hysteresis function plays an anti-interference role by mitigating the impact of noise, consequently reducing the level flip time of C.

To enable the hysteresis function, follow the steps below:

- Set LP_IOMUX_LP_GPIO_HYS[*n*] (*n* ranges from 0 ~ 15, corresponding to GPIO0 ~ GPIO15), HP_SYS_GPIO0_HYS_LOW[*n*] (*n* ranges from 0 ~ 31, corresponding to GPIO16 ~ GPIO47) or

HP_SYS_GPIO0_HYS_HIGH[*n*] (*n* ranges from 0 ~ 6, corresponding to GPIO48 ~ GPIO54) to 1 to enable the hysteresis function for GPIO*n*.

- Set LP_IOMUX_LP_GPIO_HYS[*n*] (*n* ranges from 0 ~ 15, corresponding to GPIO0 ~ GPIO15), HP_SYS_GPIO0_HYS_LOW[*n*] (*n* ranges from 0 ~ 31, corresponding to GPIO16 ~ GPIO47) or HP_SYS_GPIO0_HYS_HIGH[*n*] (*n* ranges from 0 ~ 6, corresponding to GPIO48 ~ GPIO54) to 0 to disable the hysteresis function for GPIO*n*.

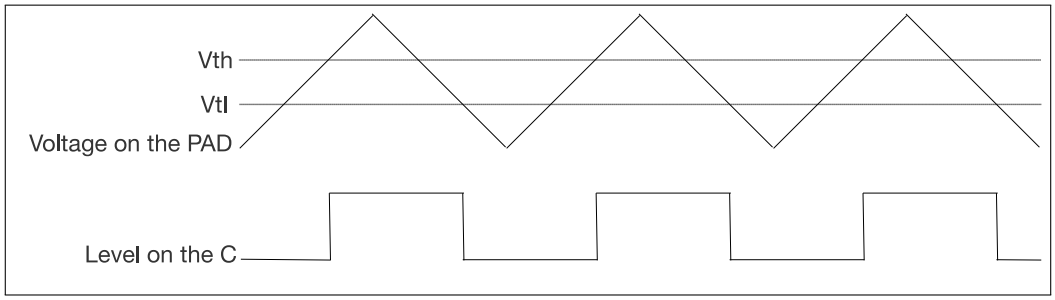


Figure 7.10-2. Example of Level Flip on the Chip Pad — Hysteresis Function Enabled

7.11 Power Supplies and Management of GPIO Pins

7.11.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-P4 Datasheet*. All the pins can be used to wake up the chip from Light-sleep, but only the pins (GPIO0 ~ GPIO15) in VDDPST1 domain can be used to wake up the chip from Deep-sleep.

7.11.2 Power Supply Management

Each ESP32-P4 pin is connected to one of the following power domains.

- VDDPST1: the input power supply for LP GPIO pins
- VDDPST2 ~ VDDPST6: the input power supply for HP GPIO pins

7.12 HP Peripheral Signal List

Table 7.12-1 shows the peripheral input/output signals via HP GPIO matrix.

Please pay attention to the configuration of the bit GPIO_FUNC*n*_OE_SEL:

- GPIO_FUNC*n*_OE_SEL = 1: the output enable is controlled by the corresponding bit *n* of GPIO_ENABLE/ENABLE1_REG:
  - GPIO_ENABLE/ENABLE1_REG = 0: output is disabled.
  - GPIO_ENABLE/ENABLE1_REG = 1: output is enabled.
- GPIO_FUNC*n*_OE_SEL = 0: use the output enable signal from peripheral, for example spi2_dqs_pad_oe in the column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0" of Table 7.12-1. Note that the signals such as spi2_dqs_pad_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of

corresponding peripherals. If it's 1'd1 in column "Output enable signal when GPIO_FUNC*n*_OE_SEL = 0", it indicates that once GPIO_FUNC*n*_OE_SEL is cleared, the output signal is always enabled by default.

Note:

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 7.12-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 7.12-1 are valid output signals.

Table 7.12-1. Peripheral Signals via HP GPIO Matrix

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
0	-	-	-	sd_card_cclk_2_pad_out	1'd1	no
1	sd_card_ccmd_2_pad_in	1	no	sd_card_ccmd_2_pad_out	sd_card_ccmd_2_pad_oe	no
2	sd_card_cdata0_2_pad_in	1	no	sd_card_cdata0_2_pad_out	sd_card_cdata0_2_pad_oe	no
3	sd_card_cdata1_2_pad_in	1	no	sd_card_cdata1_2_pad_out	sd_card_cdata1_2_pad_oe	no
4	sd_card_cdata2_2_pad_in	1	no	sd_card_cdata2_2_pad_out	sd_card_cdata2_2_pad_oe	no
5	sd_card_cdata3_2_pad_in	1	no	sd_card_cdata3_2_pad_out	sd_card_cdata3_2_pad_oe	no
6	sd_card_cdata4_2_pad_in	1	no	sd_card_cdata4_2_pad_out	sd_card_cdata4_2_pad_oe	no
7	sd_card_cdata5_2_pad_in	1	no	sd_card_cdata5_2_pad_out	sd_card_cdata5_2_pad_oe	no
8	sd_card_cdata6_2_pad_in	1	no	sd_card_cdata6_2_pad_out	sd_card_cdata6_2_pad_oe	no
9	sd_card_cdata7_2_pad_in	1	no	sd_card_cdata7_2_pad_out	sd_card_cdata7_2_pad_oe	no
10	uart0_rxd_pad_in	0	yes	uart0_txd_pad_out	1'd1	yes
11	uart0_cts_pad_in	0	yes	uart0_rts_pad_out	1'd1	yes
12	uart0_dsr_pad_in	0	no	uart0_dtr_pad_out	1'd1	no
13	uart1_rxd_pad_in	0	yes	uart1_txd_pad_out	1'd1	yes
14	uart1_cts_pad_in	0	yes	uart1_rts_pad_out	1'd1	yes
15	uart1_dsr_pad_in	0	no	uart1_dtr_pad_out	1'd1	no
16	uart2_rxd_pad_in	0	no	uart2_txd_pad_out	1'd1	no
17	uart2_cts_pad_in	0	no	uart2_rts_pad_out	1'd1	no
18	uart2_dsr_pad_in	0	no	uart2_dtr_pad_out	1'd1	no
19	uart3_rxd_pad_in	0	no	uart3_txd_pad_out	1'd1	no
20	uart3_cts_pad_in	0	no	uart3_rts_pad_out	1'd1	no
21	uart3_dsr_pad_in	0	no	uart3_dtr_pad_out	1'd1	no
22	uart4_rxd_pad_in	0	no	uart4_txd_pad_out	1'd1	no
23	uart4_cts_pad_in	0	no	uart4_rts_pad_out	1'd1	no
24	uart4_dsr_pad_in	0	no	uart4_dtr_pad_out	1'd1	no
25	i2s0_o_bck_pad_in	0	no	i2s0_o_bck_pad_out	1'd1	no
26	i2s0_mclk_pad_in	0	no	i2s0_mclk_pad_out	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
27	i2s0_o_ws_pad_in	0	no	i2s0_o_ws_pad_out	1'd1	no
28	i2s0_i_sd_pad_in	0	no	i2s0_o_sd_pad_out	1'd1	no
29	i2s0_i_bck_pad_in	0	no	i2s0_i_bck_pad_out	1'd1	no
30	i2s0_i_ws_pad_in	0	no	i2s0_i_ws_pad_out	1'd1	no
31	i2s1_o_bck_pad_in	0	no	i2s1_o_bck_pad_out	1'd1	no
32	i2s1_mclk_pad_in	0	no	i2s1_mclk_pad_out	1'd1	no
33	i2s1_o_ws_pad_in	0	no	i2s1_o_ws_pad_out	1'd1	no
34	i2s1_i_sd_pad_in	0	no	i2s1_o_sd_pad_out	1'd1	no
35	i2s1_i_bck_pad_in	0	no	i2s1_i_bck_pad_out	1'd1	no
36	i2s1_i_ws_pad_in	0	no	i2s1_i_ws_pad_out	1'd1	no
37	i2s2_o_bck_pad_in	0	no	i2s2_o_bck_pad_out	1'd1	no
38	i2s2_mclk_pad_in	0	no	i2s2_mclk_pad_out	1'd1	no
39	i2s2_o_ws_pad_in	0	no	i2s2_o_ws_pad_out	1'd1	no
40	i2s2_i_sd_pad_in	0	no	i2s2_o_sd_pad_out	1'd1	no
41	i2s2_i_bck_pad_in	0	no	i2s2_i_bck_pad_out	1'd1	no
42	i2s2_i_ws_pad_in	0	no	i2s2_i_ws_pad_out	1'd1	no
43	i2s0_i_sd1_pad_in	0	no	i2s0_o_sd1_pad_out	1'd1	no
44	i2s0_i_sd2_pad_in	0	no	spi2_dqs_pad_out	spi2_dqs_pad_oe	yes
45	i2s0_i_sd3_pad_in	0	no	spi3_cs2_pad_out	spi3_cs2_pad_oe	no
46	-	-	-	spi3_cs1_pad_out	spi3_cs1_pad_oe	no
47	spi3_ck_pad_in	0	no	spi3_ck_pad_out	spi3_ck_pad_oe	no
48	spi3_q_pad_in	0	no	spi3_qo_pad_out	spi3_q_pad_oe	no
49	spi3_d_pad_in	0	no	spi3_d_pad_out	spi3_d_pad_oe	no
50	spi3_hold_pad_in	0	no	spi3_hold_pad_out	spi3_hold_pad_oe	no
51	spi3_wp_pad_in	0	no	spi3_wp_pad_out	spi3_wp_pad_oe	no
52	spi3_cs_pad_in	0	no	spi3_cs_pad_out	spi3_cs_pad_oe	no
53	spi2_ck_pad_in	0	yes	spi2_ck_pad_out	spi2_ck_pad_oe	yes
54	spi2_q_pad_in	0	yes	spi2_q_pad_out	spi2_q_pad_oe	yes

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
55	spi2_d_pad_in	0	yes	spi2_d_pad_out	spi2_d_pad_oe	yes
56	spi2_hold_pad_in	0	yes	spi2_hold_pad_out	spi2_hold_pad_oe	yes
57	spi2_wp_pad_in	0	yes	spi2_wp_pad_out	spi2_wp_pad_oe	yes
58	spi2_io4_pad_in	0	yes	spi2_io4_pad_out	spi2_io4_pad_oe	yes
59	spi2_io5_pad_in	0	yes	spi2_io5_pad_out	spi2_io5_pad_oe	yes
60	spi2_io6_pad_in	0	yes	spi2_io6_pad_out	spi2_io6_pad_oe	yes
61	spi2_io7_pad_in	0	yes	spi2_io7_pad_out	spi2_io7_pad_oe	yes
62	spi2_cs_pad_in	0	yes	spi2_cs_pad_out	spi2_cs_pad_oe	yes
63	pcnt_rst_pad_in0	0	no	spi2_cs1_pad_out	spi2_cs1_pad_oe	no
64	pcnt_rst_pad_in1	0	no	spi2_cs2_pad_out	spi2_cs2_pad_oe	no
65	pcnt_rst_pad_in2	0	no	spi2_cs3_pad_out	spi2_cs3_pad_oe	no
66	pcnt_rst_pad_in3	0	no	spi2_cs4_pad_out	spi2_cs4_pad_oe	no
67	-	-	-	spi2_cs5_pad_out	spi2_cs5_pad_oe	no
68	i2c0_scl_pad_in	1	no	i2c0_scl_pad_out	i2c0_scl_oe_pad_out	no
69	i2c0_sda_pad_in	1	no	i2c0_sda_pad_out	i2c0_sda_oe_pad_out	no
70	i2c1_scl_pad_in	1	no	i2c1_scl_pad_out	i2c1_scl_oe_pad_out	no
71	i2c1_sda_pad_in	1	no	i2c1_sda_pad_out	i2c1_sda_oe_pad_out	no
72	-	-	-	gpio_sd0_out	1'd1	no
73	-	-	-	gpio_sd1_out	1'd1	no
74	uart0_slp_clk_pad_in	0	no	gpio_sd2_out	1'd1	no
75	uart1_slp_clk_pad_in	0	no	gpio_sd3_out	1'd1	no
76	uart2_slp_clk_pad_in	0	no	gpio_sd4_out	1'd1	no
77	uart3_slp_clk_pad_in	0	no	gpio_sd5_out	1'd1	no
78	uart4_slp_clk_pad_in	0	no	gpio_sd6_out	1'd1	no
79	-	-	-	gpio_sd7_out	1'd1	no
80	twai0_rx_pad_in	1	no	twai0_tx_pad_out	1'd1	no
81	-	-	-	twai0_bus_off_on_pad_out	1'd1	no
82	-	-	-	twai0_clkout_pad_out	1'd1	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
83	twai1_rx_pad_in	1	no	twai1_tx_pad_out	1'd1	no
84	-	-	-	twai1_bus_off_on_pad_out	1'd1	no
85	-	-	-	twai1_clkout_pad_out	1'd1	no
86	twai2_rx_pad_in	1	no	twai2_tx_pad_out	1'd1	no
87	-	-	-	twai2_bus_off_on_pad_out	1'd1	no
88	-	-	-	twai2_clkout_pad_out	1'd1	no
89	pwm0_sync0_pad_in	0	no	pwm0_ch0_a_pad_out	1'd1	no
90	pwm0_sync1_pad_in	0	no	pwm0_ch0_b_pad_out	1'd1	no
91	pwm0_sync2_pad_in	0	no	pwm0_ch1_a_pad_out	1'd1	no
92	pwm0_f0_pad_in	0	no	pwm0_ch1_b_pad_out	1'd1	no
93	pwm0_f1_pad_in	0	no	pwm0_ch2_a_pad_out	1'd1	no
94	pwm0_f2_pad_in	0	no	pwm0_ch2_b_pad_out	1'd1	no
95	pwm0_cap0_pad_in	0	no	pwm1_ch0_a_pad_out	1'd1	no
96	pwm0_cap1_pad_in	0	no	pwm1_ch0_b_pad_out	1'd1	no
97	pwm0_cap2_pad_in	0	no	pwm1_ch1_a_pad_out	1'd1	no
98	pwm1_sync0_pad_in	0	no	pwm1_ch1_b_pad_out	1'd1	no
99	pwm1_sync1_pad_in	0	no	pwm1_ch2_a_pad_out	1'd1	no
100	pwm1_sync2_pad_in	0	no	pwm1_ch2_b_pad_out	1'd1	no
101	pwm1_f0_pad_in	0	no	-	-	-
102	pwm1_f1_pad_in	0	no	-	-	-
103	pwm1_f2_pad_in	0	no	-	-	-
104	pwm1_cap0_pad_in	0	no	-	-	-
105	pwm1_cap1_pad_in	0	no	twai0_standby_pad_out	1'd1	no
106	pwm1_cap2_pad_in	0	no	twai1_standby_pad_out	1'd1	no
107	gmii_mdi_pad_in	0	no	twai2_standby_pad_out	1'd1	no
108	gmac_phy_col_pad_in	0	no	gmii_mdc_pad_out	1'd1	no
109	gmac_phy_crs_pad_in	0	no	gmii_mdo_pad_out	gmii_mdo_oe_pad_out	no
110	usb_otg11_iddig_pad_in	0	no	usb_srp_dischrgvbus_pad_out	1'd1	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
111	usb_otg11_avalid_pad_in	0	no	usb_otg11_idpullup_pad_out	1'd1	no
112	usb_srp_bvalid_pad_in	0	no	usb_otg11_dppulldown_pad_out	1'd1	no
113	usb_otg11_vbusvalid_pad_in	0	no	usb_otg11_dmpulldown_pad_out	1'd1	no
114	usb_srp_sessend_pad_in	0	no	usb_otg11_drvvbus_pad_out	1'd1	no
115	-	-	-	usb_srp_chrgvbus_pad_out	1'd1	no
116	-	-	-	-	-	-
117	ulpi_clk_pad_in	0	no	-	-	-
118	usb_hsphy_refclk_in	0	no	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	-	-	-	-	-	-
122	-	-	-	-	-	-
123	-	-	-	-	-	-
124	-	-	-	-	-	-
125	-	-	-	-	-	-
126	sd_card_detect_n_1_pad_in	0	no	ledc_ls_sig_out_pad_out0	1'd1	no
127	sd_card_detect_n_2_pad_in	0	no	ledc_ls_sig_out_pad_out1	1'd1	no
128	sd_card_int_n_1_pad_in	1	no	ledc_ls_sig_out_pad_out2	1'd1	no
129	sd_card_int_n_2_pad_in	1	no	ledc_ls_sig_out_pad_out3	1'd1	no
130	sd_card_write_prt_1_pad_in	0	no	ledc_ls_sig_out_pad_out4	1'd1	no
131	sd_card_write_prt_2_pad_in	0	no	ledc_ls_sig_out_pad_out5	1'd1	no
132	sd_data_strobe_1_pad_in	0	no	ledc_ls_sig_out_pad_out6	1'd1	no
133	sd_data_strobe_2_pad_in	0	no	ledc_ls_sig_out_pad_out7	1'd1	no
134	i3c_mst_scl_pad_in	1	no	i3c_mst_scl_pad_out	i3c_mst_scl_oe_pad_out	no
135	i3c_mst_sda_pad_in	1	no	i3c_mst_sda_pad_out	i3c_mst_sda_oe_pad_out	no
136	i3c_slv_scl_pad_in	1	no	i3c_slv_scl_pad_out	i3c_slv_scl_oe_pad_out	no
137	i3c_slv_sda_pad_in	1	no	i3c_slv_sda_pad_out	i3c_slv_sda_oe_pad_out	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
138	-	-	-	i3c_mst_scl_pullup_en_pad_out	1'd1	no
139	-	-	-	i3c_mst_sda_pullup_en_pad_out	1'd1	no
140	usb_jtag_tdo_bridge_pad_in	0	no	usb_jtag_tdi_bridge_pad_out	1'd1	no
141	pcnt_sig_ch0_pad_in0	0	no	usb_jtag_tms_bridge_pad_out	1'd1	no
142	pcnt_sig_ch0_pad_in1	0	no	usb_jtag_tck_bridge_pad_out	1'd1	no
143	pcnt_sig_ch0_pad_in2	0	no	usb_jtag_trst_bridge_pad_out	1'd1	no
144	pcnt_sig_ch0_pad_in3	0	no	lcd_cs_pad_out	1'd1	no
145	pcnt_sig_ch1_pad_in0	0	no	lcd_dc_pad_out	1'd1	no
146	pcnt_sig_ch1_pad_in1	0	no	sd_rst_n_1_pad_out	1'd1	no
147	pcnt_sig_ch1_pad_in2	0	no	sd_rst_n_2_pad_out	1'd1	no
148	pcnt_sig_ch1_pad_in3	0	no	sd_ccmd_od_pullup_en_n_pad_out	1'd1	no
149	pcnt_ctrl_ch0_pad_in0	0	no	lcd_pclk_pad_out	1'd1	no
150	pcnt_ctrl_ch0_pad_in1	0	no	cam_clk_pad_out	1'd1	no
151	pcnt_ctrl_ch0_pad_in2	0	no	lcd_h_enable_pad_out	1'd1	no
152	pcnt_ctrl_ch0_pad_in3	0	no	lcd_h_sync_pad_out	1'd1	no
153	pcnt_ctrl_ch1_pad_in0	0	no	lcd_v_sync_pad_out	1'd1	no
154	pcnt_ctrl_ch1_pad_in1	0	no	lcd_data_out_pad_out0	1'd1	no
155	pcnt_ctrl_ch1_pad_in2	0	no	lcd_data_out_pad_out1	1'd1	no
156	pcnt_ctrl_ch1_pad_in3	0	no	lcd_data_out_pad_out2	1'd1	no
157	-	-	-	lcd_data_out_pad_out3	1'd1	no
158	cam_pclk_pad_in	0	no	lcd_data_out_pad_out4	1'd1	no
159	cam_h_enable_pad_in	0	no	lcd_data_out_pad_out5	1'd1	no
160	cam_h_sync_pad_in	0	no	lcd_data_out_pad_out6	1'd1	no
161	cam_v_sync_pad_in	0	no	lcd_data_out_pad_out7	1'd1	no
162	cam_data_in_pad_in0	0	no	lcd_data_out_pad_out8	1'd1	no
163	cam_data_in_pad_in1	0	no	lcd_data_out_pad_out9	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
164	cam_data_in_pad_in2	0	no	lcd_data_out_pad_out10	1'd1	no
165	cam_data_in_pad_in3	0	no	lcd_data_out_pad_out11	1'd1	no
166	cam_data_in_pad_in4	0	no	lcd_data_out_pad_out12	1'd1	no
167	cam_data_in_pad_in5	0	no	lcd_data_out_pad_out13	1'd1	no
168	cam_data_in_pad_in6	0	no	lcd_data_out_pad_out14	1'd1	no
169	cam_data_in_pad_in7	0	no	lcd_data_out_pad_out15	1'd1	no
170	cam_data_in_pad_in8	0	no	lcd_data_out_pad_out16	1'd1	no
171	cam_data_in_pad_in9	0	no	lcd_data_out_pad_out17	1'd1	no
172	cam_data_in_pad_in10	0	no	lcd_data_out_pad_out18	1'd1	no
173	cam_data_in_pad_in11	0	no	lcd_data_out_pad_out19	1'd1	no
174	cam_data_in_pad_in12	0	no	lcd_data_out_pad_out20	1'd1	no
175	cam_data_in_pad_in13	0	no	lcd_data_out_pad_out21	1'd1	no
176	cam_data_in_pad_in14	0	no	lcd_data_out_pad_out22	1'd1	no
177	cam_data_in_pad_in15	0	no	lcd_data_out_pad_out23	1'd1	no
178	gmac_phy_rxdv_pad_in	0	yes	gmac_phy_txen_pad_out	1'd1	yes
179	gmac_phy_rxd0_pad_in	0	yes	gmac_phy_txd0_pad_out	1'd1	yes
180	gmac_phy_rxd1_pad_in	0	yes	gmac_phy_txd1_pad_out	1'd1	yes
181	gmac_phy_rxd2_pad_in	0	no	gmac_phy_txd2_pad_out	1'd1	no
182	gmac_phy_rxd3_pad_in	0	no	gmac_phy_txd3_pad_out	1'd1	no
183	gmac_phy_rxer_pad_in	0	yes	gmac_phy_txer_pad_out	1'd1	yes
184	gmac_rx_clk_pad_in	0	no	-	-	-
185	gmac_tx_clk_pad_in	0	no	-	-	-
186	parlio_rx_clk_pad_in	0	no	parlio_rx_clk_pad_out	1'd1	no
187	parlio_tx_clk_pad_in	0	no	parlio_tx_clk_pad_out	1'd1	no
188	parlio_rx_data0_pad_in	0	no	parlio_tx_data0_pad_out	1'd1	no
189	parlio_rx_data1_pad_in	0	no	parlio_tx_data1_pad_out	1'd1	no
190	parlio_rx_data2_pad_in	0	no	parlio_tx_data2_pad_out	1'd1	no
191	parlio_rx_data3_pad_in	0	no	parlio_tx_data3_pad_out	1'd1	no

GoBack

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
192	parlio_rx_data4_pad_in	0	no	parlio_tx_data4_pad_out	1'd1	no
193	parlio_rx_data5_pad_in	0	no	parlio_tx_data5_pad_out	1'd1	no
194	parlio_rx_data6_pad_in	0	no	parlio_tx_data6_pad_out	1'd1	no
195	parlio_rx_data7_pad_in	0	no	parlio_tx_data7_pad_out	1'd1	no
196	parlio_rx_data8_pad_in	0	no	parlio_tx_data8_pad_out	1'd1	no
197	parlio_rx_data9_pad_in	0	no	parlio_tx_data9_pad_out	1'd1	no
198	parlio_rx_data10_pad_in	0	no	parlio_tx_data10_pad_out	1'd1	no
199	parlio_rx_data11_pad_in	0	no	parlio_tx_data11_pad_out	1'd1	no
200	parlio_rx_data12_pad_in	0	no	parlio_tx_data12_pad_out	1'd1	no
201	parlio_rx_data13_pad_in	0	no	parlio_tx_data13_pad_out	1'd1	no
202	parlio_rx_data14_pad_in	0	no	parlio_tx_data14_pad_out	1'd1	no
203	parlio_rx_data15_pad_in	0	no	parlio_tx_data15_pad_out	1'd1	no
204	-	-	-	-	-	-
205	-	-	-	-	-	-
206	-	-	-	-	-	-
207	-	-	-	-	-	-
208	-	-	-	-	-	-
209	-	-	-	-	-	-
210	-	-	-	-	-	-
211	-	-	-	-	-	-
212	-	-	-	constant0_pad_out	1'd1	no
213	-	-	-	constant1_pad_out	1'd1	no
214	core_gpio_in_pad_in0	0	no	core_gpio_out_pad_out0	core_gpio_oe_pad_out0	no
215	core_gpio_in_pad_in1	0	no	core_gpio_out_pad_out1	core_gpio_oe_pad_out1	no
216	core_gpio_in_pad_in2	0	no	core_gpio_out_pad_out2	core_gpio_oe_pad_out2	no
217	core_gpio_in_pad_in3	0	no	core_gpio_out_pad_out3	core_gpio_oe_pad_out3	no
218	core_gpio_in_pad_in4	0	no	core_gpio_out_pad_out4	core_gpio_oe_pad_out4	no
219	core_gpio_in_pad_in5	0	no	core_gpio_out_pad_out5	core_gpio_oe_pad_out5	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
220	core_gpio_in_pad_in6	0	no	core_gpio_out_pad_out6	core_gpio_oe_pad_out6	no
221	core_gpio_in_pad_in7	0	no	core_gpio_out_pad_out7	core_gpio_oe_pad_out7	no
222	core_gpio_in_pad_in8	0	no	core_gpio_out_pad_out8	core_gpio_oe_pad_out8	no
223	core_gpio_in_pad_in9	0	no	core_gpio_out_pad_out9	core_gpio_oe_pad_out9	no
224	core_gpio_in_pad_in10	0	no	core_gpio_out_pad_out10	core_gpio_oe_pad_out10	no
225	core_gpio_in_pad_in11	0	no	core_gpio_out_pad_out11	core_gpio_oe_pad_out11	no
226	core_gpio_in_pad_in12	0	no	core_gpio_out_pad_out12	core_gpio_oe_pad_out12	no
227	core_gpio_in_pad_in13	0	no	core_gpio_out_pad_out13	core_gpio_oe_pad_out13	no
228	core_gpio_in_pad_in14	0	no	core_gpio_out_pad_out14	core_gpio_oe_pad_out14	no
229	core_gpio_in_pad_in15	0	no	core_gpio_out_pad_out15	core_gpio_oe_pad_out15	no
230	core_gpio_in_pad_in16	0	no	core_gpio_out_pad_out16	core_gpio_oe_pad_out16	no
231	core_gpio_in_pad_in17	0	no	core_gpio_out_pad_out17	core_gpio_oe_pad_out17	no
232	core_gpio_in_pad_in18	0	no	core_gpio_out_pad_out18	core_gpio_oe_pad_out18	no
233	core_gpio_in_pad_in19	0	no	core_gpio_out_pad_out19	core_gpio_oe_pad_out19	no
234	core_gpio_in_pad_in20	0	no	core_gpio_out_pad_out20	core_gpio_oe_pad_out20	no
235	core_gpio_in_pad_in21	0	no	core_gpio_out_pad_out21	core_gpio_oe_pad_out21	no
236	core_gpio_in_pad_in22	0	no	core_gpio_out_pad_out22	core_gpio_oe_pad_out22	no
237	core_gpio_in_pad_in23	0	no	core_gpio_out_pad_out23	core_gpio_oe_pad_out23	no
238	core_gpio_in_pad_in24	0	no	core_gpio_out_pad_out24	core_gpio_oe_pad_out24	no
239	core_gpio_in_pad_in25	0	no	core_gpio_out_pad_out25	core_gpio_oe_pad_out25	no
240	core_gpio_in_pad_in26	0	no	core_gpio_out_pad_out26	core_gpio_oe_pad_out26	no
241	core_gpio_in_pad_in27	0	no	core_gpio_out_pad_out27	core_gpio_oe_pad_out27	no
242	core_gpio_in_pad_in28	0	no	core_gpio_out_pad_out28	core_gpio_oe_pad_out28	no
243	core_gpio_in_pad_in29	0	no	core_gpio_out_pad_out29	core_gpio_oe_pad_out29	no
244	core_gpio_in_pad_in30	0	no	ana_comp0_out	1'd1	no
245	core_gpio_in_pad_in31	0	no	ana_comp1_out	1'd1	no
246	rmt_sig_pad_in0	0	no	rmt_sig_pad_out0	1'd1	no
247	rmt_sig_pad_in1	0	no	rmt_sig_pad_out1	1'd1	no

Signal No.	Input Signal	Default Value	Direct Input via HP IO MUX	Output Signal	Output Enable Signal when GPIO_FUNCn_OE_SEL = 0	Direct Output via HP IO MUX
248	rmt_sig_pad_in2	0	no	rmt_sig_pad_out2	1'd1	no
249	rmt_sig_pad_in3	0	no	rmt_sig_pad_out3	1'd1	no
250	sig_in_func250	0	no	sig_in_func250	1'd1	no
251	sig_in_func251	0	no	sig_in_func251	1'd1	no
252	sig_in_func252	0	no	sig_in_func252	1'd1	no
253	sig_in_func253	0	no	sig_in_func253	1'd1	no
254	sig_in_func254	0	no	sig_in_func254	1'd1	no
255	sig_in_func255	0	no	sig_in_func255	1'd1	no

7.13 LP Peripheral Signal List

Table 7.13-1 shows the peripheral input/output signals via LP GPIO matrix.

Please pay attention to the configuration of the bit LP_GPIO_FUNC*n*_OE_SEL:

- LP_GPIO_FUNC*n*_OE_SEL = 1: the output enable is controlled by the corresponding bit*n* of LP_GPIO_ENABLE_REG:
  - LP_GPIO_ENABLE_REG = 0: output is disabled.
  - LP_GPIO_ENABLE_REG = 1: output is enabled.
- LP_GPIO_FUNC*n*_OE_SEL = 0: use the output enable signal from peripheral, for example lp_spi_clk_pad_oe in the column "Output enable signal when LP_GPIO_FUNC*n*_OE_SEL = 0" of Table 7.13-1. Note that the signals such as lp_spi_clk_pad_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in column "Output enable signal when LP_GPIO_FUNC*n*_OE_SEL = 0", it indicates that once LP_GPIO_FUNC*n*_OE_SEL is cleared, the output signal is always enabled by default.

Note:

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 7.13-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 7.13-1 are valid output signals.

Table 7.13-1. Peripheral Signals via LP GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input via LP IO MUX	Output Signal	Output enable signal when LP_GPIO_FUNCn_OE_SEL = 0	Direct Output via LP IO MUX
0	lp_i2c_scl_pad_in	1	no	lp_i2c_scl_pad_out	lp_i2c_scl_pad_oe	no
1	lp_i2c_sda_pad_in	1	no	lp_i2c_sda_pad_out	lp_i2c_sda_pad_oe	no
2	lp_uart_rxd_pad_in	0	yes	lp_uart_txd_pad_out	1'd1	yes
3	lp_uart_ctsn_pad_in	1	no	lp_uart_rtsn_pad_out	1'd1	no
4	lp_uart_dsrn_pad_in	1	no	lp_uart_dtrn_pad_out	1'd1	no
5	lp_spi_ck_pad_in	0	no	lp_spi_ck_pad_out	lp_spi_ck_pad_oe	no
6	lp_spi_cs_pad_in	0	no	lp_spi_cs_pad_out	lp_spi_cs_pad_oe	no
7	lp_spi_d_pad_in	0	no	lp_spi_d_pad_out	lp_spi_d_pad_oe	no
8	lp_spi_q_pad_in	0	no	lp_spi_q_pad_out	lp_spi_q_pad_oe	no
9	lp_i2s_i_bck_pad_in	0	no	lp_i2s_i_bck_pad_out	1'd1	no
10	lp_i2s_i_sd_pad_in	0	no	lp_i2s_o_sd_pad_out	1'd1	no
11	lp_i2s_i_ws_pad_in	0	no	lp_i2s_i_ws_pad_out	1'd1	no
12	lp_i2s_o_bck_pad_in	0	no	lp_i2s_o_bck_pad_out	1'd1	no
13	lp_i2s_o_ws_pad_in	0	no	lp_i2s_o_ws_pad_out	1'd1	no
14	-	-	-	-	-	-
15	-	-	-	-	-	-
16	-	-	-	-	-	-
17	-	-	-	-	-	-
18	-	-	-	-	-	-
19	-	-	-	-	-	-
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via LP IO MUX	Output Signal	Output enable signal when LP_GPIO_FUNCn_OE_SEL = 0	Direct Output via LP IO MUX
24	-	-	-	-	-	-
25	-	-	-	-	-	-
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	-	-	-	-	-	-
29	-	-	-	-	-	-
30	-	-	-	-	-	-
31	-	-	-	-	-	-

7.14 HP IO MUX Functions List

Table 7.14-1 shows the HP IO MUX functions and default states of each HP GPIO pin.

Table 7.14-1. HP IO MUX Pin Functions

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO0	GPIO0	GPIO0	-	-	2	0	R
GPIO1	GPIO1	GPIO1	-	-	2	0	R
GPIO2	MTCK	GPIO2	-	-	2	1*	R
GPIO3	MTDI	GPIO3	-	-	2	1	R
GPIO4	MTMS	GPIO4	-	-	2	1	R
GPIO5	MTDO	GPIO5	-	-	2	0	R
GPIO6	GPIO6	GPIO6	-	SPI2_HOLD_PAD	2	0	R
GPIO7	GPIO7	GPIO7	-	SPI2_CS_PAD	2	0	R
GPIO8	GPIO8	GPIO8	UART0_RTS_PAD	SPI2_D_PAD	2	0	R

Cont'd on next page

GoBack

Table 7.14-1 – cont'd from previous page

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO9	GPIO9	GPIO9	UART0_CTS_PAD	SPI2_CK_PAD	2	0	R
GPIO10	GPIO10	GPIO10	UART1_TXD_PAD	SPI2_Q_PAD	2	0	R
GPIO11	GPIO11	GPIO11	UART1_RXD_PAD	SPI2_WP_PAD	2	0	R
GPIO12	GPIO12	GPIO12	UART1_RTS_PAD	-	2	0	R
GPIO13	GPIO13	GPIO13	UART1_CTS_PAD	-	2	0	R
GPIO14	GPIO14	GPIO14	—	-	2	0	R
GPIO15	GPIO15	GPIO15	—	-	2	0	R
GPIO16	GPIO16	GPIO16	—	-	2	0	R
GPIO17	GPIO17	GPIO17	—	-	2	0	R
GPIO18	GPIO18	GPIO18	—	-	2	0	R
GPIO19	GPIO19	GPIO19	—	-	2	0	R
GPIO20	GPIO20	GPIO20	—	-	2	0	R
GPIO21	GPIO21	GPIO21	—	-	2	0	R
GPIO22	GPIO22	GPIO22	—	-	2	0	R
GPIO23	GPIO23	GPIO23	—	REF_50M_CLK_PAD	2	0	R
GPIO24	GPIO24	GPIO24	—	-	3	0	R, USB
GPIO25	GPIO25	GPIO25	—	-	3	3*	R, USB
GPIO26	GPIO26	GPIO26	—	-	2	0	R, USB
GPIO27	GPIO27	GPIO27	—	-	2	0	R, USB
GPIO28	GPIO28	GPIO28	SPI2_CS_PAD	GMAC_PHY_RXDV_PAD	2	0	—
GPIO29	GPIO29	GPIO29	SPI2_D_PAD	GMAC_PHY_RXD0_PAD	2	0	—
GPIO30	GPIO30	GPIO30	SPI2_CK_PAD	GMAC_PHY_RXD1_PAD	2	0	—
GPIO31	GPIO31	GPIO31	SPI2_Q_PAD	GMAC_PHY_RXER_PAD	2	0	—
GPIO32	GPIO32	GPIO32	SPI2_HOLD_PAD	GMAC_RMII_CLK_PAD	2	0	—
GPIO33	GPIO33	GPIO33	SPI2_WP_PAD	GMAC_PHY_TXEN_PAD	2	0	—
GPIO34	GPIO34	GPIO34	SPI2_IO4_PAD	GMAC_PHY_TXD0_PAD	2	0	—
GPIO35	GPIO35	GPIO35	SPI2_IO5_PAD	GMAC_PHY_TXD1_PAD	2	0	—

Cont'd on next page

GoBack

Table 7.14-1 – cont'd from previous page

Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
GPIO36	GPIO36	GPIO36	SPI2_IO6_PAD	GMAC_PHY_TXER_PAD	2	0	—
GPIO37	UART0_TXD_PAD	GPIO37	SPI2_IO7_PAD	-	2	5	—
GPIO38	UART0_RXD_PAD	GPIO38	SPI2_DQS_PAD	-	2	0	—
GPIO39	SD1_CDATA0_PAD	GPIO39	—	REF_50M_CLK_PAD	2	0	—
GPIO40	SD1_CDATA1_PAD	GPIO40	—	GMAC_PHY_TXEN_PAD	2	0	—
GPIO41	SD1_CDATA2_PAD	GPIO41	—	GMAC_PHY_TXD0_PAD	2	0	—
GPIO42	SD1_CDATA3_PAD	GPIO42	—	GMAC_PHY_TXD1_PAD	2	0	—
GPIO43	SD1_CCLK_PAD	GPIO43	—	GMAC_PHY_TXER_PAD	2	0	—
GPIO44	SD1_CCMD_PAD	GPIO44	—	GMAC_RMII_CLK_PAD	2	0	—
GPIO45	SD1_CDATA4_PAD	GPIO45	—	GMAC_PHY_RXDV_PAD	2	0	—
GPIO46	SD1_CDATA5_PAD	GPIO46	—	GMAC_PHY_RXD0_PAD	2	0	—
GPIO47	SD1_CDATA6_PAD	GPIO47	—	GMAC_PHY_RXD1_PAD	2	0	—
GPIO48	SD1_CDATA7_PAD	GPIO48	—	GMAC_PHY_RXER_PAD	2	0	—
GPIO49	GPIO49	GPIO49	—	GMAC_PHY_TXEN_PAD	2	0	R
GPIO50	GPIO50	GPIO50	—	GMAC_RMII_CLK_PAD	2	0	R
GPIO51	GPIO51	GPIO51	—	GMAC_PHY_RXDV_PAD	2	0	R
GPIO52	GPIO52	GPIO52	—	GMAC_PHY_RXD0_PAD	2	0	R
GPIO53	GPIO53	GPIO53	—	GMAC_PHY_RXD1_PAD	2	0	R
GPIO54	GPIO54	GPIO54	—	GMAC_PHY_RXER_PAD	2	0	R

Drive Strength

"DRV" column shows the drive strength of each pin after reset:

- **0** - Drive current = ~5 mA
- **1** - Drive current = ~10 mA
- **2** - Drive current = ~20 mA
- **3** - Drive current = ~40 mA

Reset

The default configuration of each pin after reset:

- **0** - input disabled, in high impedance state (IE = 0)
- **1** - input enabled, in high impedance state (IE = 1)
- **1*** - When the value of eFuse bit EFUSE_DIS_PAD_JTAG is
  0 (default), input enabled, pull-up resistor enabled (IE = 1, WPU = 1)
  1, input disabled, in high impedance state (IE = 1)
- **3*** - input enabled, pull-up resistor enabled (IE = 1, WPU = 0, USB_WPU = 1). See details in Notes.
- **5** - input disabled. Output is controlled by the peripheral of Function 0, and the default output is 1.

We recommend pulling high or low GPIO pins in high impedance state to avoid unnecessary power consumption. You may add pull-up and pull-down resistors in your PCB design, or enable internal pull-up and pull-down resistors during software initialization.

Notes

- **R** - These pins have analog functions.
- **USB** - The pull-up value of a USB pin is controlled by the pin's pull-up value together with the USB pull-up value. If any of the two pull-up values is 1, the pin's pull-up resistor will be enabled.

Notice:
In HP IO MUX, unused pins must be configured to GPIO function.

7.15 LP IO MUX Functions List

Table 7.15-1 shows the LP IO MUX functions of each GPIO pin. GPIO pins are default controlled by HP IO MUX, so Table 7.15-1 only show functions of LP IO MUX.

Table 7.15-1. LP IO MUX Pin Functions

Name	Function 0	Function 1
GPIO0	LP_GPIO0	LP_GPIO0
GPIO1	LP_GPIO1	LP_GPIO1
GPIO2	LP_GPIO2	LP_GPIO2
GPIO3	LP_GPIO3	LP_GPIO3

Name	Function 0	Function 1
GPIO4	LP_GPIO4	LP_GPIO4
GPIO5	LP_GPIO5	LP_GPIO5
GPIO6	LP_GPIO6	LP_GPIO6
GPIO7	LP_GPIO7	LP_GPIO7
GPIO8	LP_GPIO8	LP_GPIO8
GPIO9	LP_GPIO9	LP_GPIO9
GPIO10	LP_GPIO10	LP_GPIO10
GPIO11	LP_GPIO11	LP_GPIO11
GPIO12	LP_GPIO12	LP_GPIO12
GPIO13	LP_GPIO13	LP_GPIO13
GPIO14	LP_UART_TXD_PAD	LP_GPIO14
GPIO15	LP_UART_RXD_PAD	LP_GPIO15

Notice:
In LP IO MUX, unused pins must be configured to LP_GPIO function.

7.16 GPIO Pin Analog Functions List

Table 7.16-1 shows the GPIO pins and their correspond analog functions.

Table 7.16-1. GPIO Pin Analog Functions

Name	Analog Function 0	Analog Function 1
GPIO0	XTAL_32K_N	—
GPIO1	XTAL_32K_P	—
GPIO2	TOUCH_CHANNEL0	—
GPIO3	TOUCH_CHANNEL1	—
GPIO4	TOUCH_CHANNEL2	—
GPIO5	TOUCH_CHANNEL3	—
GPIO6	TOUCH_CHANNEL4	—
GPIO7	TOUCH_CHANNEL5	—
GPIO8	TOUCH_CHANNEL6	—
GPIO9	TOUCH_CHANNEL7	—
GPIO10	TOUCH_CHANNEL8	—
GPIO11	TOUCH_CHANNEL9	—
GPIO12	TOUCH_CHANNEL10	—
GPIO13	TOUCH_CHANNEL11	—
GPIO14	TOUCH_CHANNEL12	—
GPIO15	TOUCH_CHANNEL13	—
GPIO16	ADC1_CHANNEL0	—
GPIO17	ADC1_CHANNEL1	—
GPIO18	ADC1_CHANNEL2	—

Name	Analog Function 0	Analog Function 1
GPIO19	ADC1_CHANNEL3	—
GPIO20	ADC1_CHANNEL4	—
GPIO21	ADC1_CHANNEL5	—
GPIO22	ADC1_CHANNEL6	—
GPIO23	ADC1_CHANNEL7	—
GPIO24	USB1P1_N0	—
GPIO25	USB1P1_P0	—
GPIO26	USB1P1_N1	—
GPIO27	USB1P1_P1	—
GPIO49	ADC2_CHANNEL2	—
GPIO50	ADC2_CHANNEL3	—
GPIO51	ADC2_CHANNEL4	ANA_COMP0
GPIO52	ADC2_CHANNEL5	ANA_COMP0
GPIO53	ADC2_CHANNEL6	ANA_COMP1
GPIO54	ADC2_CHANNEL7	ANA_COMP1

7.17 Event Task Matrix Function

In ESP32-P4, GPIO supports ETM function, that is, the ETM task of GPIO can be triggered by the ETM event of any peripheral, or the ETM task of any peripheral can be triggered by the ETM event of GPIO. For more details about ETM, please refer to Chapter 11 *Event Task Matrix (ETM)*. Only ETM tasks and ETM events related to GPIO are introduced here.

The GPIO ETM provides eight task channels *x* (0 ~ 7). The ETM tasks that each task channel can receive are:

- GPIO_TASK_CH*x*_SET: GPIO goes high when triggered.
- GPIO_TASK_CH*x*_CLEAR: GPIO goes low when triggered.
- GPIO_TASK_CH*x*_TOGGLE: GPIO toggles level when triggered.

Below is an example to configure task channel *x* to control GPIO*y*:

- Configure IO_MUX_GPIO*y*_MCU_SEL to 1, to select Function 1 listed in Table 7.14-1.
- Configure GPIO_ENABLE_REG[*y*] to 1.
- Configure GPIO_EXT_ETM_TASK_GPIO*y*_SEL to *x*.
- Set GPIO_EXT_ETM_TASK_GPIO*y*_EN, to enable ETM task channel *x* to control GPIO*y*.

Note:

- One task channel can be selected by one or more GPIOs.
- When two or three of the signals GPIO_TASK_CH*x*_SET, GPIO_TASK_CH*x*_CLEAR, and GPIO_TASK_CH*x*_TOGGLE of the task channel *x* selected by GPIO*y* are valid at the same time, then GPIO_TASK_CH*x*_SET has the highest priority, GPIO_TASK_CH*x*_CLEAR takes the second higher priority, and GPIO_TASK_CH*x*_TOGGLE has the lowest priority.

- When GPIO*y* is controlled by ETM task channel, the values of GPIO_OUT_REG, GPIO_FUNC*n*_OUT_INV_SEL, and GPIO_FUNC*n*_OUT_SEL may be modified by the hardware. For such reason, it's recommended to reconfigure these registers when the GPIO is free from the control of ETM task channel.

GPIO has eight event channels, and the ETM events that each event channel can generate are:

- GPIO_EVT_CH*x*_RISE_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 7.3-1) has a rising edge.
- GPIO_EVT_CH*x*_FALL_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 7.3-1) has a falling edge.
- GPIO_EVT_CH*x*_ANY_EDGE: Indicates that the output signal of the corresponding GPIO Filter (see Figure 7.3-1) is reversed.

The specific configuration of the event channel is as follows:

- Set GPIO_EXT_ETM_CH*x*_EVENT_EN to enable event channel *x* (0 ~ 7).
- Configure GPIO_EXT_ETM_CH*x*_EVENT_SEL to *y* (0 ~ 54), i.e., select one from the 55 GPIOs.

Note:
One GPIO can be selected by one or more event channels.

In specific applications, GPIO ETM events can be used to trigger GPIO ETM tasks. For example, event channel 0 selects GPIO0, GPIO1 selects task channel 0, and the GPIO_EVT_CH*0*_RISE_EDGE event is used to trigger the GPIO_TASK_CH*0*_TOGGLE task. When a square wave signal is input to the chip through GPIO0, the chip outputs a square wave signal with a frequency divided by 2 through GPIO1.

7.18 Interrupts

ESP32-P4's HP IO MUX and HP GPIO matrix can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- GPIO_INTR0
- GPIO_INTR1
- GPIO_INTR2
- GPIO_INTR3
- GPIO_PAD_COMP_INTR

There are several internal interrupt sources from HP IO MUX and HP GPIO matrix that can generate the above interrupt signals. The interrupt sources from HP IO MUX and HP GPIO matrix are listed with their trigger conditions and the resulted interrupt signals in Table 7.18-1. For more detailed information about GPIO_PAD_COMP_INTR, please refer to Chapter 46 *Analog Voltage Comparator*.

Table 7.18-1. HP IO MUX and HP GPIO Matrix's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal

GPIO_INTR0	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[0] (n: 0 ~ 31)	GPIO_INTR0
GPIO_INTR0	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[0] (n: 0 ~ 22)	GPIO_INTR0
GPIO_INTR1	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[1] (n: 0 ~ 31)	GPIO_INTR1
GPIO_INTR1	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[1] (n: 0 ~ 22)	GPIO_INTR1
GPIO_INTR2	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[3] (n: 0 ~ 31)	GPIO_INTR2
GPIO_INTR2	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[3] (n: 0 ~ 22)	GPIO_INTR2
GPIO_INTR3	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[4] (n: 0 ~ 31)	GPIO_INTR3
GPIO_INTR3	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[4] (n: 0 ~ 22)	GPIO_INTR3

ESP32-P4's LP IO MUX and GPIO matrix can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- LP_GPIO_INTR0

The interrupt source from LP IO MUX and LP GPIO matrix are listed with their trigger conditions and the resulted interrupt signals in Table 7.18-2.

Table 7.18-2. LP IO MUX and LP GPIO Matrix's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
LP_GPIO_INTR	LP_GPIO_STATUS_DATA[*n*] (n: 0 ~ 15)	LP_GPIO_INTR

7.19 Register Summary

7.19.1 HP GPIO Matrix Register Summary

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

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

Name	Description	Address	Access
Configuration Registers			
GPIO_OUT_REG	GPIO0 ~ GPIO31 output register	0x0004	R/W/SC/WTC
GPIO_OUT_W1TS_REG	GPIO0 ~ GPIO31 output set register	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO0 ~ GPIO31 output clear register	0x000C	WT
GPIO_OUT1_REG	GPIO32 ~ GPIO56 output register	0x0010	R/W/SC/WTC
GPIO_OUT1_W1TS_REG	GPIO32 ~ GPIO56 output set register	0x0014	WT

Name	Description	Address	Access
GPIO_OUT1_W1TC_REG	GPIO32 ~ GPIO56 output clear register	0x0018	WT
GPIO_ENABLE_REG	GPIO0 ~ GPIO31 output enable register	0x0020	R/W/WTC
GPIO_ENABLE_W1TS_REG	GPIO0 ~ GPIO31 output enable set register	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO0 ~ GPIO31 output enable clear register	0x0028	WT
GPIO_ENABLE1_REG	GPIO32 ~ GPIO56 output enable register	0x002C	R/W/WTC
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ GPIO56 output enable set register	0x0030	WT
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ GPIO56 output enable clear register	0x0034	WT
GPIO_STRAP_REG	Strapping pin register	0x0038	RO
GPIO_IN_REG	GPIO0 ~ GPIO31 input register	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ GPIO56 input register	0x0040	RO
GPIO_STATUS_REG	GPIO0 ~ GPIO31 interrupt status register	0x0044	R/W/WTC
GPIO_STATUS_W1TS_REG	GPIO0 ~ GPIO31 interrupt status set register	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO0 ~ GPIO31 interrupt status clear register	0x004C	WT
GPIO_STATUS1_REG	GPIO32 ~ GPIO56 interrupt status register	0x0050	R/W/WTC
GPIO_STATUS1_W1TS_REG	GPIO32 ~ GPIO56 interrupt status set register	0x0054	WT
GPIO_STATUS1_W1TC_REG	GPIO32 ~ GPIO56 interrupt status clear register	0x0058	WT
GPIO_INTR_0_REG	GPIO0 ~ GPIO31 interrupt 0 status register	0x005C	RO
GPIO_INTR1_0_REG	GPIO32 ~ GPIO56 interrupt 0 status register	0x0060	RO
GPIO_INTR_1_REG	GPIO0 ~ GPIO31 interrupt 1 status register	0x0064	RO
GPIO_INTR1_1_REG	GPIO32 ~ GPIO56 interrupt 1 status register	0x0068	RO
GPIO_STATUS_NEXT_REG	GPIO0 ~ GPIO31 interrupt source register	0x006C	RO
GPIO_STATUS_NEXT1_REG	GPIO32 ~ GPIO56 interrupt source register	0x0070	RO
GPIO_PIN0_REG	GPIO pin0 configuration register	0x0074	R/W
GPIO_PIN1_REG	GPIO pin1 configuration register	0x0078	R/W
GPIO_PIN2_REG	GPIO pin2 configuration register	0x007C	R/W
GPIO_PIN3_REG	GPIO pin3 configuration register	0x0080	R/W
GPIO_PIN4_REG	GPIO pin4 configuration register	0x0084	R/W
...	...	...	...
GPIO_PIN54_REG	GPIO pin54 configuration register	0x014C	R/W
GPIO_PIN55_REG	GPIO pin55 configuration register	0x0150	R/W
GPIO_PIN56_REG	GPIO pin56 configuration register	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Input signal 1 configuration register	0x015C	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	Input signal 2 configuration register	0x0160	R/W
GPIO_FUNC3_IN_SEL_CFG_REG	Input signal 3 configuration register	0x0164	R/W
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	Input signal 253 configuration register	0x054C	R/W
GPIO_FUNC254_IN_SEL_CFG_REG	Input signal 254 configuration register	0x0550	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Input signal 255 configuration register	0x0554	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 output configuration register	0x0558	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 output configuration register	0x055C	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 output configuration register	0x0560	varies
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 output configuration register	0x0564	varies
...	...	...	...

Name	Description	Address	Access
GPIO_FUNC53_OUT_SEL_CFG_REG	GPIO53 output configuration register	0x062C	varies
GPIO_FUNC54_OUT_SEL_CFG_REG	GPIO54 output configuration register	0x0630	varies
GPIO_FUNC55_OUT_SEL_CFG_REG	GPIO55 output configuration register	0x0634	varies
GPIO_FUNC56_OUT_SEL_CFG_REG	GPIO56 output configuration register	0x0638	varies
GPIO_INTR_2_REG	GPIO0 ~ GPIO31 interrupt 2 status register	0x063C	RO
GPIO_INTR1_2_REG	GPIO32 ~ GPIO56 interrupt 2 status register	0x0640	RO
GPIO_INTR_3_REG	GPIO0 ~ GPIO31 interrupt 3 status register	0x0644	RO
GPIO_INTR1_3_REG	GPIO32 ~ GPIO56 interrupt 3 status register	0x0648	RO
GPIO_CLOCK_GATE_REG	GPIO clock gate register	0x064C	R/W
GPIO_ZERO_DETO_FILTER_CNT_REG	GPIO analog comparator zero detect filter count	0x0710	R/W
GPIO_ZERO_DET1_FILTER_CNT_REG	GPIO analog comparator zero detect filter count	0x0714	R/W
GPIO Interrupt Registers			
GPIO_INT_RAW_REG	Analog comparator interrupt raw register	0x0700	R/WTC/SS
GPIO_INT_ST_REG	Analog comparator interrupt status register	0x0704	RO
GPIO_INT_ENA_REG	Analog comparator interrupt enable register	0x0708	R/W
GPIO_INT_CLR_REG	Analog comparator interrupt clear register	0x070C	WT
Version Register			
GPIO_DATE_REG	Version control register	0x07FC	R/W

7.19.2 HP IO MUX Register Summary

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

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

Name	Description	Address	Access
IO MUX Configuration Registers			
IO_MUX_GPIO0_REG	IO MUX configuration register for GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	IO MUX configuration register for GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	IO MUX configuration register for GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	IO MUX configuration register for GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	IO MUX configuration register for GPIO4	0x0014	R/W
IO_MUX_GPIO5_REG	IO MUX configuration register for GPIO5	0x0018	R/W
IO_MUX_GPIO6_REG	IO MUX configuration register for GPIO6	0x001C	R/W
IO_MUX_GPIO7_REG	IO MUX configuration register for GPIO7	0x0020	R/W
IO_MUX_GPIO8_REG	IO MUX configuration register for GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	IO MUX configuration register for GPIO9	0x0028	R/W
IO_MUX_GPIO10_REG	IO MUX configuration register for GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	IO MUX configuration register for GPIO11	0x0030	R/W
IO_MUX_GPIO12_REG	IO MUX configuration register for GPIO12	0x0034	R/W
IO_MUX_GPIO13_REG	IO MUX configuration register for GPIO13	0x0038	R/W
IO_MUX_GPIO14_REG	IO MUX configuration register for GPIO14	0x003C	R/W
IO_MUX_GPIO15_REG	IO MUX configuration register for GPIO15	0x0040	R/W

Name	Description	Address	Access
IO_MUX_GPIO16_REG	IO MUX configuration register for GPIO16	0x0044	R/W
IO_MUX_GPIO17_REG	IO MUX configuration register for GPIO17	0x0048	R/W
IO_MUX_GPIO18_REG	IO MUX configuration register for GPIO18	0x004C	R/W
IO_MUX_GPIO19_REG	IO MUX configuration register for GPIO19	0x0050	R/W
IO_MUX_GPIO20_REG	IO MUX configuration register for GPIO20	0x0054	R/W
IO_MUX_GPIO21_REG	IO MUX configuration register for GPIO21	0x0058	R/W
IO_MUX_GPIO22_REG	IO MUX configuration register for GPIO22	0x005C	R/W
IO_MUX_GPIO23_REG	IO MUX configuration register for GPIO23	0x0060	R/W
IO_MUX_GPIO24_REG	IO MUX configuration register for GPIO24	0x0064	R/W
IO_MUX_GPIO25_REG	IO MUX configuration register for GPIO25	0x0068	R/W
IO_MUX_GPIO26_REG	IO MUX configuration register for GPIO26	0x006C	R/W
IO_MUX_GPIO27_REG	IO MUX configuration register for GPIO27	0x0070	R/W
IO_MUX_GPIO28_REG	IO MUX configuration register for GPIO28	0x0074	R/W
IO_MUX_GPIO29_REG	IO MUX configuration register for GPIO29	0x0078	R/W
IO_MUX_GPIO30_REG	IO MUX configuration register for GPIO30	0x007C	R/W
IO_MUX_GPIO31_REG	IO MUX configuration register for GPIO31	0x0080	R/W
IO_MUX_GPIO32_REG	IO MUX configuration register for GPIO32	0x0084	R/W
IO_MUX_GPIO33_REG	IO MUX configuration register for GPIO33	0x0088	R/W
IO_MUX_GPIO34_REG	IO MUX configuration register for GPIO34	0x008C	R/W
IO_MUX_GPIO35_REG	IO MUX configuration register for GPIO35	0x0090	R/W
IO_MUX_GPIO36_REG	IO MUX configuration register for GPIO36	0x0094	R/W
IO_MUX_GPIO37_REG	IO MUX configuration register for GPIO37	0x0098	R/W
IO_MUX_GPIO38_REG	IO MUX configuration register for GPIO38	0x009C	R/W
IO_MUX_GPIO39_REG	IO MUX configuration register for GPIO39	0x00A0	R/W
IO_MUX_GPIO40_REG	IO MUX configuration register for GPIO40	0x00A4	R/W
IO_MUX_GPIO41_REG	IO MUX configuration register for GPIO41	0x00A8	R/W
IO_MUX_GPIO42_REG	IO MUX configuration register for GPIO42	0x00AC	R/W
IO_MUX_GPIO43_REG	IO MUX configuration register for GPIO43	0x00B0	R/W
IO_MUX_GPIO44_REG	IO MUX configuration register for GPIO44	0x00B4	R/W
IO_MUX_GPIO45_REG	IO MUX configuration register for GPIO45	0x00B8	R/W
IO_MUX_GPIO46_REG	IO MUX configuration register for GPIO46	0x00BC	R/W
IO_MUX_GPIO47_REG	IO MUX configuration register for GPIO47	0x00C0	R/W
IO_MUX_GPIO48_REG	IO MUX configuration register for GPIO48	0x00C4	R/W
IO_MUX_GPIO49_REG	IO MUX configuration register for GPIO49	0x00C8	R/W
IO_MUX_GPIO50_REG	IO MUX configuration register for GPIO50	0x00CC	R/W
IO_MUX_GPIO51_REG	IO MUX configuration register for GPIO51	0x00D0	R/W
IO_MUX_GPIO52_REG	IO MUX configuration register for GPIO52	0x00D4	R/W
IO_MUX_GPIO53_REG	IO MUX configuration register for GPIO53	0x00D8	R/W
IO_MUX_GPIO54_REG	IO MUX configuration register for GPIO54	0x00DC	R/W
Version Register			
IO_MUX_DATE_REG	Version control register	0x0104	R/W

7.19.3 GPIO EXT Register Summary

GPIO EXT registers consist of SDM registers, Glitch Filter registers, and ETM registers.

The addresses in this section are relative to (HP GPIO matrix base address + 0x0F00). GPIO base address is provided in Table 5.3-2 in Chapter *5 System and Memory*.

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

Name	Description	Address	Access
SDM Configuration Registers			
GPIO_EXT_SIGMADELTA0_REG	SDM channel 0 duty cycle configuration register	0x0000	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM channel 1 duty cycle configuration register	0x0004	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM channel 2 duty cycle configuration register	0x0008	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM channel 3 duty cycle configuration register	0x000C	R/W
GPIO_EXT_SIGMADELTA4_REG	SDM channel 4 duty cycle configuration register	0x0010	R/W
GPIO_EXT_SIGMADELTA5_REG	SDM channel 5 duty cycle configuration register	0x0014	R/W
GPIO_EXT_SIGMADELTA6_REG	SDM channel 6 duty cycle configuration register	0x0018	R/W
GPIO_EXT_SIGMADELTA7_REG	SDM channel 7 duty cycle configuration register	0x001C	R/W
GPIO_EXT_SIGMADELTA_MISC_REG	MISC register	0x0024	R/W
Glitch Filter Configuration Registers			
GPIO_EXT_GLITCH_FILTER_CH0_REG	Glitch Filter configuration register for channel 0	0x0030	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	Glitch Filter configuration register for channel 1	0x0034	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	Glitch Filter configuration register for channel 2	0x0038	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	Glitch Filter configuration register for channel 3	0x003C	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	Glitch Filter configuration register for channel 4	0x0040	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	Glitch Filter configuration register for channel 5	0x0044	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	Glitch Filter configuration register for channel 6	0x0048	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	Glitch Filter configuration register for channel 7	0x004C	R/W
ETM Configuration Registers			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM configuration register for channel 0	0x0060	R/W

Name	Description	Address	Access
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM configuration register for channel 1	0x0064	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM configuration register for channel 2	0x0068	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM configuration register for channel 3	0x006C	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM configuration register for channel 4	0x0070	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM configuration register for channel 5	0x0074	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM configuration register for channel 6	0x0078	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM configuration register for channel 7	0x007C	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	GPIO selection register 0 for ETM	0x00A0	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	GPIO selection register 1 for ETM	0x00A4	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	GPIO selection register 2 for ETM	0x00A8	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	GPIO selection register 3 for ETM	0x00AC	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	GPIO selection register 4 for ETM	0x00B0	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	GPIO selection register 5 for ETM	0x00B4	R/W
GPIO_EXT_ETM_TASK_P6_CFG_REG	GPIO selection register 6 for ETM	0x00B8	R/W
GPIO_EXT_ETM_TASK_P7_CFG_REG	GPIO selection register 7 for ETM	0x00BC	R/W
GPIO_EXT_ETM_TASK_P8_CFG_REG	GPIO selection register 8 for ETM	0x00C0	R/W
GPIO_EXT_ETM_TASK_P9_CFG_REG	GPIO selection register 9 for ETM	0x00C4	R/W
GPIO_EXT_ETM_TASK_P10_CFG_REG	GPIO selection register 10 for ETM	0x00C8	R/W
GPIO_EXT_ETM_TASK_P11_CFG_REG	GPIO selection register 11 for ETM	0x00CC	R/W
GPIO_EXT_ETM_TASK_P12_CFG_REG	GPIO selection register 12 for ETM	0x00D0	R/W
GPIO_EXT_ETM_TASK_P13_CFG_REG	GPIO selection register 13 for ETM	0x00D4	R/W
Version Register			
GPIO_EXT_VERSION_REG	Version control register	0x00FC	R/W

7.19.4 LP GPIO Matrix Register Summary

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

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

Name	Description	Address	Access
Clock Enable Register			
LP_GPIO_CLK_EN_REG	LP GPIO clock gate register	0x0000	R/W
Configuration Registers			
LP_GPIO_OUT_REG	LP GPIO output register	0x0008	R/W/WTC
LP_GPIO_OUT_W1TS_REG	LP GPIO output set register	0x000C	WT
LP_GPIO_OUT_W1TC_REG	LP GPIO output clear register	0x0010	WT
LP_GPIO_ENABLE_REG	LP GPIO output enable register	0x0014	R/W/WTC
LP_GPIO_ENABLE_W1TS_REG	LP GPIO output enable set register	0x0018	WT
LP_GPIO_ENABLE_W1TC_REG	LP GPIO output enable clear register	0x001C	WT
LP_GPIO_STATUS_REG	LP GPIO interrupt status register	0x0020	R/W/WTC
LP_GPIO_STATUS_W1TS_REG	LP GPIO interrupt status set register	0x0024	WT
LP_GPIO_STATUS_W1TC_REG	LP GPIO interrupt status clear register	0x0028	WT

Name	Description	Address	Access
LP_GPIO_STATUS_NEXT_REG	LP GPIO interrupt source register	0x002C	RO
LP_GPIO_IN_REG	LP GPIO input register	0x0030	RO
LP_GPIO_PIN0_REG	LP GPIO0 configuration register	0x0034	varies
LP_GPIO_PIN1_REG	LP GPIO1 configuration register	0x0038	varies
LP_GPIO_PIN2_REG	LP GPIO2 configuration register	0x003C	varies
LP_GPIO_PIN3_REG	LP GPIO3 configuration register	0x0040	varies
LP_GPIO_PIN4_REG	LP GPIO4 configuration register	0x0044	varies
LP_GPIO_PIN5_REG	LP GPIO5 configuration register	0x0048	varies
LP_GPIO_PIN6_REG	LP GPIO6 configuration register	0x004C	varies
LP_GPIO_PIN7_REG	LP GPIO7 configuration register	0x0050	varies
LP_GPIO_PIN8_REG	LP GPIO8 configuration register	0x0054	varies
LP_GPIO_PIN9_REG	LP GPIO9 configuration register	0x0058	varies
LP_GPIO_PIN10_REG	LP GPIO10 configuration register	0x005C	varies
LP_GPIO_PIN11_REG	LP GPIO11 configuration register	0x0060	varies
LP_GPIO_PIN12_REG	LP GPIO12 configuration register	0x0064	varies
LP_GPIO_PIN13_REG	LP GPIO13 configuration register	0x0068	varies
LP_GPIO_PIN14_REG	LP GPIO14 configuration register	0x006C	varies
LP_GPIO_PIN15_REG	LP GPIO15 configuration register	0x0070	varies
LP_GPIO_FUNC0_IN_SEL_CFG_REG	Configuration register for input signal 0	0x0074	R/W
LP_GPIO_FUNC1_IN_SEL_CFG_REG	Configuration register for input signal 1	0x0078	R/W
LP_GPIO_FUNC2_IN_SEL_CFG_REG	Configuration register for input signal 2	0x007C	R/W
LP_GPIO_FUNC3_IN_SEL_CFG_REG	Configuration register for input signal 3	0x0080	R/W
LP_GPIO_FUNC4_IN_SEL_CFG_REG	Configuration register for input signal 4	0x0084	R/W
LP_GPIO_FUNC5_IN_SEL_CFG_REG	Configuration register for input signal 5	0x0088	R/W
LP_GPIO_FUNC6_IN_SEL_CFG_REG	Configuration register for input signal 6	0x008C	R/W
LP_GPIO_FUNC7_IN_SEL_CFG_REG	Configuration register for input signal 7	0x0090	R/W
LP_GPIO_FUNC8_IN_SEL_CFG_REG	Configuration register for input signal 8	0x0094	R/W
LP_GPIO_FUNC9_IN_SEL_CFG_REG	Configuration register for input signal 9	0x0098	R/W
LP_GPIO_FUNC10_IN_SEL_CFG_REG	Configuration register for input signal 10	0x009C	R/W
LP_GPIO_FUNC11_IN_SEL_CFG_REG	Configuration register for input signal 11	0x00A0	R/W
LP_GPIO_FUNC12_IN_SEL_CFG_REG	Configuration register for input signal 12	0x00A4	R/W
LP_GPIO_FUNC13_IN_SEL_CFG_REG	Configuration register for input signal 13	0x00A8	R/W
LP_GPIO_FUNC0_OUT_SEL_CFG_REG	Configuration register for LP GPIO0 output	0x00F4	R/W
LP_GPIO_FUNC1_OUT_SEL_CFG_REG	Configuration register for LP GPIO1 output	0x00F8	R/W
LP_GPIO_FUNC2_OUT_SEL_CFG_REG	Configuration register for LP GPIO2 output	0x00FC	R/W
LP_GPIO_FUNC3_OUT_SEL_CFG_REG	Configuration register for LP GPIO3 output	0x0100	R/W
LP_GPIO_FUNC4_OUT_SEL_CFG_REG	Configuration register for LP GPIO4 output	0x0104	R/W
LP_GPIO_FUNC5_OUT_SEL_CFG_REG	Configuration register for LP GPIO5 output	0x0108	R/W
LP_GPIO_FUNC6_OUT_SEL_CFG_REG	Configuration register for LP GPIO6 output	0x010C	R/W
LP_GPIO_FUNC7_OUT_SEL_CFG_REG	Configuration register for LP GPIO7 output	0x0110	R/W
LP_GPIO_FUNC8_OUT_SEL_CFG_REG	Configuration register for LP GPIO8 output	0x0114	R/W
LP_GPIO_FUNC9_OUT_SEL_CFG_REG	Configuration register for LP GPIO9 output	0x0118	R/W
LP_GPIO_FUNC10_OUT_SEL_CFG_REG	Configuration register for LP GPIO10 output	0x011C	R/W

Name	Description	Address	Access
LP_GPIO_FUNC11_OUT_SEL_CFG_REG	Configuration register for LP GPIO11 output	0x0120	R/W
LP_GPIO_FUNC12_OUT_SEL_CFG_REG	Configuration register for LP GPIO12 output	0x0124	R/W
LP_GPIO_FUNC13_OUT_SEL_CFG_REG	Configuration register for LP GPIO13 output	0x0128	R/W
LP_GPIO_FUNC14_OUT_SEL_CFG_REG	Configuration register for LP GPIO14 output	0x012C	R/W
LP_GPIO_FUNC15_OUT_SEL_CFG_REG	Configuration register for LP GPIO15 output	0x0130	R/W
Version Register			
LP_GPIO_VER_DATE_REG	Version control register	0x0004	R/W

7.19.5 LP IO MUX Register Summary

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

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

Name	Description	Address	Access
Clock Enable Register			
LP_IOMUX_CLK_EN_REG	LP IO MUX clock gate register	0x0000	R/W
Configuration Registers			
LP_IOMUX_PAD0_REG	LP IO MUX configuration register for GPIO0	0x0008	R/W
LP_IOMUX_PAD1_REG	LP IO MUX configuration register for GPIO1	0x000C	R/W
LP_IOMUX_PAD2_REG	LP IO MUX configuration register for GPIO2	0x0010	R/W
LP_IOMUX_PAD3_REG	LP IO MUX configuration register for GPIO3	0x0014	R/W
LP_IOMUX_PAD4_REG	LP IO MUX configuration register for GPIO4	0x0018	R/W
LP_IOMUX_PAD5_REG	LP IO MUX configuration register for GPIO5	0x001C	R/W
LP_IOMUX_PAD6_REG	LP IO MUX configuration register for GPIO6	0x0020	R/W
LP_IOMUX_PAD7_REG	LP IO MUX configuration register for GPIO7	0x0024	R/W
LP_IOMUX_PAD8_REG	LP IO MUX configuration register for GPIO8	0x0028	R/W
LP_IOMUX_PAD9_REG	LP IO MUX configuration register for GPIO9	0x002C	R/W
LP_IOMUX_PAD10_REG	LP IO MUX configuration register for GPIO10	0x0030	R/W
LP_IOMUX_PAD11_REG	LP IO MUX configuration register for GPIO11	0x0034	R/W
LP_IOMUX_PAD12_REG	LP IO MUX configuration register for GPIO12	0x0038	R/W
LP_IOMUX_PAD13_REG	LP IO MUX configuration register for GPIO13	0x003C	R/W
LP_IOMUX_PAD14_REG	LP IO MUX configuration register for GPIO14	0x0040	R/W
LP_IOMUX_PAD15_REG	LP IO MUX configuration register for GPIO15	0x0044	R/W
LP_IOMUX_LP_PAD_HOLD_REG	LP GPIO PAD hold control register	0x004C	R/W
LP_IOMUX_LP_PAD_HYS_REG	LP GPIO PAD hysteresis control register	0x0050	R/W
Version Register			
LP_IOMUX_VER_DATE_REG	Version control register	0x0004	R/W

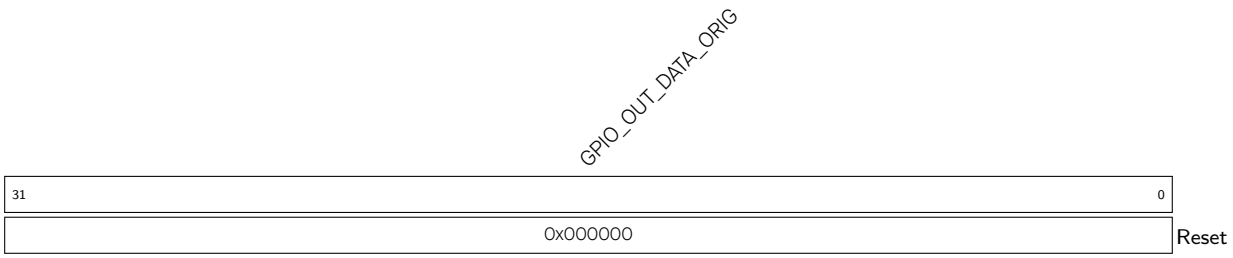
7.20 Registers

7.20.1 HP GPIO Matrix Registers

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

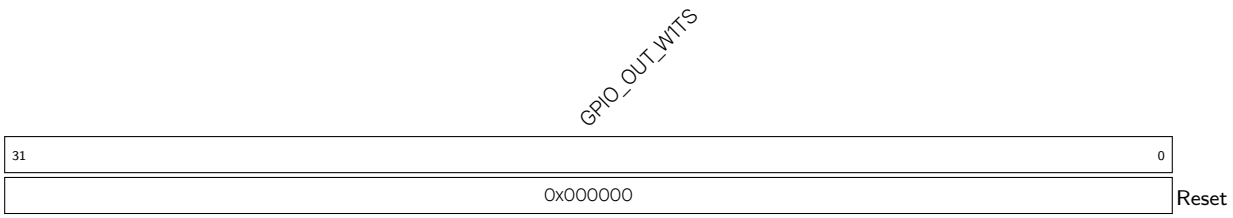
For how to program reserved fields, please refer to Section IX *Programming Reserved Register Field*.

Register 7.1. GPIO_OUT_REG (0x0004)



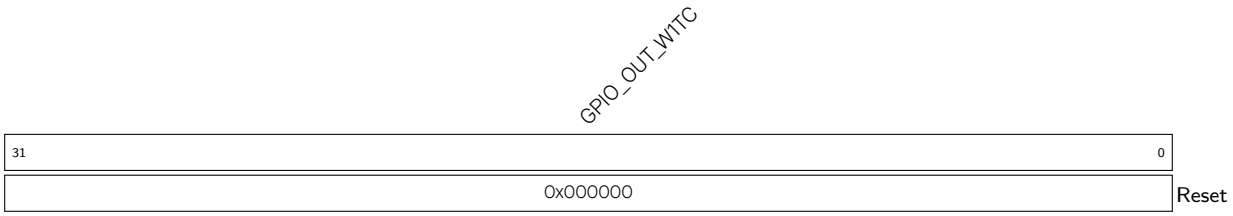
GPIO_OUT_DATA_ORIG Configures the output value of GPIO0 ~ GPIO31 output in simple GPIO output mode.
0: Low level
1: High level
The value of bit0 ~ bit31 correspond to the output value of GPIO0 ~ GPIO31 respectively. (R/W/SC/WTC)

Register 7.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS Configures whether or not to set the output register GPIO_OUT_REG of GPIO0 ~ GPIO31.
0: Not set
1: The corresponding bit in GPIO_OUT_REG will be set to 1
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to set GPIO_OUT_REG.
(WT)

Register 7.3. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC Configures whether or not to clear the output register GPIO_OUT_REG of GPIO0 ~ GPIO31 output.

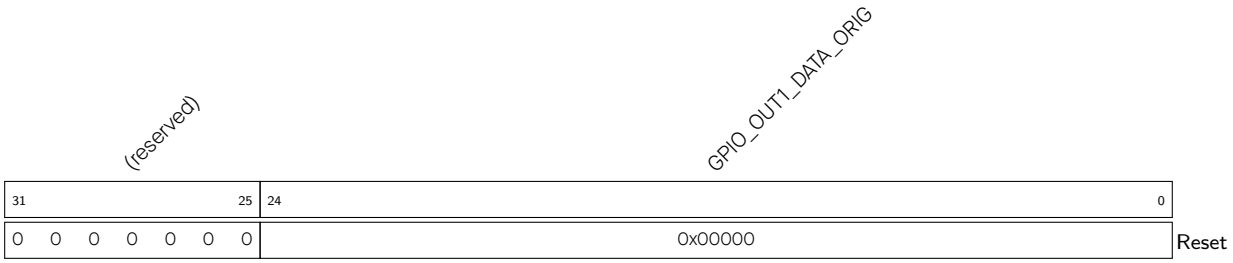
0: Not clear

1: The corresponding bit in GPIO_OUT_REG will be cleared.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 7.4. GPIO_OUT1_REG (0x0010)



GPIO_OUT1_DATA_ORIG Configures the output value of GPIO32 ~ GPIO54 output in simple GPIO output mode.

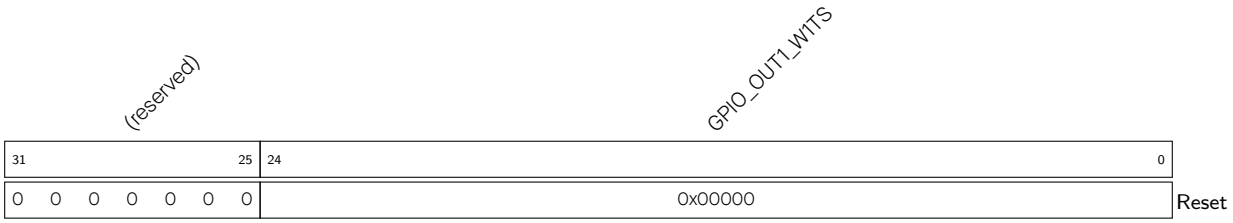
0: Low level

1: High level

The value of bit0 ~ bit22 correspond to the output value of GPIO32 ~ GPIO54 respectively.

(R/W/SC/WTC)

Register 7.5. GPIO_OUT1_W1TS_REG (0x0014)



GPIO_OUT1_W1TS Configures whether or not to set the output register GPIO_OUT1_REG of GPIO32 ~ GPIO54.

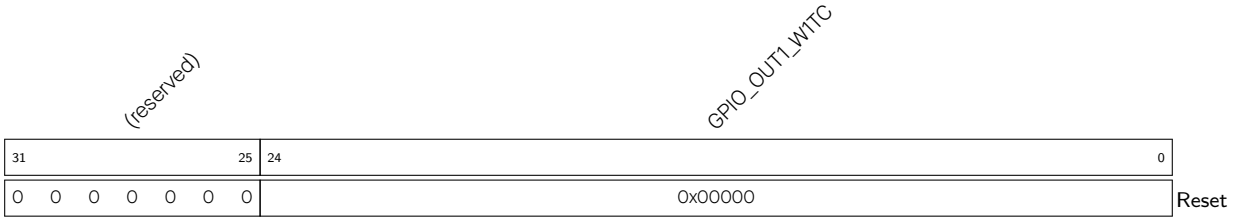
0: Not set

1: The corresponding bit in GPIO_OUT1_REG will be set to 1

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to set GPIO_OUT1_REG.

(WT)

Register 7.6. GPIO_OUT1_W1TC_REG (0x0018)



GPIO_OUT1_W1TC Configures whether or not to clear the output register GPIO_OUT1_REG of GPIO32 ~ GPIO54 output.

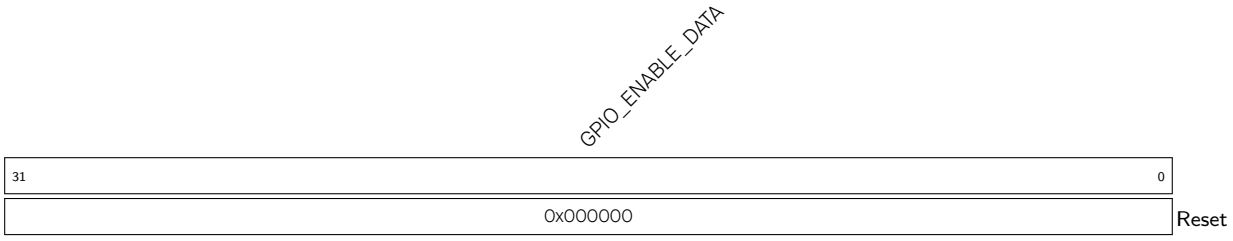
0: Not clear

1: The corresponding bit in GPIO_OUT1_REG will be cleared.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to clear GPIO_OUT_REG.

(WT)

Register 7.7. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_DATA Configures whether or not to enable the output of GPIO0 ~ GPIO31.

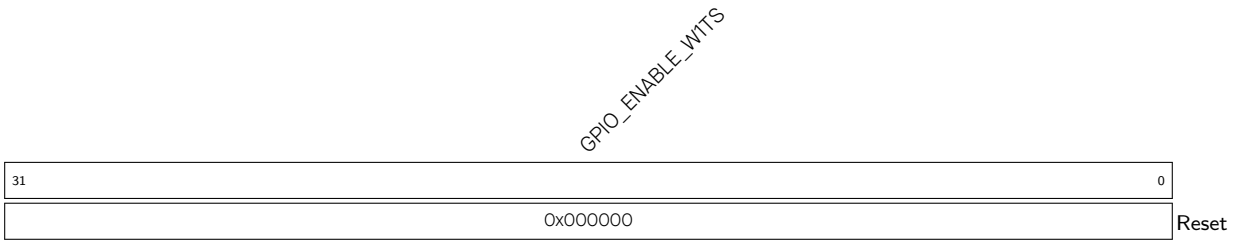
0: Not enable

1: Enable

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31

(R/W/WTC)

Register 7.8. GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS Configures whether or not to set the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO31.

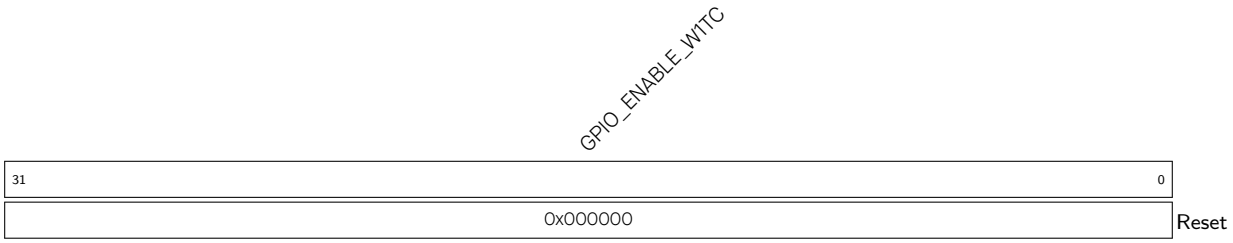
0: Not set

1: The corresponding bit in GPIO_ENABLE_REG will be set to 1

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Recommended operation: use this register to set GPIO_ENABLE_REG.

(WT)

Register 7.9. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC Configures whether or not to clear the output enable register GPIO_ENABLE_REG of GPIO0 ~ GPIO31.

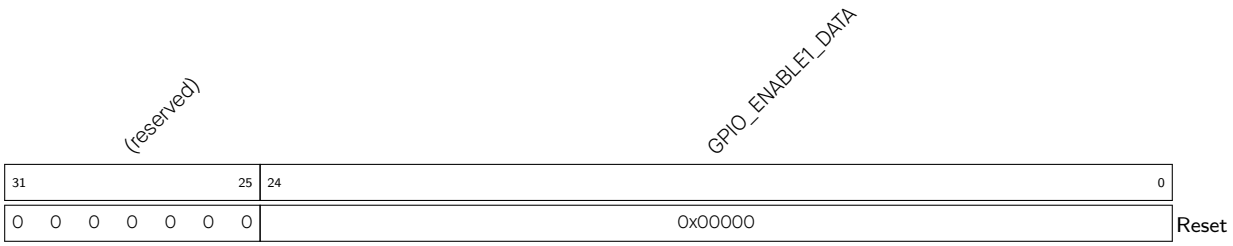
0: Not clear

1: The corresponding bit in GPIO_ENABLE_REG will be cleared

Bit0 ~ bit31 are corresponding to GPIO0 ~ 31. Recommended operation: use this register to clear GPIO_ENABLE_REG.

(WT)

Register 7.10. GPIO_ENABLE1_REG (0x002C)



GPIO_ENABLE1_DATA Configures whether or not to enable the output of GPIO32 ~ GPIO54.

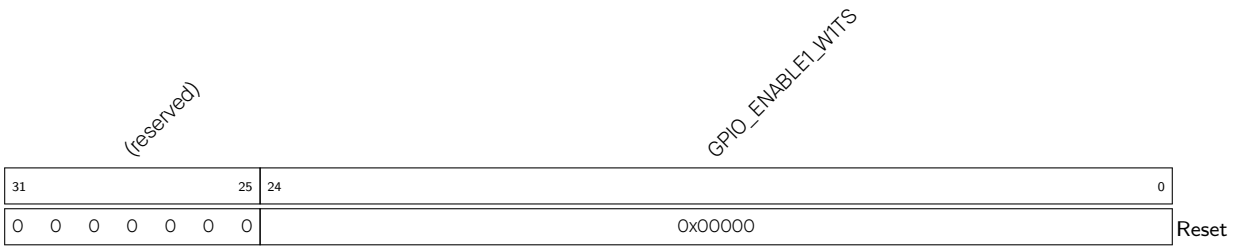
0: Not enable

1: Enable

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

(R/W/WTC)

Register 7.11. GPIO_ENABLE1_W1TS_REG (0x0030)



GPIO_ENABLE1_W1TS Configures whether or not to set the output enable register GPIO_ENABLE1_REG of GPIO32 ~ GPIO54.

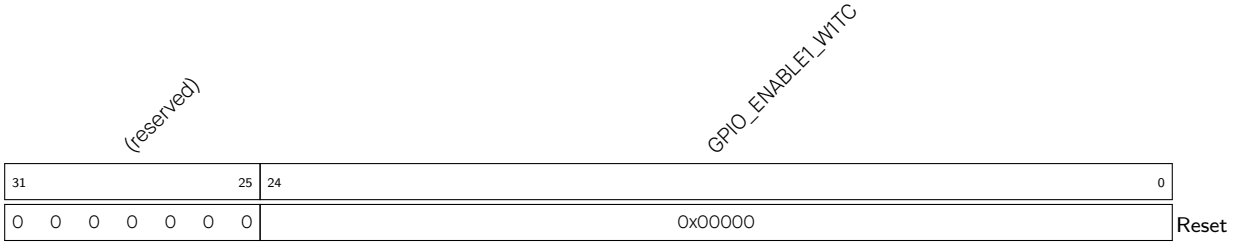
0: Not set

1: The corresponding bit in GPIO_ENABLE1_REG will be set to 1

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Recommended operation: use this register to set GPIO_ENABLE1_REG.

(WT)

Register 7.12. GPIO_ENABLE1_W1TC_REG (0x0034)



GPIO_ENABLE1_W1TC Configures whether or not to clear the output enable register GPIO_ENABLE1_REG of GPIO32 ~ GPIO54.

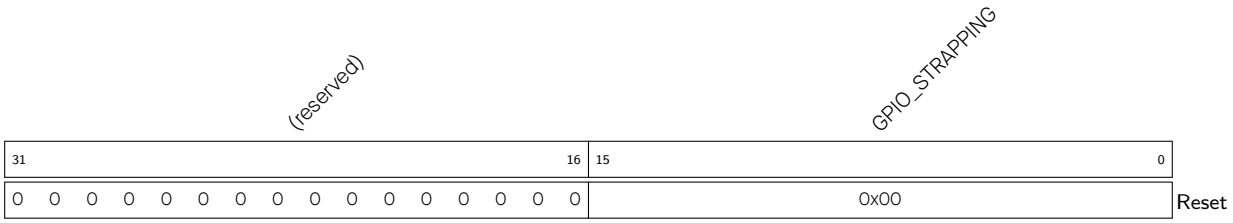
0: Not clear

1: The corresponding bit in GPIO_ENABLE1_REG will be cleared

Bit0 ~ bit22 are corresponding to GPIO32 ~ 54. Recommended operation: use this register to clear GPIO_ENABLE1_REG.

(WT)

Register 7.13. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING Represents the values of GPIO strapping pins.

bit0: GPIO38

bit1: GPIO37

bit2: GPIO36

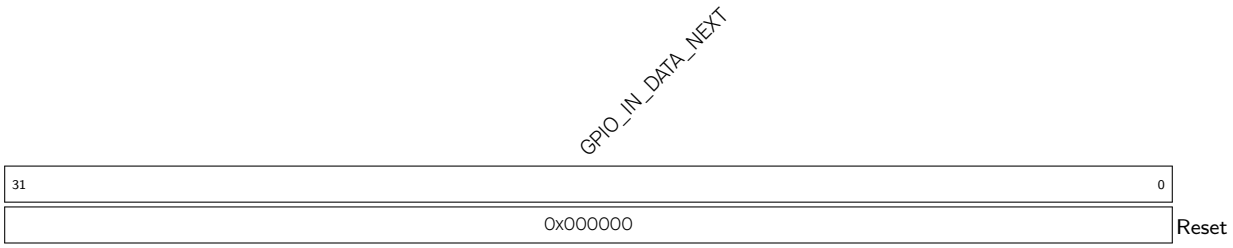
bit3: GPIO35

bit4: GPIO34

bit5 ~ bit15: invalid

For more information about the functions controlled by strapping pins, see *ESP32-P4 Datasheet* > Section *Strapping Pins*. (RO)

Register 7.14. GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT Represents the input value of GPIO0 ~ GPIO31. Each bit represents a pin input value:

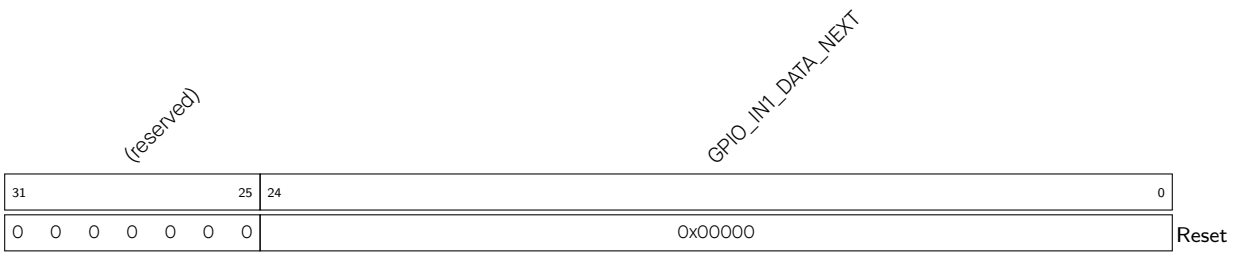
0: Low level

1: High level

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

(RO)

Register 7.15. GPIO_IN1_REG (0x0040)



GPIO_IN1_DATA_NEXT Represents the input value of GPIO32 ~ GPIO54. Each bit represents a pin input value:

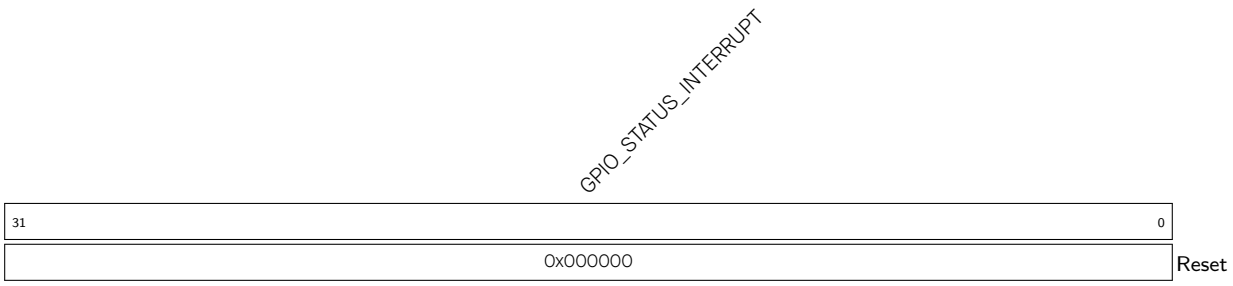
0: Low level

1: High level

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

(RO)

Register 7.16. GPIO_STATUS_REG (0x0044)



GPIO_STATUS_INTERRUPT The interrupt status of GPIO0 ~ GPIO31, can be configured by the software.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

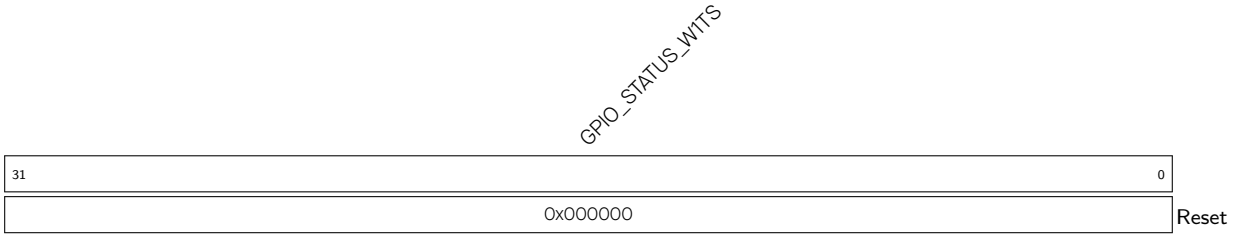
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 7.17. GPIO_STATUS_W1TS_REG (0x0048)



GPIO_STATUS_W1TS Configures whether or not to set the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO31.

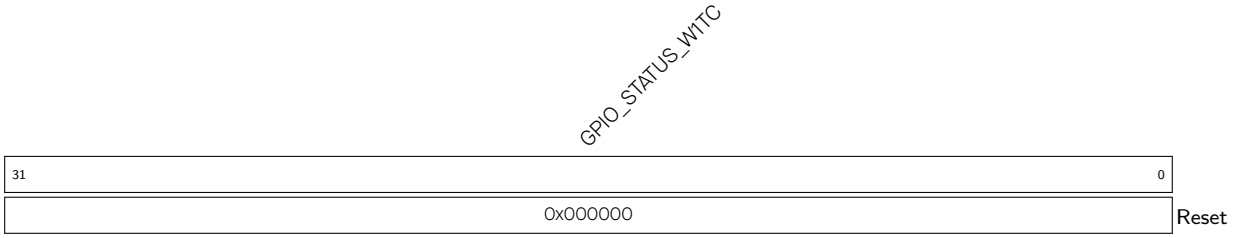
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1.

Recommended operation: use this register to set GPIO_STATUS_INTERRUPT.

(WT)

Register 7.18. GPIO_STATUS_W1TC_REG (0x004C)



GPIO_STATUS_W1TC Configures whether or not to clear the interrupt status register GPIO_STATUS_INTERRUPT of GPIO0 ~ GPIO31.

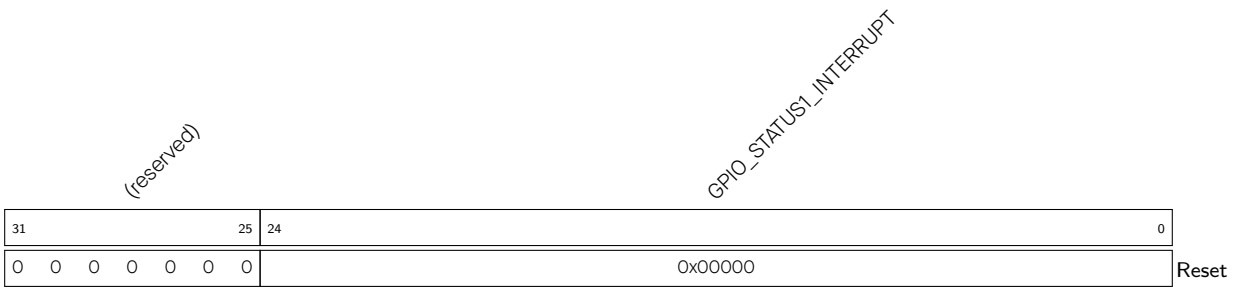
Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared.

Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT.

(WT)

Register 7.19. GPIO_STATUS1_REG (0x0050)



GPIO_STATUS1_INTERRUPT The interrupt status of GPIO32 ~ GPIO54, can be configured by the software.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

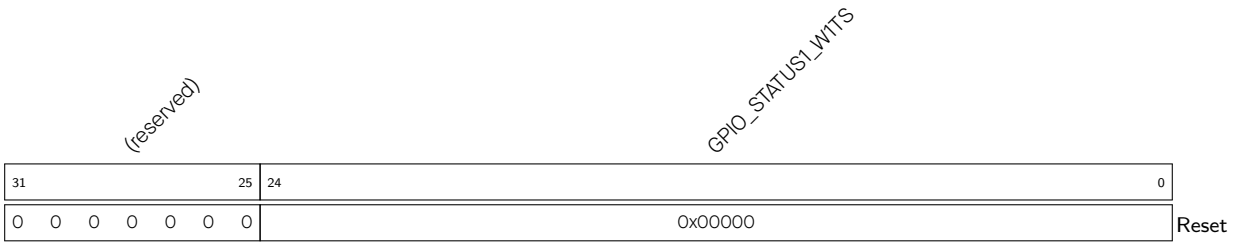
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 7.20. GPIO_STATUS1_W1TS_REG (0x0054)



GPIO_STATUS1_W1TS Configures whether or not to set the interrupt status register GPIO_STATUS1_INTERRUPT of GPIO32 ~ GPIO54.

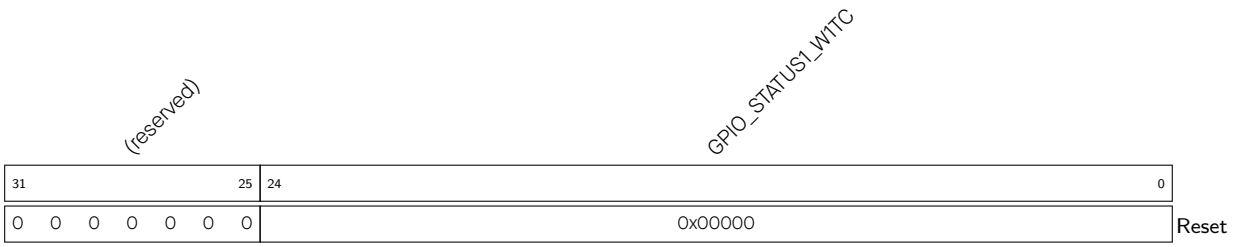
Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_INTERRUPT will be set to 1.

Recommended operation: use this register to set GPIO_STATUS1_INTERRUPT.

(WT)

Register 7.21. GPIO_STATUS1_W1TC_REG (0x0058)



GPIO_STATUS1_W1TC Configures whether or not to clear the interrupt status register GPIO_STATUS1_INTERRUPT of GPIO32 ~ GPIO54.

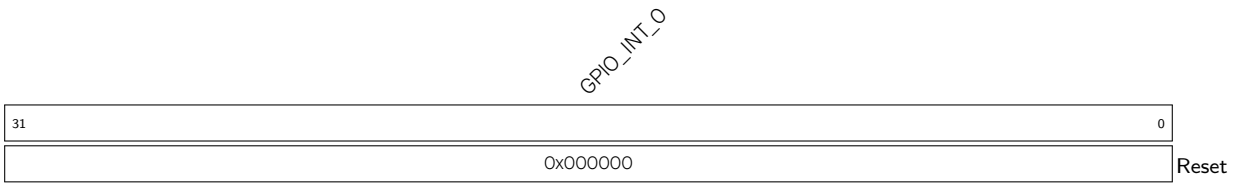
Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54.

If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_INTERRUPT will be cleared.

Recommended operation: use this register to clear GPIO_STATUS1_INTERRUPT.

(WT)

Register 7.22. GPIO_INTR_0_REG (0x005C)



GPIO_INT_0 Represents the CPU interrupt 0 status of GPIO0 ~ GPIO31. Each bit represents:

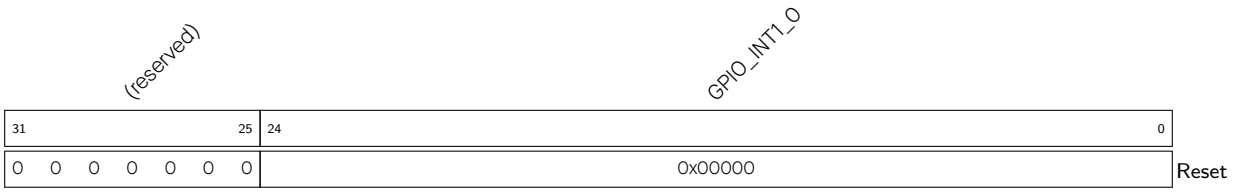
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG).

(RO)

Register 7.23. GPIO_INTR1_0_REG (0x0060)



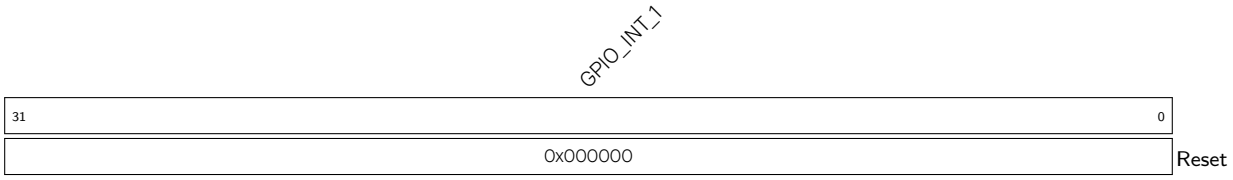
GPIO_INT1_0 Represents the CPU interrupt 0 status of GPIO32 ~ GPIO54. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 7.24. GPIO_INTR_1_REG (0x0064)



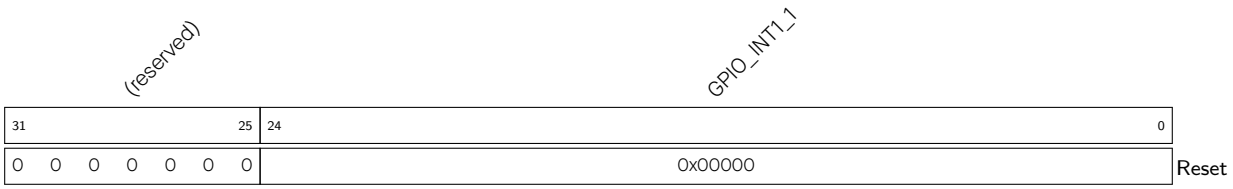
GPIO_INT_1 Represents the CPU interrupt 1 status of GPIO0 ~ GPIO31. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit14 of GPIO_PIN*n*_REG). (RO)

Register 7.25. GPIO_INTR1_1_REG (0x0068)



GPIO_INT1_1 Represents the CPU interrupt 1 status of GPIO32 ~ GPIO54. Each bit represents:

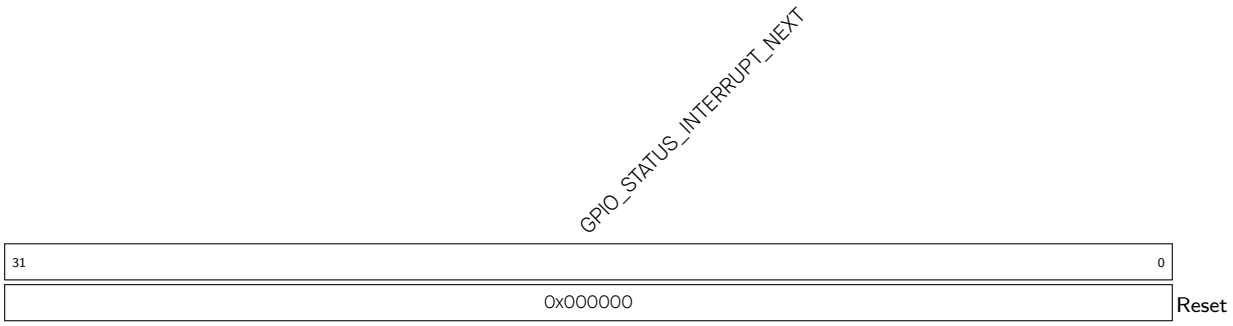
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit14 of GPIO_PIN*n*_REG).

(RO)

Register 7.26. GPIO_STATUS_NEXT_REG (0x006C)



GPIO_STATUS_INTERRUPT_NEXT Represents the interrupt source signal of GPIO0 ~ GPIO31.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. Each bit represents:

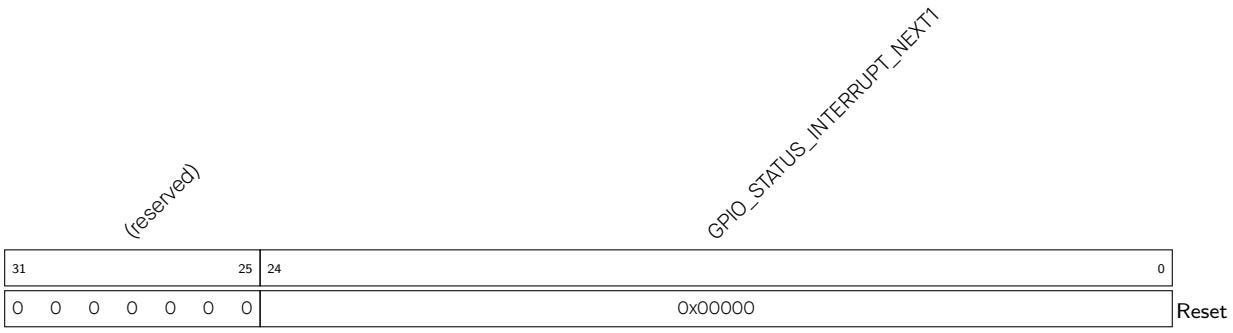
0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: The GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.

(RO)

Register 7.27. GPIO_STATUS_NEXT1_REG (0x0070)



GPIO_STATUS_INTERRUPT_NEXT1 Represents the interrupt source signal of GPIO32 ~ GPIO54. Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. Each bit represents:

0: The GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: The GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE.

The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.

(RO)

Register 7.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)



GPIO_PIN*n*_SYNC2_BYPASS Configures whether or not to synchronize GPIO input data on either edge of HP IO MUX operating clock for the second-level synchronization.

0: Not synchronize

1: Synchronize on falling edge

2: Synchronize on rising edge

3: Synchronize on rising edge

(R/W)

GPIO_PIN*n*_PAD_DRIVER Configures to select pin drive mode.

0: Normal output

1: Open drain output

(R/W)

GPIO_PIN*n*_SYNC1_BYPASS Configures whether or not to synchronize GPIO input data on either edge of HP IO MUX operating clock for the first-level synchronization.

0: Not synchronize

1: Synchronize on falling edge

2: Synchronize on rising edge

3: Synchronize on rising edge

(R/W)

GPIO_PIN*n*_INT_TYPE Configures GPIO interrupt type.

0: GPIO interrupt disabled

1: Rising edge trigger

2: Falling edge trigger

3: Any edge trigger

4: Low level trigger

5: High level trigger

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE Configures whether or not to enable GPIO wake-up function.

0: Disable

1: Enable

This function only wakes up the CPU from Light-sleep.

(R/W)

Continued on the next page...

Register 7.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)

Continued from the previous page...

GPIO_PIN*n*_INT_ENA Configures whether or not to enable CPU interrupt or CPU non-maskable interrupt.

bit13: Configures whether or not to enable CPU interrupt 0:

0: Disable

1: Enable

bit14: Configures whether or not to enable CPU interrupt 1:

0: Disable

1: Enable

bit15: invalid

bit16: Configures whether or not to enable CPU interrupt 2:

0: Disable

1: Enable

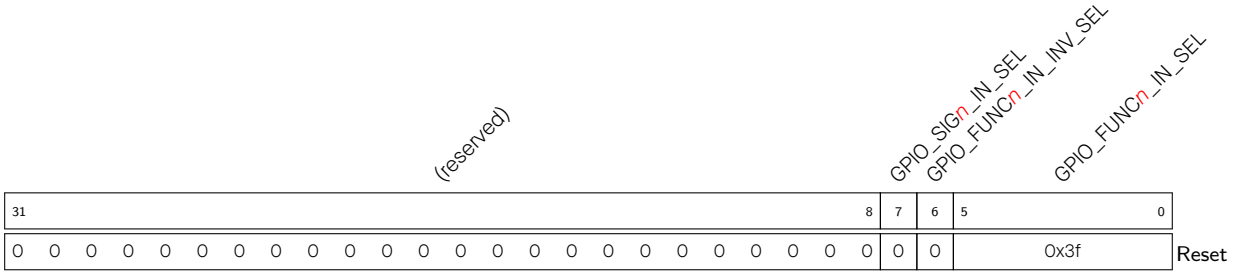
bit17: Configures whether or not to enable CPU interrupt 3:

0: Disable

1: Enable

(R/W)

Register 7.29. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 255) (0x015C+4**n*)



GPIO_FUNC*n*_IN_SEL Configures to select a pin from the 55 GPIO pins to connect the input signal *n*.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

55 ~ 61: invalid

Or

62: A constantly low input

63: A constantly high input

(R/W)

GPIO_FUNC*n*_IN_INV_SEL Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

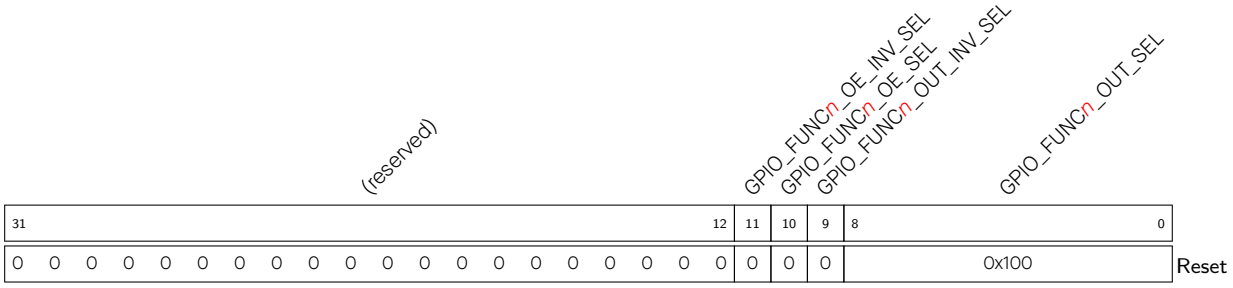
GPIO_SIG*n*_IN_SEL Configures whether or not to route signals via HP GPIO matrix.

0: Bypass HP GPIO matrix, i.e., connect signals directly to peripheral configured in HP IO MUX.

1: Route signals via HP GPIO matrix.

(R/W)

Register 7.30. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 54) (0x0558+0x4n*)**



GPIO_FUNC*n*_OUT_SEL Configures to select a signal *Y* (0 <= *Y* < 256) from 256 peripheral signals to be output to GPIO*n*.

0: Select signal 0

1: Select signal 1

.....

254: Select signal 254

255: Select signal 255

Or

256: Bit*n* of GPIO_OUT_REG and GPIO_OUT1_REG are selected as the output value, GPIO_ENABLE_REG and GPIO_ENABLE1_REG are selected as the output enable.

257 ~ 511: invalid

For the detailed signal list, see Table 7.12-1.

(R/W/SC)

GPIO_FUNC*n*_OUT_INV_SEL Configures whether or not to invert the output value.

0: Not invert

1: Invert

(R/W/SC)

GPIO_FUNC*n*_OE_SEL Configures to select the source of output enable signal.

0: Use output enable signal from peripheral.

1: Force the output enable signal to be sourced from bit*n* (*n*: 0 ~ 31) of GPIO_ENABLE_REG and bit*n*-32 (*n*: 32 ~ 54) of GPIO_ENABLE1_REG.

(R/W)

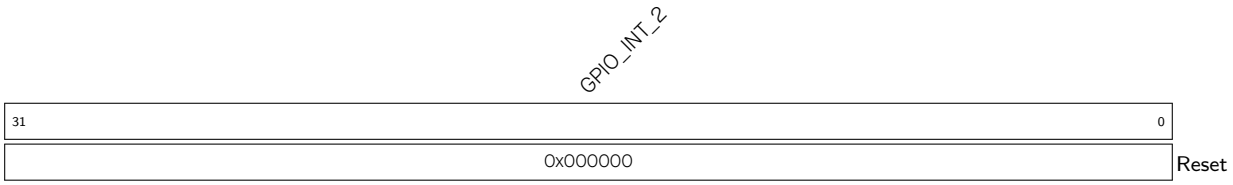
GPIO_FUNC*n*_OE_INV_SEL Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

Register 7.31. GPIO_INTR_2_REG (0x063C)



GPIO_INT_2 Represents the CPU interrupt 2 status of GPIO0 ~ GPIO31. Each bit represents:

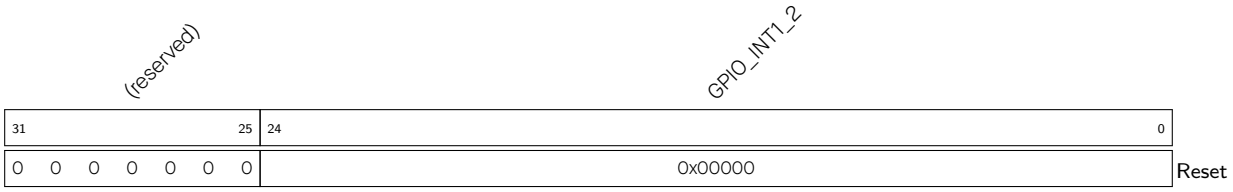
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit16 of GPIO_PIN*n*_REG).

(RO)

Register 7.32. GPIO_INTR1_2_REG (0x0640)



GPIO_INT1_2 Represents the CPU interrupt 2 status of GPIO32 ~ GPIO54. Each bit represents:

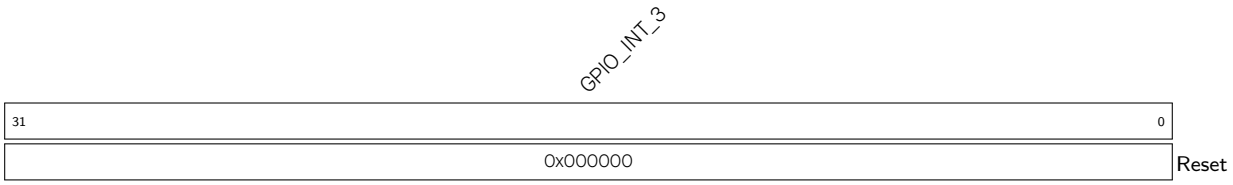
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit16 of GPIO_PIN*n*_REG).

(RO)

Register 7.33. GPIO_INTR_3_REG (0x0644)



GPIO_INT_3 Represents the CPU interrupt 3 status of GPIO0 ~ GPIO31. Each bit represents:

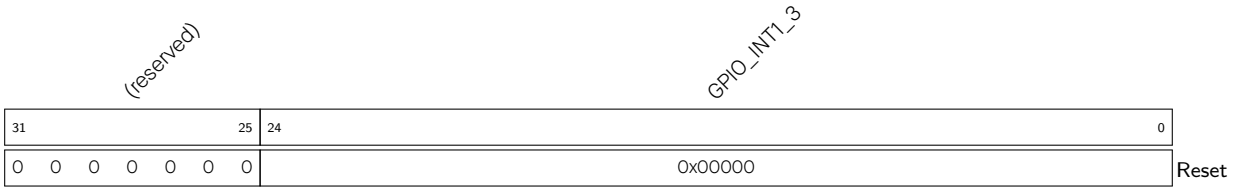
0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit31 are corresponding to GPIO0 ~ GPIO31. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit17 of GPIO_PIN*n*_REG).

(RO)

Register 7.34. GPIO_INTR1_3_REG (0x0648)



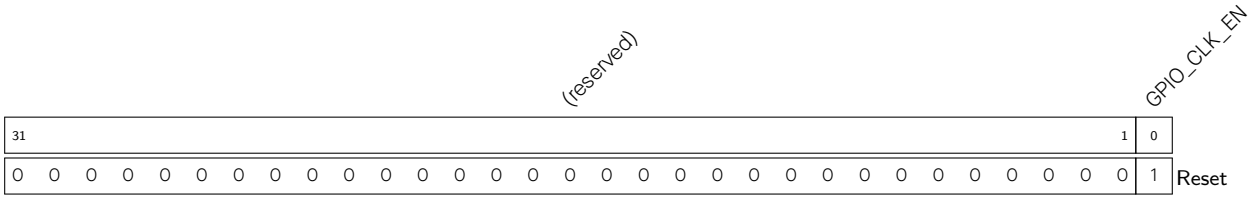
GPIO_INT1_3 Represents the CPU interrupt 3 status of GPIO32 ~ GPIO54. Each bit represents:

0: Represents CPU interrupt is not enabled, or the GPIO does not generate the interrupt configured by GPIO_PIN*n*_INT_TYPE.

1: Represents the GPIO generates an interrupt configured by GPIO_PIN*n*_INT_TYPE after the CPU interrupt is enabled.

Bit0 ~ bit22 are corresponding to GPIO32 ~ GPIO54. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit17 of GPIO_PIN*n*_REG).

Register 7.35. GPIO_CLOCK_GATE_REG (0x064C)



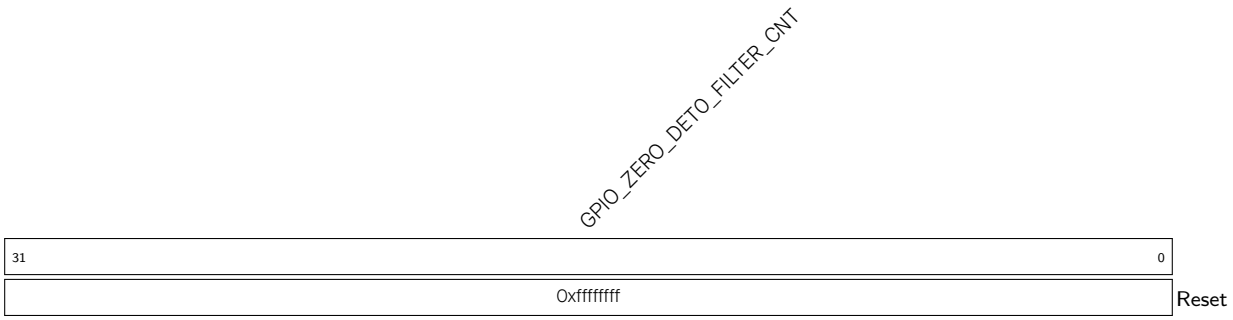
GPIO_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

Register 7.36. GPIO_ZERO_DET0_FILTER_CNT_REG (0x0710)



GPIO_ZERO_DET0_FILTER_CNT Configures to change GPIO analog comparator0 zero detect filter's threshold.

Interrupts in GPIO_ZERO_DET0_FILTER_CNT iomux_core_clk cycles after one interrupt will be fillterred, and not be seen by CPU. (R/W)

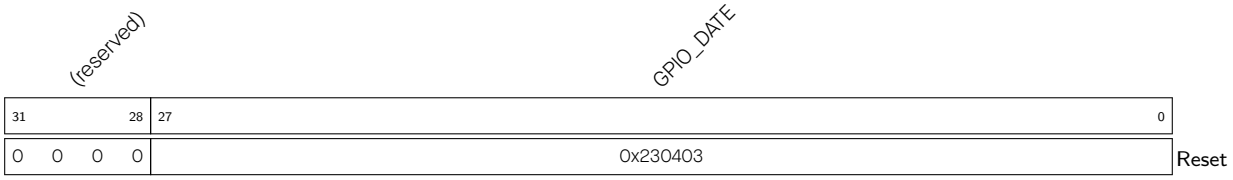
Register 7.37. GPIO_ZERO_DET1_FILTER_CNT_REG (0x0714)



GPIO_ZERO_DET1_FILTER_CNT Configures to change GPIO analog comparator1 zero detect filter's threshold.

Interrupts in GPIO_ZERO_DET0_FILTER_CNT iomux_core_clk cycles after one interrupt will be fillterred, and not be seen by CPU. (R/W)

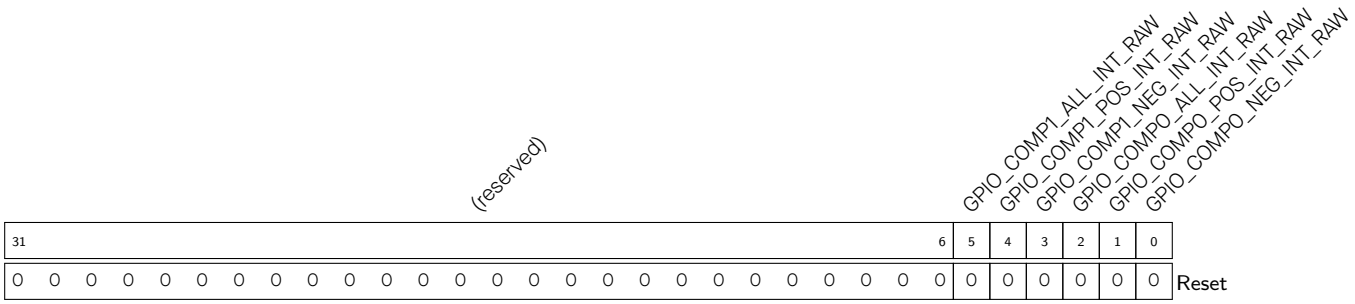
Register 7.38. GPIO_DATE_REG (0x07FC)



GPIO_DATE Version control register.

(R/W)

Register 7.39. GPIO_INT_RAW_REG (0x0700)



GPIO_COMP0_NEG_INT_RAW The raw interrupt status of GPIO_COMP0_NEG_INT. (R/WTC/SS)

GPIO_COMP0_POS_INT_RAW The raw interrupt status of GPIO_COMP0_POS_INT. (R/WTC/SS)

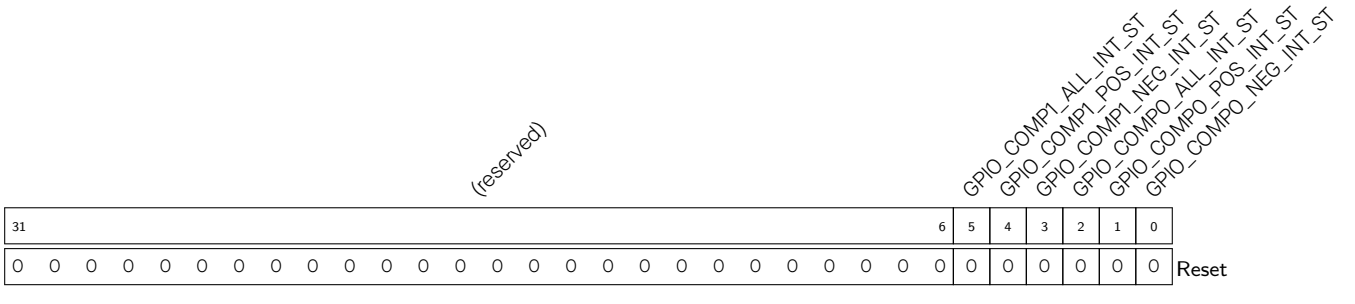
GPIO_COMP0_ALL_INT_RAW The raw interrupt status of GPIO_COMP0_ALL_INT. (R/WTC/SS)

GPIO_COMP1_NEG_INT_RAW The raw interrupt status of GPIO_COMP1_NEG_INT. (R/WTC/SS)

GPIO_COMP1_POS_INT_RAW The raw interrupt status of GPIO_COMP1_POS_INT. (R/WTC/SS)

GPIO_COMP1_ALL_INT_RAW The raw interrupt status of GPIO_COMP1_ALL_INT. (R/WTC/SS)

Register 7.40. GPIO_INT_ST_REG (0x0704)



GPIO_COMP0_NEG_INT_ST The masked interrupt status of GPIO_COMP0_NEG_INT. (RO)

GPIO_COMP0_POS_INT_ST The masked interrupt status of GPIO_COMP0_POS_INT. (RO)

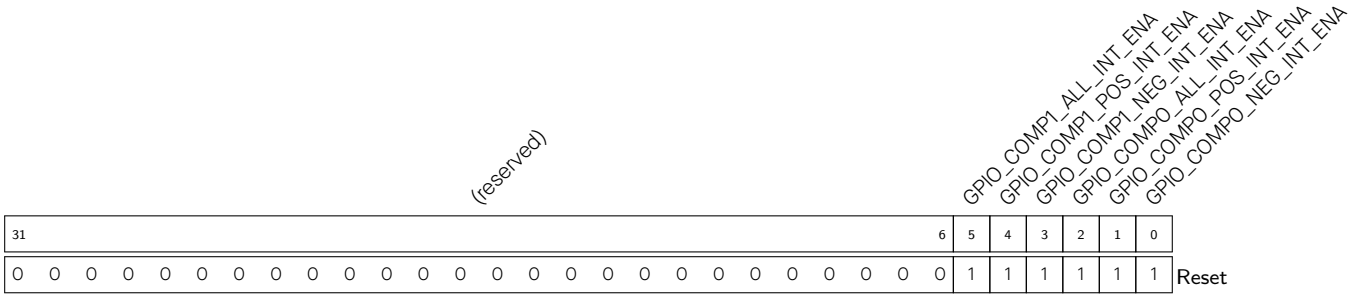
GPIO_COMP0_ALL_INT_ST The masked interrupt status of GPIO_COMP0_ALL_INT. (RO)

GPIO_COMP1_NEG_INT_ST The masked interrupt status of GPIO_COMP1_NEG_INT. (RO)

GPIO_COMP1_POS_INT_ST The masked interrupt status of GPIO_COMP1_POS_INT. (RO)

GPIO_COMP1_ALL_INT_ST The masked interrupt status of GPIO_COMP1_ALL_INT. (RO)

Register 7.41. GPIO_INT_ENA_REG (0x0708)



GPIO_COMP0_NEG_INT_ENA Write 1 to enable GPIO_COMP0_NEG_INT. (R/W)

GPIO_COMP0_POS_INT_ENA Write 1 to enable GPIO_COMP0_POS_INT. (R/W)

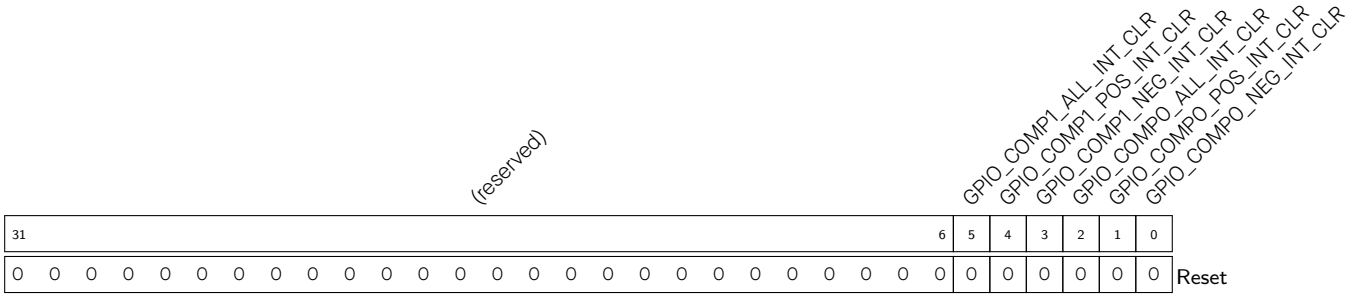
GPIO_COMP0_ALL_INT_ENA Write 1 to enable GPIO_COMP0_ALL_INT. (R/W)

GPIO_COMP1_NEG_INT_ENA Write 1 to enable GPIO_COMP1_NEG_INT. (R/W)

GPIO_COMP1_POS_INT_ENA Write 1 to enable GPIO_COMP1_POS_INT. (R/W)

GPIO_COMP1_ALL_INT_ENA Write 1 to enable GPIO_COMP1_ALL_INT. (R/W)

Register 7.42. GPIO_INT_CLR_REG (0x070C)



GPIO_COMP0_NEG_INT_CLR Write 1 to clear GPIO_COMP0_NEG_INT. (WT)

GPIO_COMP0_POS_INT_CLR Write 1 to clear GPIO_COMP0_POS_INT. (WT)

GPIO_COMP0_ALL_INT_CLR Write 1 to clear GPIO_COMP0_ALL_INT. (WT)

GPIO_COMP1_NEG_INT_CLR Write 1 to clear GPIO_COMP1_NEG_INT. (WT)

GPIO_COMP1_POS_INT_CLR Write 1 to clear GPIO_COMP1_POS_INT. (WT)

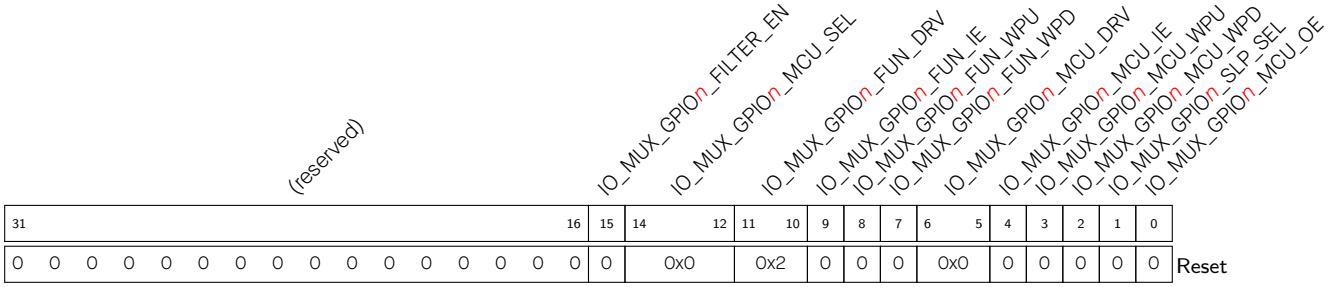
GPIO_COMP1_ALL_INT_CLR Write 1 to clear GPIO_COMP1_ALL_INT. (WT)

7.20.2 HP IO MUX Registers

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

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

Register 7.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)



IO_MUX_GPIO*n*_MCU_OE Configures whether or not to enable the output of GPIO*n* in sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_SLP_SEL Configures whether or not to enter sleep mode for GPIO*n*.

0: Not enter

1: Enter

(R/W)

IO_MUX_GPIO*n*_MCU_WPD Configure whether or not to enable pull-down resistor of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_WPU Configures whether or not to enable pull-up resistor of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_IE Configures whether or not to enable the input of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_MCU_DRV Configures the drive strength of GPIO*n* during sleep mode.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

Continued on the next page...

Register 7.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)

Continued from the previous page...

IO_MUX_GPIO*n*_FUN_WPD Configures whether or not to enable pull-down resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_WPU Configures whether or not enable pull-up resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_IE Configures whether or not to enable input of GPIO*n*.

0: Disable

1: Enable

(R/W)

IO_MUX_GPIO*n*_FUN_DRV Configures the drive strength of GPIO*n*.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_MCU_SEL Configures to select IO MUX function for this pin.

0: Select Function 0

1: Select Function 1

.....

(R/W)

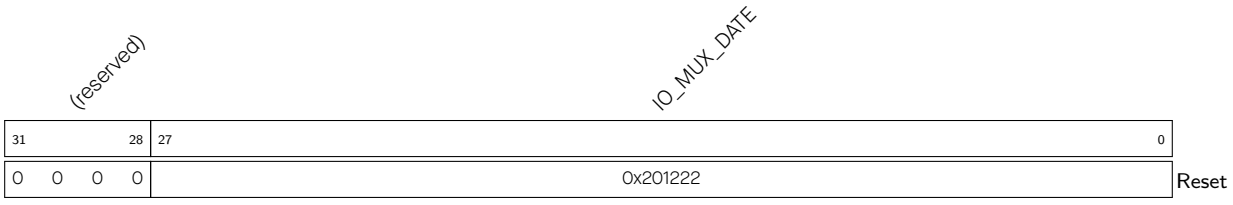
IO_MUX_GPIO*n*_FILTER_EN Configures whether or not to enable filter for pin input signals.

0: Disable

1: Enable

(R/W)

Register 7.44. IO_MUX_DATE_REG (0x0104)



IO_MUX_DATE Version control register.

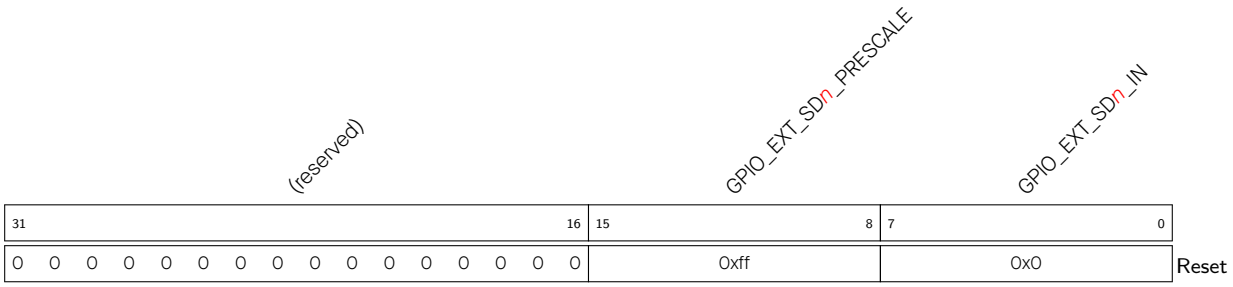
(R/W)

7.20.3 GPIO EXT Registers

The addresses in this section are relative to (HP GPIO matrix base address + 0x0F00). GPIO base address is provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 7.45. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0 - 7) (0x0000+0x4n*)**



GPIO_EXT_SD*n*_IN (*n*: 0 - 7) Configures the duty cycle of sigma delta modulation output. (R/W)

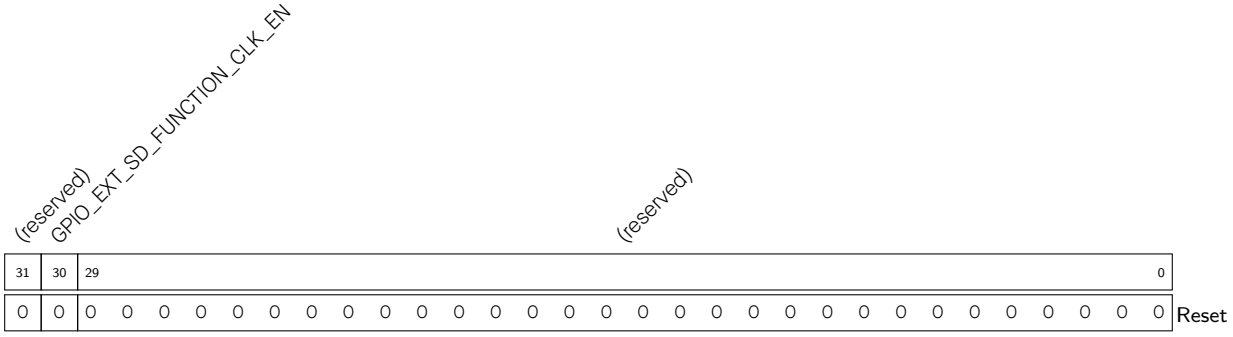
GPIO_EXT_SD*n*_PRESCALE (*n*: 0 - 7) Configures the divider value to divide HP IO MUX operating clock. Can be 0 ~ 255.

0: Do not divide

1 ~ 255: The clock is divided by 2 ~ 256

(R/W)

Register 7.46. GPIO_EXT_SIGMADELTA_MISC_REG (0x0024)



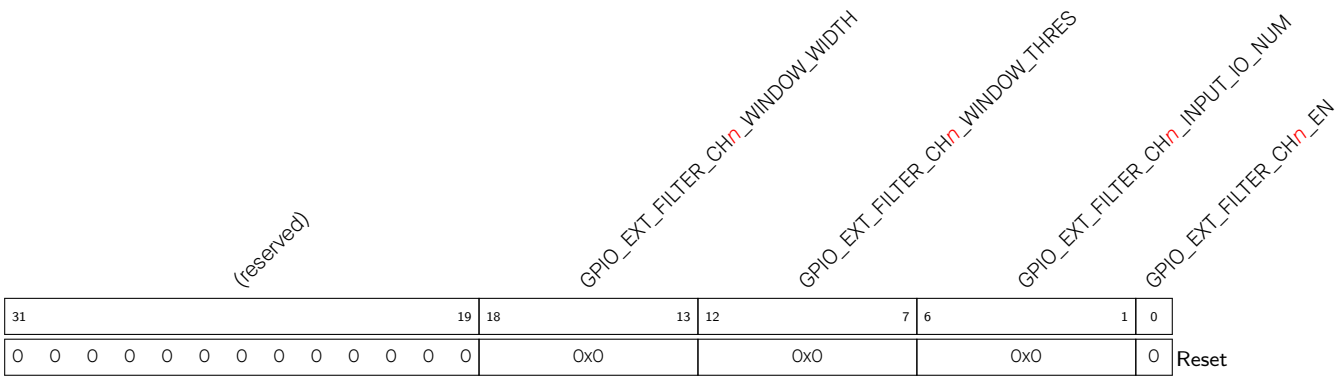
GPIO_EXT_SD_FUNCTION_CLK_EN Configures whether or not to enable the clock for sigma delta modulation.

0: Not enable

1: Enable

(R/W)

Register 7.47. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0 - 7) (0x0030+0x4**n*)



GPIO_EXT_FILTER_CH*n*_EN Configures whether or not to enable channel *n* of Glitch Filter.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM Configures to select the input GPIO for Glitch Filter.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES Configures the window threshold for Glitch Filter. The window threshold should be less than or equal to GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH.

Measurement unit: HP IO MUX operating clock cycle

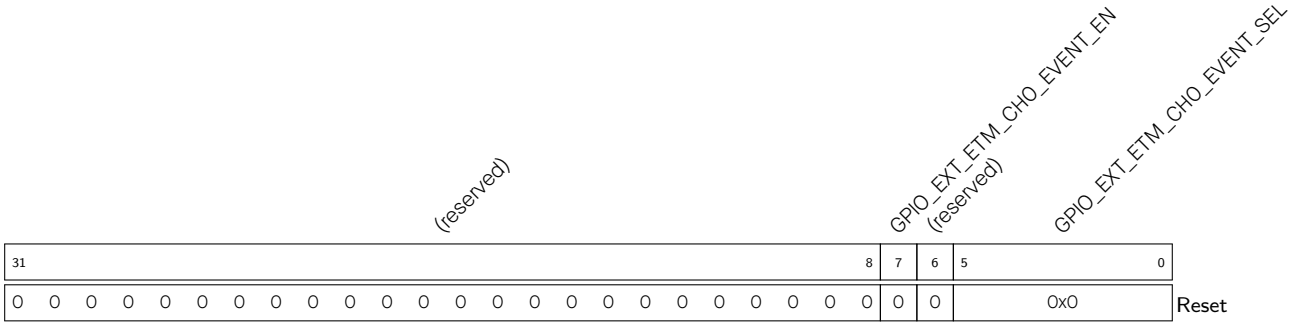
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH Configures the window width for Glitch Filter. The effective value of window width is 0 ~ 62. 63 is a reserved value and can not be used.

Measurement unit: HP IO MUX operating clock cycle

(R/W)

Register 7.48. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0 - 7) (0x0060+0x4**n*)



GPIO_EXT_ETM_CH*n*_EVENT_SEL Configures to select GPIO for ETM event channel.

0: Select GPIO0

1: Select GPIO1

.....

53: Select GPIO53

54: Select GPIO54

(R/W)

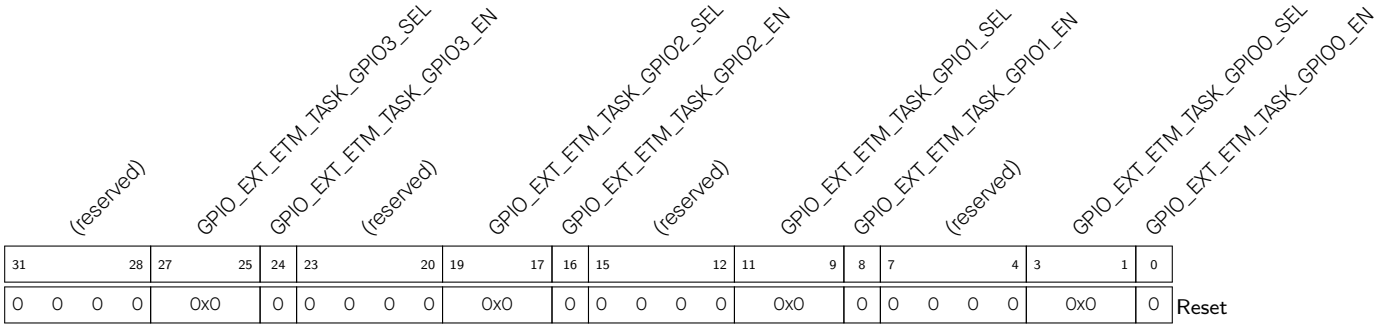
GPIO_EXT_ETM_CH*n*_EVENT_EN Configures whether or not to enable ETM event send.

0: Not enable

1: Enable

(R/W)

Register 7.49. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x00A0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 0 - 3) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 0 - 3) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

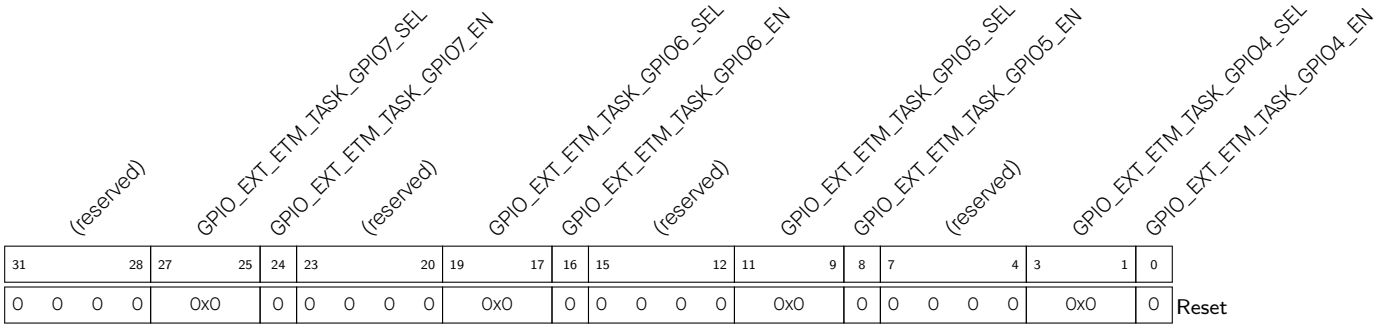
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.50. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x00A4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 4 - 7) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 4 - 7) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.51. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x00A8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 8 - 11) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 8 - 11) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

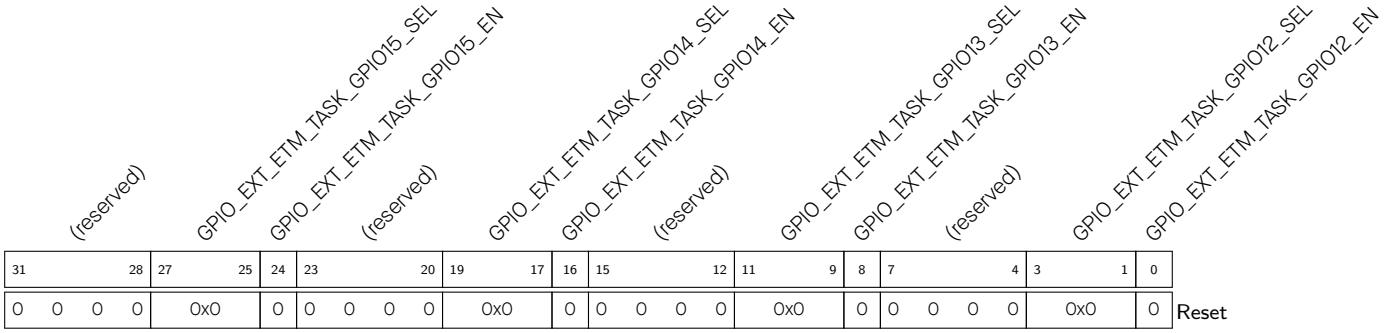
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.52. GPIO_EXT_ETM_TASK_P3_CFG_REG (0x00AC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 12 - 15) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 12 - 15) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

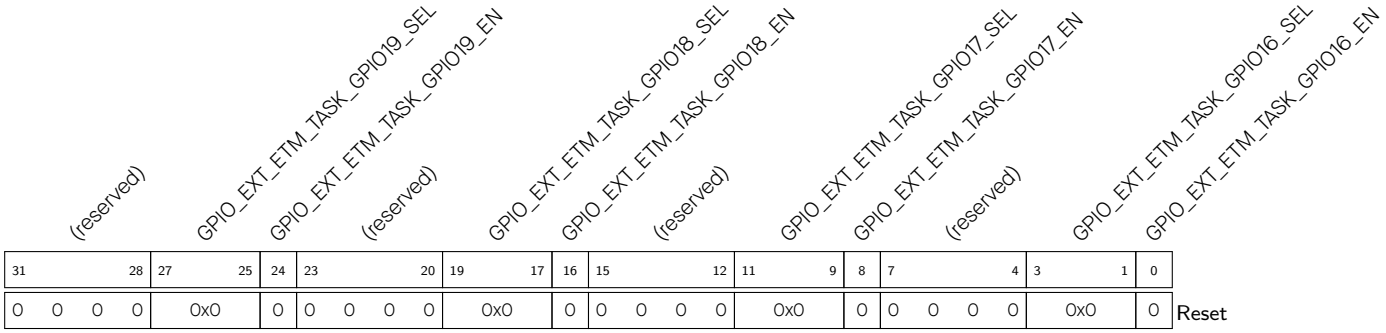
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.53. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x00B0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 16 - 19) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 16 - 19) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

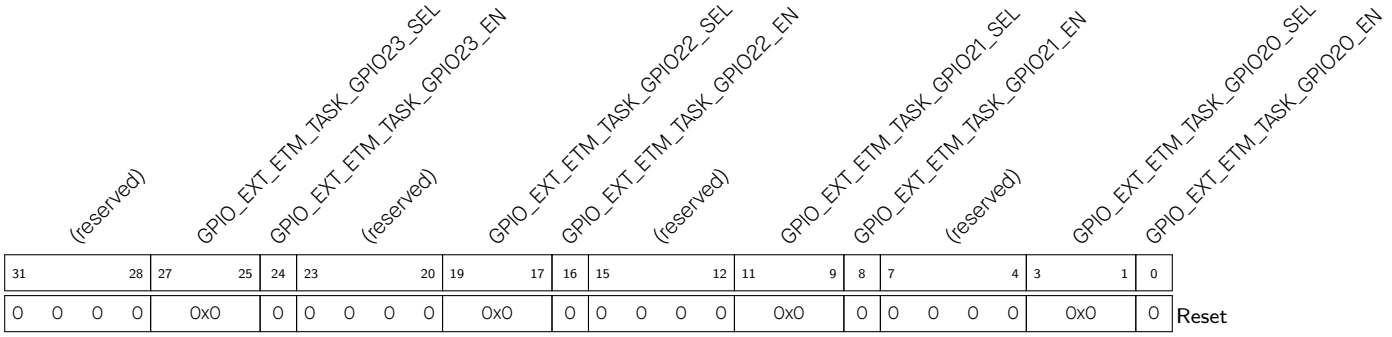
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.54. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x00B4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 20 - 23) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 20 - 23) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

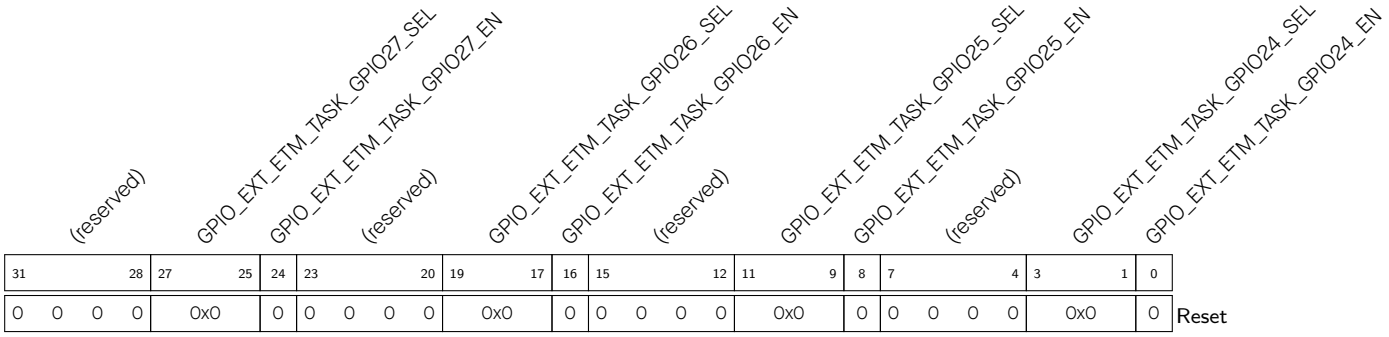
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.55. GPIO_EXT_ETM_TASK_P6_CFG_REG (0x00B8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 24 - 27) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 24 - 27) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

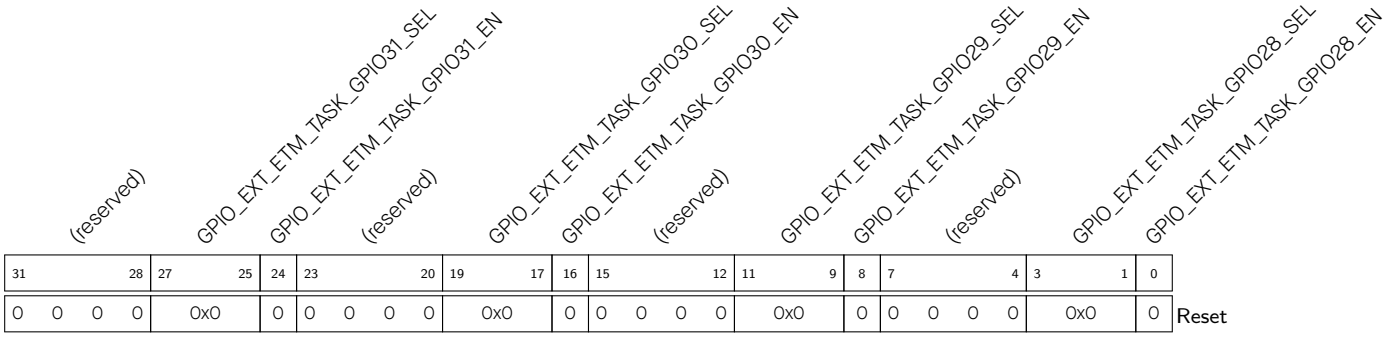
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.56. GPIO_EXT_ETM_TASK_P7_CFG_REG (0x00BC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 28 - 31) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 28 - 31) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

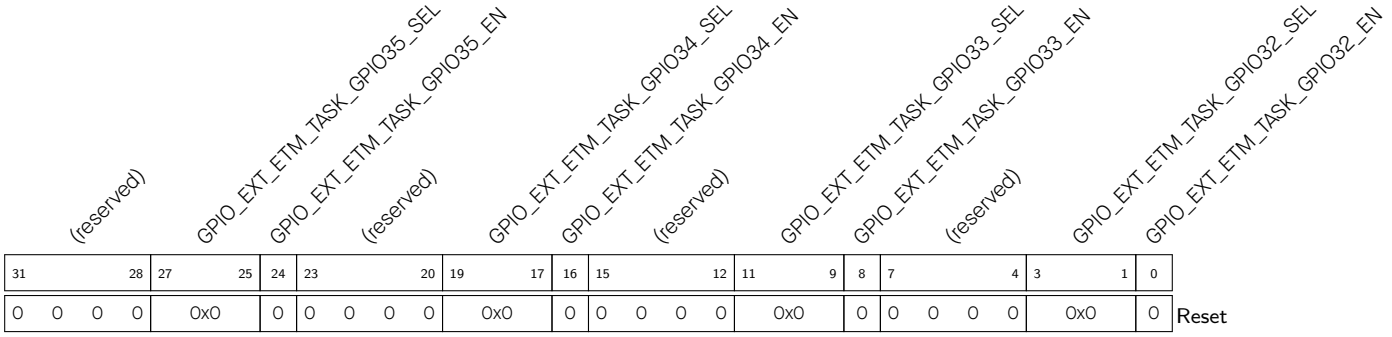
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.57. GPIO_EXT_ETM_TASK_P8_CFG_REG (0x00C0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 32 - 35) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 32 - 35) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

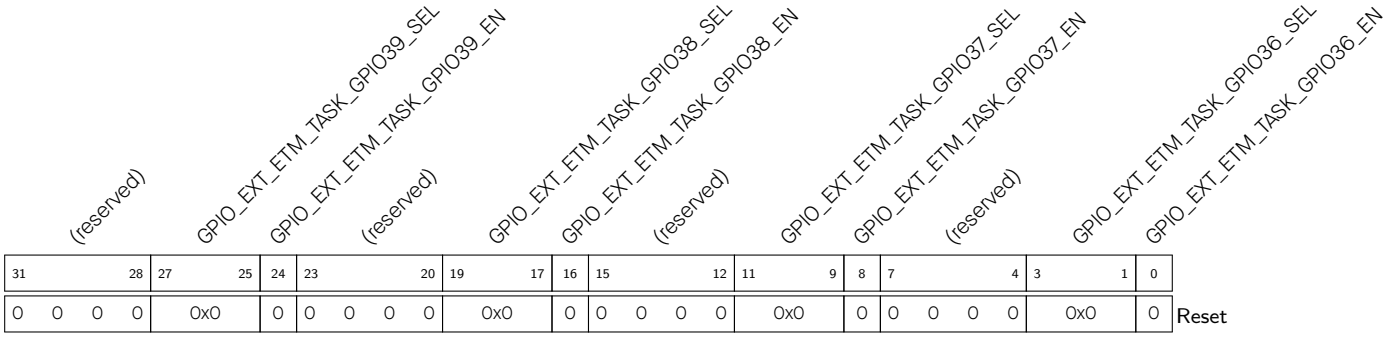
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.58. GPIO_EXT_ETM_TASK_P9_CFG_REG (0x00C4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 36 - 39) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 36 - 39) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

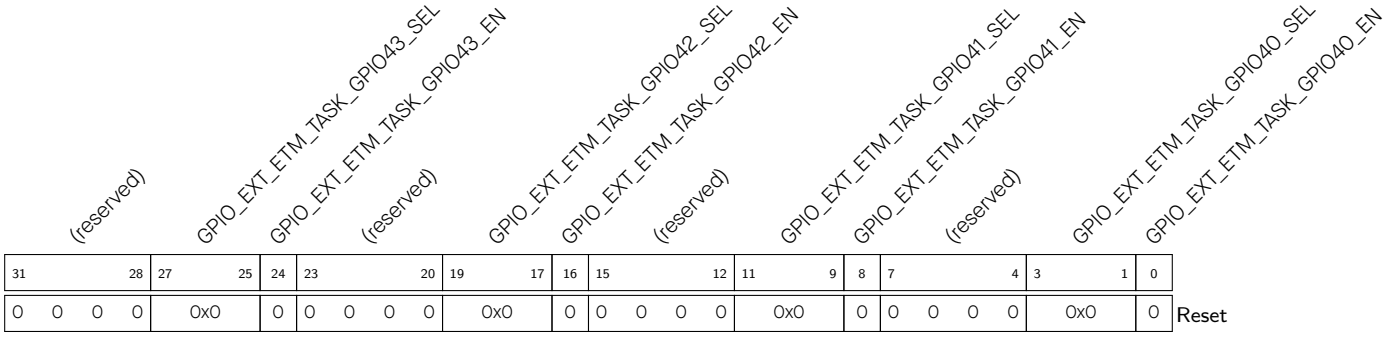
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.59. GPIO_EXT_ETM_TASK_P10_CFG_REG (0x00C8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 40 - 43) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 40 - 43) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.60. GPIO_EXT_ETM_TASK_P11_CFG_REG (0x00CC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 44 - 47) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 44 - 47) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

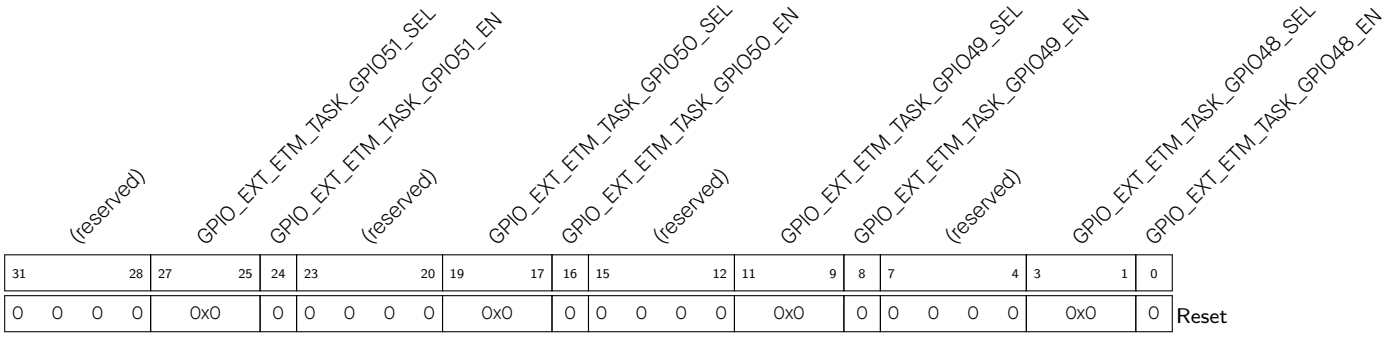
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.61. GPIO_EXT_ETM_TASK_P12_CFG_REG (0x00D0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 48 - 51) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 48 - 51) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.62. GPIO_EXT_ETM_TASK_P13_CFG_REG (0x00D4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 52 - 54) Configures whether or not to enable GPIO*n* to response ETM task.

0: Not enable

1: Enable

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 52 - 54) Configures to select an ETM task channel for GPIO*n*.

0: Select channel 0

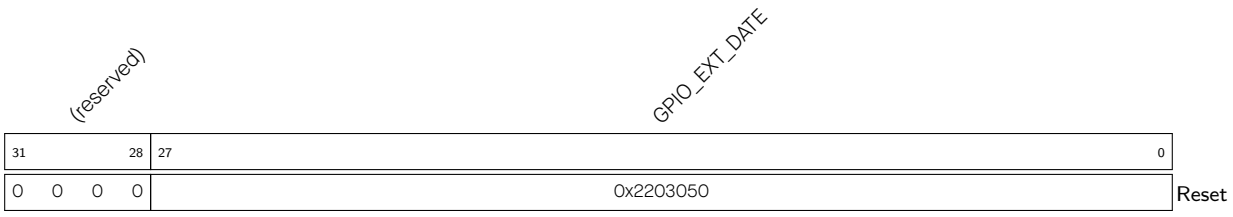
1: Select channel 1

.....

7: Select channel 7

(R/W)

Register 7.63. GPIO_EXT_VERSION_REG (0x00FC)



GPIO_EXT_DATE Version control register.

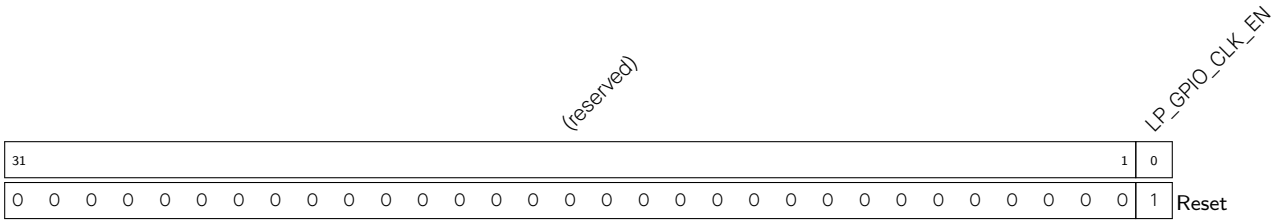
(R/W)

7.20.4 LP GPIO Matrix Registers

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

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

Register 7.64. LP_GPIO_CLK_EN_REG (0x0000)



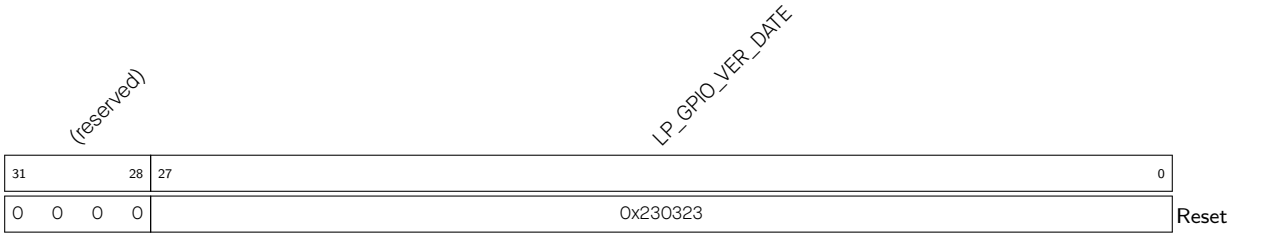
LP_GPIO_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

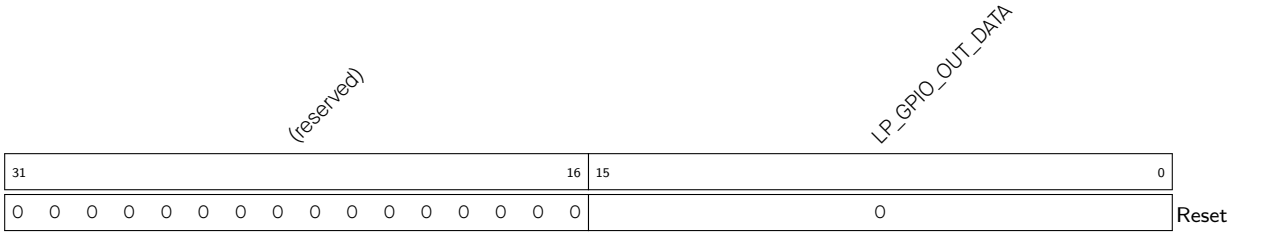
Register 7.65. LP_GPIO_VER_DATE_REG (0x0004)



LP_GPIO_VER_DATE Version control register.

(R/W)

Register 7.66. LP_GPIO_OUT_REG (0x0008)



LP_GPIO_OUT_DATA Configures the output value of GPIO0 ~ GPIO15 output in simple GPIO output mode.

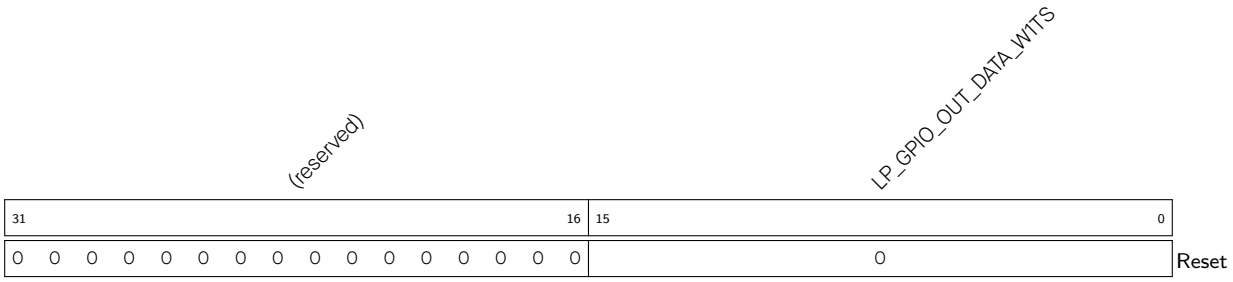
0: Low level

1: High level

The value of bit0 ~ bit15 correspond to the output value of GPIO0 ~ GPIO15 respectively. Bit16 ~ bit31 are invalid.

(R/W/WTC)

Register 7.67. LP_GPIO_OUT_W1TS_REG (0x000C)



LP_GPIO_OUT_DATA_W1TS Configures whether or not to set the output register LP_GPIO_OUT_REG of GPIO0 ~ GPIO15.

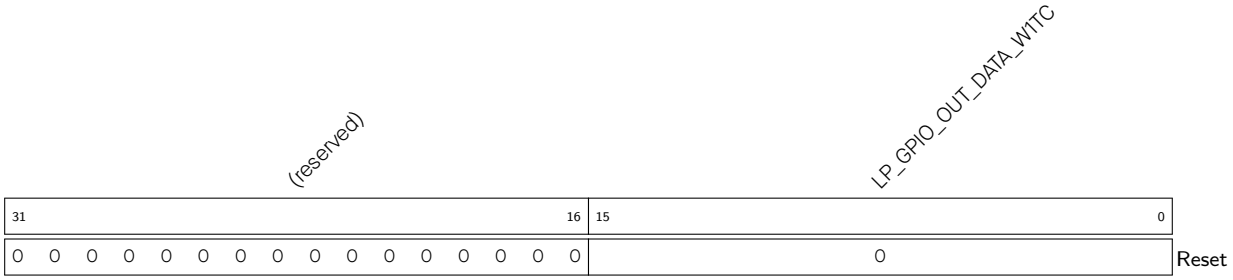
0: Not set

1: The corresponding bit in LP_GPIO_OUT_REG will be set to 1

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to set LP_GPIO_OUT_REG.

(WT)

Register 7.68. LP_GPIO_OUT_W1TC_REG (0x0010)



LP_GPIO_OUT_DATA_W1TC Configures whether or not to clear the output register LP_GPIO_OUT_REG of GPIO0 ~ GPIO15 output.

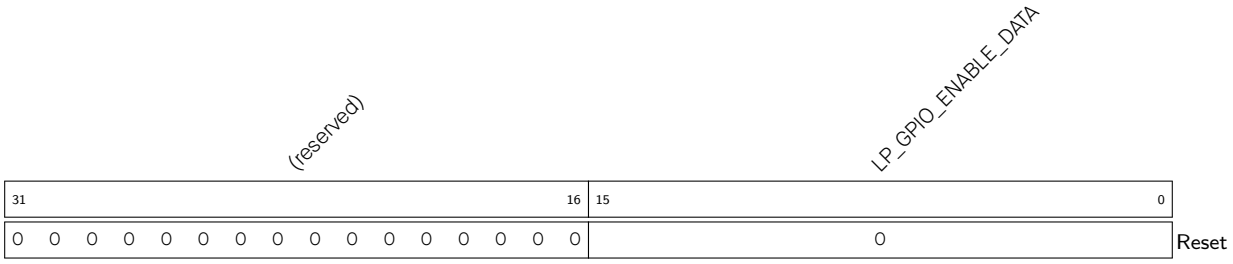
0: Not clear

1: The corresponding bit in LP_GPIO_OUT_REG will be cleared.

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to clear LP_GPIO_OUT_REG.

(WT)

Register 7.69. LP_GPIO_ENABLE_REG (0x0014)



LP_GPIO_ENABLE_DATA Configures whether or not to enable the output of GPIO0 ~ GPIO15.

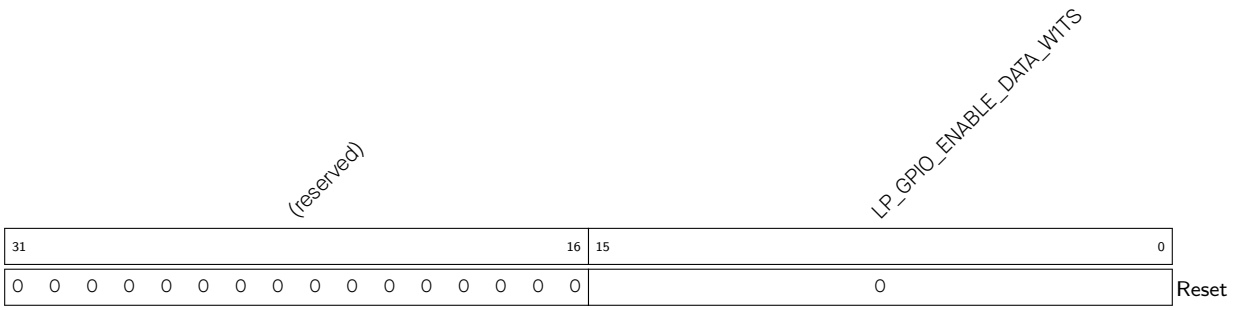
0: Not enable

1: Enable

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

(R/W/WTC)

Register 7.70. LP_GPIO_ENABLE_W1TS_REG (0x0018)



LP_GPIO_ENABLE_DATA_W1TS Configures whether or not to set the output enable register LP_GPIO_ENABLE_REG of GPIO0 ~ GPIO15.

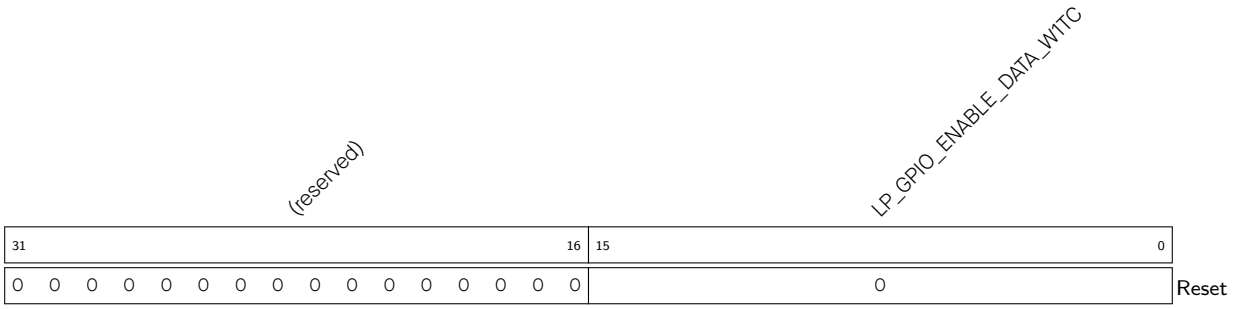
0: Not set

1: The corresponding bit in LP_GPIO_ENABLE_REG will be set to 1

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to set LP_GPIO_ENABLE_REG.

(WT)

Register 7.71. LP_GPIO_ENABLE_W1TC_REG (0x001C)



LP_GPIO_ENABLE_DATA_W1TC Configures whether or not to clear the output enable register LP_GPIO_ENABLE_REG of GPIO0 ~ GPIO15.

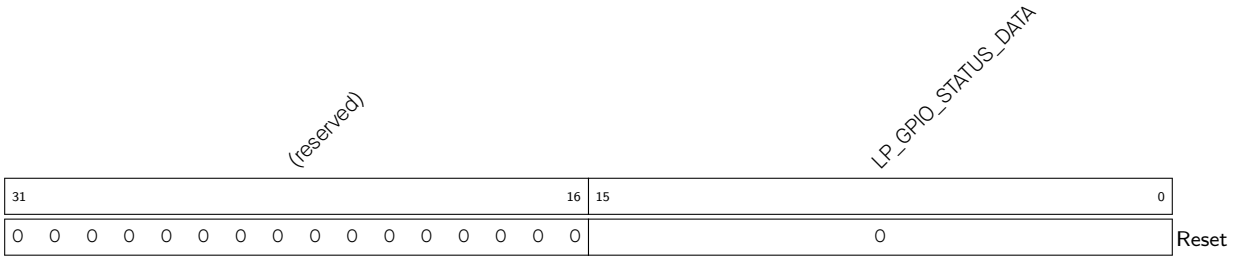
0: Not clear

1: The corresponding bit in LP_GPIO_ENABLE_REG will be cleared

Bit0 ~ bit15 are corresponding to GPIO0 ~ 15. Bit16 ~ bit31 are invalid. Recommended operation: use this register to clear LP_GPIO_ENABLE_REG.

(WT)

Register 7.72. LP_GPIO_STATUS_REG (0x0020)



LP_GPIO_STATUS_DATA The interrupt status of GPIO0 ~ GPIO15, can be configured by the software.

Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

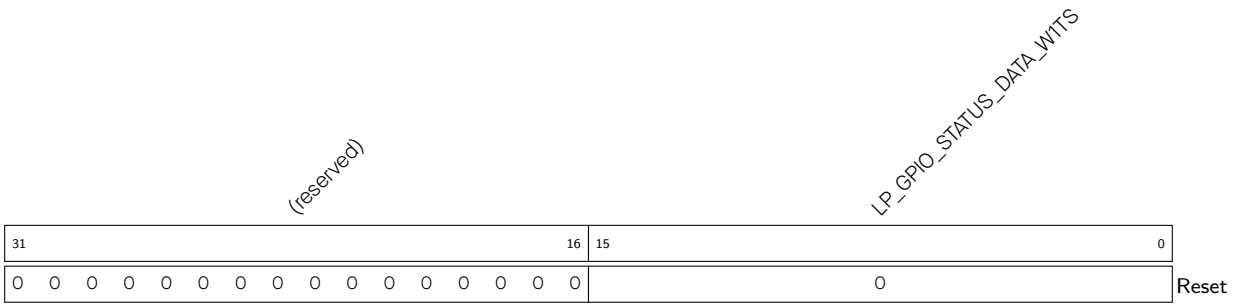
Each bit represents the status of its corresponding GPIO:

0: Represents the GPIO does not generate the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE, or this bit is configured to 0 by the software.

1: Represents the GPIO generates the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE, or this bit is configured to 1 by the software.

(R/W/WTC)

Register 7.73. LP_GPIO_STATUS_W1TS_REG (0x0024)



LP_GPIO_STATUS_DATA_W1TS Configures whether or not to set the interrupt status register LP_GPIO_STATUS_DATA of GPIO0 ~ GPIO15.

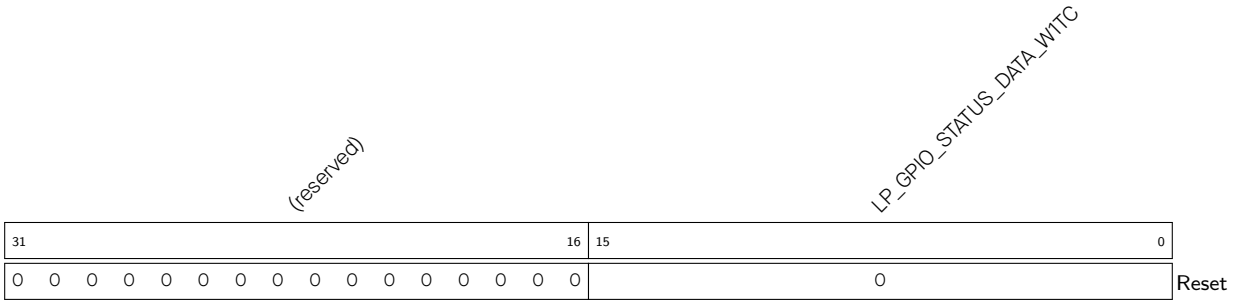
Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

If the value 1 is written to a bit here, the corresponding bit in LP_GPIO_STATUS_DATA will be set to 1.

Recommended operation: use this register to set LP_GPIO_STATUS_DATA.

(WT)

Register 7.74. LP_GPIO_STATUS_W1TC_REG (0x0028)



LP_GPIO_STATUS_DATA_W1TC Configures whether or not to clear the interrupt status register LP_GPIO_STATUS_DATA of GPIO0 ~ GPIO15.

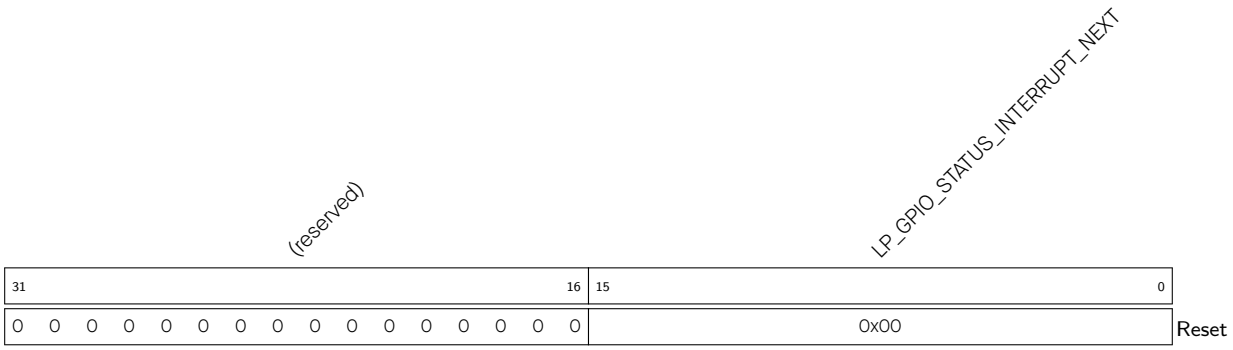
Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.

If the value 1 is written to a bit here, the corresponding bit in LP_GPIO_STATUS_DATA will be cleared.

Recommended operation: use this register to clear LP_GPIO_STATUS_DATA.

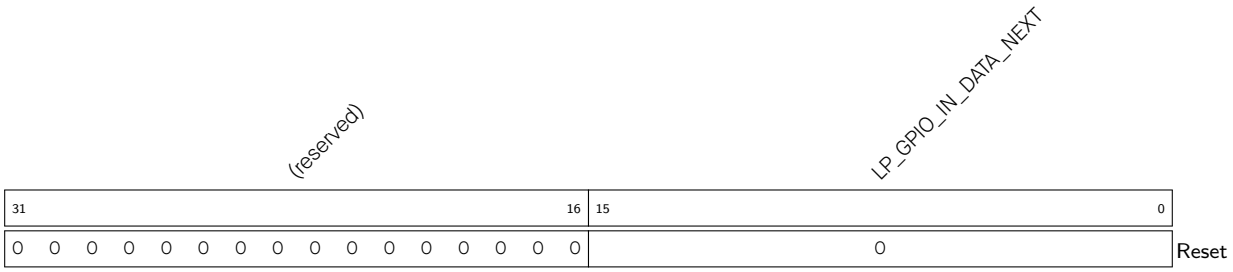
(WT)

Register 7.75. LP_GPIO_STATUS_NEXT_REG (0x002C)



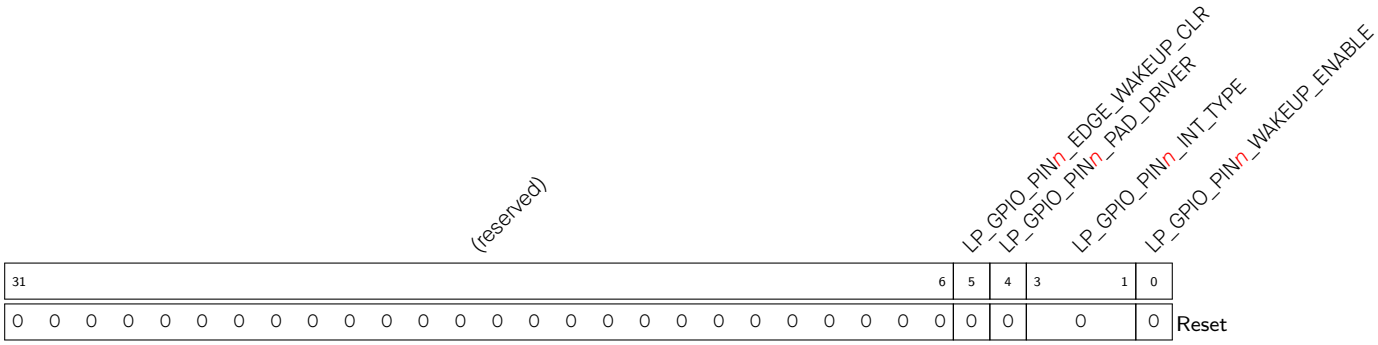
LP_GPIO_STATUS_INTERRUPT_NEXT Represents the interrupt source signal of GPIO0 ~ GPIO15. Bit0 ~ bit15 are corresponding to GPIO0 ~ 15. Bit16 ~ bit31 are invalid. Each bit represents:
0: The GPIO does not generate the interrupt configured by LP_GPIO_PIN*n*_INT_TYPE.
1: The GPIO generates an interrupt configured by LP_GPIO_PIN*n*_INT_TYPE.
The interrupt could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt.
(RO)

Register 7.76. LP_GPIO_IN_REG (0x0030)



LP_GPIO_IN_DATA_NEXT Represents the input value of GPIO0 ~ GPIO15. Each bit represents a pin input value:
0: Low level
1: High level
Bit0 ~ bit15 are corresponding to GPIO0 ~ GPIO15. Bit16 ~ bit31 are invalid.
(RO)

Register 7.77. LP_GPIO_PIN*n*_REG (*n*: 0 - 15) (0x0034+4**n*)



LP_GPIO_PIN*n*_WAKEUP_ENABLE Configures whether or not to enable GPIO wake-up function.

0: Disable

1: Enable

This function only wakes up system from Deep-sleep.

(R/W)

LP_GPIO_PIN*n*_INT_TYPE Configures GPIO interrupt type.

0: GPIO interrupt disabled

1: Rising edge trigger

2: Falling edge trigger

3: Any edge trigger

4: Low level trigger

5: High level trigger

(R/W)

LP_GPIO_PIN*n*_PAD_DRIVER Configures to select pin drive mode.

0: Normal output

1: Open drain output

(R/W)

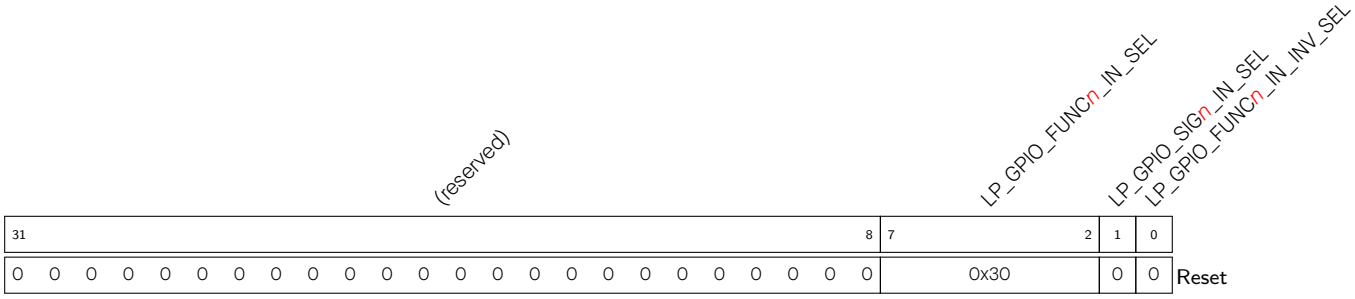
LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR Configures to clear edge-triggered GPIO wake-up source in PMU.

1: The corresponding bit in LP_PMU_xxx will be cleared.

0: No effect.

(WT)

Register 7.78. LP_GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 15) (0x0074+4n*)**



LP_GPIO_FUNC*n*_IN_INV_SEL Configures whether or not to invert the input value.

0: Not invert

1: Invert

(R/W)

LP_GPIO_SIG*n*_IN_SEL Configures whether or not to route signals via LP GPIO matrix.

0: Bypass LP GPIO matrix, i.e., connect signals directly to peripheral configured in LP IO MUX.

1: Route signals via LP GPIO matrix.

(R/W)

LP_GPIO_FUNC*n*_IN_SEL Configures to select a pin from the 16 LP GPIO pins to connect the input signal *n*.

0: Select GPIO0

1: Select GPIO1

.....

14: Select GPIO14

15: Select GPIO15

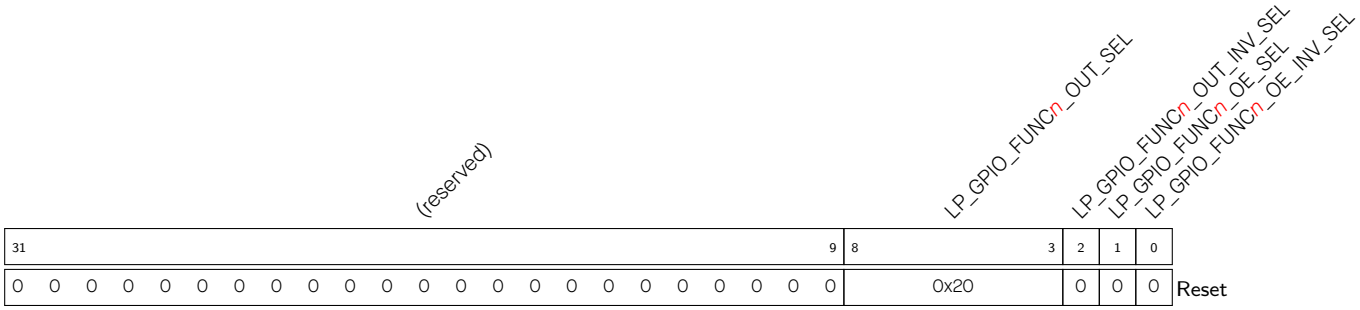
16 ~ 31: Invalid

32 ~ 47: A constantly low input

48 ~ 63: A constantly high input

(R/W)

Register 7.79. LP_GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 15) (0x00F4+4**n*)



LP_GPIO_FUNC*n*_OE_INV_SEL Configures whether or not to invert the output enable signal.

0: Not invert

1: Invert

(R/W)

LP_GPIO_FUNC*n*_OE_SEL Configures to select the source of output enable signal.

0: Use output enable signal from peripheral.

1: Force the output enable signal to be sourced from bit*n* of LP_GPIO_ENABLE_REG.

(R/W)

LP_GPIO_FUNC*n*_OUT_INV_SEL Configures whether or not to invert the output value.

0: Not invert

1: Invert

(R/W)

LP_GPIO_FUNC*n*_OUT_SEL Configures to select a signal *Y* (0 <= *Y* < 32) from 31 peripheral signals to be output to GPIO*n*.

0: Select signal 0

1: Select signal 1

.....

30: Select signal 30

31: Select signal 31

Or

32: Bit*n* of LP_GPIO_OUT_REG and LP_GPIO_ENABLE_REG are selected as the output value and output enable.

For the detailed signal list, see Table 7.13-1.

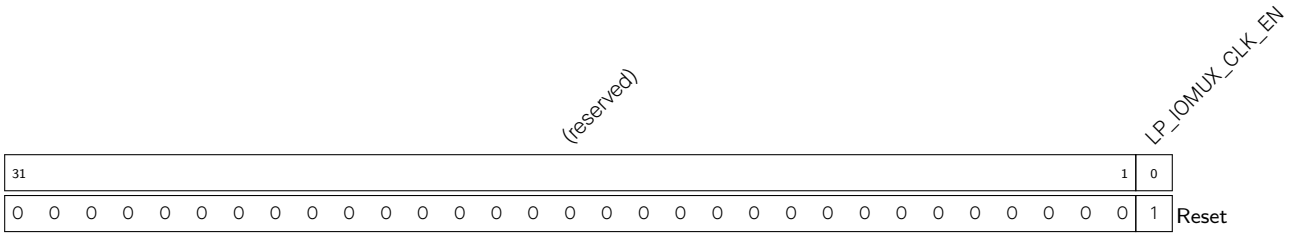
(R/W)

7.20.5 LP IO MUX Registers

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

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

Register 7.80. LP_IOMUX_CLK_EN_REG (0x0000)



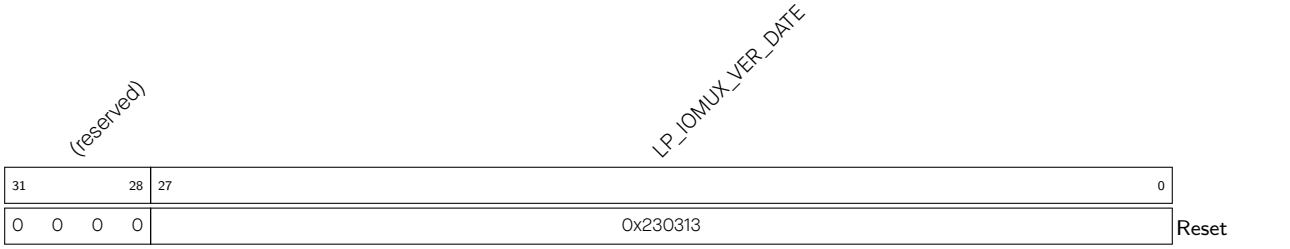
LP_IOMUX_CLK_EN Configures whether or not to enable clock gate.

0: Not enable

1: Enable, the clock is free running.

(R/W)

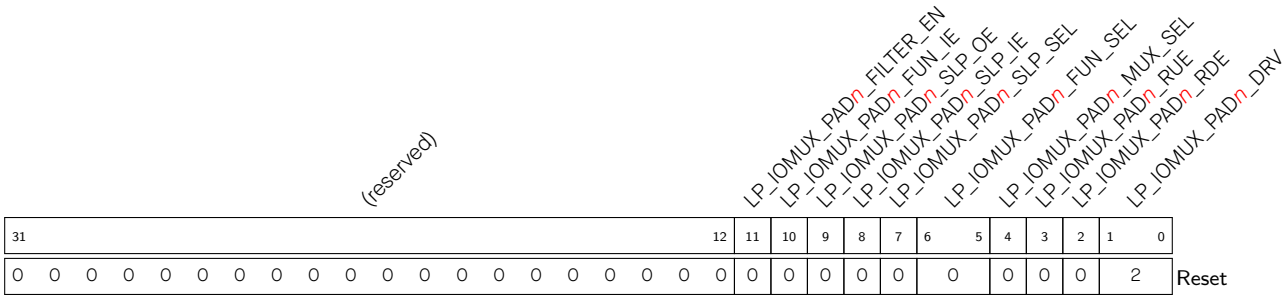
Register 7.81. LP_IOMUX_VER_DATE_REG (0x0004)



LP_IOMUX_VER_DATE Version control register.

(R/W)

Register 7.82. LP_IOMUX_PAD*n*_REG (*n* = 0 - 15) (0x0008+4**n*)



LP_IOMUX_PAD*n*_DRV Configures the drive strength of GPIO*n*.

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

LP_IOMUX_PAD*n*_RDE Configures whether or not enable pull-down resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PAD*n*_RUE Configures whether or not to enable pull-up resistor of GPIO*n*.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PAD*n*_MUX_SEL Configures to decide GPIO*n* is used by HP IO MUX or LP IO MUX.

0: GPIO*n* is used by HP IO MUX

1: GPIO*n* is used by LP IO MUX

(R/W)

LP_IOMUX_PAD*n*_FUN_SEL Configures to select LP IO MUX function for this signal.

0: Select Function 0

1: Select Function 1

.....

(R/W)

LP_IOMUX_PAD*n*_SLP_SEL Configures whether or not to enter sleep mode for GPIO*n*.

0: Not enter

1: Enter

(R/W)

LP_IOMUX_PAD*n*_SLP_IE Configures whether or not to enable the input of GPIO*n* during sleep mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 7.83. LP_IOMUX_PAD*n*_REG (*n* = 0 - 15) (0x0008+4**n*)

Continued from the previous page...

LP_IOMUX_PAD*n*_SLP_OE Configures whether or not to enable the output of GPIO*n* in sleep mode.

0: Disable

1: Enable

(R/W)

LP_IOMUX_PAD*n*_FUN_IE Configures whether or not to enable input of GPIO*n*.

0: Disable

1: Enable

(R/W)

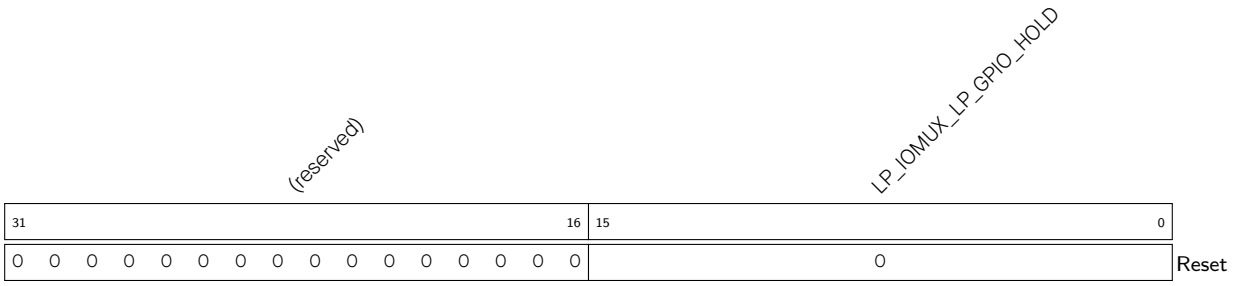
LP_IOMUX_PAD*n*_FILTER_EN Configures whether or not to enable filter for pin input signals.

0: Disable

1: Enable

(R/W)

Register 7.84. LP_IOMUX_LP_PAD_HOLD_REG (0x004C)



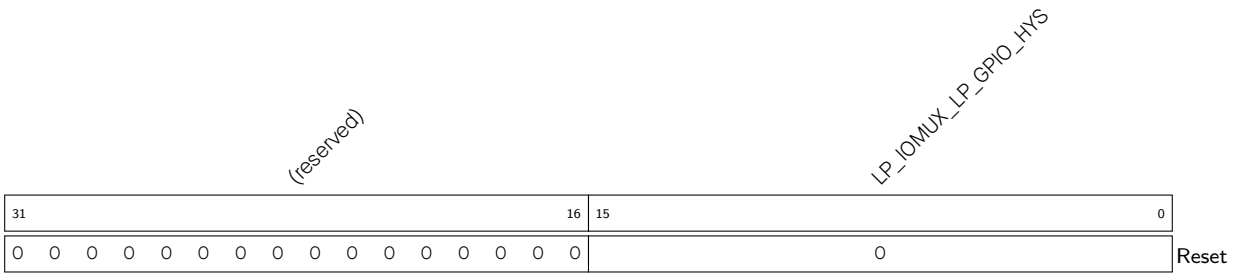
LP_IOMUX_LP_GPIO_HOLD Configures whether or not to enable hold function of GPIO*n*.

0: Disable

1: Enable

(R/W)

Register 7.85. LP_IOMUX_LP_PAD_HYS_REG (0x0050)



LP_IOMUX_LP_GPIO_HYS Configures whether or not to enable hysteresis function of GPIO*n*.

0: Disable

1: Enable

Bit0 bit15 are corresponding to GPIO0 GPIO15.

(R/W)

Chapter 8

Reset and Clock

8.1 Reset

8.1.1 Overview

ESP32-P4 provides four types of reset that occur at different levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset. All reset types mentioned above (except Chip Reset) preserve the data stored in internal memory. Figure 8.1-1 shows the scopes of affected subsystems by each type of reset.

8.1.2 Architectural Overview

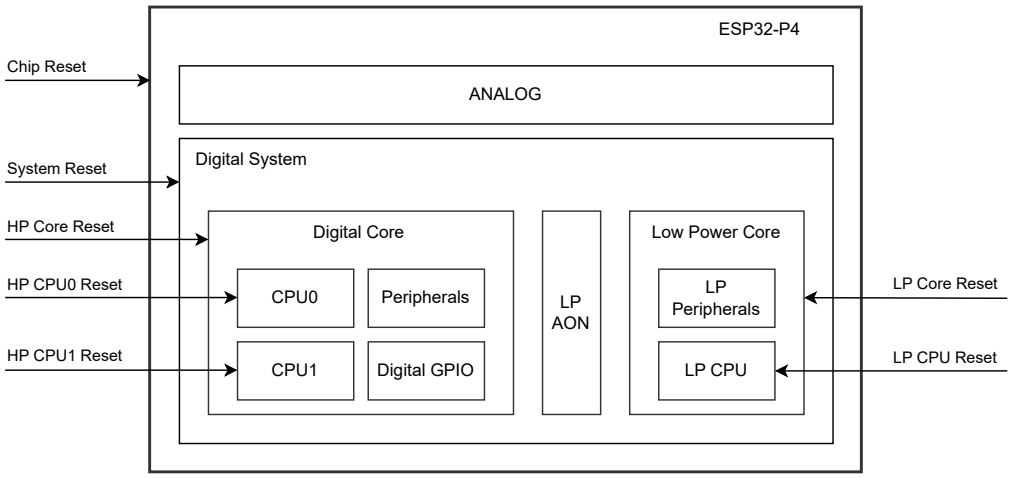


Figure 8.1-1. Reset Types

ESP32-P4's Digital System consists of High Performance System (HP system) that includes Digital Core and HP memory, and Low Power System (LP system) that includes the Low Power Always On (LP AON) system, LP Core, and LP memory. See Figure 8.1-1 for details (note that HP memory and LP memory would not be reset, so they are not shown in the figure).

8.1.3 Features

- Four reset types:
  - CPU Reset: resets CPU core. HP CPU0, HP CPU1, and LP CPU can be reset independently:
    * HP CPU0 will be automatically released from reset after chip power-up, and start execution from CPU Reset Vector (0x30100000).

    * HP CPU1 is at reset by default after chip power-up, and needs to be manually released from reset by writing 0 to HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL. Once released from reset, HP CPU1 will start execution from CPU Reset Vector (0x30100000).
    * LP CPU is at reset after chip power-up, and needs to be manually released from reset by configuring the power management unit (PMU). Once released from reset, LP CPU will start execution from LP_SYSTEM_LP_CPU_BOOT_ADDR (0x50100000 by default). For details, see Chapter 5 *Low-Power Management [to be added later]*.
  - Core Reset: resets the whole digital system except for LP AON. HP core and LP core can be reset independently: HP Core Reset resets HP CPU0, HP CPI1, HP peripherals, HP GPIO, etc., and LP Core Reset resets LP CPU and LP peripherals.
  - System Reset: resets the whole digital system, including the LP system.
  - Chip Reset: resets the whole chip.
- Software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 5 *Low-Power Management [to be added later]*.
  - Hardware Reset: triggered directly by the hardware.

8.1.4 Functional Description

CPU will be reset immediately when any type of reset above occurs. After the reset is released, users can retrieve reset source codes by reading the following fields:

- HP CPU0: LP_CLKRST_HPCORE0_RESET_CAUSE
- HP CPU1: LP_CLKRST_HPCORE1_RESET_CAUSE
- LP CPU: LP_CLKRST_LPCORE_RESET_CAUSE

Table 8.1-1 and Table 8.1-2 list possible reset sources and the types of reset they trigger.

Table 8.1-1. HP CPU Reset Source

Code	Source	Reset Type	Note
0x01	Chip reset[1]	Chip Reset	—
0x03	Software system reset	HP Core Reset	Triggered by configuring LP_SYSTEM_SYS_SW_RST
0x05	PMU core reset	HP Core Reset	See Chapter 5 *Low-Power Management [to be added later]*
0x07	MWDT core reset[2]	HP Core Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x09	RWDT core reset	HP Core Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x0B	MWDT CPU reset	HP CPU0/1 Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x0C	Software CPU reset	HP CPU0/1 Reset	Triggered by configuring LP_CLKRST_HPCORE0_SW_RESET or LP_CLKRST_HPCORE1_SW_RESET
0x0D	RWDT CPU reset	HP CPU0/1 Reset	See Chapter 14 *Watchdog Timers (WDT)*

Cont'd on next page

Table 8.1-1 – cont'd from previous page

Code	Source	Reset Type	Note
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by the brown-out detector. Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on the register configuration. See Chapter 5 *Low-Power Management [to be added later]*
0x10	RWDT system reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x12	Super Watchdog reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x13	Power glitch reset	System Reset	See Chapter 19 *Brown-out Detector*
0x14	eFuse reset	HP Core Reset	Triggered by eFuse CRC error
0x16	USB (JTAG) reset	HP Core Reset	Triggered when external USB host sends a specific command to the JTAG interface of USB Serial/JTAG Controller. See Chapter 36 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
0x17	USB (UART) reset	HP Core Reset	Triggered when external USB host sends a specific command to the serial interface of USB Serial/JTAG Controller. See Chapter 36 *USB Serial/JTAG Controller (USB_SERIAL_JTAG)*
0x18	JTAG CPU reset	HP CPU0/1 Reset	Triggered when a "GDB Resetting CPU" instruction is received
0x1A	Lockup reset	HP CPU0/1 Reset	Triggered when the CPU enters lockup. HP CPU0 and HP CPU1 can be reset in lockup state by configuring LP_CLKRST_HPCOREx_LOCKUP_RESET_EN

[1] Chip Reset can be triggered by the following sources:

- Chip power-up
- Brown-out detector

[2] The HP system contains two watchdog timers called Main System Watchdog Timers (MWDT), and the MWDT that can trigger reset can be configured via HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL and HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL for HP CPU0 and HP CPU1 respectively.

Table 8.1-2. LP CPU Reset Source

Code	Source	Reset Type	Note
0x01	Chip reset	Chip Reset	—
0x09	PMU LP Peripheral reset	LP Core Reset	See Chapter 5 *Low-Power Management [to be added later]*
0x0A	PMU LP CPU reset[1]	LP CPU Reset	See Chapter 5 *Low-Power Management [to be added later]*
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by the brown-out detector
0x10	RWDT system reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x12	Super Watchdog reset	System Reset	See Chapter 14 *Watchdog Timers (WDT)*
0x13	Power glitch reset	System Reset	See Chapter 19 *Brown-out Detector*

Cont'd on next page

Table 8.1-2 – cont'd from previous page

Code	Source	Reset Type	Note
0x14	Software LP CPU reset	LP CPU Reset	Triggered by configuring LPPERI_RST_EN_LP_CORE

[1] LP CPU is reset by PMU after chip power-up. To retrieve the correct reset source code (0x01) when the LP CPU runs for the first time, the source code for PMU LP CPU reset (0x0A) will be masked. After the CPU's first run, clear LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK to unmask the 0x0A source code.

8.1.5 Peripheral Reset

Peripherals can be reset individually by configuring corresponding registers, or globally by Core Reset, System Reset, or Chip Reset.

ESP32-P4 has three groups of peripheral reset registers, prefixed with HP_SYS_CLKRST, LP_CLKRST, and LPPERI. See Section 8.4 *Register Summary* for detailed information.

8.2 Clock

8.2.1 Overview

ESP32-P4 clocks are mainly sourced from oscillator (OSC, including Resistor-Capacitor circuit), crystal (XTAL), and PLL circuit, and then processed by the dividers or selectors, which allows most functional modules to select their working clock according to their power consumption and performance requirements. Figure 8.2-1 shows the HP and LP system clock structure, including the main clock path and typical peripheral clock generator circuit. ESP32-P4 has a large number of peripherals, and limited by space, not all peripheral clocks are shown in the figure.

8.2.2 Architectural Overview

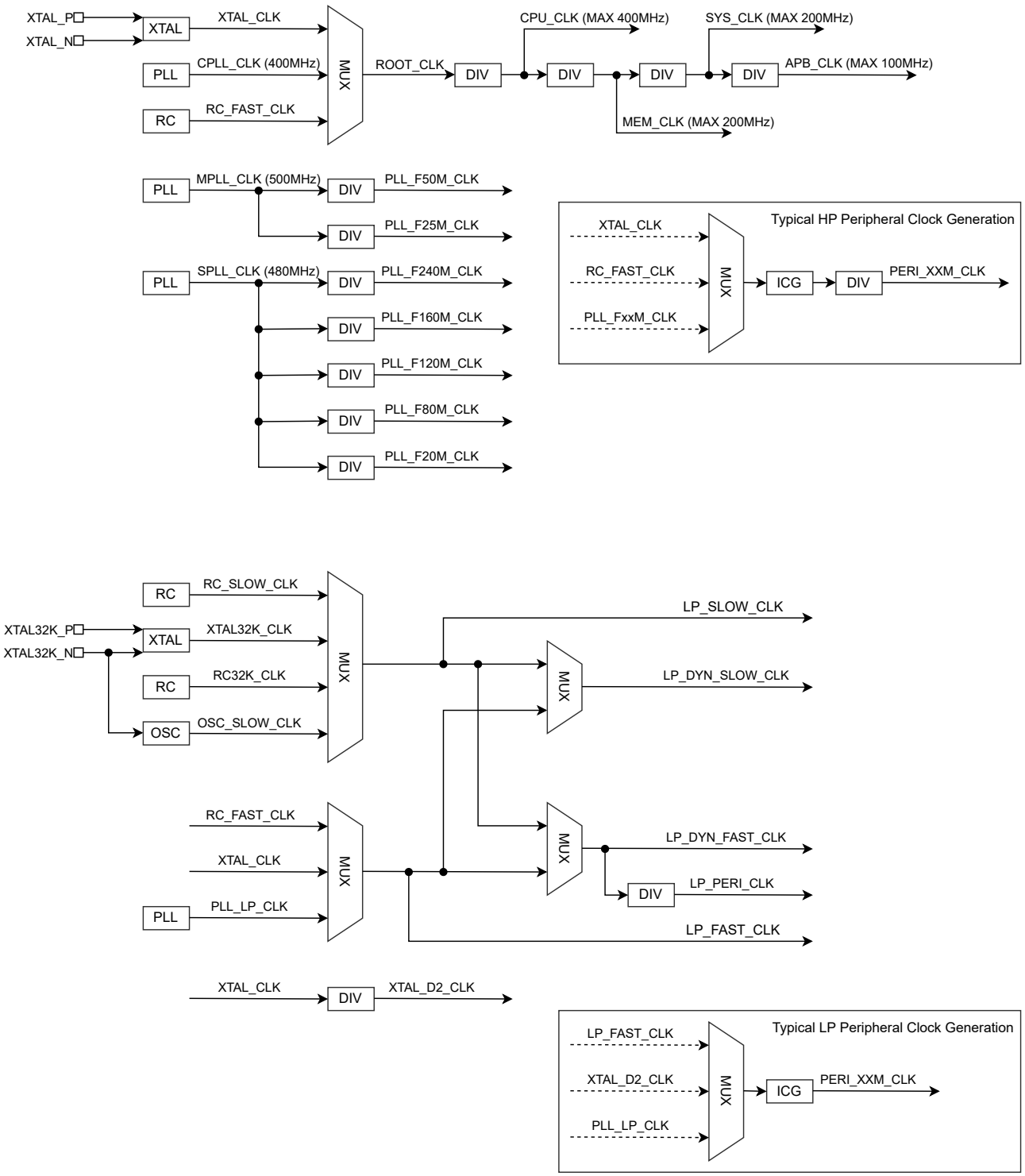


Figure 8.2-1. HP and LP System Clock

8.2.3 Features

ESP32-P4 clocks can be classified into two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as HP CPU0/1 and digital peripherals

  - CPLL_CLK: internal 400 MHz PLL clock. Its reference clock is XTAL_CLK
  - MPLL_CLK: internal 500 MHz PLL clock. Its reference clock is XTAL_CLK
  - SPLL_CLK: internal 480 MHz PLL clock. Its reference clock is XTAL_CLK
- Slow speed clocks for LP system and some peripherals working in low-power mode
  - XTAL32K_CLK: external 32 kHz crystal clock
  - RC_SLOW_CLK: internal slow RC oscillator with adjustable frequency (150 kHz by default)
  - OSC_SLOW_CLK: external slow clock input through XTAL_32K_N, with a frequency of 32 kHz by default. After configuring this GPIO, also configure the Hold function (see Chapter 7 *GPIO Matrix and IO MUX* > 7.9 *Pin Hold Feature*)
  - RC32K_CLK: internal 32 kHz slow RC oscillator
  - XTAL_CLK: 40 MHz external crystal clock
  - RC_FAST_CLK: internal fast RC oscillator with adjustable frequency (20 MHz by default)
  - PLL_LP_CLK: internal PLL clock with a frequency of 8 MHz by default. Its reference clock can be selected from RC32K_CLK or XTAL32K_CLK

8.2.4 Functional Description

8.2.4.1 HP System Clock

As Figure 8.2-1 shows, XTAL_CLK, CPLL_CLK, and RC_FAST_CLK are the multiplexer inputs to generate ROOT_CLK. For selecting which input, see Table 8.2-1.

Table 8.2-1. ROOT_CLK Clock Source

LP_CLKRST_HP_ROOT_CLK_SRC_SEL	Clock Source
0	XTAL_CLK
1	CPLL_CLK
2	RC_FAST_CLK

ROOT_CLK is then divided by a series of dividers, and sequentially derives:

- CPU_CLK, which drives HP CPUs and their logic
- MEM_CLK, which drives the internal memories (L2 Cache, L2MEM, ROM) and their logic
- SYS_CLK, which is the HP high-speed bus clock that drives the AXI and AHB bus logic
- APB_CLK, which is the HP low-speed bus clock that drives the APB bus logic

The above four derived clocks are synchronous SoC clocks, asynchronous to peripheral clocks described in Section 8.2.4.3 *Peripheral Clocks*. Peripheral clocks are asynchronous with each other unless otherwise stated.

The maximum frequencies allowed for these SoC clocks are as follows. Software should ensure that the frequencies of these clocks do not exceed the maximum frequency.

- CPU_CLK: 400 MHz

- MEM_CLK: 200 MHz
- SYS_CLK: 200 MHz
- APB_CLK: 100 MHz

Each module has an individual clock gate for the SoC clock it uses. For example, users can enable the clock for HP CPU0 and disable the clock for HP CPU1 by setting the HP_SYS_CLKRST_CORE0_CPU_CLK_EN bit and clearing the HP_SYS_CLKRST_CORE1_CPU_CLK_EN bit. "CPU_CLK" in bit names indicates that the corresponding clock belongs to the CPU_CLK group shown in Figure 8.2-1.

The internal PLL clocks of ESP32-P4, namely MPLL_CLK and SPLL_CLK, generate reference clocks for peripherals through a set of parallel clock dividers.

Besides these two PLL clocks, there are two additional PLL clocks dedicated to specific modules:

- SDIO_PLL, which generates three clocks:
  - SDIO_PLL0_CLK, used for SDIO's high-speed SDIO_SLF_CLK (internal signal clock) and the source of SDIO's low-speed clock
  - SDIO_PLL1_CLK, users for SDIO's high-speed SDIO_DRV_CLK (output signal driving clock)
  - SDIO_PLL2_CLK, used for SDIO's high-speed SDIO_SAM_CLK (input signal sampling clock)
- APLL_CLK, which is dedicated to I2S

A typical peripheral clock generation circuit consists of a clock selector, a clock gating unit, and a divider. In high-performance mode, the peripheral can select a high-speed clock source without division for higher processing speed. In low-power mode, the peripheral can select a low-speed clock source or divide it by a large divisor to lower power consumption. When the peripheral is disabled, its clock can be directly disabled.

Note:
Updates to HP_SYS_CLKRST_ROOT_CLK_CTRL0/1/2/3_REG will take effect only after HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE is set.

8.2.4.2 LP System Clock

The LP system can operate when most other clocks are disabled. LP system clocks include LP_SLOW_CLK, LP_FAST_CLK, LP_DYN_SLOW_CLK, LP_DYN_FAST_CLK, and XTAL_D2_CLK.

The clock sources for LP_SLOW_CLK and LP_FAST_CLK are low-frequency clocks:

- LP_SLOW_CLK has four clock sources, see Table 8.2-2:

Table 8.2-2. LP_SLOW_CLK Clock Source Selection

LP_CLKRST_SLOW_CLK_SEL	Clock Source
0	RC_SLOW_CLK
1	XTAL32K_CLK
2	RC32K_CLK
3	OSC_SLOW_CLK

- LP_FAST_CLK has three clock sources, see Table 8.2-3:

Table 8.2-3. LP_FAST_CLK Clock Source Selection

LP_CLKRST_FAST_CLK_SEL	Clock Source
0	XTAL_CLK
1	RC_FAST_CLK
2	PLL_LP_CLK

LP_DYN_SLOW_CLK and LP_DYN_FAST_CLK can be derived from LP_SLOW_CLK or LP_FAST_CLK depending on the chip's power mode (see Chapter 5 *Low-Power Management [to be added later]*), and should be able to operate in all power modes to drive LP AON modules. These two clocks are always synchronous with each other.

- In Active mode, select LP_FAST_CLK as the clock source. In this case, LP_DYN_SLOW_CLK has the same frequency with LP_SLOW_CLK, and the same phase with LP_FAST_CLK;
- in Light-sleep and Deep-sleep mode, select LP_SLOW_CLK as the clock source. In this case, LP_DYN_FAST_CLK has the same frequency and phase as LP_SLOW_CLK.

LP_PERI_CLK is derived from LP_DYN_FAST_CLK and drives all the buses for LP peripherals.

XTAL_D2_CLK is derived from XTAL_CLK divided by 2.

8.2.4.3 Peripheral Clocks

Table 8.2-4, Table 8.2-5, Table 8.2-6, and Table 8.2-7 list the derived HP/LP clock sources and HP clocks/LP clocks for each peripheral.

Table 8.2-4. Derived HP Clock sources

Derived Clock	Source Clock							Derived Clock						
	RC_FAST_CLK	XTAL_CLK	CPLL_CLK	MPLL_CLK	SPLL_CLK	APLL_CLK	SDIO_PLL0_CLK	ROOT_CLK	CPU_CLK	MEM_CLK	SYS_CLK	PLL_F80M_CLK	PLL_F160M_CLK	PLL_F240M_CLK
ROOT_CLK	Y	Y	Y											
CPU_CLK								Y						
MEM_CLK									Y					
SYS_CLK										Y				
APB_CLK											Y			
PLL_F50M_CLK				Y										
PLL_F25M_CLK				Y										
PLL_F240M_CLK					Y									
PLL_F160M_CLK					Y									
PLL_F120M_CLK					Y									
PLL_F80M_CLK					Y									
PLL_F20M_CLK					Y									
PLL_F12M_CLK					Y									
PLL_F48M_CLK					Y									
FLASH_CLK		Y	Y		Y									
PSRAM_CLK		Y	Y	Y	Y									
GPSPI_CLK	Y	Y			Y	Y	Y							
ADC_CLK	Y	Y										Y		
UART_CLK	Y	Y										Y		
CRYPTO_CLK	Y	Y											Y	Y

Table 8.2-5. HP Peripheral Clocks

Peripheral	Clock	Clock Source													Derived Clock						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F50M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
SPI0/1 (flash)	FLASH_PLL_CLK														Y						
	FLASH_CORE_CLK														Y						
SPI0/1 (PSRAM)	PSRAM_PLL_CLK															Y					
	PSRAM_CORE_CLK															Y					
I2S Controller (I2S)	I2S_RX_CLK		Y	Y																	PAD_I2S_MCLK
	I2S_TX_CLK		Y	Y																	PAD_I2S_MCLK
I2C Controller (I2C)	I2C_CLK	Y	Y																		
Motor Control PWM (MCPWM)	MCPWM_CLK	Y	Y										Y								
Timer Group (TIMG)	TIMERGRP_T0_CLK	Y	Y								Y										
	TIMERGRP_T1_CLK	Y	Y								Y										
Main System Watchdog Timer (MWDT)	TIMERGRP_WDT_CLK	Y	Y								Y										
Two-Wire Automotive Interface (TWAI)	TAWI_CLK	Y	Y																		
GP-SPI2/GP-SPI3	GPSPI_HS_CLK																Y				
	GPSPI_MST_CLK																Y				

Cont'd on next page

GoBack

GoBack

Table 8.2-5 – cont'd from previous page

Peripheral	Clock	Clock Source													Derived Clock						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F50M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
Ethernet Media Access Controller (EMAC)	EMAC_RMII_CLK																				PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK PAD_EMAC_TX_CLK
	EMAC_RX_CLK																				PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK
	EMAC_TX_CLK																				PAD_EMAC_TXRX_CLK PAD_EMAC_TX_CLK
	EMAC_PTP_REF_CLK		Y								Y										
CAM (CAM OUT)	CAM_CLK		Y	Y									Y								
IO MUX	IOMUX_CLK		Y								Y										
LCD	LCD_CLK		Y	Y									Y								
LED PWM Controller (LEDC)	LEDC_CLK	Y	Y								Y										
ADC Controller (ADC)	HPADC_CLK																	Y			
	HPADC1_SARCLK																	Y			
	HPADC2_SARCLK																	Y			
AES Accelerator (AES) SHA Accelerator (SHA) RSA Accelerator (RSA) ECC Accelerator (ECC) Digital Signature Algorithm (DSA) HMAC Accelerator (HMAC) Elliptic Curve Digital Signature Algorithm (ECDSA) Key Manager [to be added later]	AES_CLK SHA_CLK RSA_CLK ECC_CLK DSA_CLK HMAC_CLK ECDSA_CLK KM_CLK SEC_CLK																			Y	
RMT	RMT_CLK	Y	Y								Y										
System Timer	SYSTIMER_CLK	Y	Y																		
UART Controller (UART)	UART_FCLK																		Y		
	UART_SCLK																		Y		
	UART_SLP_CLK																				PAD_UART_SLP_CLK
Parallel IO Controller (PARLIO)	PARLIO_RX_CLK	Y	Y										Y								PAD_PARLIO_RX_CLK
	PARLIO_TX_CLK	Y	Y										Y								PAD_PARLIO_TX_CLK
I3C Controller [to be added later]	I3C_MST_CLK		Y									Y	Y								
Image Signal Processor [to be added later]	ISP_CLK		Y										Y	Y							
MIPI DSI [to be added later]	DSI_DPHY_CFG_CLK DSI_DPHY_PLL_REFCLK	Y					Y	Y													
	DSI_DPICLK		Y										Y	Y							
MIPI CSI [to be added later]	CSI_DPHY_CFG_CLK	Y					Y	Y													
H264 Encoder [to be added later]	H264_CLK		Y											Y							
USB Serial/JTAG Controller (USB_SERIAL_JTAG)	USB_SERIAL_JTAG_CLK								Y												
USB 2.0 OTG Full-Speed [to be added later]	USB_OTG11_CLK								Y												
USB 2.0 OTG High-Speed [to be added later]	USB_HS_PHY_REFCLK					Y		Y													PAD_USB_HSPHY_REFCLK
SD/MMC Host Controller (SDHOST)	SDIO_DRV_CLK (low-speed) SDIO_SLF_CLK (low-speed) SDIO_SAM_CLK (low-speed)				Y								Y								

Table 8.2-6. Derived LP Clock Sources

Derived clock	Source clock							Derived clock					Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	RC32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	XTAL_D2_CLK	
LP_FAST_CLK					Y	Y	Y						
LP_SLOW_CLK	Y	Y	Y	Y									
LP_DYN_FAST_CLK								Y	Y				
LP_DYN_SLOW_CLK								Y	Y				
LP_PERI_CLK											Y		
XTAL_D2_CLK						Y							

Table 8.2-7. LP Peripheral Clocks

Derived clock	Source clock							Derived clock						Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	RC32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	LP_PERI_CLK	XTAL_D2_CLK	
eFuse Controller													Y	
RTC Watchdog Timer (RWDT)	Y									Y	Y			
RTC Timer										Y	Y			
Brown-out Detector											Y			
Power Management Unit (PMU)	Y								Y	Y	Y			
LP UART							Y				Y		Y	PAD_LP_UART_SLP_CK
LP SPI														PAD_LP_SPI_CK
LP I2C							Y				Y		Y	
Analog I2C Controller									Y					
LP I2S Controller [to be added later]							Y				Y		Y	
LP ADC Controller												Y		

Note:
To protect the encryption and decryption peripherals from DPA (Differential Power Analysis) attacks, a random divider strategy is implemented for its function clock. Four security levels are available, depending on the range of the random divider. Users can select the security level by configuring HP_SYS_CLKRST_SEC_DPA_CFG_SEL.
If HP_SYS_CLKRST_SEC_DPA_CFG_SEL is set to 1, the security level is determined by the configuration of HP_SYS_CLKRST_SEC_DPA_LEVEL. Otherwise, the security level is determined by the value of EFUSE_SEC_DPA_LEVEL.

8.3 Programming Procedures

8.3.1 HP System Clock Configuration

- Configure the clock divisor for CPU_CLK: HP_SYS_CLKRST_CPU_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
- Configure the clock divisor for MEM_CLK: HP_SYS_CLKRST_MEM_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
- Configure the clock divisor for SYS_CLK: HP_SYS_CLKRST_SYS_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
- Configure the clock divisor for APB_CLK: HP_SYS_CLKRST_APB_CLK_DIV_NUM for the integral part, HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR for the numerator of the fractional part, and HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR for the denominator of the fractional part.
- Write 1 to HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE to apply the frequency divisor configurations for CPU_CLK, MEM_CLK, SYS_CLK, and APB_CLK.
- Configure the clock source for ROOT_CLK via LP_CLKRST_HP_ROOT_CLK_SRC_SEL.

  Notice: It is recommended to first configure the frequency divisors for the SoC clocks and apply the configurations, and then configure the clock source for ROOT_CLK.
  Before changing the clock source for ROOT_CLK, software needs to ensure that the configured clock divisors do not result in clock frequencies exceeding the allowed maximum frequencies.

8.3.2 LP System Clock Configuration

- Configure the clock source of LP_SLOW_CLK via LP_CLKRST_SLOW_CLK_SEL
- Configure the clock source of LP_FAST_CLK via LP_CLKRST_FAST_CLK_SEL
- Configure the clock divisor of LP_PERI_CLK via LP_CLKRST_LP_PERI_DIV_NUM

8.3.3 Peripheral Clock Reset and Configuration

Notice:

ESP32-P4 features low power consumption. This is why some peripheral clocks are gated (disabled) by default. Before using any of these peripherals, it is mandatory to enable the clock for the given peripheral by setting the corresponding CLK_EN bit to 1, and release the peripheral from the reset state to make it operational by setting the RST_EN bit to 0.

The clocks of most peripherals can be classified into two types:

- Bus clock: used to configure peripheral registers.
- Function clock: such as UART's reference clock, used by peripherals to operate.

The function clock of most peripherals can be selected from multiple clock sources. In the description of the gating registers, it will be stated whether a register is a bus clock (SYS_CLK, APB_CLK) gating register or a function clock gating register.

Bus clock enable/disable registers, function clock enable/disable registers, clock source selection registers, and clock frequency division registers are grouped into the HP_SYS_CLKRST, LP_CLKRST, and LPPERI modules. Except for EMA, SDMMAC, USB Serial/JTAG, Full-Speed USB 2.0 OTG, and High-Speed USB 2.0 OTG, whose reset configuration registers are in the LP_CLKRST module, all the other HP clock and reset registers are in the HP_SYS_CLKRST module. For more information, see Section 8.4 *Register Summary*.

When changing the clock configuration, it is recommended to turn off the corresponding clock gate first, update the clock divider configurations, clock sources, and other configurations, and then turn it on again.

When a peripheral is not working, users can turn off its function clock by configuring related registers. Turning off the peripheral's function clock does not affect the rest of the system.

Take I2C clock configuration as an example.

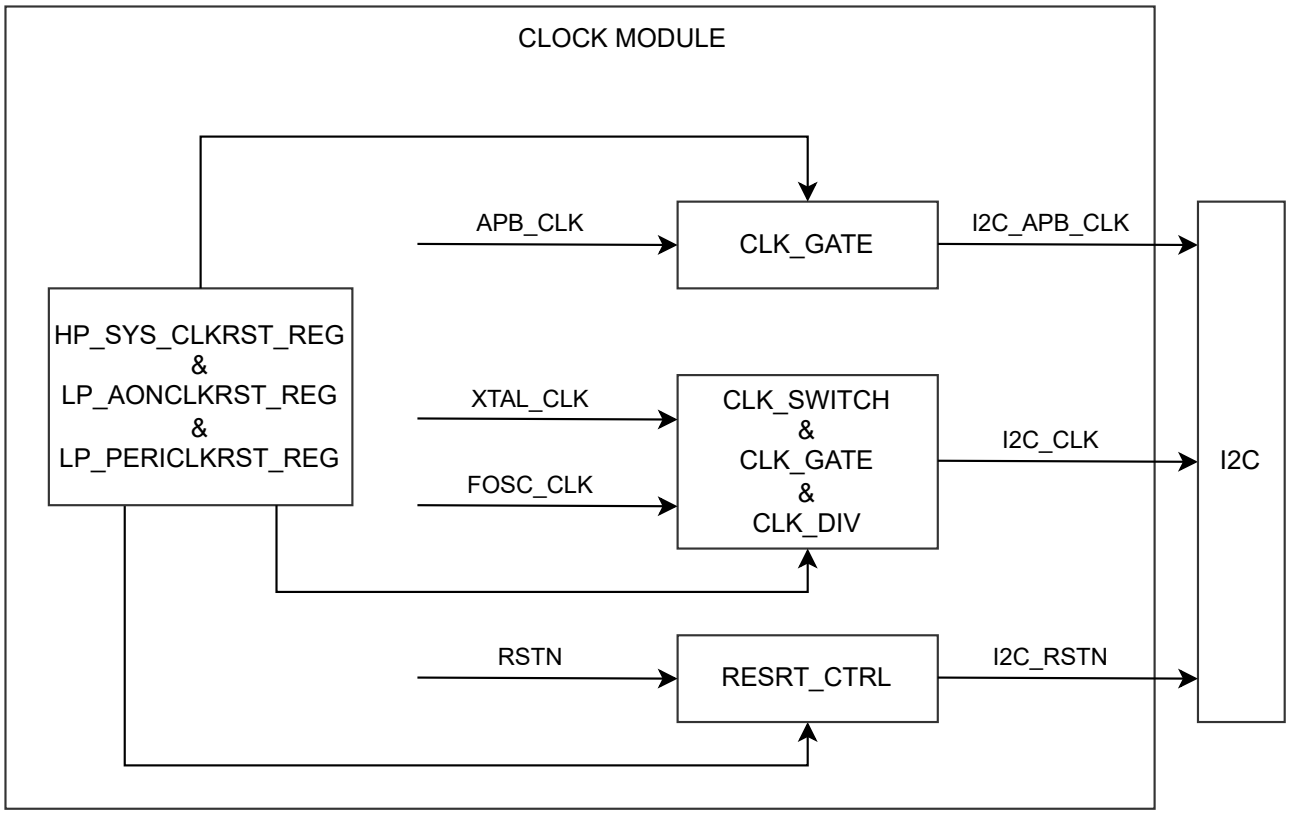


Figure 8.3-1. Clock Configuration Example

Figure 8.3-1 shows the clock structure of I2C. The clock structure of other peripherals is similar to this one. CLK_SWITCH is used to select a clock output and CLK_GATE to turn on/off the clock.

In scenarios that require low power consumption, when the peripheral is not in use, in addition to turning off the function clock, the bus clock of the peripheral can also be turned off to further lower power consumption.

Note that if you turn off the bus clock first, the functional clock may continue working. Therefore, when turning off clocks, it is recommended to turn off the functional clock first and then the bus clock; when turning on clocks, it is recommended to turn on the bus clock first and then the functional clock.

Note:
In this chapter, all divisor configuration registers are configured with the actual divisor minus 1.

8.4 Register Summary

8.4.1 Reset and Clock (HP_SYS_CLKRST) Register Summary

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

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

Name	Description	Address	Access
Clock gate enable register			
HP_SYS_CLKRST_CLK_ENO_REG	Register clock gate enable register	0x0000	R/W
HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG	System and peripheral clock gate configuration register	0x00B4	R/W
Root clock configuration register			
HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG	Root clock configuration register 0	0x0004	varies
HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG	Root clock configuration register 1	0x0008	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG	Root clock configuration register 2	0x000C	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG	Root clock configuration register 3	0x0010	R/W
Bus clock configuration register			
HP_SYS_CLKRST_SOC_CLK_CTRL0_REG	Bus clock configuration register 0	0x0014	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL1_REG	Bus clock configuration register 1	0x0018	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL2_REG	Bus clock configuration register 2	0x001C	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL3_REG	Bus clock configuration register 3	0x0020	R/W
Reference clock configuration register			
HP_SYS_CLKRST_REF_CLK_CTRL0_REG	Reference clock configuration register 0	0x0024	R/W
HP_SYS_CLKRST_REF_CLK_CTRL1_REG	Reference clock configuration register 1	0x0028	R/W
HP_SYS_CLKRST_REF_CLK_CTRL2_REG	Reference clock configuration register 2	0x002C	R/W
Peripheral clock configuration register			
HP_SYS_CLKRST_PERI_CLK_CTRL00_REG	Peripheral clock configuration register	0x0030	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL01_REG	Peripheral clock configuration register	0x0034	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL02_REG	Peripheral clock configuration register	0x0038	varies
HP_SYS_CLKRST_PERI_CLK_CTRL03_REG	Peripheral clock configuration register	0x003C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL10_REG	Peripheral clock configuration register	0x0040	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL11_REG	Peripheral clock configuration register	0x0044	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL12_REG	Peripheral clock configuration register	0x0048	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL13_REG	Peripheral clock configuration register	0x004C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL14_REG	Peripheral clock configuration register	0x0050	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL15_REG	Peripheral clock configuration register	0x0054	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL16_REG	Peripheral clock configuration register	0x0058	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL17_REG	Peripheral clock configuration register	0x005C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL18_REG	Peripheral clock configuration register	0x0060	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL19_REG	Peripheral clock configuration register	0x0064	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL110_REG	Peripheral clock configuration register	0x0068	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL111_REG	Peripheral clock configuration register	0x006C	R/W

Name	Description	Address	Access
HP_SYS_CLKRST_PERI_CLK_CTRL112_REG	Peripheral clock configuration register	0x0070	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL113_REG	Peripheral clock configuration register	0x0074	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL114_REG	Peripheral clock configuration register	0x0078	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL115_REG	Peripheral clock configuration register	0x007C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL116_REG	Peripheral clock configuration register	0x0080	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL117_REG	Peripheral clock configuration register	0x0084	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL118_REG	Peripheral clock configuration register	0x0088	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL119_REG	Peripheral clock configuration register	0x008C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL120_REG	Peripheral clock configuration register	0x0090	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL20_REG	Peripheral clock configuration register	0x0094	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL21_REG	Peripheral clock configuration register	0x0098	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL22_REG	Peripheral clock configuration register	0x009C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL23_REG	Peripheral clock configuration register	0x00A0	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL24_REG	Peripheral clock configuration register	0x00A4	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL25_REG	Peripheral clock configuration register	0x00A8	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL26_REG	Peripheral clock configuration register	0x00AC	R/W
DPA clock configuration register			
HP_SYS_CLKRST_DPA_CTRL0_REG	DPA clock configuration register	0x00B8	R/W
Analog PLL calibration configuration register			
HP_SYS_CLKRST_ANA_PLL_CTRL0_REG	Analog PLL calibration configuration register	0x00BC	varies
Peripheral reset register			
HP_SYS_CLKRST_HP_RST_EN0_REG	Peripheral reset register 0	0x00C0	R/W
HP_SYS_CLKRST_HP_RST_EN1_REG	Peripheral reset register 1	0x00C4	R/W
HP_SYS_CLKRST_HP_RST_EN2_REG	Peripheral reset register 2	0x00C8	R/W
HP_SYS_CLKRST_HP_FORCE_NORST0_REG	Peripheral reset disable register register 0	0x00CC	R/W
HP_SYS_CLKRST_HP_FORCE_NORST1_REG	Peripheral reset disable register register 1	0x00D0	R/W
HP CPU WDT reset configuration register			
HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG	HP CPU0 WDT reset configuration register	0x00D4	R/W
HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG	HP CPU1 WDT reset configuration register	0x00D8	R/W
HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG	HP CPU WDT reset source configuration register	0x00EC	R/W
CPU clock status register			
HP_SYS_CLKRST_CPU_SRC_FREQ0_REG	CPU source clock frequency register	0x00DC	RO
HP_SYS_CLKRST_CPU_CLK_STATUS0_REG	CPU clock status register	0x00E0	RO

8.4.2 LP Always on Clock and Reset (LP_CLKRST) Register Summary

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

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

Name	Description	Address	Access
Configuration registers			
LP_CLKRST_LP_CLK_CONF_REG	LP root clock configuration register	0x0000	R/W
LP_CLKRST_LP_CLK_EN_REG	LP always on peripheral clock configuration register	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP always on peripheral reset configuration register	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	HP CPU and LP CPU reset cause register	0x0010	varies
LP_CLKRST_HPCPU_RESET_CTRL0_REG	HP CPU reset configuration register 0	0x0014	varies
LP_CLKRST_HPCPU_RESET_CTRL1_REG	HP CPU reset configuration register 1	0x0018	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK frequency configuration register	0x001C	R/W
LP_CLKRST_RC32K_CNTL_REG	RC32K_CLK frequency configuration register	0x0020	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K_CLK configuration register	0x0030	R/W
LP_CLKRST_HP_CLK_CTRL_REG	HP clock control register	0x0040	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL0_REG	USB clock and reset control register 0	0x0044	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL1_REG	USB clock and reset control register 1	0x0048	R/W
LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG	SDMMC and EMAC reset control register	0x004C	R/W
Version register			
LP_CLKRST_DATE_REG	Version control register	0x03FC	R/W

8.4.3 LP Peripheral Clock and Reset (LPPERI) Register Summary

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

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

Name	Description	Address	Access
Configuration registers			
LPPERI_CLK_EN_REG	LP peripheral bus clock configuration register	0x0000	R/W
LPPERI_CORE_CLK_SEL_REG	LP peripheral function clock configuration register	0x0004	R/W
LPPERI_RESET_EN_REG	LP peripheral reset configuration register	0x0008	varies
LPPERI_CPU_REG	LP CPU access configuration register	0x000C	R/W
LPPERI_MEM_CTRL_REG	LP UART miscellaneous control register	0x0028	varies
LPPERI_ADC_CTRL_REG	LP ADC clock configuration register	0x002C	R/W
LPPERI_LP_I2S_RXCLK_DIV_NUM_REG	LP I2S RX clock configuration register	0x0030	R/W
LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG	LP I2S RX clock configuration register	0x0034	R/W
Version register			
LPPERI_DATE_REG	Version control register	0x03FC	R/W

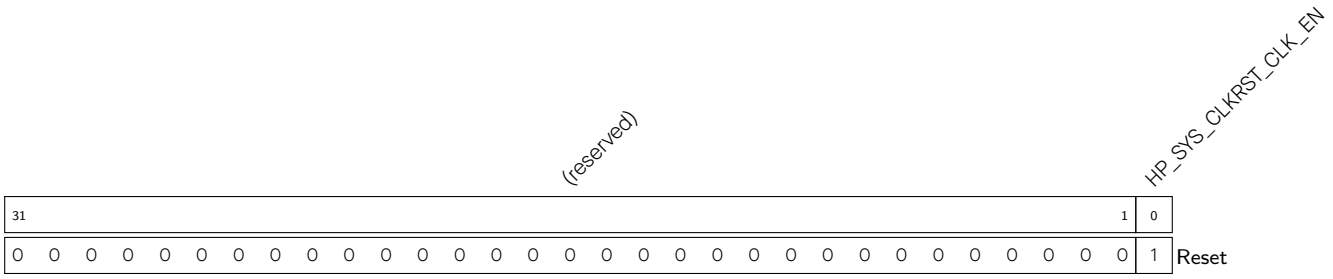
8.5 Registers

8.5.1 Reset and Clock (HP_SYS_CLKRST) Registers

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

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

Register 8.1. HP_SYS_CLKRST_CLK_EN0_REG (0x0000)



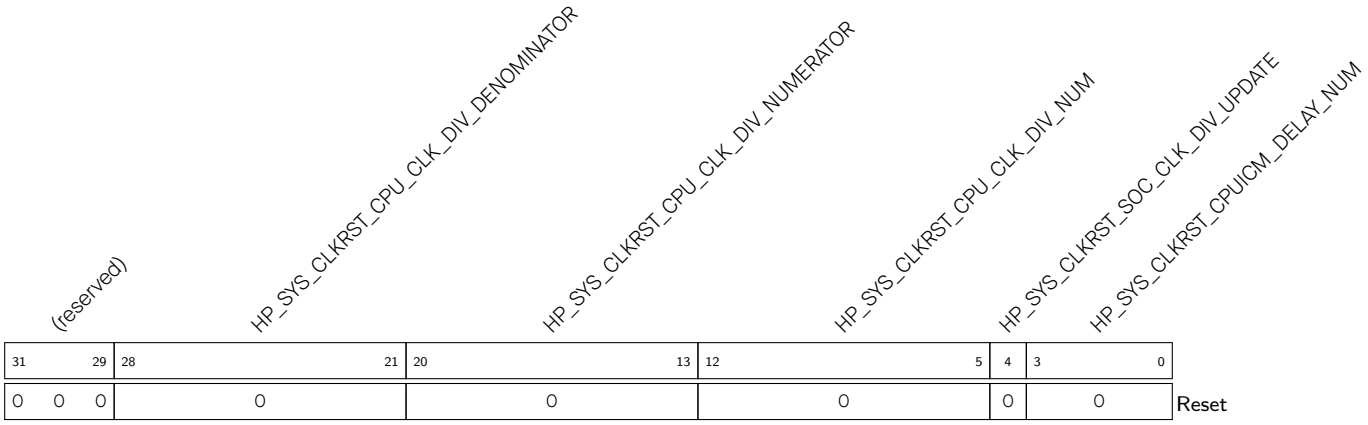
HP_SYS_CLKRST_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Register 8.2. HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG (0x0004)



HP_SYS_CLKRST_CPUICM_DELAY_NUM Configures the time required from entering WFI mode to the actual shutdown of the CPU clock.

Measurement unit: clock cycles. (R/W)

HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE Configures whether to update the divisors for CPU_CLK, MEM_CLK, SYS_CLK, APB_CLK.

0: Not update

1: Update

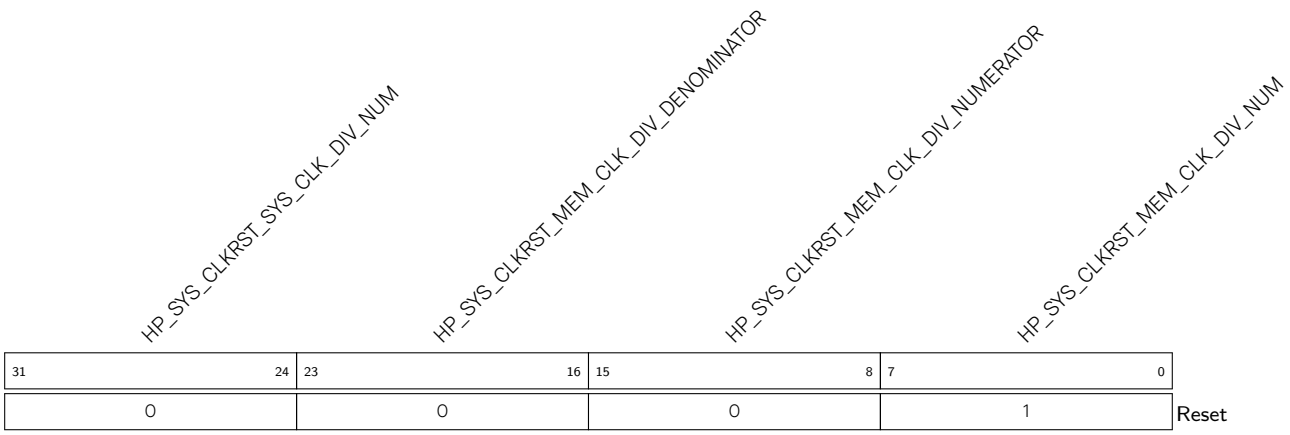
(WT)

HP_SYS_CLKRST_CPU_CLK_DIV_NUM Configures the integer part of the CPU_CLK clock divisor. (R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for CPU_CLK. (R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for CPU_CLK. (R/W)

Register 8.3. HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG (0x0008)



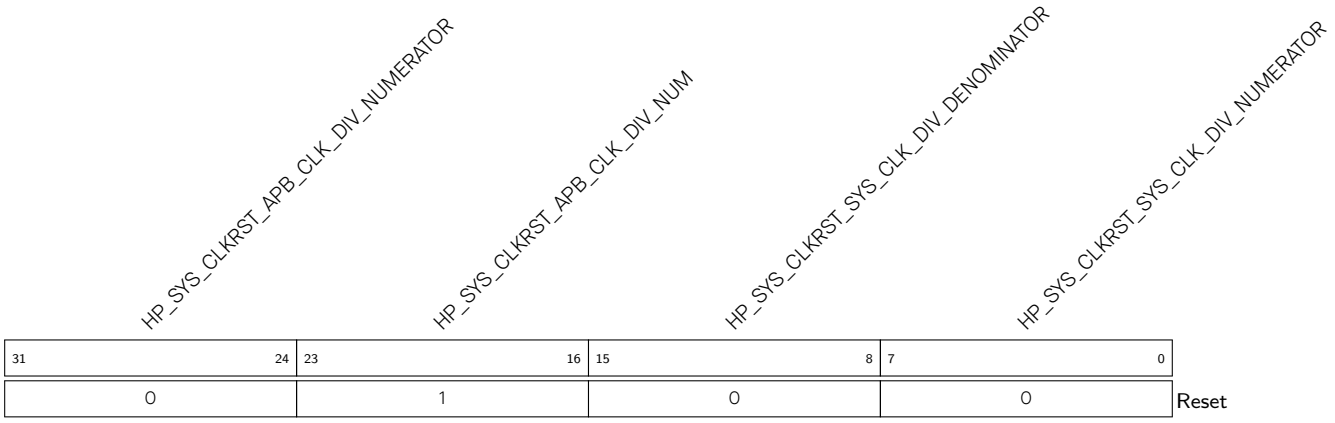
HP_SYS_CLKRST_MEM_CLK_DIV_NUM Configures the integer part of the MEM_CLK clock divisor. (R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for MEM_CLK. (R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for MEM_CLK . (R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_NUM Configures the integer part of the SYS_CLK clock divisor. (R/W)

Register 8.4. HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG (0x000C)



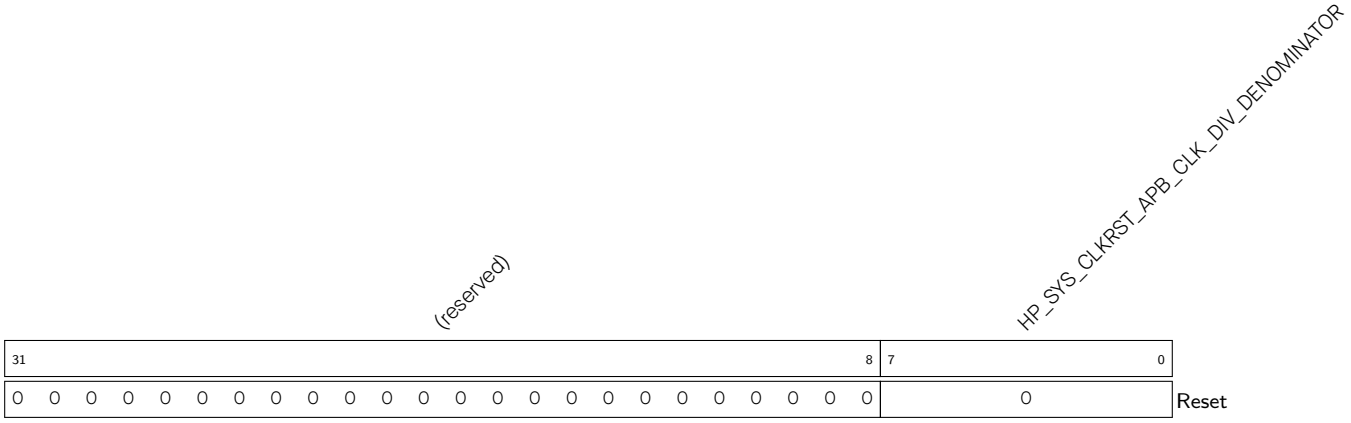
HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for SYS_CLK. (R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for SYS_CLK. (R/W)

HP_SYS_CLKRST_APB_CLK_DIV_NUM Configures the integer part of the APB_CLK fclock divisor. (R/W)

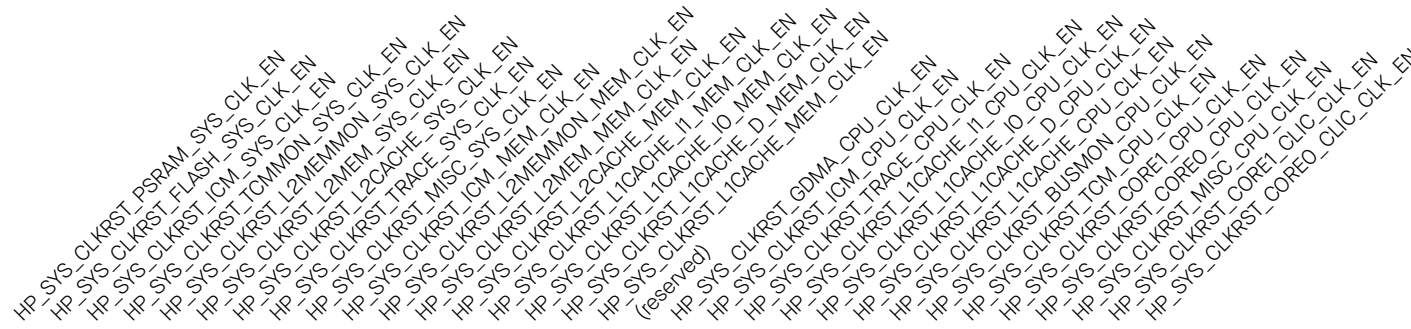
HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for APB_CLK. (R/W)

Register 8.5. HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG (0x0010)



HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for APB_CLK. (R/W)

Register 8.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)



HP_SYS_CLKRST_COREO_CLIC_CLK_EN Configures whether to enable the bus clock of HP CPU0 CLIC.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CORE1_CLIC_CLK_EN Configures whether to enable the bus clock of HP CPU1 CLIC.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MISC_CPU_CLK_EN Configures whether to enable the bus clock of miscellaneous modules (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_COREO_CPU_CLK_EN Configures whether to enable the bus clock of HP CPU0 (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CORE1_CPU_CLK_EN Configures whether to enable the bus clock of HP CPU1 (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TCM_CPU_CLK_EN Configures whether to enable the bus clock of TCM monitor (CPU_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_BUSMON_CPU_CLK_EN Configures whether to enable the bus clock of bus monitor (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_EN Configures whether to enable the bus clock of L1 Cache (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_EN Configures whether to enable the bus clock of L1 D-Cache (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_CPU_CLK_EN Configures whether to enable the bus clock of L1 I-Cache 0 (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_EN Configures whether to enable the bus clock of L1 I-Cache 1 (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_EN Configures whether to enable the bus clock of RISC-V Trace Encoder (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_CPU_CLK_EN Configures the bus clock of the bus module (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_GDMA_CPU_CLK_EN Configures whether to enable the bus clock of DW-GDMA (CPU_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_EN Configures whether to enable the bus clock of L1 Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_EN Configures whether to enable the bus clock of L1 D-Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_EN Configures whether to enable the bus clock of L1 I-Cache0 (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_EN Configures whether to enable the bus clock of L1 I-Cache1 (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2CACHE_MEM_CLK_EN Configures whether to enable the bus clock of L2 Cache (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_L2MEM_MEM_CLK_EN Configures whether to enable the bus clock of L2 memory (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2MEMMON_MEM_CLK_EN Configures whether to enable the bus clock of L1 memory monitor (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_MEM_CLK_EN Configures the bus clock of the bus module (MEM_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_MISC_SYS_CLK_EN Configures whether to enable the bus clock of miscellaneous modules (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_EN Configures whether to enable the bus clock of RISC-V Trace Encoder (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2CACHE_SYS_CLK_EN Configures whether to enable the bus clock of L2 Cache (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_L2MEM_SYS_CLK_EN Configures whether to enable the bus clock of L2 memory (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

Continued from the previous page...

HP_SYS_CLKRST_L2MEMMON_SYS_CLK_EN Configures whether to enable the bus clock of L2 memory monitor (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

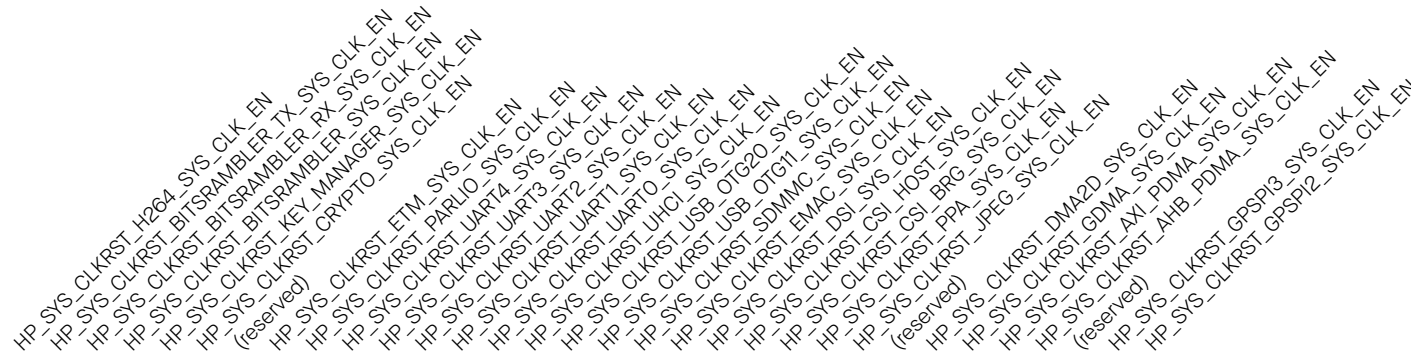
HP_SYS_CLKRST_TCMMON_SYS_CLK_EN Configures whether to enable the bus clock of L2 TCM monitor (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ICM_SYS_CLK_EN Configures the bus clock of the bus module (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_FLASH_SYS_CLK_EN Configures whether to enable the bus clock of flash (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PSRAM_SYS_CLK_EN Configures whether to enable the bus clock of PSRAM (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 8.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	Reset

HP_SYS_CLKRST_GPSPI2_SYS_CLK_EN Configures whether to enable the bus clock of GP-SPI2 (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI3_SYS_CLK_EN Configures whether to enable the bus clock of GP-SPI3 (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_AHB_PDMA_SYS_CLK_EN Configures whether to enable the bus clock of GDMA-AHB (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_AXI_PDMA_SYS_CLK_EN Configures whether to enable the bus clock of GDMA-AXI (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GDMA_SYS_CLK_EN Configures whether to enable the bus clock of DW-GDMA (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_DMA2D_SYS_CLK_EN Configures whether to enable the bus clock of 2DDMA (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_JPEG_SYS_CLK_EN Configures whether to enable the bus clock of JPEG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PPA_SYS_CLK_EN Configures whether to enable the bus clock of PPA (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CSI_BRG_SYS_CLK_EN Configures whether to enable the bus clock of CSI Bridge and ISP (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CSI_HOST_SYS_CLK_EN Configures whether to enable the bus clock of CSI Host (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_DSI_SYS_CLK_EN Configures whether to enable the bus clock of DSI (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_EMAC_SYS_CLK_EN Configures whether to enable the bus clock of EMAC (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_SDMMC_SYS_CLK_EN Configures whether to enable the bus clock of SDMMC (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_OTG11_SYS_CLK_EN Configures whether to enable the bus clock of Full-Speed USB 2.0 OTG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_OTG20_SYS_CLK_EN Configures whether to enable the bus clock of High-Speed USB 2.0 OTG (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UHCI_SYS_CLK_EN Configures whether to enable the bus clock of UHCI (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART0_SYS_CLK_EN Configures whether to enable the bus clock of UART0 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART1_SYS_CLK_EN Configures whether to enable the bus clock of UART1 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

HP_SYS_CLKRST_UART2_SYS_CLK_EN Configures whether to enable the bus clock of UART2 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART3_SYS_CLK_EN Configures whether to enable the bus clock of UART3 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART4_SYS_CLK_EN Configures whether to enable the bus clock of UART4 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PARLIO_SYS_CLK_EN Configures whether to enable the bus clock of PARLIO (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_ETM_SYS_CLK_EN Configures whether to enable the bus clock of ETM (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W) 0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_CRYPTO_SYS_CLK_EN Configures whether to enable the bus clock of the crypto modules (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_KEY_MANAGER_SYS_CLK_EN Configures whether to enable the bus clock of the Key Manager (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

Continued from the previous page...

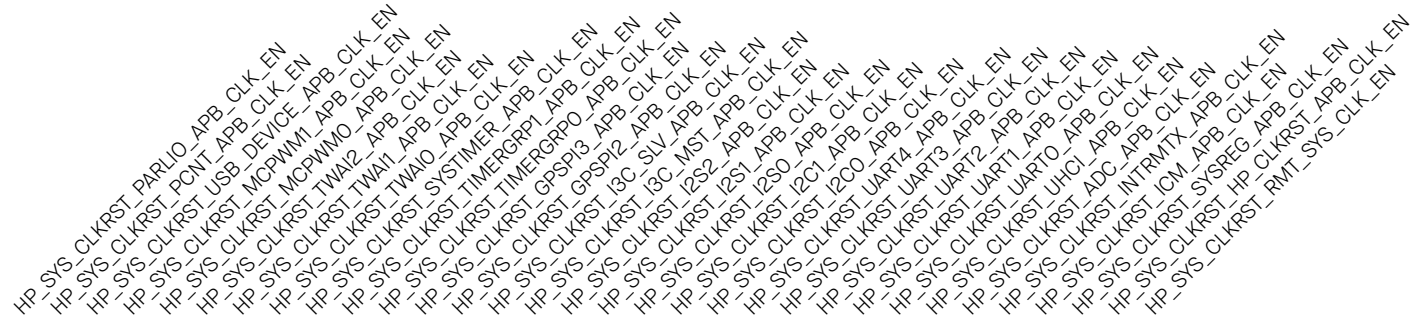
HP_SYS_CLKRST_BITSRAMBLER_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_BITSRAMBLER_RX_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler RX (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_BITSRAMBLER_TX_SYS_CLK_EN Configures whether to enable the bus clock of the Bit Scrambler TX (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_H264_SYS_CLK_EN Configures whether to enable the bus clock of H264 (SYS_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 8.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	0	Reset

HP_SYS_CLKRST_RMT_SYS_CLK_EN Configures whether to enable the bus clock of RMT (SYS_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_HP_CLKRST_APB_CLK_EN Configures whether to enable the bus clock of the Reset and Clock module (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SYSREG_APB_CLK_EN Configures whether to enable the bus clock of the HP System Registers (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ICM_APB_CLK_EN Configures the bus clock of the bus module (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_INTRMTX_APB_CLK_EN Configures whether to enable the bus clock of the Interrupt Matrix (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ADC_APB_CLK_EN Configures whether to enable the bus clock of the ADC (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_UHCI_APB_CLK_EN Configures whether to enable the bus clock of the UHCI (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART0_APB_CLK_EN Configures whether to enable the bus clock of UART0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART1_APB_CLK_EN Configures whether to enable the bus clock of UART1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART2_APB_CLK_EN Configures whether to enable the bus clock of UART2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART3_APB_CLK_EN Configures whether to enable the bus clock of UART3 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_UART4_APB_CLK_EN Configures whether to enable the bus clock of UART4 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2C0_APB_CLK_EN Configures whether to enable the bus clock of I2C0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_I2C1_APB_CLK_EN Configures whether to enable the bus clock of I2C1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S0_APB_CLK_EN Configures whether to enable the bus clock of I2S0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S1_APB_CLK_EN Configures whether to enable the bus clock of I2S1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S2_APB_CLK_EN Configures whether to enable the bus clock of I2S2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_MST_APB_CLK_EN Configures whether to enable the bus clock of the I3C Master (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_SLV_APB_CLK_EN Configures whether to enable the bus clock of the I3C Slave (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_GPSPI2_APB_CLK_EN Configures whether to enable the bus clock of GP-SPI2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_GPSPI3_APB_CLK_EN Configures whether to enable the bus clock of GP-SPI3 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_APB_CLK_EN Configures whether to enable the bus clock of Timer Group 0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP1_APB_CLK_EN Configures whether to enable the bus clock of Timer Group 1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_SYSTIMER_APB_CLK_EN Configures whether to enable the bus clock of the System Timer (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI0_APB_CLK_EN Configures whether to enable the bus clock of TWAI0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI1_APB_CLK_EN Configures whether to enable the bus clock of TWAI1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TWAI2_APB_CLK_EN Configures whether to enable the bus clock of TWAI2 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

Continued from the previous page...

HP_SYS_CLKRST_MCPWM0_APB_CLK_EN Configures whether to enable the bus clock of MCPWM0 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

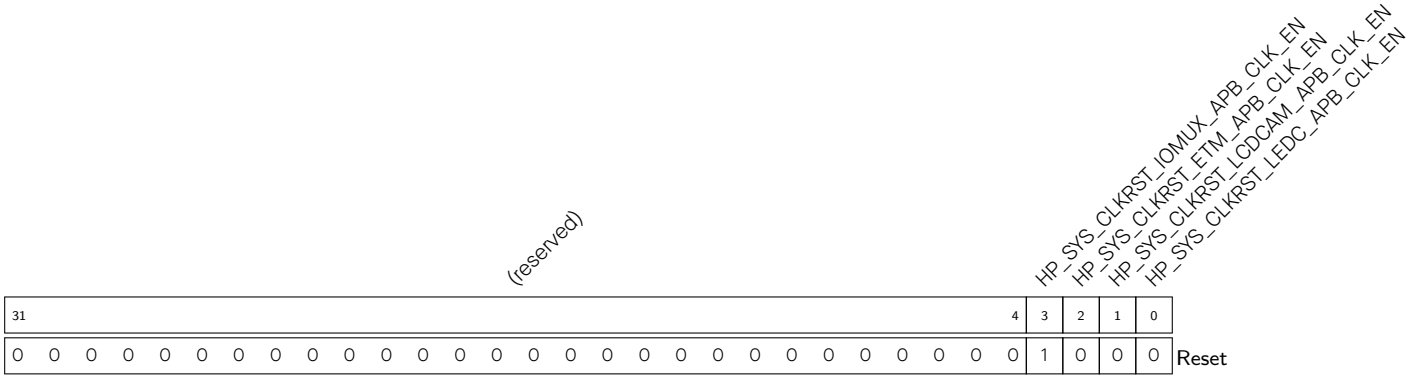
HP_SYS_CLKRST_MCPWM1_APB_CLK_EN Configures whether to enable the bus clock of MCPWM1 (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_USB_DEVICE_APB_CLK_EN Configures whether to enable the bus clock of the USB Serial/JTAG (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PCNT_APB_CLK_EN Configures whether to enable the bus clock of the PCNT (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_PARLIO_APB_CLK_EN Configures whether to enable the bus clock of the PARLIO (APB_CLK clock domain).
0: Disable
1: Enable
(R/W)

Register 8.9. HP_SYS_CLKRST_SOC_CLK_CTRL3_REG (0x0020)



HP_SYS_CLKRST_LEDC_APB_CLK_EN Configures whether to enable the bus clock of LEDC (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_LCDCAM_APB_CLK_EN Configures whether to enable the bus clock of the LCD_CAM (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ETM_APB_CLK_EN Configures whether to enable the bus clock of ETM (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

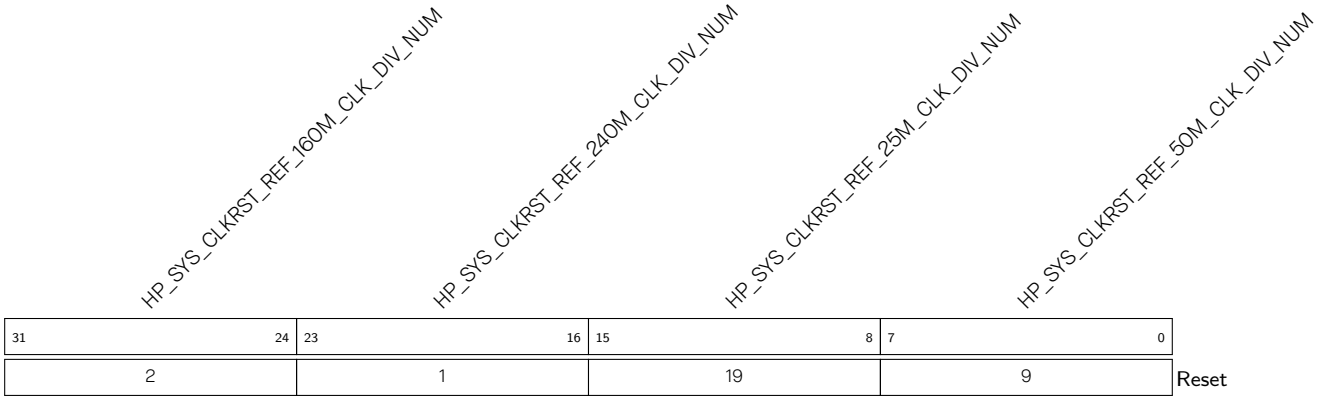
HP_SYS_CLKRST_IOMUX_APB_CLK_EN Configures whether to enable the bus clock of IO MUX (APB_CLK clock domain).

0: Disable

1: Enable

(R/W)

Register 8.10. HP_SYS_CLKRST_REF_CLK_CTRL0_REG (0x0024)



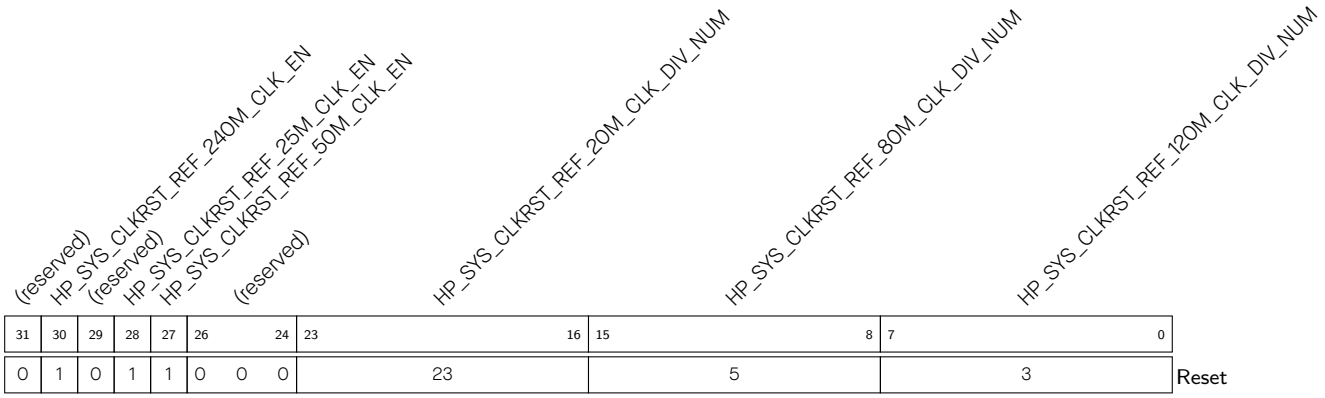
HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM Configures the divisor of PLL_F50M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_25M_CLK_DIV_NUM Configures the divisor of PLL_F25M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_240M_CLK_DIV_NUM Configures the divisor of PLL_F240M_CLK. It is not recommended for users to change. (R/W)

HP_SYS_CLKRST_REF_160M_CLK_DIV_NUM Configures the divisor of PLL_F160M_CLK. It is not recommended for users to change. (R/W)

Register 8.11. HP_SYS_CLKRST_REF_CLK_CTRL1_REG (0x0028)



HP_SYS_CLKRST_REF_120M_CLK_DIV_NUM Configures the divisor of PLL_F120M_CLK. (R/W)

HP_SYS_CLKRST_REF_80M_CLK_DIV_NUM Configures the divisor of PLL_F80M_CLK. (R/W)

HP_SYS_CLKRST_REF_20M_CLK_DIV_NUM Configures the divisor of PLL_F20M_CLK. (R/W)

HP_SYS_CLKRST_REF_50M_CLK_EN Configures whether to enable PLL_F50M_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_REF_25M_CLK_EN Configures whether to enable PLL_F25M_CLK.

0: Disable

1: Enable

(R/W)

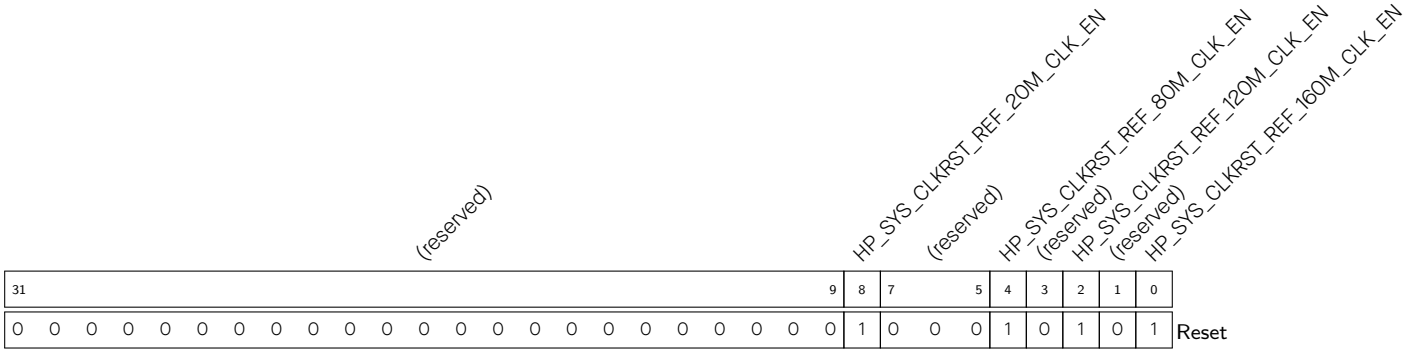
HP_SYS_CLKRST_REF_240M_CLK_EN Configures whether to enable PLL_F240M_CLK.

0: Disable

1: Enable

(R/W)

Register 8.12. HP_SYS_CLKRST_REF_CLK_CTRL2_REG (0x002C)



HP_SYS_CLKRST_REF_160M_CLK_EN Configures whether to enable PLL_F160M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

HP_SYS_CLKRST_REF_120M_CLK_EN Configures whether to enable PLL_F120M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

HP_SYS_CLKRST_REF_80M_CLK_EN Configures whether to enable PLL_F80M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

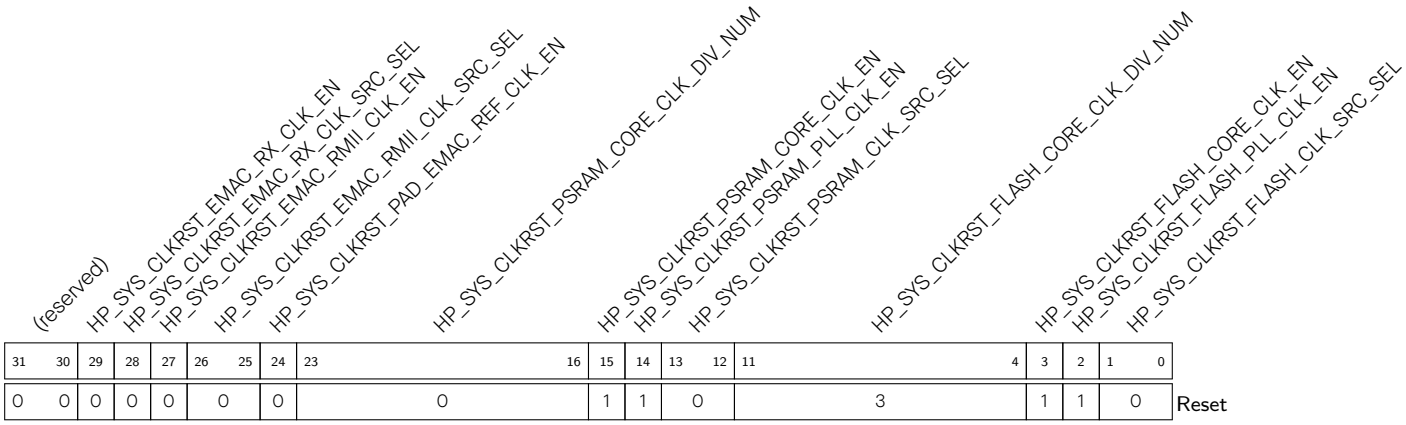
HP_SYS_CLKRST_REF_20M_CLK_EN Configures whether to enable PLL_F20M_CLK.

0: Disable

1: Enable

User modification is not recommended. (R/W)

Register 8.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)



HP_SYS_CLKRST_FLASH_CLK_SRC_SEL Configures the clock source for FLASH_CLK.

0: XTAL_CLK

1: SPLL_CLK (480 MHz)

2: CPLL_CLK (400 MHz)

3: Invalid

(R/W)

HP_SYS_CLKRST_FLASH_PLL_CLK_EN Configures whether to enable FLASH_PLL_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_EN Configures whether to enable FLASH_CORE_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_DIV_NUM Configures the clock divisor of FLASH_CORE_CLK. (R/W)

HP_SYS_CLKRST_PSRAM_CLK_SRC_SEL Configures the clock source for PSRAM_CLK.

0: XTAL_CLK

1: MPLL_CLK

2: SPLL_CLK (480 MHz)

3: CPLL_CLK (400 MHz)

(R/W)

HP_SYS_CLKRST_PSRAM_PLL_CLK_EN Configures whether to enable PSRAM_PLL_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PSRAM_CORE_CLK_EN Configures whether to enable PSRAM_CORE_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)

Continued from the previous page...

HP_SYS_CLKRST_PSRAM_CORE_CLK_DIV_NUM Configures the clock divisor of PSRAM_CORE_CLK. (R/W)

HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN Configures whether to enable PAD_EMAC_REF_CLK.
0: Disable
1: Enable
(R/W)

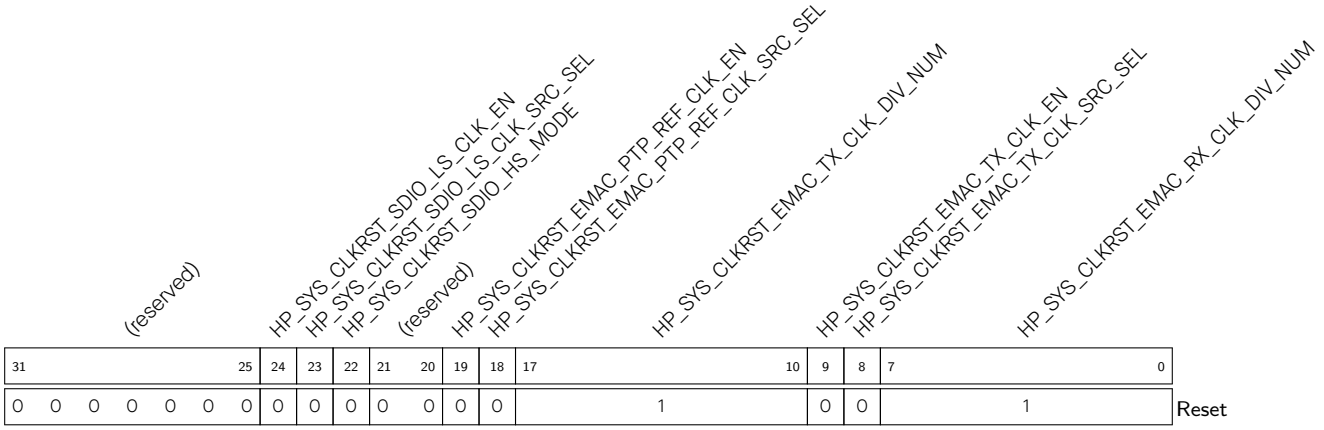
HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL Configures the clock source for EMAC_RMII_CLK.
0: PAD_EMAC_TXRX_CLK
1: PAD_EMAC_RX_CLK
2: PAD_EMAC_TX_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_EMAC_RMII_CLK_EN Configures whether to enable EMAC_RMII_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL Configures the clock source for EMAC_RX_CLK.
0: PAD_EMAC_TXRX_CLK
1: PAD_EMAC_RX_CLK
(R/W)

HP_SYS_CLKRST_EMAC_RX_CLK_EN Configures whether to enable EMAC_RX_CLK.
0: Disable
1: Enable
(R/W)

Register 8.14. HP_SYS_CLKRST_PERI_CLK_CTRL01_REG (0x0034)



HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM Configures the clock divisor for EMAC_RX_CLK. (R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL Configures the clock source for EMAC_TX_CLK.

0: PAD_EMAC_TXRX_CLK

1: PAD_EMAC_TX_CLK

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_EN Configures whether to enable EMAC_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM Configures the clock divisor for EMAC_TX_CLK. (R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_SRC_SEL Configures the clock source for EMAC_PTP_REF_CLK.

0: XTAL_CLK

1: PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_EN Configures whether to enable EMAC_PTP_REF_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.14. HP_SYS_CLKRST_PERI_CLK_CTRL01_REG (0x0034)

Continued from the previous page...

HP_SYS_CLKRST_SDIO_HS_MODE Configures the mode of SDIO clock.

0: Low-speed mode

1: High-speed mode

(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_SRC_SEL Configures the low-speed clock source for SDIO.

0: PLL_F160M_CLK

1: APLL_CLK

(R/W)

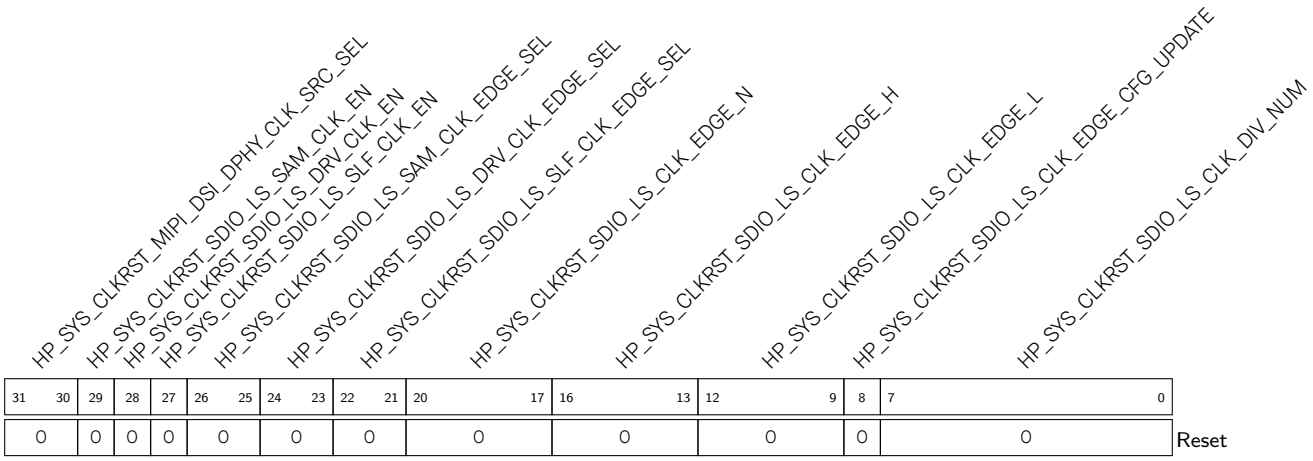
HP_SYS_CLKRST_SDIO_LS_CLK_EN Configures whether to enable SDIO low-speed clock.

0: Disable

1: Enable

(R/W)

Register 8.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)



HP_SYS_CLKRST_SDIO_LS_CLK_DIV_NUM Configures the divisor of the SDIO low-speed clock. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_CFG_UPDATE Configures whether to update the SDIO low-speed clock phase configuration.

0: Not update

1: Update (WT)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_L Configures the threshold for the SDIO low-speed clock signal to switch to the low level. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_H Configures the threshold for the SDIO low-speed clock signal to switch to the high level. (R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_N Configures the threshold for clearing the count value of the SDIO low-speed clock. (R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EDGE_SEL Configures SDIO_SLF_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EDGE_SEL Configures SDIO_DRV_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EDGE_SEL Configures SDIO_SAM_CLK to select the SDIO low-speed clock phase. (R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EN Configures whether to enable SDIO_SLF_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)

Continued from the previous page...

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EN Configures whether to enable SDIO_DRV_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EN Configures whether to enable SDIO_SAM_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPHY_CLK_SRC_SEL Configures the clock source for DSI_DPHY_CFG_CLK and DSI_DPHY_PLL_REFCLK.

0: PLL_F20M_CLK

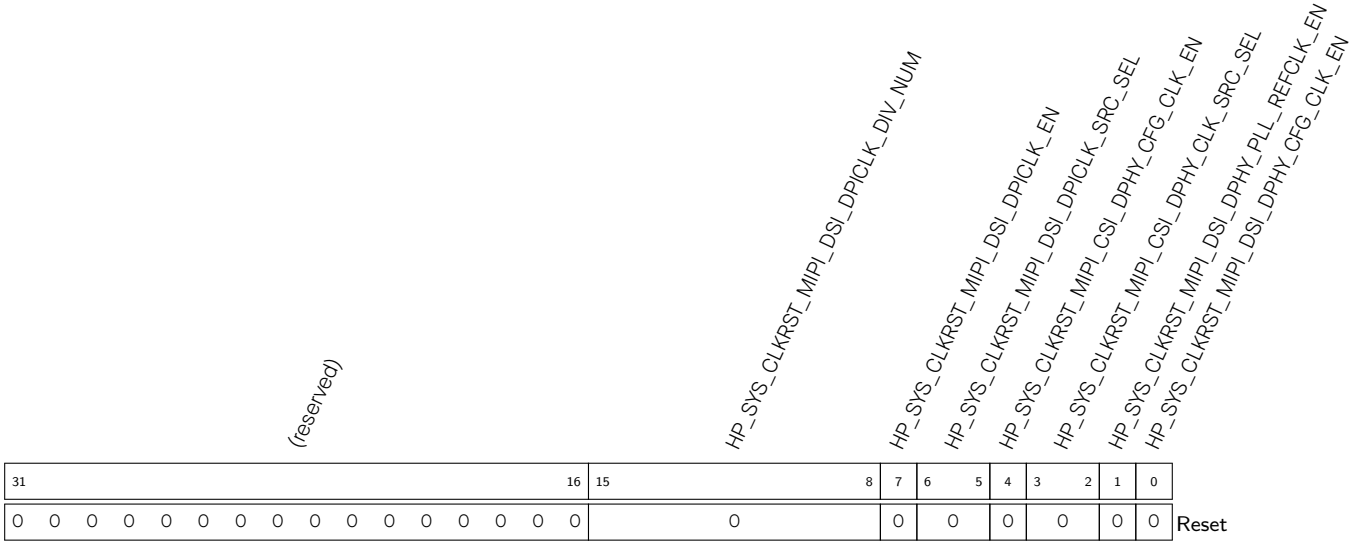
1: RC_FAST_CLK

2: PLL_F25M_CLK

3: Invalid

(R/W)

Register 8.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)



HP_SYS_CLKRST_MIPI_DSI_DPHY_CFG_CLK_EN Configures whether to enable DSI_DPHY_CFG_CLK clock.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPHY_PLL_REFCLK_EN Configures whether to enable DSI_DPHY_PLL_REFCLK clock.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CLK_SRC_SEL Configures the clock source for CSI_DPHY_CFG_CLK.
0: PLL_F20M_CLK
1: RC_FAST_CLK
2: PLL_F25M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CFG_CLK_EN Configures whether to enable CSI_DPHY_CFG_CLK clock.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_SRC_SEL Configures the clock source for DSI_DPI_CLK.
0: XTAL_CLK
1: PLL_F240M_CLK
2: PLL_F160M_CLK
3: Invalid
(R/W)

Continued on the next page...

Register 8.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)

Continued from the previous page...

HP_SYS_CLKRST_MIPI_DSI_DPICLK_EN Configures whether to enable DSI_DPI_CLK clock.

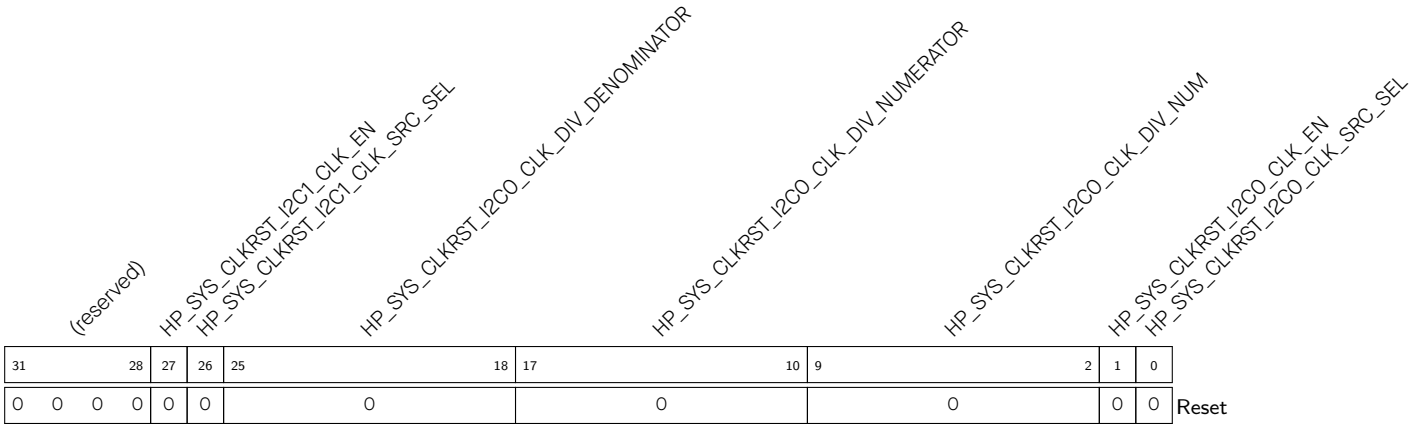
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_DIV_NUM Configures the clock divisor of DSI_DPI_CLK. (R/W)

Register 8.17. HP_SYS_CLKRST_PERI_CLK_CTRL10_REG (0x0040)



HP_SYS_CLKRST_I2C0_CLK_SRC_SEL Configures the clock source for I2C0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_I2C0_CLK_EN Configures whether to enable I2C0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUM Configures the integer part of the clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for clock divisor of I2C0_CLK. (R/W)

HP_SYS_CLKRST_I2C1_CLK_SRC_SEL Configures the clock source for I2C1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

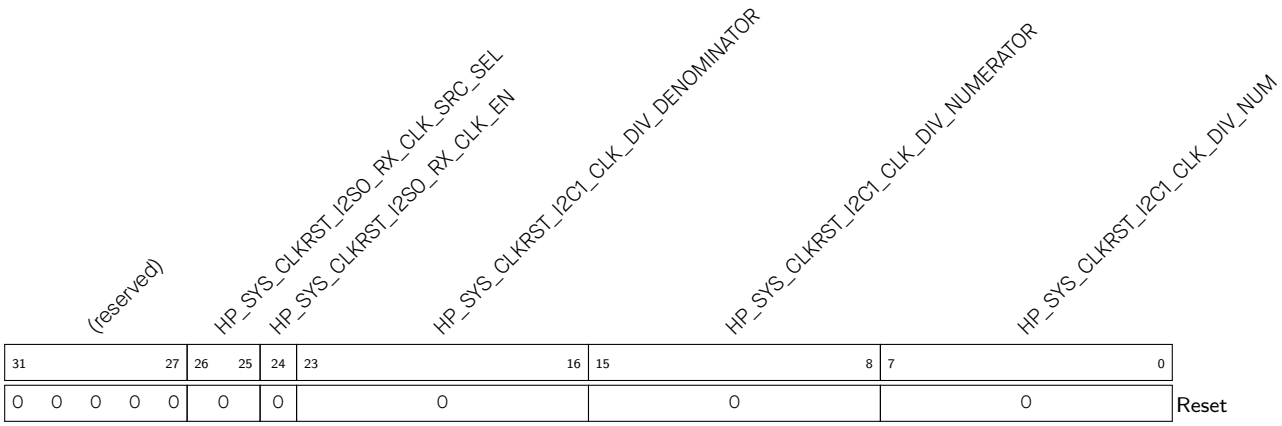
HP_SYS_CLKRST_I2C1_CLK_EN Configures whether to enable I2C1_CLK.

0: Disable

1: Enable

(R/W)

Register 8.18. HP_SYS_CLKRST_PERI_CLK_CTRL11_REG (0x0044)



HP_SYS_CLKRST_I2C1_CLK_DIV_NUM Configures the integer part of the I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for I2C1_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_EN Configures whether to enable I2S0_RX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_SRC_SEL Configures the clock source for I2S0_RX_CLK.

0: XTAL_CLK

1: APLL_CLK

2: PAD_I2S0_MCLK

3: Invalid

(R/W)

Register 8.19. HP_SYS_CLKRST_PERI_CLK_CTRL12_REG (0x0048)

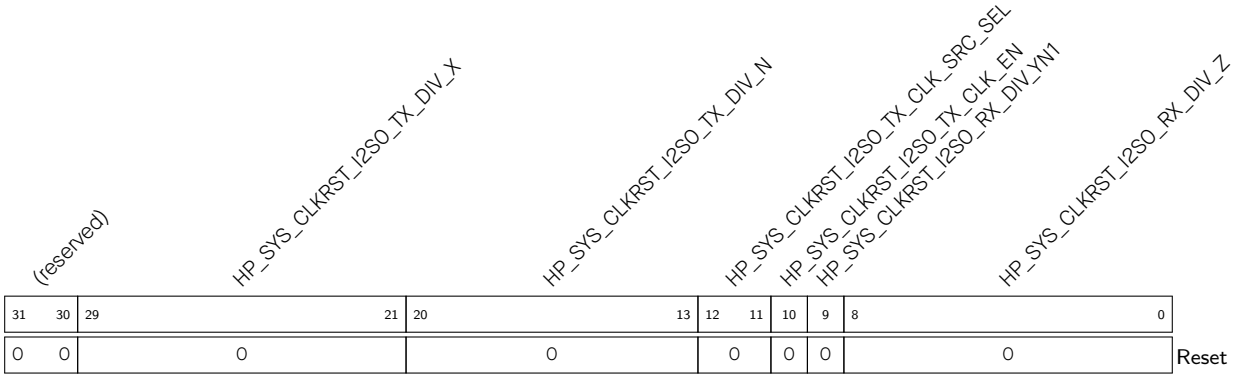


HP_SYS_CLKRST_I2S0_RX_DIV_N Configures the integer part of the I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_X Configures the coefficient x of the I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_Y Configures the coefficient y of the I2S0_RX_CLK clock divisor. (R/W)

Register 8.20. HP_SYS_CLKRST_PERI_CLK_CTRL13_REG (0x004C)



HP_SYS_CLKRST_I2S0_RX_DIV_Z Configures the coefficient z of I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_YN1 Configures the coefficient yn1 of I2S0_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_EN Configures whether to enable I2S0_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_SRC_SEL Configures the clock source for I2S0_TX_CLK.

0: XTAL_CLK

1: APLL_CLK

2: PAD_I2S0_MCLK

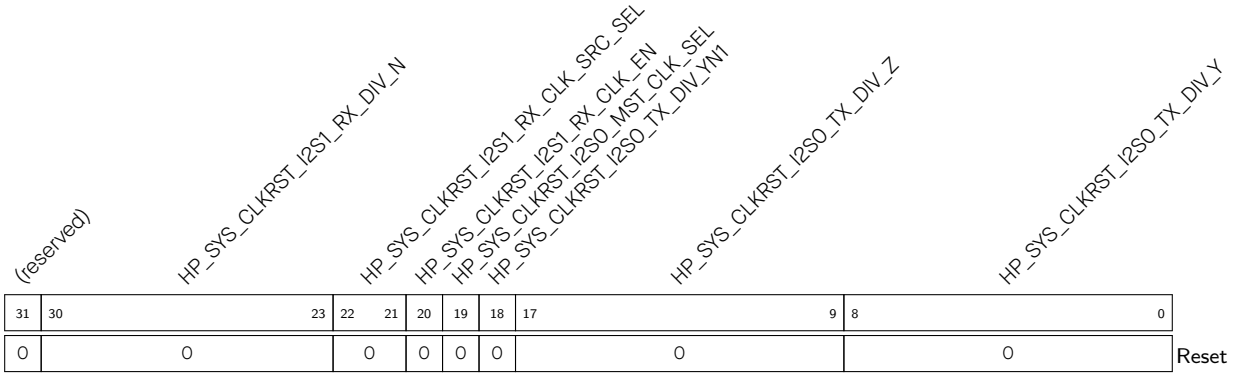
3: Invalid

(R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_N Configures the integer part of the clock divisor of I2S0_TX_CLK. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_X Configures the coefficient x of I2S0_TX_CLK clock divisor. (R/W)

Register 8.21. HP_SYS_CLKRST_PERI_CLK_CTRL14_REG (0x0050)



HP_SYS_CLKRST_I2S0_TX_DIV_Y Configures the coefficient y for I2S0_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_Z Configures the coefficient z for I2S0_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_YN1 Configures the coefficient yn1 for I2S0_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S0_MST_CLK_SEL Configures the clock source for the output clock PAD_I2S0_MST_CLK.

0: I2S0_RX_CLK

1: I2S0_TX_CLK

(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_EN Configures whether to enable I2S1_RX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_SRC_SEL Configures the clock source for I2S1_RX_CLK.

0: XTAL_CLK

1: APLL_CLK

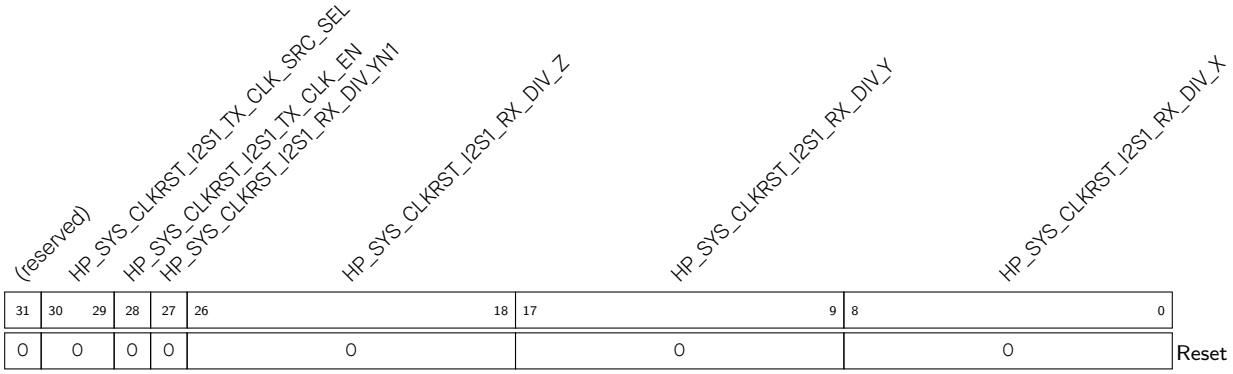
2: PAD_I2S1_MCLK

3: Invalid

(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_N Configures the integer part of the clock divisor for I2S1_RX_CLK. (R/W)

Register 8.22. HP_SYS_CLKRST_PERI_CLK_CTRL15_REG (0x0054)



HP_SYS_CLKRST_I2S1_RX_DIV_X Configures the coefficient x for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Y Configures the coefficient y for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Z Configures the coefficient z for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_YN1 Configures the coefficient yn1 for I2S1_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_EN Configures whether to enable PI2S1_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_SRC_SEL Configures the clock source for I2S1_TX_CLK.

0: XTAL_CLK

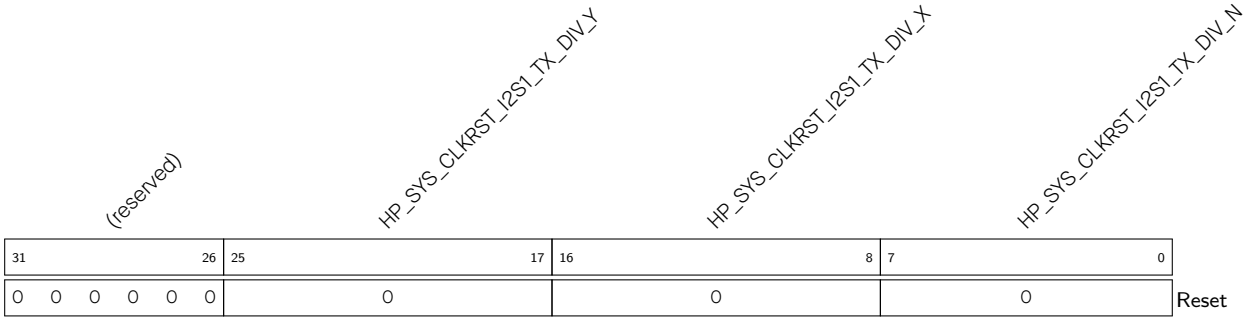
1: APLL_CLK

2: PAD_I2S1_MCLK

3: Invalid

(R/W)

Register 8.23. HP_SYS_CLKRST_PERI_CLK_CTRL16_REG (0x0058)

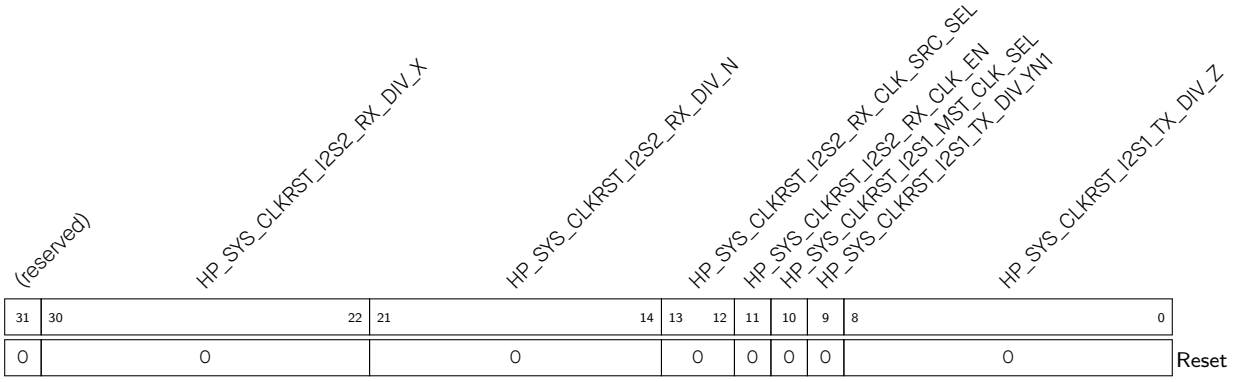


HP_SYS_CLKRST_I2S1_TX_DIV_N Configures the integer part of the clock divisor for I2S1_TX_CLK. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_X Configures the coefficient x for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_Y Configures the coefficient y for I2S1_TX_CLK clock divisor. (R/W)

Register 8.24. HP_SYS_CLKRST_PERI_CLK_CTRL17_REG (0x005C)



HP_SYS_CLKRST_I2S1_TX_DIV_Z Configures the coefficient z for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_YN1 Configures the coefficient yn1 for I2S1_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S1_MST_CLK_SEL Configures the clock source for the output clock PAD_I2S1_MST_CLK.
0: I2S1_RX_CLK
1: I2S1_TX_CLK
(R/W)

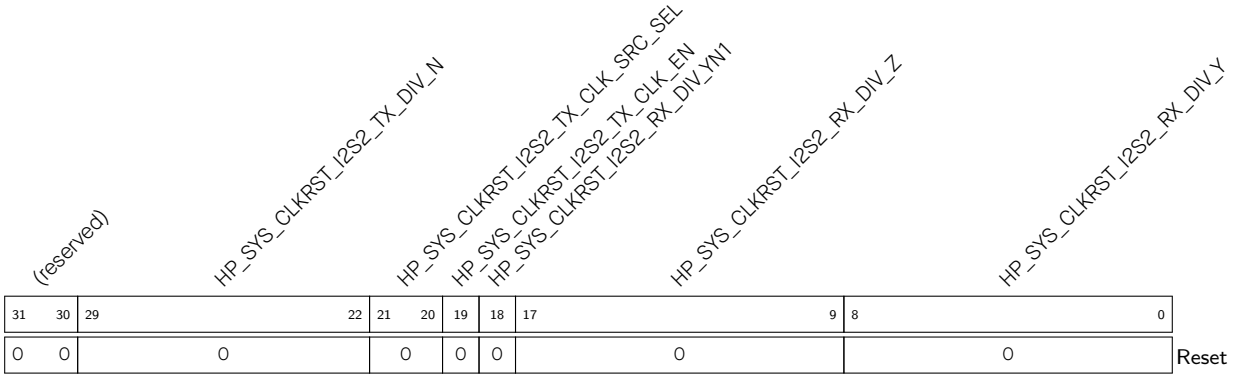
HP_SYS_CLKRST_I2S2_RX_CLK_EN Configures whether to enable I2S2_RX_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I2S2_RX_CLK_SRC_SEL Configures the clock source for I2S2_RX_CLK.
0: XTAL_CLK
1: APLL_CLK
2: PAD_I2S2_MCLK
3: Invalid
(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_N Configures the integer part of the clock divisor for I2S2_RX_CLK. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_X Configures the coefficient x for I2S2_RX_CLK clock divisor. (R/W)

Register 8.25. HP_SYS_CLKRST_PERI_CLK_CTRL18_REG (0x0060)



HP_SYS_CLKRST_I2S2_RX_DIV_Y Configures the coefficient y for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_Z Configures the coefficient z for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_YN1 Configures the coefficient yn1 for I2S2_RX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_EN Configures whether to enable I2S2_TX_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_SRC_SEL Configures the clock source for I2S2_TX_CLK.

0: XTAL_CLK

1: APLL_CLK

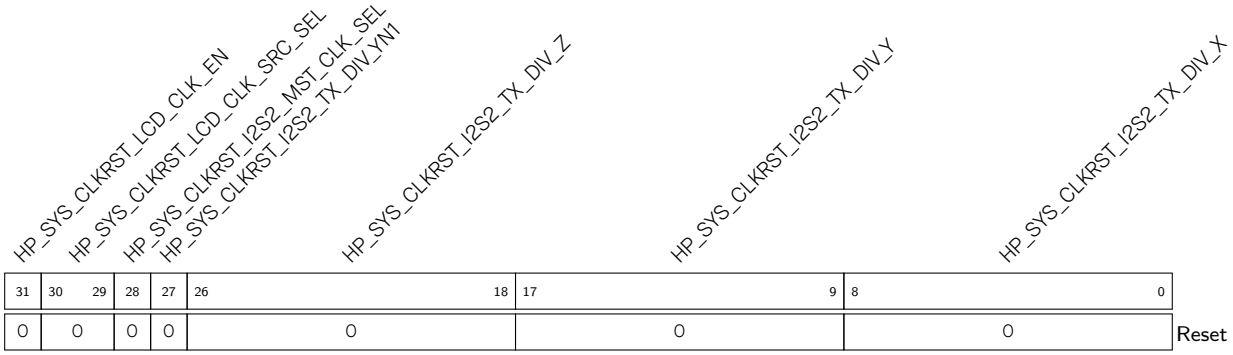
2: PAD_I2S2_MCLK

3: Invalid

(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_N Configures the integer part for I2S2_TX_CLK clock divisor. (R/W)

Register 8.26. HP_SYS_CLKRST_PERI_CLK_CTRL19_REG (0x0064)



HP_SYS_CLKRST_I2S2_TX_DIV_X Configures the coefficient x of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Y Configures the coefficient y of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Z Configures the coefficient z of I2S2_TX_CLK clock divisor. (R/W)

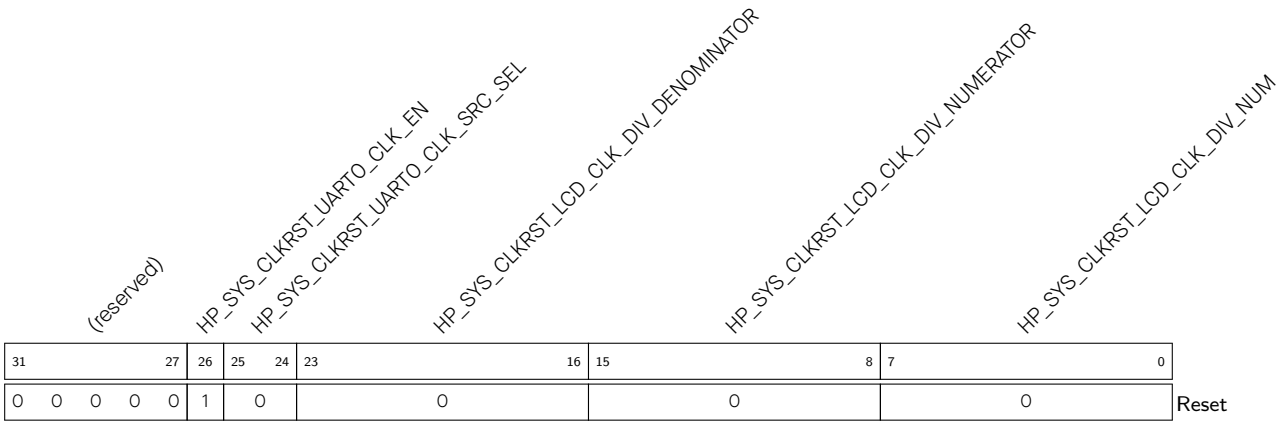
HP_SYS_CLKRST_I2S2_TX_DIV_YN1 Configures the coefficient yn1 of I2S2_TX_CLK clock divisor. (R/W)

HP_SYS_CLKRST_I2S2_MST_CLK_SEL Configures the clock source for output clock PAD_I2S2_MST_CLK.
0: I2S2_RX_CLK
1: I2S2_TX_CLK
(R/W)

HP_SYS_CLKRST_LCD_CLK_SRC_SEL Configures the clock source for LCD_CLK.
0: XTAL_CLK
1: PLL_F160M_CLK
2: APLL_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_LCD_CLK_EN Configures whether to enable LCD_CLK.
0: Disable
1: Enable
(R/W)

Register 8.27. HP_SYS_CLKRST_PERI_CLK_CTRL110_REG (0x0068)



HP_SYS_CLKRST_LCD_CLK_DIV_NUM Configures the integer part of the LCD_CLK clock divisor. (R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for LCD_CLK. (R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for LCD_CLK. (R/W)

HP_SYS_CLKRST_UART0_CLK_SRC_SEL Configures the clock source for UART0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

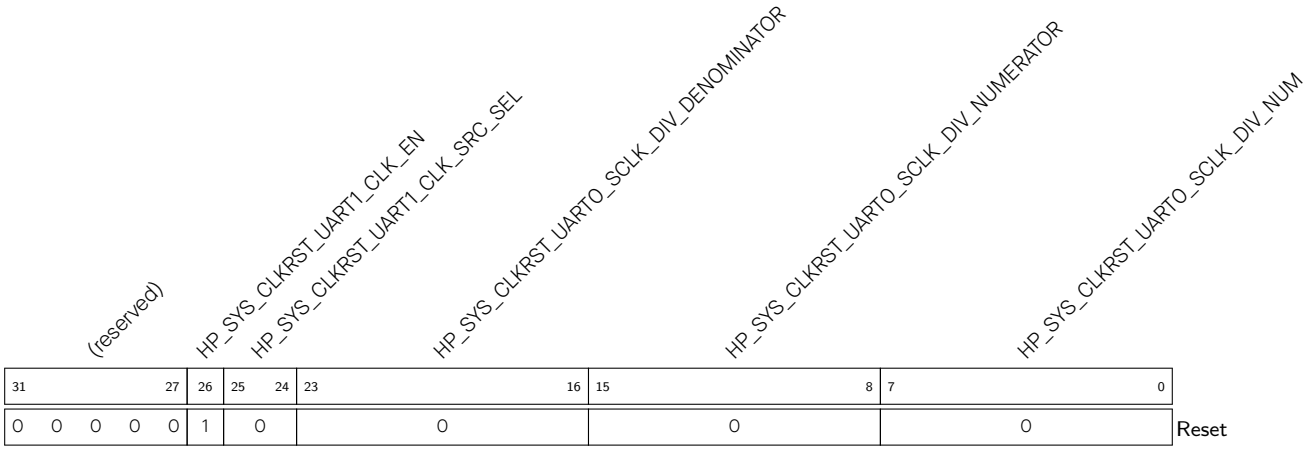
HP_SYS_CLKRST_UART0_CLK_EN Configures whether to enable UART0_CLK.

0: Disable

1: Enable

(R/W)

Register 8.28. HP_SYS_CLKRST_PERI_CLK_CTRL111_REG (0x006C)



HP_SYS_CLKRST_UART0_SCLK_DIV_NUM Configures the integer part of the UART0_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for UART0_SCLK. (R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for UART0_SCLK. (R/W)

HP_SYS_CLKRST_UART1_CLK_SRC_SEL Configures the clock source for UART1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

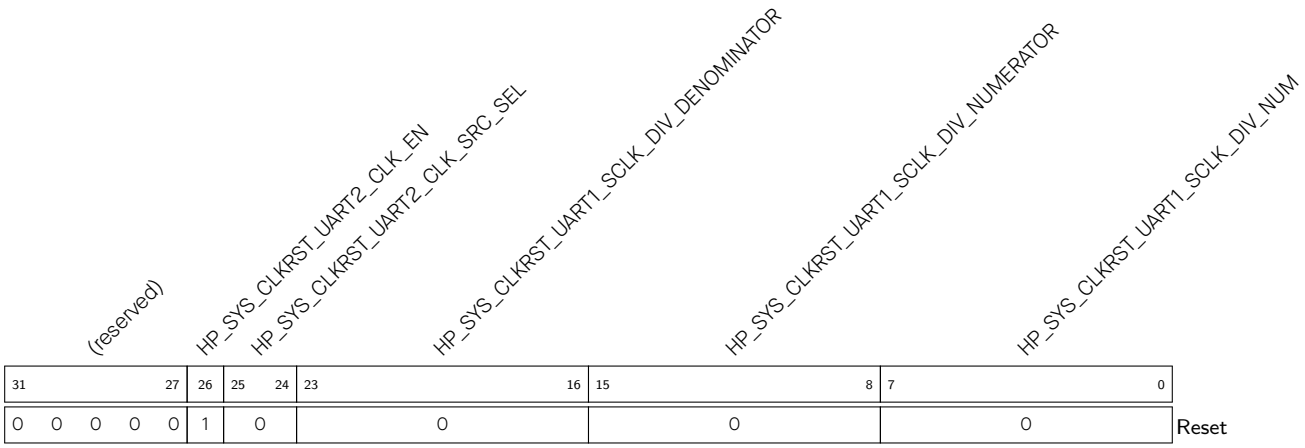
HP_SYS_CLKRST_UART1_CLK_EN Configures whether to enable UART1_CLK.

0: Disable

1: Enable

(R/W)

Register 8.29. HP_SYS_CLKRST_PERI_CLK_CTRL112_REG (0x0070)



HP_SYS_CLKRST_UART1_SCLK_DIV_NUM Configures the integer part of the UART1_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for UART1_SCLK. (R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for UART1_SCLK. (R/W)

HP_SYS_CLKRST_UART2_CLK_SRC_SEL Configures the clock source for UART2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

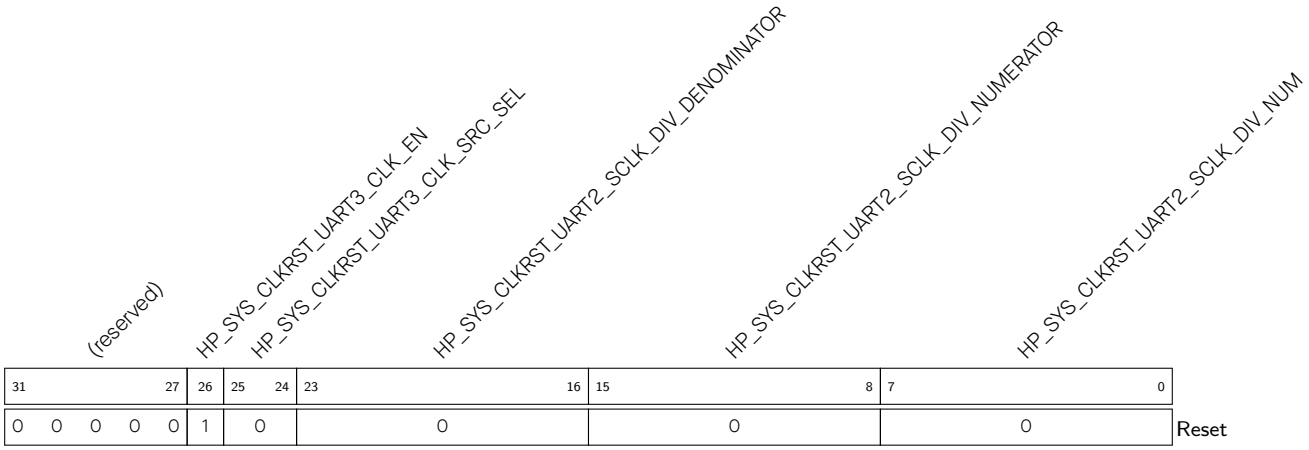
HP_SYS_CLKRST_UART2_CLK_EN Configures whether to enable the UART2_CLK clock.

0: Disable

1: Enable

(R/W)

Register 8.30. HP_SYS_CLKRST_PERI_CLK_CTRL113_REG (0x0074)



HP_SYS_CLKRST_UART2_SCLK_DIV_NUM Configures the integer part of the UART2_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART2_SCLK. (R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART2_SCLK. (R/W)

HP_SYS_CLKRST_UART3_CLK_SRC_SEL Configures the clock source for UART3_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

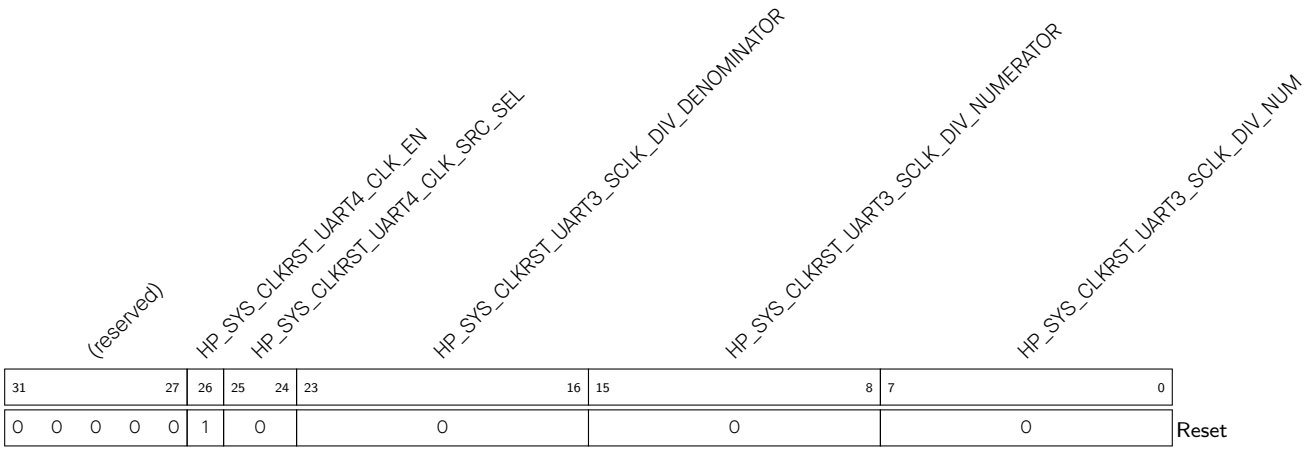
HP_SYS_CLKRST_UART3_CLK_EN Configures whether to enable the UART3_CLK clock.

0: Disable

1: Enable

(R/W)

Register 8.31. HP_SYS_CLKRST_PERI_CLK_CTRL114_REG (0x0078)



HP_SYS_CLKRST_UART3_SCLK_DIV_NUM Configures the integer part of the UART3_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART3_SCLK. (R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART3_SCLK. (R/W)

HP_SYS_CLKRST_UART4_CLK_SRC_SEL Configures the clock source for UART4_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

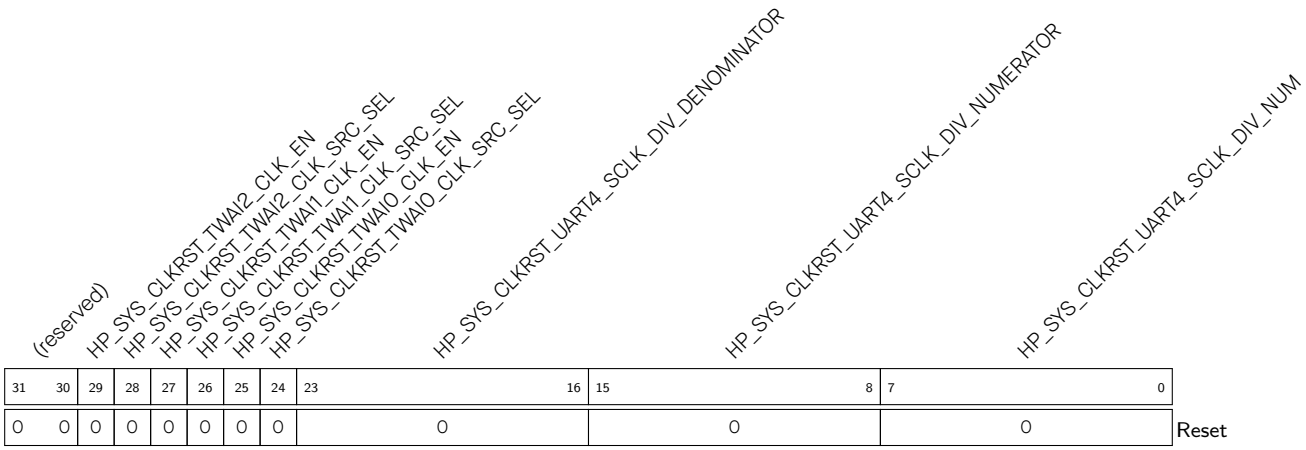
HP_SYS_CLKRST_UART4_CLK_EN Configures whether to enable the UART4_CLK clock.

0: Disable

1: Enable

(R/W)

Register 8.32. HP_SYS_CLKRST_PERI_CLK_CTRL115_REG (0x007C)



HP_SYS_CLKRST_UART4_SCLK_DIV_NUM Configures the integer part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_NUMERATOR Configures the numerator of the fractional part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_DENOMINATOR Configures the denominator of the fractional part of the UART4_SCLK clock divisor. (R/W)

HP_SYS_CLKRST_TWAI0_CLK_SRC_SEL Configures the clock source for TWAI0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI0_CLK_EN Configures whether to enable TWAI0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_SRC_SEL Configures the clock source for TWAI1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_EN Configures whether to enable TWAI1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TWAI2_CLK_SRC_SEL Configures the clock source for TWAI2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

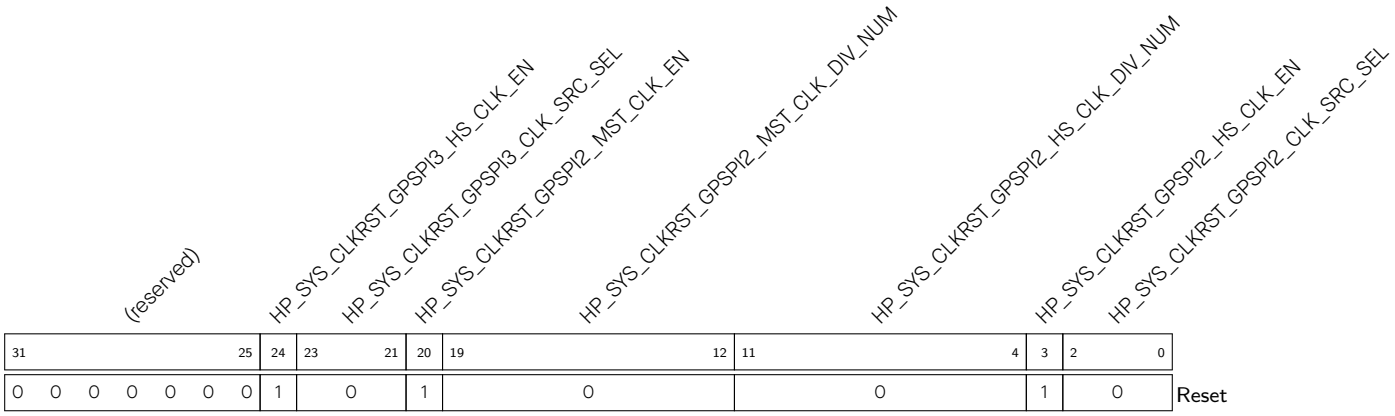
HP_SYS_CLKRST_TWAI2_CLK_EN Configures whether to enable TWAI2_CLK.

0: Disable

1: Enable

(R/W)

Register 8.33. HP_SYS_CLKRST_PERI_CLK_CTRL116_REG (0x0080)



HP_SYS_CLKRST_GPSPI2_CLK_SRC_SEL Configures the clock source for GPSPI2_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: SDIO_PLL0_CLK

3: APLL_CLK

4: SPLL_CLK (480 MHz)

5-7: Invalid

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_EN Configures whether to enable GPSPI2_HS_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_DIV_NUM Configures the integer part of the clock divisor of GPSPI2_HS_CLK. (R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_DIV_NUM Configures the integer part of the clock divisor of GPSPI2_MST_CLK. (R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_EN Configures whether to enable GPSPI2_MST_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_GPSPI3_CLK_SRC_SEL Configures the clock source for GPSPI3_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: SDIO_PLL0_CLK

3: APLL_CLK

4: SPLL_CLK (480 MHz)

5-7: Invalid

(R/W)

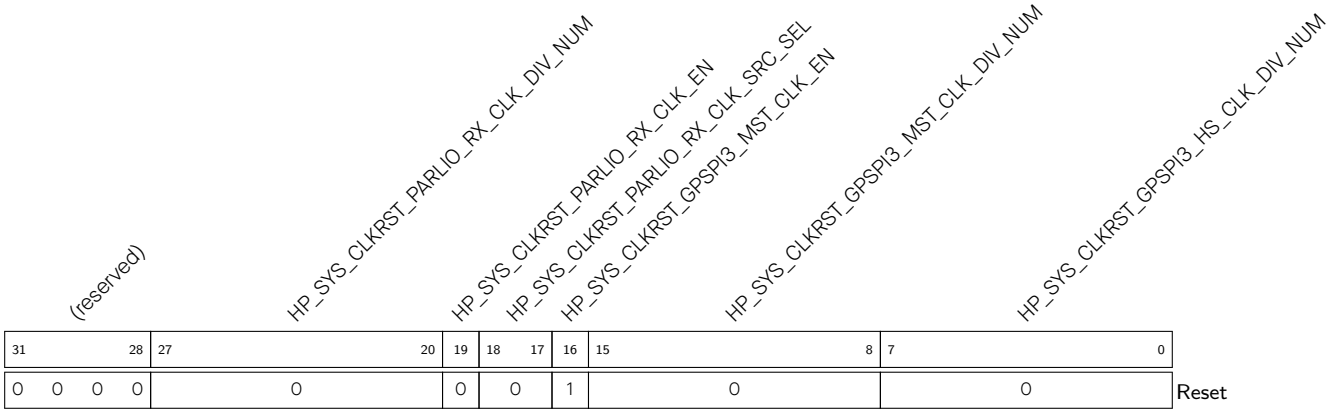
HP_SYS_CLKRST_GPSPI3_HS_CLK_EN Configures whether to enable GPSPI3_HS_CLK.

0: Disable

1: Enable

(R/W)

Register 8.34. HP_SYS_CLKRST_PERI_CLK_CTRL117_REG (0x0084)



HP_SYS_CLKRST_GPSPI3_HS_CLK_DIV_NUM Configures the integer part of the clock divisor for GPSPI3_HS_CLK. (R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_DIV_NUM Configures the integer part of the clock divisor for GPSPI3_MST_CLK. (R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_EN Configures whether to enable GPSPI3_MST_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL Configures the clock source for PARLIO_RX_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160M_CLK

3: PAD_PARLIO_RX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_EN Configures whether to enable PARLIO_RX_CLK.

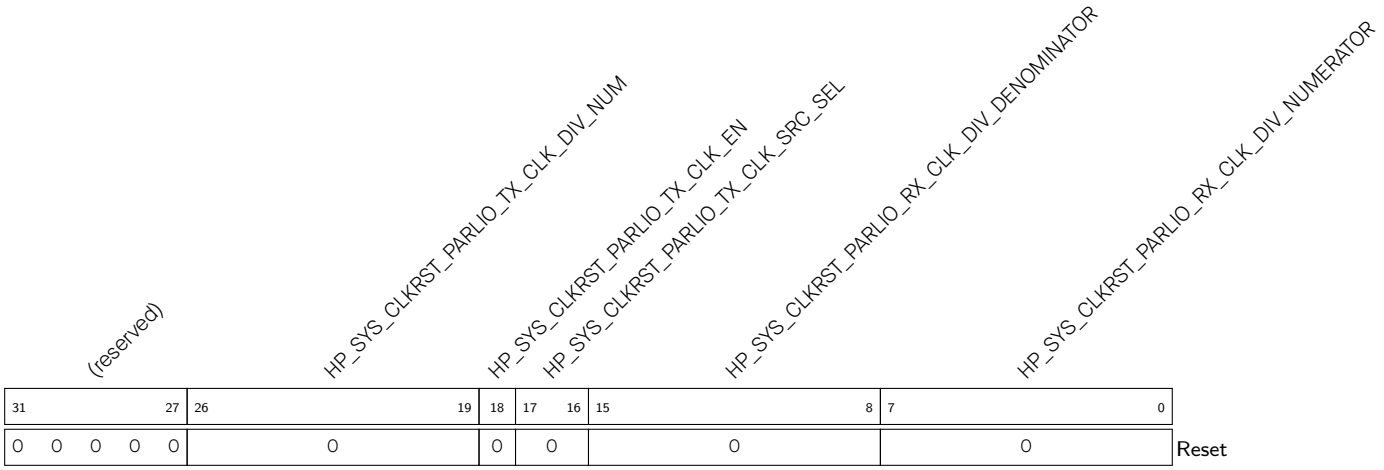
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM Configures the integer part of the clock divisor for PARLIO_RX_CLK. (R/W)

Register 8.35. HP_SYS_CLKRST_PERI_CLK_CTRL118_REG (0x0088)



HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for clock divisor for PARLIO_RX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for clock divisor for PARLIO_RX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL Configures the clock source for PARLIO_TX_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160M_CLK

3: PAD_PARLIO_TX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_EN Configures whether to enable PARLIO_TX_CLK.

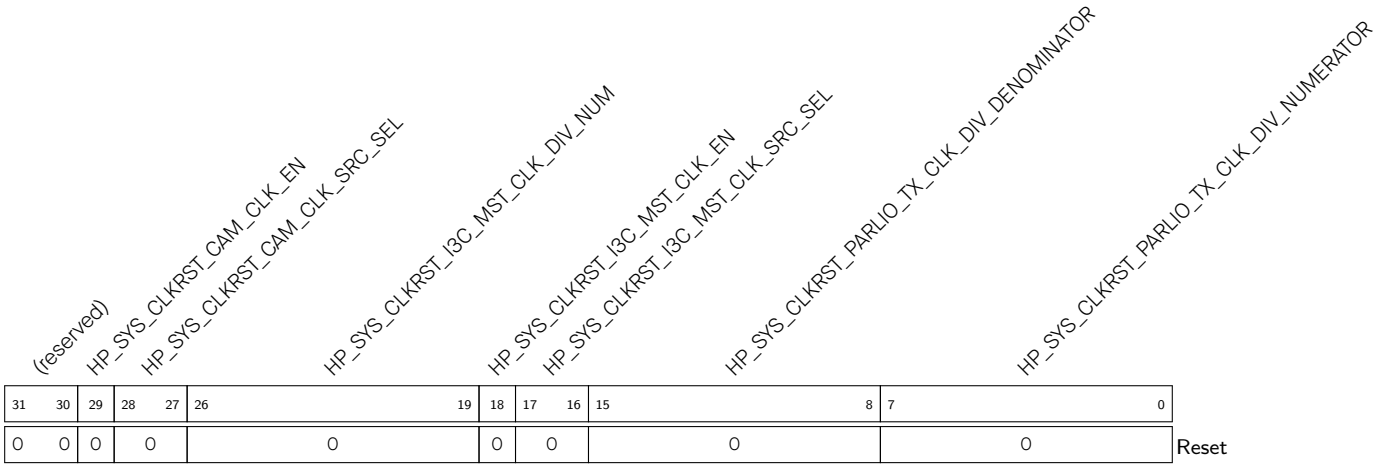
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM Configures the integer part of the clock divisor for PARLIO_TX_CLK. (R/W)

Register 8.36. HP_SYS_CLKRST_PERI_CLK_CTRL119_REG (0x008C)



HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for PARLIO_TX_CLK. (R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for PARLIO_TX_CLK. (R/W)

HP_SYS_CLKRST_I3C_MST_CLK_SRC_SEL Configures the clock source for I3C_MST_CLK.

0: XTAL_CLK
1: PLL_F160M_CLK
2: PLL_F120M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_EN Configures whether to enable I3C_MST_CLK.

0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_DIV_NUM Configures the integer part of the I3C_MST_CLK clock divisor. (R/W)

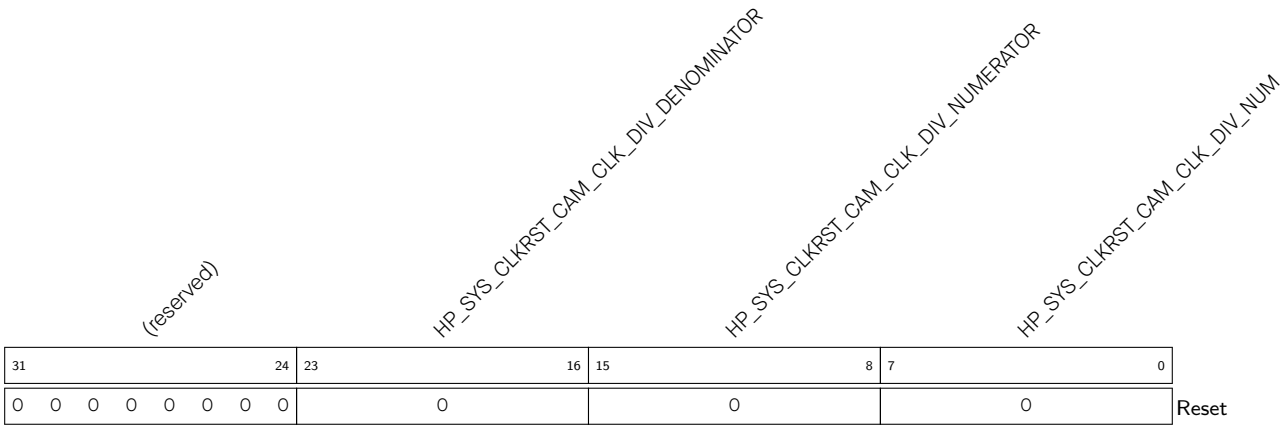
HP_SYS_CLKRST_CAM_CLK_SRC_SEL Configures the clock source for the output clock CAM_CLK.

0: XTAL_CLK
1: PLL_F160M_CLK
2: APLL_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_CAM_CLK_EN Configures whether to enable the output clock CAM_CLK.

0: Disable
1: Enable
(R/W)

Register 8.37. HP_SYS_CLKRST_PERI_CLK_CTRL120_REG (0x0090)

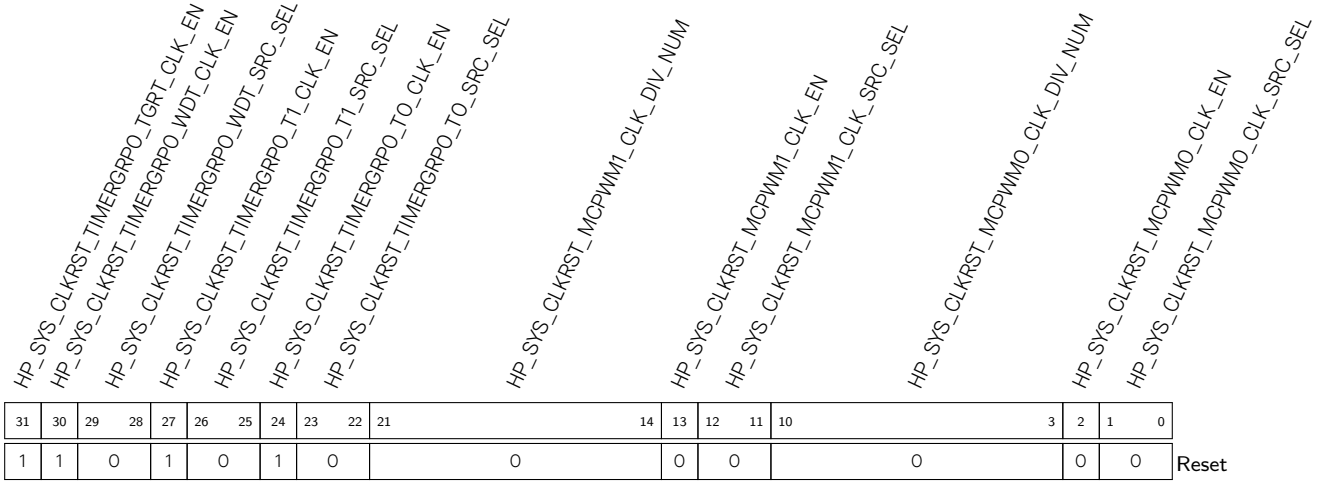


HP_SYS_CLKRST_CAM_CLK_DIV_NUM Configures the integer part of the CAM_CLK clock divisor. (R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for CAM_CLK. (R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for CAM_CLK. (R/W)

Register 8.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)



HP_SYS_CLKRST_MCPWM0_CLK_SRC_SEL Configures the clock source for MCPWM0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_EN Configures whether to enable MCPWM0_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_DIV_NUM Configures the clock divisor of MCPWM0_CLK. (R/W)

HP_SYS_CLKRST_MCPWM1_CLK_SRC_SEL Configures the clock source for MCPWM1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F160_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_EN Configures whether to enable MCPWM1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_DIV_NUM Configures the clock divisor of MCPWM1_CLK. (R/W)

HP_SYS_CLKRST_TIMERGRP0_T0_SRC_SEL Configures the clock source for TIMERGRP0_T0_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

Continued on the next page...

Register 8.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)

Continued from the previous page...

HP_SYS_CLKRST_TIMERGRP0_TO_CLK_EN Configures whether to enable TIMERGRP0_TO_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_T1_SRC_SEL Configures the clock source for TIMERGRP0_T1_CLK.
0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

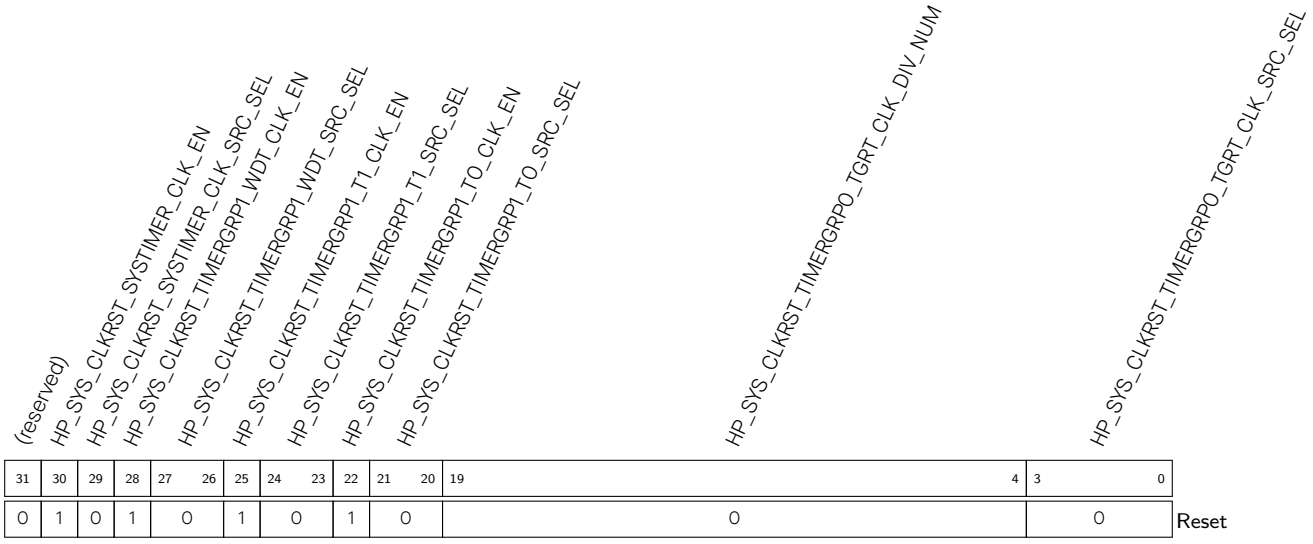
HP_SYS_CLKRST_TIMERGRP0_T1_CLK_EN Configures whether to enable TIMERGRP0_T1_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_SRC_SEL Configures the clock source for TIMERGRP0_WDT_CLK.
0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_CLK_EN Configures whether to enable TIMERGRP0_WDT_CLK.
0: Disable
1: Enable
(R/W)

HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN Configures whether to enable TIMERGRP0_TGRT_CLK.
0: Disable
1: Enable
(R/W)

Register 8.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)



HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL Configures the clock source for TIMERGRP0_TGRT_CLK.

0: MPLL_CLK (500 MHz)
1: SPLL_CLK (480 MHz)
2: CPLL_CLK (400 MHz)
3: APLL_CLK
4: SDIO_PLL0_CLK
5: SDIO_PLL1_CLK
6: SDIO_PLL2_CLK
7: RC_FAST_CLK
8: RC_SLOW_CLK
9: RC32K_CLK
10: XTAL32K_CLK
11: PLL_LP_CLK
12-15: Clock tie 0
(R/W)

HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM Configures the clock divisor of TIMERGRP0_TGRT_CLK. (R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_SRC_SEL Configures the clock source for TIMERGRP1_T0_CLK.

0: XTAL_CLK
1: RC_FAST_CLK
2: PLL_F80M_CLK
3: Invalid
(R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_CLK_EN Configures whether to enable TIMERGRP1_T0_CLK.

0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)

Continued from the previous page...

HP_SYS_CLKRST_TIMERGRP1_T1_SRC_SEL Configures the clock source for TIMERGRP1_T1_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_TIMERGRP1_T1_CLK_EN Configures whether to enable TIMERGRP1_T1_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_SRC_SEL Configures the clock source for TIMERGRP1_WDT_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_CLK_EN Configures whether to enable TIMERGRP1_WDT_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_SYSTIMER_CLK_SRC_SEL Configures the clock source for SYSTIMER_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

(R/W)

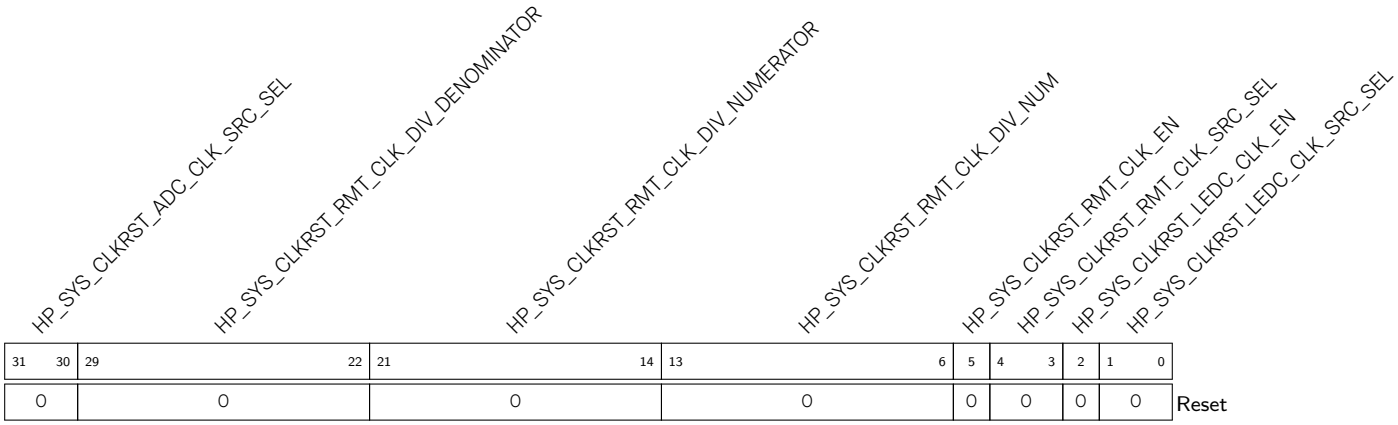
HP_SYS_CLKRST_SYSTIMER_CLK_EN Configures whether to enable SYSTIMER_CLK.

0: Disable

1: Enable

(R/W)

Register 8.40. HP_SYS_CLKRST_PERI_CLK_CTRL22_REG (0x009C)



HP_SYS_CLKRST_LEDC_CLK_SRC_SEL Configures the clock source for LEDC_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_LEDC_CLK_EN Configures whether to enable LEDC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_RMT_CLK_SRC_SEL Configures the clock source for RMT_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

HP_SYS_CLKRST_RMT_CLK_EN Configures whether to enable RMT_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUM Configures the integer part of the RMT_CLK clock divisor. (R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for RMT_CLK. (R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for RMT_CLK. (R/W)

Continued on the next page...

Register 8.40. HP_SYS_CLKRST_PERI_CLK_CTRL22_REG (0x009C)

Continued from the previous page...

HP_SYS_CLKRST_ADC_CLK_SRC_SEL Configures the clock source for HPADC_CLK.

0: XTAL_CLK

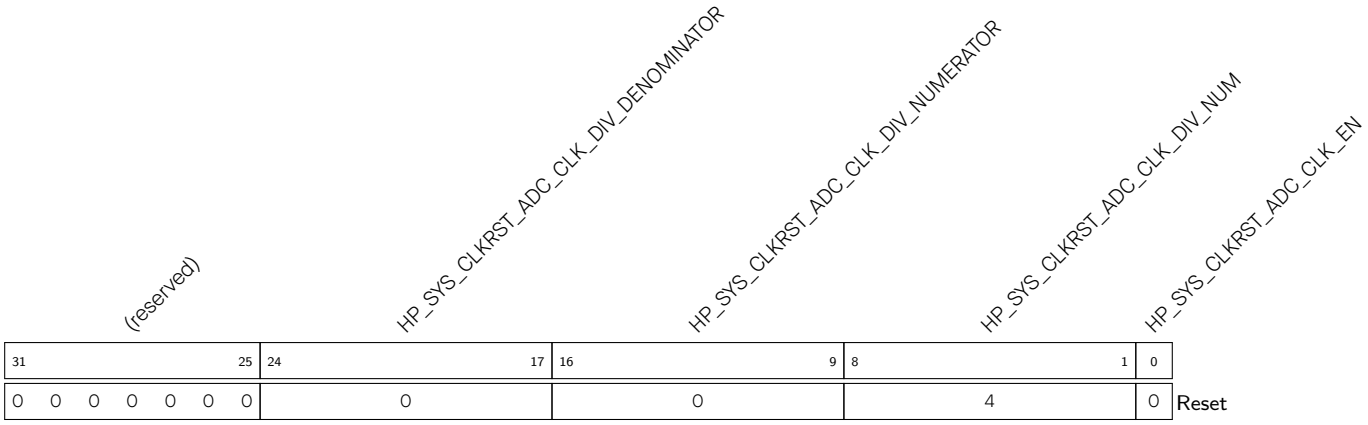
1: RC_FAST_CLK

2: PLL_F80M_CLK

3: Invalid

(R/W)

Register 8.41. HP_SYS_CLKRST_PERI_CLK_CTRL23_REG (0x00A0)



HP_SYS_CLKRST_ADC_CLK_EN Configures whether to enable HPADC_CLK.

0: Disable

1: Enable

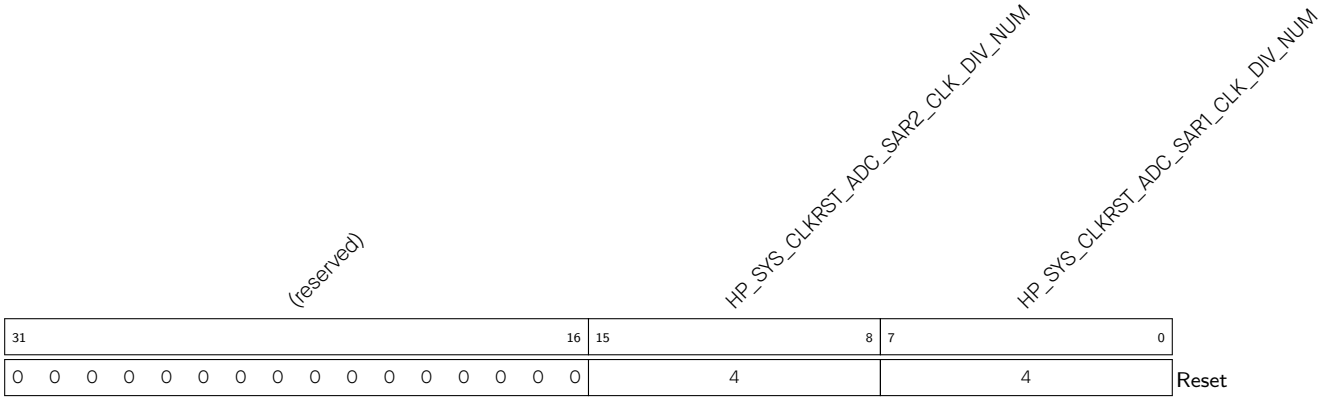
(R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUM Configures the integer part of the HPADC_CLK clock divisor. (R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUMERATOR Configures the numerator of the divisor's fractional part for HPADC_CLK. (R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_DENOMINATOR Configures the denominator of the divisor's fractional part for HPADC_CLK. (R/W)

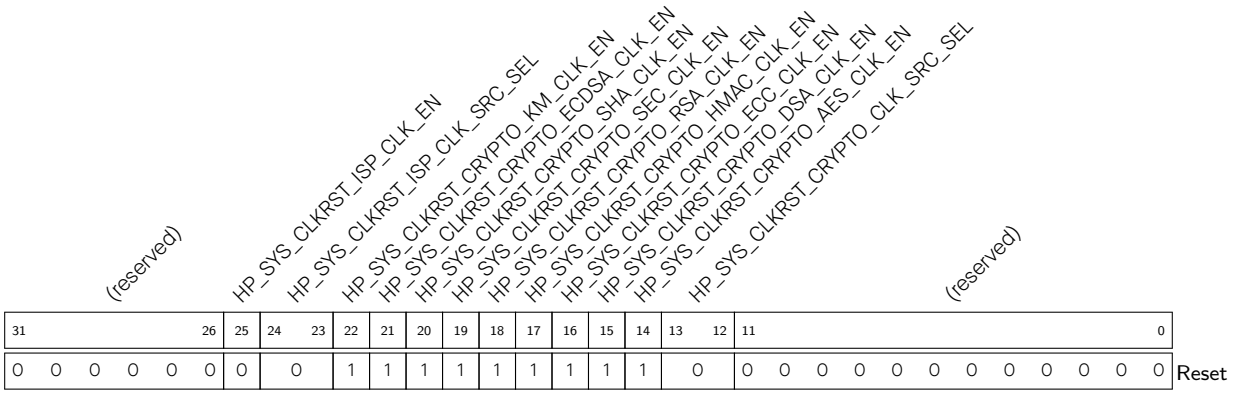
Register 8.42. HP_SYS_CLKRST_PERI_CLK_CTRL24_REG (0x00A4)



HP_SYS_CLKRST_ADC_SAR1_CLK_DIV_NUM Configures the integer part of the clock divisor for ADC_SAR1_CLK. (R/W)

HP_SYS_CLKRST_ADC_SAR2_CLK_DIV_NUM Configures the integer part of the clock divisor for ADC_SAR2_CLK. (R/W)

Register 8.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)



HP_SYS_CLKRST_CRYPTO_CLK_SRC_SEL Configures the clock source for CRYPTO_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_F240M_CLK

3: PLL_F160M_CLK

(R/W)

HP_SYS_CLKRST_CRYPTO_AES_CLK_EN Configures whether to enable AES_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_DSA_CLK_EN Configures whether to enable DSA_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN Configures whether to enable ECC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_HMAC_CLK_EN Configures whether to enable HMAC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_RSA_CLK_EN Configures whether to enable RSA_CLK.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)

Continued from the previous page...

HP_SYS_CLKRST_CRYPTO_SEC_CLK_EN Configures whether to enable SEC_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN Configures whether to enable SHA_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_ECDSA_CLK_EN Configures whether to enable ECDSA_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_CRYPTO_KM_CLK_EN Configures whether to enable KM_CLK.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_ISP_CLK_SRC_SEL Configures the clock source for ISP_CLK.

0: XTAL_CLK

1: PLL_F240M_CLK

2: PLL_F160M_CLK

3: Invalid

(R/W)

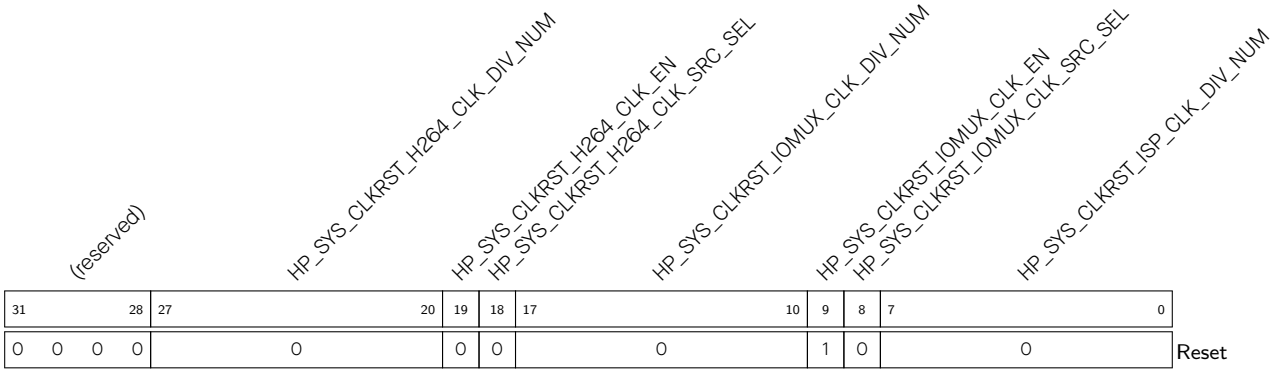
HP_SYS_CLKRST_ISP_CLK_EN Configures whether to enable ISP_CLK.

0: Disable

1: Enable

(R/W)

Register 8.44. HP_SYS_CLKRST_PERI_CLK_CTRL26_REG (0x00AC)



HP_SYS_CLKRST_ISP_CLK_DIV_NUM Configures the clock divisor of ISP_CLK. (R/W)

HP_SYS_CLKRST_IOMUX_CLK_SRC_SEL Configures the clock source for IOMUX_CLK.

0: XTAL_CLK

1: PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_EN Configures whether to enable the IOMUX_CLK clock.

0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_DIV_NUM Configures the clock divisor of IOMUX_CLK. (R/W)

HP_SYS_CLKRST_H264_CLK_SRC_SEL Configures the clock source for H264_CLK.

0: XTAL_CLK

1: PLL_F240M_CLK

(R/W)

HP_SYS_CLKRST_H264_CLK_EN Configures whether to enable the H264_CLK clock.

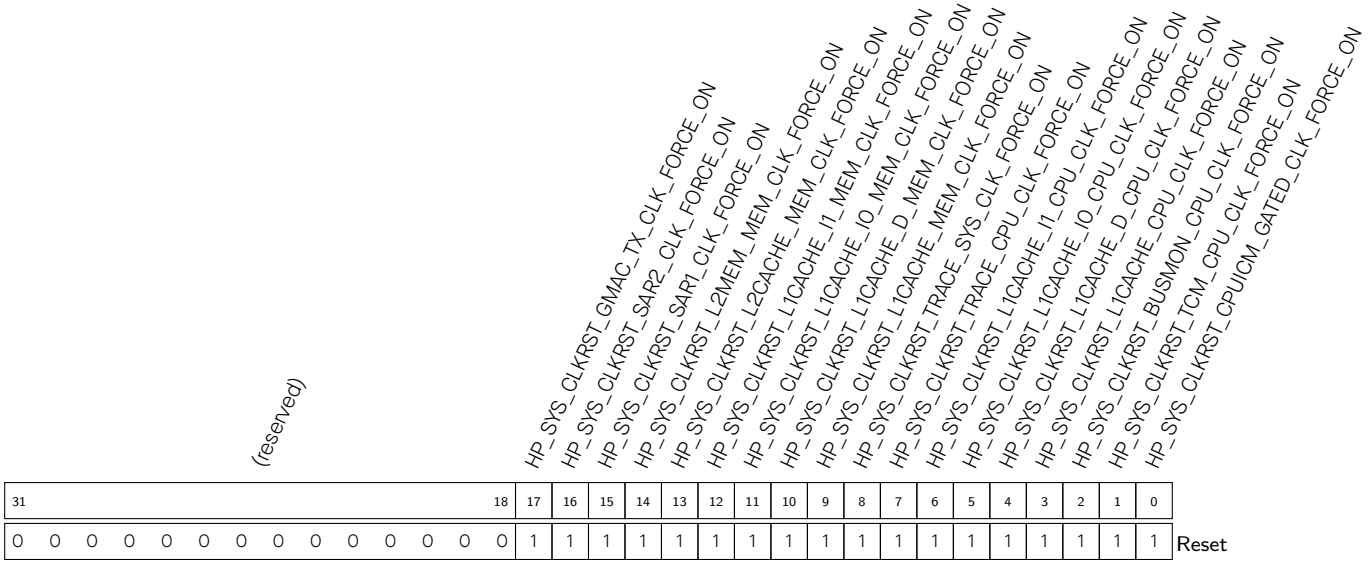
0: Disable

1: Enable

(R/W)

HP_SYS_CLKRST_H264_CLK_DIV_NUM Configures the clock divisor of H264_CLK. (R/W)

Register 8.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)



HP_SYS_CLKRST_CPUICM_GATED_CLK_FORCE_ON Configures the CPUICM_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_TCM_CPU_CLK_FORCE_ON Configures the TCM_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_BUSMON_CPU_CLK_FORCE_ON Configures the BUSMON_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_FORCE_ON Configures the L1CACHE_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_FORCE_ON Configures the L1DCACHE_CPU_CLK clock gating path.

0: Clock gating is enabled

1: Force on the clock, bypassing the clock gating

(R/W)

Continued on the next page...

Register 8.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

Continued from the previous page...

HP_SYS_CLKRST_L1CACHE_I0_CPU_CLK_FORCE_ON Configures the L1I0CACHE_CPU_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_FORCE_ON Configures the L1I1CACHE_CPU_CLK clock gating path.
(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_FORCE_ON Configures the TRACE_CPU_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_FORCE_ON Configures the TRACE_SYS_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_FORCE_ON Configures the L1CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_FORCE_ON Configures the L1DCACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_FORCE_ON Configures the L1I0CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_FORCE_ON Configures the L1I1CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

Continued on the next page...

Register 8.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

Continued from the previous page...

HP_SYS_CLKRST_L2CACHE_MEM_CLK_FORCE_ON Configures the L2CACHE_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

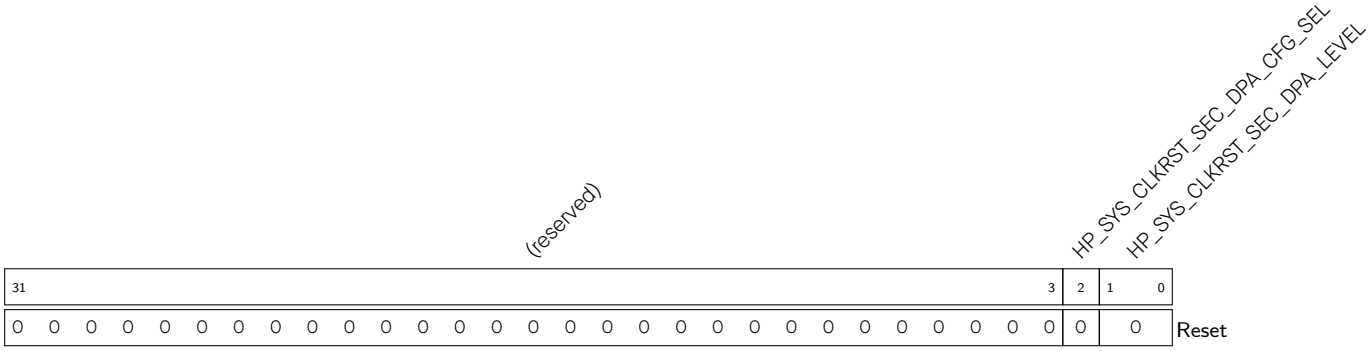
HP_SYS_CLKRST_L2MEM_MEM_CLK_FORCE_ON Configures the L2MEM_MEM_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_SAR1_CLK_FORCE_ON Configures the SAR1_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_SAR2_CLK_FORCE_ON Configures the SAR2_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

HP_SYS_CLKRST_GMAC_TX_CLK_FORCE_ON Configures the GMAC_TX_CLK clock gating path.
0: Clock gating is enabled
1: Force on the clock, bypassing the clock gating
(R/W)

Register 8.46. HP_SYS_CLKRST_DPA_CTRL0_REG (0x00B8)



HP_SYS_CLKRST_SEC_DPA_LEVEL Configures the security level of CRYPTO_CLK against DPA attacks.

0: Low security level

1: High security level

(R/W)

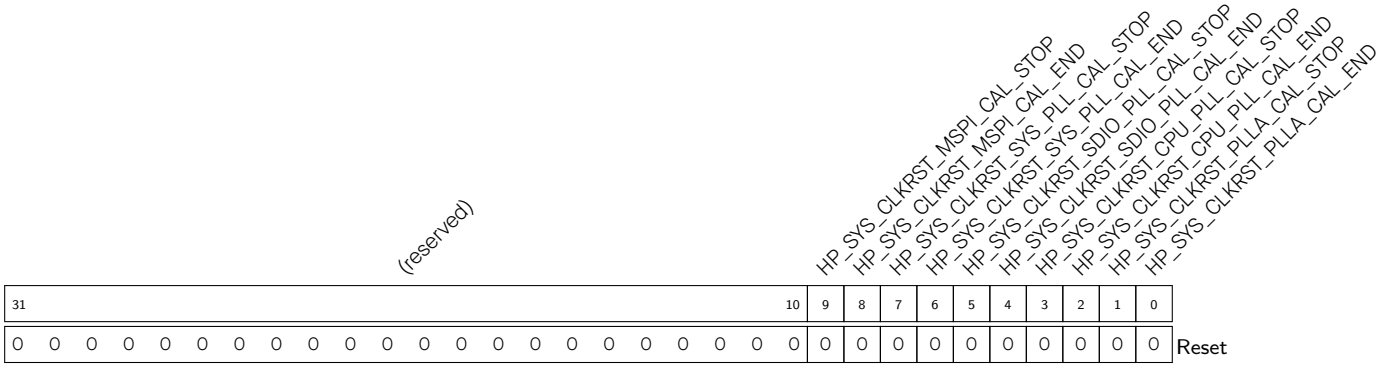
HP_SYS_CLKRST_SEC_DPA_CFG_SEL Configures the source of the security level of CRYPTO_CLK against DPA attacks.

0: The security level is determined by EFUSE_SEC_DPA_LEVEL

1: The security level is determined by HP_SYS_CLKRST_SEC_DPA_LEVEL

(R/W)

Register 8.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)



HP_SYS_CLKRST_PLLA_CAL_END Represents whether the APLL_CLK calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_PLLA_CAL_STOP Configures whether to start or stop the APLL_CLK calibration.

0: Start calibration

1: Stop calibration

(R/W)

HP_SYS_CLKRST_CPU_PLL_CAL_END Represents whether the CPLL_CLK calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_CPU_PLL_CAL_STOP Configures whether to start or stop the CPLL_CLK calibration.

0: Start calibration

1: Stop calibration

(R/W)

HP_SYS_CLKRST_SDIO_PLL_CAL_END Represents whether the SDIO_PLL calibration is finished.

0: Calibration in progress

1: Calibration done

(RO)

HP_SYS_CLKRST_SDIO_PLL_CAL_STOP Configures whether to start or stop the SDIO_PLL calibration.

0: Start calibration

1: Stop calibration

(R/W)

Continued on the next page...

Register 8.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)

Continued from the previous page...

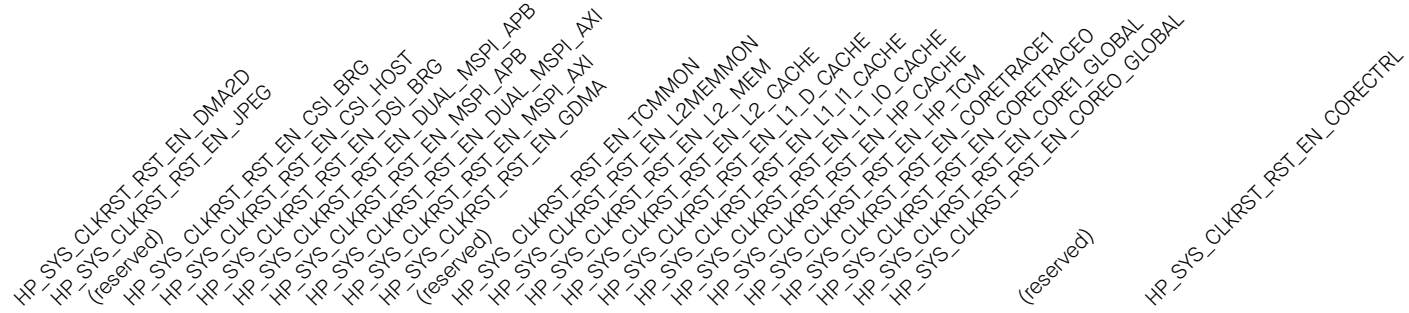
HP_SYS_CLKRST_SYS_PLL_CAL_END Represents whether the SPLL_CLK calibration is finished.
0: Calibration in progress
1: Calibration done
(RO)

HP_SYS_CLKRST_SYS_PLL_CAL_STOP Configures whether to start or stop the SPLL_CLK calibration.
0: Start calibration
1: Stop calibration
(R/W)

HP_SYS_CLKRST_MSPI_CAL_END Represents whether the MPLL_CLK calibration is finished.
0: Calibration in progress
1: Calibration done
(RO)

HP_SYS_CLKRST_MSPI_CAL_STOP Configures whether to start or stop the MPLL_CLK calibration.
0: Start calibration
1: Stop calibration
(R/W)

Register 8.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)



HP_SYS_CLKRST_RST_EN_CORECTRL Configures whether to reset CORECTRL.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_COREO_GLOBAL Configures whether to reset HP CPU0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL Configures whether to reset HP CPU1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE0 Configures whether to reset RISC-V Trace Encoder0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE1 Configures whether to reset RISC-V Trace Encoder1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_HP_TCM Configures whether to reset TCM.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_HP_CACHE Configures whether to reset L1 I-Cache0, L1 I-Cache1, L1 D-Cache, L2 Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I0_CACHE Configures whether to reset L1 I-Cache0.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I1_CACHE Configures whether to reset L1 I-Cache1.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L1_D_CACHE Configures whether to reset L1 D-Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2_CACHE Configures whether to reset L2 Cache.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2_MEM Configures whether to reset L2 MEM.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_L2MEMMON Configures whether to reset L2 MEM monitor.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_TCMMON Configures whether to reset TCM monitor.
0: Release from reset
1: Reset
(R/W)

HP_SYS_CLKRST_RST_EN_GDMA Configures whether to reset GDMA.
0: Release from reset
1: Reset
(R/W)

Continued on the next page...

Register 8.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_MSPI_AXI Configures whether to reset FLASH MSPI AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_AXI Configures whether to reset PSRAM MSPI AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_MSPI_APB Configures whether to reset FLASH MSPI APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_APB Configures whether to reset PSRAM MSPI APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DSI_BRG Configures whether to reset DSI Bridge.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_HOST Configures whether to reset CSI Host.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_BRG Configures whether to reset CSI Bridge and ISP.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_JPEG Configures whether to reset JPEG.

0: Release from reset

1: Reset

(R/W)

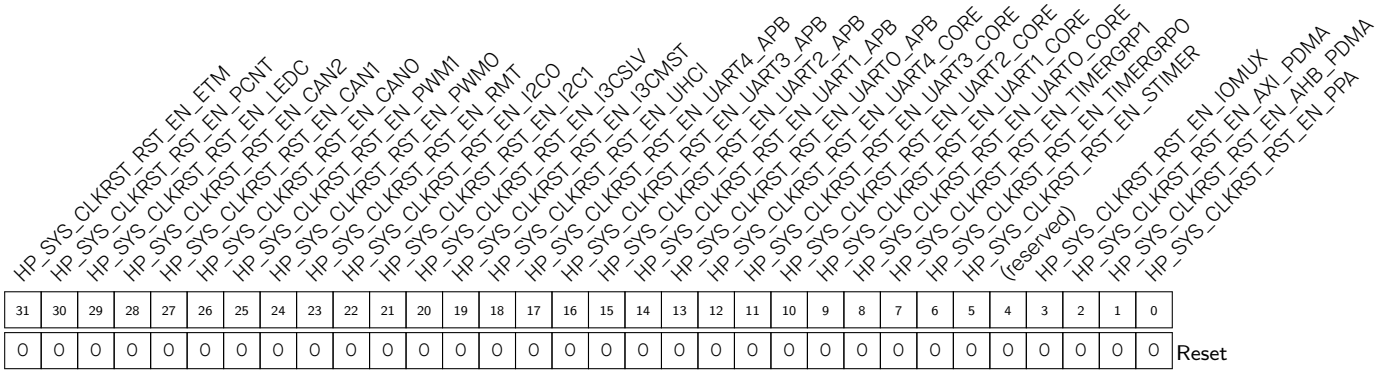
HP_SYS_CLKRST_RST_EN_DMA2D Configures whether to reset 2DDMA.

0: Release from reset

1: Reset

(R/W)

Register 8.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

HP_SYS_CLKRST_RST_EN_PPA Configures whether to reset PPA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_AHB_PDMA Configures whether to reset GDMA-AHB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_AXI_PDMA Configures whether to reset GDMA-AXI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_IOMUX Configures whether to reset IO MUX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_STIMER Configures whether to reset the System Timer.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP0 Configures whether to reset Timer Group 0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP1 Configures whether to reset Timer Group 1.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_UART0_CORE Configures whether to reset UART0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_CORE Configures whether to reset UART1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_CORE Configures whether to reset UART2.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART3_CORE Configures whether to reset UART3.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART4_CORE Configures whether to reset UART4.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART0_APB Configures whether to reset UART0 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_APB Configures whether to reset UART1 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_APB Configures whether to reset UART2 APB.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_UART3_APB Configures whether to reset UART3 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UART4_APB Configures whether to reset UART4 APB.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_UHCI Configures whether to reset UHCI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I3CMST Configures whether to reset I3C Master.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I3CSLV Configures whether to reset I3C Slave.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2C1 Configures whether to reset I2C1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2C0 Configures whether to reset I2C0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_RMT Configures whether to reset RMT.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PWM0 Configures whether to reset MCPMW0.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_PWM1 Configures whether to reset MCPWM1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN0 Configures whether to reset TWAI0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN1 Configures whether to reset TWAI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CAN2 Configures whether to reset TWAI.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_LEDC Configures whether to reset LEDC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PCNT Configures whether to reset PCNT.

0: Release from reset

1: Reset

(R/W)

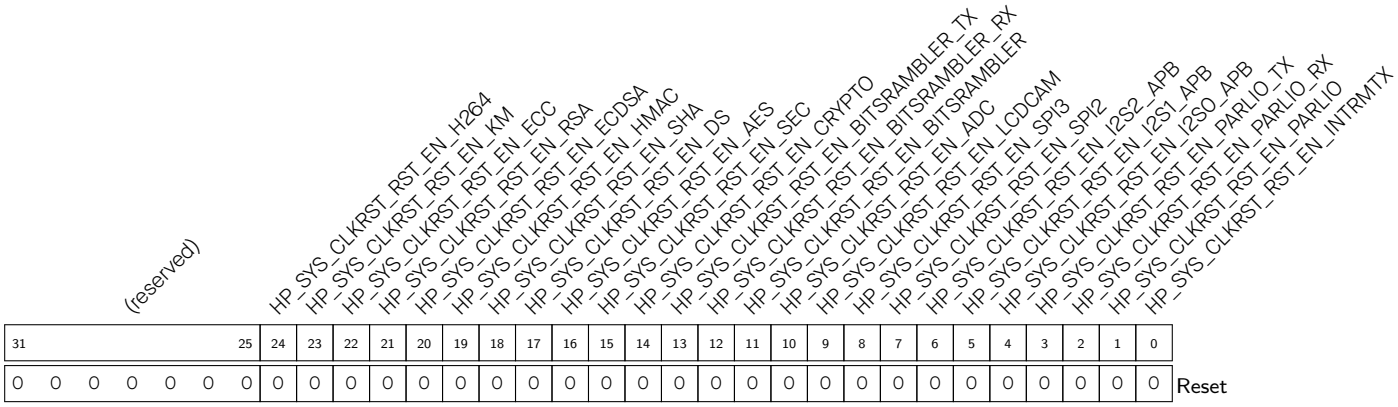
HP_SYS_CLKRST_RST_EN_ETM Configures whether to reset ETM.

0: Release from reset

1: Reset

(R/W)

Register 8.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)



HP_SYS_CLKRST_RST_EN_INTRMTX Configures whether to reset the Interrupt Matrix.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO Configures whether to reset PARLIO.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_RX Configures whether to reset PARLIO RX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_TX Configures whether to reset PARLIO TX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S0_APB Configures whether to reset I2S0.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S1_APB Configures whether to reset I2S1.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_I2S2_APB Configures whether to reset I2S2.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_SPI2 Configures whether to reset SPI2.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SPI3 Configures whether to reset SPI3.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_LCDCAM Configures whether to reset the LCD_CAM.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ADC Configures whether to reset ADC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER Configures whether to reset the Bit Scrambler.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_RX Configures whether to reset the Bit Scrambler RX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_TX Configures whether to reset the Bit Scrambler TX.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_CRYPTO Configures whether to reset the crypto modules.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SEC Configures whether to reset SEC.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

Continued from the previous page...

HP_SYS_CLKRST_RST_EN_AES Configures whether to reset AES.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_DS Configures whether to reset DS.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_SHA Configures whether to reset SHA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_HMAC Configures whether to reset HMAC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ECDSA Configures whether to reset ECDSA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_RSA Configures whether to reset RSA.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_ECC Configures whether to reset ECC.

0: Release from reset

1: Reset

(R/W)

HP_SYS_CLKRST_RST_EN_KM Configures whether to reset the Key Manager.

0: Release from reset

1: Reset

(R/W)

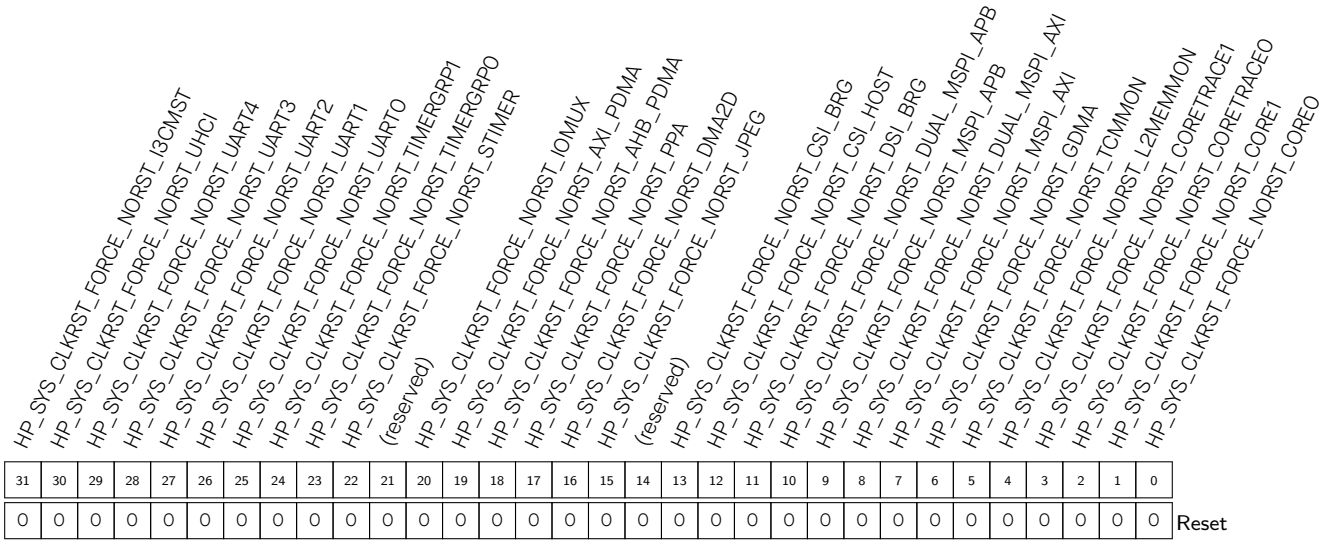
HP_SYS_CLKRST_RST_EN_H264 Configures whether to reset H264.

0: Release from reset

1: Reset

(R/W)

Register 8.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)



HP_SYS_CLKRST_FORCE_NORST_CORE0 Configures whether HP CPU0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORE1 Configures whether HP CPU1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE0 Configures whether RISC-V Trace Encoder 0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE1 Configures whether RISC-V Trace Encoder 1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_L2MEMMON Configures whether L2 MEM monitor can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TCMMON Configures whether TCM monitor can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_GDMA Configures whether DW-GDMA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_MSPI_AXI Configures whether FLASH MSPI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_AXI Configures whether PSRAM MSPI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_MSPI_APB Configures whether FLASH MSPI APB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_APB Configures whether PSRAM MSPI APB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DSI_BRG Configures whether DSI Bridge can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_HOST Configures whether CSI Host can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_BRG Configures whether CSI Bridge, ISP can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_JPEG Configures whether JPEG can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DMA2D Configures whether 2DDMA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PPA Configures whether PPA can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AHB_PDMA Configures whether GDMA-AHB can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AXI_PDMA Configures whether GDMA-AXI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_IOMUX Configures whether IO MUX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_STIMER Configures whether the System Timer can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP0 Configures whether Timer Group 0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP1 Configures whether Timer Group 1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_UART0 Configures whether UART0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART1 Configures whether UART1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART2 Configures whether UART2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART3 Configures whether UART3 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART4 Configures whether UART4 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UHCI Configures whether UHCI can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

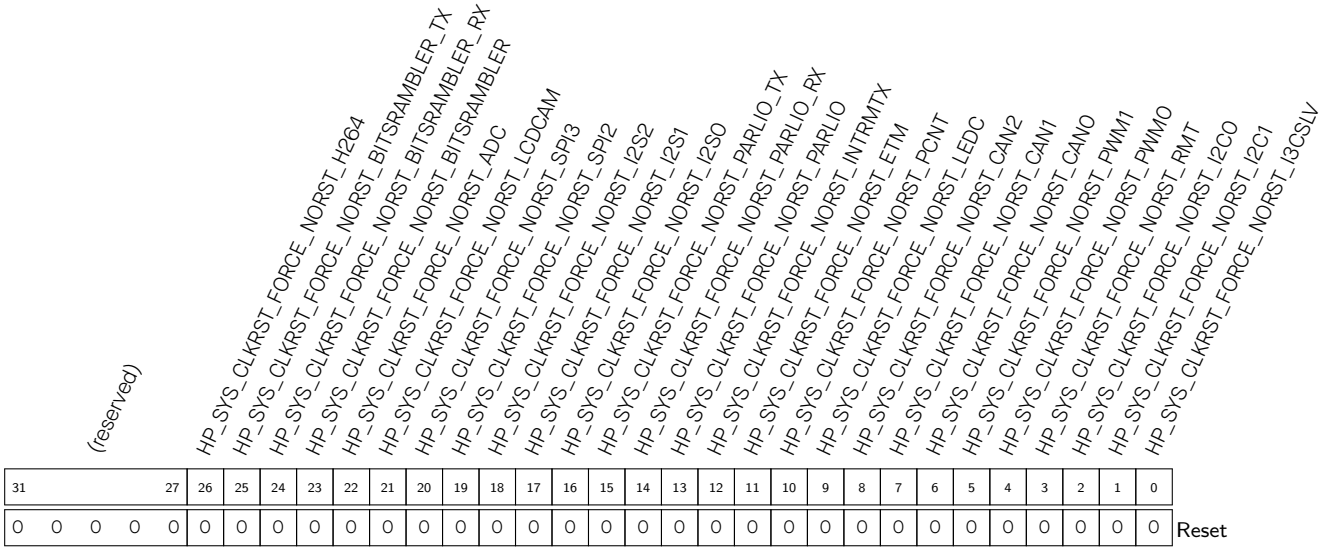
HP_SYS_CLKRST_FORCE_NORST_I3CMST Configures whether I3C Master can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Register 8.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)



HP_SYS_CLKRST_FORCE_NORST_I3CSLV Configures whether I3C Slave can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C1 Configures whether I2C1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C0 Configures whether I2C0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_RMT Configures whether RMT can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM0 Configures whether MCPWM0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM1 Configures whether MCPWM1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN0 Configures whether TWAI0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_CAN1 Configures whether TWAI1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN2 Configures whether TWAI2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LEDC Configures whether LEDC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PCNT Configures whether PCNT can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ETM Configures whether ETM can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_INTRMTX Configures whether the interrupt matrix can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO Configures whether PARLIO can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_RX Configures whether PARLIO RX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_TX Configures whether PARLIO TX can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

HP_SYS_CLKRST_FORCE_NORST_I2S0 Configures whether I2S0 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S1 Configures whether I2S1 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S2 Configures whether I2S2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI2 Configures whether SPI2 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI3 Configures whether SPI3 can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LCDCAM Configures whether the LCD_CAM can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ADC Configures whether ADC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Continued on the next page...

Register 8.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

Continued from the previous page...

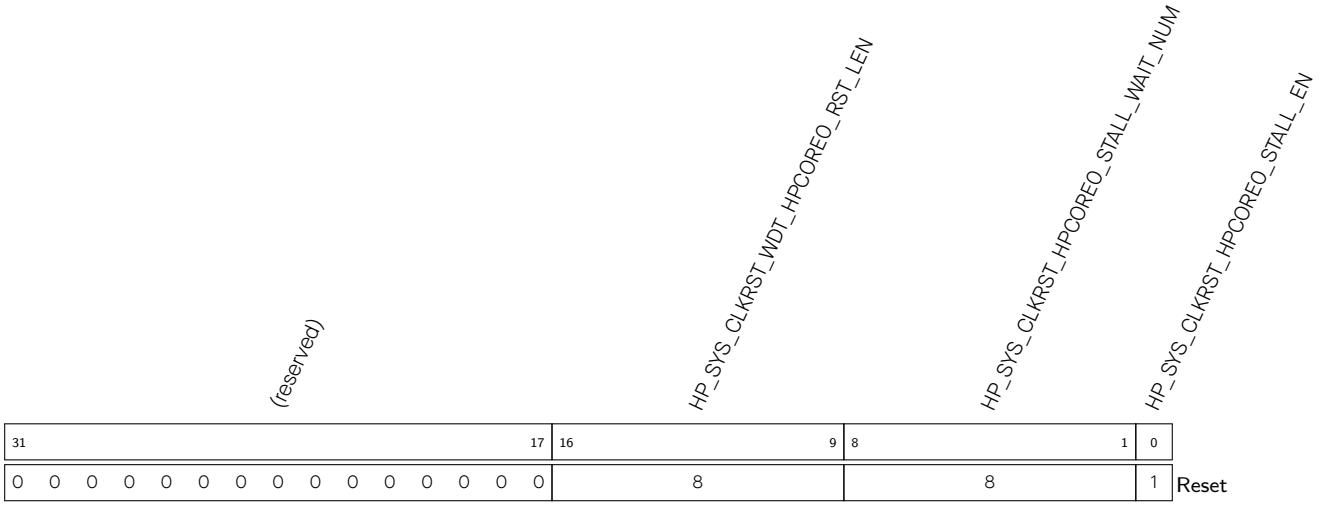
HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER Configures whether the Bit Scrambler can be reset.
0: Can be reset
1: Force not to be reset
(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_RX Configures whether the Bit Scrambler RX can be reset.
0: Can be reset
1: Force not to be reset
(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_TX Configures whether the Bit Scrambler TX can be reset.
0: Can be reset
1: Force not to be reset
(R/W)

HP_SYS_CLKRST_FORCE_NORST_H264 Configures whether H264 can be reset.
0: Can be reset
1: Force not to be reset
(R/W)

Register 8.53. HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG (0x00D4)



HP_SYS_CLKRST_HPCORE0_STALL_EN Configures the behavior when MWDT triggers HP CPU0 reset.

0: Immediately reset HP CPU0

1: Stall HP CPU0 first

(R/W)

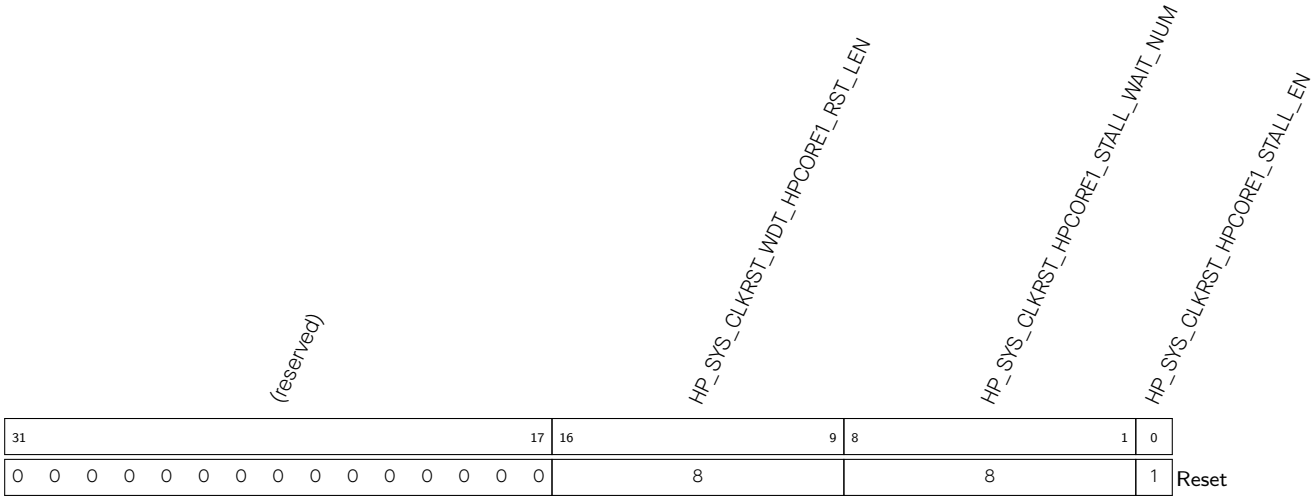
HP_SYS_CLKRST_HPCORE0_STALL_WAIT_NUM Configures the duration of HP CPU0 stall when MWDT triggers HP CPU0 reset and HP_SYS_CLKRST_HPCORE0_STALL_EN is set to 1.

Measurement unit: Clock cycles. (R/W)

HP_SYS_CLKRST_WDT_HPCORE0_RST_LEN Configures the duration of the reset when MWDT triggers HP CPU0 reset.

Measurement unit: Clock cycles. (R/W)

Register 8.54. HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG (0x00D8)



HP_SYS_CLKRST_HPCORE1_STALL_EN Configures the behavior when MWDT triggers HP CPU1 reset.

0: Immediately reset HP CPU1

1: Stall HP CPU1 first

(R/W)

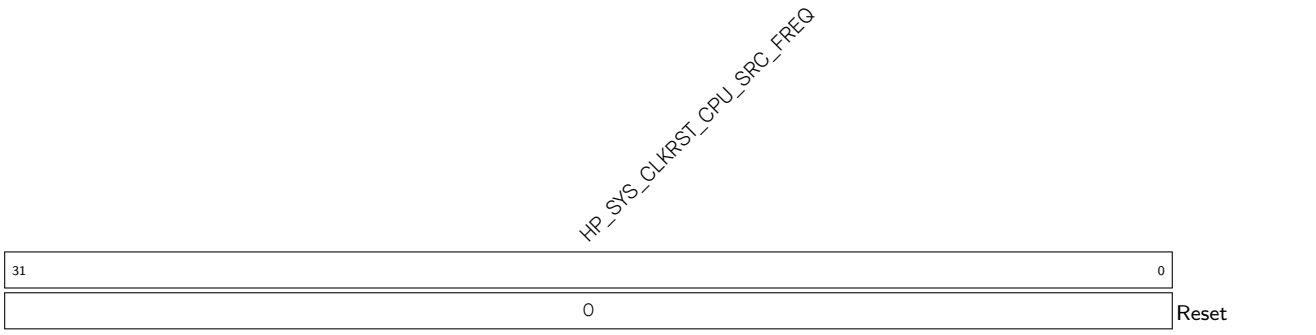
HP_SYS_CLKRST_HPCORE1_STALL_WAIT_NUM Configures the duration of HP CPU1 stall when MWDT triggers HP CPU1 reset and HP_SYS_CLKRST_HPCORE1_STALL_EN is set to 1.

Measurement unit: Clock cycles. (R/W)

HP_SYS_CLKRST_WDT_HPCORE1_RST_LEN Configures the duration of the reset when MWDT triggers HP CPU1 reset.

Measurement unit: Clock cycles. (R/W)

Register 8.55. HP_SYS_CLKRST_CPU_SRC_FREQ0_REG (0x00DC)



HP_SYS_CLKRST_CPU_SRC_FREQ Represents the CPU source clock frequency.

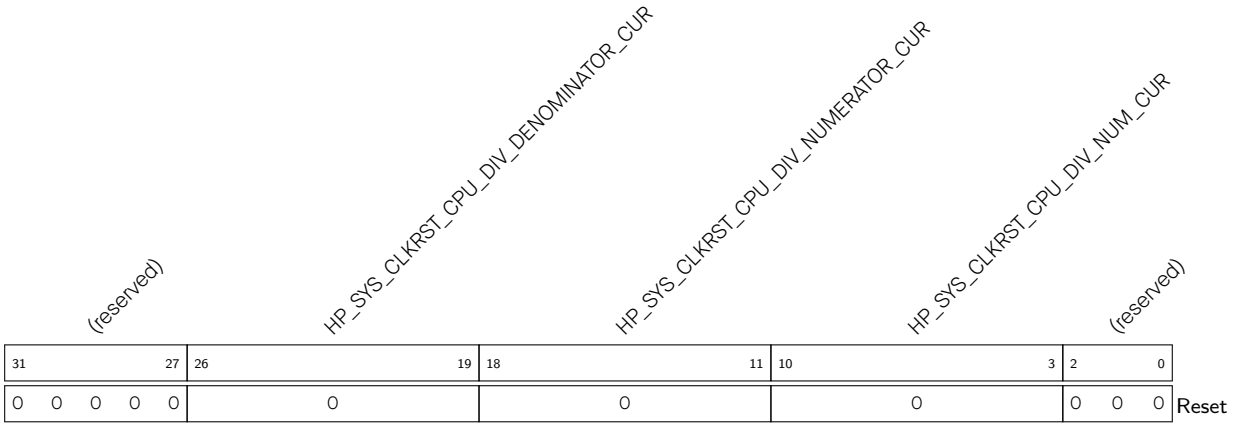
80: RC_FAST_CLK

160: XTAL_CLK

1600: CPLL_CLK

(RO)

Register 8.56. HP_SYS_CLKRST_CPU_CLK_STATUS0_REG (0x00E0)

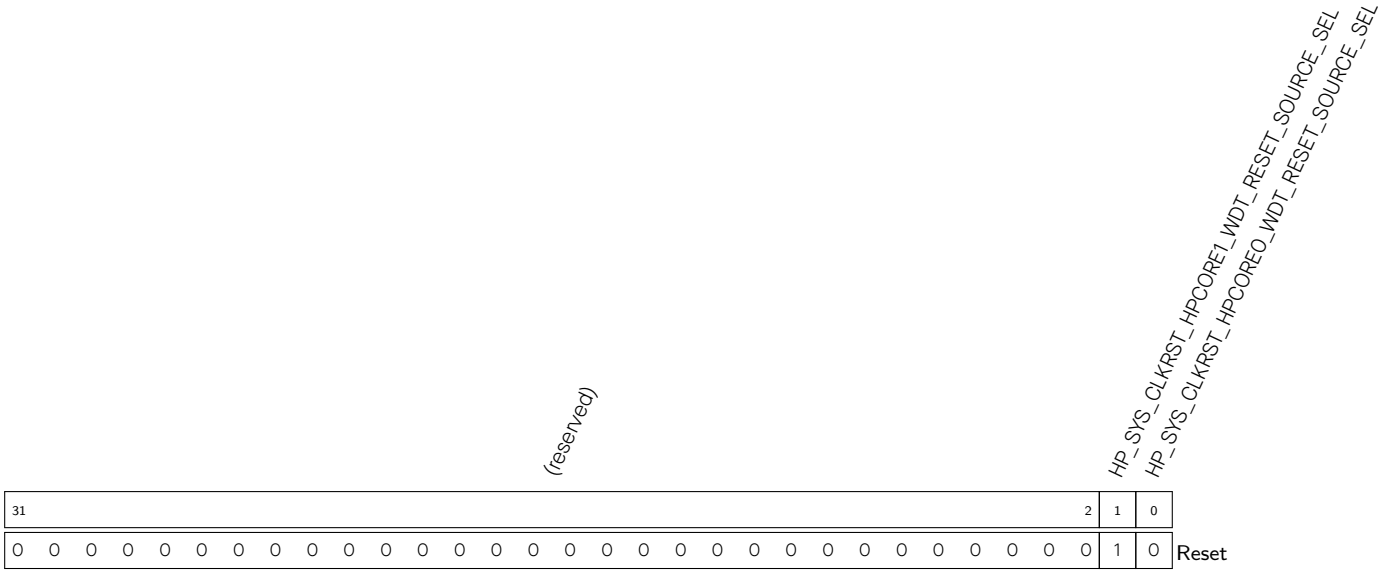


HP_SYS_CLKRST_CPU_DIV_NUM_CUR Represents the integer part of the current CPU_CLK clock divisor. (RO)

HP_SYS_CLKRST_CPU_DIV_NUMERATOR_CUR Represents the current numerator of the divisor's fractional part for the CPU_CLK. (RO)

HP_SYS_CLKRST_CPU_DIV_DENOMINATOR_CUR Represents the current denominator of the divisor's fractional part for the CPU_CLK. (RO)

Register 8.57. HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG (0x00EC)



HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL Configures the MWDT used to trigger the HP CPU0 reset.

0: MWDT0

1: MWDT1

(R/W)

HP_SYS_CLKRST_HPCORE1_WDT_RESET_SOURCE_SEL Configures the MWDT used to trigger the HP CPU1 reset.

0: MWDT0

1: MWDT1

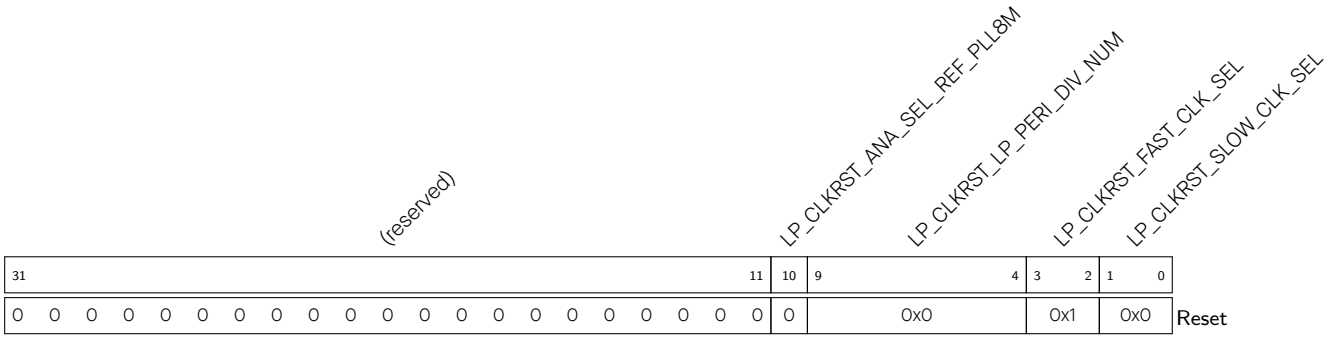
(R/W)

8.5.2 LP Always on Clock and Reset (LP_CLKRST) Registers

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

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

Register 8.58. LP_CLKRST_LP_CLK_CONF_REG (0x0000)



LP_CLKRST_SLOW_CLK_SEL Configures the clock source of LP_SLOW_CLK.

0: RC_SLOW_CLK

1: XTAL32K_CLK

2: RC32K_CLK

3: OSC_SLOW_CLK

(R/W)

LP_CLKRST_FAST_CLK_SEL Configures the clock source of LP_FAST_CLK.

0: XTAL_CLK

1: RC_FAST_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LP_CLKRST_LP_PERI_DIV_NUM Configures the clock divisor for LP_PERI_CLK. (R/W)

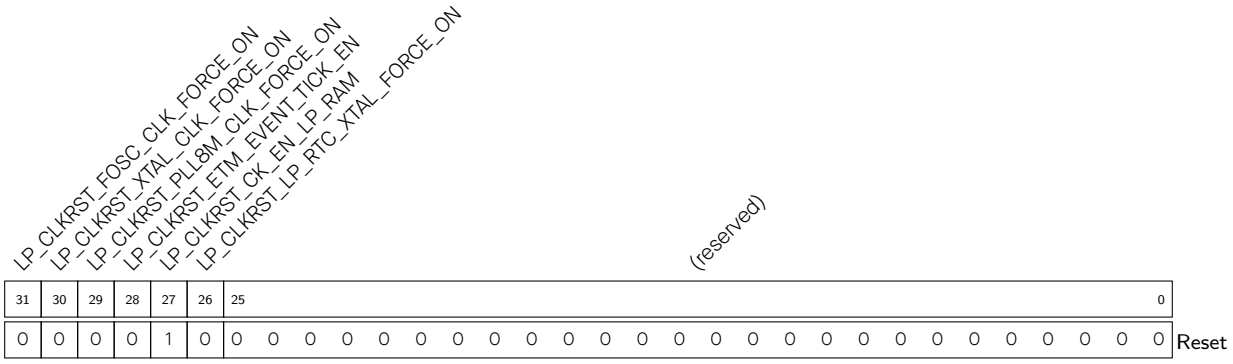
LP_CLKRST_ANA_SEL_REF_PLL8M Configures the reference clock of PLL_LP_CLK.

0: RC32K_CLK

1: XTAL32K_CLK

(R/W)

Register 8.59. LP_CLKRST_LP_CLK_EN_REG (0x0008)



LP_CLKRST_LP_RTC_XTAL_FORCE_ON Configures whether to force on the XTAL_CLK control of LP CALI.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LP_CLKRST_CK_EN_LP_RAM Configurewhether to enable the clock of LP RAM.

0: Disable

1: Enable

(R/W)

LP_CLKRST_ETM_EVENT_TICK_EN Configures whether to enable the ETM event timer.

0: Disable

1: Enable

(R/W)

LP_CLKRST_PLL8M_CLK_FORCE_ON Configures whether to force on PLL_LP_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LP_CLKRST_XTAL_CLK_FORCE_ON Configures whether to force on XTAL_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

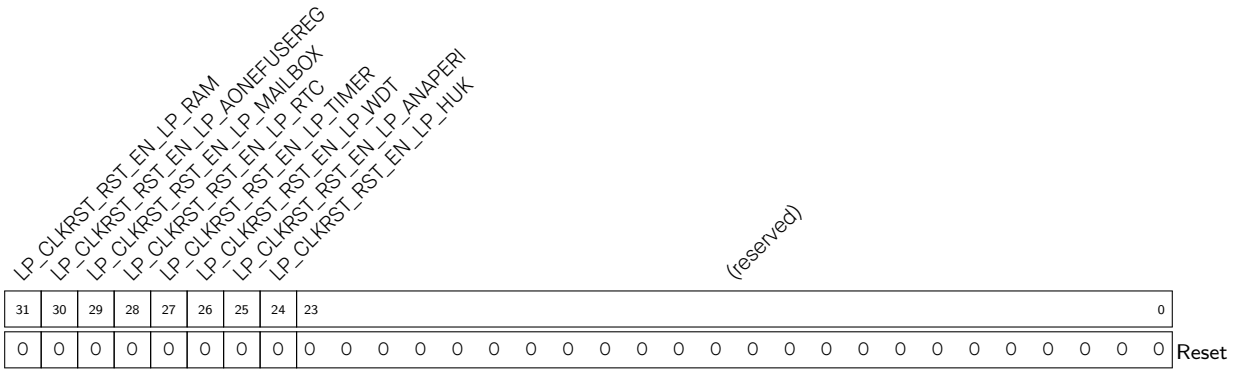
LP_CLKRST_FOSC_CLK_FORCE_ON Configures whether to force on RC_FAST_CLK as LP_FAST_CLK source.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

Register 8.60. LP_CLKRST_LP_RST_EN_REG (0x000C)



LP_CLKRST_RST_EN_LP_HUK Configures whether to reset LP HUK.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_ANAPERI Configures whether to reset LP analog peripherals.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_WDT Configures whether to reset RWDT.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_TIMER Configures whether to reset LP timer.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_RTC Configures whether to reset the LP clock calibration module.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_MAILBOX Configures whether to reset LP Mailbox.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_LP_AONEFUSEREG Configures whether to reset LP eFuse.

0: Release from reset

1: Reset

(R/W)

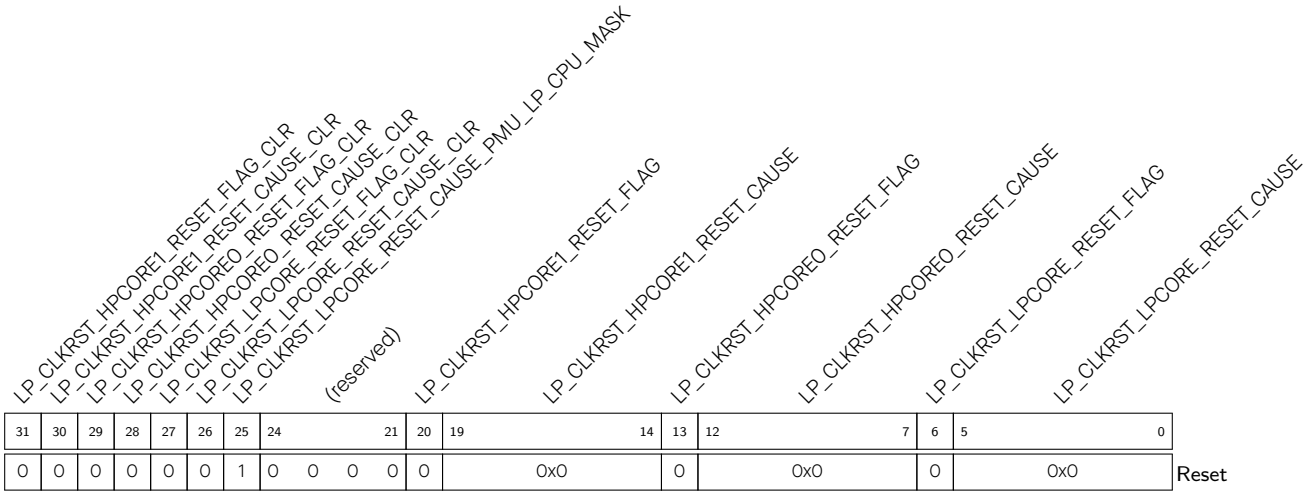
LP_CLKRST_RST_EN_LP_RAM Configures whether to reset LP RAM.

0: Release from reset

1: Reset

(R/W)

Register 8.61. LP_CLKRST_RESET_CAUSE_REG (0x0010)



LP_CLKRST_LPCORE_RESET_CAUSE Represents the reset source of LP CPU.

0x01: Chip reset
0x09: PMU LP Peripheral reset
0x0A: PMU LP CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: Software LP CPU reset
(RO)

LP_CLKRST_LPCORE_RESET_FLAG Represents the reset flag of LP CPU.

0: Not reset
1: Reset
(RO)

Continued on the next page...

Register 8.61. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_HPCORE0_RESET_CAUSE Represents the reset source of HP CPU0.

0x01: Chip reset
0x03: Software system reset
0x05: PMU core reset
0x07: MWDT core reset
0x09: RWDT core reset
0x0B: MWDT CPU reset
0x0C: Software CPU reset
0x0D: RWDT CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: eFuse reset
0x16: USB (JTAG) reset
0x17: USB (UART) reset
0x18: JTAG CPU reset
0x1A: Lockup reset
(RO)

LP_CLKRST_HPCORE0_RESET_FLAG Represents the reset flag of HP CPU0.

0: Not reset
1: Reset
(RO)

Continued on the next page...

Register 8.61. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_HPCORE1_RESET_CAUSE Represents the reset source of HP CPU1.

0x01: Chip reset
0x03: Software system reset
0x05: PMU core reset
0x07: MWDT core reset
0x09: RWDT core reset
0x0B: MWDT CPU reset
0x0C: Software CPU reset
0x0D: RWDT CPU reset
0x0F: Brown-out system reset
0x10: RWDT system reset
0x12: Super watchdog reset
0x13: Power glitch reset
0x14: eFuse reset
0x16: USB (JTAG) reset
0x17: USB (UART) reset
0x18: JTAG CPU reset
0x1A: Lockup reset
(RO)

LP_CLKRST_HPCORE1_RESET_FLAG Represents the reset flag of HP CPU1.

0: Not reset
1: Reset
(RO)

LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK Configures whether to mask the reset source code 0x0A of PMU LP CPU reset.

0: Not mask
1: Mask
(R/W)

Continued on the next page...

Register 8.61. LP_CLKRST_RESET_CAUSE_REG (0x0010)

Continued from the previous page...

LP_CLKRST_LPCORE_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_LPCORE_RESET_CAUSE. (WT)

LP_CLKRST_LPCORE_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_LPCORE_RESET_FLAG. (WT)

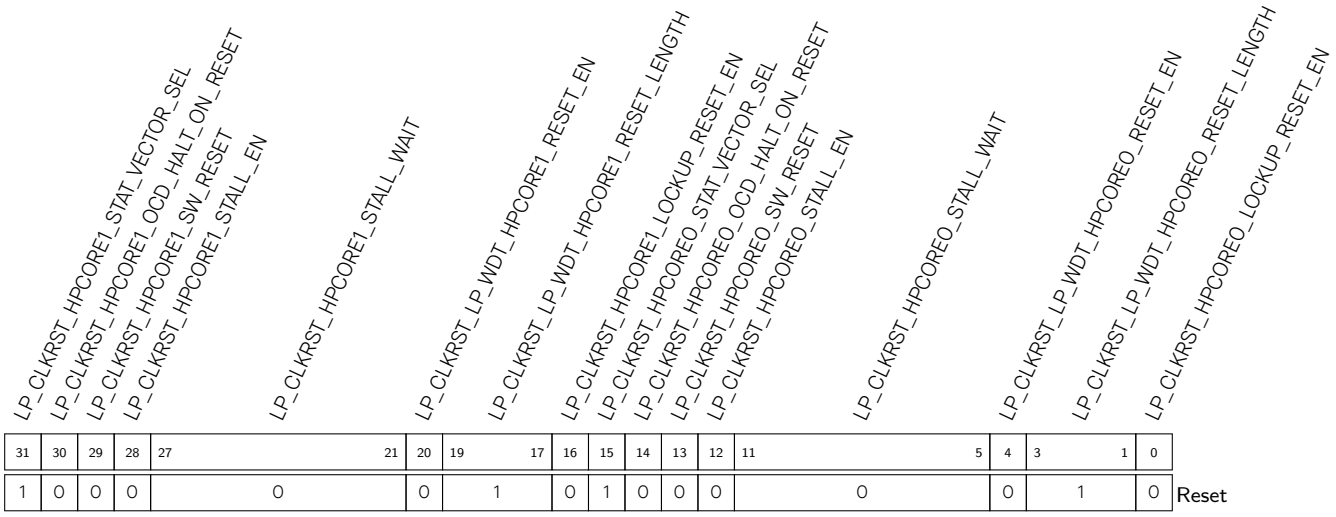
LP_CLKRST_HPCORE0_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_HPCORE0_RESET_CAUSE. (WT)

LP_CLKRST_HPCORE0_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_HPCORE0_RESET_FLAG. (WT)

LP_CLKRST_HPCORE1_RESET_CAUSE_CLR Write 1 to clear LP_CLKRST_HPCORE1_RESET_CAUSE. (WT)

LP_CLKRST_HPCORE1_RESET_FLAG_CLR Write 1 to clear LP_CLKRST_HPCORE1_RESET_FLAG. (WT)

Register 8.62. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)



LP_CLKRST_HPCORE0_LOCKUP_RESET_EN Configures whether to enable Lockup to trigger HP CPU0 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_LENGTH Configures the duration of the reset when RWDT and software trigger HP CPU0 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_EN Configures whether to enable RWDT to trigger HP CPU0 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HPCORE0_STALL_WAIT Configures the time HP CPU0 stalls when RWDT and software trigger HP CPU0 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_HPCORE0_STALL_EN Configures the behavior when RWDT and software trigger HP CPU0 reset.
0: Immediately reset HP CPU0
1: Stall HP CPU0 first
(R/W)

LP_CLKRST_HPCORE0_SW_RESET Write 1 to trigger HP CPU0 software reset. (WT)

LP_CLKRST_HPCORE0_OCD_HALT_ON_RESET Configures whether HP CPU0 halts at the first instruction after released from reset.
0: No halt
1: Halt
(R/W)

Continued on the next page...

Register 8.62. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)

Continued from the previous page...

LP_CLKRST_HPCORE0_STAT_VECTOR_SEL Configures the address from which the HP CPU0 starts.
0: From LP TCM RAM (0x50108000)
1: From HP TCM ROM (0x4FC00000)
(R/W)

LP_CLKRST_HPCORE1_LOCKUP_RESET_EN Configures whether to enable Lockup to trigger HP CPU1 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_LENGTH Configures the duration of the reset when RWDT and software trigger HP CPU1 reset.
Measurement unit: Clock cycles. (R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_EN Configures whether to enable RWDT to trigger HP CPU1 reset.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HPCORE1_STALL_WAIT Configures the time HP CPU1 stalls when RWDT and software trigger HP CPU1 reset.
Measurement unit: Clock cycles. (R/W)

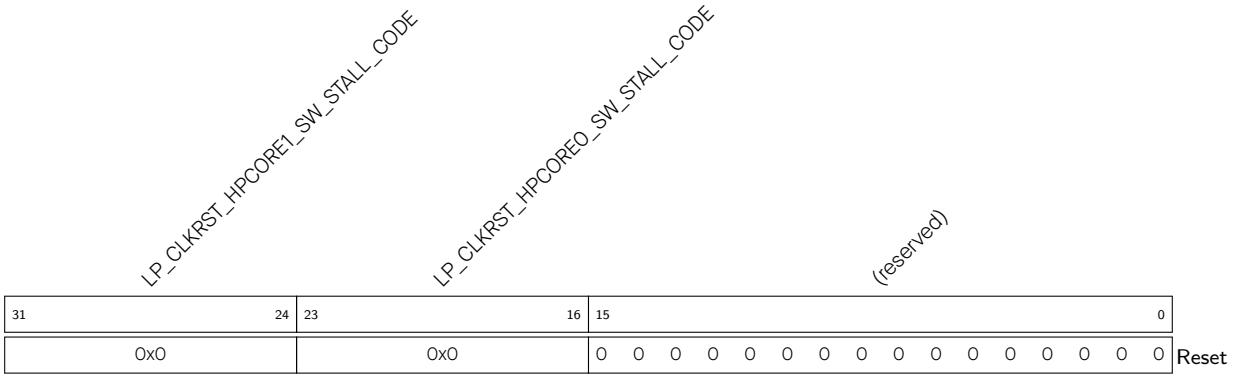
LP_CLKRST_HPCORE1_STALL_EN Configures the behavior when RWDT and software trigger HP CPU1 reset.
0: Immediately reset HP CPU1
1: Stall HP CPU1 first
(R/W)

LP_CLKRST_HPCORE1_SW_RESET Write 1 to trigger HP CPU1 software reset. (WT)

LP_CLKRST_HPCORE1_OCD_HALT_ON_RESET Configures whether HP CPU1 halts at the first instruction after released from reset.
0: No halt
1: Halt
(R/W)

LP_CLKRST_HPCORE1_STAT_VECTOR_SEL Configures the address from which the HP CPU1 starts.
0: From LP TCM RAM (0x50108000)
1: From HP TCM ROM (0x4FC00000)
(R/W)

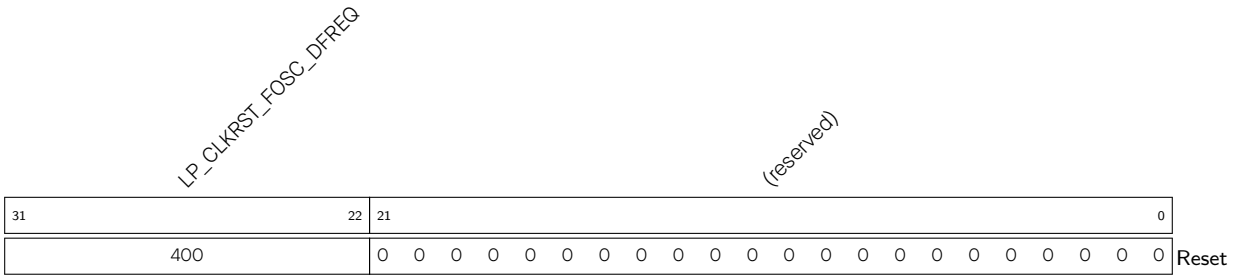
Register 8.63. LP_CLKRST_HPCPU_RESET_CTRL1_REG (0x0018)



LP_CLKRST_HPCORE0_SW_STALL_CODE Configures the threshold for HP CPU0 stall triggered by software. (R/W)

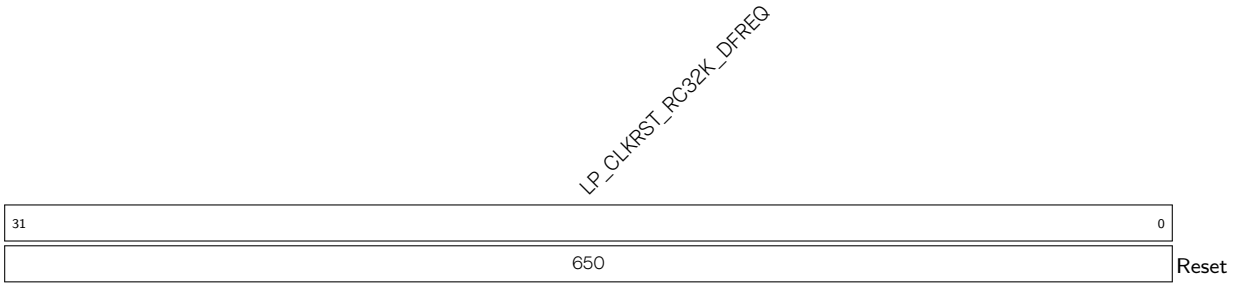
LP_CLKRST_HPCORE1_SW_STALL_CODE Configures the threshold for HP CPU1 stall triggered by software. (R/W)

Register 8.64. LP_CLKRST_FOSC_CNTL_REG (0x001C)



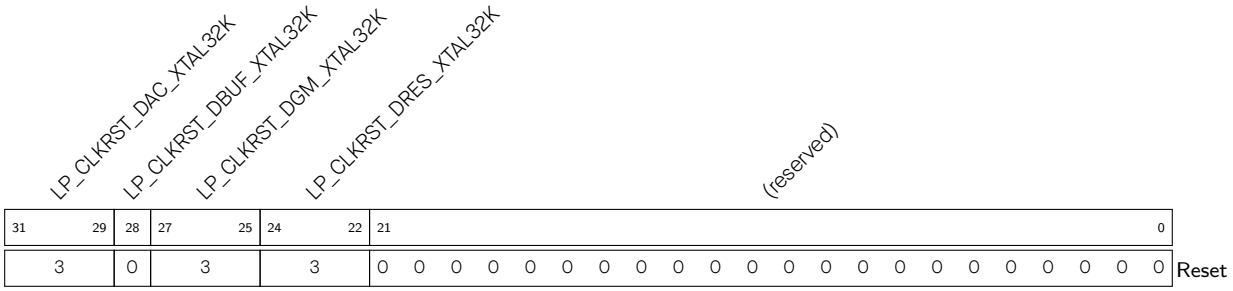
LP_CLKRST_FOSC_DFREQ Configures the frequency of RC_FAST_CLK. (R/W)

Register 8.65. LP_CLKRST_RC32K_CNTL_REG (0x0020)



LP_CLKRST_RC32K_DFREQ Configures the frequency of RC32K_CLK. (R/W)

Register 8.66. LP_CLKRST_XTAL32K_REG (0x0030)



LP_CLKRST_DRES_XTAL32K Configures the bias resistor of XTAL32K_CLK.

0: Turn off internal resistor, use external resistor

1-3: Internal bias resistor value

(R/W)

LP_CLKRST_DGM_XTAL32K Configures the gm of XTAL32K_CLK. (R/W)

LP_CLKRST_DBUF_XTAL32K Configures the buffer of XTAL32K_CLK.

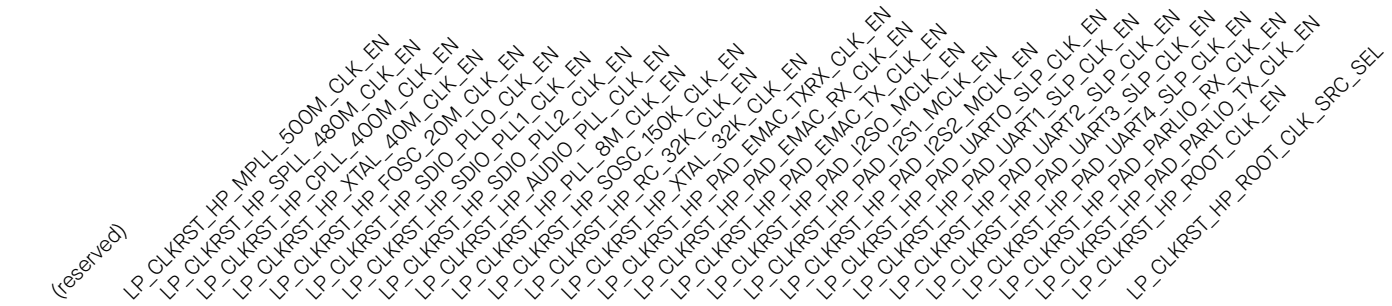
0: Single-ended buffer

1: Differential buffer

(R/W)

LP_CLKRST_DAC_XTAL32K Configures the bias current DAC of XTAL32K_CLK. (R/W)

Register 8.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)



LP_CLKRST_HP_ROOT_CLK_SRC_SEL Configures the clock source of ROOT_CLK.

0: XTAL_CLK

1: CPLL_CLK (400 MHz)

2: RC_FAST_CLK

3: Invalid

(R/W)

LP_CLKRST_HP_ROOT_CLK_EN Configures whether to enable ROOT_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_PARLIO_TX_CLK_EN Configures whether to enable PAD_PARLIO_TX_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_PARLIO_RX_CLK_EN Configures whether to enable PAD_PARLIO_RX_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_UART4_SLP_CLK_EN Configures whether to enable PAD_UART4_SLP_CLK from IO pin.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_PAD_UART3_SLP_CLK_EN Configures whether to enable PAD_UART3_SLP_CLK from IO pin.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_PAD_UART2_SLP_CLK_EN Configures whether to enable PAD_UART2_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_UART1_SLP_CLK_EN Configures whether to enable PAD_UART1_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_UART0_SLP_CLK_EN Configures whether to enable PAD_UART0_SLP_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S2_MCLK_EN Configures whether to enable PAD_I2S2_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S1_MCLK_EN Configures whether to enable PAD_I2S1_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_I2S0_MCLK_EN Configures whether to enable PAD_I2S0_MCLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_EMAC_TX_CLK_EN Configures whether to enable PAD_EMAC_RX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_PAD_EMAC_RX_CLK_EN Configures whether to enable PAD_EMAC_TX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PAD_EMAC_TXRX_CLK_EN Configures whether to enable PAD_EMAC_TXRX_CLK from IO pin.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_XTAL_32K_CLK_EN Configures whether to enable XTAL32K_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_RC_32K_CLK_EN Configures whether to enable RC32K_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_SOSC_150K_CLK_EN Configures whether to enable RC_SLOW_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_PLL_8M_CLK_EN Configures whether to enable PLL_LP_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_AUDIO_PLL_CLK_EN Configures whether to enable APLL_CLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_HP_SDIO_PLL2_CLK_EN Configures whether to enable SDIO_PLL2_CLK.
0: Disable
1: Enable
(R/W)

Continued on the next page...

Register 8.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

Continued from the previous page...

LP_CLKRST_HP_SDIO_PLL1_CLK_EN Configures whether to enable SDIO_PLL1_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_SDIO_PLL0_CLK_EN Configures whether to enable SDIO_PLL0_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_FOSC_20M_CLK_EN Configures whether to enable RC_FAST_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_XTAL_40M_CLK_EN Configures whether to enable XTAL_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_CPLL_400M_CLK_EN Configures whether to enable CPLL_CLK.

0: Disable

1: Enable

(R/W)

LP_CLKRST_HP_SPLL_480M_CLK_EN Configures whether to enable SPLL_CLK.

0: Disable

1: Enable

(R/W)

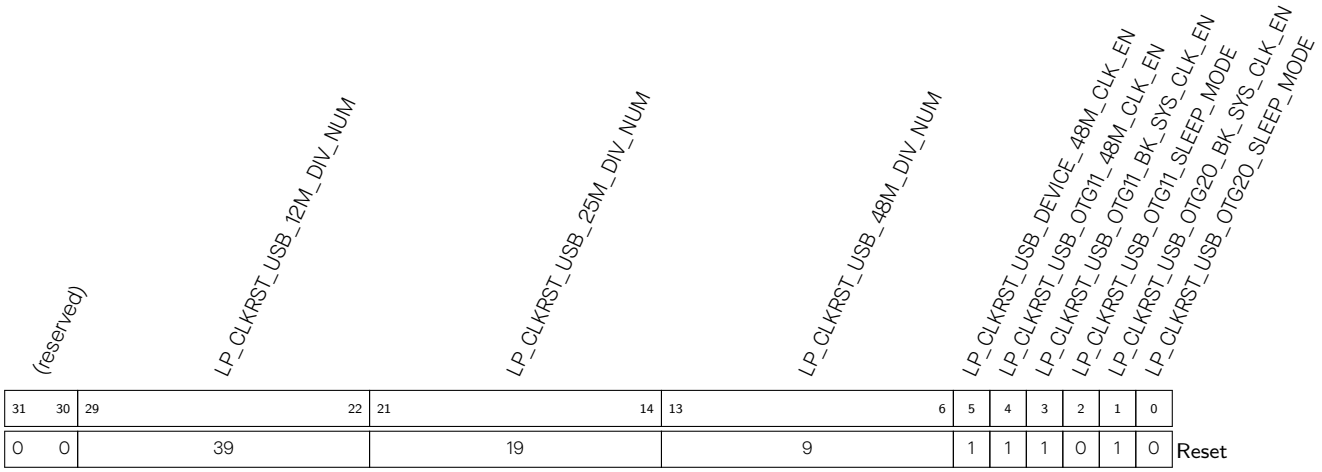
LP_CLKRST_HP_MPLL_500M_CLK_EN Configures whether to enable MPLL_CLK.

0: Disable

1: Enable

(R/W)

Register 8.68. LP_CLKRST_HP_USB_CLKRST_CTRL0_REG (0x0044)



LP_CLKRST_USB_OTG20_SLEEP_MODE Not used. (R/W)

LP_CLKRST_USB_OTG20_BK_SYS_CLK_EN Not used. (R/W)

LP_CLKRST_USB_OTG11_SLEEP_MODE Not used. (R/W)

LP_CLKRST_USB_OTG11_BK_SYS_CLK_EN Not used. (R/W)

LP_CLKRST_USB_OTG11_48M_CLK_EN Configures whether to enable theFull-Speed USB 2.0 OTG PHY clock.
0: Disable
1: Enable
(R/W)

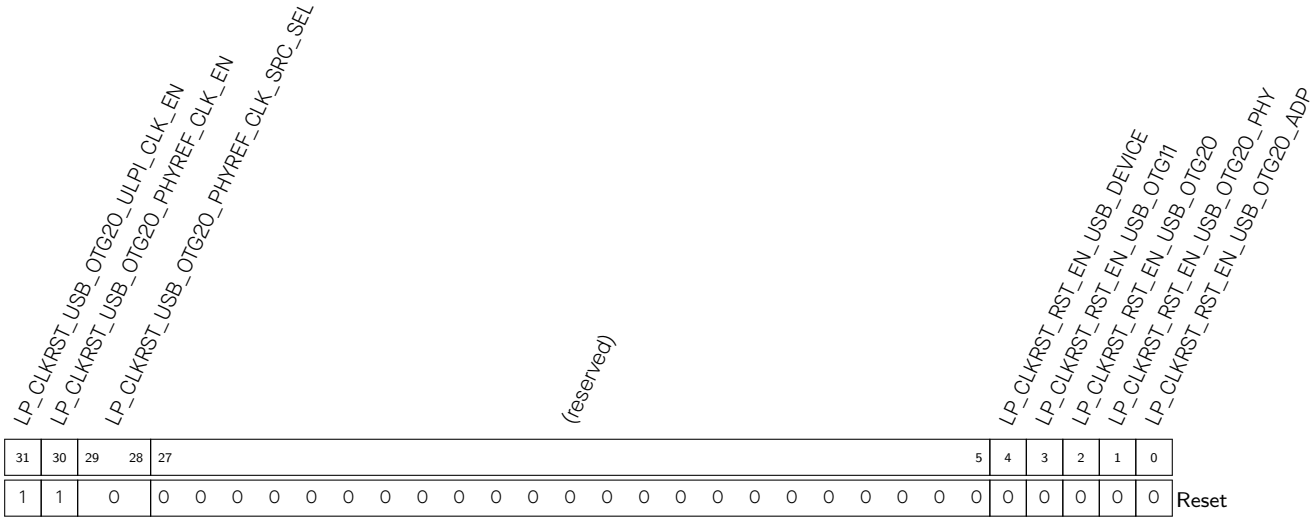
LP_CLKRST_USB_DEVICE_48M_CLK_EN Configures whether to enable the USB Serial/JTAG PHY clock.
0: Disable
1: Enable
(R/W)

LP_CLKRST_USB_48M_DIV_NUM Configures the division number from USB 480M to 25M. (R/W)

LP_CLKRST_USB_25M_DIV_NUM Configures the division number from USB 500M to 25M. (R/W)

LP_CLKRST_USB_12M_DIV_NUM Configures the division number from USB 480M to 12M. (R/W)

Register 8.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)



LP_CLKRST_RST_EN_USB_OTG20_ADP Configures whether to reset High-Speed USB 2.0 OTG ADP.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG20_PHY Configures whether to reset High-Speed USB 2.0 OTG PHY.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG20 Configures whether to reset High-Speed USB 2.0 OTG.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_OTG11 Configures whether to reset Full-Speed USB 2.0 OTG.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_RST_EN_USB_DEVICE Configures whether to reset USB Serial/JTAG.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)

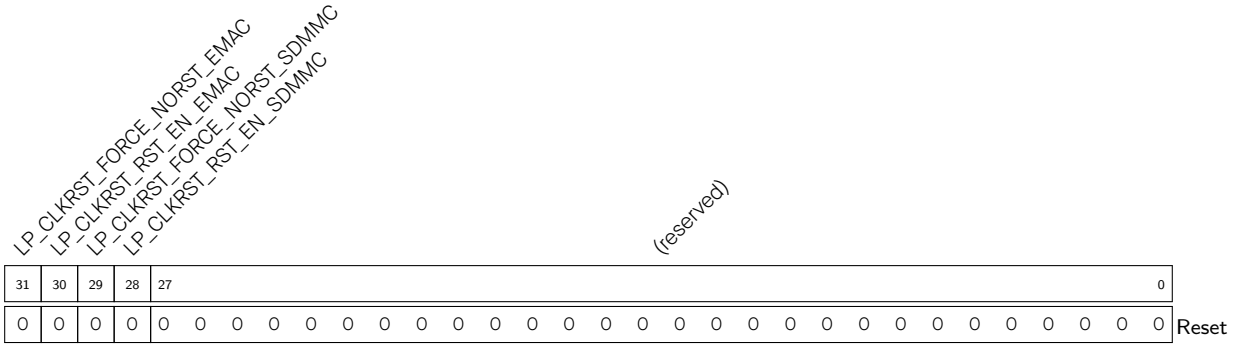
Continued from the previous page...

LP_CLKRST_USB_OTG20_PHYREF_CLK_SRC_SEL Configures the clock source of High-Speed USB 2.0 OTG HS PHY.
0: PLL_F12M_CLK
1: PLL_F25M_CLK
2: PAD_USB_HSPHY_REFCLK
3: Invalid
(R/W)

LP_CLKRST_USB_OTG20_PHYREF_CLK_EN Configures whether to enable High-Speed USB 2.0 OTG HS PHY REFCLK.
0: Disable
1: Enable
(R/W)

LP_CLKRST_USB_OTG20_ULPI_CLK_EN Configures whether to enable High-Speed USB 2.0 OTG ULPI clock.
0: Disable
1: Enable
(R/W)

Register 8.70. LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG (0x004C)



LP_CLKRST_RST_EN_SDMMC Configures whether to reset SDMMC.

0: Release from reset

1: Reset

(R/W)

LP_CLKRST_FORCE_NORST_SDMMC Configures whether SDMMC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

LP_CLKRST_RST_EN_EMAC Configures whether to reset EMAC.

0: Release from reset

1: Reset

(R/W)

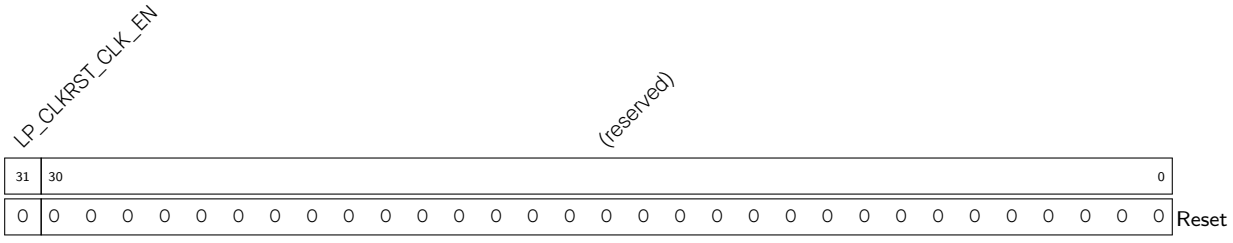
LP_CLKRST_FORCE_NORST_EMAC Configures whether EMAC can be reset.

0: Can be reset

1: Force not to be reset

(R/W)

Register 8.71. LP_CLKRST_DATE_REG (0x03FC)



LP_CLKRST_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

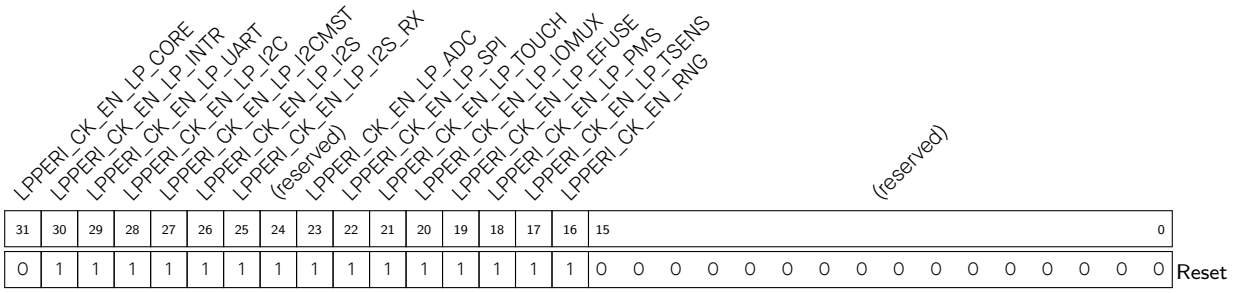
(R/W)

8.5.3 LP Peripheral Clock and Reset (LPPERI) Registers

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

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

Register 8.72. LPPERI_CLK_EN_REG (0x0000)



LPPERI_CK_EN_RNG Configures whether to enable LP RNG clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_TSENS Configures whether to enable LP temperature sensor clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_PMS Configures whether to enable LP PMS clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_EFUSE Configures whether to enable LP eFuse clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_IOMUX Configures whether to enable LP IO MUX clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_TOUCH Configures whether to enable LP touch sensor clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_SPI Configures whether to enable LP SPI clock.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 8.72. LPPERI_CLK_EN_REG (0x0000)

Continued from the previous page...

LPPERI_CK_EN_LP_ADC Configures whether to enable LP ADC clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2S_RX Configures whether to enable LP I2S RX clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2S Configures whether to enable LP I2S clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2CMST Configures whether to enable LP I2C master clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_I2C Configures whether to enable LP I2C clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_UART Configures whether to enable LP UART clock.

0: Disable

1: Enable

(R/W)

LPPERI_CK_EN_LP_INTR Configures whether to enable LP interrupt matrix clock.

0: Disable

1: Enable

(R/W)

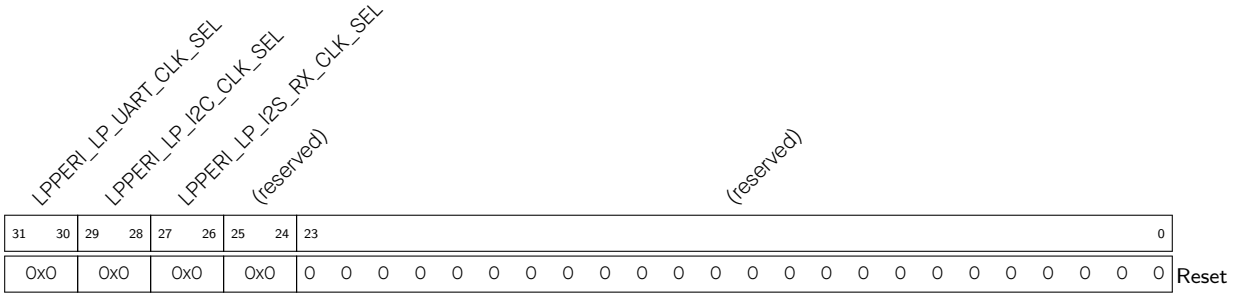
LPPERI_CK_EN_LP_CORE Configures whether to force on LP CPU clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

Register 8.73. LPPERI_CORE_CLK_SEL_REG (0x0004)



LPPERI_LP_I2S_RX_CLK_SEL Configures LP I2S RX clock source.

0: LP_FAST_CLK

1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LPPERI_LP_I2C_CLK_SEL Configures LP I2C clock source.

0: LP_FAST_CLK

1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

LPPERI_LP_UART_CLK_SEL Configures LP UART clock source.

0: LP_FAST_CLK

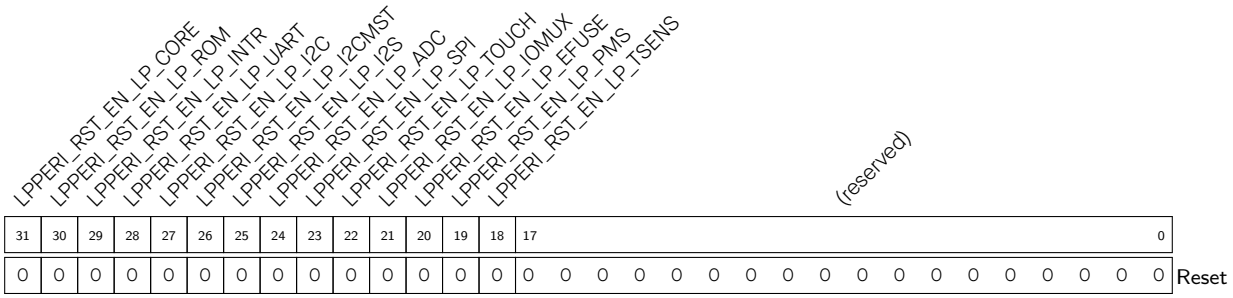
1: XTAL_D2_CLK

2: PLL_LP_CLK

3: Invalid

(R/W)

Register 8.74. LPPERI_RESET_EN_REG (0x0008)



LPPERI_RST_EN_LP_TSENS Configures whether to reset LP temperature sensor.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_PMS Configures whether to reset LP PMS.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_EFUSE Configures whether to reset LP eFuse.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_IOMUX Configures whether to reset LP IO MUX.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_TOUCH Configures whether to reset LP touch sensor.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_SPI Configures whether to reset LP SPI.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_ADC Configures whether to reset LP ADC.

0: Release from reset

1: Reset

(R/W)

Continued on the next page...

Register 8.74. LPPERI_RESET_EN_REG (0x0008)

Continued from the previous page...

LPPERI_RST_EN_LP_I2S Configures whether to reset LP I2S.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_I2CMST Configures whether to reset LP I2C master.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_I2C Configures whether to reset LP I2C.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_UART Configures whether to reset LP UART.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_INTR Configures whether to reset LP interrupt matrix.

0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_ROM Configures whether to reset LP ROM.

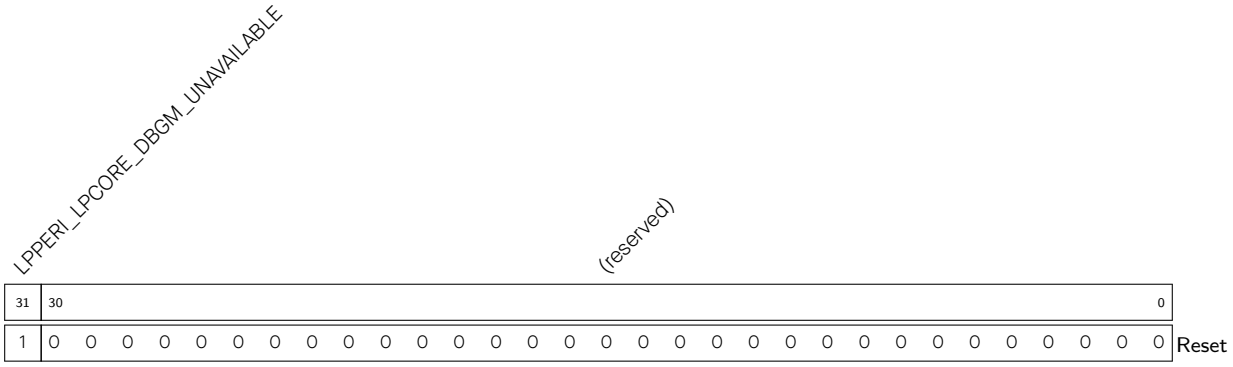
0: Release from reset

1: Reset

(R/W)

LPPERI_RST_EN_LP_CORE Writing 1 triggers LP CPU software reset. (WT)

Register 8.75. LPPERI_CPU_REG (0x000C)



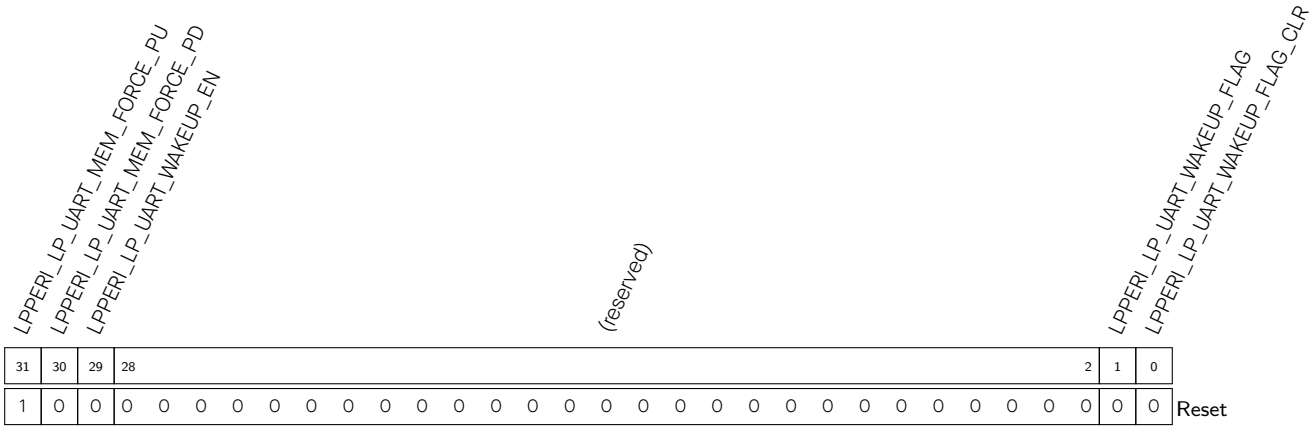
LPPERI_LPCORE_DBGM_UNAVAILABLE Configures whether the LP CPU can be accessed by the HP CPU.

0: Not accessible

1: Accessible

(R/W)

Register 8.76. LPPERI_MEM_CTRL_REG (0x0028)



LPPERI_LP_UART_WAKEUP_FLAG_CLR Write 1 to clear LPPERI_LP_UART_WAKEUP_FLAG. (WT)

LPPERI_LP_UART_WAKEUP_FLAG Represents whether an LP UART wakeup has occurred.

0: No LP UART wakeup occurred

1: LP UART wakeup occurred

(R/WTC/SS)

LPPERI_LP_UART_WAKEUP_EN Configures whether to enable the LP UART wakeup function.

0: Enable

1: Disable

(R/W)

LPPERI_LP_UART_MEM_FORCE_PD Configures whether to force the LP UART memory to power down.

0: Do not force power down

1: Force power down

(R/W)

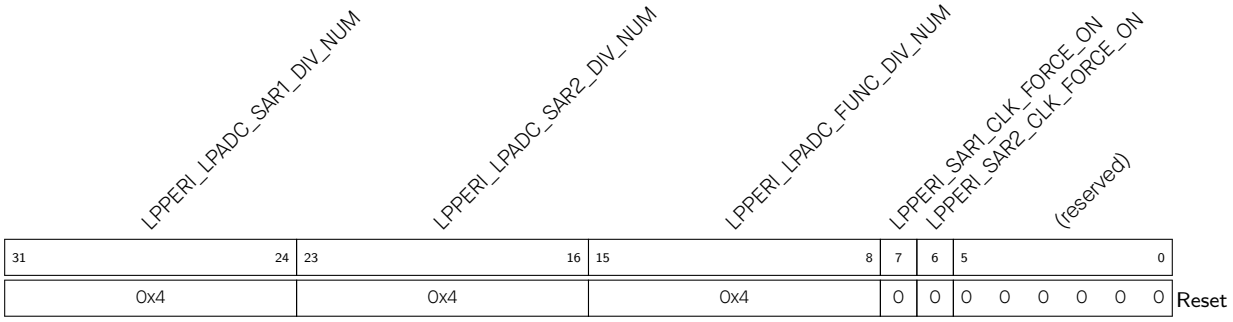
LPPERI_LP_UART_MEM_FORCE_PU Configures whether to force the LP UART memory to power up.

0: Do not force power up

1: Force power up

(R/W)

Register 8.77. LPPERI_ADC_CTRL_REG (0x002C)



LPPERI_SAR2_CLK_FORCE_ON Configures whether to force on the LP ADC SAR1 clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

(R/W)

LPPERI_SAR1_CLK_FORCE_ON Configures whether to force on the LP ADC SAR2 clock.

0: Controlled by hardware

1: Force on, bypassing hardware control

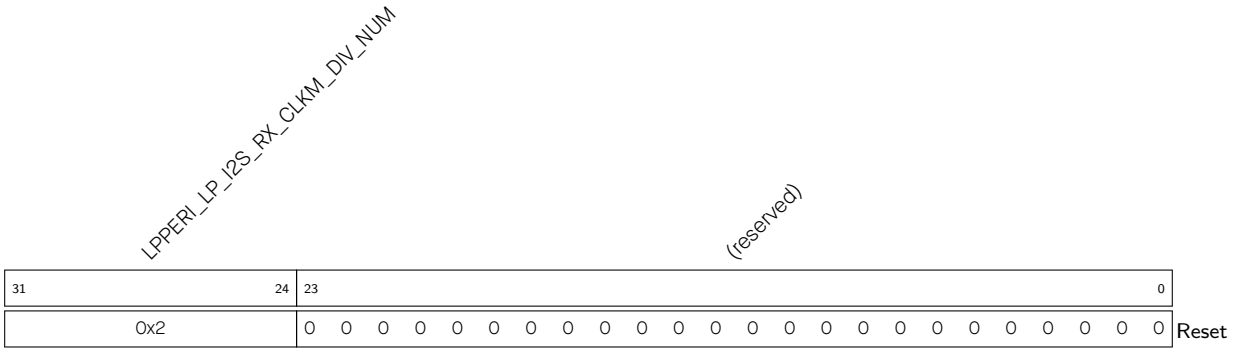
(R/W)

LPPERI_LPADC_FUNC_DIV_NUM Configures the LP ADC function clock divisor. (R/W)

LPPERI_LPADC_SAR2_DIV_NUM Configures the LP ADC SAR1 clock divisor.(R/W)

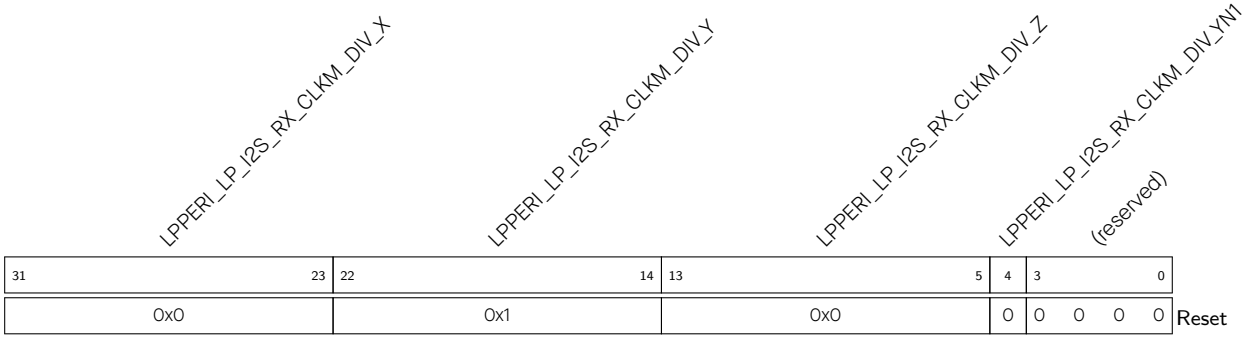
LPPERI_LPADC_SAR1_DIV_NUM Configures the LP ADC SAR2 clock divisor.(R/W)

Register 8.78. LPPERI_LP_I2S_RXCLK_DIV_NUM_REG (0x0030)



LPPERI_LP_I2S_RX_CLKM_DIV_NUM Configures the integer part of the LP I2S RX clock divisor. (R/W)

Register 8.79. LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG (0x0034)



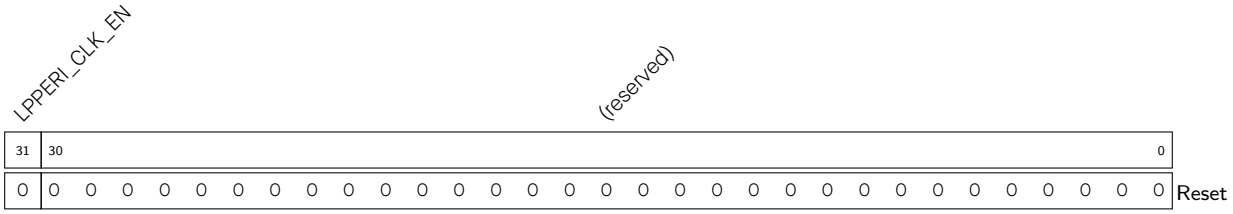
LPPERI_LP_I2S_RX_CLKM_DIV_YN1 Configures the yn1 coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Z Configures the z coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Y Configures the y coefficient of the LP I2S RX clock divisor. (R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_X Configures the x coefficient of the LP I2S RX clock divisor. (R/W)

Register 8.80. LPPERI_DATE_REG (0x03FC)



LPPERI_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Chapter 9

Chip Boot Control

9.1 Overview

Chip boot process and some chip functions are determined on power-on or hardware reset using strapping pins and eFuse bits. The following functionality can be determined:

- chip boot mode
- enable or disable of ROM messages printing
- source of JTAG signals

ESP32-P4 has five strapping pins:

- GPIO34
- GPIO35
- GPIO36
- GPIO37
- GPIO38

During power-on reset (see Chapter 8 *Reset and Clock*), hardware samples and stores the voltage level of strapping pins as strapping bit of "0" or "1" in latches, and holds these bits until the chip is powered down. Software can read the latch status (strapping value) from GPIO_STRAPPING.

9.2 Functional Description

This section provides description of the chip functions and the patterns of the strapping pins and eFuse values to invoke each function.

Notice:

Only documented patterns should be used. If an undocumented pattern is used, it may trigger unexpected behaviors.

9.2.1 Default Configuration

By default, GPIO35 is connected to the chip's internal pull-up resistor. If GPIO35 is not connected or is connected to an external high-impedance circuit, the internal weak pull-up determines the default input level of this strapping pin (see Table 9.2-1).

Table 9.2-1. Default Configuration of Strapping Pins

Strapping Pin	Default Configuration
GPIO34	Floating
GPIO35	Pull-up
GPIO36	Floating
GPIO37	Floating
GPIO38	Floating

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-P4. After the reset is released, the strapping pins work as normal-function pins.

9.2.2 Boot Mode Control

The values of GPIO35, GPIO36, GPIO37 and GPIO38 at reset determine the boot mode after the reset is released. Table 9.2-2 shows the strapping pin values of GPIO35, GPIO36, GPIO37 and GPIO38 and the associated boot modes.

Table 9.2-2. Boot Mode Control

Boot Mode	GPIO35	GPIO36	GPIO37	GPIO38
SPI Boot mode (default)	1	x[1]	x	x
Joint Download Boot mode[2]	0	1	x	x
SPI Download Boot mode[3]	0	0	0	1
Invalid Combination[4]	0	0	1	x
	0	0	0	0

[1] x: values that have no effect on the result and can therefore be ignored.

[2] Joint Download Boot mode: Joint Download Boot mode supports the following download methods:

- USB-Serial-JTAG Download Boot
- UART Download Boot
- SPI Slave Download Boot
- USB 2.0 OTG Download Boot

[3] SPI Download Boot mode: GPIO37 and GPIO38 need to be reserved only when using SPI Download Boot mode. GPIO37 and GPIO38 are floating by default and are in a high-impedance state at reset.

[4] Invalid Combination: This combination can trigger unexpected behavior and should be avoided.

In SPI Boot mode, the ROM bootloader loads and executes the program from SPI flash to boot the system. SPI Boot mode can be further classified as follows:

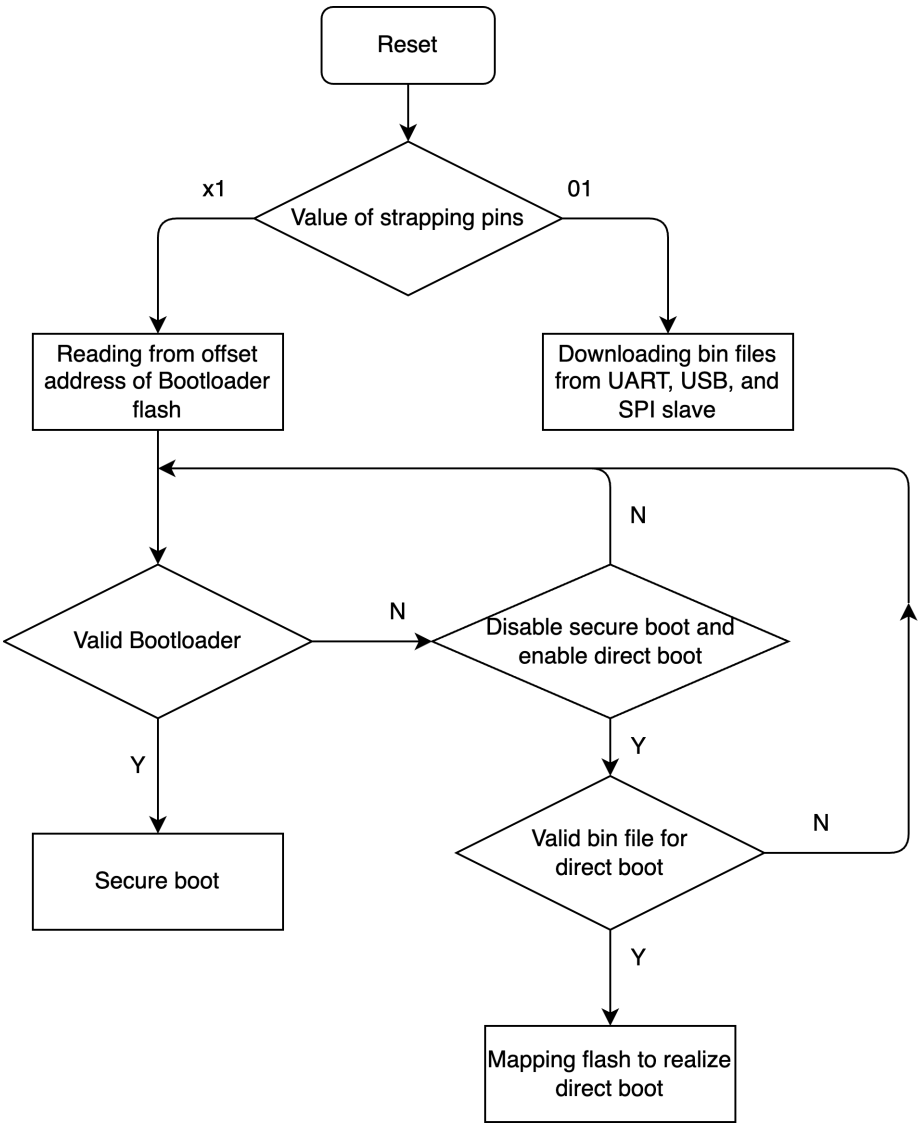
- Normal flash Boot: supports Secure Boot. The ROM bootloader loads the program from flash into L2MEM and executes it. In most practical scenarios, this program is the 2nd stage bootloader, which later boots the target application.

- Direct Boot: does not support Secure Boot and programs run directly from flash. To enable this mode, make sure that the first two words of the bin file downloaded to flash are 0xaedb041d. For more detailed process, see Figure 9.2-1.

In Joint Download Boot mode, users can download binary files into flash using UART0, SPI slave, USB 2.0 OTG or USB Serial/JTAG interface. It is also possible to download binary files into L2MEM and execute it from L2MEM.

In SPI Download Boot mode, users can download binary files into flash using SPI interface. It is also possible to download binary files into L2MEM and execute it from L2MEM.

Figure 9.2-1 shows the detailed boot flow of the chip.



Note: "x1" and "01" are combination values of strapping pins GPIO35 and GPIO36, see Table 9.2-2.

Figure 9.2-1. Chip Boot Flow

The following eFuse bits allows controlling boot mode behaviors:

- EFUSE_DIS_FORCE_DOWNLOAD

  If this eFuse is 0 (default), software can force switch the chip from SPI Boot mode to Joint Download

Boot mode by setting register LPSYSREG_FORCE_DOWNLOAD_BOOT and triggering a CPU reset. If this eFuse is 1, LPSYSREG_FORCE_DOWNLOAD_BOOT is disabled.

- EFUSE_DIS_DOWNLOAD_MODE

  If this eFuse is 1, Joint Download Boot mode is permanently disabled.

- EFUSE_ENABLE_SECURITY_DOWNLOAD

  If this eFuse is 1, Joint Download Boot mode only allows reading, writing, and erasing plaintext flash and does not support any L2MEM or register operations. Ignore this eFuse if Download Boot mode is disabled.

- EFUSE_DIS_DIRECT_BOOT

  If this eFuse is 1, Direct Boot would be disabled in SPI Boot Mode.

USB Serial/JTAG Controller can also force switch the chip to Joint Download Boot mode from SPI Boot mode, and vice versa. For detailed information, please refer to Chapter *36 USB Serial/JTAG Controller (USB_SERIAL_JTAG)*.

You can set eFuse bit EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE to disable USB Serial/JTAG Controller from force switching to Joint Download Boot mode.

9.2.3 ROM Messages Printing Control

During early SPI Boot process, the messages by the ROM code can be printed to:

- **(Default) UART0 and USB Serial/JTAG controller**
- **UART0**
- **USB Serial/JTAG controller**

EFUSE_UART_PRINT_CONTROL and GPIO36 control ROM messages printing to **UART0** as shown in Table 9.2-3 ROM Message Printing Control.

Table 9.2-3. ROM Message Printing Control

eFuse[1]	GPIO36	ROM Code Printing
0	x[2]	Always enabled
1	0	Enabled
	1	Disabled
2	0	Disabled
	1	Enabled
3	x	Always disabled

[1] eFuse: EFUSE_UART_PRINT_CONTROL

[2] x: values that have no effect on the result and can therefore be ignored.

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT controls the printing to **USB Serial/JTAG controller**. When this bit is 1, printing to USB Serial/JTAG controller is disabled. When this bit is 0, and USB Serial/JTAG controller is enabled via EFUSE_DIS_USB_SERIAL_JTAG, ROM messages can be printed to USB Serial/JTAG controller.

Note that if EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT is set to 0 to print to USB, but USB Serial/JTAG Controller has been disabled, then ROM messages will not be printed to USB Serial/JTAG Controller.

9.2.4 JTAG Signal Source Control

GPIO34 controls the source of JTAG signals during the early boot process. This GPIO is used together with EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_JTAG_SEL_ENABLE. See Table 9.2-4.

Table 9.2-4. JTAG Signal Source Control

eFuse 1[a]	eFuse 2[b]	eFuse 3[c]	GPIO34	Signal Source
0	0	0	x[d]	JTAG signals come from USB Serial/JTAG Controller.
		1	0	JTAG signals come from corresponding pins MTDI, MTCK, MTMS, and MTDO.
			1	JTAG signals come from USB Serial/JTAG Controller.
0	1	x	x	JTAG signals come from corresponding pins MTDI, MTCK, MTMS, and MTDO.
1	0	x	x	JTAG signals come from USB Serial/JTAG Controller.
1	1	x	x	JTAG is disabled.

[a] eFuse 1: EFUSE_DIS_PAD_JTAG

[b] eFuse 2: EFUSE_DIS_USB_JTAG

[c] eFuse 3: EFUSE_JTAG_SEL_ENABLE

[d] x: values that have no effect on the result and can therefore be ignored.

Chapter 10

Interrupt Matrix

10.1 Overview

The interrupt matrix embedded in ESP32-P4 independently routes peripheral interrupt sources to the ESP-RISC-V CPU's peripheral interrupts to timely inform HP CPU0 or HP CPU1 to process the coming interrupts.

Peripheral interrupt sources must be routed to HP CPU0/HP CPU1 peripheral interrupts via this interrupt matrix due to the following considerations:

- ESP32-P4 has 126 peripheral interrupt sources. To map them to 32 HP CPU0 interrupts or 32 HP CPU1 interrupts, this matrix is needed.
- Through this matrix, one peripheral interrupt source can be mapped to multiple HP CPU0 interrupts or HP CPU1 interrupts according to application requirements.

Note:
This chapter focuses on how to map peripheral interrupt sources to HP CPU0/HP CPU1 interrupts. For more details about interrupt configuration, vector, and interrupt handling operations recommended by the ISA, please refer to Chapter 1 *High-Performance CPU [to be added later]*.

10.2 Interrupt Terminology in ESP32-P4

The following terms related to interrupts are defined in the context of the *ESP32-P4 Technical Reference Manual* to help readers better understand this document:

10.2.1 Interrupt

An interrupt refers to the event or condition that occurs, causing the CPU to temporarily suspend its current execution and handle a higher-priority task. It is a mechanism that allows the CPU to respond to specific events promptly.

The *ESP32-P4 Technical Reference Manual* may use the term "interrupt" in a broader sense to refer to both interrupt signal and interrupt source.

10.2.2 Interrupt Signal/interrupt Source

Interrupt signal and interrupt source are only defined from different perspectives and mean the same thing.

From the perspective of peripherals, interrupt signals are generated by the peripheral's internal interrupt sources and are sent to the interrupt matrix.

From the perspective of the interrupt matrix, it receives the interrupt signals sent from the peripheral and considers them interrupt sources. The interrupt matrix then outputs CPU peripheral interrupt signals to the CPU.

From the perspective of the CPU, the interrupt signals from the interrupt matrix become sources and are sent to the CPU core together with the core local interrupt sources.

10.2.3 Interrupt Flow in ESP32-P4

Figure 10.2-1 shows the interrupt flow in ESP32-P4.

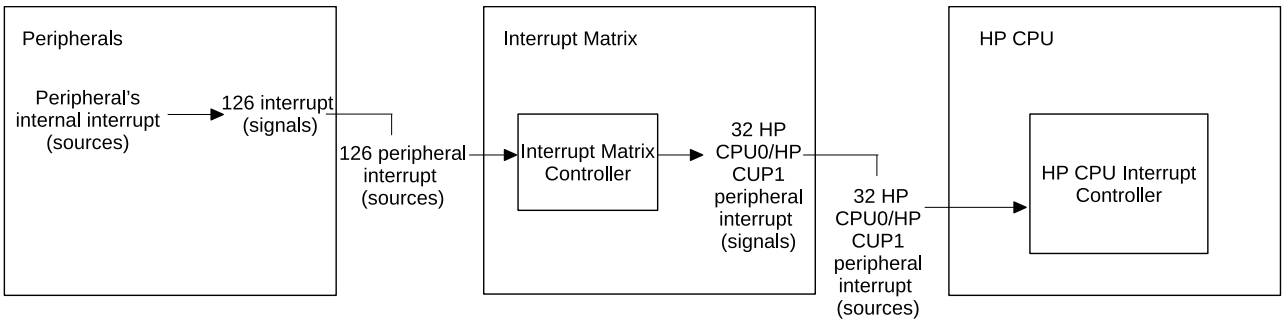


Figure 10.2-1. Interrupt Flow in ESP32-P4

10.3 Features

The interrupt matrix embedded in ESP32-P4 has the following features:

- 126 peripheral interrupt sources accepted as input
- 32 HP CPU0 peripheral interrupts and 32 HP CPU1 peripheral interrupts generated to HP CPU as output
- Current interrupt status query of peripheral interrupt sources
- Multiple interrupt sources mapping to a single HP CPU0 or HP CPU1 interrupt (i.e., shared interrupts)

Figure 10.3-1 shows the structure of the interrupt matrix.

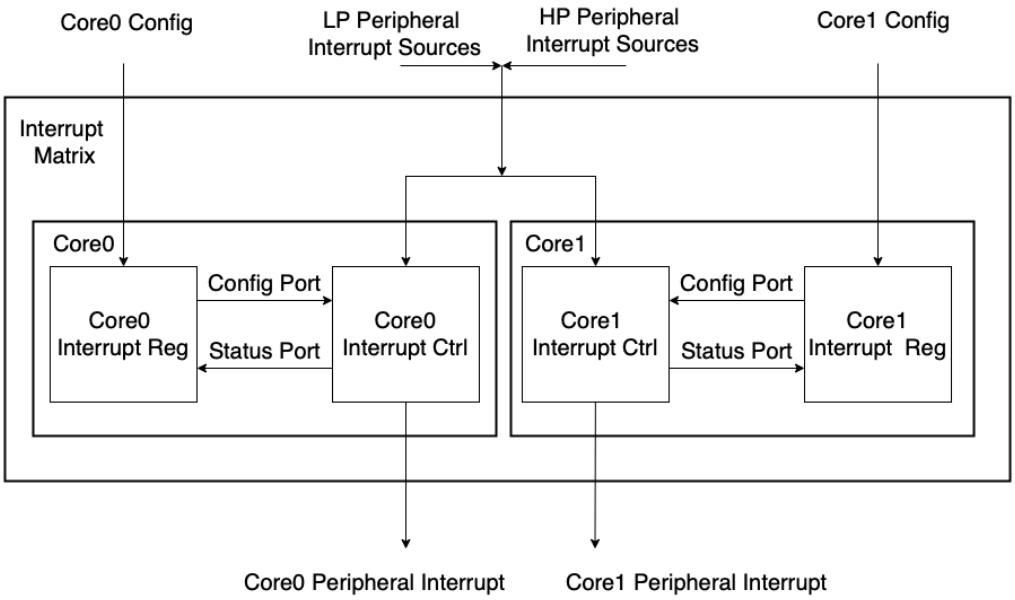


Figure 10.3-1. Interrupt Matrix Structure

All the interrupts generated by the peripheral interrupt sources can be handled by HP CPU0 or HP CPU1. Users can configure HP CPU0 interrupt registers ("Core0 Interrupt Reg" module in Figure 10.3-1) to assign peripheral interrupt sources to HP CPU0, or configure HP CPU1 interrupt registers ("Core1 Interrupt Reg" module in Figure 10.3-1) to assign peripheral interrupt sources to HP CPU1. Peripheral interrupt sources can be assigned both to HP CPU0 and HP CPU1 simultaneously, and if so, HP CPU0 and HP CPU1 will both accept the interrupts.

10.4 Functional Description

10.4.1 Peripheral Interrupt Sources

ESP32-P4 has 126 peripheral interrupt sources in total. Table 10.4-1 lists all these sources and their mapping/status registers.

- Column "No.": Peripheral interrupt source number, can be 0 ~ 98, 100 ~ 111, 113 ~ 127 (99 and 112 are reserved).
- Column "Chapter": in which chapter the interrupt source is described in detail.
- Column "Interrupt Source": Name of the peripheral interrupt source.
- Column "Interrupt Source Mapping Register": Registers used for routing the peripheral interrupt sources to HP CPU peripheral interrupts.
- Column "Interrupt Status Register": Registers used for indicating the interrupt status of peripheral interrupt sources.
  - Column "Interrupt Status Register - Bit": Bit position in status register, indicating the interrupt status.
  - Column "Interrupt Status Register - Name": Name of status registers.

The register in column "Interrupt Source Mapping Register" and in column "Interrupt Status Register" correspond to the peripheral interrupt source in column "Interrupt Source". For example, the interrupt source

mapping register for interrupt source RMT_INTR is CORE*x*_RMT_INT_MAP_REG, and its status register is CORE*x*_INTR_STATUS_REG_1_REG.

Note that CORE*x* in the table can be CORE0 (HP CPU0) or CORE1 (HP CPU1).

Table 10.4-1. CPU Peripheral Interrupt Source Mapping/Status Registers and Peripheral Interrupt Sources

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
0	*RTC Timer*	LP_RTC_INTR	COREx LP_RTC_INT_MAP_REG	0	COREx_INTR_STATUS_REG_0_REG
1	*Watchdog Timers (WDT)*	LP_WDT_INTR	COREx_LP_WDT_INT_MAP_REG	1	
2	*RTC Timer*	LP_TIMER_REG_0_INTR	COREx_LP_TIMER_REG_0_INT_MAP_REG	2	
3	*RTC Timer*	LP_TIMER_REG_1_INTR	COREx_LP_TIMER_REG_1_INT_MAP_REG	3	
4	*LP Mailbox*	MB_LP_INTR	COREx_MB_LP_INT_MAP_REG	4	
5	*LP Mailbox*	MB_HP_INTR	COREx_MB_LP_INT_MAP_REG	5	
6	*Low-Power Management [to be added later]*	PMU_REG_0_INTR	COREx_PMU_REG_0_INT_MAP_REG	6	
7	*Low-Power Management [to be added later]*	PMU_REG_1_INTR	COREx_PMU_REG_1_INT_MAP_REG	7	
8	*Brown-out Detector*	LP_ANAPERI_INTR	COREx_PMU_REG_0_INT_MAP_REG	8	
9	*ADC Controller (ADC)*	LP_ADC_INTR	COREx_PMU_REG_0_INT_MAP_REG	9	
10	*GPIO Matrix and IO MUX*	LP_GPIO_INTR	COREx_LP_GPIO_INT_MAP_REG	10	
11	*I2C Controller (I2C)*	LP_I2C_INTR	COREx_LP_I2C_INT_MAP_REG	11	
12	*I2S Controller (I2S)*	LP_I2S_INTR	COREx_LP_I2S_INT_MAP_REG	12	
13	*SPI Controller [to be added later]*	LP_SPI_INTR	COREx_LP_SPI_INT_MAP_REG	13	
14	*Touch Sensor [to be added later]*	LP_TOUCH_INTR	COREx_LP_TOUCH_INT_MAP_REG	14	
15	*Temperature Sensor (TSENS)*	LP_TSENS_INTR	COREx_LP_TSENS_INT_MAP_REG	15	
16	*UART Controller (UART)*	LP_UART_INTR	COREx_LP_UART_INT_MAP_REG	16	
17	*eFuse Controller*	LP_EFUSE_INTR	COREx_LP_EFUSE_INT_MAP_REG	17	
18	*Low-Power CPU*	LP_SW_INTR	COREx_LP_SW_INT_MAP_REG	18	
19	*System Registers [to be added later]*	LP_SYSREG_INTR	COREx_LP_SYSREG_INT_MAP_REG	19	
20	*Key Manager [to be added later]*	LP_HUK_INTR	COREx_LP_HUK_INT_MAP_REG	20	
21	*System Registers [to be added later]*	SYS_ICM_INTR	COREx_SYS_ICM_INT_MAP_REG	21	
22	*USB Serial/JTAG Controller (USB_SERIAL_JTAG)*	USB_DEVICE_INTR	COREx_USB_DEVICE_INT_MAP_REG	22	
23	*SD/MMC Host Controller (SDHOST)*	SDIO_HOST_INTR	COREx_SDIO_HOST_INT_MAP_REG	23	
24	*DW-GDMA Controller [to be added later]*	GDMA_INTR	COREx_GDMA_INT_MAP_REG	24	
25	*SPI Controller [to be added later]*	SPI2_INTR	COREx_SPI2_INT_MAP_REG	25	
26	*SPI Controller [to be added later]*	SPI3_INTR	COREx_SPI3_INT_MAP_REG	26	
27	*I2S Controller (I2S)*	I2S0_INTR	COREx_I2S0_INT_MAP_REG	27	
28	*I2S Controller (I2S)*	I2S1_INR	COREx_I2S1_INT_MAP_REG	28	
29	*I2S Controller (I2S)*	I2S2_INTR_0	COREx_I2S2_INT_MAP_REG	29	
30	*UART Controller (UART)*	UHCI0_INTR	COREx_UHCI0_INT_MAP_REG	30	
31	*UART Controller (UART)*	UART0_INTR	COREx_UART0_INT_MAP_REG	31	

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
32	*UART Controller (UART)*	UART1_INTR	CORE*x*_UART1_INT_MAP_REG	0	CORE*x*_INTR_STATUS_REG_1_REG
33	*UART Controller (UART)*	UART2_INTR	CORE*x*_UART2_INT_MAP_REG	1	
34	*UART Controller (UART)*	UART3_INTR	CORE*x*_UART3_INT_MAP_REG	2	
35	*UART Controller (UART)*	UART4_INTR	CORE*x*_UART4_INT_MAP_REG	3	
36	*LCD and Camera Controller (LCD_CAM)*	LCD_CAM_INTR	CORE*x*_LCD_CAM_INT_MAP_REG	4	
37	*ADC Controller (ADC)*	ADC_INTR	CORE*x*_ADC_INT_MAP_REG	5	
38	*Motor Control PWM (MCPWM)*	PWM0_INTR	CORE*x*_PWM0_INT_MAP_REG	6	
39	*Motor Control PWM (MCPWM)*	PWM1_INTR	CORE*x*_PWM1_INT_MAP_REG	7	
40	*Two-Wire Automotive Interface (TWAI)*	TWAI0_INTR	CORE*x*_TWAI0_INT_MAP_REG	8	
41	*Two-Wire Automotive Interface (TWAI)*	TWAI1_INTR	CORE*x*_TWAI1_INT_MAP_REG	9	
42	*Two-Wire Automotive Interface (TWAI)*	TWAI2_INTR	CORE*x*_TWAI2_INT_MAP_REG	10	
43	*Remote Control Peripheral (RMT)*	RMT_INTR	CORE*x*_RMT_INT_MAP_REG	11	
44	*I2C Controller (I2C)*	I2C0_INTR	CORE*x*_I2C0_INT_MAP_REG	12	
45	*I2C Controller (I2C)*	I2C1_INTR	CORE*x*_I2C1_INT_MAP_REG	13	
46	*Timer Group (TIMG)*	TIMERGRP0_T0_INTR	CORE*x*_TIMRTGRP0_T0_INT_MAP_REG	14	
47	*Timer Group (TIMG)*	TIMERGRP0_T1_INTR	CORE*x*_TIMERGRP0_T1_INT_MAP_REG	15	
48	*Timer Group (TIMG)*	TIMERGRP0_WDT_INTR	CORE*x*_TIMERGRP0_WDT_INT_MAP_REG	16	
49	*Timer Group (TIMG)*	TIMERGRP1_T0_INTR	CORE*x*_TIMRTGRP1_T0_INT_MAP_REG	17	
50	*Timer Group (TIMG)*	TIMERGRP1_T1_INTR	CORE*x*_TIMERGRP1_T1_INT_MAP_REG	18	
51	*Timer Group (TIMG)*	TIMERGRP1_WDT_INTR	CORE*x*_TIMERGRP1_WDT_INT_MAP_REG	19	
52	*LED PWM Controller (LEDC)*	LEDC_INTR	CORE*x*_LEDC_INTR	20	
53	*System Timer*	SYSTIMER_TARGET0_INTR	CORE*x*_SYSTIMER_TARGET0_INTR	21	
54	*System Timer*	SYSTIMER_TARGET1_INTR	CORE*x*_SYSTIMER_TARGET1_INTR	22	
55	*System Timer*	SYSTIMER_TARGET2_INTR	CORE*x*_SYSTIMER_TARGET2_INTR	23	
56	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH0_INTR	CORE*x*_AHB_PDMA_IN_CH0_INT_MAP_REG	24	
57	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH1_INTR	CORE*x*_AHB_PDMA_IN_CH1_INT_MAP_REG	25	
58	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_IN_CH2_INTR	CORE*x*_AHB_PDMA_IN_CH2_INT_MAP_REG	26	
59	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH0_INTR	CORE*x*_AHB_PDMA_OUT_CH0_INT_MAP_REG	27	
60	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH1_INTR	CORE*x*_AHB_PDMA_OUT_CH1_INT_MAP_REG	28	
61	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AHB_PDMA_OUT_CH2_INTR	CORE*x*_AHB_PDMA_OUT_CH2_INT_MAP_REG	29	
62	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH0_INTR	CORE*x*_AXI_PDMA_IN_CH0_INT_MAP_REG	30	
63	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH1_INTR	CORE*x*_AXI_PDMA_IN_CH1_INT_MAP_REG	31	

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
64	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_IN_CH2_INTR	COREx_AXI_PDMA_IN_CH2_INT_MAP_REG	0	
65	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH0_INTR	COREx_AXI_PDMA_OUT_CH0_INT_MAP_REG	1	
66	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH1_INTR	COREx_AXI_PDMA_OUT_CH1_INT_MAP_REG	2	
67	*GDMA Controller (GDMA-AHB, GDMA-AXI)*	AXI_PDMA_OUT_CH2_INTR	COREx_AXI_PDMA_OUT_CH2_INT_MAP_REG	3	
68	*RSA Accelerator (RSA)*	RSA_INTR	COREx_RSA_INT_MAP_REG	4	
69	*AES Accelerator (AES)*	AES_INTR	COREx_AES_INT_MAP_REG	5	
70	*SHA Accelerator (SHA)*	SHA_INTR	COREx_SHA_INT_MAP_REG	6	COREx_INTR_STATUS_REG_2_REG
71	*SHA Accelerator (SHA)*	ECC_INTR	COREx_ECC_INT_MAP_REG	7	
72	*Elliptic Curve Digital Signature Algorithm (ECDSA)*	ECDSA_INTR	COREx_ECDSA_INT_MAP_REG	8	
73	*Key Manager [to be added later]*	KM_INTR	COREx_KM_INT_MAP_REG	9	
74	*GPIO Matrix and IO MUX*	GPIO_INTR0	COREx_GPIO_INTR0_MAP_REG	10	
75	*GPIO Matrix and IO MUX*	GPIO_INTR1	COREx_GPIO_INTR1_MAP_REG	11	
76	*GPIO Matrix and IO MUX*	GPIO_INTR2	COREx_GPIO_INTR2_MAP_REG	12	
77	*GPIO Matrix and IO MUX*	GPIO_INTR3	COREx_GPIO_INTR3_MAP_REG	13	
78	*GPIO Matrix and IO MUX*	GPIO_PAD_COMP_INTR	COREx_GPIO_PAD_COMP_INT_MAP_REG	14	
79	*System Registers [to be added later]*	CPU_INTR_FROM_CPU_0	COREx_CPU_INTR_FROM_CPU_0_MAP_REG	15	
80	*System Registers [to be added later]*	CPU_INTR_FROM_CPU_1	COREx_CPU_INTR_FROM_CPU_1_MAP_REG	16	
81	*System Registers [to be added later]*	CPU_INTR_FROM_CPU_2	COREx_CPU_INTR_FROM_CPU_2_MAP_REG	17	
82	*System Registers [to be added later]*	CPU_INTR_FROM_CPU_3	COREx_CPU_INTR_FROM_CPU_3_MAP_REG	18	
83	*Cache [to be added later]*	CACHE_INTR	COREx_CACHE_INT_MAP_REG	19	
84	*SPI Controller [to be added later]*	FLASH_MSPI_INTR	COREx_FLASH_MSPI_INT_MAP_REG	20	
85	*MIPI CSI [to be added later]*	CSI_BRIDGE_INTR	COREx_CSI_BRIDGE_INT_MAP_REG	21	
86	*MIPI DSI [to be added later]*	DSI_BRIDGE_INTR	COREx_DSI_BRIDGE_INT_MAP_REG	22	
87	*MIPI CSI [to be added later]*	CSI_INTR	COREx_CSI_INT_MAP_REG	23	
88	*MIPI DSI [to be added later]*	DSI_INTR	COREx_DSI_INT_MAP_REG	24	
89	*Ethernet Media Access Controller (EMAC)*	GMII_PHY_INTR	COREx_GMII_PHY_INT_MAP_REG	25	
90	*Ethernet Media Access Controller (EMAC)*	LPI_INTR	COREx_LPI_INT_MAP_REG	26	
91	*Ethernet Media Access Controller (EMAC)*	PMT_INTR	COREx_PMT_INT_MAP_REG	27	
92	*Ethernet Media Access Controller (EMAC)*	ETH_MAC_INTR	COREx_ETH_MAC_INT_MAP_REG	28	
93	*USB 2.0 OTG High-Speed [to be added later]*	USB_OTG_INTR	COREx_USB_OTG_INT_MAP_REG	29	
94	*USB 2.0 OTG High-Speed [to be added later]*	USB_OTG_ENDP_MULTI_PROC_INTR	COREx_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	30	
95	*JPEG Codec*	JPEG_INTR	COREx_JPEG_INT_MAP_REG	31	
96	*Pixel-Processing Accelerator (PPA)*	PPA_INTR	COREx_PPA_INT_MAP_REG	0	
97	*RISC-V Trace Encoder (TRACE)*	COREO_TRACE_INTR	COREx_COREO_TRACE_INT_MAP_REG	1	

COREx_INTR_STATUS_REG_3_REG

No.	Chapter	Interrupt Source	Interrupt Source Mapping Register	Interrupt Status Register	
				Bit	Name
98	*RISC-V Trace Encoder (TRACE)*	CORE1_TRACE_INTR	CORE*x*_CORE1_TRACE_INT_MAP_REG	2	
99	n/a	reserved	reserved	3	
100	*Image Signal Processor [to be added later]*	ISP_INTR	CORE*x*_ISP_INT_MAP_REG	4	
101	*I3C Controller [to be added later]*	I3C_MST_INTR	CORE*x*_I3C_MST_INT_MAP_REG	5	
102	*I3C Controller [to be added later]*	I3C_SLV_INTR	CORE*x*_I3C_SLV_INT_MAP_REG	6	
103	*USB 2.0 OTG Full-Speed [to be added later]*	USB_OTG11_INTR	CORE*x*_USB_OTG11_INT_MAP_REG	7	
104	*2D-DMA Controller (2D-DMA)*	DMA2D_IN_CH0_INTR	CORE*x*_DMA2D_IN_CH0_INT_MAP_REG	8	
105	*2D-DMA Controller (2D-DMA)*	DMA2D_IN_CH1_INTR	CORE*x*_DMA2D_IN_CH1_INT_MAP_REG	9	
106	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH0_INTR	CORE*x*_DMA2D_OUT_CH0_INT_MAP_REG	10	
107	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH1_INTR	CORE*x*_DMA2D_OUT_CH1_INT_MAP_REG	11	
108	*2D-DMA Controller (2D-DMA)*	DMA2D_OUT_CH2_INTR	CORE*x*_DMA2D_OUT_CH2_INT_MAP_REG	12	
109	*SPI Controller [to be added later]*	PSRAM_MSPI_INTR	CORE*x*_PSRAM_MSPI_INT_MAP_REG	13	
110	*System Registers [to be added later]*	HP_SYSREG_INTR	CORE*x*_HP_SYSREG_INT_MAP_REG	14	
111	*Pulse Count Controller (PCNT)*	PCNT_INTR	CORE*x*_PCNT_INT_MAP_REG	15	
112	n/a	reserved	reserved	16	
113	*Parallel IO Controller (PARLIO)*	HP_PARLIO_RX_INTR	CORE*x*_HP_PARLIO_RX_INT_MAP_REG	17	
114	*Parallel IO Controller (PARLIO)*	HP_PARLIO_TX_INTR	CORE*x*_HP_PARLIO_TX_INT_MAP_REG	18	
115	*H264 Encoder [to be added later]*	H264_DMA2D_OUT_CH0_INTR	CORE*x*_H264_DMA2D_OUT_CH0_INT_MAP_REG	19	
116	*H264 Encoder [to be added later]*	H264_DMA2D_OUT_CH1_INTR	CORE*x*_H264_DMA2D_OUT_CH1_INT_MAP_REG	20	
117	*H264 Encoder [to be added later]*	H264_DMA2D_OUT_CH2_INTR	CORE*x*_H264_DMA2D_OUT_CH2_INT_MAP_REG	21	
118	*H264 Encoder [to be added later]*	H264_DMA2D_OUT_CH3_INTR	CORE*x*_H264_DMA2D_OUT_CH3_INT_MAP_REG	22	
119	*H264 Encoder [to be added later]*	H264_DMA2D_OUT_CH4_INTR	CORE*x*_H264_DMA2D_OUT_CH4_INT_MAP_REG	23	
120	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH0_INTR	CORE*x*_H264_DMA2D_IN_CH0_INT_MAP_REG	24	
121	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH1_INTR	CORE*x*_H264_DMA2D_IN_CH1_INT_MAP_REG	25	
122	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH2_INTR	CORE*x*_H264_DMA2D_IN_CH2_INT_MAP_REG	26	
123	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH3_INTR	CORE*x*_H264_DMA2D_IN_CH3_INT_MAP_REG	27	
124	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH4_INTR	CORE*x*_H264_DMA2D_IN_CH4_INT_MAP_REG	28	
125	*H264 Encoder [to be added later]*	H264_DMA2D_IN_CH5_INTR	CORE*x*_H264_DMA2D_IN_CH5_INT_MAP_REG	29	
126	*H264 Encoder [to be added later]*	H264_REG_INTR	CORE*x*_H264_REG_INT_MAP_REG	30	
127	*Debug Assistant*	ASSIST_DEBUG_INTR	CORE*x*_ASSIST_DEBUG_INT_MAP_REG	31	

10.4.2 Assign Peripheral Interrupt Source to HP CPU Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- Source_*Y*: stands for a peripheral interrupt source, wherein *Y* means the number of this interrupt source in Table 10.4-1.
- CORE*x*_SOURCE_*Y*_MAP_REG: stands for an interrupt source mapping register for the peripheral interrupt source (Source_*Y*) of HP CPU*x*.
- Num_P: the index of HP CPU interrupts which can be 16 ~ 47.
- Interrupt_P: stands for the HP CPU interrupt numbered as Num_P.

10.4.2.1 Assign One Peripheral Interrupt Source (Source_*Y*) to HP CPU*x*

Setting the corresponding source mapping register CORE*x*_SOURCE_*Y*_MAP_REG of Source_*Y* to Num_P assigns this interrupt source to Interrupt_P.

10.4.2.2 Assign Multiple Peripheral Interrupt Sources (Source_*Y*) to HP CPU*x*

Setting the corresponding source mapping register CORE*x*_SOURCE_*Y*_MAP_REG of each interrupt source to the same Num_P assigns multiple sources to the same Interrupt_P of HP CPU*x*. Any of these sources can trigger CPU Interrupt_P of HP CPU*x*. In other words, the CPU Interrupt_p will be shared between these interrupt sources. When an interrupt signal is generated, HP CPU*x* should check the interrupt status registers to figure out which peripheral generated the interrupt. For more information, see Chapter 1 *High-Performance CPU [to be added later]*.

10.4.2.3 Disable HP CPU*x* Peripheral Interrupt Source (Source_*x*)

Writing 0 to the CORE*x*_SOURCE_*Y*_MAP_REG register disables the corresponding interrupt source.

10.4.3 Query Current Interrupt Status of HP CPU*x* Peripheral Interrupt Source

After enabling peripheral interrupt sources, users can query current interrupt status of HP CPU*x* peripheral interrupt source by reading the bit value in CORE*x*_INTR_STATUS_*n* (read only). For the mapping between CORE*x*_INTR_STATUS_*n* and peripheral interrupt sources, please refer to Table 10.4-1.

10.5 Register Summary

The addresses in this section are relative to the interrupt matrix HP CPU0/HP CPU1 base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

10.5.1 HP CPU0 Interrupt Matrix Register Summary

Name	Description	Address	Access
COREO LP RTC INT MAP REG			
COREO_LP_RTC_INT_MAP_REG	LP_RTC_INTR mapping register	0x0000	R/W
COREO LP WDT INT MAP REG			
COREO_LP_WDT_INT_MAP_REG	LP_WDT_INTR mapping register	0x0004	R/W
COREO LP TIMER REG 0 INT MAP REG			
COREO_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR mapping register	0x0008	R/W
COREO LP TIMER REG 1 INT MAP REG			
COREO_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR mapping register	0x000C	R/W
COREO MB HP INT MAP REG			
COREO_MB_HP_INT_MAP_REG	MB_HP_INTR mapping register	0x0010	R/W
COREO MB LP INT MAP REG			
COREO_MB_LP_INT_MAP_REG	MB_LP_INTR mapping register	0x0014	R/W
COREO PMU REG 0 INT MAP REG			
COREO_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR mapping register	0x0018	R/W
COREO PMU REG 1 INT MAP REG			
COREO_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR mapping register	0x001C	R/W
COREO LP ANAPERI INT MAP REG			
COREO_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR mapping register	0x0020	R/W
COREO LP ADC INT MAP REG			
COREO_LP_ADC_INT_MAP_REG	LP_ADC_INTR mapping register	0x0024	R/W
COREO LP GPIO INT MAP REG			

GoBack

Name	Description	Address	Access
COREO_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR mapping register	0x0028	R/W
COREO LP I2C INT MAP REG			
COREO_LP_I2C_INT_MAP_REG	LP_I2C_INTR mapping register	0x002C	R/W
COREO LP I2S INT MAP REG			
COREO_LP_I2S_INT_MAP_REG	LP_I2S_INTR mapping register	0x0030	R/W
COREO LP SPI INT MAP REG			
COREO_LP_SPI_INT_MAP_REG	LP_SPI_INTR mapping register	0x0034	R/W
COREO LP TOUCH INT MAP REG			
COREO_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR mapping register	0x0038	R/W
COREO LP TSENS INT MAP REG			
COREO_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR mapping register	0x003C	R/W
COREO LP UART INT MAP REG			
COREO_LP_UART_INT_MAP_REG	LP_UART_INTR mapping register	0x0040	R/W
COREO LP EFUSE INT MAP REG			
COREO_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR mapping register	0x0044	R/W
COREO LP SW INT MAP REG			
COREO_LP_SW_INT_MAP_REG	LP_SW_INTR mapping register	0x0048	R/W
COREO LP SYSREG INT MAP REG			
COREO_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR mapping register	0x004C	R/W
COREO LP HUK INT MAP REG			
COREO_LP_HUK_INT_MAP_REG	LP_HUK_INTR mapping register	0x0050	R/W
COREO SYS ICM INT MAP REG			
COREO_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR mapping register	0x0054	R/W
COREO USB DEVICE INT MAP REG			
COREO_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR mapping register	0x0058	R/W
COREO SDIO HOST INT MAP REG			
COREO_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR mapping register	0x005C	R/W
COREO GDMA INT MAP REG			
COREO_GDMA_INT_MAP_REG	GDMA_INTR mapping register	0x0060	R/W

GoBack

Name	Description	Address	Access
COREO SPI2 INT MAP REG			
COREO_SPI2_INT_MAP_REG	SPI2_INTR mapping register	0x0064	R/W
COREO SPI3 INT MAP REG			
COREO_SPI3_INT_MAP_REG	SPI3_INTR mapping register	0x0068	R/W
COREO I2S0 INT MAP REG			
COREO_I2S0_INT_MAP_REG	I2S0_INTR mapping register	0x006C	R/W
COREO I2S1 INT MAP REG			
COREO_I2S1_INT_MAP_REG	I2S1_INTR mapping register	0x0070	R/W
COREO I2S2 INT MAP REG			
COREO_I2S2_INT_MAP_REG	I2S2_INTR mapping register	0x0074	R/W
COREO UHCI0 INT MAP REG			
COREO_UHCI0_INT_MAP_REG	UHCI0_INTR mapping register	0x0078	R/W
COREO UART0 INT MAP REG			
COREO_UART0_INT_MAP_REG	UART0_INTR mapping register	0x007C	R/W
COREO UART1 INT MAP REG			
COREO_UART1_INT_MAP_REG	UART1_INTR mapping register	0x0080	R/W
COREO UART2 INT MAP REG			
COREO_UART2_INT_MAP_REG	UART2_INTR mapping register	0x0084	R/W
COREO UART3 INT MAP REG			
COREO_UART3_INT_MAP_REG	UART3_INTR mapping register	0x0088	R/W
COREO UART4 INT MAP REG			
COREO_UART4_INT_MAP_REG	UART4_INTR mapping register	0x008C	R/W
COREO LCD CAM INT MAP REG			
COREO_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR mapping register	0x0090	R/W
COREO ADC INT MAP REG			
COREO_ADC_INT_MAP_REG	ADC_INTR mapping register	0x0094	R/W
COREO PWM0 INT MAP REG			
COREO_PWM0_INT_MAP_REG	PWM0_INTR mapping register	0x0098	R/W
COREO PWM1 INT MAP REG			

Name	Description	Address	Access
COREO_PWM1_INT_MAP_REG	PWM1_INTR mapping register	0x009C	R/W
COREO TWAI0 INT MAP REG			
COREO_TWAI0_INT_MAP_REG	TWAI0_INTR mapping register	0x00A0	R/W
COREO TWAI1 INT MAP REG			
COREO_TWAI1_INT_MAP_REG	TWAI1_INTR mapping register	0x00A4	R/W
COREO TWAI2 INT MAP REG			
COREO_TWAI2_INT_MAP_REG	TWAI2_INTR mapping register	0x00A8	R/W
COREO RMT INT MAP REG			
COREO_RMT_INT_MAP_REG	RMT_INTR mapping register	0x00AC	R/W
COREO I2C0 INT MAP REG			
COREO_I2C0_INT_MAP_REG	I2C0_INTR mapping register	0x00B0	R/W
COREO I2C1 INT MAP REG			
COREO_I2C1_INT_MAP_REG	I2C1_INTR mapping register	0x00B4	R/W
COREO TIMERGRP0 T0 INT MAP REG			
COREO_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR mapping register	0x00B8	R/W
COREO TIMERGRP0 T1 INT MAP REG			
COREO_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR mapping register	0x00BC	R/W
COREO TIMERGRP0 WDT INT MAP REG			
COREO_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR mapping register	0x00C0	R/W
COREO TIMERGRP1 T0 INT MAP REG			
COREO_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR mapping register	0x00C4	R/W
COREO TIMERGRP1 T1 INT MAP REG			
COREO_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR mapping register	0x00C8	R/W
COREO TIMERGRP1 WDT INT MAP REG			
COREO_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR mapping register	0x00CC	R/W
COREO LEDC INT MAP REG			
COREO_LEDC_INT_MAP_REG	LEDC_INTR mapping register	0x00D0	R/W
COREO SYSTIMER TARGET0 INT MAP REG			
COREO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR mapping register	0x00D4	R/W

Name	Description	Address	Access
COREO SYSTIMER TARGET1 INT MAP REG			
COREO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR mapping register	0x00D8	R/W
COREO SYSTIMER TARGET2 INT MAP REG			
COREO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR mapping register	0x00DC	R/W
COREO AHB PDMA IN CH0 INT MAP REG			
COREO_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR mapping register	0x00E0	R/W
COREO AHB PDMA IN CH1 INT MAP REG			
COREO_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR mapping register	0x00E4	R/W
COREO AHB PDMA IN CH2 INT MAP REG			
COREO_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR mapping register	0x00E8	R/W
COREO AHB PDMA OUT CH0 INT MAP REG			
COREO_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR mapping register	0x00EC	R/W
COREO AHB PDMA OUT CH1 INT MAP REG			
COREO_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR mapping register	0x00F0	R/W
COREO AHB PDMA OUT CH2 INT MAP REG			
COREO_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR mapping register	0x00F4	R/W
COREO AXI PDMA IN CH0 INT MAP REG			
COREO_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR mapping register	0x00F8	R/W
COREO AXI PDMA IN CH1 INT MAP REG			
COREO_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR mapping register	0x00FC	R/W
COREO AXI PDMA IN CH2 INT MAP REG			
COREO_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR mapping register	0x0100	R/W
COREO AXI PDMA OUT CH0 INT MAP REG			
COREO_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR mapping register	0x0104	R/W
COREO AXI PDMA OUT CH1 INT MAP REG			
COREO_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR mapping register	0x0108	R/W
COREO AXI PDMA OUT CH2 INT MAP REG			
COREO_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR mapping register	0x010C	R/W
COREO RSA INT MAP REG			

Name	Description	Address	Access
COREO_RSA_INT_MAP_REG	RSA_INTR mapping register	0x0110	R/W
COREO AES INT MAP REG			
COREO_AES_INT_MAP_REG	AES_INTR mapping register	0x0114	R/W
COREO SHA INT MAP REG			
COREO_SHA_INT_MAP_REG	SHA_INTR mapping register	0x0118	R/W
COREO ECC INT MAP REG			
COREO_ECC_INT_MAP_REG	ECC_INTR mapping register	0x011C	R/W
COREO ECDSA INT MAP REG			
COREO_ECDSA_INT_MAP_REG	ECDSA_INTR mapping register	0x0120	R/W
COREO KM INT MAP REG			
COREO_KM_INT_MAP_REG	KM_INTR mapping register	0x0124	R/W
COREO GPIO INT0 MAP REG			
COREO_GPIO_INT0_MAP_REG	GPIO_INTR0 mapping register	0x0128	R/W
COREO GPIO INT1 MAP REG			
COREO_GPIO_INT1_MAP_REG	GPIO_INTR1 mapping register	0x012C	R/W
COREO GPIO INT2 MAP REG			
COREO_GPIO_INT2_MAP_REG	GPIO_INTR2 mapping register	0x0130	R/W
COREO GPIO INT3 MAP REG			
COREO_GPIO_INT3_MAP_REG	GPIO_INTR3 mapping register	0x0134	R/W
COREO GPIO PAD COMP INT MAP REG			
COREO_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR mapping register	0x0138	R/W
COREO CPU INT FROM CPU 0 MAP REG			
COREO_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x013C	R/W
COREO CPU INT FROM CPU 1 MAP REG			
COREO_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0140	R/W
COREO CPU INT FROM CPU 2 MAP REG			
COREO_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0144	R/W
COREO CPU INT FROM CPU 3 MAP REG			
COREO_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x0148	R/W

Name	Description	Address	Access
COREO CACHE INT MAP REG			
COREO_CACHE_INT_MAP_REG	CACHE_INTR mapping register	0x014C	R/W
COREO FLASH MSPI INT MAP REG			
COREO_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR mapping register	0x0150	R/W
COREO CSI BRIDGE INT MAP REG			
COREO_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR mapping register	0x0154	R/W
COREO DSI BRIDGE INT MAP REG			
COREO_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR mapping register	0x0158	R/W
COREO CSI INT MAP REG			
COREO_CSI_INT_MAP_REG	CSI_INTR mapping register	0x015C	R/W
COREO DSI INT MAP REG			
COREO_DSI_INT_MAP_REG	DSI_INTR mapping register	0x0160	R/W
COREO GMII PHY INT MAP REG			
COREO_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR mapping register	0x0164	R/W
COREO LPI INT MAP REG			
COREO_LPI_INT_MAP_REG	LPI_INTR mapping register	0x0168	R/W
COREO PMT INT MAP REG			
COREO_PMT_INT_MAP_REG	PMT_INTR mapping register	0x016C	R/W
COREO SBD INT MAP REG			
COREO_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR mapping register	0x0170	R/W
COREO USB OTG INT MAP REG			
COREO_USB_OTG_INT_MAP_REG	USB_OTG_INTR mapping register	0x0174	R/W
COREO USB OTG ENDP MULTI PROC INT MAP REG			
COREO_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR mapping register	0x0178	R/W
COREO JPEG INT MAP REG			
COREO_JPEG_INT_MAP_REG	JPEG_INTR mapping register	0x017C	R/W
COREO PPA INT MAP REG			
COREO_PPA_INT_MAP_REG	PPA_INTR mapping register	0x0180	R/W

Name	Description	Address	Access
COREO COREO TRACE INT MAP REG			
COREO_COREO_TRACE_INT_MAP_REG	COREO_TRACE_INTR mapping register	0x0184	R/W
COREO CORE1 TRACE INT MAP REG			
COREO_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR mapping register	0x0188	R/W
COREO ISP INT MAP REG			
COREO_ISP_INT_MAP_REG	ISP_INTR mapping register	0x0190	R/W
COREO I3C MST INT MAP REG			
COREO_I3C_MST_INT_MAP_REG	I3C_MST_INTR mapping register	0x0194	R/W
COREO I3C SLV INT MAP REG			
COREO_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR mapping register	0x0198	R/W
COREO USB OTG11 INT MAP REG			
COREO_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR mapping register	0x019C	R/W
COREO DMA2D IN CH0 INT MAP REG			
COREO_DMA2D_IN_CH0_INT_MAP_REG	DMA2D_IN_CH0_INTR mapping register	0x01A0	R/W
COREO DMA2D IN CH1 INT MAP REG			
COREO_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR mapping register	0x01A4	R/W
COREO DMA2D OUT CH0 INT MAP REG			
COREO_DMA2D_OUT_CH0_INT_MAP_REG	DMA2D_OUT_CH0_INTR mapping register	0x01A8	R/W
COREO DMA2D OUT CH1 INT MAP REG			
COREO_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR mapping register	0x01AC	R/W
COREO DMA2D OUT CH2 INT MAP REG			
COREO_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR mapping register	0x01B0	R/W
COREO PSRAM MSPI INT MAP REG			
COREO_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR mapping register	0x01B4	R/W
COREO HP SYSREG INT MAP REG			
COREO_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR mapping register	0x01B8	R/W
COREO PCNT INT MAP REG			
COREO_PCNT_INT_MAP_REG	PCNT_INTR mapping register	0x01BC	R/W
COREO HP PARLIO RX INT MAP REG			

Name	Description	Address	Access
COREO_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR mapping register	0x01C4	R/W
COREO HP PARLIO TX INT MAP REG			
COREO_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR mapping register	0x01C8	R/W
COREO H264 DMA2D OUT CH0 INT MAP REG			
COREO_H264_DMA2D_OUT_CH0_INT_MAP_REG	H264_DMA2D_OUT_CH0_INTR mapping register	0x01CC	R/W
COREO H264 DMA2D OUT CH1 INT MAP REG			
COREO_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR mapping register	0x01D0	R/W
COREO H264 DMA2D OUT CH2 INT MAP REG			
COREO_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR mapping register	0x01D4	R/W
COREO H264 DMA2D OUT CH3 INT MAP REG			
COREO_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR mapping register	0x01D8	R/W
COREO H264 DMA2D OUT CH4 INT MAP REG			
COREO_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR mapping register	0x01DC	R/W
COREO H264 DMA2D IN CH0 INT MAP REG			
COREO_H264_DMA2D_IN_CH0_INT_MAP_REG	H264_DMA2D_IN_CH0_INTR mapping register	0x01E0	R/W
COREO H264 DMA2D IN CH1 INT MAP REG			
COREO_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR mapping register	0x01E4	R/W
COREO H264 DMA2D IN CH2 INT MAP REG			
COREO_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR mapping register	0x01E8	R/W
COREO H264 DMA2D IN CH3 INT MAP REG			
COREO_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR mapping register	0x01EC	R/W
COREO H264 DMA2D IN CH4 INT MAP REG			
COREO_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR mapping register	0x01F0	R/W
COREO H264 DMA2D IN CH5 INT MAP REG			

Name	Description	Address	Access
COREO_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR mapping register	0x01F4	R/W
COREO H264 REG INT MAP REG			
COREO_H264_REG_INT_MAP_REG	H264_REG_INTR mapping register	0x01F8	R/W
COREO ASSIST DEBUG INT MAP REG			
COREO_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x01FC	R/W
COREO INTR STATUS REG 0 REG			
COREO_INTR_STATUS_REG_0_REG	Status register for interrupt sources 0 ~ 31	0x0200	RO
COREO INTR STATUS REG 1 REG			
COREO_INTR_STATUS_REG_1_REG	Status register for interrupt sources 32 ~ 63	0x0204	RO
COREO INTR STATUS REG 2 REG			
COREO_INTR_STATUS_REG_2_REG	Status register for interrupt sources 64 ~ 95	0x0208	RO
COREO INTR STATUS REG 3 REG			
COREO_INTR_STATUS_REG_3_REG	Status register for interrupt sources 96 ~ 127	0x020C	RO
COREO CLOCK GATE REG			
COREO_CLOCK_GATE_REG	Clock gating register	0x0210	R/W
COREO INTERRUPT REG DATE REG			
COREO_INTERRUPT_REG_DATE_REG	Version control register	0x03FC	R/W

10.5.2 HP CPU1 Interrupt Matrix Register Summary

Name	Description	Address	Access
CORE1 LP RTC INT MAP REG			
CORE1_LP_RTC_INT_MAP_REG	LP_RTC_INTR mapping register	0x0000	R/W
CORE1 LP WDT INT MAP REG			
CORE1_LP_WDT_INT_MAP_REG	LP_WDT_INTR mapping register	0x0004	R/W
CORE1 LP TIMER REG 0 INT MAP REG			
CORE1_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR mapping register	0x0008	R/W
CORE1 LP TIMER REG 1 INT MAP REG			

Name	Description	Address	Access
CORE1_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR mapping register	0x000C	R/W
CORE1 MB HP INT MAP REG			
CORE1_MB_HP_INT_MAP_REG	MB_HP_INTR mapping register	0x0010	R/W
CORE1 MB LP INT MAP REG			
CORE1_MB_LP_INT_MAP_REG	MB_LP_INTR mapping register	0x0014	R/W
CORE1 PMU REG 0 INT MAP REG			
CORE1_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR mapping register	0x0018	R/W
CORE1 PMU REG 1 INT MAP REG			
CORE1_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR mapping register	0x001C	R/W
CORE1 LP ANAPERI INT MAP REG			
CORE1_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR mapping register	0x0020	R/W
CORE1 LP ADC INT MAP REG			
CORE1_LP_ADC_INT_MAP_REG	LP_ADC_INTR mapping register	0x0024	R/W
CORE1 LP GPIO INT MAP REG			
CORE1_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR mapping register	0x0028	R/W
CORE1 LP I2C INT MAP REG			
CORE1_LP_I2C_INT_MAP_REG	LP_I2C_INTR mapping register	0x002C	R/W
CORE1 LP I2S INT MAP REG			
CORE1_LP_I2S_INT_MAP_REG	LP_I2S_INTR mapping register	0x0030	R/W
CORE1 LP SPI INT MAP REG			
CORE1_LP_SPI_INT_MAP_REG	LP_SPI_INTR mapping register	0x0034	R/W
CORE1 LP TOUCH INT MAP REG			
CORE1_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR mapping register	0x0038	R/W
CORE1 LP TSENS INT MAP REG			
CORE1_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR mapping register	0x003C	R/W
CORE1 LP UART INT MAP REG			
CORE1_LP_UART_INT_MAP_REG	LP_UART_INTR mapping register	0x0040	R/W
CORE1 LP EFUSE INT MAP REG			
CORE1_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR mapping register	0x0044	R/W

Name	Description	Address	Access
CORE1 LP SW INT MAP REG			
CORE1_LP_SW_INT_MAP_REG	LP_SW_INTR mapping register	0x0048	R/W
CORE1 LP SYSREG INT MAP REG			
CORE1_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR mapping register	0x004C	R/W
CORE1 LP HUK INT MAP REG			
CORE1_LP_HUK_INT_MAP_REG	LP_HUK_INTR mapping register	0x0050	R/W
CORE1 SYS ICM INT MAP REG			
CORE1_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR mapping register	0x0054	R/W
CORE1 USB DEVICE INT MAP REG			
CORE1_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR mapping register	0x0058	R/W
CORE1 SDIO HOST INT MAP REG			
CORE1_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR mapping register	0x005C	R/W
CORE1 GDMA INT MAP REG			
CORE1_GDMA_INT_MAP_REG	GDMA_INTR mapping register	0x0060	R/W
CORE1 SPI2 INT MAP REG			
CORE1_SPI2_INT_MAP_REG	SPI2_INTR mapping register	0x0064	R/W
CORE1 SPI3 INT MAP REG			
CORE1_SPI3_INT_MAP_REG	SPI3_INTR mapping register	0x0068	R/W
CORE1 I2S0 INT MAP REG			
CORE1_I2S0_INT_MAP_REG	I2S0_INTR mapping register	0x006C	R/W
CORE1 I2S1 INT MAP REG			
CORE1_I2S1_INT_MAP_REG	I2S1_INTR mapping register	0x0070	R/W
CORE1 I2S2 INT MAP REG			
CORE1_I2S2_INT_MAP_REG	I2S2_INTR mapping register	0x0074	R/W
CORE1 UHCI0 INT MAP REG			
CORE1_UHCI0_INT_MAP_REG	UHCI0_INTR mapping register	0x0078	R/W
CORE1 UART0 INT MAP REG			
CORE1_UART0_INT_MAP_REG	UART0_INTR mapping register	0x007C	R/W
CORE1 UART1 INT MAP REG			

Name	Description	Address	Access
CORE1_UART1_INT_MAP_REG	UART1_INTR mapping register	0x0080	R/W
CORE1 UART2 INT MAP REG			
CORE1_UART2_INT_MAP_REG	UART2_INTR mapping register	0x0084	R/W
CORE1 UART3 INT MAP REG			
CORE1_UART3_INT_MAP_REG	UART3_INTR mapping register	0x0088	R/W
CORE1 UART4 INT MAP REG			
CORE1_UART4_INT_MAP_REG	UART4_INTR mapping register	0x008C	R/W
CORE1 LCD CAM INT MAP REG			
CORE1_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR mapping register	0x0090	R/W
CORE1 ADC INT MAP REG			
CORE1_ADC_INT_MAP_REG	ADC_INTR mapping register	0x0094	R/W
CORE1 PWM0 INT MAP REG			
CORE1_PWM0_INT_MAP_REG	PWM0_INTR mapping register	0x0098	R/W
CORE1 PWM1 INT MAP REG			
CORE1_PWM1_INT_MAP_REG	PWM1_INTR mapping register	0x009C	R/W
CORE1 TWAI0 INT MAP REG			
CORE1_TWAI0_INT_MAP_REG	TWAI0_INTR mapping register	0x00A0	R/W
CORE1 TWAI1 INT MAP REG			
CORE1_TWAI1_INT_MAP_REG	TWAI1_INTR mapping register	0x00A4	R/W
CORE1 TWAI2 INT MAP REG			
CORE1_TWAI2_INT_MAP_REG	TWAI2_INTR mapping register	0x00A8	R/W
CORE1 RMT INT MAP REG			
CORE1_RMT_INT_MAP_REG	RMT_INTR mapping register	0x00AC	R/W
CORE1 I2C0 INT MAP REG			
CORE1_I2C0_INT_MAP_REG	I2C0_INTR mapping register	0x00B0	R/W
CORE1 I2C1 INT MAP REG			
CORE1_I2C1_INT_MAP_REG	I2C1_INTR mapping register	0x00B4	R/W
CORE1 TIMERGRP0 T0 INT MAP REG			
CORE1_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR mapping register	0x00B8	R/W

Name	Description	Address	Access
CORE1 TIMERGRP0 T1 INT MAP REG			
CORE1_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR mapping register	0x00BC	R/W
CORE1 TIMERGRP0 WDT INT MAP REG			
CORE1_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR mapping register	0x00C0	R/W
CORE1 TIMERGRP1 T0 INT MAP REG			
CORE1_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR mapping register	0x00C4	R/W
CORE1 TIMERGRP1 T1 INT MAP REG			
CORE1_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR mapping register	0x00C8	R/W
CORE1 TIMERGRP1 WDT INT MAP REG			
CORE1_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR mapping register	0x00CC	R/W
CORE1 LEDC INT MAP REG			
CORE1_LEDC_INT_MAP_REG	LEDC_INTR mapping register	0x00D0	R/W
CORE1 SYSTIMER TARGET0 INT MAP REG			
CORE1_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR mapping register	0x00D4	R/W
CORE1 SYSTIMER TARGET1 INT MAP REG			
CORE1_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR mapping register	0x00D8	R/W
CORE1 SYSTIMER TARGET2 INT MAP REG			
CORE1_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR mapping register	0x00DC	R/W
CORE1 AHB PDMA IN CH0 INT MAP REG			
CORE1_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR mapping register	0x00E0	R/W
CORE1 AHB PDMA IN CH1 INT MAP REG			
CORE1_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR mapping register	0x00E4	R/W
CORE1 AHB PDMA IN CH2 INT MAP REG			
CORE1_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR mapping register	0x00E8	R/W
CORE1 AHB PDMA OUT CH0 INT MAP REG			
CORE1_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR mapping register	0x00EC	R/W
CORE1 AHB PDMA OUT CH1 INT MAP REG			
CORE1_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR mapping register	0x00F0	R/W
CORE1 AHB PDMA OUT CH2 INT MAP REG			

GoBack

Name	Description	Address	Access
CORE1_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR mapping register	0x00F4	R/W
CORE1 AXI PDMA IN CH0 INT MAP REG			
CORE1_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR mapping register	0x00F8	R/W
CORE1 AXI PDMA IN CH1 INT MAP REG			
CORE1_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR mapping register	0x00FC	R/W
CORE1 AXI PDMA IN CH2 INT MAP REG			
CORE1_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR mapping register	0x0100	R/W
CORE1 AXI PDMA OUT CH0 INT MAP REG			
CORE1_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR mapping register	0x0104	R/W
CORE1 AXI PDMA OUT CH1 INT MAP REG			
CORE1_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR mapping register	0x0108	R/W
CORE1 AXI PDMA OUT CH2 INT MAP REG			
CORE1_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR mapping register	0x010C	R/W
CORE1 RSA INT MAP REG			
CORE1_RSA_INT_MAP_REG	RSA_INTR mapping register	0x0110	R/W
CORE1 AES INT MAP REG			
CORE1_AES_INT_MAP_REG	AES_INTR mapping register	0x0114	R/W
CORE1 SHA INT MAP REG			
CORE1_SHA_INT_MAP_REG	SHA_INTR mapping register	0x0118	R/W
CORE1 ECC INT MAP REG			
CORE1_ECC_INT_MAP_REG	ECC_INTR mapping register	0x011C	R/W
CORE1 ECDSA INT MAP REG			
CORE1_ECDSA_INT_MAP_REG	ECDSA_INTR mapping register	0x0120	R/W
CORE1 KM INT MAP REG			
CORE1_KM_INT_MAP_REG	KM_INTR mapping register	0x0124	R/W
CORE1 GPIO INT0 MAP REG			
CORE1_GPIO_INT0_MAP_REG	GPIO_INTR0 mapping register	0x0128	R/W
CORE1 GPIO INT1 MAP REG			
CORE1_GPIO_INT1_MAP_REG	GPIO_INTR1 mapping register	0x012C	R/W

GoBack

Name	Description	Address	Access
CORE1 GPIO INT2 MAP REG			
CORE1_GPIO_INT2_MAP_REG	GPIO_INTR2 mapping register	0x0130	R/W
CORE1 GPIO INT3 MAP REG			
CORE1_GPIO_INT3_MAP_REG	GPIO_INTR3 mapping register	0x0134	R/W
CORE1 GPIO PAD COMP INT MAP REG			
CORE1_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR mapping register	0x0138	R/W
CORE1 CPU INT FROM CPU 0 MAP REG			
CORE1_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x013C	R/W
CORE1 CPU INT FROM CPU 1 MAP REG			
CORE1_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0140	R/W
CORE1 CPU INT FROM CPU 2 MAP REG			
CORE1_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0144	R/W
CORE1 CPU INT FROM CPU 3 MAP REG			
CORE1_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x0148	R/W
CORE1 CACHE INT MAP REG			
CORE1_CACHE_INT_MAP_REG	CACHE_INTR mapping register	0x014C	R/W
CORE1 FLASH MSPI INT MAP REG			
CORE1_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR mapping register	0x0150	R/W
CORE1 CSI BRIDGE INT MAP REG			
CORE1_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR mapping register	0x0154	R/W
CORE1 DSI BRIDGE INT MAP REG			
CORE1_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR mapping register	0x0158	R/W
CORE1 CSI INT MAP REG			
CORE1_CSI_INT_MAP_REG	CSI_INTR mapping register	0x015C	R/W
CORE1 DSI INT MAP REG			
CORE1_DSI_INT_MAP_REG	DSI_INTR mapping register	0x0160	R/W
CORE1 GMII PHY INT MAP REG			
CORE1_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR mapping register	0x0164	R/W
CORE1 LPI INT MAP REG			

Name	Description	Address	Access
CORE1_LPI_INT_MAP_REG	LPI_INTR mapping register	0x0168	R/W
CORE1 PMT INT MAP REG			
CORE1_PMT_INT_MAP_REG	PMT_INTR mapping register	0x016C	R/W
CORE1 SBD INT MAP REG			
CORE1_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR mapping register	0x0170	R/W
CORE1 USB OTG INT MAP REG			
CORE1_USB_OTG_INT_MAP_REG	USB_OTG_INTR mapping register	0x0174	R/W
CORE1 USB OTG ENDP MULTI PROC INT MAP REG			
CORE1_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR mapping register	0x0178	R/W
CORE1 JPEG INT MAP REG			
CORE1_JPEG_INT_MAP_REG	JPEG_INTR mapping register	0x017C	R/W
CORE1 PPA INT MAP REG			
CORE1_PPA_INT_MAP_REG	PPA_INTR mapping register	0x0180	R/W
CORE1 COREO TRACE INT MAP REG			
CORE1_COREO_TRACE_INT_MAP_REG	COREO_TRACE_INTR mapping register	0x0184	R/W
CORE1 CORE1 TRACE INT MAP REG			
CORE1_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR mapping register	0x0188	R/W
CORE1 ISP INT MAP REG			
CORE1_ISP_INT_MAP_REG	ISP_INTR mapping register	0x0190	R/W
CORE1 I3C MST INT MAP REG			
CORE1_I3C_MST_INT_MAP_REG	I3C_MST_INTR mapping register	0x0194	R/W
CORE1 I3C SLV INT MAP REG			
CORE1_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR mapping register	0x0198	R/W
CORE1 USB OTG11 INT MAP REG			
CORE1_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR mapping register	0x019C	R/W
CORE1 DMA2D IN CHO INT MAP REG			
CORE1_DMA2D_IN_CHO_INT_MAP_REG	DMA2D_IN_CHO_INTR mapping register	0x01A0	R/W
CORE1 DMA2D IN CH1 INT MAP REG			

Name	Description	Address	Access
CORE1_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR mapping register	0x01A4	R/W
CORE1 DMA2D OUT CH0 INT MAP REG			
CORE1_DMA2D_OUT_CH0_INT_MAP_REG	DMA2D_OUT_CH0_INTR mapping register	0x01A8	R/W
CORE1 DMA2D OUT CH1 INT MAP REG			
CORE1_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR mapping register	0x01AC	R/W
CORE1 DMA2D OUT CH2 INT MAP REG			
CORE1_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR mapping register	0x01B0	R/W
CORE1 PSRAM MSPI INT MAP REG			
CORE1_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR mapping register	0x01B4	R/W
CORE1 HP SYSREG INT MAP REG			
CORE1_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR mapping register	0x01B8	R/W
CORE1 PCNT INT MAP REG			
CORE1_PCNT_INT_MAP_REG	PCNT_INTR mapping register	0x01BC	R/W
CORE1 HP PARLIO RX INT MAP REG			
CORE1_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR mapping register	0x01C4	R/W
CORE1 HP PARLIO TX INT MAP REG			
CORE1_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR mapping register	0x01C8	R/W
CORE1 H264 DMA2D OUT CH0 INT MAP REG			
CORE1_H264_DMA2D_OUT_CH0_INT_MAP_REG	H264_DMA2D_OUT_CH0_INTR mapping register	0x01CC	R/W
CORE1 H264 DMA2D OUT CH1 INT MAP REG			
CORE1_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR mapping register	0x01D0	R/W
CORE1 H264 DMA2D OUT CH2 INT MAP REG			
CORE1_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR mapping register	0x01D4	R/W
CORE1 H264 DMA2D OUT CH3 INT MAP REG			
CORE1_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR mapping register	0x01D8	R/W

Name	Description	Address	Access
CORE1 H264 DMA2D OUT CH4 INT MAP REG			
CORE1_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR mapping register	0x01DC	R/W
CORE1 H264 DMA2D IN CH0 INT MAP REG			
CORE1_H264_DMA2D_IN_CH0_INT_MAP_REG	H264_DMA2D_IN_CH0_INTR mapping register	0x01E0	R/W
CORE1 H264 DMA2D IN CH1 INT MAP REG			
CORE1_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR mapping register	0x01E4	R/W
CORE1 H264 DMA2D IN CH2 INT MAP REG			
CORE1_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR mapping register	0x01E8	R/W
CORE1 H264 DMA2D IN CH3 INT MAP REG			
CORE1_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR mapping register	0x01EC	R/W
CORE1 H264 DMA2D IN CH4 INT MAP REG			
CORE1_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR mapping register	0x01F0	R/W
CORE1 H264 DMA2D IN CH5 INT MAP REG			
CORE1_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR mapping register	0x01F4	R/W
CORE1 H264 REG INT MAP REG			
CORE1_H264_REG_INT_MAP_REG	H264_REG_INTR mapping register	0x01F8	R/W
CORE1 ASSIST DEBUG INT MAP REG			
CORE1_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x01FC	R/W
CORE1 INTR STATUS REG 0 REG			
CORE1_INTR_STATUS_REG_0_REG	Status register for interrupt sources 0 ~ 31	0x0200	RO
CORE1 INTR STATUS REG 1 REG			
CORE1_INTR_STATUS_REG_1_REG	Status register for interrupt sources 32 ~ 63	0x0204	RO
CORE1 INTR STATUS REG 2 REG			
CORE1_INTR_STATUS_REG_2_REG	Status register for interrupt sources 64 ~ 95	0x0208	RO
CORE1 INTR STATUS REG 3 REG			
CORE1_INTR_STATUS_REG_3_REG	Status register for interrupt sources 96 ~ 127	0x020C	RO
CORE1 CLOCK GATE REG			
CORE1_CLOCK_GATE_REG	Clock gating register	0x0210	R/W

GoBack

Name	Description	Address	Access
CORE1 INTERRUPT REG DATE REG			
CORE1_INTERRUPT_REG_DATE_REG	Version control register	0x03FC	R/W

10.6 Registers

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

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

10.6.1 HP CPU0 Interrupt Matrix Registers

Register 10.1. COREO_*LP_RTC_INT*_MAP_REG (0x0000)

Register 10.2. COREO_*LP_WDT_INT*_MAP_REG (0x0004)

Register 10.3. COREO_*LP_TIMER_REG_O_INT*_MAP_REG (0x0008)

Register 10.4. COREO_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 10.5. COREO_*MB_HP_INT*_MAP_REG (0x0010)

Register 10.6. COREO_*MB_LP_INT*_MAP_REG (0x0014)

Register 10.7. COREO_*PMU_REG_O_INT*_MAP_REG (0x0018)

Register 10.8. COREO_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 10.9. COREO_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 10.10. COREO_*LP_ADC_INT*_MAP_REG (0x0024)

Register 10.11. COREO_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 10.12. COREO_*LP_I2C_INT*_MAP_REG (0x002C)

Register 10.13. COREO_*LP_I2S_INT*_MAP_REG (0x0030)

Register 10.14. COREO_*LP_SPI_INT*_MAP_REG (0x0034)

Register 10.15. COREO_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 10.16. COREO_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 10.17. COREO_*LP_UART_INT*_MAP_REG (0x0040)

Register 10.18. COREO_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 10.19. COREO_*LP_SW_INT*_MAP_REG (0x0048)

Register 10.20. COREO_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 10.21. COREO_*LP_HUK_INT*_MAP_REG (0x0050)

Register 10.22. COREO_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 10.23. COREO_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 10.24. COREO_*SDIO_HOST_INT*_MAP_REG (0x005C)

Register 10.25. COREO_*GDMA_INT*_MAP_REG (0x0060)

Register 10.26. COREO_*SPI2_INT*_MAP_REG (0x0064)

Register 10.27. COREO_*SPI3_INT*_MAP_REG (0x0068)

Register 10.28. COREO_*I2SO_INT*_MAP_REG (0x006C)

Register 10.29. COREO_*I2S1_INT*_MAP_REG (0x0070)

Register 10.30. COREO_*I2S2_INT*_MAP_REG (0x0074)

Register 10.31. COREO_*UHCI0_INT*_MAP_REG (0x0078)

Register 10.32. COREO_*UART0_INT*_MAP_REG (0x007C)

Register 10.33. COREO_*UART1_INT*_MAP_REG (0x0080)

Register 10.34. COREO_*UART2_INT*_MAP_REG (0x0084)

Register 10.35. COREO_*UART3_INT*_MAP_REG (0x0088)

Register 10.36. COREO_*UART4_INT*_MAP_REG (0x008C)

Register 10.37. COREO_*LCD_CAM_INT*_MAP_REG (0x0090)

Register 10.38. COREO_*ADC_INT*_MAP_REG (0x0094)

Register 10.39. COREO_*PWM0_INT*_MAP_REG (0x0098)

Register 10.40. COREO_*PWM1_INT*_MAP_REG (0x009C)

Register 10.41. COREO_*TWAI0_INT*_MAP_REG (0x00A0)

Register 10.42. COREO_*TWAI1_INT*_MAP_REG (0x00A4)

Register 10.43. COREO_*TWAI2_INT*_MAP_REG (0x00A8)

Register 10.44. COREO_*RMT_INT*_MAP_REG (0x00AC)

Register 10.45. COREO_*I2C0_INT*_MAP_REG (0x00B0)

Register 10.46. COREO_*I2C1_INT*_MAP_REG (0x00B4)

Register 10.47. COREO_*TIMERGRP0_T0_INT*_MAP_REG (0x00B8)

Register 10.48. COREO_*TIMERGRP0_T1_INT*_MAP_REG (0x00BC)

Register 10.49. COREO_*TIMERGRP0_WDT_INT*_MAP_REG (0x00C0)

Register 10.50. COREO_*TIMERGRP1_T0_INT*_MAP_REG (0x00C4)

Register 10.51. COREO_*TIMERGRP1_T1_INT*_MAP_REG (0x00C8)

Register 10.52. COREO_*TIMERGRP1_WDT_INT*_MAP_REG (0x00CC)

Register 10.53. COREO_*LEDC_INT*_MAP_REG (0x00D0)

Register 10.54. COREO_*SYSTIMER_TARGET0_INT*_MAP_REG (0x00D4)

Register 10.55. COREO_*SYSTIMER_TARGET1_INT*_MAP_REG (0x00D8)

Register 10.56. COREO_*SYSTIMER_TARGET2_INT*_MAP_REG (0x00DC)

Register 10.57. COREO_*AHB_PDMA_IN_CH0_INT*_MAP_REG (0x00E0)

Register 10.58. COREO_*AHB_PDMA_IN_CH1_INT*_MAP_REG (0x00E4)

Register 10.59. COREO_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 10.60. COREO_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 10.61. COREO_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 10.62. COREO_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 10.63. COREO_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 10.64. COREO_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 10.65. COREO_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 10.66. COREO_*AXI_PDMA_OUT_CH0_INT*_MAP_REG (0x0104)

Register 10.67. COREO_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 10.68. COREO_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 10.69. COREO_*RSA_INT*_MAP_REG (0x0110)

Register 10.70. COREO_*AES_INT*_MAP_REG (0x0114)

Register 10.71. COREO_*SHA_INT*_MAP_REG (0x0118)

Register 10.72. COREO_*ECC_INT*_MAP_REG (0x011C)

Register 10.73. COREO_*ECDSA_INT*_MAP_REG (0x0120)

Register 10.74. COREO_*KM_INT*_MAP_REG (0x0124)

Register 10.75. COREO_*GPIO_INT0*_MAP_REG (0x0128)

Register 10.76. COREO_*GPIO_INT1*_MAP_REG (0x012C)

Register 10.77. COREO_*GPIO_INT2*_MAP_REG (0x0130)

Register 10.78. COREO_*GPIO_INT3*_MAP_REG (0x0134)

Register 10.79. COREO_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 10.80. COREO_*CPU_INT_FROM_CPU_0*_MAP_REG (0x013C)

Register 10.81. COREO_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 10.82. COREO_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 10.83. COREO_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 10.84. COREO_*CACHE_INT*_MAP_REG (0x014C)

Register 10.85. COREO_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 10.86. COREO_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 10.87. COREO_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 10.88. COREO_*CSI_INT*_MAP_REG (0x015C)

Register 10.89. COREO_*DSI_INT*_MAP_REG (0x0160)

Register 10.90. COREO_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 10.91. COREO_*LPI_INT*_MAP_REG (0x0168)

Register 10.92. COREO_*PMT_INT*_MAP_REG (0x016C)

Register 10.93. COREO_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 10.94. COREO_*USB_OTG_INT*_MAP_REG (0x0174)

Register 10.95. COREO_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 10.96. COREO_*JPEG_INT*_MAP_REG (0x017C)

Register 10.97. COREO_*PPA_INT*_MAP_REG (0x0180)

Register 10.98. COREO_*COREO_TRACE_INT*_MAP_REG (0x0184)

Register 10.99. COREO_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 10.100. COREO_*ISP_INT*_MAP_REG (0x0190)

Register 10.101. COREO_*I3C_MST_INT*_MAP_REG (0x0194)

Register 10.102. COREO_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 10.103. COREO_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 10.104. COREO_*DMA2D_IN_CH0_INT*_MAP_REG (0x01A0)

Register 10.105. COREO_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 10.106. COREO_*DMA2D_OUT_CH0_INT*_MAP_REG (0x01A8)

Register 10.107. COREO_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 10.108. COREO_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 10.109. COREO_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 10.110. COREO_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 10.111. COREO_*PCNT_INT*_MAP_REG (0x01BC)

Register 10.112. COREO_*HP_PAU_INT*_MAP_REG (0x01C0)

Register 10.113. COREO_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 10.114. COREO_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 10.115. COREO_*H264_DMA2D_OUT_CH0_INT*_MAP_REG (0x01CC)

Register 10.116. COREO_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 10.117. COREO_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 10.118. COREO_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 10.119. COREO_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 10.120. COREO_*H264_DMA2D_IN_CH0_INT*_MAP_REG (0x01E0)

Register 10.121. COREO_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 10.122. COREO_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 10.123. COREO_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

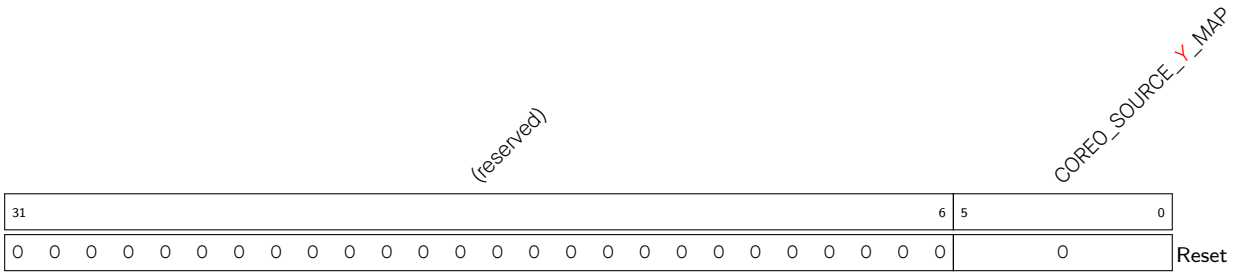
Register 10.124. COREO_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 10.125. COREO_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 10.126. COREO_*H264_REG_INT*_MAP_REG (0x01F8)

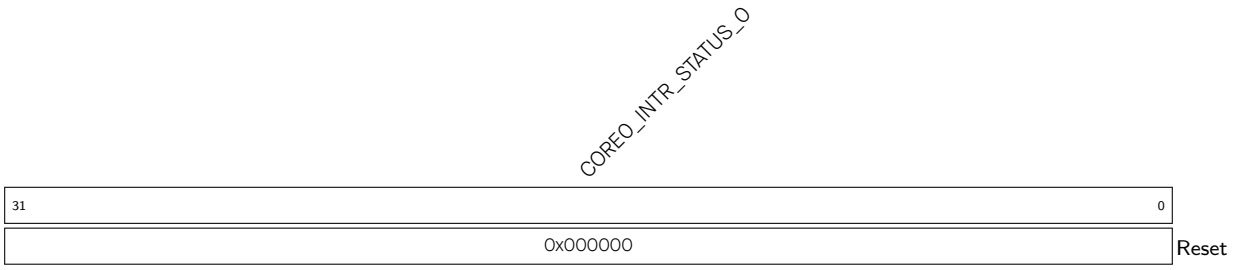
Register 10.127. COREO_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 10.128. COREO_*SOURCE_Y*_MAP_REG (0x0000 - 0x01FC)



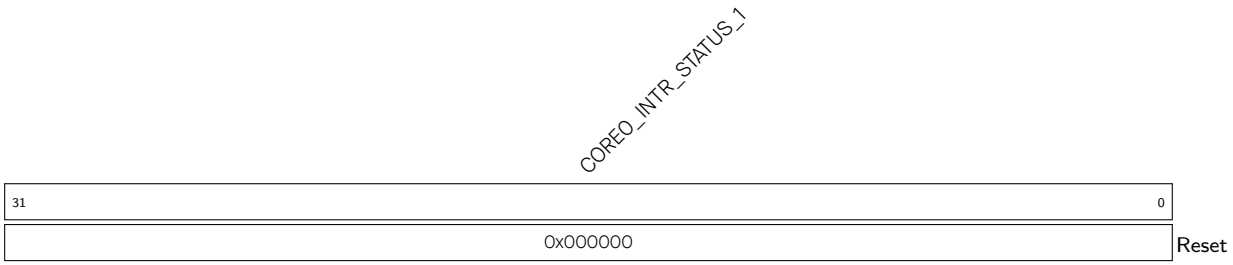
COREO_SOURCE_*Y*_MAP Map interrupt signal of source (Source_*Y*) to one of CPU0's external interrupts, can be configured as 16 ~ 47. Writing 0 ~ 15 will disable the interrupt source (0 is the recommended value). The remaining values are invalid. For Source_*Y*, see Table10.4-1. (R/W)

Register 10.129. COREO_INTR_STATUS_REG_0_REG (0x0200)



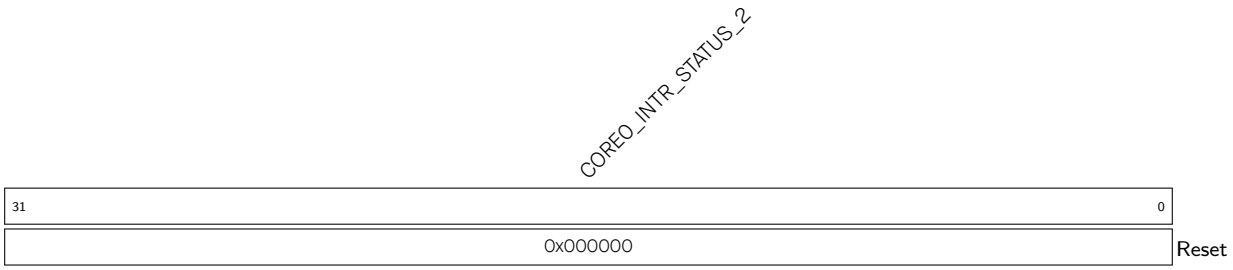
COREO_INTR_STATUS_0 Represents the status of the interrupt sources from 0 ~ 31. Each bit corresponds to one interrupt source. (RO)

Register 10.130. COREO_INTR_STATUS_REG_1_REG (0x0204)



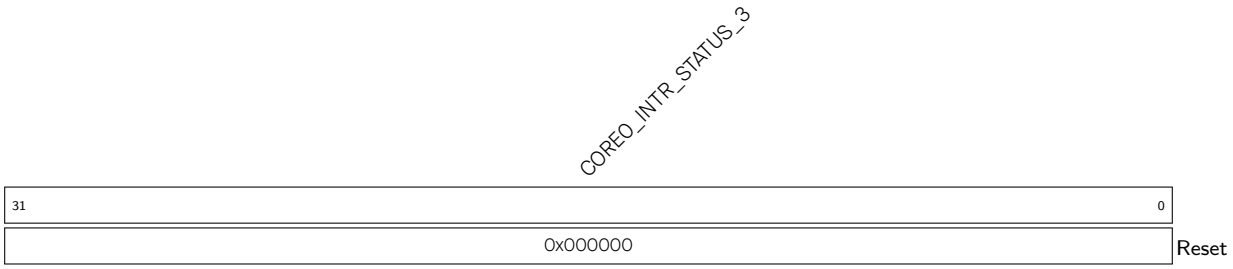
COREO_INTR_STATUS_1 Represents the status of the interrupt sources from 32 ~ 63. Each bit corresponds to one interrupt source. (RO)

Register 10.131. COREO_INTR_STATUS_REG_2_REG (0x0208)



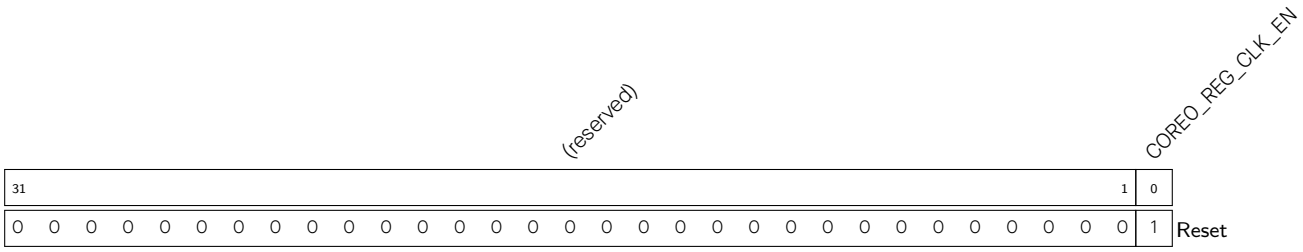
COREO_INTR_STATUS_2 Represents the status of the interrupt sources from 64 ~ 95. Each bit corresponds to one interrupt source. (RO)

Register 10.132. COREO_INTR_STATUS_REG_3_REG (0x020C)



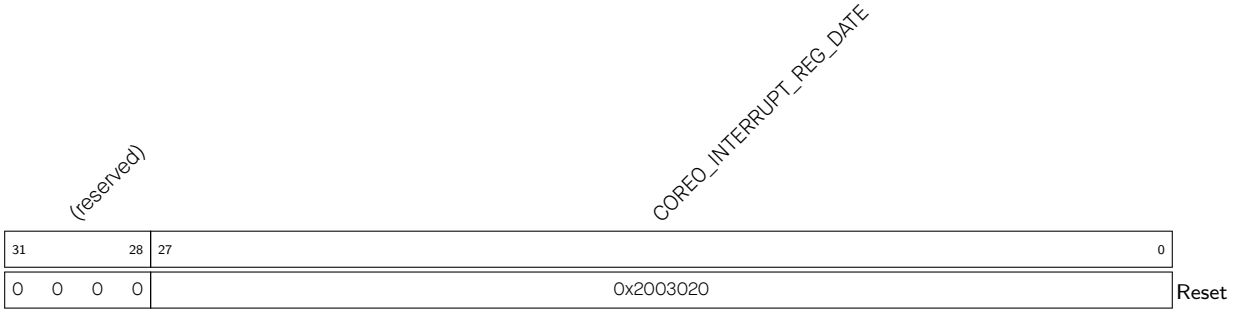
COREO_INTR_STATUS_3 Represents the status of the interrupt sources numbered from 96 ~ 127 (99 and 112 are reserved). Bit 0 ~ 31 each corresponds to one interrupt source.
0: (RO)

Register 10.133. COREO_CLOCK_GATE_REG (0x0210)



COREO_REG_CLK_EN Configures whether to force interrupt register clock-date on.
0: No effect
1: Force interrupt register clock-gate on// (R/W)

Register 10.134. COREO_INTERRUPT_REG_DATE_REG (0x03FC)



COREO_INTERRUPT_REG_DATE Version control register. (R/W)

10.6.2 HP CPU1 Interrupt Matrix Registers

Register 10.135. CORE1_*LP_RTC_INT*_MAP_REG (0x0000)

Register 10.136. CORE1_*LP_WDT_INT*_MAP_REG (0x0004)

Register 10.137. CORE1_*LP_TIMER_REG_0_INT*_MAP_REG (0x0008)

Register 10.138. CORE1_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 10.139. CORE1_*MB_HP_INT*_MAP_REG (0x0010)

Register 10.140. CORE1_*MB_LP_INT*_MAP_REG (0x0014)

Register 10.141. CORE1_*PMU_REG_0_INT*_MAP_REG (0x0018)

Register 10.142. CORE1_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 10.143. CORE1_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 10.144. CORE1_*LP_ADC_INT*_MAP_REG (0x0024)

Register 10.145. CORE1_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 10.146. CORE1_*LP_I2C_INT*_MAP_REG (0x002C)

Register 10.147. CORE1_*LP_I2S_INT*_MAP_REG (0x0030)

Register 10.148. CORE1_*LP_SPI_INT*_MAP_REG (0x0034)

Register 10.149. CORE1_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 10.150. CORE1_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 10.151. CORE1_*LP_UART_INT*_MAP_REG (0x0040)

Register 10.152. CORE1_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 10.153. CORE1_*LP_SW_INT*_MAP_REG (0x0048)

Register 10.154. CORE1_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 10.155. CORE1_*LP_HUK_INT*_MAP_REG (0x0050)

Register 10.156. CORE1_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 10.157. CORE1_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 10.158. CORE1_*SDIO_HOST_INT*_MAP_REG (0x005C)

Register 10.159. CORE1_*GDMA_INT*_MAP_REG (0x0060)

Register 10.160. CORE1_*SPI2_INT*_MAP_REG (0x0064)

Register 10.161. CORE1_*SPI3_INT*_MAP_REG (0x0068)

Register 10.162. CORE1_*I2S0_INT*_MAP_REG (0x006C)

Register 10.163. CORE1_*I2S1_INT*_MAP_REG (0x0070)

Register 10.164. CORE1_*I2S2_INT*_MAP_REG (0x0074)

Register 10.165. CORE1_*UHCI0_INT*_MAP_REG (0x0078)

Register 10.166. CORE1_*UART0_INT*_MAP_REG (0x007C)

Register 10.167. CORE1_*UART1_INT*_MAP_REG (0x0080)

Register 10.168. CORE1_*UART2_INT*_MAP_REG (0x0084)

Register 10.169. CORE1_*UART3_INT*_MAP_REG (0x0088)

Register 10.170. CORE1_*UART4_INT*_MAP_REG (0x008C)

Register 10.171. CORE1_*LCD_CAM_INT*_MAP_REG (0x0090)

Register 10.172. CORE1_*ADC_INT*_MAP_REG (0x0094)

Register 10.173. CORE1_*PWM0_INT*_MAP_REG (0x0098)

Register 10.174. CORE1_*PWM1_INT*_MAP_REG (0x009C)

Register 10.175. CORE1_*TWAI0_INT*_MAP_REG (0x00A0)

Register 10.176. CORE1_*TWAI1_INT*_MAP_REG (0x00A4)

Register 10.177. CORE1_*TWAI2_INT*_MAP_REG (0x00A8)

Register 10.178. CORE1_*RMT_INT*_MAP_REG (0x00AC)

Register 10.179. CORE1_*I2C0_INT*_MAP_REG (0x00B0)

Register 10.180. CORE1_*I2C1_INT*_MAP_REG (0x00B4)

Register 10.181. CORE1_*TIMERGRP0_T0_INT*_MAP_REG (0x00B8)

Register 10.182. CORE1_*TIMERGRP0_T1_INT*_MAP_REG (0x00BC)

Register 10.183. CORE1_*TIMERGRP0_WDT_INT*_MAP_REG (0x00C0)

Register 10.184. CORE1_*TIMERGRP1_T0_INT*_MAP_REG (0x00C4)

Register 10.185. CORE1_*TIMERGRP1_T1_INT*_MAP_REG (0x00C8)

Register 10.186. CORE1_*TIMERGRP1_WDT_INT*_MAP_REG (0x00CC)

Register 10.187. CORE1_*LEDC_INT*_MAP_REG (0x00D0)

Register 10.188. CORE1_*SYSTIMER_TARGET0_INT*_MAP_REG (0x00D4)

Register 10.189. CORE1_*SYSTIMER_TARGET1_INT*_MAP_REG (0x00D8)

Register 10.190. CORE1_*SYSTIMER_TARGET2_INT*_MAP_REG (0x00DC)

Register 10.191. CORE1_*AHB_PDMA_IN_CH0_INT*_MAP_REG (0x00E0)

Register 10.192. CORE1_*AHB_PDMA_IN_CH1_INT*_MAP_REG (0x00E4)

Register 10.193. CORE1_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 10.194. CORE1_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 10.195. CORE1_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 10.196. CORE1_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 10.197. CORE1_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 10.198. CORE1_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 10.199. CORE1_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 10.200. CORE1_*AXI_PDMA_OUT_CH0_INT*_MAP_REG (0x0104)

Register 10.201. CORE1_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 10.202. CORE1_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 10.203. CORE1_*RSA_INT*_MAP_REG (0x0110)

Register 10.204. CORE1_*AES_INT*_MAP_REG (0x0114)

Register 10.205. CORE1_*SHA_INT*_MAP_REG (0x0118)

Register 10.206. CORE1_*ECC_INT*_MAP_REG (0x011C)

Register 10.207. CORE1_*ECDSA_INT*_MAP_REG (0x0120)

Register 10.208. CORE1_*KM_INT*_MAP_REG (0x0124)

Register 10.209. CORE1_*GPIO_INTO*_MAP_REG (0x0128)

Register 10.210. CORE1_*GPIO_INT1*_MAP_REG (0x012C)

Register 10.211. CORE1_*GPIO_INT2*_MAP_REG (0x0130)

Register 10.212. CORE1_*GPIO_INT3*_MAP_REG (0x0134)

Register 10.213. CORE1_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 10.214. CORE1_*CPU_INT_FROM_CPU_O*_MAP_REG (0x013C)

Register 10.215. CORE1_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 10.216. CORE1_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 10.217. CORE1_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 10.218. CORE1_*CACHE_INT*_MAP_REG (0x014C)

Register 10.219. CORE1_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 10.220. CORE1_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 10.221. CORE1_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 10.222. CORE1_*CSI_INT*_MAP_REG (0x015C)

Register 10.223. CORE1_*DSI_INT*_MAP_REG (0x0160)

Register 10.224. CORE1_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 10.225. CORE1_*LPI_INT*_MAP_REG (0x0168)

Register 10.226. CORE1_*PMT_INT*_MAP_REG (0x016C)

Register 10.227. CORE1_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 10.228. CORE1_*USB_OTG_INT*_MAP_REG (0x0174)

Register 10.229. CORE1_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 10.230. CORE1_*JPEG_INT*_MAP_REG (0x017C)

Register 10.231. CORE1_*PPA_INT*_MAP_REG (0x0180)

Register 10.232. CORE1_*COREO_TRACE_INT*_MAP_REG (0x0184)

Register 10.233. CORE1_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 10.234. CORE1_*ISP_INT*_MAP_REG (0x0190)

Register 10.235. CORE1_*I3C_MST_INT*_MAP_REG (0x0194)

Register 10.236. CORE1_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 10.237. CORE1_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 10.238. CORE1_*DMA2D_IN_CHO_INT*_MAP_REG (0x01A0)

Register 10.239. CORE1_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 10.240. CORE1_*DMA2D_OUT_CHO_INT*_MAP_REG (0x01A8)

Register 10.241. CORE1_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 10.242. CORE1_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 10.243. CORE1_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 10.244. CORE1_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 10.245. CORE1_*PCNT_INT*_MAP_REG (0x01BC)

Register 10.246. CORE1_*HP_PAU_INT*_MAP_REG (0x01C0)

Register 10.247. CORE1_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 10.248. CORE1_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 10.249. CORE1_*H264_DMA2D_OUT_CH0_INT*_MAP_REG (0x01CC)

Register 10.250. CORE1_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 10.251. CORE1_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 10.252. CORE1_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 10.253. CORE1_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 10.254. CORE1_*H264_DMA2D_IN_CH0_INT*_MAP_REG (0x01E0)

Register 10.255. CORE1_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 10.256. CORE1_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 10.257. CORE1_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

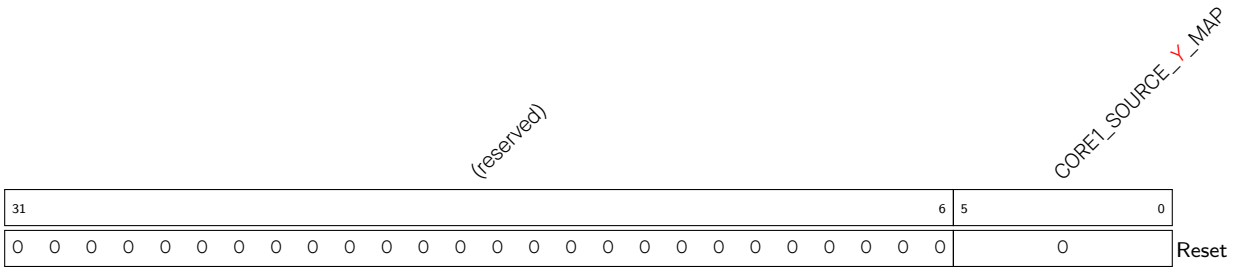
Register 10.258. CORE1_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 10.259. CORE1_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 10.260. CORE1_*H264_REG_INT*_MAP_REG (0x01F8)

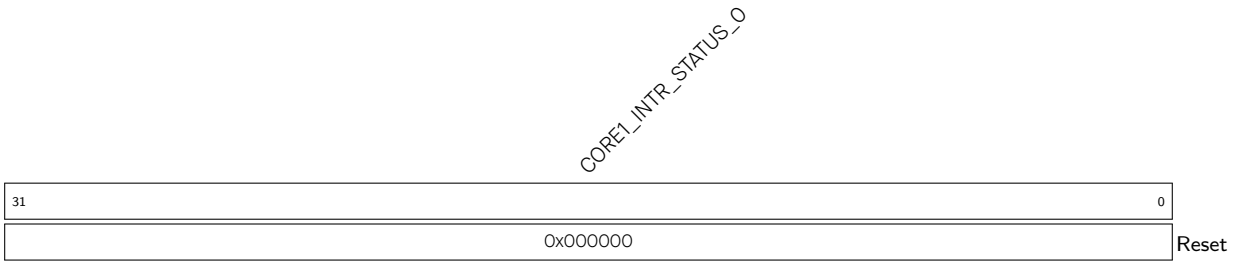
Register 10.261. CORE1_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 10.262. CORE1_*SOURCE_Y*_MAP_REG (0x0000 - 0x01FC)



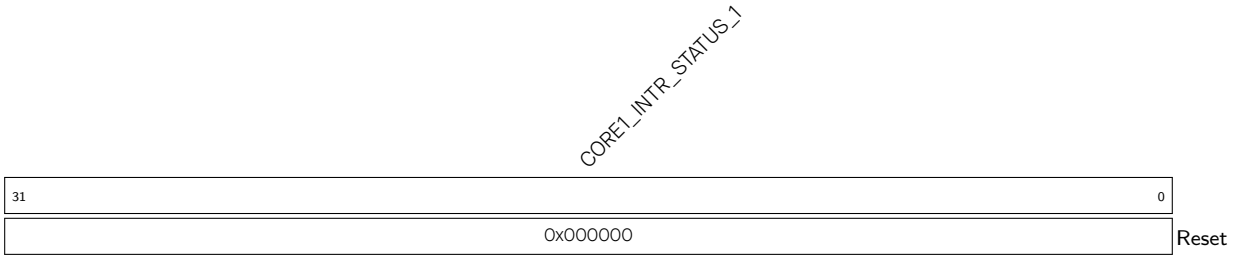
CORE1_SOURCE_*Y*_MAP Map interrupt signal of source (Source_*Y*) to one of CPU1 external interrupts, can be configured as 16 ~ 47. Writing 0 ~ 15 will disable the interrupt source (0 is the recommended value). The remaining values are invalid. For Source_*Y*, see Table10.4-1. (R/W)

Register 10.263. CORE1_INTR_STATUS_REG_0_REG (0x0200)



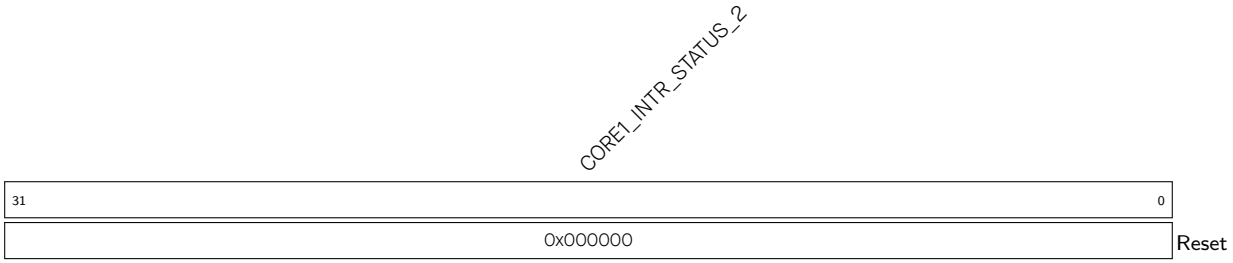
CORE1_INTR_STATUS_0 Represents the status of the interrupt sources from 0 ~ 31. Each bit corresponds to one interrupt source. (RO)

Register 10.264. CORE1_INTR_STATUS_REG_1_REG (0x0204)



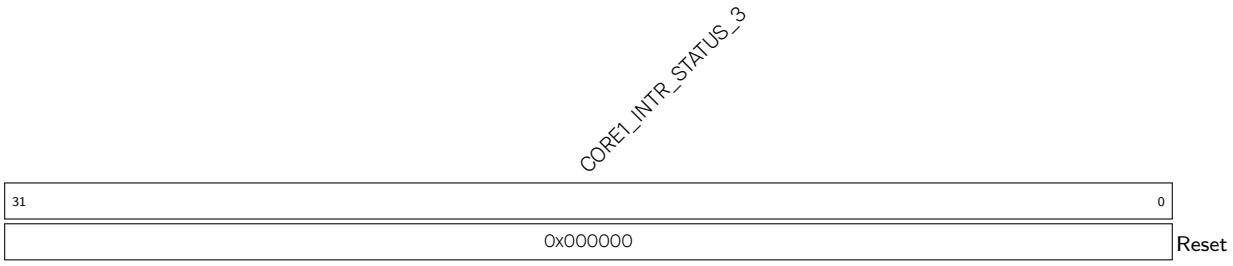
CORE1_INTR_STATUS_1 Represents the status of the interrupt sources from 32 ~ 63. Each bit corresponds to one interrupt source. (RO)

Register 10.265. CORE1_INTR_STATUS_REG_2_REG (0x0208)



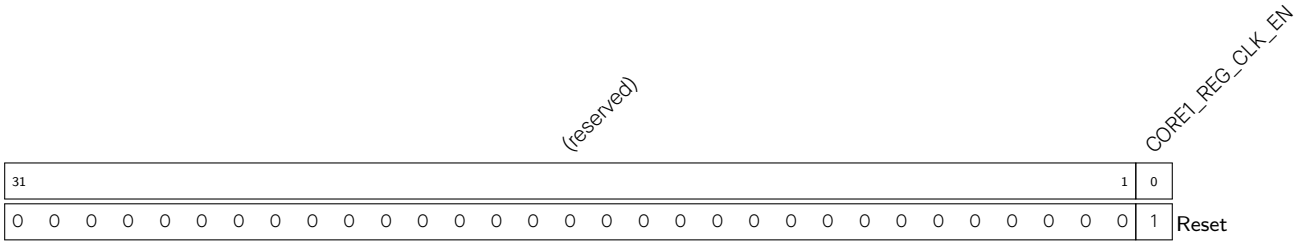
CORE1_INTR_STATUS_2 Represents the status of the interrupt sources from 64 ~ 95. Each bit corresponds to one interrupt source. (RO)

Register 10.266. CORE1_INTR_STATUS_REG_3_REG (0x020C)



CORE1_INTR_STATUS_3 Represents the status of the interrupt sources numbered from 96 ~ 127. Bit 0 ~ 31 each corresponds to one interrupt source. (RO)

Register 10.267. CORE1_CLOCK_GATE_REG (0x0210)



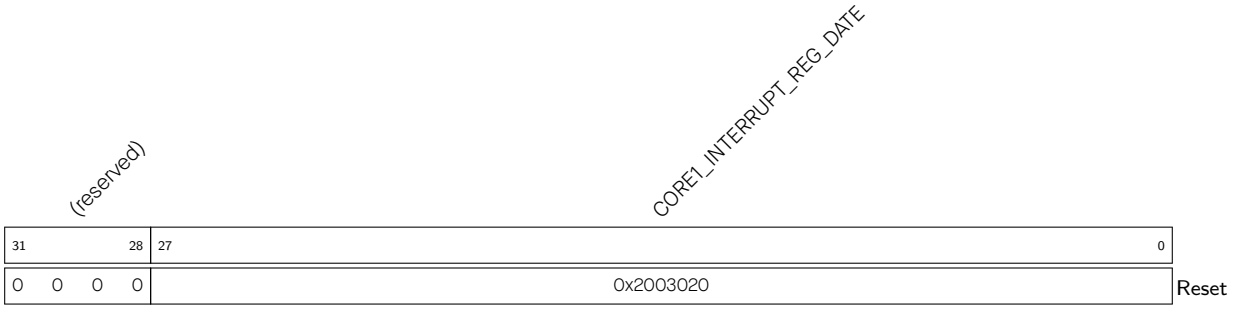
CORE1_REG_CLK_EN Configures whether to force interrupt register clock-date on.

0: No effect

1: Force interrupt register clock-gate on

(R/W)

Register 10.268. CORE1_INTERRUPT_REG_DATE_REG (0x03FC)



CORE1_INTERRUPT_REG_DATE Version control register. (R/W)

Chapter 11

Event Task Matrix (ETM)

11.1 Overview

The Event Task Matrix (ETM) peripheral contains 50 configurable channels. Each channel can map an event of any specified peripheral to a task of any specified peripheral. In this way, peripherals can be triggered to execute specified tasks without CPU intervention.

11.2 Features

The Event Task Matrix has the following features:

- Receive various events from multiple peripherals
- Generate various tasks for multiple peripherals
- 50 independently configurable ETM channels
- An ETM channel can be set up to receive any event, and map it to any task
- Each ETM channel can be enabled independently. If not enabled, the channel will not respond to the configured event and generate the task mapped to that event
- Support for checking event and task status
- Peripherals supporting ETM include GPIO, LED PWM, general-purpose timers, RTC Timer, system timer, MCPWM, temperature sensor, ADC, I2S, LP CPU, GDMA-AHB, GDMA-AXI, 2D DMA, and PMU

Note that the 50 ETM channels are identical regarding their features and operations. Thus, in the following sections, ETM channels are collectively referred to as channel*n* (where *n* ranges from 0 to 49).

11.3 Functional Description

11.3.1 Architecture

The Event Task Matrix has 50 independent channels. A channel can choose any event as input, and map the event to any task as output (see Section 11.3.2 and Section 11.3.3 respectively). Each channel has an individual enable bit (see Section 11.3.6).

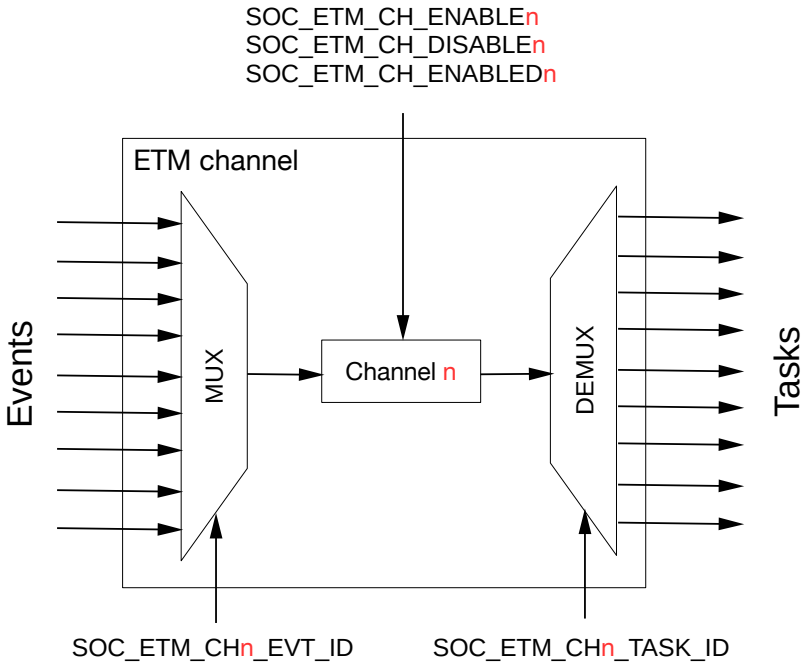


Figure 11.3-1. ETM Channel*n* Architecture

Figure 11.3-1 illustrates the structure of an ETM channel. The SOC_ETM_CH*n*_EVT_ID field configures the MUX (multiplexer) to select one of the events as the input of channel*n*. The SOC_ETM_CH*n*_TASK_ID field configures the DEMUX (demultiplexer) to map the event selected by channel*n* to one of the tasks. SOC_ETM_CH_ENABLE*n* and SOC_ETM_CH_DISABLE*n* are used to enable or disable channel*n*. SOC_ETM_CH_ENABLED*n* is used to indicate the status of the channel*n*.

11.3.2 Events

An ETM channel can be set up to choose which event to receive by configuring the SOC_ETM_CH*n*_EVT_ID field. Table 11.3-1 shows the configuration values of SOC_ETM_CH*n*_EVT_ID and their corresponding events.

Table 11.3-1. Selectable Events for ETM Channel*n*

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
1	GPIO_EVT_CH0_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CH0_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
11	GPIO_EVT_CH2_FALL_EDGE	
12	GPIO_EVT_CH3_FALL_EDGE	
13	GPIO_EVT_CH4_FALL_EDGE	
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	GPIO_EVT_ZERO_DET_POS0	
26	GPIO_EVT_ZERO_DET_NEG0	
27	GPIO_EVT_ZERO_DET_POS1	
28	GPIO_EVT_ZERO_DET_NEG1	
29	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM Controller (LEDC)
30	LEDC_EVT_DUTY_CHNG_END_CH1	
31	LEDC_EVT_DUTY_CHNG_END_CH2	
32	LEDC_EVT_DUTY_CHNG_END_CH3	
33	LEDC_EVT_DUTY_CHNG_END_CH4	
34	LEDC_EVT_DUTY_CHNG_END_CH5	
35	LEDC_EVT_DUTY_CHNG_END_CH6	
36	LEDC_EVT_DUTY_CHNG_END_CH7	
37	LEDC_EVT_OVF_CNT_PLS_CH0	
38	LEDC_EVT_OVF_CNT_PLS_CH1	
39	LEDC_EVT_OVF_CNT_PLS_CH2	
40	LEDC_EVT_OVF_CNT_PLS_CH3	
41	LEDC_EVT_OVF_CNT_PLS_CH4	
42	LEDC_EVT_OVF_CNT_PLS_CH5	
43	LEDC_EVT_OVF_CNT_PLS_CH6	
44	LEDC_EVT_OVF_CNT_PLS_CH7	
45	LEDC_EVT_TIME_OVF_TIMER0	
46	LEDC_EVT_TIME_OVF_TIMER1	
47	LEDC_EVT_TIME_OVF_TIMER2	
48	LEDC_EVT_TIME_OVF_TIMER3	
49	LEDC_EVT_TIMER0_CMP	
50	LEDC_EVT_TIMER1_CMP	
51	LEDC_EVT_TIMER2_CMP	
52	LEDC_EVT_TIMER3_CMP	
53	TG0_EVT_CNT_CMP_TIMER0	General-purpose timer group 0
55	TG1_EVT_CNT_CMP_TIMER0	General-purpose timer group 1
57	SYSTIMER_EVT_CNT_CMP0	System Timer

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
58	SYSTIMER_EVT_CNT_CMP1	
59	SYSTIMER_EVT_CNT_CMP2	
60	MCPWM0_EVT_TIMER0_STOP	MCPWM0
61	MCPWM0_EVT_TIMER1_STOP	
62	MCPWM0_EVT_TIMER2_STOP	
63	MCPWM0_EVT_TIMER0_TEZ	
64	MCPWM0_EVT_TIMER1_TEZ	
65	MCPWM0_EVT_TIMER2_TEZ	
66	MCPWM0_EVT_TIMER0_TEP	
67	MCPWM0_EVT_TIMER1_TEP	
68	MCPWM0_EVT_TIMER2_TEP	
69	MCPWM0_EVT_OP0_TEA	
70	MCPWM0_EVT_OP1_TEA	
71	MCPWM0_EVT_OP2_TEA	
72	MCPWM0_EVT_OP0_TEB	
73	MCPWM0_EVT_OP1_TEB	
74	MCPWM0_EVT_OP2_TEB	
75	MCPWM0_EVT_F0	
76	MCPWM0_EVT_F1	
77	MCPWM0_EVT_F2	
78	MCPWM0_EVT_F0_CLR	
79	MCPWM0_EVT_F1_CLR	
80	MCPWM0_EVT_F2_CLR	
81	MCPWM0_EVT_TZ0_CBC	
82	MCPWM0_EVT_TZ1_CBC	
83	MCPWM0_EVT_TZ2_CBC	
84	MCPWM0_EVT_TZ0_OST	
85	MCPWM0_EVT_TZ1_OST	
86	MCPWM0_EVT_TZ2_OST	
87	MCPWM0_EVT_CAP0	
88	MCPWM0_EVT_CAP1	
89	MCPWM0_EVT_CAP2	
90	MCPWM0_EVT_OP0_TEE1	
91	MCPWM0_EVT_OP1_TEE1	
92	MCPWM0_EVT_OP2_TEE1	
93	MCPWM0_EVT_OP0_TEE2	
94	MCPWM0_EVT_OP1_TEE2	
95	MCPWM0_EVT_OP2_TEE2	
96	MCPWM1_EVT_TIMER0_STOP	MCPWM1
97	MCPWM1_EVT_TIMER1_STOP	
98	MCPWM1_EVT_TIMER2_STOP	
99	MCPWM1_EVT_TIMER0_TEZ	
100	MCPWM1_EVT_TIMER1_TEZ	
101	MCPWM1_EVT_TIMER2_TEZ	
102	MCPWM1_EVT_TIMER0_TEP	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
103	MCPWM1_EVT_TIMER1_TEP	
104	MCPWM1_EVT_TIMER2_TEP	
105	MCPWM1_EVT_OP0_TEA	
106	MCPWM1_EVT_OP1_TEA	
107	MCPWM1_EVT_OP2_TEA	
108	MCPWM1_EVT_OP0_TEB	
109	MCPWM1_EVT_OP1_TEB	
110	MCPWM1_EVT_OP2_TEB	
111	MCPWM1_EVT_F0	
112	MCPWM1_EVT_F1	
113	MCPWM1_EVT_F2	
114	MCPWM1_EVT_F0_CLR	
115	MCPWM1_EVT_F1_CLR	
116	MCPWM1_EVT_F2_CLR	
117	MCPWM1_EVT_TZ0_CBC	
118	MCPWM1_EVT_TZ1_CBC	
119	MCPWM1_EVT_TZ2_CBC	
120	MCPWM1_EVT_TZ0_OST	
121	MCPWM1_EVT_TZ1_OST	
122	MCPWM1_EVT_TZ2_OST	
123	MCPWM1_EVT_CAP0	
124	MCPWM1_EVT_CAP1	
125	MCPWM1_EVT_CAP2	
126	MCPWM1_EVT_OP0_TEE1	
127	MCPWM1_EVT_OP1_TEE1	
128	MCPWM1_EVT_OP2_TEE1	
129	MCPWM1_EVT_OP0_TEE2	
130	MCPWM1_EVT_OP1_TEE2	
131	MCPWM1_EVT_OP2_TEE2	
132	ADC_EVT_CONV_CMPLT0	ADC Controller (ADC)
133	ADC_EVT_EQ_ABOVE_THRESH0	
134	ADC_EVT_EQ_ABOVE_THRESH1	
135	ADC_EVT_EQ_BELOW_THRESH0	
136	ADC_EVT_EQ_BELOW_THRESH1	
138	ADC_EVT_STOPPED0	
139	ADC_EVT_STARTED0	
148	TMPSNSR_EVT_OVER_LIMIT	Temperature Sensor (TSENS)
149	I2S0_EVT_RX_DONE	I2S0
150	I2S0_EVT_TX_DONE	
151	I2S0_EVT_X_WORDS_RECEIVED	
152	I2S0_EVT_X_WORDS_SENT	
153	I2S1_EVT_RX_DONE	I2S1
154	I2S1_EVT_TX_DONE	
155	I2S1_EVT_X_WORDS_RECEIVED	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
156	I2S1_EVT_X_WORDS_SENT	
157	I2S2_EVT_RX_DONE	I2S2
158	I2S2_EVT_TX_DONE	
159	I2S2_EVT_X_WORDS_RECEIVED	
160	I2S2_EVT_X_WORDS_SENT	
161	ULP_EVT_ERR_INTR	Low-Power CPU
162	ULP_EVT_HALT	
163	ULP_EVT_START_INTR	
164	RTC_EVT_TICK	RTC Timer
165	RTC_EVT_OVF	
166	RTC_EVT_CMP	
167	GDMA_AHB_EVT_IN_DONE_CH0	GDMA-AHB
168	GDMA_AHB_EVT_IN_DONE_CH1	
169	GDMA_AHB_EVT_IN_DONE_CH2	
170	GDMA_AHB_EVT_IN_SUC_EOF_CH0	
171	GDMA_AHB_EVT_IN_SUC_EOF_CH1	
172	GDMA_AHB_EVT_IN_SUC_EOF_CH2	
173	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0	
174	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1	
175	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2	
176	GDMA_AHB_EVT_IN_FIFO_FULL_CH0	
177	GDMA_AHB_EVT_IN_FIFO_FULL_CH1	
178	GDMA_AHB_EVT_IN_FIFO_FULL_CH2	
179	GDMA_AHB_EVT_OUT_DONE_CH0	
180	GDMA_AHB_EVT_OUT_DONE_CH1	
181	GDMA_AHB_EVT_OUT_DONE_CH2	
182	GDMA_AHB_EVT_OUT_EOF_CH0	
183	GDMA_AHB_EVT_OUT_EOF_CH1	
184	GDMA_AHB_EVT_OUT_EOF_CH2	
185	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0	
186	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1	
187	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2	
188	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0	
189	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1	
190	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2	
191	GDMA_AHB_EVT_OUT_FIFO_FULL_CH0	
192	GDMA_AHB_EVT_OUT_FIFO_FULL_CH1	
193	GDMA_AHB_EVT_OUT_FIFO_FULL_CH2	
194	GDMA_AXI_EVT_IN_DONE_CH0	GDMA-AXI
195	GDMA_AXI_EVT_IN_DONE_CH1	
196	GDMA_AXI_EVT_IN_DONE_CH2	
197	GDMA_AXI_EVT_IN_SUC_EOF_CH0	
198	GDMA_AXI_EVT_IN_SUC_EOF_CH1	
199	GDMA_AXI_EVT_IN_SUC_EOF_CH2	
200	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0	

SOC_ETM_CH*n*_EVT_ID	Selected Event	Peripheral Generating This Event
201	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1	
202	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2	
203	GDMA_AXI_EVT_IN_FIFO_FULL_CH0	
204	GDMA_AXI_EVT_IN_FIFO_FULL_CH1	
205	GDMA_AXI_EVT_IN_FIFO_FULL_CH2	
206	GDMA_AXI_EVT_OUT_DONE_CH0	
207	GDMA_AXI_EVT_OUT_DONE_CH1	
208	GDMA_AXI_EVT_OUT_DONE_CH2	
209	GDMA_AXI_EVT_OUT_EOF_CH0	
210	GDMA_AXI_EVT_OUT_EOF_CH1	
211	GDMA_AXI_EVT_OUT_EOF_CH2	
212	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0	
213	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1	
214	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2	
215	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0	
216	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1	
217	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2	
218	GDMA_AXI_EVT_OUT_FIFO_FULL_CH0	
219	GDMA_AXI_EVT_OUT_FIFO_FULL_CH1	
220	GDMA_AXI_EVT_OUT_FIFO_FULL_CH2	
221	PMU_EVT_SLEEP_WEEKUP	PMU
222	DMA2D_EVT_IN_DONE_CH0	2D-DMA Controller (2D-DMA)
223	DMA2D_EVT_IN_DONE_CH1	
224	DMA2D_EVT_IN_SUC_EOF_CH0	
225	DMA2D_EVT_IN_SUC_EOF_CH1	
226	DMA2D_EVT_OUT_DONE_CH0	
227	DMA2D_EVT_OUT_DONE_CH1	
228	DMA2D_EVT_OUT_DONE_CH2	
229	DMA2D_EVT_OUT_EOF_CH0	
230	DMA2D_EVT_OUT_EOF_CH1	
231	DMA2D_EVT_OUT_EOF_CH2	
232	DMA2D_EVT_OUT_TOTAL_EOF_CH0	
233	DMA2D_EVT_OUT_TOTAL_EOF_CH1	
234	DMA2D_EVT_OUT_TOTAL_EOF_CH2	

Whenever any of these events occurs, the corresponding peripheral generates a pulse signal. As soon as the pulse signal is high, the event is considered as being received.

For more detailed descriptions of an event, please refer to the chapter for the peripheral generating this event.

11.3.3 Tasks

An ETM channel can be set up to map its event to one of the tasks by configuring the SOC_ETM_CH*n*_TASK_ID field. Table11.3-2 shows the configuration values of SOC_ETM_CH*n*_TASK_ID and

their corresponding tasks.

Table 11.3-2. Mappable Tasks for ETM Channel*n*

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM Controller (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
29	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
30	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
35	LEDC_TASK_DUTY_SCALE_UPDATE_CH6	
36	LEDC_TASK_DUTY_SCALE_UPDATE_CH7	
37	LEDC_TASK_TIMER0_CAP	
38	LEDC_TASK_TIMER1_CAP	
39	LEDC_TASK_TIMER2_CAP	
40	LEDC_TASK_TIMER3_CAP	
41	LEDC_TASK_SIG_OUT_DIS_CH0	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
42	LEDC_TASK_SIG_OUT_DIS_CH1	
43	LEDC_TASK_SIG_OUT_DIS_CH2	
44	LEDC_TASK_SIG_OUT_DIS_CH3	
45	LEDC_TASK_SIG_OUT_DIS_CH4	
46	LEDC_TASK_SIG_OUT_DIS_CH5	
47	LEDC_TASK_SIG_OUT_DIS_CH6	
48	LEDC_TASK_SIG_OUT_DIS_CH7	
49	LEDC_TASK_OVF_CNT_RST_CH0	
50	LEDC_TASK_OVF_CNT_RST_CH1	
51	LEDC_TASK_OVF_CNT_RST_CH2	
52	LEDC_TASK_OVF_CNT_RST_CH3	
53	LEDC_TASK_OVF_CNT_RST_CH4	
54	LEDC_TASK_OVF_CNT_RST_CH5	
55	LEDC_TASK_OVF_CNT_RST_CH6	
56	LEDC_TASK_OVF_CNT_RST_CH7	
57	LEDC_TASK_TIMER0_RST	
58	LEDC_TASK_TIMER1_RST	
59	LEDC_TASK_TIMER2_RST	
60	LEDC_TASK_TIMER3_RST	
61	LEDC_TASK_TIMER0_RESUME	
62	LEDC_TASK_TIMER1_RESUME	
63	LEDC_TASK_TIMER2_RESUME	
64	LEDC_TASK_TIMER3_RESUME	
65	LEDC_TASK_TIMER0_PAUSE	
66	LEDC_TASK_TIMER1_PAUSE	
67	LEDC_TASK_TIMER2_PAUSE	
68	LEDC_TASK_TIMER3_PAUSE	
69	LEDC_TASK_GAMMA_RESTART_CH0	
70	LEDC_TASK_GAMMA_RESTART_CH1	
71	LEDC_TASK_GAMMA_RESTART_CH2	
72	LEDC_TASK_GAMMA_RESTART_CH3	
73	LEDC_TASK_GAMMA_RESTART_CH4	
74	LEDC_TASK_GAMMA_RESTART_CH5	
75	LEDC_TASK_GAMMA_RESTART_CH6	
76	LEDC_TASK_GAMMA_RESTART_CH7	
77	LEDC_TASK_GAMMA_PAUSE_CH0	
78	LEDC_TASK_GAMMA_PAUSE_CH1	
79	LEDC_TASK_GAMMA_PAUSE_CH2	
80	LEDC_TASK_GAMMA_PAUSE_CH3	
81	LEDC_TASK_GAMMA_PAUSE_CH4	
82	LEDC_TASK_GAMMA_PAUSE_CH5	
83	LEDC_TASK_GAMMA_PAUSE_CH6	
84	LEDC_TASK_GAMMA_PAUSE_CH7	
85	LEDC_TASK_GAMMA_RESUME_CH0	
86	LEDC_TASK_GAMMA_RESUME_CH1	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
87	LEDC_TASK_GAMMA_RESUME_CH2	
88	LEDC_TASK_GAMMA_RESUME_CH3	
89	LEDC_TASK_GAMMA_RESUME_CH4	
90	LEDC_TASK_GAMMA_RESUME_CH5	
91	LEDC_TASK_GAMMA_RESUME_CH6	
92	LEDC_TASK_GAMMA_RESUME_CH7	
93	TG0_TASK_CNT_START_TIMER0	General-purpose timer group 0
94	TG0_TASK_ALARM_START_TIMER0	
95	TG0_TASK_CNT_STOP_TIMER0	
96	TG0_TASK_CNT_RELOAD_TIMER0	
97	TG0_TASK_CNT_CAP_TIMER0	
103	TG1_TASK_CNT_START_TIMER0	General-purpose timer group 1
104	TG1_TASK_ALARM_START_TIMER0	
105	TG1_TASK_CNT_STOP_TIMER0	
106	TG1_TASK_CNT_RELOAD_TIMER0	
107	TG1_TASK_CNT_CAP_TIMER0	
113	MCPWM0_TASK_CMPR0_A_UP	MCPWM0
114	MCPWM0_TASK_CMPR1_A_UP	
115	MCPWM0_TASK_CMPR2_A_UP	
116	MCPWM0_TASK_CMPR0_B_UP	
117	MCPWM0_TASK_CMPR1_B_UP	
118	MCPWM0_TASK_CMPR2_B_UP	
119	MCPWM0_TASK_GEN_STOP	
120	MCPWM0_TASK_TIMER0_SYN	
121	MCPWM0_TASK_TIMER1_SYN	
122	MCPWM0_TASK_TIMER2_SYN	
123	MCPWM0_TASK_TIMER0_PERIOD_UP	
124	MCPWM0_TASK_TIMER1_PERIOD_UP	
125	MCPWM0_TASK_TIMER2_PERIOD_UP	
126	MCPWM0_TASK_TZ0_OST	
127	MCPWM0_TASK_TZ1_OST	
128	MCPWM0_TASK_TZ2_OST	
129	MCPWM0_TASK_CLR0_OST	
130	MCPWM0_TASK_CLR1_OST	
131	MCPWM0_TASK_CLR2_OST	
132	MCPWM0_TASK_CAP0	
133	MCPWM0_TASK_CAP1	
134	MCPWM0_TASK_CAP2	
135	MCPWM1_TASK_CMPR0_A_UP	MCPWM1
136	MCPWM1_TASK_CMPR1_A_UP	
137	MCPWM1_TASK_CMPR2_A_UP	
138	MCPWM1_TASK_CMPR0_B_UP	
139	MCPWM1_TASK_CMPR1_B_UP	
140	MCPWM1_TASK_CMPR2_B_UP	
141	MCPWM1_TASK_GEN_STOP	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
142	MCPWM1_TASK_TIMER0_SYN	
143	MCPWM1_TASK_TIMER1_SYN	
144	MCPWM1_TASK_TIMER2_SYN	
145	MCPWM1_TASK_TIMER0_PERIOD_UP	
146	MCPWM1_TASK_TIMER1_PERIOD_UP	
147	MCPWM1_TASK_TIMER2_PERIOD_UP	
148	MCPWM1_TASK_TZ0_OST	
149	MCPWM1_TASK_TZ1_OST	
150	MCPWM1_TASK_TZ2_OST	
151	MCPWM1_TASK_CLR0_OST	
152	MCPWM1_TASK_CLR1_OST	
153	MCPWM1_TASK_CLR2_OST	
154	MCPWM1_TASK_CAP0	
155	MCPWM1_TASK_CAP1	
156	MCPWM1_TASK_CAP2	
157	ADC_TASK_SAMPLE0	ADC Controller (ADC)
158	ADC_TASK_SAMPLE1	
159	ADC_TASK_START0	
160	ADC_TASK_STOP0	
165	TMPSNSR_TASK_START_SAMPLE	Temperature Sensor (TSENS)
166	TMPSNSR_TASK_STOP_SAMPLE	
167	I2S0_TASK_START_RX	I2S0
168	I2S0_TASK_START_TX	
169	I2S0_TASK_STOP_RX	
170	I2S0_TASK_STOP_TX	
171	I2S1_TASK_START_RX	I2S1
172	I2S1_TASK_START_TX	
173	I2S1_TASK_STOP_RX	
174	I2S1_TASK_STOP_TX	
175	I2S2_TASK_START_RX	I2S2
176	I2S2_TASK_START_TX	
177	I2S2_TASK_STOP_RX	
178	I2S2_TASK_STOP_TX	
179	ULP_TASK_WEAKUP_CPU	Low-Power CPU
180	ULP_TASK_INT_CPU	
185	GDMA_AHB_TASK_IN_START_CH0	GDMA-AHB
186	GDMA_AHB_TASK_IN_START_CH1	
187	GDMA_AHB_TASK_IN_START_CH2	
188	GDMA_AHB_TASK_OUT_START_CH0	
189	GDMA_AHB_TASK_OUT_START_CH1	
190	GDMA_AHB_TASK_OUT_START_CH2	
191	GDMA_AXI_TASK_IN_START_CH0	GDMA-AXI
192	GDMA_AXI_TASK_IN_START_CH1	
193	GDMA_AXI_TASK_IN_START_CH2	
194	GDMA_AXI_TASK_OUT_START_CH0	

SOC_ETM_CH*n*_TASK_ID	Mapped Task	Peripheral Receiving This Task
195	GDMA_AXI_TASK_OUT_START_CH1	
196	GDMA_AXI_TASK_OUT_START_CH2	
197	PMU_TASK_SLEEP_REQ	PMU
198	DMA2D_TASK_IN_START_CH0	2D-DMA Controller (2D-DMA)
199	DMA2D_TASK_IN_START_CH1	
200	DMA2D_TASK_IN_DSCR_READY_CH0	
201	DMA2D_TASK_IN_DSCR_READY_CH1	
202	DMA2D_TASK_OUT_START_CH0	
203	DMA2D_TASK_OUT_START_CH1	
204	DMA2D_TASK_OUT_START_CH2	
205	DMA2D_TASK_OUT_DSCR_READY_CH0	
206	DMA2D_TASK_OUT_DSCR_READY_CH1	
207	DMA2D_TASK_OUT_DSCR_READY_CH2	

When a channel receives a valid event pulse signal, it generates the mapped task pulse signal.

For more detailed descriptions of a task, please refer to the chapter for the peripheral receiving this task.

Events from different channels can be optionally mapped to the same task. For example, field SOC_ETM_CH*n*_TASK_ID of multiple channels can be configured with the same value, and field SOC_ETM_CH*n*_EVT_ID can be configured with the same or different values. In this case, when the event received by any of the channels is valid, the task is generated. If events received by multiple channels are valid at the same time, the task will be generated only once.

11.3.4 Event and Task Status

The ETM module supports checking the status of events and tasks by reading registers as follows:

- Event status can be checked by reading the SOC_ETM_EVT_ST*n*_REG register. If the field corresponding to an event is 1, it indicates that the event has been received by ETM. Otherwise, it indicates that the event has not been received by ETM. The fields in the SOC_ETM_EVT_ST*n*_REG register can be cleared by writing 1 to the corresponding fields in the SOC_ETM_EVT_ST*n*_CLR_REG register.
- Task status can be checked by reading the SOC_ETM_TASK_ST*n*_REG register. If the field corresponding to a task is 1, it indicates that the task has been generated by ETM. Otherwise, it indicates that the task has not been generated by ETM. The fields in the SOC_ETM_TASK_ST*n*_REG register can be cleared by writing 1 to the corresponding fields in the SOC_ETM_TASK_ST*n*_CLR_REG register.

11.3.5 Timing Considerations

Figure 11.3-2 shows the structure of clocks that drive received events, sent tasks, and ETM channels.

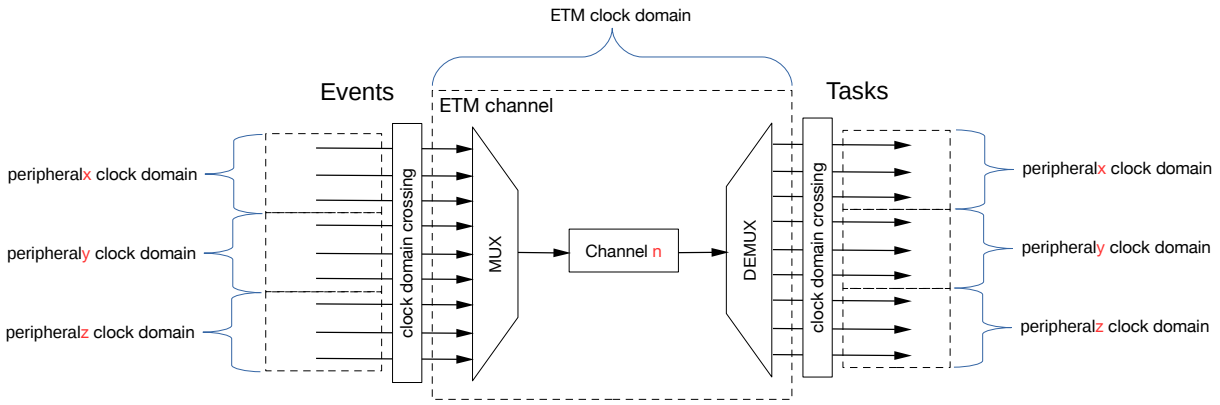


Figure 11.3-2. Event Task Matrix Clock Architecture

ETM is running at the SYS_CLK domain (see Chapter 8 *Reset and Clock*). Each event corresponds to a pulse signal generated by the corresponding peripheral in its clock domain, while each task is mapped by the ETM to a pulse signal under its corresponding peripheral clock domain. The peripherals generating events, the Event Task Matrix, and peripherals receiving tasks are not necessarily running off the same clock and as such need to be synchronized. Therefore, to avoid event loss, there should be a delay between two consecutive event pulses which is dependent on the clock of the peripheral generating the event, on the ETM's clock, and on the clock of the peripheral executing the task.

To make sure the Event Task Matrix receives every event successfully, for peripherals generating event pulses, the interval between two consecutive pulses must be greater than one ETM clock cycle, namely $ceil(\frac{peripheral\_clock\_frequency}{ETM\_clock\_frequency})$ in the unit of peripheral clock cycles.

For example, assuming that peripheral A is in the 80 MHz clock domain (REF_80M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). Peripheral A generates event 1, which should be received by ETM. To receive each event 1 successfully, the interval between two consecutive event 1 must be greater than two peripheral A clock cycles (i.e., one ETM clock cycle).

Likewise, to make sure the Event Task Matrix maps the received event (i.e., event synchronized to the ETM's clock domain) successfully to a task, the interval between two consecutive event pulses in the ETM clock domain must be greater than one peripheral clock cycle, namely $ceil(\frac{ETM\_clock\_frequency}{peripheral\_clock\_frequency})$ in the unit of ETM clock cycles.

For example, assuming that peripheral B is in the 20 MHz clock domain (FOSC_20M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). ETM maps an event to task 1, which should be received by peripheral B. To map each received event successfully to task 1, the interval between two consecutive events must be greater than two ETM clock cycles (i.e., one peripheral B clock cycle).

As a result, to map two consecutive events generated by peripheral A to peripheral B, the interval between these two events must be $ceil(\frac{peripheral\_A\_clock\_frequency}{ETM\_clock\_frequency}) * ceil(\frac{ETM\_clock\_frequency}{peripheral\_B\_clock\_frequency})$ in the unit of peripheral A clock cycles.

For example, assuming that peripheral A is in the 80 MHz clock domain (REF_80M_CLK), peripheral B is in the 20 MHz clock domain (FOSC_20M_CLK), and the ETM runs in the 40 MHz clock domain (SYS_CLK). ETM maps event 1 generated by peripheral A to task 1 received by peripheral B. To successfully map each event 1 to task 1, the interval between two consecutive event 1 must be greater than $2 * 2 = 4$ peripheral A clock

cycles.

11.3.6 Channel Control

Each ETM channel can be independently configured to be enabled or disabled. When channel*n* is enabled and receives the event configured via SOC_ETM_CH*n*_EVT_ID, it maps the event to the task configured via SOC_ETM_CH*n*_TASK_ID. When channel*n* is disabled, even if it receives the event configured via SOC_ETM_CH*n*_EVT_ID, no task will be generated.

To enable ETM channel*n*, write 1 to SOC_ETM_CH_ENABLE*n*. To disable ETM channel*n*, Write 1 to SOC_ETM_CH_DISABLE*n*.

The status of ETM channel*n* can be obtained by reading SOC_ETM_CH_ENABLED*n*. 1 indicates that channel*n* has been enabled, and 0 indicates disabled.

If SOC_ETM_CH*n*_EVT_ID or SOC_ETM_CH*n*_TASK_ID is configured to 0, ETM channel*n* will also be disabled.

The complete procedure to configure ETM channel*n* is as follows:

1. Enable the ETM's clock by writing 1 to HP_SYS_CLKRST_REG_ETM_SYS_CLK_EN
2. Select the event to be received by channel*n* via SOC_ETM_CH*n*_EVT_ID
3. Select the task mapped to the received event via SOC_ETM_CH*n*_TASK_ID
4. Enable channel*n* by setting SOC_ETM_CH_ENABLE*n*
5. When channel*n* no longer needs to map the selected event to the selected task, disable channel*n* by setting SOC_ETM_CH_DISABLE*n*. To configure a new event and task mapping, repeat Steps 2 to 4. If no configurations, channel*n* will remain disabled
6. The whole ETM module (i.e., all ETM channels) can be reset by writing 1 and then 0 to the HP_SYS_CLKRST_REG_RST_EN_ETM field

11.4 Register Summary

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

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

Name	Description	Address	Access
Status register			
SOC_ETM_CH_ENA_AD0_REG	Channel status register	0x0000	R/WTC/WTS
SOC_ETM_CH_ENA_AD1_REG	Channel status register	0x000C	R/WTC/WTS
SOC_ETM_EVT_ST0_REG	Event status register	0x01A8	R/WTC/SS
SOC_ETM_EVT_ST1_REG	Event status register	0x01B0	R/WTC/SS
SOC_ETM_EVT_ST2_REG	Event status register	0x01B8	R/WTC/SS
SOC_ETM_EVT_ST3_REG	Event status register	0x01C0	R/WTC/SS
SOC_ETM_EVT_ST4_REG	Event status register	0x01C8	R/WTC/SS
SOC_ETM_EVT_ST5_REG	Event status register	0x01D0	R/WTC/SS
SOC_ETM_EVT_ST6_REG	Event status register	0x01D8	R/WTC/SS
SOC_ETM_EVT_ST7_REG	Event status register	0x01E0	R/WTC/SS
SOC_ETM_TASK_ST0_REG	Task status register	0x01E8	R/WTC/SS
SOC_ETM_TASK_ST1_REG	Task status register	0x01F0	R/WTC/SS
SOC_ETM_TASK_ST2_REG	Task status register	0x01F8	R/WTC/SS
SOC_ETM_TASK_ST3_REG	Task status register	0x0200	R/WTC/SS
SOC_ETM_TASK_ST4_REG	Task status register	0x0208	R/WTC/SS
SOC_ETM_TASK_ST5_REG	Task status register	0x0210	R/WTC/SS
SOC_ETM_TASK_ST6_REG	Task status register	0x0218	R/WTC/SS
Configuration Register			
SOC_ETM_CH_ENA_AD0_SET_REG	Channel enable register	0x0004	WT
SOC_ETM_CH_ENA_AD0_CLR_REG	Channel disable register	0x0008	WT
SOC_ETM_CH_ENA_AD1_SET_REG	Channel enable register	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	Channel disable register	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	Channel0 event ID register	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	Channel0 task ID register	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	Channel1 event ID register	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	Channel1 task ID register	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	Channel2 event ID register	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	Channel2 task ID register	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	Channel3 event ID register	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	Channel3 task ID register	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	Channel4 event ID register	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	Channel4 task ID register	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	Channel5 event ID register	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	Channel5 task ID register	0x0044	R/W
SOC_ETM_CH6_EVT_ID_REG	Channel6 event ID register	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	Channel6 task ID register	0x004C	R/W

Name	Description	Address	Access
SOC_ETM_CH7_EVT_ID_REG	Channel7 event ID register	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	Channel7 task ID register	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	Channel8 event ID register	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	Channel8 task ID register	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	Channel9 event ID register	0x0060	R/W
SOC_ETM_CH9_TASK_ID_REG	Channel9 task ID register	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	Channel10 event ID register	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	Channel10 task ID register	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	Channel11 event ID register	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	Channel11 task ID register	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	Channel12 event ID register	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	Channel12 task ID register	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	Channel13 event ID register	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	Channel13 task ID register	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	Channel14 event ID register	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	Channel14 task ID register	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	Channel15 event ID register	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	Channel15 task ID register	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	Channel16 event ID register	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	Channel16 task ID register	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	Channel17 event ID register	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	Channel17 task ID register	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	Channel18 event ID register	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	Channel18 task ID register	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	Channel19 event ID register	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	Channel19 task ID register	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	Channel20 event ID register	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	Channel20 task ID register	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	Channel21 event ID register	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	Channel21 task ID register	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	Channel22 event ID register	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	Channel22 task ID register	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	Channel23 event ID register	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	Channel23 task ID register	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	Channel24 event ID register	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	Channel24 task ID register	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	Channel25 event ID register	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	Channel25 task ID register	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	Channel26 event ID register	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	Channel26 task ID register	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	Channel27 event ID register	0x00F0	R/W
SOC_ETM_CH27_TASK_ID_REG	Channel27 task ID register	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	Channel28 event ID register	0x00F8	R/W

Name	Description	Address	Access
SOC_ETM_CH28_TASK_ID_REG	Channel28 task ID register	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	Channel29 event ID register	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	Channel29 task ID register	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	Channel30 event ID register	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	Channel30 task ID register	0x010C	R/W
SOC_ETM_CH31_EVT_ID_REG	Channel31 event ID register	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	Channel31 task ID register	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	Channel32 event ID register	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	Channel32 task ID register	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	Channel33 event ID register	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	Channel33 task ID register	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	Channel34 event ID register	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	Channel34 task ID register	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	Channel35 event ID register	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	Channel35 task ID register	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	Channel36 event ID register	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	Channel36 task ID register	0x013C	R/W
SOC_ETM_CH37_EVT_ID_REG	Channel37 event ID register	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	Channel37 task ID register	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	Channel38 event ID register	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	Channel38 task ID register	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	Channel39 event ID register	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	Channel39 task ID register	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	Channel40 event ID register	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	Channel40 task ID register	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	Channel41 event ID register	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	Channel41 task ID register	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	Channel42 event ID register	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	Channel42 task ID register	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	Channel43 event ID register	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	Channel43 task ID register	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	Channel44 event ID register	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	Channel44 task ID register	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	Channel45 event ID register	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	Channel45 task ID register	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	Channel46 event ID register	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	Channel46 task ID register	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	Channel47 event ID register	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	Channel47 task ID register	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	Channel48 event ID register	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	Channel48 task ID register	0x019C	R/W
SOC_ETM_CH49_EVT_ID_REG	Channel49 event ID register	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	Channel49 task ID register	0x01A4	R/W

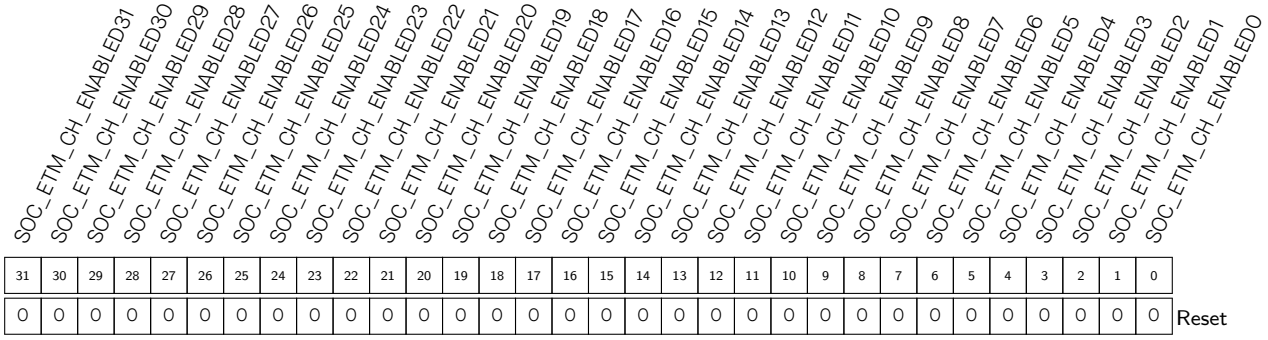
Name	Description	Address	Access
SOC_ETM_EVT_ST0_CLR_REG	Event status clear register	0x01AC	WT
SOC_ETM_EVT_ST1_CLR_REG	Event status clear register	0x01B4	WT
SOC_ETM_EVT_ST2_CLR_REG	Event status clear register	0x01BC	WT
SOC_ETM_EVT_ST3_CLR_REG	Event status clear register	0x01C4	WT
SOC_ETM_EVT_ST4_CLR_REG	Event status clear register	0x01CC	WT
SOC_ETM_EVT_ST5_CLR_REG	Event status clear register	0x01D4	WT
SOC_ETM_EVT_ST6_CLR_REG	Event status clear register	0x01DC	WT
SOC_ETM_EVT_ST7_CLR_REG	Event status clear register	0x01E4	WT
SOC_ETM_TASK_ST0_CLR_REG	Task status clear register	0x01EC	WT
SOC_ETM_TASK_ST1_CLR_REG	Task status clear register	0x01F4	WT
SOC_ETM_TASK_ST2_CLR_REG	Task status clear register	0x01FC	WT
SOC_ETM_TASK_ST3_CLR_REG	Task status clear register	0x0204	WT
SOC_ETM_TASK_ST4_CLR_REG	Task status clear register	0x020C	WT
SOC_ETM_TASK_ST5_CLR_REG	Task status clear register	0x0214	WT
SOC_ETM_TASK_ST6_CLR_REG	Task status clear register	0x021C	WT
SOC_ETM_CLK_EN_REG	ETM clock enable register	0x0220	R/W
Version Register			
SOC_ETM_DATE_REG	Version control register	0x0224	R/W

11.5 Registers

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

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

Register 11.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



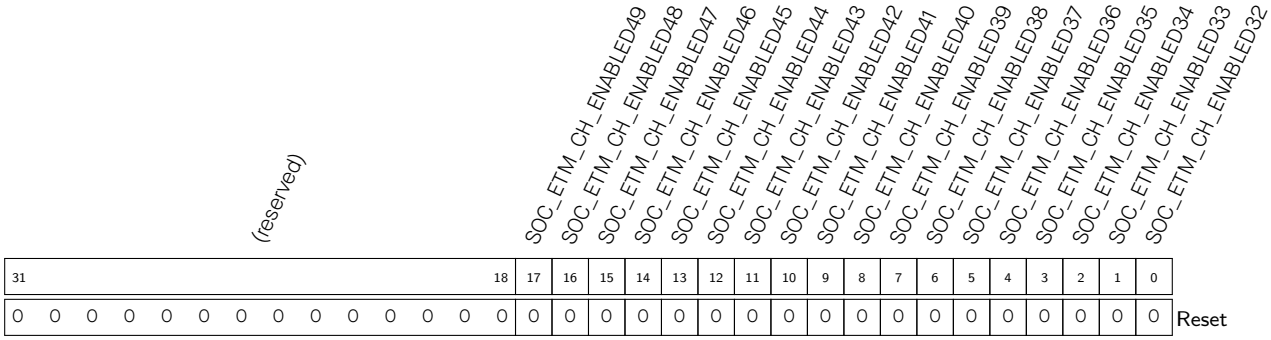
SOC_ETM_CH_ENABLED*n* (*n*: 0-31) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 11.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)



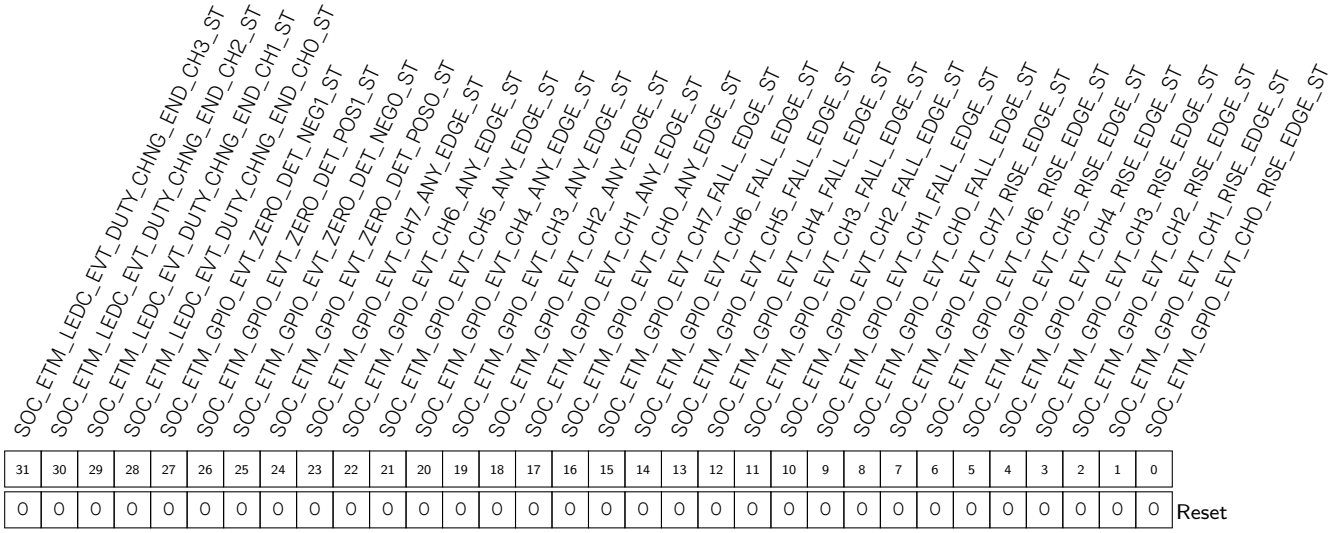
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) Represents the status of channel*n*.

0: Disabled

1: Enabled

(R/WTC/SS)

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST Represents the status of GPIO_EVT_CH0_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST Represents the status of GPIO_EVT_CH1_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST Represents the status of GPIO_EVT_CH2_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST Represents the status of GPIO_EVT_CH3_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST Represents the status of GPIO_EVT_CH4_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST Represents the status of GPIO_EVT_CH5_RISE_EDGE.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST Represents the status of GPIO_EVT_CH6_RISE_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST Represents the status of GPIO_EVT_CH7_RISE_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST Represents the status of GPIO_EVT_CH0_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST Represents the status of GPIO_EVT_CH1_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST Represents the status of GPIO_EVT_CH2_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST Represents the status of GPIO_EVT_CH3_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST Represents the status of GPIO_EVT_CH4_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST Represents the status of GPIO_EVT_CH5_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST Represents the status of GPIO_EVT_CH6_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST Represents the status of GPIO_EVT_CH7_FALL_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST Represents the status of GPIO_EVT_CH0_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST Represents the status of GPIO_EVT_CH1_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST Represents the status of GPIO_EVT_CH2_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST Represents the status of GPIO_EVT_CH3_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST Represents the status of GPIO_EVT_CH4_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST Represents the status of GPIO_EVT_CH5_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST Represents the status of GPIO_EVT_CH6_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST Represents the status of GPIO_EVT_CH7_ANY_EDGE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST Represents the status of GPIO_EVT_ZERO_DET_POS0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST Represents the status of GPIO_EVT_ZERO_DET_NEG0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST Represents the status of GPIO_EVT_ZERO_DET_POS1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST Represents the status of GPIO_EVT_ZERO_DET_NEG1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.3. SOC_ETM_EVT_ST0_REG (0x01A8)

Continued from the previous page...

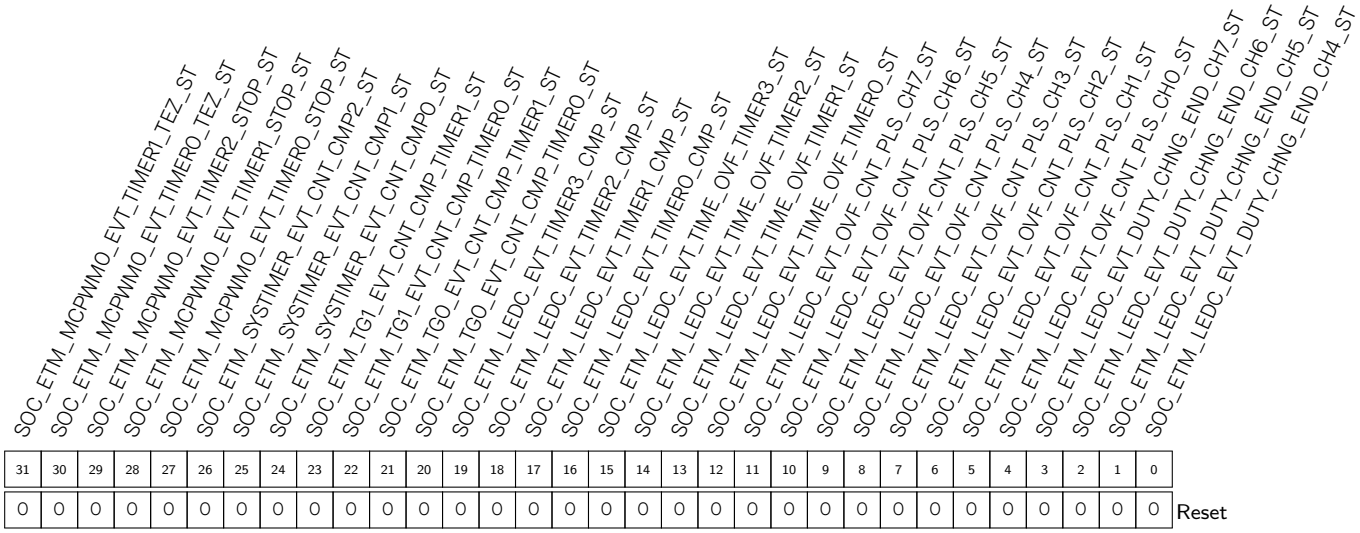
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH3.
0: Not received
1: Received
(R/WTC/SS)

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH4.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH5.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH6.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST Represents the status of LEDC_EVT_DUTY_CHNG_END_CH7.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH4.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH5.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH6.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST Represents the status of LEDC_EVT_OVF_CNT_PLS_CH7.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST Represents the status of LEDC_EVT_TIME_OVF_TIMER3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST Represents the status of LEDC_EVT_TIMER0_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST Represents the status of LEDC_EVT_TIMER1_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST Represents the status of LEDC_EVT_TIMER2_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST Represents the status of LEDC_EVT_TIMER3_CMP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST Represents the status of TG0_EVT_CNT_CMP_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST Represents the status of TG0_EVT_CNT_CMP_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST Represents the status of TG1_EVT_CNT_CMP_TIMER0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST Represents the status of TG1_EVT_CNT_CMP_TIMER1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST Represents the status of SYSTIMER_EVT_CNT_CMP0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST Represents the status of SYSTIMER_EVT_CNT_CMP1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST Represents the status of SYSTIMER_EVT_CNT_CMP2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST Represents the status of MCPWM0_EVT_TIMER0_STOP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST Represents the status of MCPWM0_EVT_TIMER1_STOP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST Represents the status of MCPWM0_EVT_TIMER2_STOP.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

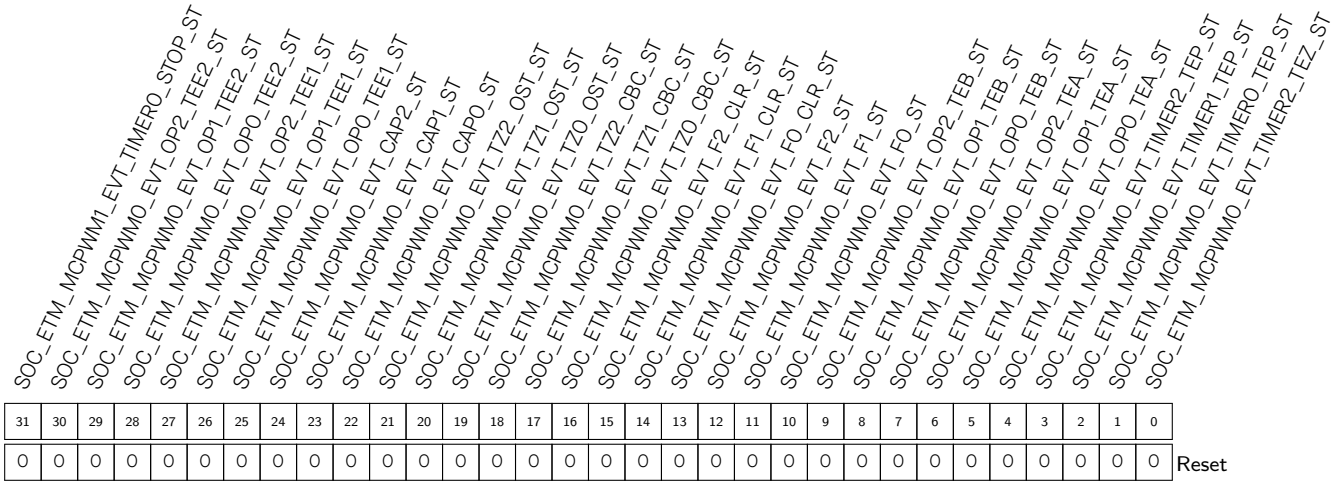
Register 11.4. SOC_ETM_EVT_ST1_REG (0x01B0)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST Represents the status of MCPWM0_EVT_TIMER0_TEZ.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST Represents the status of MCPWM0_EVT_TIMER1_TEZ.
0: Not received
1: Received
(R/WTC/SS)

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST Represents the status of MCPWM0_EVT_TIMER2_TEZ.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST Represents the status of MCPWM0_EVT_TIMER0_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST Represents the status of MCPWM0_EVT_TIMER1_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST Represents the status of MCPWM0_EVT_TIMER2_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST Represents the status of MCPWM0_EVT_OP0_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST Represents the status of MCPWM0_EVT_OP1_TEA.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST Represents the status of MCPWM0_EVT_OP2_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST Represents the status of MCPWM0_EVT_OP0_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST Represents the status of MCPWM0_EVT_OP1_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST Represents the status of MCPWM0_EVT_OP2_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_ST Represents the status of MCPWM0_EVT_F0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_ST Represents the status of MCPWM0_EVT_F1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_ST Represents the status of MCPWM0_EVT_F2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST Represents the status of MCPWM0_EVT_F0_CLR.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST Represents the status of MCPWM0_EVT_F1_CLR.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_F2_CLR_ST Represents the status of MCPWM0_EVT_F2_CLR.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST Represents the status of MCPWM0_EVT_TZ0_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST Represents the status of MCPWM0_EVT_TZ1_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST Represents the status of MCPWM0_EVT_TZ2_CBC.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST Represents the status of MCPWM0_EVT_TZ0_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST Represents the status of MCPWM0_EVT_TZ1_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST Represents the status of MCPWM0_EVT_TZ2_OST.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP0_ST Represents the status of MCPWM0_EVT_CAP0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP1_ST Represents the status of MCPWM0_EVT_CAP1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.5. SOC_ETM_EVT_ST2_REG (0x01B8)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_CAP2_ST Represents the status of MCPWM0_EVT_CAP2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST Represents the status of MCPWM0_EVT_OP0_TEE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST Represents the status of MCPWM0_EVT_OP1_TEE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST Represents the status of MCPWM0_EVT_OP2_TEE1.
0: Not received
1: Received
(R/WTC/SS)

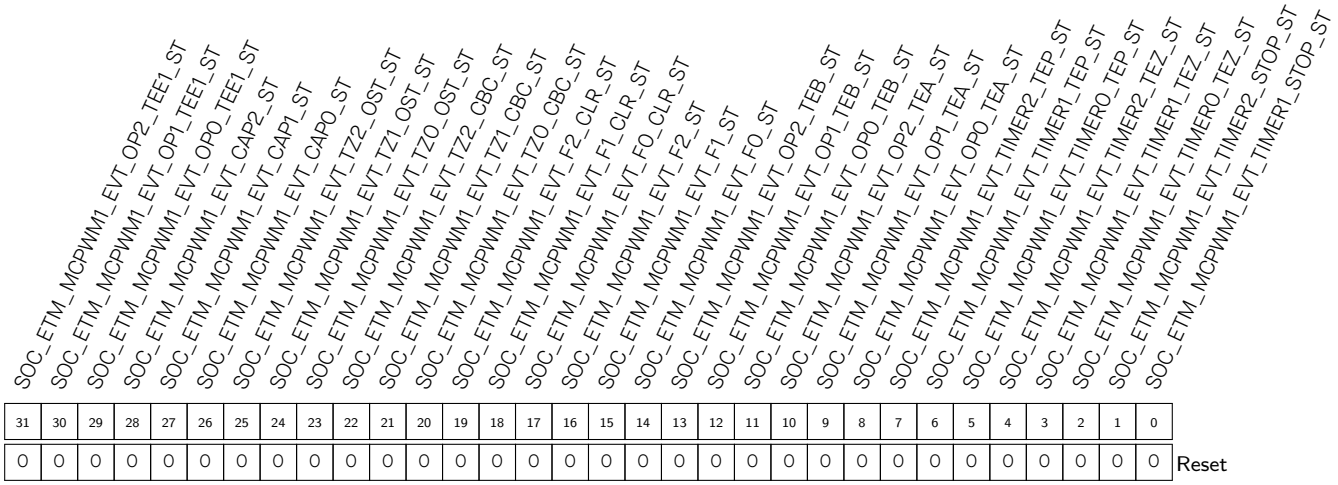
SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST Represents the status of MCPWM0_EVT_OP0_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST Represents the status of MCPWM0_EVT_OP1_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST Represents the status of MCPWM0_EVT_OP2_TEE2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST Represents the status of MCPWM1_EVT_TIMER0_STOP.
0: Not received
1: Received
(R/WTC/SS)

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST Represents the status of MCPWM1_EVT_TIMER1_STOP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST Represents the status of MCPWM1_EVT_TIMER2_STOP.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMERO_TEZ_ST Represents the status of MCPWM1_EVT_TIMERO_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST Represents the status of MCPWM1_EVT_TIMER1_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST Represents the status of MCPWM1_EVT_TIMER2_TEZ.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMERO_TEP_ST Represents the status of MCPWM1_EVT_TIMERO_TEP.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST Represents the status of MCPWM1_EVT_TIMER1_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST Represents the status of MCPWM1_EVT_TIMER2_TEP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST Represents the status of MCPWM1_EVT_OP0_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST Represents the status of MCPWM1_EVT_OP1_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST Represents the status of MCPWM1_EVT_OP2_TEA.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST Represents the status of MCPWM1_EVT_OP0_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST Represents the status of MCPWM1_EVT_OP1_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST Represents the status of MCPWM1_EVT_OP2_TEB.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_ST Represents the status of MCPWM1_EVT_F0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_F1_ST Represents the status of MCPWM1_EVT_F1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F2_ST Represents the status of MCPWM1_EVT_F2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST Represents the status of MCPWM1_EVT_F0_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST Represents the status of MCPWM1_EVT_F1_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST Represents the status of MCPWM1_EVT_F2_CLR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST Represents the status of MCPWM1_EVT_TZ0_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST Represents the status of MCPWM1_EVT_TZ1_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST Represents the status of MCPWM1_EVT_TZ2_CBC.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST Represents the status of MCPWM1_EVT_TZ0_OST.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.6. SOC_ETM_EVT_ST3_REG (0x01C0)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST Represents the status of MCPWM1_EVT_TZ1_OST.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST Represents the status of MCPWM1_EVT_TZ2_OST.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP0_ST Represents the status of MCPWM1_EVT_CAP0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP1_ST Represents the status of MCPWM1_EVT_CAP1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP2_ST Represents the status of MCPWM1_EVT_CAP2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST Represents the status of MCPWM1_EVT_OP0_TEE1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST Represents the status of MCPWM1_EVT_OP1_TEE1.

0: Not received

1: Received

(R/WTC/SS)

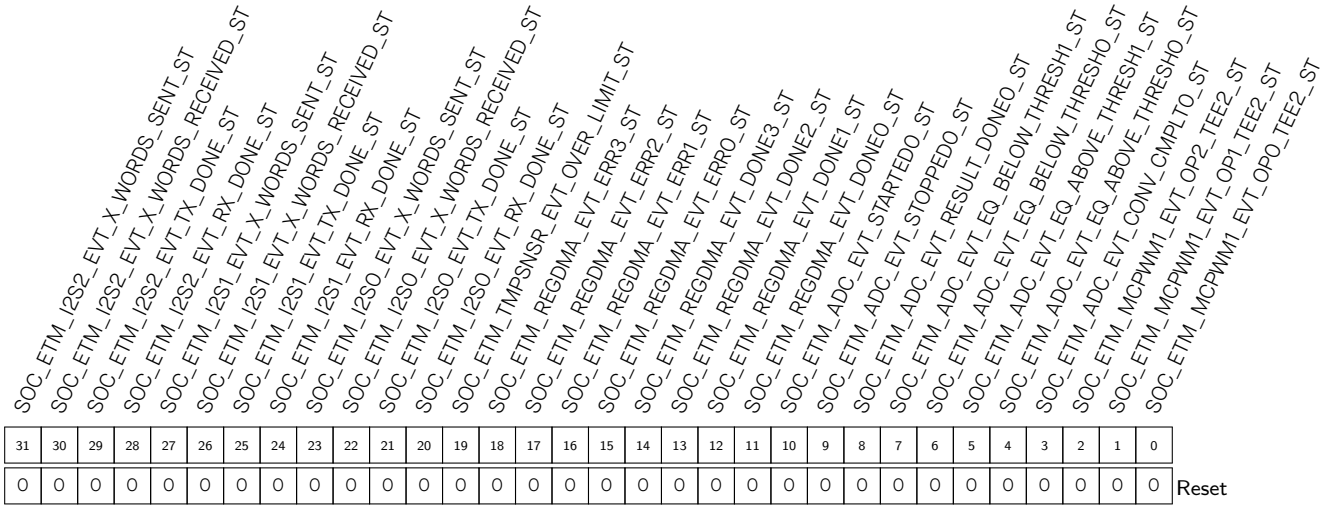
SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST Represents the status of MCPWM1_EVT_OP2_TEE1.

0: Not received

1: Received

(R/WTC/SS)

Register 11.7. SOC_ETM_EVT_ST4_REG (0x01C8)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST Represents the status of MCPWM1_EVT_OP0_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST Represents the status of MCPWM1_EVT_OP1_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST Represents the status of MCPWM1_EVT_OP2_TEE2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_CONV_CMPLT0_ST Represents the status of ADC_EVT_CONV_CMPLT0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST Represents the status of ADC_EVT_EQ_ABOVE_THRESH0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST Represents the status of ADC_EVT_EQ_ABOVE_THRESH1.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST Represents the status of ADC_EVT_EQ_BELOW_THRESH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST Represents the status of ADC_EVT_EQ_BELOW_THRESH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST Represents the status of ADC_EVT_RESULT_DONE0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_STOPPED0_ST Represents the status of ADC_EVT_STOPPED0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_ADC_EVT_STARTED0_ST Represents the status of ADC_EVT_STARTED0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE0_ST Represents the status of REGDMA_EVT_DONE0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE1_ST Represents the status of REGDMA_EVT_DONE1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE2_ST Represents the status of REGDMA_EVT_DONE2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_REGDMA_EVT_DONE3_ST Represents the status of REGDMA_EVT_DONE3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR0_ST Represents the status of REGDMA_EVT_ERR0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR1_ST Represents the status of REGDMA_EVT_ERR1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR2_ST Represents the status of REGDMA_EVT_ERR2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR3_ST Represents the status of REGDMA_EVT_ERR3.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST Represents the status of TMPSNSR_EVT_OVER_LIMIT.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_RX_DONE_ST Represents the status of I2S0_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_TX_DONE_ST Represents the status of I2S0_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S0_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.7. SOC_ETM_EVT_ST4_REG (0x01C8)

Continued from the previous page...

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST Represents the status of I2S0_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_RX_DONE_ST Represents the status of I2S1_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_TX_DONE_ST Represents the status of I2S1_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S1_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST Represents the status of I2S1_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

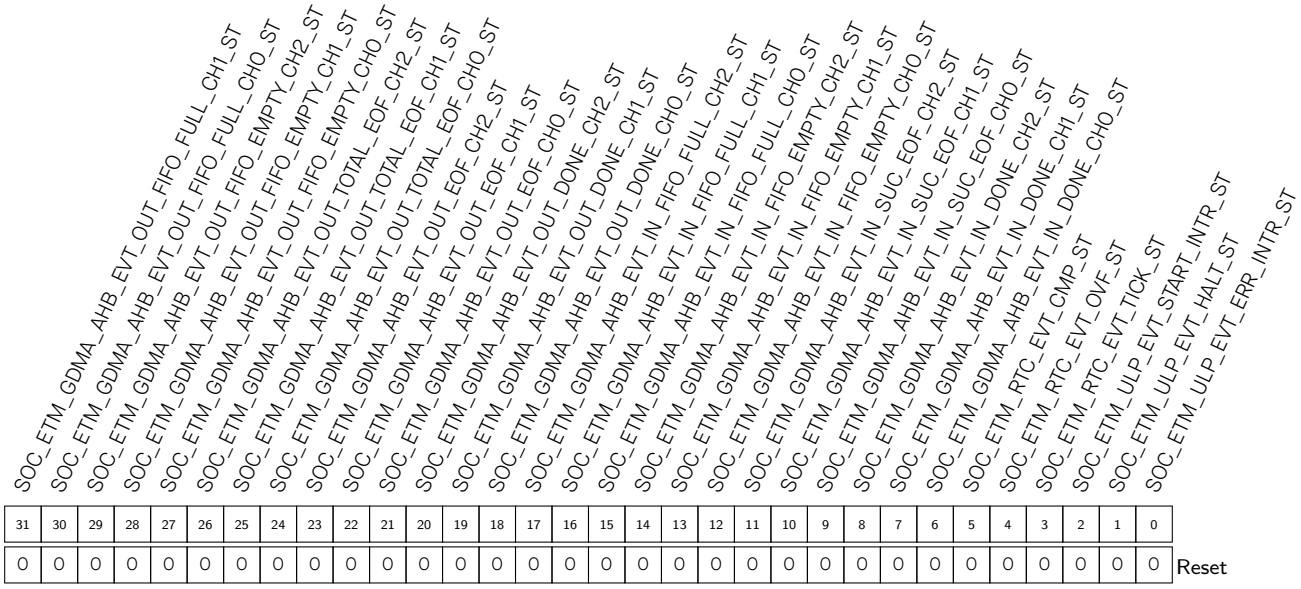
SOC_ETM_I2S2_EVT_RX_DONE_ST Represents the status of I2S2_EVT_RX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_TX_DONE_ST Represents the status of I2S2_EVT_TX_DONE.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST Represents the status of I2S2_EVT_X_WORDS_RECEIVED.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST Represents the status of I2S2_EVT_X_WORDS_SENT.
0: Not received
1: Received
(R/WTC/SS)

Register 11.8. SOC_ETM_EVT_ST5_REG (0x01D0)



SOC_ETM_ULP_EVT_ERR_INTR_ST Represents the status of ULP_EVT_ERR_INTR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ULP_EVT_HALT_ST Represents the status of ULP_EVT_HALT.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_ULP_EVT_START_INTR_ST Represents the status of ULP_EVT_START_INTR.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_TICK_ST Represents the status of RTC_EVT_TICK.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_OVF_ST Represents the status of RTC_EVT_OVF.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_RTC_EVT_CMP_ST Represents the status of RTC_EVT_CMP.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST Represents the status of GDMA_AHB_EVT_IN_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_IN_SUC_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST Represents the status of GDMA_AHB_EVT_IN_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.8. SOC_ETM_EVT_ST5_REG (0x01D0)

Continued from the previous page...

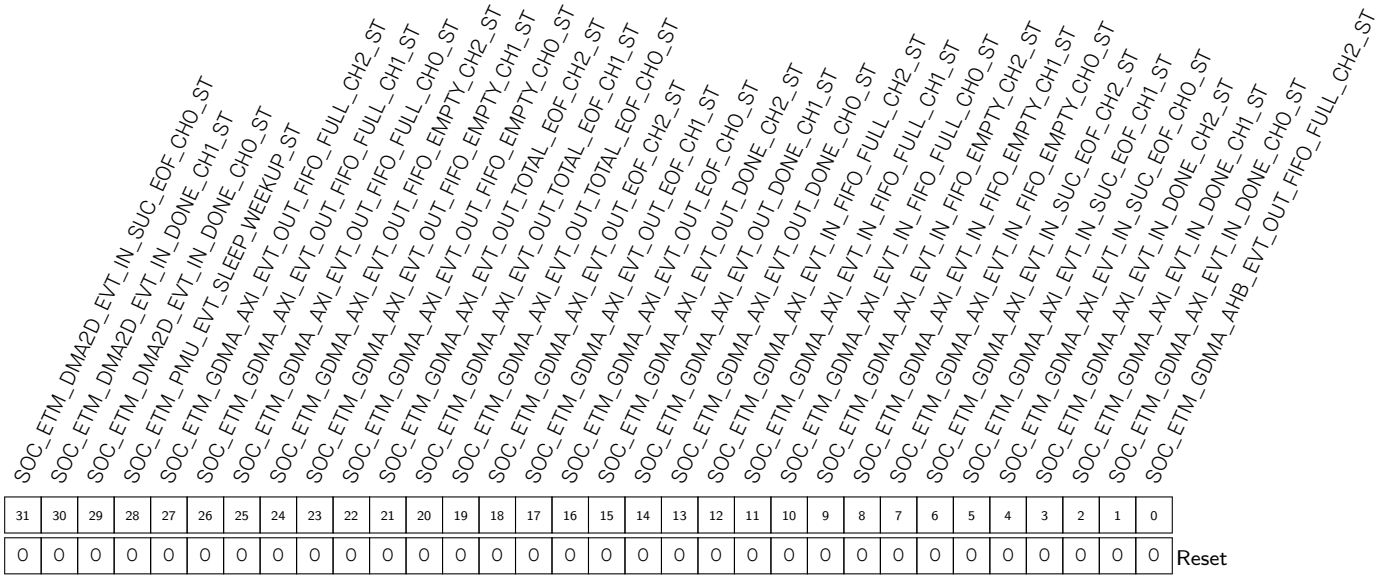
SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

Register 11.9. SOC_ETM_EVT_ST6_REG (0x01D8)



SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST Represents the status of GDMA_AHB_EVT_OUT_FIFO_FULL_CH2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH0.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH1.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST Represents the status of GDMA_AXI_EVT_IN_DONE_CH2.

0: Not received

1: Received

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH0.

0: Not received

1: Received

(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_IN_SUC_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST Represents the status of GDMA_AXI_EVT_IN_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH1.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.9. SOC_ETM_EVT_ST6_REG (0x01D8)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST Represents the status of GDMA_AXI_EVT_OUT_FIFO_FULL_CH2.
0: Not received
1: Received
(R/WTC/SS)

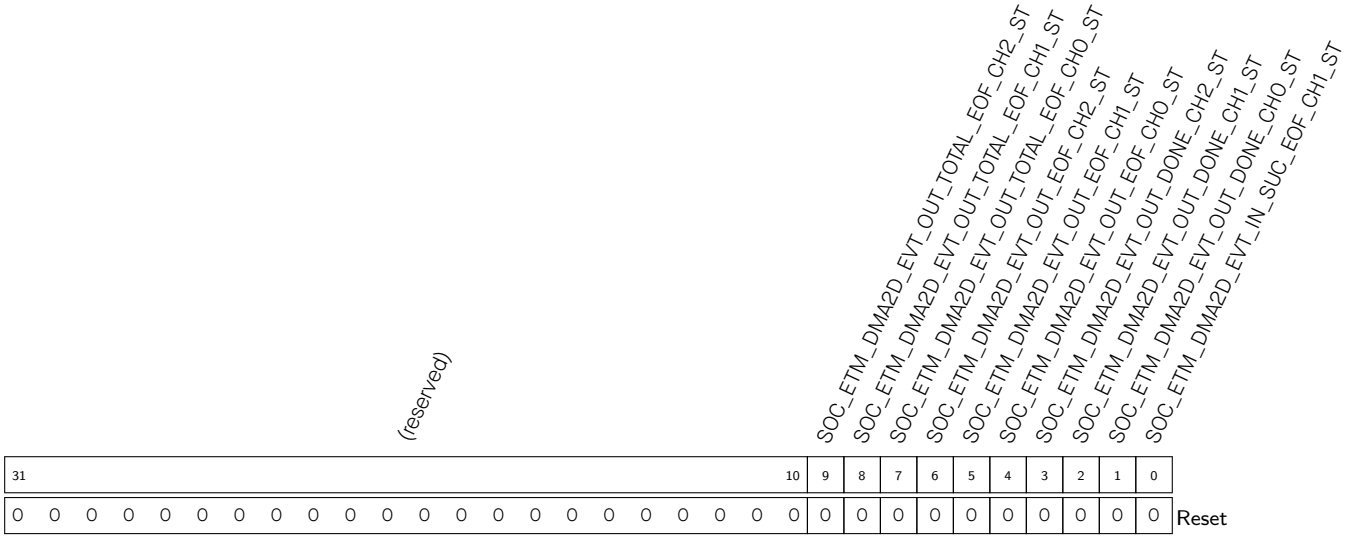
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST Represents the status of PMU_EVT_SLEEP_WEEKUP.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST Represents the status of DMA2D_EVT_IN_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST Represents the status of DMA2D_EVT_IN_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST Represents the status of DMA2D_EVT_IN_SUC_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Register 11.10. SOC_ETM_EVT_ST7_REG (0x01E0)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST Represents the status of DMA2D_EVT_IN_SUC_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST Represents the status of DMA2D_EVT_OUT_DONE_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST Represents the status of DMA2D_EVT_OUT_DONE_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST Represents the status of DMA2D_EVT_OUT_DONE_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST Represents the status of DMA2D_EVT_OUT_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

Continued on the next page...

Register 11.10. SOC_ETM_EVT_ST7_REG (0x01E0)

Continued from the previous page...

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST Represents the status of DMA2D_EVT_OUT_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

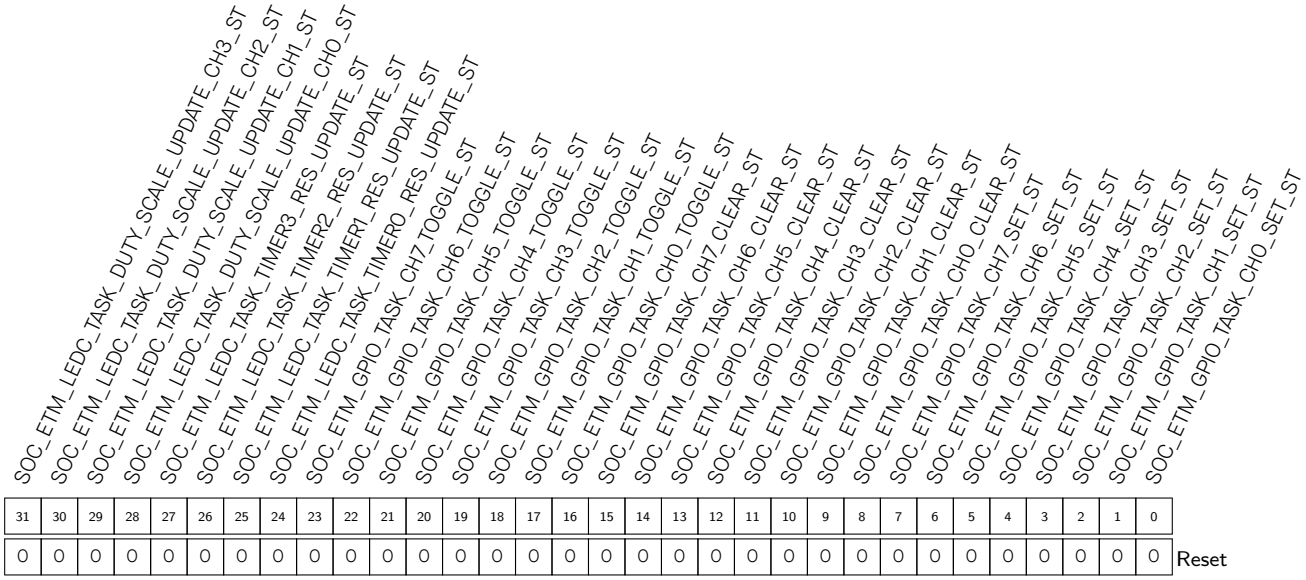
SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST Represents the status of DMA2D_EVT_OUT_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH0.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH1.
0: Not received
1: Received
(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST Represents the status of DMA2D_EVT_OUT_TOTAL_EOF_CH2.
0: Not received
1: Received
(R/WTC/SS)

Register 11.11. SOC_ETM_TASK_ST0_REG (0x01E8)



SOC_ETM_GPIO_TASK_CH0_SET_ST Represents the status of GPIO_TASK_CH0_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_SET_ST Represents the status of GPIO_TASK_CH1_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_SET_ST Represents the status of GPIO_TASK_CH2_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_SET_ST Represents the status of GPIO_TASK_CH3_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_SET_ST Represents the status of GPIO_TASK_CH4_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_SET_ST Represents the status of GPIO_TASK_CH5_SET.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_SET_ST Represents the status of GPIO_TASK_CH6_SET.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH7_SET_ST Represents the status of GPIO_TASK_CH7_SET.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST Represents the status of GPIO_TASK_CH0_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST Represents the status of GPIO_TASK_CH1_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST Represents the status of GPIO_TASK_CH2_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST Represents the status of GPIO_TASK_CH3_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST Represents the status of GPIO_TASK_CH4_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST Represents the status of GPIO_TASK_CH5_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST Represents the status of GPIO_TASK_CH6_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST Represents the status of GPIO_TASK_CH7_CLEAR.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST Represents the status of GPIO_TASK_CH0_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST Represents the status of GPIO_TASK_CH1_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST Represents the status of GPIO_TASK_CH2_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST Represents the status of GPIO_TASK_CH3_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST Represents the status of GPIO_TASK_CH4_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST Represents the status of GPIO_TASK_CH5_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST Represents the status of GPIO_TASK_CH6_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST Represents the status of GPIO_TASK_CH7_TOGGLE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER0_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.11. SOC_ETM_TASK_ST0_REG (0x01E8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER1_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER2_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST Represents the status of LEDC_TASK_TIMER3_RES_UPDATE.
0: Not generated
1: Generated
(R/WTC/SS)

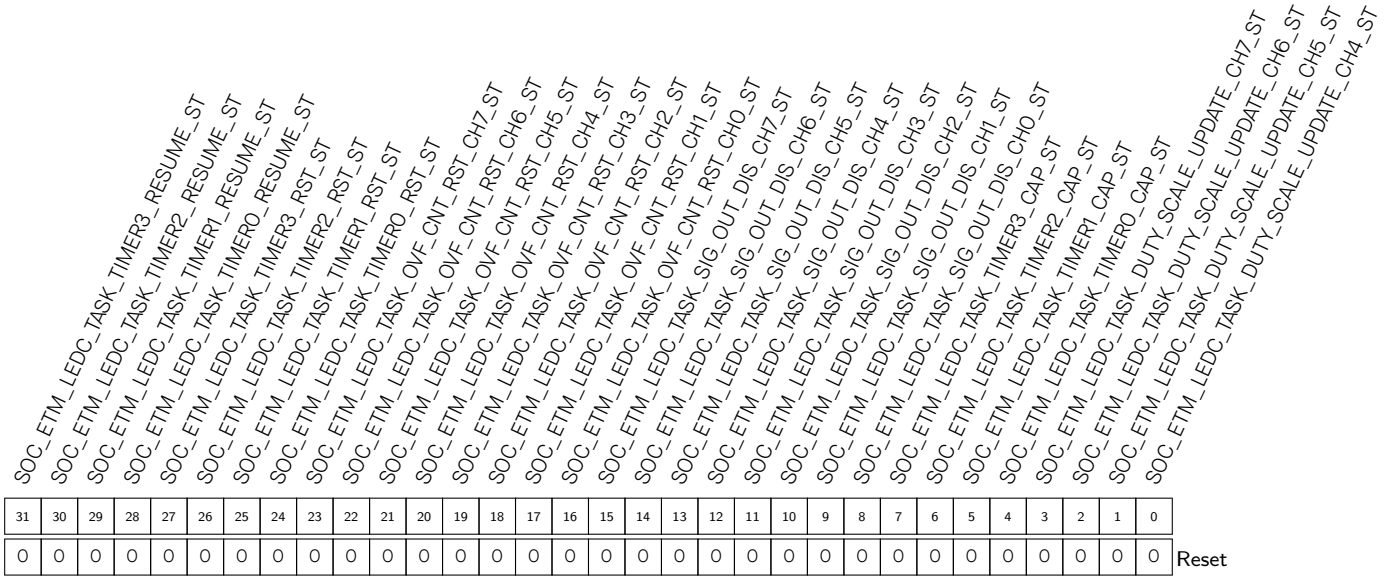
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

Register 11.12. SOC_ETM_TASK_ST1_REG (0x01F0)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH4.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH5.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH6.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST Represents the status of LEDC_TASK_DUTY_SCALE_UPDATE_CH7.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST Represents the status of LEDC_TASK_TIMER0_CAP.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST Represents the status of LEDC_TASK_TIMER1_CAP.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST Represents the status of LEDC_TASK_TIMER2_CAP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST Represents the status of LEDC_TASK_TIMER3_CAP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST Represents the status of LEDC_TASK_SIG_OUT_DIS_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST Represents the status of LEDC_TASK_OVF_CNT_RST_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST Represents the status of LEDC_TASK_TIMER0_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST Represents the status of LEDC_TASK_TIMER1_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST Represents the status of LEDC_TASK_TIMER2_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST Represents the status of LEDC_TASK_TIMER3_RST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST Represents the status of LEDC_TASK_TIMER0_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST Represents the status of LEDC_TASK_TIMER1_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

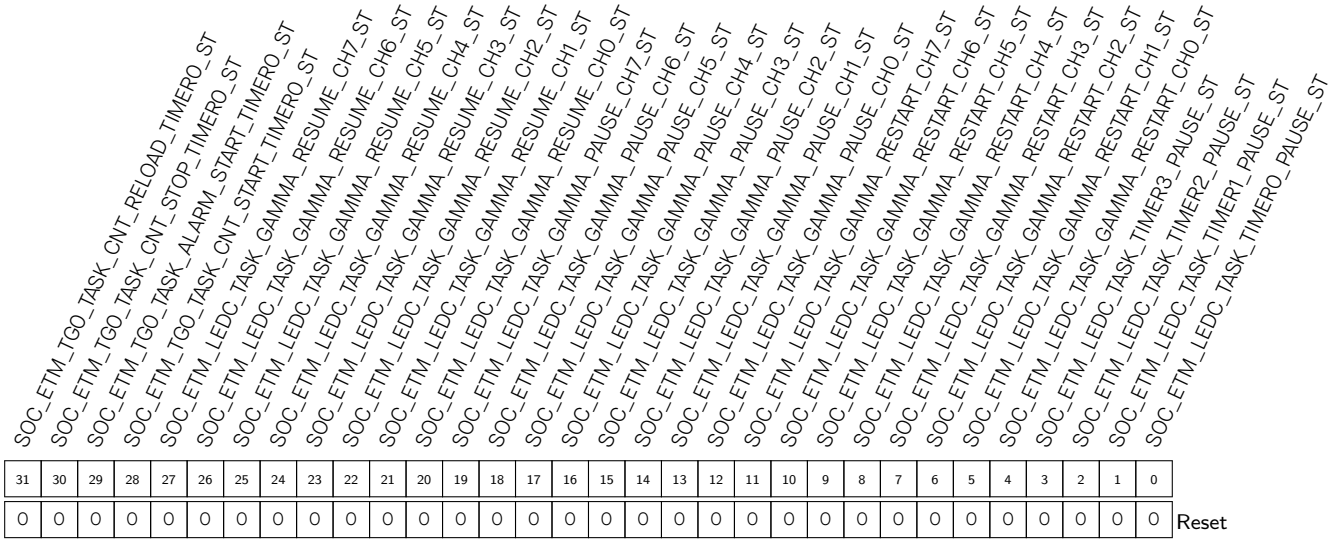
Register 11.12. SOC_ETM_TASK_ST1_REG (0x01F0)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST Represents the status of LEDC_TASK_TIMER2_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST Represents the status of LEDC_TASK_TIMER3_RESUME.
0: Not generated
1: Generated
(R/WTC/SS)

Register 11.13. SOC_ETM_TASK_ST2_REG (0x01F8)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST Represents the status of LEDC_TASK_TIMER0_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST Represents the status of LEDC_TASK_TIMER1_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST Represents the status of LEDC_TASK_TIMER2_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST Represents the status of LEDC_TASK_TIMER3_PAUSE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH0.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST Represents the status of LEDC_TASK_GAMMA_RESTART_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST Represents the status of LEDC_TASK_GAMMA_PAUSE_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH3.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH4.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH5.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH6.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.13. SOC_ETM_TASK_ST2_REG (0x01F8)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST Represents the status of LEDC_TASK_GAMMA_RESUME_CH7.
0: Not generated
1: Generated
(R/WTC/SS)

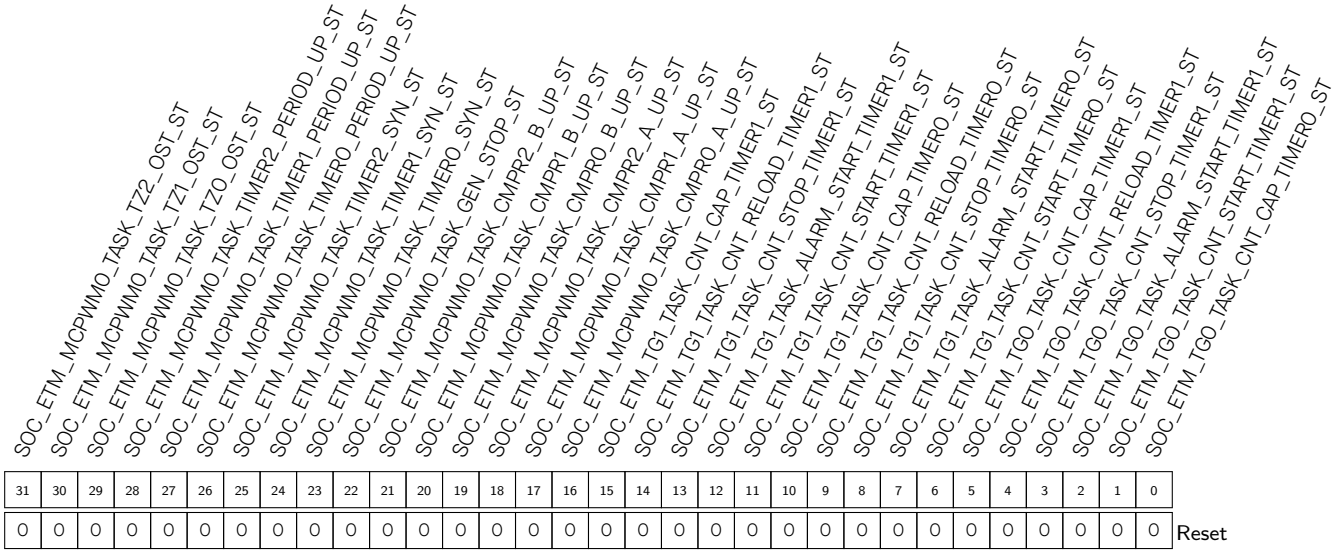
SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST Represents the status of TGO_TASK_CNT_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST Represents the status of TGO_TASK_ALARM_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST Represents the status of TGO_TASK_CNT_STOP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST Represents the status of TGO_TASK_CNT_RELOAD_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

Register 11.14. SOC_ETM_TASK_ST3_REG (0x0200)



SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST Represents the status of TG0_TASK_CNT_CAP_TIMER0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST Represents the status of TG0_TASK_CNT_START_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST Represents the status of TG0_TASK_ALARM_START_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST Represents the status of TG0_TASK_CNT_STOP_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST Represents the status of TG0_TASK_CNT_RELOAD_TIMER1.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST Represents the status of TG0_TASK_CNT_CAP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST Represents the status of TG1_TASK_CNT_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST Represents the status of TG1_TASK_ALARM_START_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST Represents the status of TG1_TASK_CNT_STOP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST Represents the status of TG1_TASK_CNT_RELOAD_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST Represents the status of TG1_TASK_CNT_CAP_TIMER0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST Represents the status of TG1_TASK_CNT_START_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST Represents the status of TG1_TASK_ALARM_START_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST Represents the status of TG1_TASK_CNT_STOP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST Represents the status of TG1_TASK_CNT_RELOAD_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST Represents the status of TG1_TASK_CNT_CAP_TIMER1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST Represents the status of MCPWM0_TASK_CMPR0_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST Represents the status of MCPWM0_TASK_CMPR1_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST Represents the status of MCPWM0_TASK_CMPR2_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST Represents the status of MCPWM0_TASK_CMPR0_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST Represents the status of MCPWM0_TASK_CMPR1_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST Represents the status of MCPWM0_TASK_CMPR2_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST Represents the status of MCPWM0_TASK_GEN_STOP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST Represents the status of MCPWM0_TASK_TIMER0_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST Represents the status of MCPWM0_TASK_TIMER1_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST Represents the status of MCPWM0_TASK_TIMER2_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER0_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.14. SOC_ETM_TASK_ST3_REG (0x0200)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER1_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

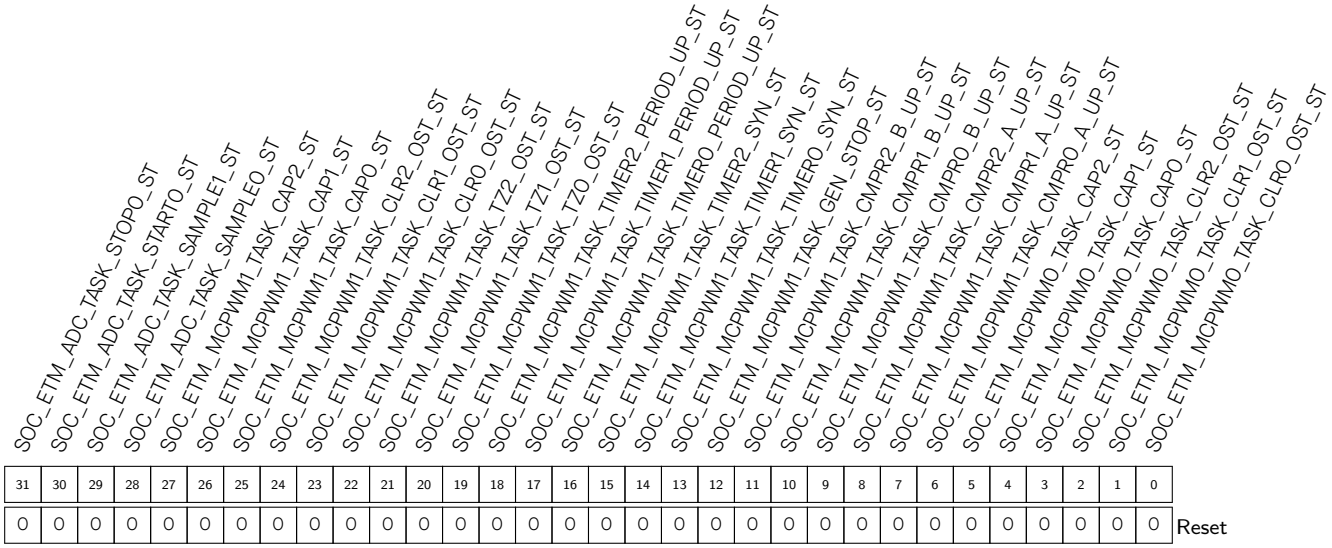
SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST Represents the status of MCPWM0_TASK_TIMER2_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST Represents the status of MCPWM0_TASK_TZ0_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST Represents the status of MCPWM0_TASK_TZ1_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST Represents the status of MCPWM0_TASK_TZ2_OST.
0: Not generated
1: Generated
(R/WTC/SS)

Register 11.15. SOC_ETM_TASK_ST4_REG (0x0208)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST Represents the status of MCPWM0_TASK_CLR0_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST Represents the status of MCPWM0_TASK_CLR1_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST Represents the status of MCPWM0_TASK_CLR2_OST.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP0_ST Represents the status of MCPWM0_TASK_CAP0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP1_ST Represents the status of MCPWM0_TASK_CAP1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP2_ST Represents the status of MCPWM0_TASK_CAP2.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST Represents the status of MCPWM1_TASK_CMPR0_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST Represents the status of MCPWM1_TASK_CMPR1_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST Represents the status of MCPWM1_TASK_CMPR2_A_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST Represents the status of MCPWM1_TASK_CMPR0_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST Represents the status of MCPWM1_TASK_CMPR1_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST Represents the status of MCPWM1_TASK_CMPR2_B_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST Represents the status of MCPWM1_TASK_GEN_STOP.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST Represents the status of MCPWM1_TASK_TIMER0_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST Represents the status of MCPWM1_TASK_TIMER1_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST Represents the status of MCPWM1_TASK_TIMER2_SYN.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER0_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER1_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST Represents the status of MCPWM1_TASK_TIMER2_PERIOD_UP.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST Represents the status of MCPWM1_TASK_TZ0_OST.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST Represents the status of MCPWM1_TASK_TZ1_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST Represents the status of MCPWM1_TASK_TZ2_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST Represents the status of MCPWM1_TASK_CLR0_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST Represents the status of MCPWM1_TASK_CLR1_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST Represents the status of MCPWM1_TASK_CLR2_OST.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP0_ST Represents the status of MCPWM1_TASK_CAP0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP1_ST Represents the status of MCPWM1_TASK_CAP1.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.15. SOC_ETM_TASK_ST4_REG (0x0208)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CAP2_ST Represents the status of MCPWM1_TASK_CAP2.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE0_ST Represents the status of ADC_TASK_SAMPLE0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE1_ST Represents the status of ADC_TASK_SAMPLE1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ADC_TASK_START0_ST Represents the status of ADC_TASK_START0.

0: Not generated

1: Generated

(R/WTC/SS)

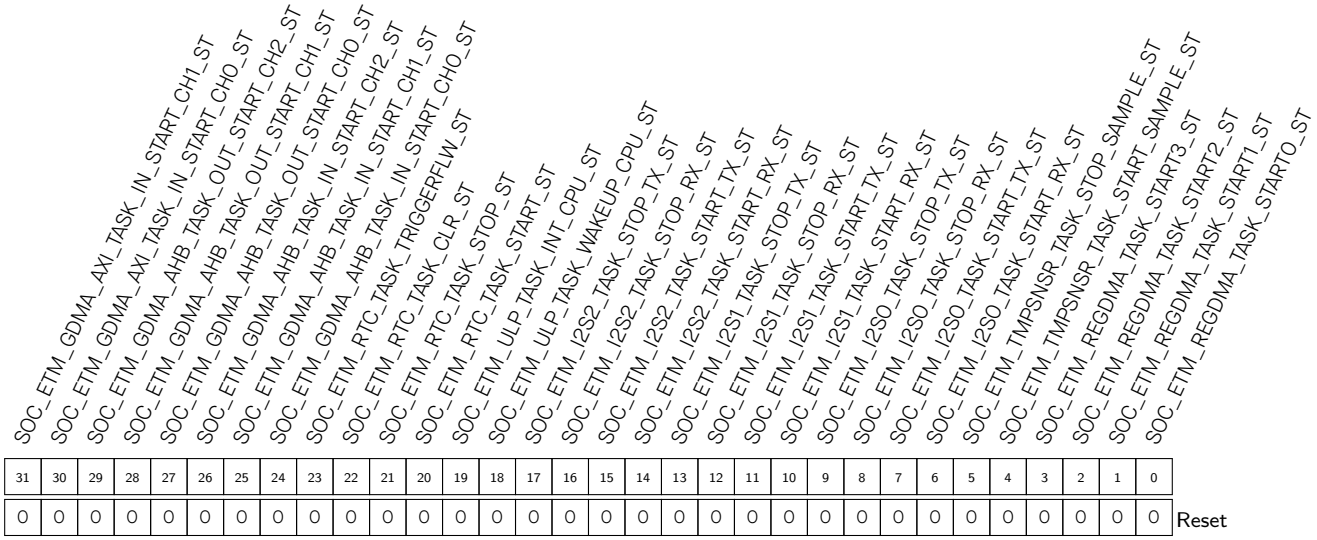
SOC_ETM_ADC_TASK_STOP0_ST Represents the status of ADC_TASK_STOP0.

0: Not generated

1: Generated

(R/WTC/SS)

Register 11.16. SOC_ETM_TASK_ST5_REG (0x0210)



SOC_ETM_REGDMA_TASK_START0_ST Represents the status of REGDMA_TASK_START0.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START1_ST Represents the status of REGDMA_TASK_START1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START2_ST Represents the status of REGDMA_TASK_START2.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START3_ST Represents the status of REGDMA_TASK_START3.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST Represents the status of TMPSNSR_TASK_START_SAMPLE.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST Represents the status of TMPSNSR_TASK_STOP_SAMPLE.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_I2S0_TASK_START_RX_ST Represents the status of I2S0_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_START_TX_ST Represents the status of I2S0_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_RX_ST Represents the status of I2S0_TASK_STOP_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_TX_ST Represents the status of I2S0_TASK_STOP_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_RX_ST Represents the status of I2S1_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_TX_ST Represents the status of I2S1_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_RX_ST Represents the status of I2S1_TASK_STOP_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_TX_ST Represents the status of I2S1_TASK_STOP_TX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S2_TASK_START_RX_ST Represents the status of I2S2_TASK_START_RX.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_I2S2_TASK_START_TX_ST Represents the status of I2S2_TASK_START_TX.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_I2S2_TASK_STOP_RX_ST Represents the status of I2S2_TASK_STOP_RX.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_I2S2_TASK_STOP_TX_ST Represents the status of I2S2_TASK_STOP_TX.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST Represents the status of ULP_TASK_WAKEUP_CPU.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_ULP_TASK_INT_CPU_ST Represents the status of ULP_TASK_INT_CPU.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_START_ST Represents the status of RTC_TASK_START.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_STOP_ST Represents the status of RTC_TASK_STOP.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_CLR_ST Represents the status of RTC_TASK_CLR.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST Represents the status of RTC_TASK_TRIGGERFLW.

0: Not generated

1: Generated

(R/WTC/SS)

Continued on the next page...

Register 11.16. SOC_ETM_TASK_ST5_REG (0x0210)

Continued from the previous page...

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST Represents the status of GDMA_AHB_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST Represents the status of GDMA_AHB_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST Represents the status of GDMA_AHB_TASK_IN_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

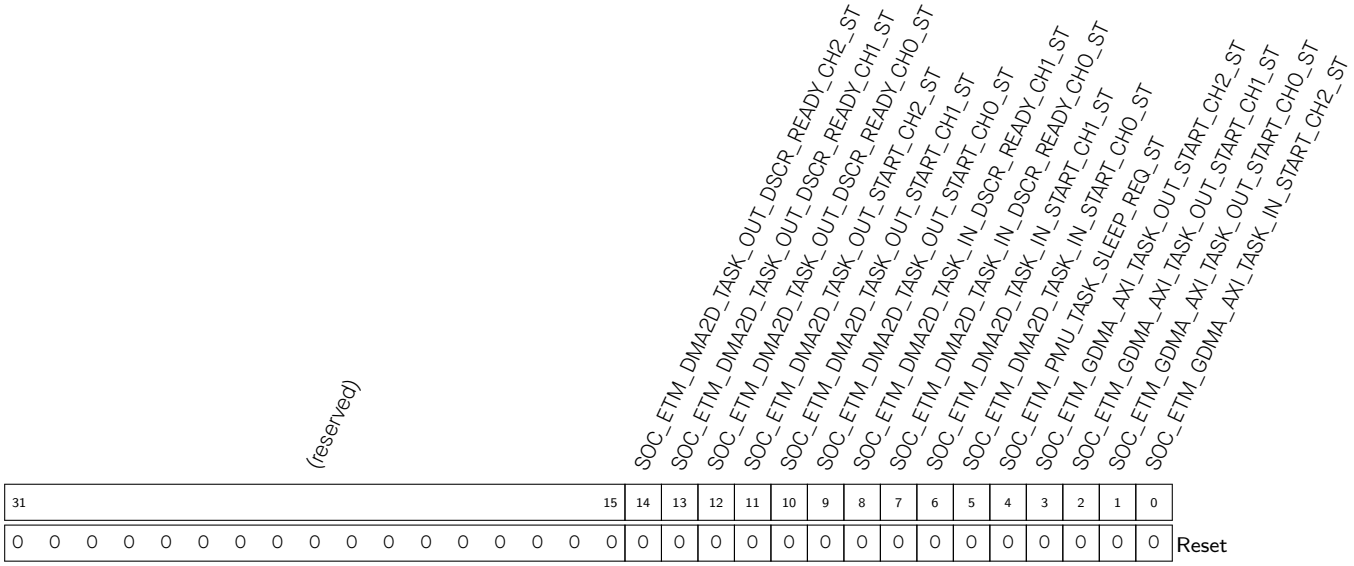
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST Represents the status of GDMA_AHB_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST Represents the status of GDMA_AXI_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST Represents the status of GDMA_AXI_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

Register 11.17. SOC_ETM_TASK_ST6_REG (0x0218)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST Represents the status of GDMA_AXI_TASK_IN_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST Represents the status of GDMA_AXI_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST Represents the status of PMU_TASK_SLEEP_REQ.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST Represents the status of DMA2D_TASK_IN_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.17. SOC_ETM_TASK_ST6_REG (0x0218)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST Represents the status of DMA2D_TASK_IN_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST Represents the status of DMA2D_TASK_IN_DSCR_READY_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST Represents the status of DMA2D_TASK_IN_DSCR_READY_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST Represents the status of DMA2D_TASK_OUT_START_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST Represents the status of DMA2D_TASK_OUT_START_CH1.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST Represents the status of DMA2D_TASK_OUT_START_CH2.
0: Not generated
1: Generated
(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH0.
0: Not generated
1: Generated
(R/WTC/SS)

Continued on the next page...

Register 11.17. SOC_ETM_TASK_ST6_REG (0x0218)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH1.

0: Not generated

1: Generated

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST Represents the status of DMA2D_TASK_OUT_DSCR_READY_CH2.

0: Not generated

1: Generated

(R/WTC/SS)

Register 11.18. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SOC_ETM_CH_ENABLE31	SOC_ETM_CH_ENABLE30	SOC_ETM_CH_ENABLE29	SOC_ETM_CH_ENABLE28	SOC_ETM_CH_ENABLE27	SOC_ETM_CH_ENABLE26	SOC_ETM_CH_ENABLE25	SOC_ETM_CH_ENABLE24	SOC_ETM_CH_ENABLE23	SOC_ETM_CH_ENABLE22	SOC_ETM_CH_ENABLE21	SOC_ETM_CH_ENABLE20	SOC_ETM_CH_ENABLE19	SOC_ETM_CH_ENABLE18	SOC_ETM_CH_ENABLE17	SOC_ETM_CH_ENABLE16	SOC_ETM_CH_ENABLE15	SOC_ETM_CH_ENABLE14	SOC_ETM_CH_ENABLE13	SOC_ETM_CH_ENABLE12	SOC_ETM_CH_ENABLE11	SOC_ETM_CH_ENABLE10	SOC_ETM_CH_ENABLE9	SOC_ETM_CH_ENABLE8	SOC_ETM_CH_ENABLE7	SOC_ETM_CH_ENABLE6	SOC_ETM_CH_ENABLE5	SOC_ETM_CH_ENABLE4	SOC_ETM_CH_ENABLE3	SOC_ETM_CH_ENABLE2	SOC_ETM_CH_ENABLE1	SOC_ETM_CH_ENABLE0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

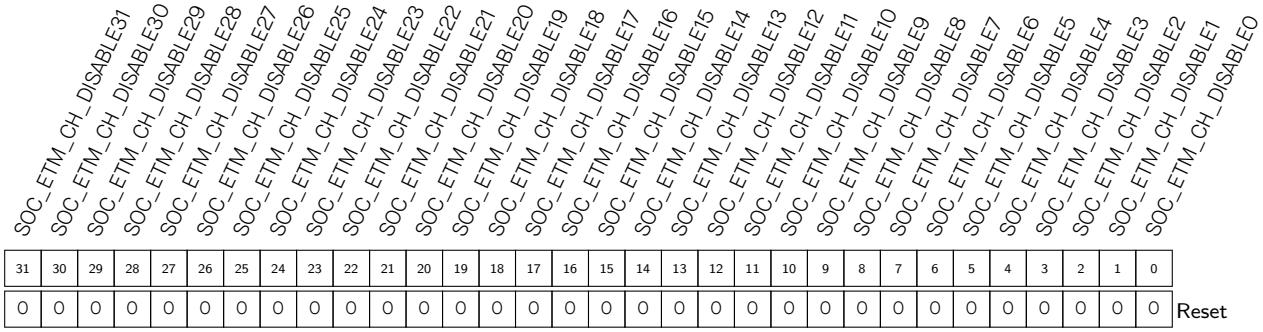
SOC_ETM_CH_ENABLE*n* (*n*: 0-31) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 11.19. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



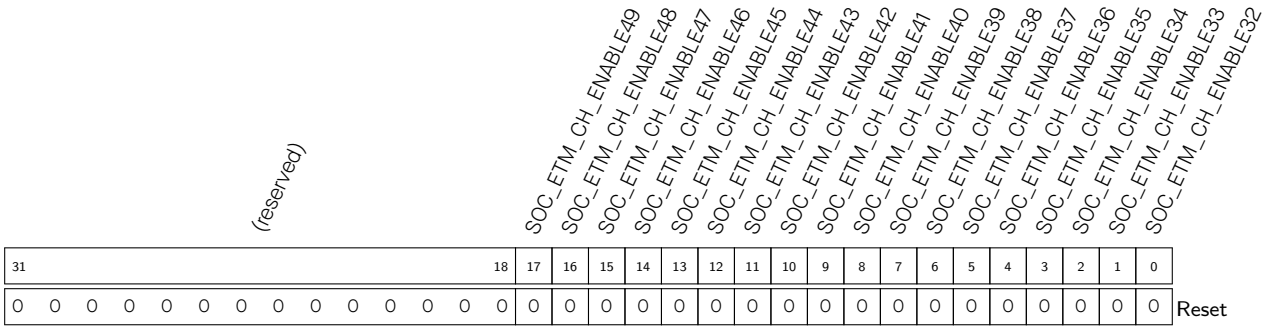
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) Configures whether to disable channel 0.

0: Invalid. No effect

1: Disable

(WT)

Register 11.20. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



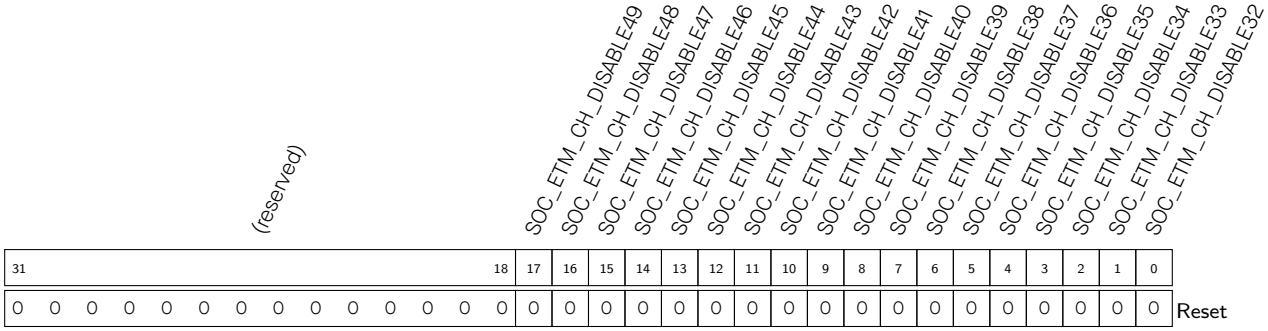
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) Configures whether to enable channel*n*.

0: Invalid. No effect

1: Enable

(WT)

Register 11.21. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



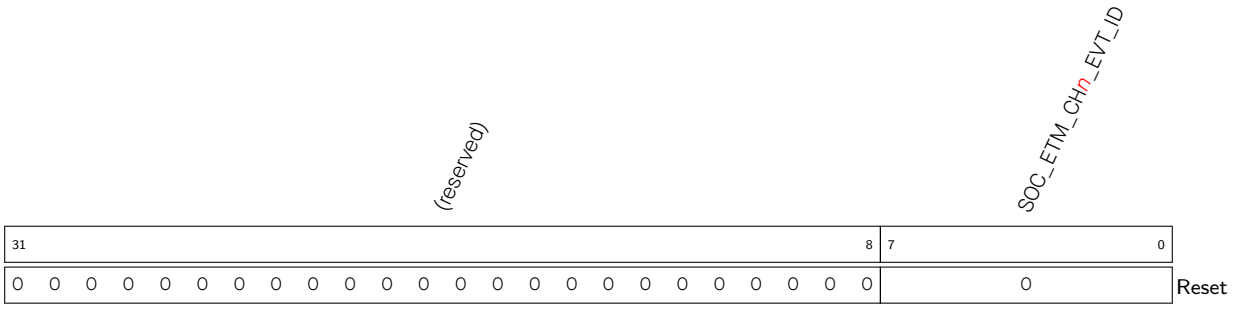
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) Configures whether to disable channel*n*.

0: Invalid. No effect

1: Disable

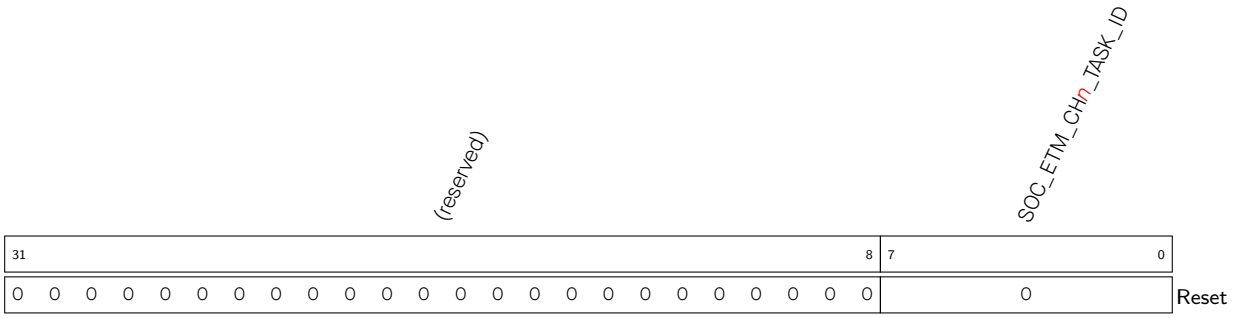
(WT)

Register 11.22. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8**n*)



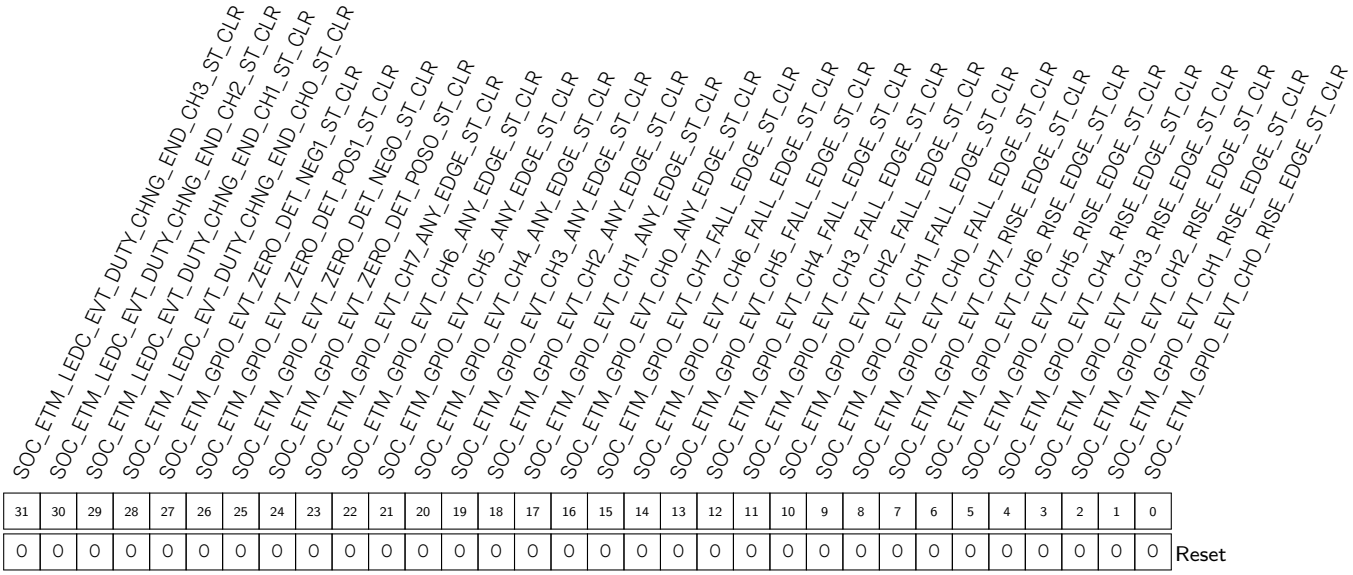
SOC_ETM_CH*n*_EVT_ID (*n*: 0-49) Configures the event ID of channel*n*. See Table 11.3-1. (R/W)

Register 11.23. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)



SOC_ETM_CH*n*_TASK_ID (*n*: 0-49) Configures the task ID of channel*n*. See Table 11.3-2. (R/W)

Register 11.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_RISE_EDGE.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_RISE_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_FALL_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH0_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH1_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH2_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH3_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH4_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH5_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH6_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR Configures whether to clear GPIO_EVT_CH7_ANY_EDGE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_POS0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_NEG0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

Continued from the previous page...

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_POS1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST_CLR Configures whether to clear GPIO_EVT_ZERO_DET_NEG1.
0: Invalid. No effect
1: Clear
(WT)

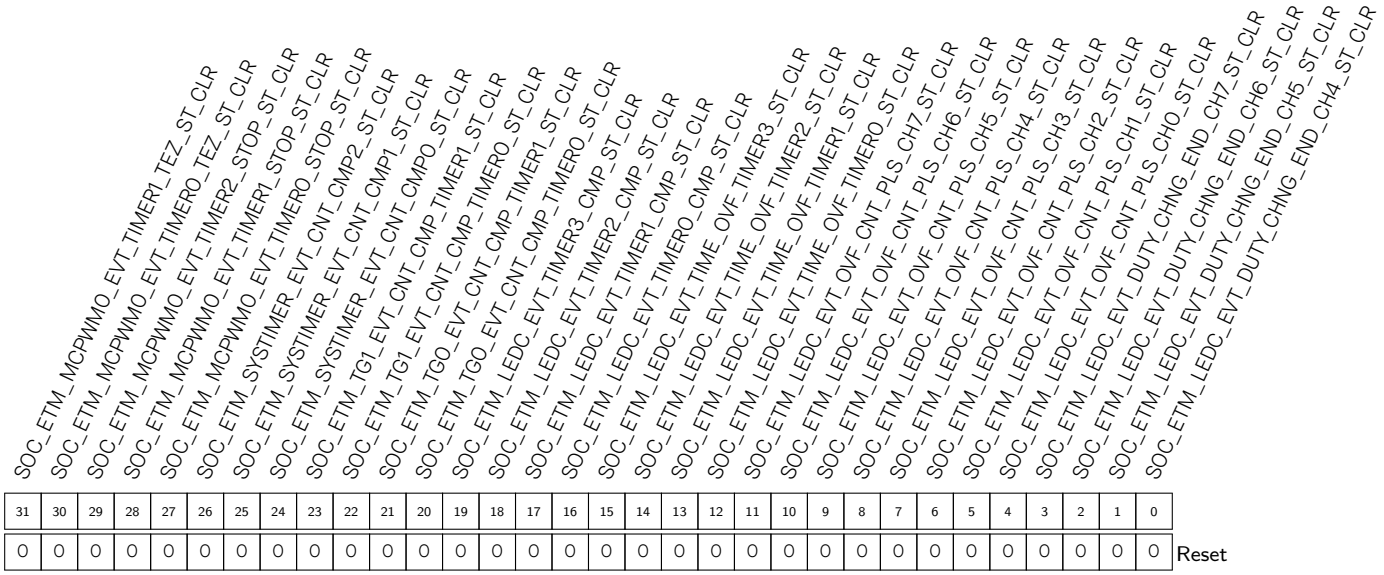
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH3.
0: Invalid. No effect
1: Clear
(WT)

Register 11.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH4.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH5.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH6.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST_CLR Configures whether to clear LEDC_EVT_DUTY_CHNG_END_CH7.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST_CLR Configures whether to clear LEDC_EVT_OVF_CNT_PLS_CH7.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR Configures whether to clear LEDC_EVT_TIME_OVF_TIMER3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER0_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER1_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER2_CMP.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR Configures whether to clear LEDC_EVT_TIMER3_CMP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST_CLR Configures whether to clear TG0_EVT_CNT_CMP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST_CLR Configures whether to clear TG0_EVT_CNT_CMP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR Configures whether to clear TG1_EVT_CNT_CMP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR Configures whether to clear TG1_EVT_CNT_CMP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

Continued from the previous page...

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR Configures whether to clear SYSTIMER_EVT_CNT_CMP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_STOP.
0: Invalid. No effect
1: Clear
(WT)

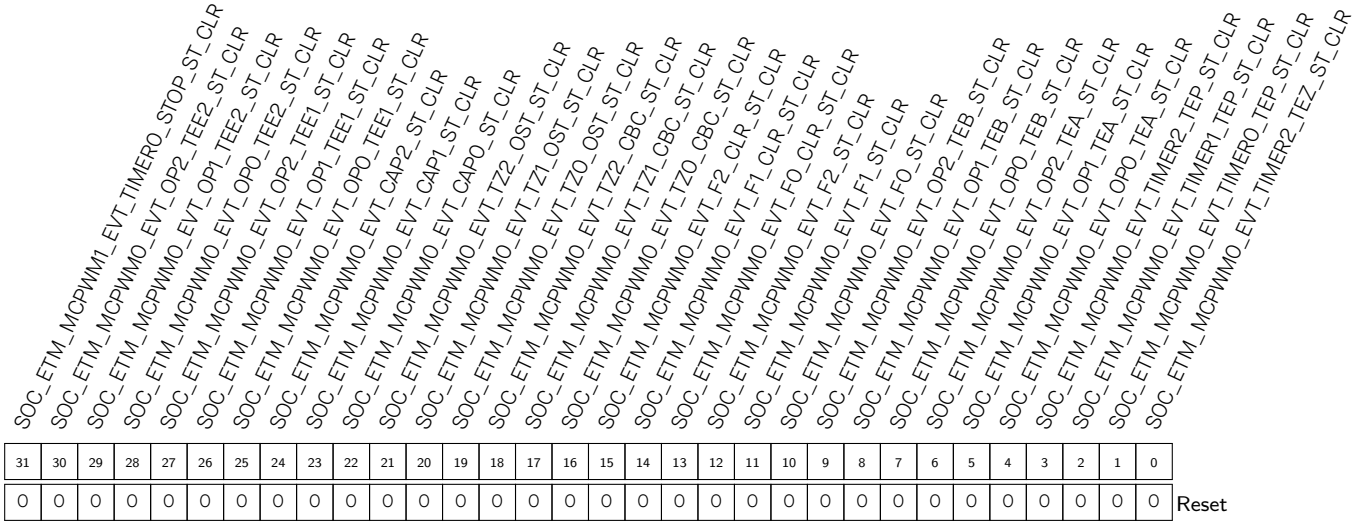
SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_TEZ.
0: Invalid. No effect
1: Clear
(WT)

Register 11.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)



SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER0_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER1_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST_CLR Configures whether to clear MCPWM0_EVT_TIMER2_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEA.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F0_ST_CLR Configures whether to clear MCPWM0_EVT_F0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F1_ST_CLR Configures whether to clear MCPWM0_EVT_F1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F2_ST_CLR Configures whether to clear MCPWM0_EVT_F2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F0_CLR.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_F1_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F1_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_F2_CLR_ST_CLR Configures whether to clear MCPWM0_EVT_F2_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ0_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ1_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST_CLR Configures whether to clear MCPWM0_EVT_TZ2_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST_CLR Configures whether to clear MCPWM0_EVT_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_CAP0_ST_CLR Configures whether to clear MCPWM0_EVT_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_CAP1_ST_CLR Configures whether to clear MCPWM0_EVT_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_CAP2_ST_CLR Configures whether to clear MCPWM0_EVT_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP0_TEE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP1_TEE2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

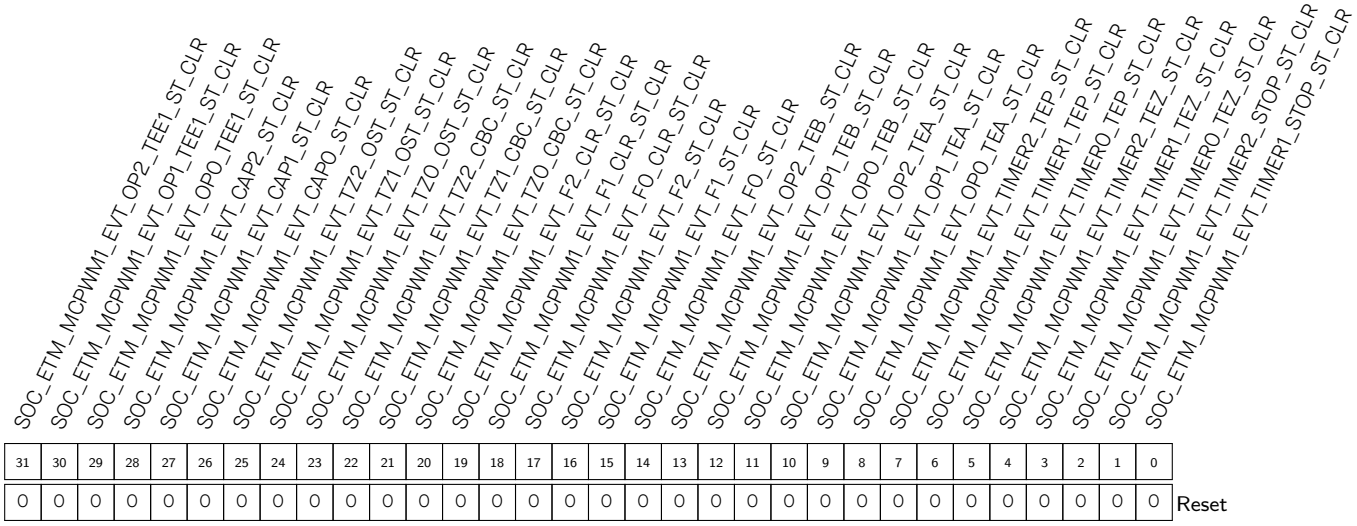
Register 11.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

Continued from the previous page...

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST_CLR Configures whether to clear MCPWM0_EVT_OP2_TEE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER0_STOP.
0: Invalid. No effect
1: Clear
(WT)

Register 11.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMERO_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMERO_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_TEZ.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMERO_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMERO_TEP.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER1_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST_CLR Configures whether to clear MCPWM1_EVT_TIMER2_TEP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEA.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEB.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEB.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F0_ST_CLR Configures whether to clear MCPWM1_EVT_F0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F1_ST_CLR Configures whether to clear MCPWM1_EVT_F1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F2_ST_CLR Configures whether to clear MCPWM1_EVT_F2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F0_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F1_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST_CLR Configures whether to clear MCPWM1_EVT_F2_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ0_CBC.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ1_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST_CLR Configures whether to clear MCPWM1_EVT_TZ2_CBC.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST_CLR Configures whether to clear MCPWM1_EVT_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_CAP0_ST_CLR Configures whether to clear MCPWM1_EVT_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_CAP1_ST_CLR Configures whether to clear MCPWM1_EVT_CAP1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

Continued from the previous page...

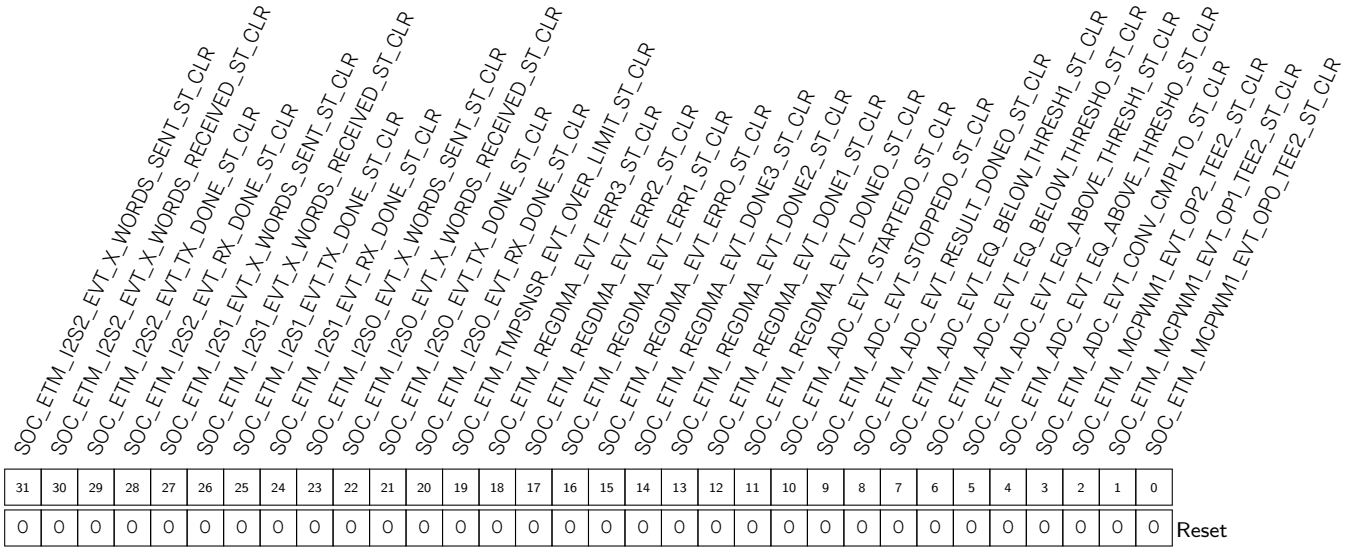
SOC_ETM_MCPWM1_EVT_CAP2_ST_CLR Configures whether to clear MCPWM1_EVT_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEE1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEE1.
0: Invalid. No effect
1: Clear
(WT)

Register 11.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP0_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP1_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST_CLR Configures whether to clear MCPWM1_EVT_OP2_TEE2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ADC_EVT_CONV_CMPLT0_ST_CLR Configures whether to clear ADC_EVT_CONV_CMPLT0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST_CLR Configures whether to clear ADC_EVT_EQ_ABOVE_THRESH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR Configures whether to clear ADC_EVT_EQ_ABOVE_THRESH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR Configures whether to clear ADC_EVT_EQ_BELOW_THRESH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR Configures whether to clear ADC_EVT_EQ_BELOW_THRESH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR Configures whether to clear ADC_EVT_RESULT_DONE0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_STOPPED0_ST_CLR Configures whether to clear ADC_EVT_STOPPED0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_EVT_STARTED0_ST_CLR Configures whether to clear ADC_EVT_STARTED0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE0_ST_CLR Configures whether to clear REGDMA_EVT_DONE0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE1_ST_CLR Configures whether to clear REGDMA_EVT_DONE1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_REGDMA_EVT_DONE2_ST_CLR Configures whether to clear REGDMA_EVT_DONE2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_DONE3_ST_CLR Configures whether to clear REGDMA_EVT_DONE3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR0_ST_CLR Configures whether to clear REGDMA_EVT_ERR0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR1_ST_CLR Configures whether to clear REGDMA_EVT_ERR1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR2_ST_CLR Configures whether to clear REGDMA_EVT_ERR2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_REGDMA_EVT_ERR3_ST_CLR Configures whether to clear REGDMA_EVT_ERR3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR Configures whether to clear TMPSNSR_EVT_OVER_LIMIT.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_I2S0_EVT_RX_DONE_ST_CLR Configures whether to clear I2S0_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_TX_DONE_ST_CLR Configures whether to clear I2S0_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S0_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S0_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_RX_DONE_ST_CLR Configures whether to clear I2S1_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_TX_DONE_ST_CLR Configures whether to clear I2S1_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S1_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

Continued from the previous page...

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S1_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

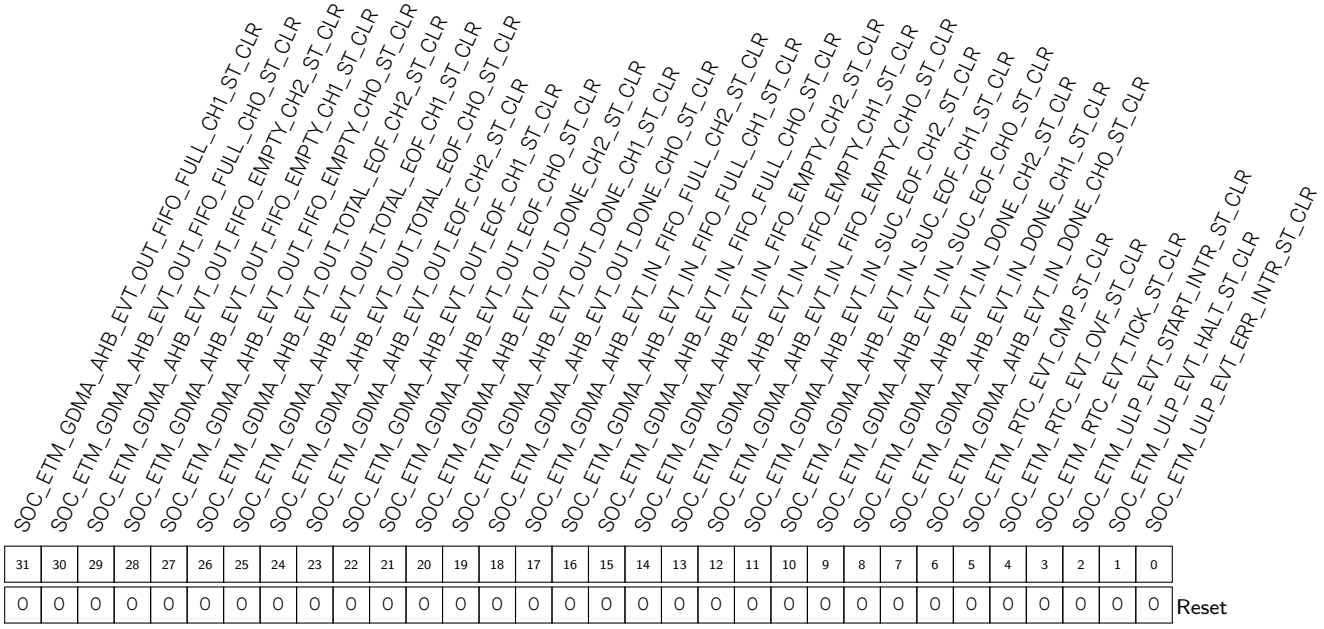
SOC_ETM_I2S2_EVT_RX_DONE_ST_CLR Configures whether to clear I2S2_EVT_RX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_TX_DONE_ST_CLR Configures whether to clear I2S2_EVT_TX_DONE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST_CLR Configures whether to clear I2S2_EVT_X_WORDS_RECEIVED.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST_CLR Configures whether to clear I2S2_EVT_X_WORDS_SENT.
0: Invalid. No effect
1: Clear
(WT)

Register 11.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_ULP_EVT_ERR_INTR_ST_CLR Configures whether to clear ULP_EVT_ERR_INTR.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ULP_EVT_HALT_ST_CLR Configures whether to clear ULP_EVT_HALT.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_ULP_EVT_START_INTR_ST_CLR Configures whether to clear ULP_EVT_START_INTR.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_TICK_ST_CLR Configures whether to clear RTC_EVT_TICK.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_OVF_ST_CLR Configures whether to clear RTC_EVT_OVF.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_RTC_EVT_CMP_ST_CLR Configures whether to clear RTC_EVT_CMP.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_SUC_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_IN_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

Continued from the previous page...

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

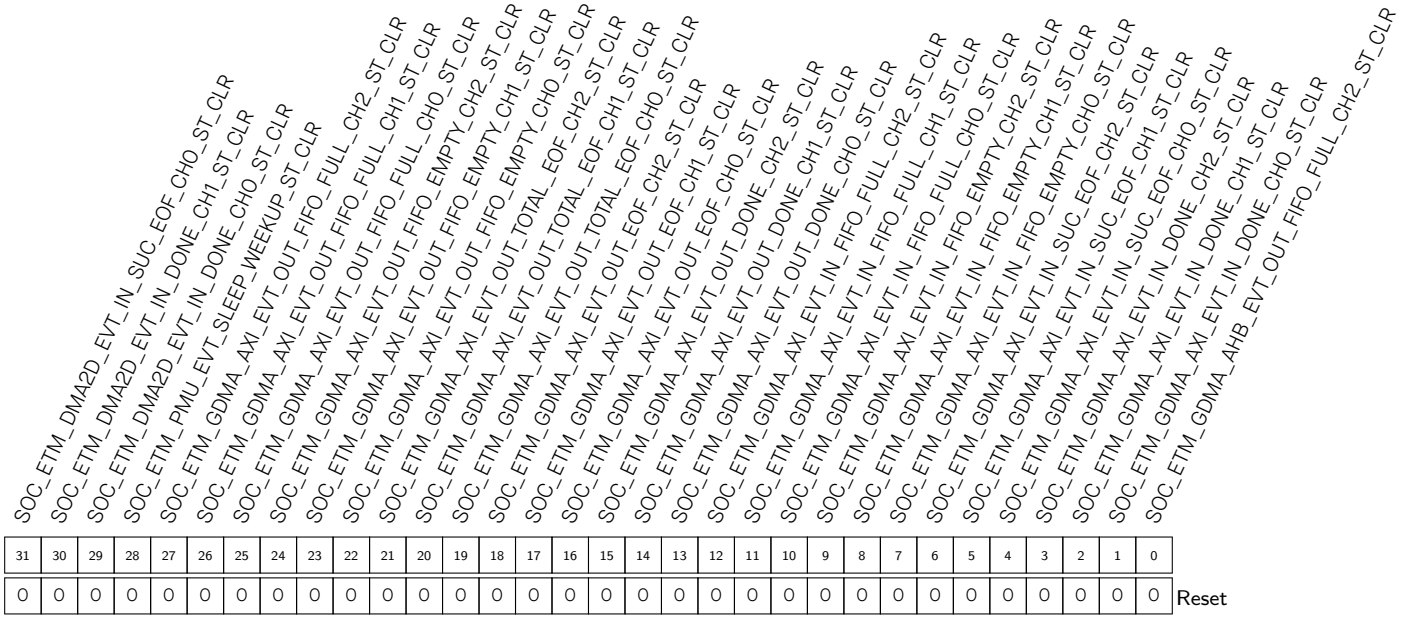
SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

Register 11.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)



SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AHB_EVT_OUT_FIFO_FULL_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_DONE_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_SUC_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_IN_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_DONE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH0.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

Continued from the previous page...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST_CLR Configures whether to clear GDMA_AXI_EVT_OUT_FIFO_FULL_CH2.
0: Invalid. No effect
1: Clear
(WT)

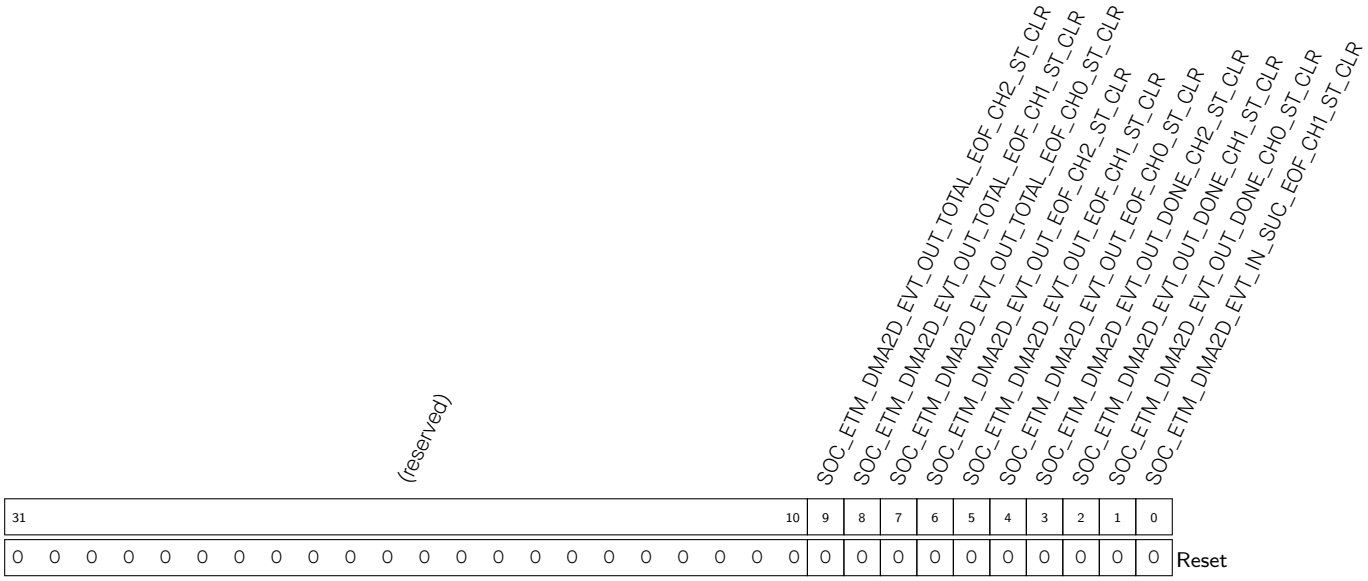
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR Configures whether to clear PMU_EVT_SLEEP_WEEKUP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST_CLR Configures whether to clear DMA2D_EVT_IN_DONE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST_CLR Configures whether to clear DMA2D_EVT_IN_DONE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_IN_SUC_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

Register 11.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_IN_SUC_EOF_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_DONE_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)

Continued from the previous page...

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

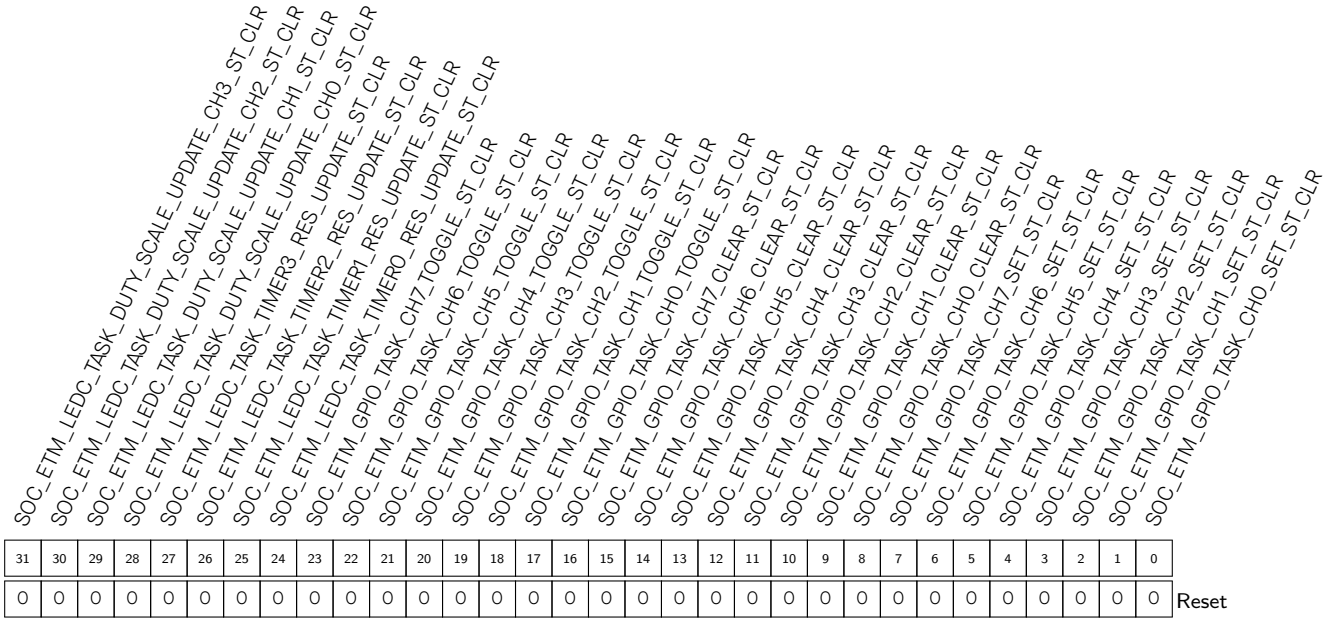
SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST_CLR Configures whether to clear DMA2D_EVT_OUT_TOTAL_EOF_CH2.
0: Invalid. No effect
1: Clear
(WT)

Register 11.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)



SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR Configures whether to clear GPIO_TASK_CH0_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR Configures whether to clear GPIO_TASK_CH1_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR Configures whether to clear GPIO_TASK_CH2_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR Configures whether to clear GPIO_TASK_CH3_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR Configures whether to clear GPIO_TASK_CH4_SET.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR Configures whether to clear GPIO_TASK_CH5_SET.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR Configures whether to clear GPIO_TASK_CH6_SET.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR Configures whether to clear GPIO_TASK_CH7_SET.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH0_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH1_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH2_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH3_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH4_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH5_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH6_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR Configures whether to clear GPIO_TASK_CH7_CLEAR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH0_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH1_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH2_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH3_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH4_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH5_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH6_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR Configures whether to clear GPIO_TASK_CH7_TOGGLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RES_UPDATE.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

Continued from the previous page...

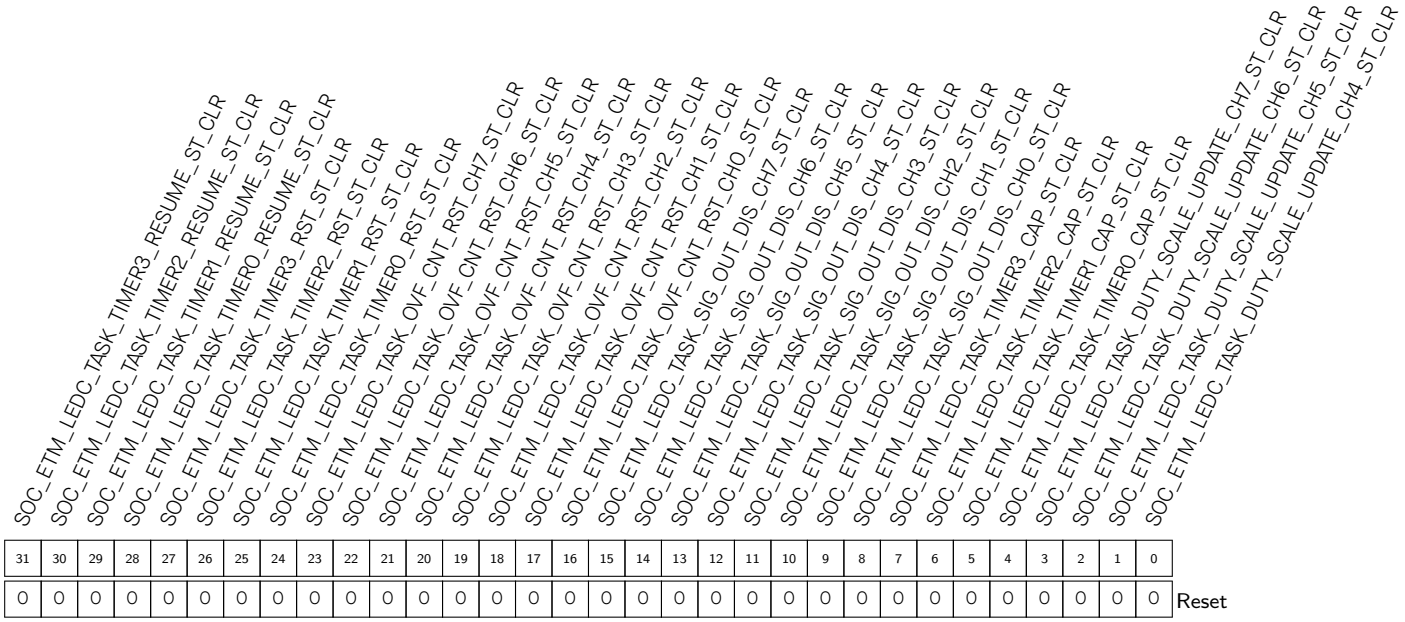
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH3.
0: Invalid. No effect
1: Clear
(WT)

Register 11.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH4.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH5.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH6.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST_CLR Configures whether to clear LEDC_TASK_DUTY_SCALE_UPDATE_CH7.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_CAP.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_CAP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH3.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST_CLR Configures whether to clear LEDC_TASK_SIG_OUT_DIS_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST_CLR Configures whether to clear LEDC_TASK_OVF_CNT_RST_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

Continued from the previous page...

SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RST.
0: Invalid. No effect
1: Clear
(WT)

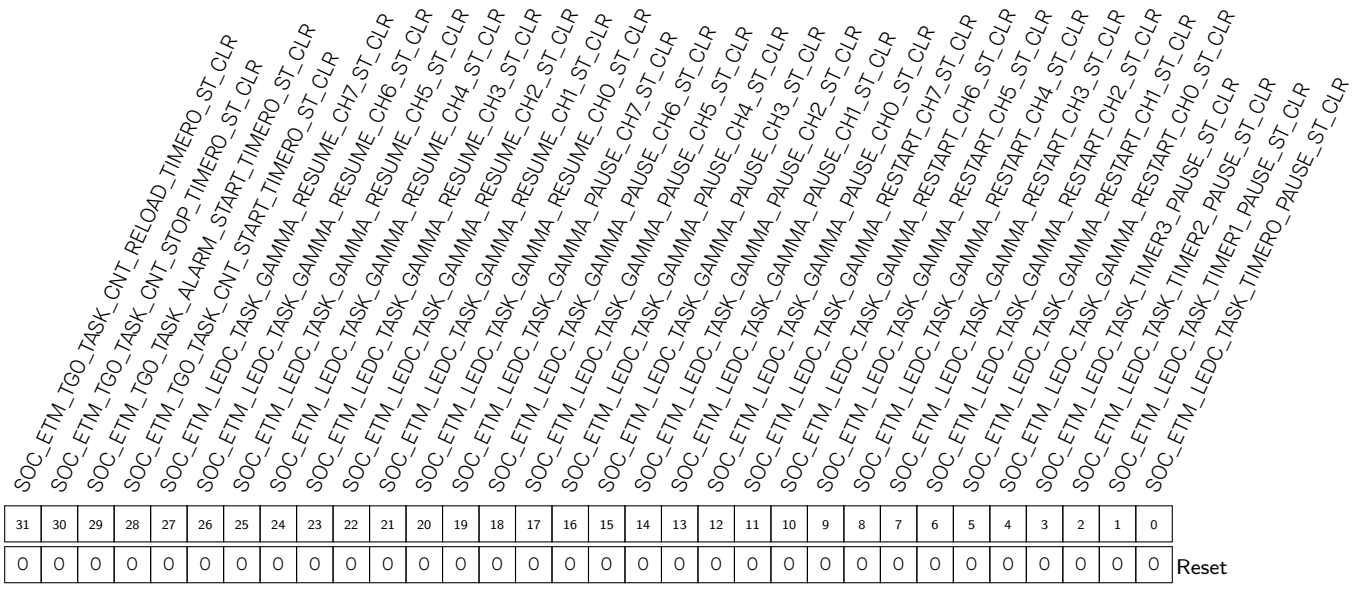
SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_RESUME.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_RESUME.
0: Invalid. No effect
1: Clear
(WT)

Register 11.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)



SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER0_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER1_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER2_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR Configures whether to clear LEDC_TASK_TIMER3_PAUSE.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH0.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESTART_CH7.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH5.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH6.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_PAUSE_CH7.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH3.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH4.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH5.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

Continued from the previous page...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH6.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST_CLR Configures whether to clear LEDC_TASK_GAMMA_RESUME_CH7.
0: Invalid. No effect
1: Clear
(WT)

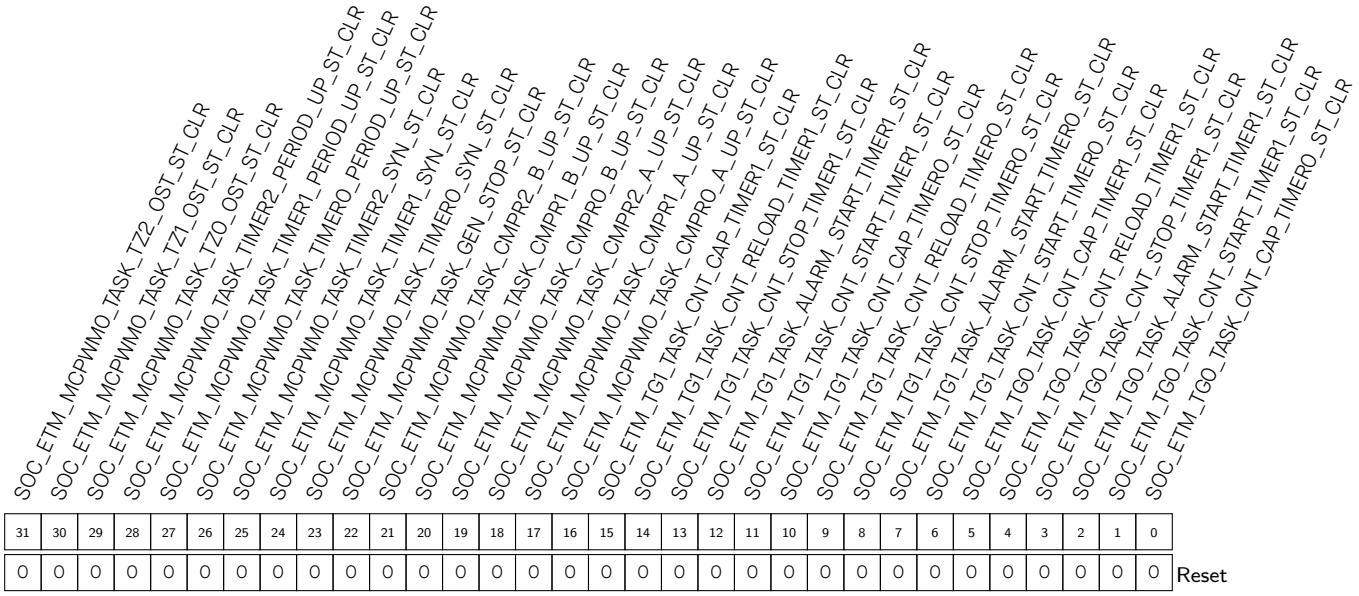
SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR Configures whether to clear TGO_TASK_ALARM_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_STOP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST_CLR Configures whether to clear TGO_TASK_CNT_RELOAD_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

Register 11.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST_CLR Configures whether to clear TG0_TASK_CNT_CAP_TIMER0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_START_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST_CLR Configures whether to clear TG0_TASK_ALARM_START_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_STOP_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_RELOAD_TIMER1.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST_CLR Configures whether to clear TG0_TASK_CNT_CAP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR Configures whether to clear TG1_TASK_ALARM_START_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_STOP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_RELOAD_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR Configures whether to clear TG1_TASK_CNT_CAP_TIMER0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_START_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR Configures whether to clear TG1_TASK_ALARM_START_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_STOP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_RELOAD_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR Configures whether to clear TG1_TASK_CNT_CAP_TIMER1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR0_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR1_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR2_A_UP.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR0_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR1_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST_CLR Configures whether to clear MCPWM0_TASK_CMPR2_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST_CLR Configures whether to clear MCPWM0_TASK_GEN_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER0_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER1_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER2_SYN.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

Continued from the previous page...

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER0_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER1_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

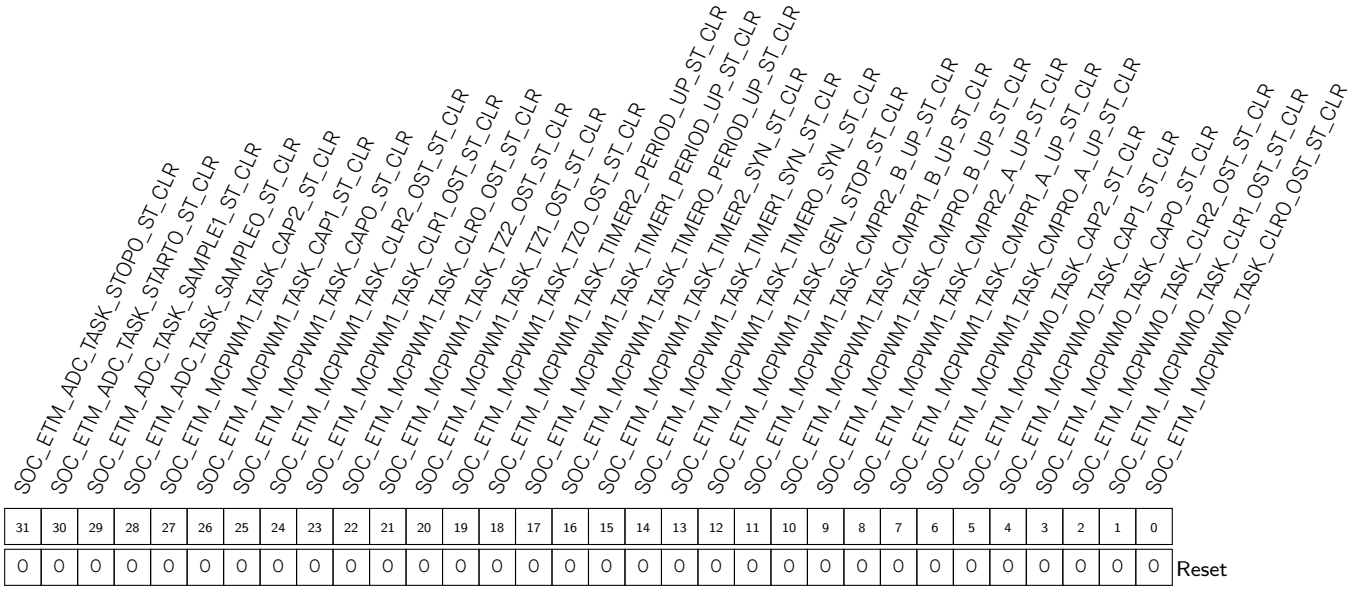
SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST_CLR Configures whether to clear MCPWM0_TASK_TIMER2_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST_CLR Configures whether to clear MCPWM0_TASK_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Register 11.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST_CLR Configures whether to clear MCPWM0_TASK_CLR2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP0_ST_CLR Configures whether to clear MCPWM0_TASK_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP1_ST_CLR Configures whether to clear MCPWM0_TASK_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM0_TASK_CAP2_ST_CLR Configures whether to clear MCPWM0_TASK_CAP2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR0_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR1_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR2_A_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR0_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR1_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST_CLR Configures whether to clear MCPWM1_TASK_CMPR2_B_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST_CLR Configures whether to clear MCPWM1_TASK_GEN_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER0_SYN.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER1_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER2_SYN.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER0_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER1_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST_CLR Configures whether to clear MCPWM1_TASK_TIMER2_PERIOD_UP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST_CLR Configures whether to clear MCPWM1_TASK_TZ2_OST.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR0_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR1_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST_CLR Configures whether to clear MCPWM1_TASK_CLR2_OST.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP0_ST_CLR Configures whether to clear MCPWM1_TASK_CAP0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP1_ST_CLR Configures whether to clear MCPWM1_TASK_CAP1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_MCPWM1_TASK_CAP2_ST_CLR Configures whether to clear MCPWM1_TASK_CAP2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR Configures whether to clear ADC_TASK_SAMPLE0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ADC_TASK_SAMPLE1_ST_CLR Configures whether to clear ADC_TASK_SAMPLE1.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

Continued from the previous page...

SOC_ETM_ADC_TASK_START0_ST_CLR Configures whether to clear ADC_TASK_START0.

0: Invalid. No effect

1: Clear

(WT)

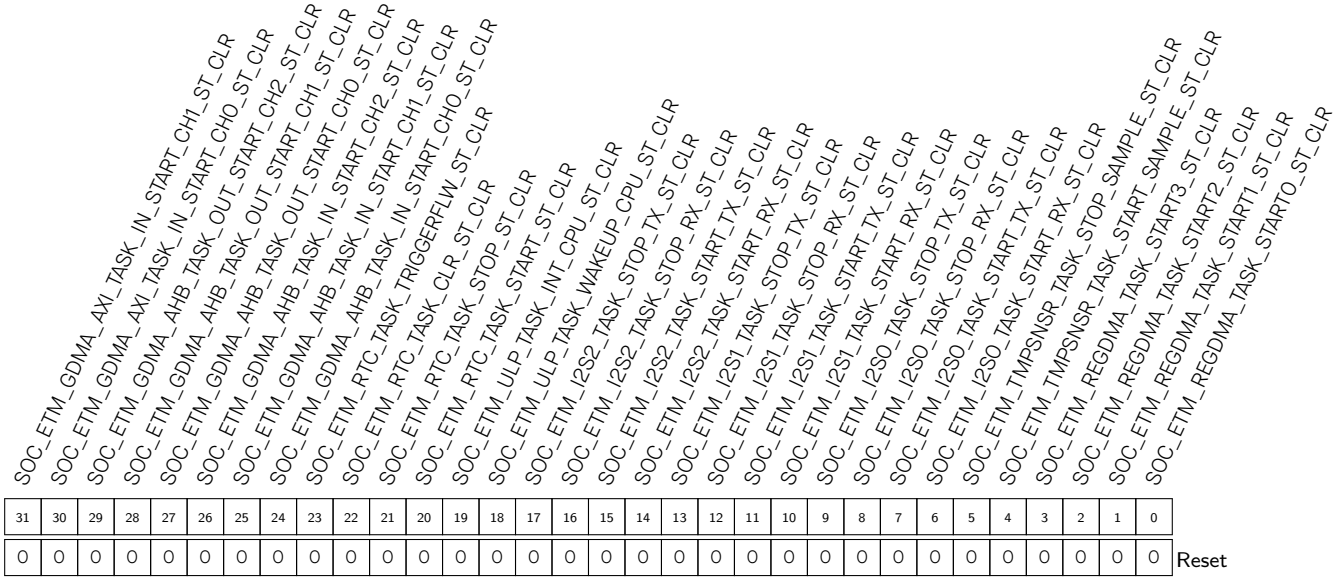
SOC_ETM_ADC_TASK_STOP0_ST_CLR Configures whether to clear ADC_TASK_STOP0.

0: Invalid. No effect

1: Clear

(WT)

Register 11.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)



SOC_ETM_REGDMA_TASK_START0_ST_CLR Configures whether to clear REGDMA_TASK_START0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START1_ST_CLR Configures whether to clear REGDMA_TASK_START1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START2_ST_CLR Configures whether to clear REGDMA_TASK_START2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_REGDMA_TASK_START3_ST_CLR Configures whether to clear REGDMA_TASK_START3.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR Configures whether to clear TMPSNSR_TASK_START_SAMPLE.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR Configures whether to clear TMPSNSR_TASK_STOP_SAMPLE.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_START_RX_ST_CLR Configures whether to clear I2S0_TASK_START_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_START_TX_ST_CLR Configures whether to clear I2S0_TASK_START_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_STOP_RX_ST_CLR Configures whether to clear I2S0_TASK_STOP_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S0_TASK_STOP_TX_ST_CLR Configures whether to clear I2S0_TASK_STOP_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_TASK_START_RX_ST_CLR Configures whether to clear I2S1_TASK_START_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_TASK_START_TX_ST_CLR Configures whether to clear I2S1_TASK_START_TX.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_I2S1_TASK_STOP_RX_ST_CLR Configures whether to clear I2S1_TASK_STOP_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S1_TASK_STOP_TX_ST_CLR Configures whether to clear I2S1_TASK_STOP_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_TASK_START_RX_ST_CLR Configures whether to clear I2S2_TASK_START_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_TASK_START_TX_ST_CLR Configures whether to clear I2S2_TASK_START_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_TASK_STOP_RX_ST_CLR Configures whether to clear I2S2_TASK_STOP_RX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_I2S2_TASK_STOP_TX_ST_CLR Configures whether to clear I2S2_TASK_STOP_TX.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST_CLR Configures whether to clear ULP_TASK_WAKEUP_CPU.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_ULP_TASK_INT_CPU_ST_CLR Configures whether to clear ULP_TASK_INT_CPU.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_START_ST_CLR Configures whether to clear RTC_TASK_START.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_STOP_ST_CLR Configures whether to clear RTC_TASK_STOP.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_CLR_ST_CLR Configures whether to clear RTC_TASK_CLR.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR Configures whether to clear RTC_TASK_TRIGGERFLW.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST_CLR Configures whether to clear GDMA_AHB_TASK_IN_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

Continued from the previous page...

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

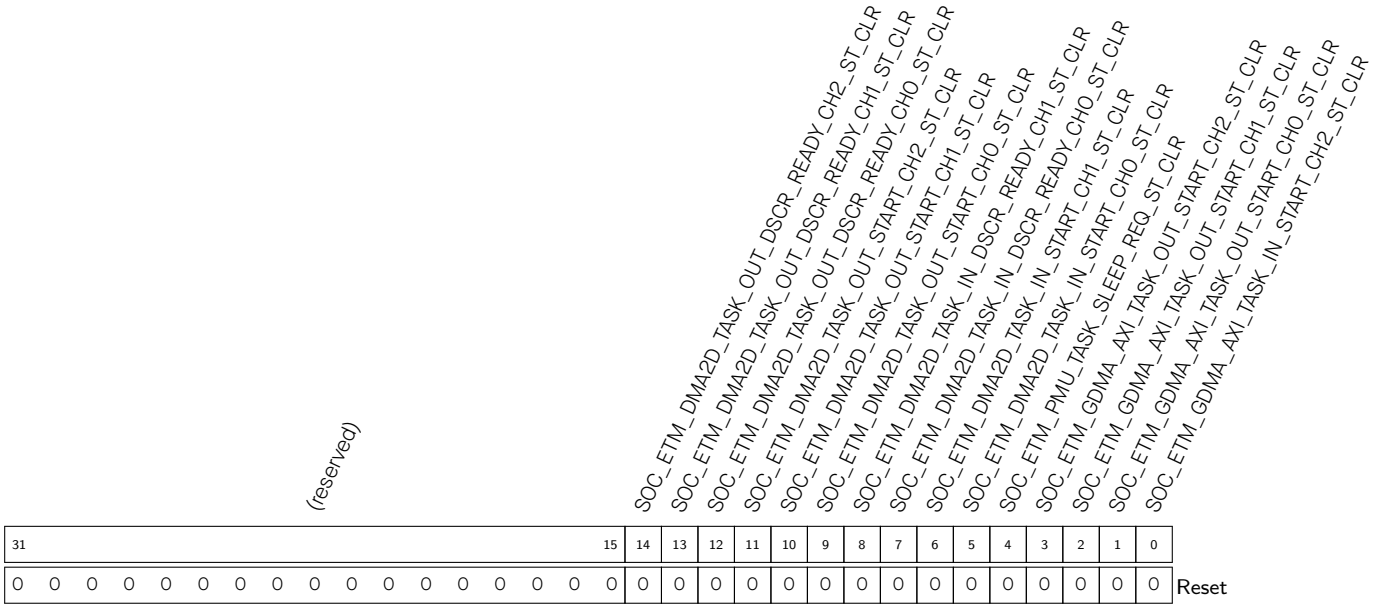
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST_CLR Configures whether to clear GDMA_AHB_TASK_OUT_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

Register 11.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST_CLR Configures whether to clear GDMA_AXI_TASK_IN_START_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH1.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST_CLR Configures whether to clear GDMA_AXI_TASK_OUT_START_CH2.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR Configures whether to clear PMU_TASK_SLEEP_REQ.

0: Invalid. No effect

1: Clear

(WT)

Continued on the next page...

Register 11.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST_CLR Configures whether to clear DMA2D_TASK_IN_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST_CLR Configures whether to clear DMA2D_TASK_IN_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST_CLR Configures whether to clear DMA2D_TASK_IN_DSCR_READY_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST_CLR Configures whether to clear DMA2D_TASK_IN_DSCR_READY_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH0.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH1.
0: Invalid. No effect
1: Clear
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST_CLR Configures whether to clear DMA2D_TASK_OUT_START_CH2.
0: Invalid. No effect
1: Clear
(WT)

Continued on the next page...

Register 11.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

Continued from the previous page...

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH0.

0: Invalid. No effect

1: Clear

(WT)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH1.

0: Invalid. No effect

1: Clear

(WT)

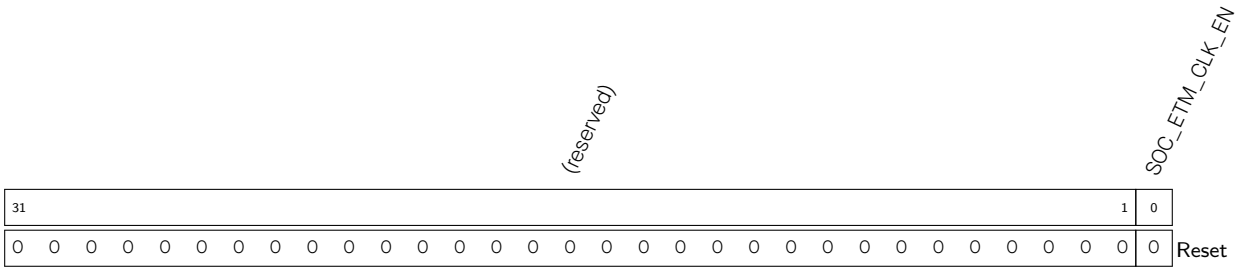
SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST_CLR Configures whether to clear DMA2D_TASK_OUT_DSCR_READY_CH2.

0: Invalid. No effect

1: Clear

(WT)

Register 11.39. SOC_ETM_CLK_EN_REG (0x0220)



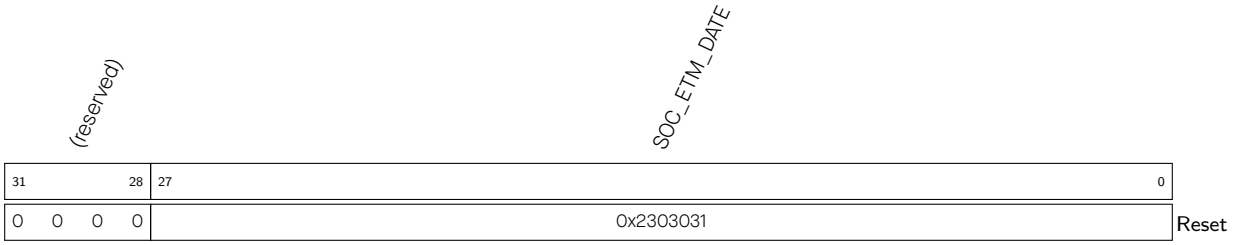
SOC_ETM_CLK_EN Configures resister clock gating.

0: Support clock only when application writes registers

1: Force on clock gating for registers

(R/W)

Register 11.40. SOC_ETM_DATE_REG (0x0224)



SOC_ETM_DATE Version control register. (R/W)

Chapter 12

System Timer

12.1 Overview

ESP32-P4 provides a 52-bit system timer, which can be used to generate tick interrupts for the operating system, or be used as a general timer to generate periodic interrupts or one-time interrupts.

12.2 Features

The system timer has the following features:

- Two 52-bit counters and three 52-bit comparators
- Software accessing registers clocked by APB_CLK
- CNT_CLK used for counting, with an average frequency of 16 MHz in two counting cycles
- 40 MHz XTAL_CLK as the clock source of CNT_CLK
- 52-bit alarm values (t) and 26-bit alarm periods (δt)
- Two modes to generate alarms:
  - Target mode: only a one-time alarm is generated based on the alarm value (t)
  - Period mode: periodic alarms are generated based on the alarm period (δt)
- Three comparators generating three independent interrupts based on configured alarm value (t) or alarm period (δt)
- Software configuring the reference count value. For example, the system timer is able to load back the sleep time recorded by RTC timer via software after Light-sleep
- Able to stall or continue running when CPU stalls or enters the on-chip-debugging mode
- Alarm for Event Task Matrix (ETM) event

12.3 System Timer Structure

The timer consists of two counters: UNIT0 and UNIT1. The count values can be monitored by three comparators, COMP0, COMP1, and COMP2. See the timer block diagram in Figure 12.3-1.

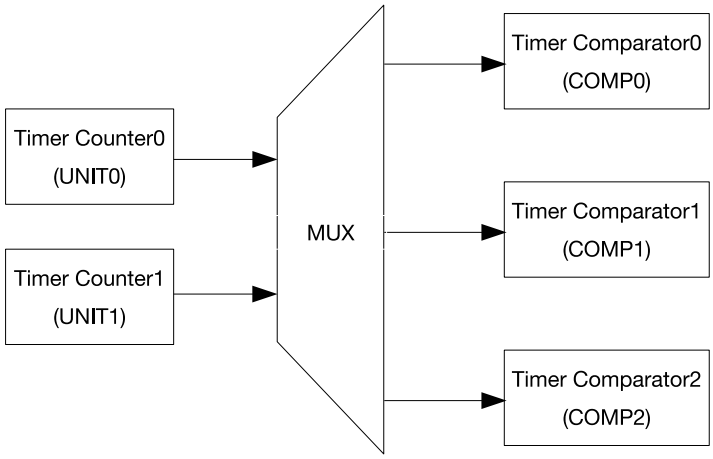


Figure 12.3-1. System Timer Structure

12.4 Clock Source Selection

The counters and comparators use XTAL_CLK or RC_FAST_CLK as the clock sources. The clock source can be selected by configuring field HP_SYS_CLKRST_REG_SYSTIMER_CLK_SRC_SEL in register HP_SYS_CLKRST_PERI_CLK_CTRL21_REG. After XTAL_CLK is scaled by 2.5, a counter clock signal CNT_CLK clock is generated with a frequency of $f_{XTAL\_CLK}/2.5$. The average clock frequency of CNT_CLK is 16 MHz, as shown in Figure 12.5-1. The timer counter is incremented by 1/16 μs on each CNT_CLK cycle.

Software operation such as configuring registers is clocked by APB_CLK. For more information about APB_CLK, see Chapter 8 *Reset and Clock*.

The following two bits of system registers are also used to control the system timer:

- Set HP_SYS_CLKRST_REG_SYSTIMER_APB_CLK_EN in register HP_SYS_CLKRST_SOC_CLK_CTRL2_REG to enable APB_CLK signal to the system timer.
- Set HP_SYS_CLKRST_REG_RST_EN_SYSTIMER in register HP_SYS_CLKRST_HP_RST_EN1_REG to reset the system timer.

Note that if the timer is reset, its registers will be restored to their default values. For more information, please refer to Chapter 8 *Reset and Clock*.

12.5 Functional Description

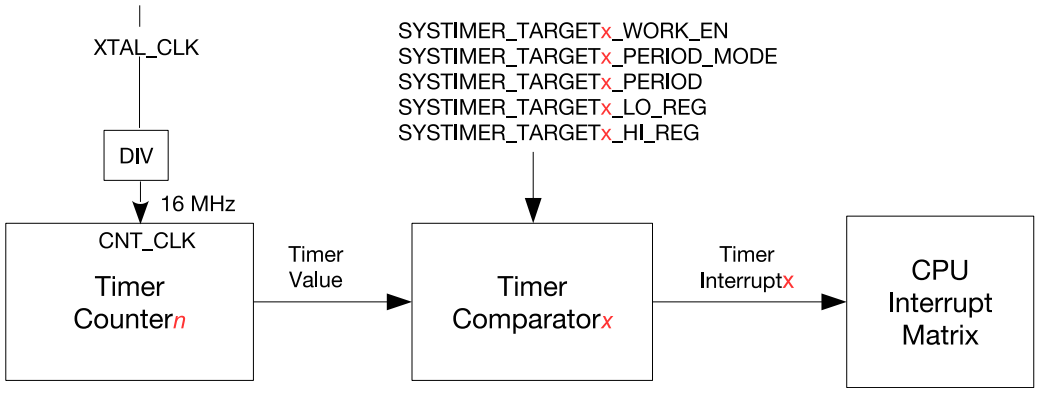


Figure 12.5-1. System Timer Alarm Generation

Figure 12.5-1 shows the procedure to generate alarm in system timer. In this process, one timer counter and one timer comparator are used. An alarm interrupt will be generated accordingly based on the comparison result in the comparator.

12.5.1 Counter

The system timer has two 52-bit timer counters, shown as UNIT*n* (*n* = 0 or 1). Their counting clock source is a 16 MHz clock, i.e., CNT_CLK. Whether UNIT*n* works or not is controlled by three bits in register SYSTIMER_CONF_REG:

- SYSTIMER_TIMER_UNIT*n*_WORK_EN: set this bit to enable the counter UNIT*n* in the system timer.
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU0 is stalled. The counter continues its counting after the CPU resumes.
- SYSTIMER_TIMER_UNIT*n*_CORE1_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU1 is stalled. The counter continues its counting after the CPU resumes.

The configuration of the bits to control the counter UNIT*n* is shown below, assuming that CPU is stalled.

Table 12.5-1. UNIT*n* Configuration Bits

SYSTIMER_TIMER_ UNIT*n*_WORK_EN	SYSTIMER_TIMER_ UNIT*n*_CORE0_STALL_EN	SYSTIMER_TIMER_ UNIT*n*_CORE1_STALL_EN	Counter UNIT*n*
0	x *	x *	Not at work
1	x	1	Stop counting, but will continue its counting after the CPU0 resumes
1	x	1	Stop counting, but will continue its counting after the CPU1 resumes
1	0	0	Keep counting

* x: Don't-care.

When the counter UNIT*n* is at work, the count value is incremented on each counting cycle. When the counter UNIT*n* is stopped, the count value stops increasing and keeps unchanged.

The lower 32 and higher 20 bits of the initial count value are loaded from the registers SYSTIMER_TIMER_UNIT*n*_LOAD_LO and SYSTIMER_TIMER_UNIT*n*_LOAD_HI. Writing 1 to the bit SYSTIMER_TIMER_UNIT*n*_LOAD will trigger a reload event, and the current count value will be changed immediately. If UNIT*n* is at work, the counter will continue to count up from the new reloaded value.

Writing 1 to SYSTIMER_TIMER_UNIT*n*_UPDATE will trigger an update event. The lower 32 and higher 20 bits of the current count value will be locked into the registers SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI, and then SYSTIMER_TIMER_UNIT*n*_VALUE_VALID is asserted. Before the next update event, the values of SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI remain unchanged.

12.5.2 Comparator and Alarm

The system timer has three 52-bit comparators, shown as COMP*x* (*x* = 0, 1, or 2). The comparators can generate independent interrupts based on different alarm values (t) or alarm periods (δt).

Configure SYSTIMER_TARGET*x*_PERIOD_MODE to choose from the two alarm modes for each COMP*x*:

- 1: period mode
- 0: target mode

In period mode, the alarm period (δt) is provided by the register SYSTIMER_TARGET*x*_PERIOD. Assuming that current count value is t1, when it reaches (t1 + δt), an alarm interrupt will be generated. When the count value reaches (t1 + 2*δt), another alarm interrupt also will be generated. By such way, periodic alarms are generated.

In target mode, the lower 32 bits and higher 20 bits of the alarm value (t) are provided by SYSTIMER_TIMER_TARGET*x*_LO and SYSTIMER_TIMER_TARGET*x*_HI. Assuming that current count value is t2 (t2 <= t), an alarm interrupt will be generated when the count value reaches the alarm value (t). Unlike in period mode, only one alarm interrupt is generated in target mode.

SYSTIMER_TARGET*x*_TIMER_UNIT_SEL is used to choose the count value from which timer counter to be compared to generate alarms:

- 1: Use the count value from UNIT*1*
- 0: Use the count value from UNIT*0*

Finally, set SYSTIMER_TARGET*x*_WORK_EN and COMP*x* starts to compare the count value:

- In target mode, COMP*x* compares it with the alarm value (t).
- In period mode, COMP*x* compares it with the alarm period (t1 + n*δt).

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value + n*alarm period δt (n = 1, 2, 3...) in period mode. But if the alarm value (t) set in registers is less than the current count value, i.e., the target has already passed, when the current count value is larger than the alarm value (t) within a range (0 ~ 2^{51} - 1), an alarm interrupt will also be generated immediately. No matter in target mode or period mode, the low 32 bits and high 20 bits of the real alarm value can always be read from SYSTIMER_TARGET*x*_LO_RO and SYSTIMER_TARGET*x*_HI_RO. The alarm trigger point and the relationship between current count value t_c and the alarm value t_t are shown below.

Table 12.5-2. Trigger Point

Relationship Between t_c and t_t	Trigger Point
t_c- t_t <= 0	t_c = t_t, an alarm is triggered.
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} and t_t < 2^{51}, or t_c >= 2^{51} and t_t >= 2^{51})	An alarm is triggered immediately.
t_c - t_t >= 2^{51} - 1	t_c overflows after counting to its maximum value 52'hfffffffffffff, and then starts counting up from 0. When its value reaches t_t, an alarm is triggered.

12.5.3 Event Task Matrix

The system timer on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows the system timer's ETM tasks to be triggered by any peripherals' ETM events, or system timer's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the system timer. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The system timer can generate the following ETM event:

- SYSTIMER_EVT_CNT_CMP*x*: Indicates the alarm events generated by COMP*x*.

When SYSTIMER_ETM_EN is set to 1, the alarm events can trigger the ETM event.

12.5.4 Synchronization Operation

The frequency of the clock used for software configuration is different from that of the clock driving the timer counters and comparators. Therefore, synchronization is required for configuring some registers. The steps are as follows:

1. Software writes specific values to configuration fields. See the first column in Table 12.5-3.
2. Software writes 1 to corresponding bits to start synchronization. See the second column in Table 12.5-3.

Table 12.5-3. Synchronization Operation for Configuration Registers

Configuration Fields	Synchronization Enable Bit
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

The frequency of the clock used for software reading is different from that of the clock driving the timer counters and comparators. Therefore, synchronization is also needed for reading some registers. The steps are as follows:

1. Software writes specific values to the updating register SYSTIMER_TIMER_UNIT*n*_UPDATE.
2. Software reads the corresponding bit SYSTIMER_TIMER_UNIT*n*_VALUE_VALID to be valid to confirm synchronization is done.

3. Software reads the corresponding status registers SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.

12.6 Interrupts

ESP32-P4's system timer can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- SYSTIMER_TARGET0_INT
- SYSTIMER_TARGET1_INT
- SYSTIMER_TARGET2_INT

There are several internal interrupt sources from the system timer that can generate the above interrupt signals. The interrupt sources from the system timer are listed with their trigger conditions and the resulted interrupt signals in Table 12.6-1.

Table 12.6-1. System Timer's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
SYSTIMER_INT0	When COMP0 alarms	SYSTIMER_TARGET0_INT
SYSTIMER_INT1	When COMP1 alarms	SYSTIMER_TARGET1_INT
SYSTIMER_INT2	When COMP2 alarms	SYSTIMER_TARGET2_INT

The above interrupts are level-type alarm interrupts. The interrupt signal is asserted high when the comparator starts to alarm. Until the software clears the interrupt, it remains high. To enable interrupts, set the bit SYSTIMER_TARGET*x*_INT_ENA.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 12.8 *Register Summary*.

12.7 Programming Procedure

When configuring COMP*x* and UNIT*n*, please ensure the corresponding COMP and UNIT are at work.

12.7.1 Read Current Count Value

1. Set SYSTIMER_TIMER_UNIT*n*_UPDATE to fill the current count value of COMP*x* into SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.
2. Poll the reading of SYSTIMER_TIMER_UNIT*n*_VALUE_VALID till it is 1. Then, user can read the count value from SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.

3. Read the lower 32 bits and higher 20 bits from SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI respectively.

12.7.2 Configure a One-Time Alarm in Target Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for comparison with COMP*x*.
2. Read the current count value with reference to Section 12.7.1. This value will be used to calculate the alarm value (t) in Step 4.
3. Clear SYSTIMER_TARGET*x*_PERIOD_MODE to enable target mode.
4. Set an alarm value (t), and fill its lower 32 bits into SYSTIMER_TIMER_TARGET*x*_LO, and the higher 20 bits into SYSTIMER_TIMER_TARGET*x*_HI.
5. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm value (t) to COMP*x*, i.e., load the alarm value (t) to the COMP*x*.
6. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the alarm value (t).
7. Set SYSTIMER_TARGET*x*_INT_ENA to enable the timer interrupt. When UNIT*n* reaches the alarm value (t), a SYSTIMER_TARGET*x*_INT interrupt is triggered.

12.7.3 Configure Periodic Alarms in Period Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for comparison with COMP*x*.
2. Set an alarm period (δt), and fill it into SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm period (δt) to COMP*x*, i.e., load the alarm period (δt) to COMP*x*.
4. Clear and then set SYSTIMER_TARGET*x*_PERIOD_MODE to configure COMP*x* into period mode.
5. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the sum of (start value + n*δt) (n = 1, 2, 3...).
6. Set SYSTIMER_TARGET*x*_INT_ENA to enable the timer interrupt. A SYSTIMER_TARGET*x*_INT interrupt is triggered when UNIT*n* reaches start value + n*δt (n = 1, 2, 3...) set in Step 2.

12.7.4 Update After Light-sleep

1. Configure RTC timer before the chip goes to Light-sleep mode, to record the exact sleep time. For more information, see Chapter 5 *Low-Power Management [to be added later]*.
2. Read the sleep time from the RTC timer when the chip wakes up from Light-sleep mode.
3. Read the current count value of system timer, see Section 12.7.1.
4. Convert the time value recorded by RTC timer from the clock cycles based on RTC_SLOW_CLK to that based on 16 MHz CNT_CLK. For example, if the frequency of RTC_SLOW_CLK is 32 kHz, the recorded RTC timer value should be converted by multiplying by 500.

5. Add the converted RTC value to the current count value of system timer:
    - Fill the new value into SYSTIMER_TIMER_UNIT*n*_LOAD_LO (low 32 bits) and SYSTIMER_TIMER_UNIT*n*_LOAD_HI (high 20 bits).
    - Set SYSTIMER_TIMER_UNIT*n*_LOAD to load the new timer value into the system timer. By such way, the system timer is updated.

12.8 Register Summary

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

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

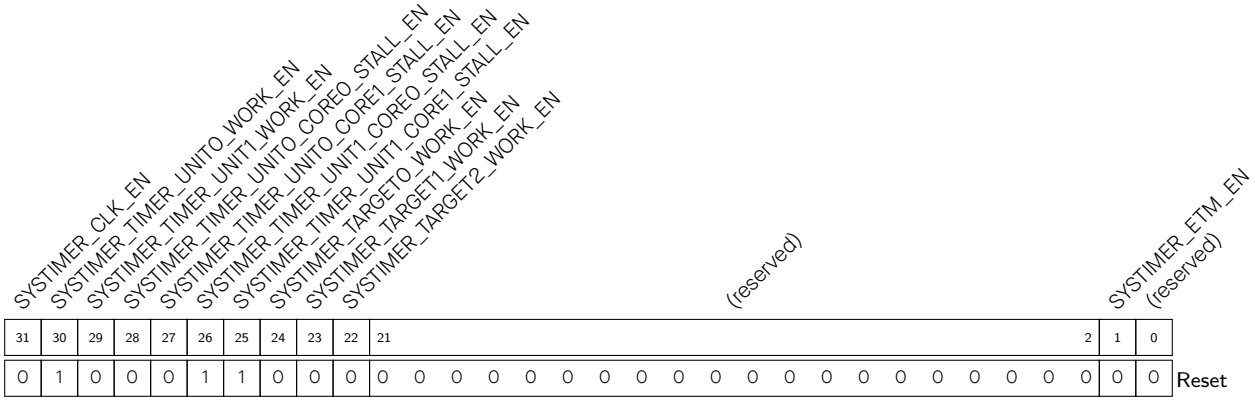
Name	Description	Address	Access
Clock Control Register			
SYSTIMER_CONF_REG	Configure system timer clock	0x0000	R/W
UNIT*0* Control and Configuration Registers			
SYSTIMER_UNIT0_OP_REG	Read UNIT0 value to registers	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	High 20 bits to be loaded to UNIT0	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	Low 32 bits to be loaded to UNIT0	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 value, high 20 bits	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 value, low 32 bits	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	UNIT0 synchronization register	0x005C	WT
UNIT*1* Control and Configuration Registers			
SYSTIMER_UNIT1_OP_REG	Read UNIT1 value to registers	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	High 20 bits to be loaded to UNIT1	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	Low 32 bits to be loaded to UNIT1	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	UNIT1 value, high 20 bits	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	UNIT1 value, low 32 bits	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	UNIT1 synchronization register	0x0060	WT
Comparator*0* Control and Configuration Registers			
SYSTIMER_TARGET0_HI_REG	Alarm value to be loaded to COMP0, high 20 bits	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	Alarm value to be loaded to COMP0, low 32 bits	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	Configure COMP0 alarm mode	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	COMP0 synchronization register	0x0050	WT
Comparator*1* Control and Configuration Registers			
SYSTIMER_TARGET1_HI_REG	Alarm value to be loaded to COMP1, high 20 bits	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	Alarm value to be loaded to COMP1, low 32 bits	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	Configure COMP1 alarm mode	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	COMP1 synchronization register	0x0054	WT
Comparator*2* Control and Configuration Registers			
SYSTIMER_TARGET2_HI_REG	Alarm value to be loaded to COMP2, high 20 bits	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	Alarm value to be loaded to COMP2, low 32 bits	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	Configure COMP2 alarm mode	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	COMP2 synchronization register	0x0058	WT
Interrupt Registers			
SYSTIMER_INT_ENA_REG	Interrupt enable register of system timer	0x0064	R/W
SYSTIMER_INT_RAW_REG	Interrupt raw register of system timer	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	Interrupt clear register of system timer	0x006C	WT

Name	Description	Address	Access
SYSTIMER_INT_ST_REG	Interrupt status register of system timer	0x0070	RO
COMP*0* Status Registers			
SYSTIMER_REAL_TARGET0_LO_REG	Actual target value of COMP0, low 32 bits	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	Actual target value of COMP0, high 20 bits	0x0078	RO
COMP*1* Status Registers			
SYSTIMER_REAL_TARGET1_LO_REG	Actual target value of COMP1, low 32 bits	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	Actual target value of COMP1, high 20 bits	0x0080	RO
COMP*2* Status Registers			
SYSTIMER_REAL_TARGET2_LO_REG	Actual target value of COMP2, low 32 bits	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	Actual target value of COMP2, high 20 bits	0x0088	RO
Version Register			
SYSTIMER_DATE_REG	Version control register	0x00FC	R/W

12.9 Registers

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

Register 12.1. SYSTIMER_CONF_REG (0x0000)



SYSTIMER_ETM_EN Configures whether to enable generation of ETM events.

0: Disable

1: Enable

(R/W)

SYSTIMER_TARGET2_WORK_EN Configures whether to enable COMP2.

0: Disable

1: Enable

(R/W)

SYSTIMER_TARGET1_WORK_EN Configures whether to enable COMP1. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

SYSTIMER_TARGET0_WORK_EN Configures whether to enable COMP0. See details in SYSTIMER_TARGET2_WORK_EN. (R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN Configures whether UNIT1 is stalled when CORE1 is stalled.

0: UNIT1 is not stalled.

1: UNIT1 is stalled.

(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN Configures whether UNIT1 is stalled when CORE0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN Configures whether UNIT0 is stalled when CORE1 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN Configures whether UNIT0 is stalled when CORE0 is stalled. See details in SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN. (R/W)

Continued on the next page...

Register 12.1. SYSTIMER_CONF_REG (0x0000)

Continued from the previous page...

SYSTIMER_TIMER_UNIT1_WORK_EN Configures whether to enable UNIT1.

0: Disable

1: Enable

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN Configures whether to enable UNIT0.

0: Disable

1: Enable

(R/W)

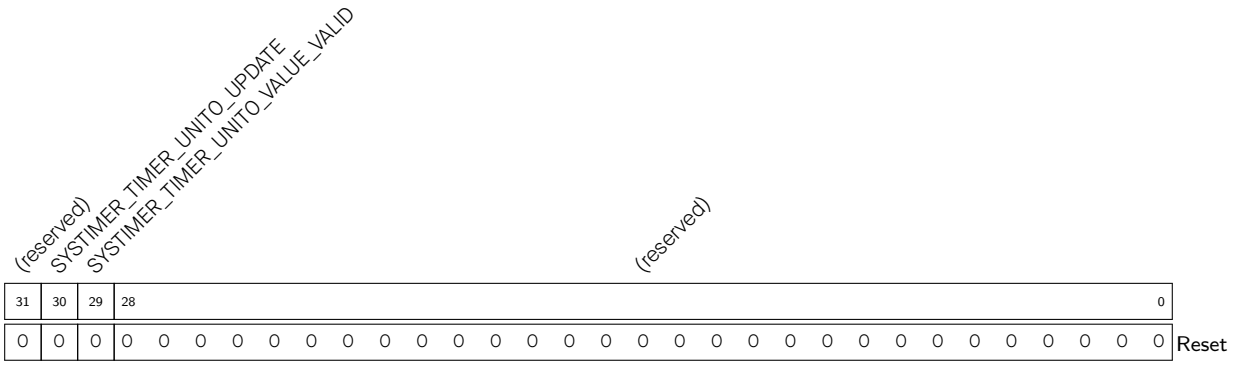
SYSTIMER_CLK_EN Configures register clock gating.

0: Only enable needed clock for register read or write operations.

1: Register clock is always enabled for read and write operations.

(R/W)

Register 12.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID Represents whether UNIT0 value is synchronized and valid.

0: UNIT0 value is neither synchronized nor valid

1: UNIT0 value is synchronized and valid

(R/SS/WTC)

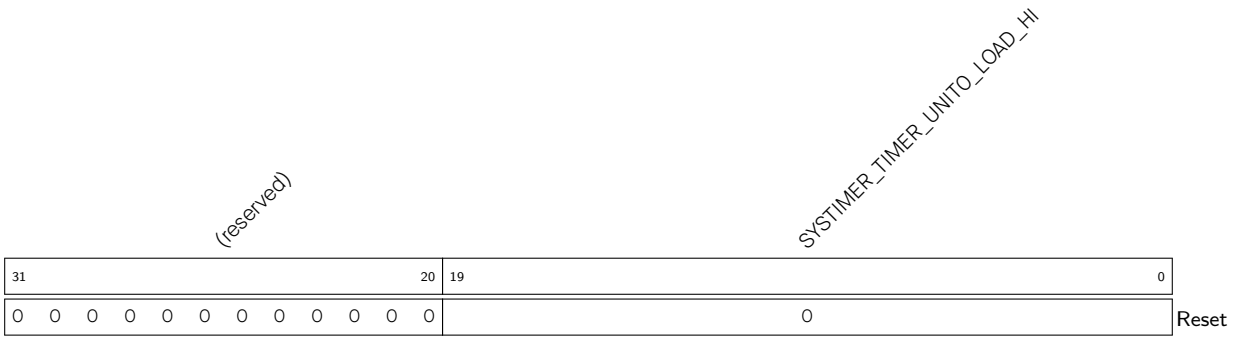
SYSTIMER_TIMER_UNIT0_UPDATE Configures whether to update timer UNIT0, i.e., reads the UNIT0 count value to SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO.

0: No effect

1: Update timer UNIT0

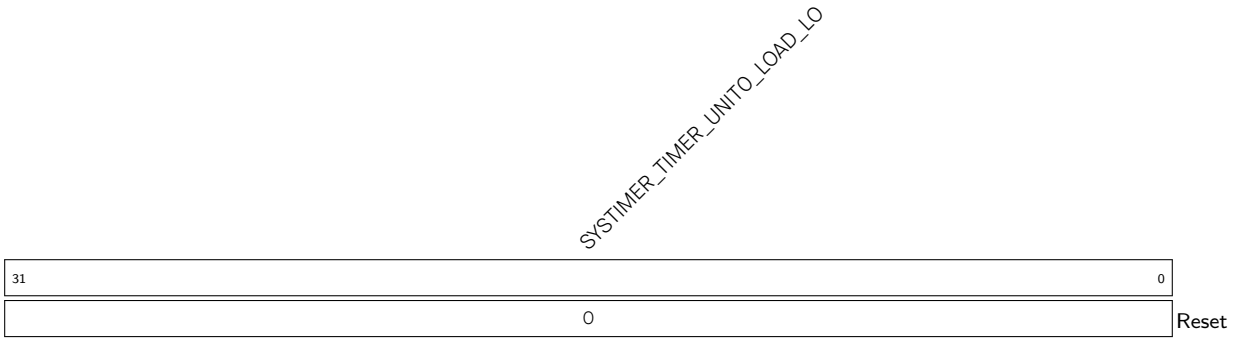
(WT)

Register 12.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



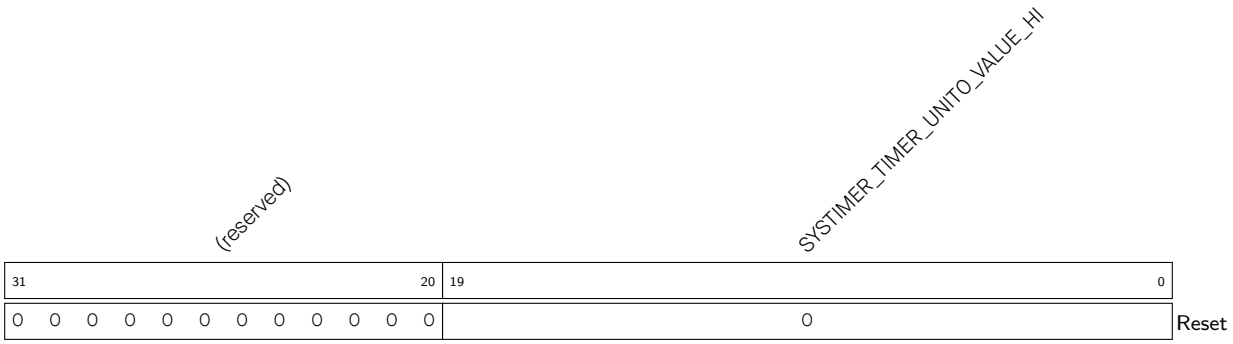
SYSTIMER_TIMER_UNIT0_LOAD_HI Configures the value to be loaded to UNIT0, high 20 bits. (R/W)

Register 12.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



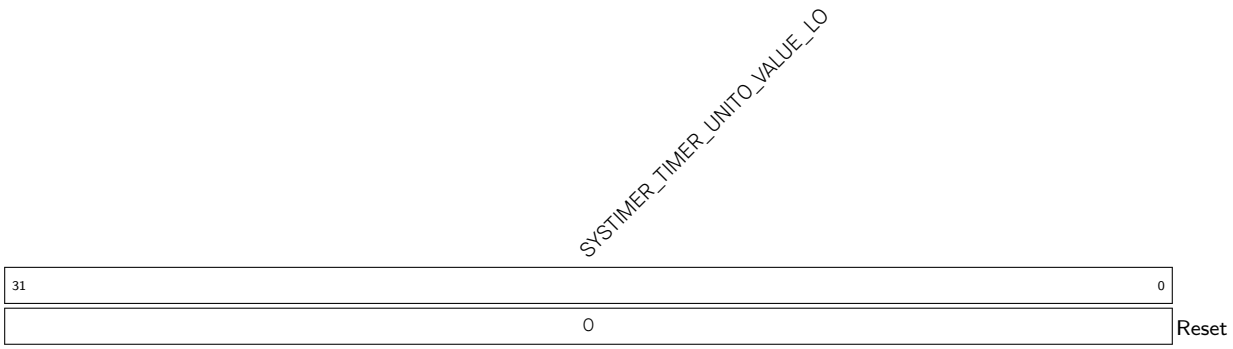
SYSTIMER_TIMER_UNIT0_LOAD_LO Configures the value to be loaded to UNIT0, low 32 bits. (R/W)

Register 12.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



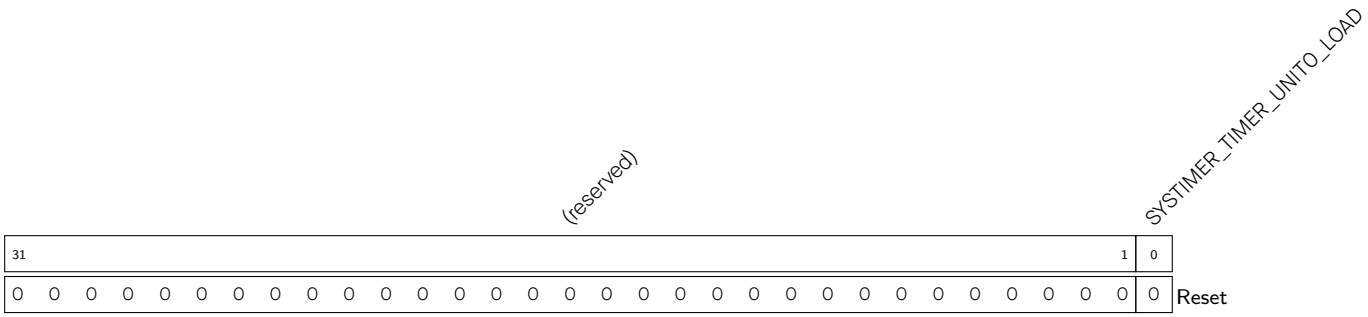
SYSTIMER_TIMER_UNIT0_VALUE_HI Represents UNIT0 read value, high 20 bits. (RO)

Register 12.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO Represents UNIT0 read value, low 32 bits. (RO)

Register 12.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



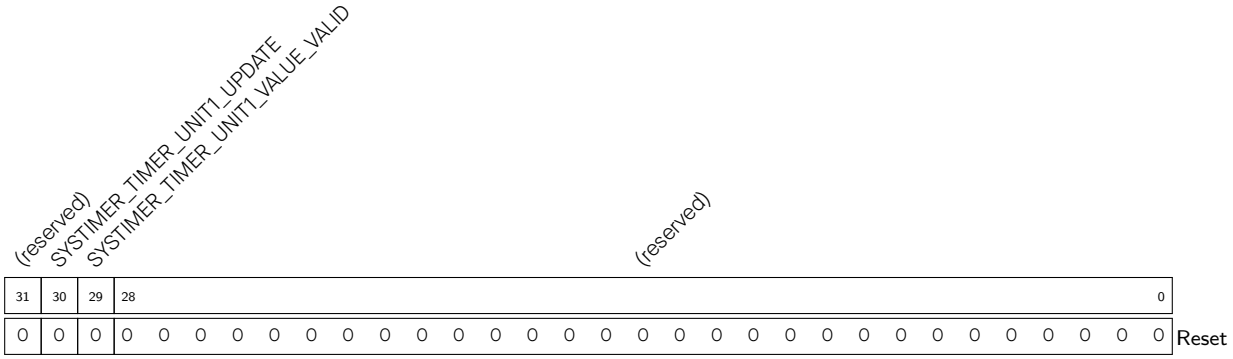
SYSTIMER_TIMER_UNIT0_LOAD Configures whether to reload the value of UNIT0, i.e., reloads the values of SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO to UNIT0.

0: No effect

1: Reload the value of UNIT0

(WT)

Register 12.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID Represents UNIT1 value is synchronized and valid. (R/SS/WTC)

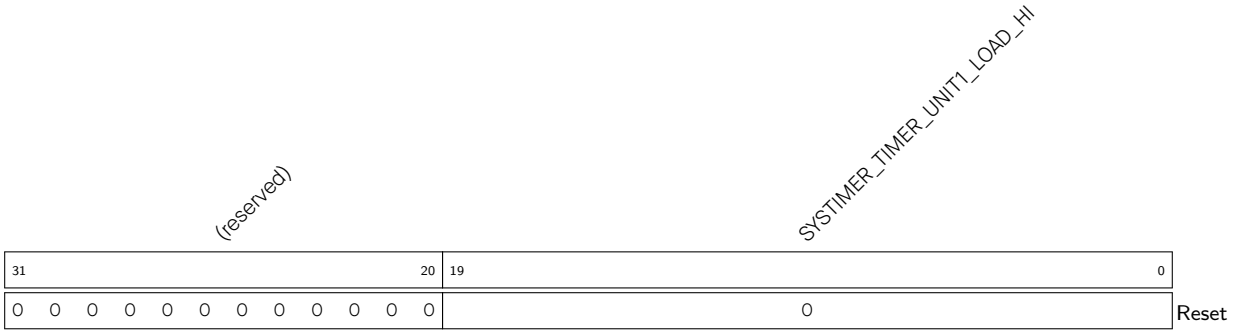
SYSTIMER_TIMER_UNIT1_UPDATE Configures whether to update timer UNIT1, i.e., reads the UNIT1 count value to SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO.

0: No effect

1: Update timer UNIT1

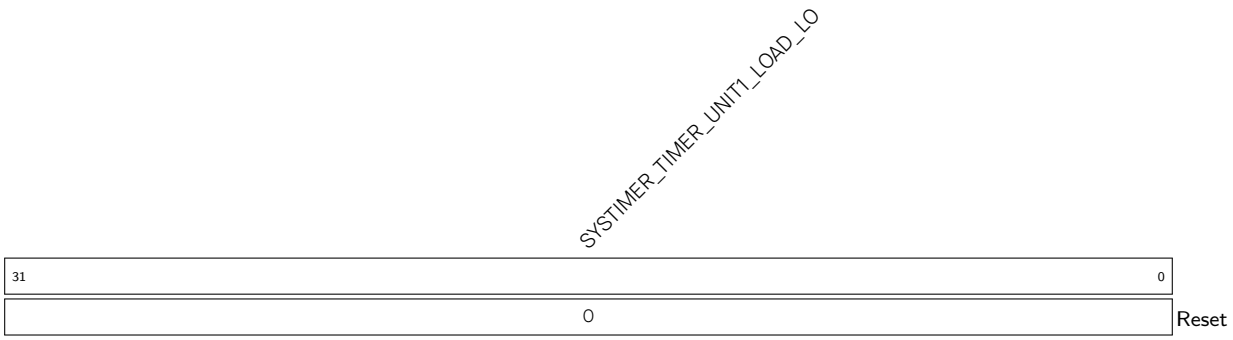
(WT)

Register 12.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



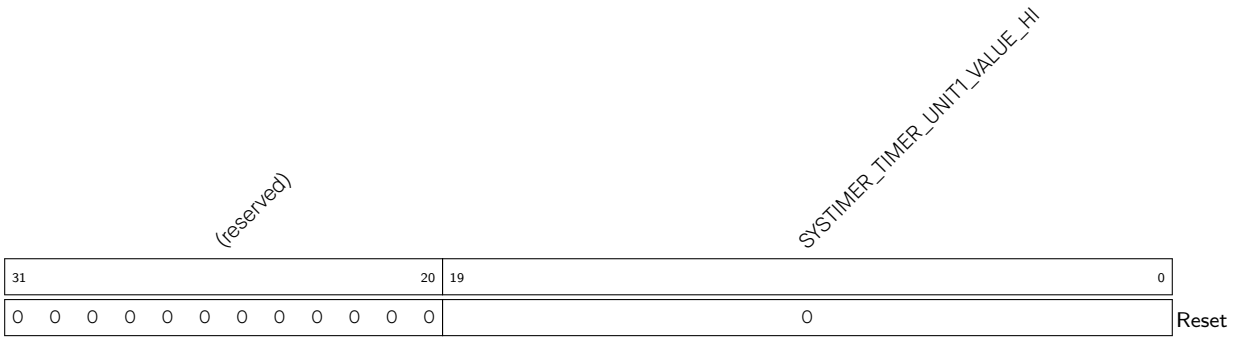
SYSTIMER_TIMER_UNIT1_LOAD_HI Configures the value to be loaded to UNIT1, high 20 bits. (R/W)

Register 12.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



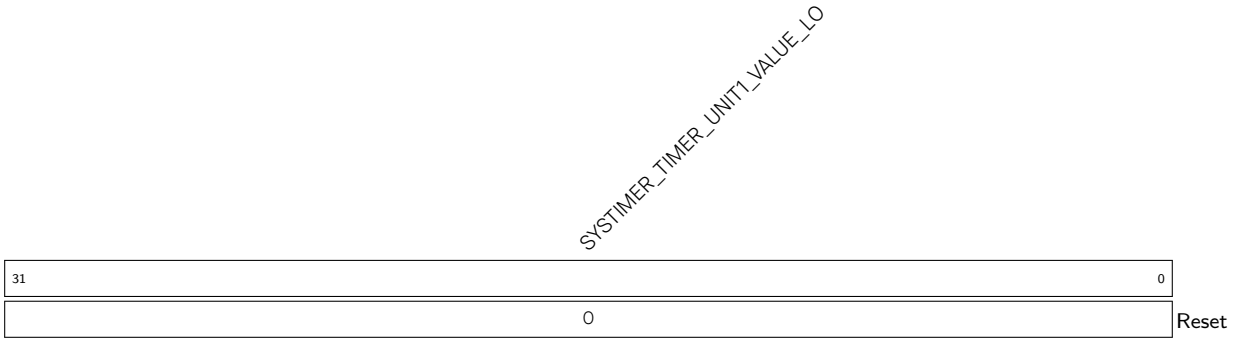
SYSTIMER_TIMER_UNIT1_LOAD_LO Configures the value to be loaded to UNIT1, low 32 bits. (R/W)

Register 12.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



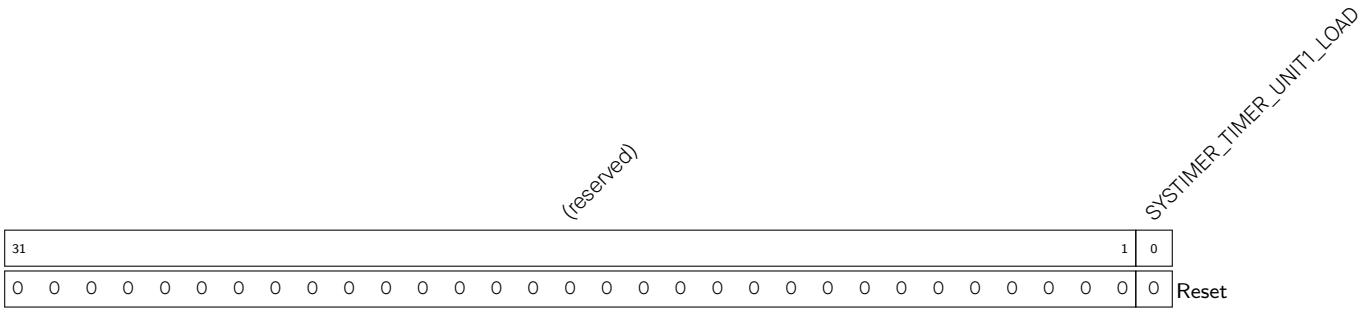
SYSTIMER_TIMER_UNIT1_VALUE_HI Represents UNIT1 read value, high 20 bits. (RO)

Register 12.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



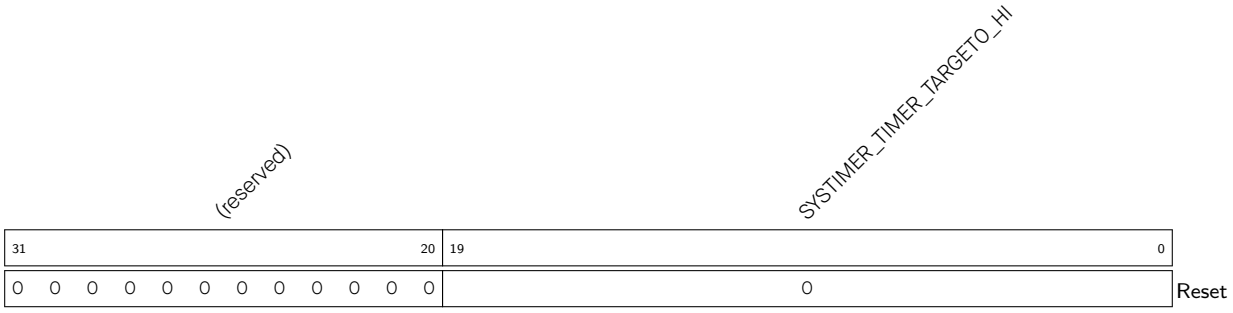
SYSTIMER_TIMER_UNIT1_VALUE_LO Represents UNIT1 read value, low 32 bits. (RO)

Register 12.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



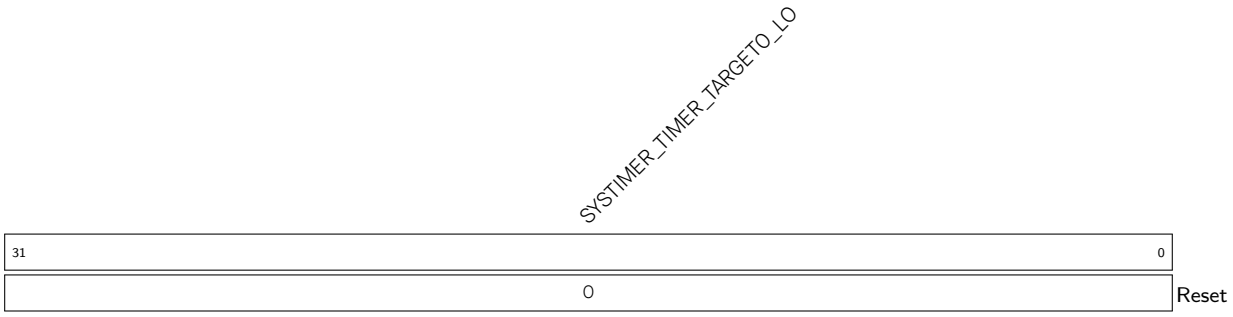
SYSTIMER_TIMER_UNIT1_LOAD Configures whether to reload the value of UNIT1, i.e., reload the values of SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO to UNIT1.
0: No effect
1: Reload the value of UNIT1
(WT)

Register 12.14. SYSTIMER_TARGET0_HI_REG (0x001C)



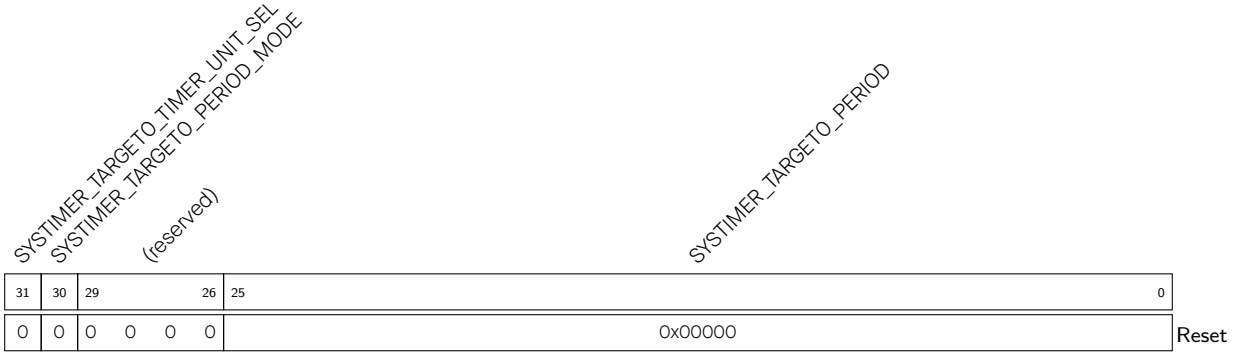
SYSTIMER_TIMER_TARGET0_HI Configures the alarm value to be loaded to COMP0, high 20 bits. (R/W)

Register 12.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO Configures the alarm value to be loaded to COMP0, low 32 bits. (R/W)

Register 12.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD Configures COMP0 alarm period. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Selects the two alarm modes for COMP0.

0: Target mode

1: Period mode

(R/W)

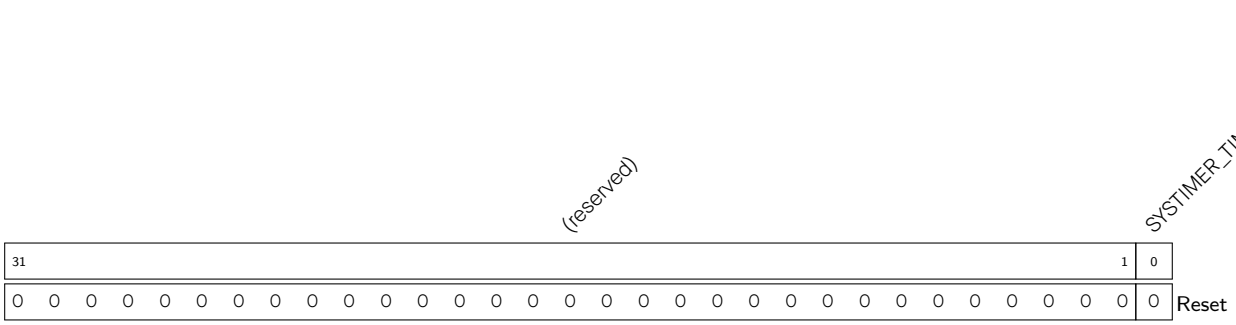
SYSTIMER_TARGET0_TIMER_UNIT_SEL Chooses the count value for comparison with COMP0.

0: Use the count value from UNIT*0*

1: Use the count value from UNIT*1*

(R/W)

Register 12.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



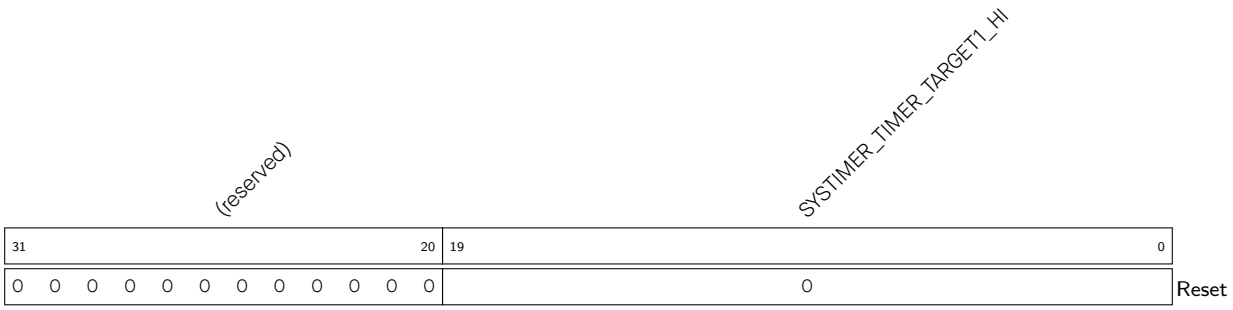
SYSTIMER_TIMER_COMP0_LOAD Configures whether to enable COMP0 synchronization, i.e., reload the alarm value/period to COMP0.

0: No effect

1: Enable COMP0 synchronization

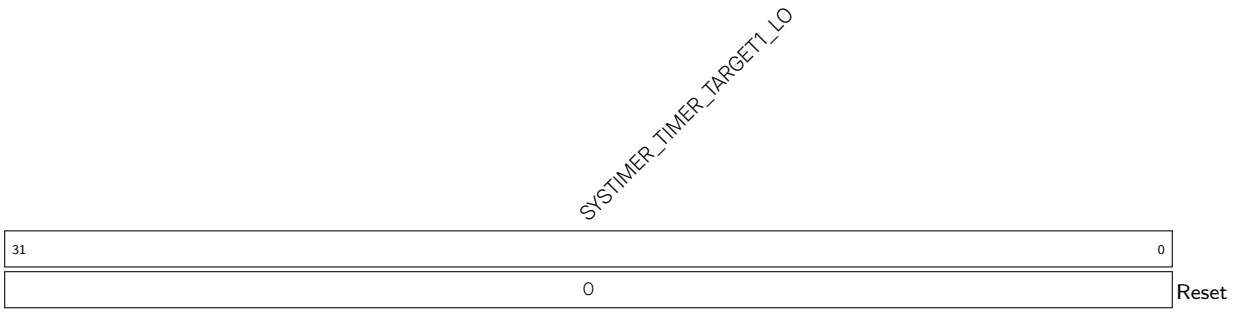
(WT)

Register 12.18. SYSTIMER_TARGET1_HI_REG (0x0024)



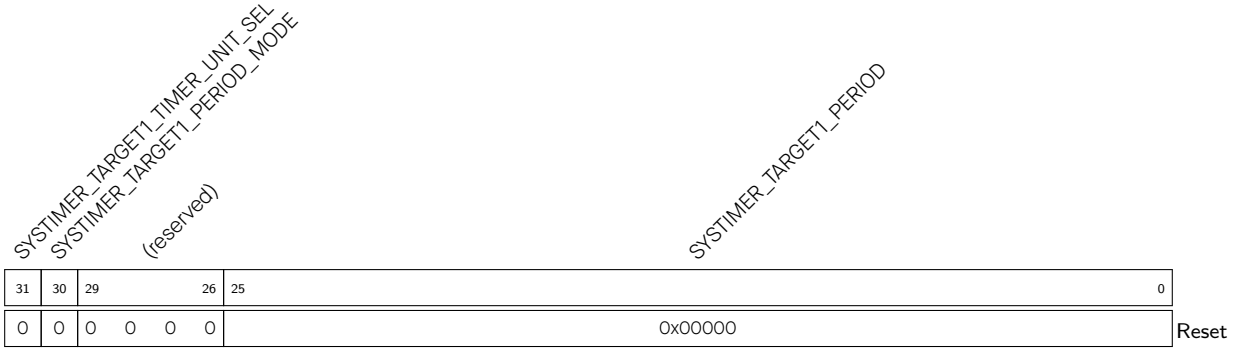
SYSTIMER_TIMER_TARGET1_HI Configures the alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 12.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO Configures the alarm value to be loaded to COMP1, low 32 bits. (R/W)

Register 12.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

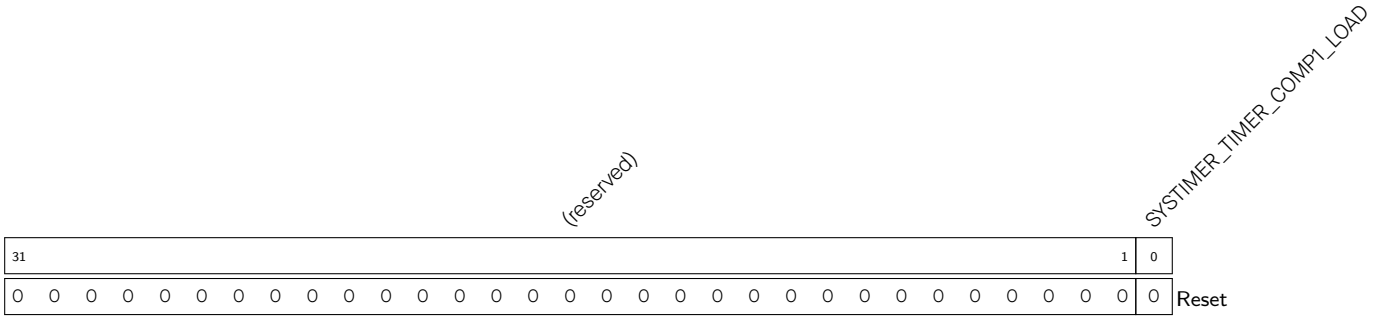


SYSTIMER_TARGET1_PERIOD Configures COMP1 alarm period. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Selects the two alarm modes for COMP1. See details in SYSTIMER_TARGET0_PERIOD_MODE. (R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL Chooses the count value for comparison with COMP1. See details in SYSTIMER_TARGET0_TIMER_UNIT_SEL. (R/W)

Register 12.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



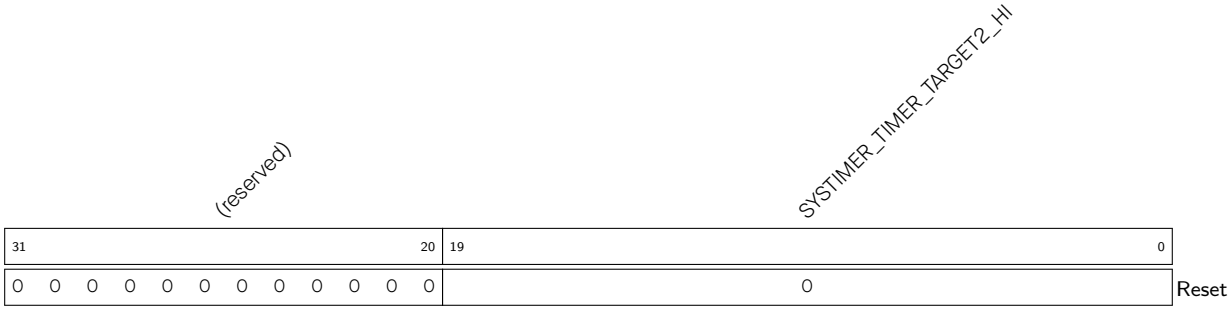
SYSTIMER_TIMER_COMP1_LOAD Configures whether to enable COMP1 synchronization, i.e., reload the alarm value/period to COMP1.

0: No effect

1: Enable COMP1 synchronization

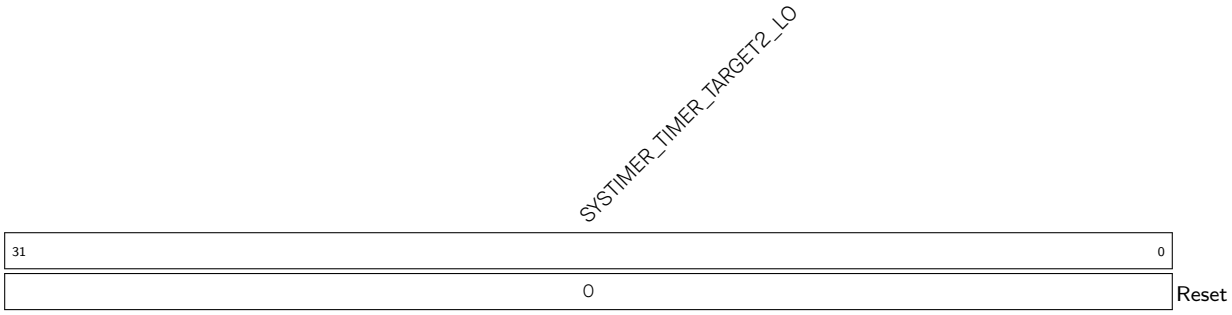
(WT)

Register 12.22. SYSTIMER_TARGET2_HI_REG (0x002C)



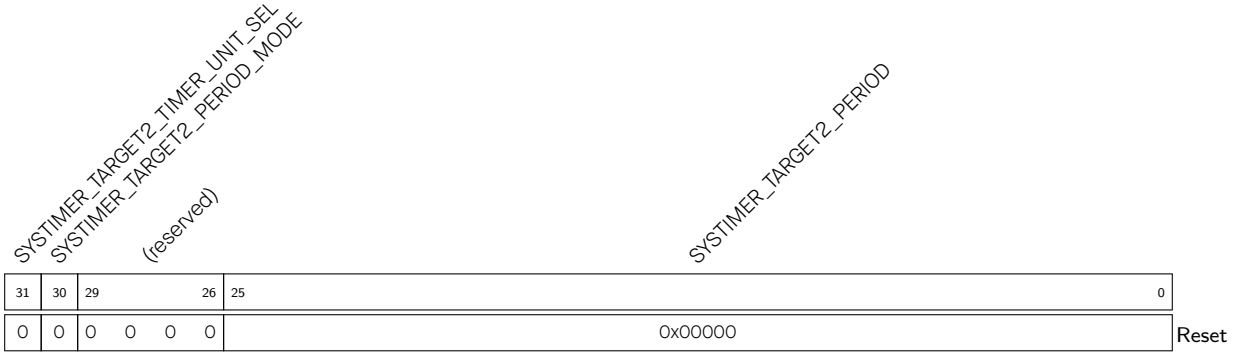
SYSTIMER_TIMER_TARGET2_HI Configures the alarm value to be loaded to COMP2, high 20 bits. (R/W)

Register 12.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO Configures the alarm value to be loaded to COMP2, low 32 bits. (R/W)

Register 12.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

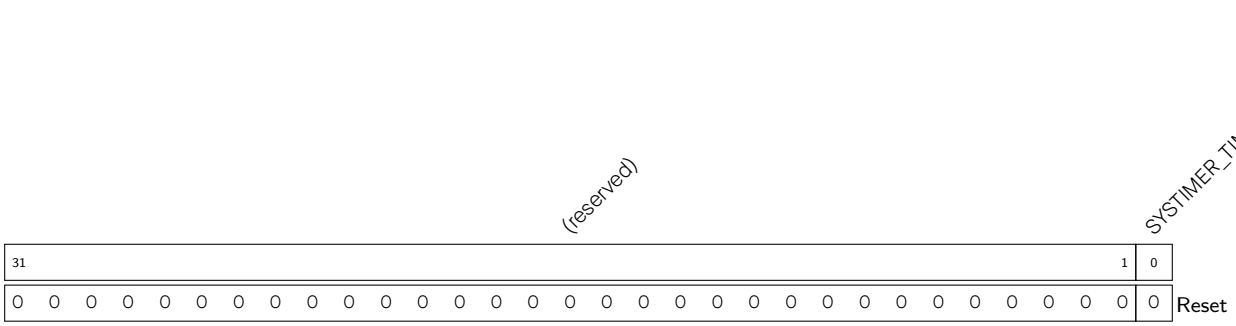


SYSTIMER_TARGET2_PERIOD Configures COMP2 alarm period. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Configures Configures the two alarm modes for COMP2. See details in SYSTIMER_TARGET0_PERIOD_MODE. (R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL Chooses the count value for comparison with COMP2. See details in SYSTIMER_TARGET0_TIMER_UNIT_SEL. (R/W)

Register 12.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



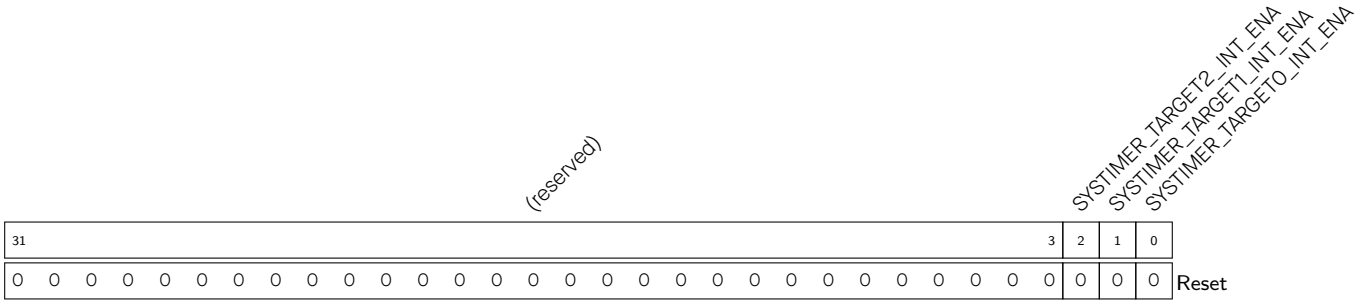
SYSTIMER_TIMER_COMP2_LOAD Configures whether to enable COMP2 synchronization, i.e., reload the alarm value/period to COMP2.

0: No effect

1: Enable COMP2 synchronization

(WT)

Register 12.26. SYSTIMER_INT_ENA_REG (0x0064)

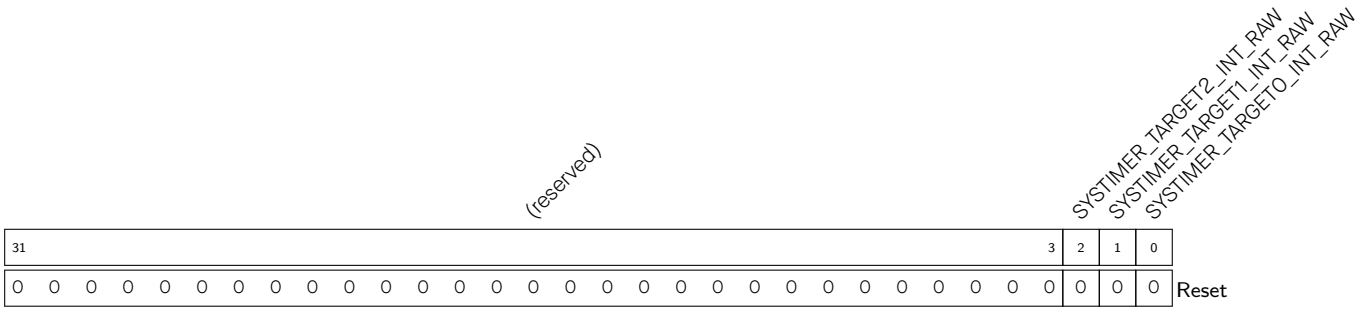


SYSTIMER_TARGET0_INT_ENA Write 1 to enable SYSTIMER_TARGET0_INT. (R/W)

SYSTIMER_TARGET1_INT_ENA Write 1 to enable SYSTIMER_TARGET1_INT. (R/W)

SYSTIMER_TARGET2_INT_ENA Write 1 to enable SYSTIMER_TARGET2_INT. (R/W)

Register 12.27. SYSTIMER_INT_RAW_REG (0x0068)

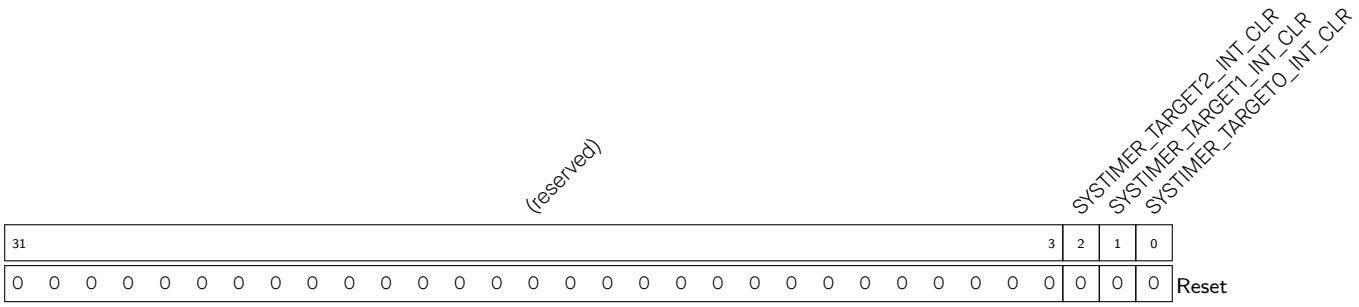


SYSTIMER_TARGET0_INT_RAW The raw interrupt status of SYSTIMER_TARGET0_INT. (R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW The raw interrupt status of SYSTIMER_TARGET1_INT. (R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW The raw interrupt status of SYSTIMER_TARGET2_INT. (R/WTC/SS)

Register 12.28. SYSTIMER_INT_CLR_REG (0x006C)

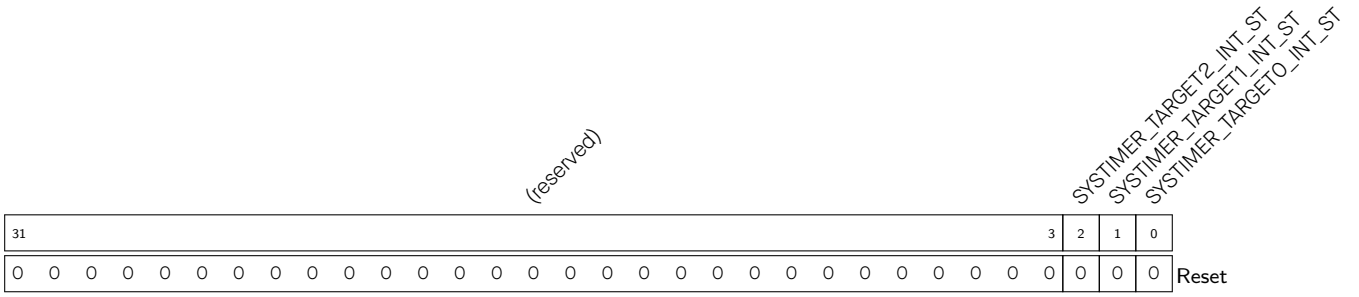


SYSTIMER_TARGET0_INT_CLR Write 1 to clear SYSTIMER_TARGET0_INT. (WT)

SYSTIMER_TARGET1_INT_CLR Write 1 to clear SYSTIMER_TARGET1_INT. (WT)

SYSTIMER_TARGET2_INT_CLR Write 1 to clear SYSTIMER_TARGET2_INT. (WT)

Register 12.29. SYSTIMER_INT_ST_REG (0x0070)

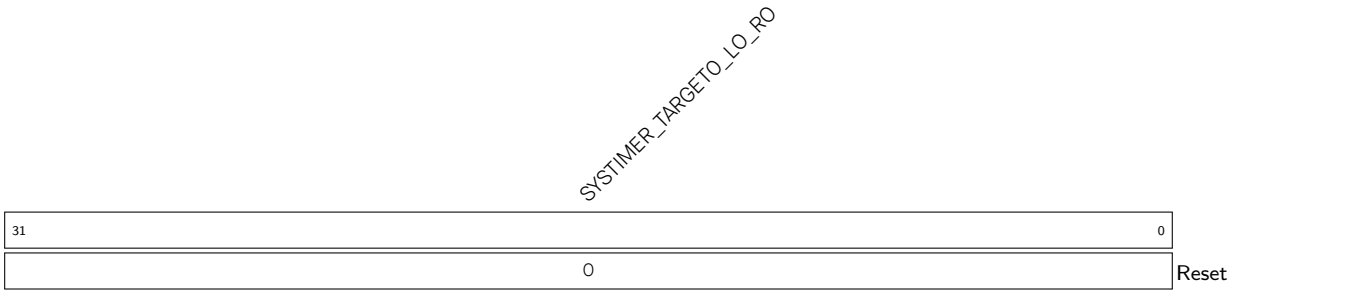


SYSTIMER_TARGET0_INT_ST The masked interrupt status of SYSTIMER_TARGET0_INT. (RO)

SYSTIMER_TARGET1_INT_ST The masked interrupt status of SYSTIMER_TARGET1_INT. (RO)

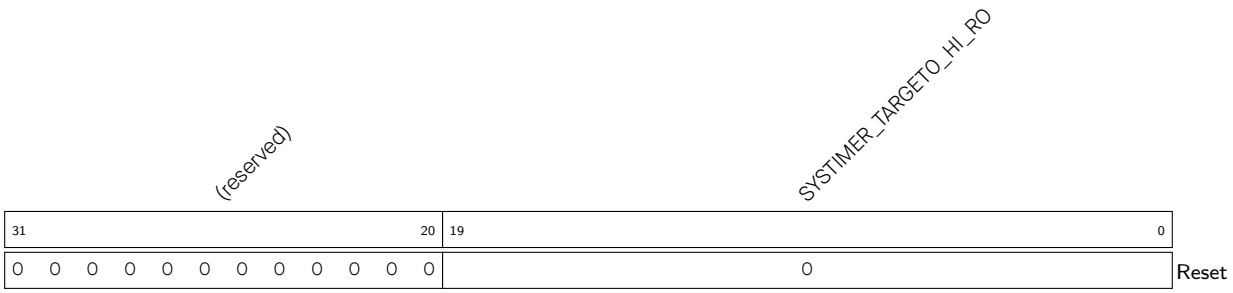
SYSTIMER_TARGET2_INT_ST The masked interrupt status of SYSTIMER_TARGET2_INT. (RO)

Register 12.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



SYSTIMER_TARGET0_LO_RO Represents the actual target value of COMP0, low 32 bits. (RO)

Register 12.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



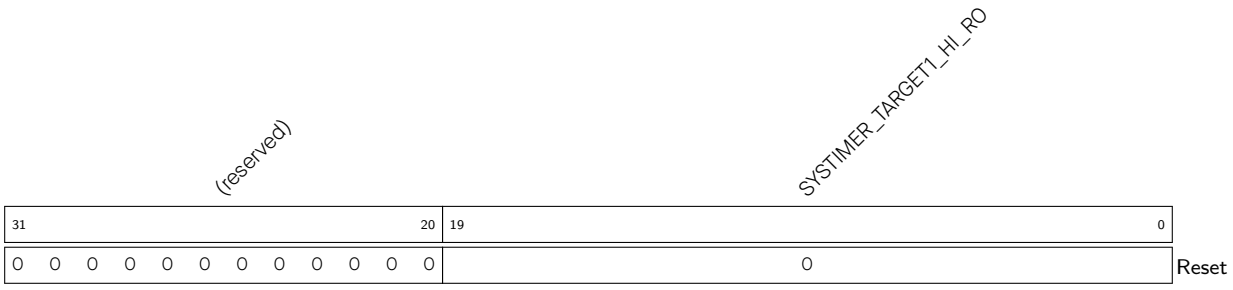
SYSTIMER_TARGET0_HI_RO Represents the actual target value of COMP0, high 20 bits. (RO)

Register 12.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)



SYSTIMER_TARGET1_LO_RO Represents the actual target value of COMP1, low 32 bits. (RO)

Register 12.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



SYSTIMER_TARGET1_HI_RO Represents the actual target value of COMP1, high 20 bits. (RO)

Register 12.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



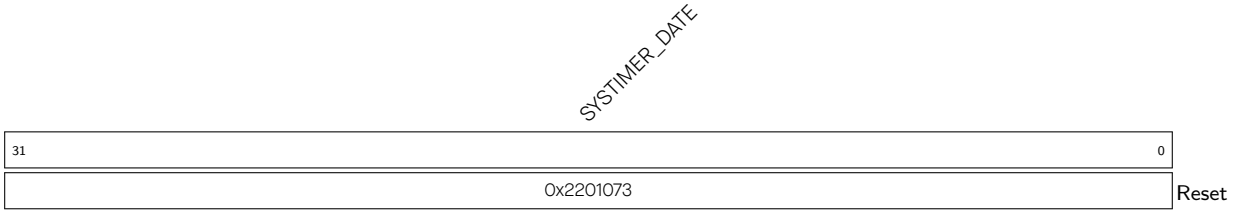
SYSTIMER_TARGET2_LO_RO Represents the actual target value of COMP2, low 32 bits. (RO)

Register 12.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)



SYSTIMER_TARGET2_HI_RO Represents the actual target value of COMP2, high 20 bits. (RO)

Register 12.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

Chapter 13

Timer Group (TIMG)

13.1 Overview

General-purpose timers can be used to precisely time an interval, trigger an interrupt after a particular interval (periodically and aperiodically), or act as a hardware clock. As shown in Figure 13.1-1, the ESP32-P4 chip contains two timer groups, namely timer group 0 and timer group 1 (hereinafter referred to as TIMG*n*, where *n* can be 0 or 1). Each timer group consists of two general-purpose timers (hereinafter referred to as T*x*, where *x* can be 0 or 1) and one Main System Watchdog Timer (MWDT). The general-purpose timer is based on a 16-bit prescaler and a 54-bit auto-reload-capable up-down counter.

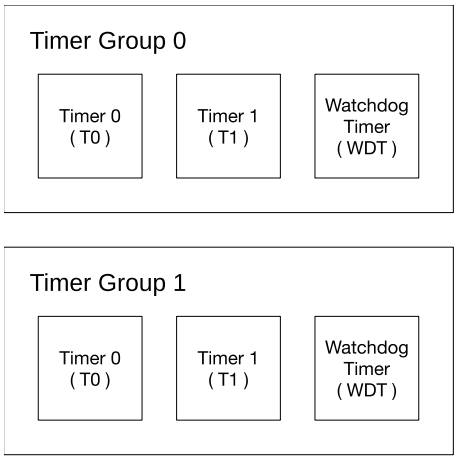


Figure 13.1-1. Timer Group Overview

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in Chapter 14 *Watchdog Timers (WDT)*. Therefore, the term "timer" within this chapter refers to the general-purpose timer.

13.2 Features

The timer's features are summarized as follows:

- A 54-bit time-base counter programmable to incrementing or decrementing
- Three clock sources: PLL_F80M_CLK or XTAL_CLK or RC_FAST_CLK
- A 16-bit clock prescaler, from 2 to 65536
- Able to read real-time value of the time-base counter
- Able to halt and resume the time-base counter

- Programmable alarm generation
- Timer value reload — Auto-reload at alarm or software-controlled instant reload
- Calculate clock frequency — Calculate the measured frequency of the clock, which can be called TIMG0_CALI_CLK, based on the crystal clock
- Level interrupt generation
- Support several ETM tasks and events

13.3 Functional Description

Figure 13.3-1 shows Timer T*x* in timer group TIMG*n*. T*x* contains a 16-bit integer divider as a prescaler, a timer-based counter, and a comparator for alarm generation.

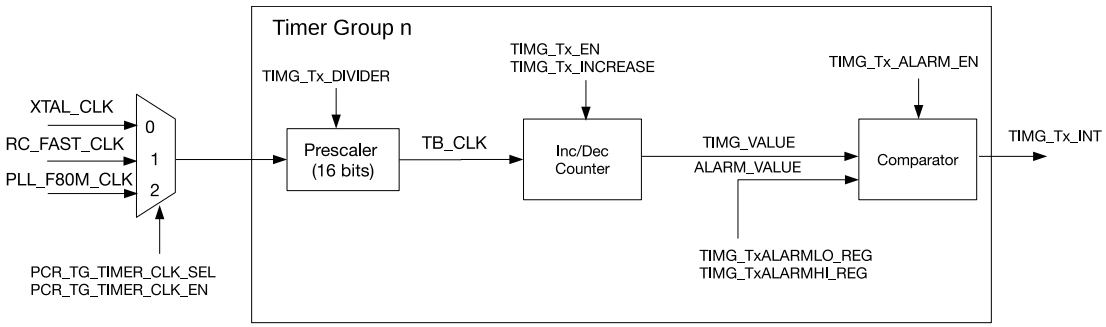


Figure 13.3-1. Timer Group Architecture

13.3.1 16-bit Prescaler and Clock Selection

Take the Timer T*x* in timer group TIMG*n* as an example:

- The timer can select its clock source by setting the HP_SYS_CLKRST_TIMERGRP*n*_T*x*_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register or the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register. When the field is 0, XTAL_CLK is selected; when the field is 1, RC_FAST_CLK is selected and when the field is 2, PLL_F80M_CLK is selected.
- The selected clock can be switched on by setting HP_SYS_CLKRST_TIMERGRP*n*_T*x*_CLK_EN field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register or the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register to 1 and switched off by setting it to 0. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB_CLK) used by the time-base counter. The divisor of the prescaler can be configured through the TIMG_T*x*_DIVIDER field.

TIMG_T*x*_DIVIDER field can be configured as 0 ~ 65535 for a divisor range of 2 ~ 65536. To be more specific, when TIMG_T*x*_DIVIDER is configured as:

- 0: the divisor is 65536
- 1: the divisor is 2
- 2: the divisor is also 2
- 3 ~ 65535: the divisor is 3 ~ 65535

To modify the 16-bit prescaler, please first configure the TIMG_T*x*_DIVIDER field, and then set TIMG_T*x*_DIVCNT_RST to 1. Meanwhile, the timer must be disabled (i.e., TIMG_T*x*_EN should be cleared). Otherwise, the result can be unpredictable.

13.3.2 54-bit Time-base Counter

The 54-bit time-base counter is based on TB_CLK and can be configured to increment or decrement via the TIMG_T*x*_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG_T*x*_EN field, respectively. When enabled, the time-base counter increments or decrements on each cycle of TB_CLK. When disabled, the time-base counter is essentially frozen. Note that the TIMG_T*x*_INCREASE field can be changed no matter whether TIMG_T*x*_EN is set or not, and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_T*x*UPDATE_REG, the current value of the 54-bit timer starts to be latched into the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers containing the lower 32-bits and higher 22-bits, respectively. When TIMG_T*x*UPDATE_REG is cleared by hardware, it indicates the latch operation has been completed and current timer value can be read from the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers. TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_T*x*UPDATE_REG is written to again.

13.3.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 13.3.4).

The 54-bit alarm value is configured using TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG, which represent the lower 32-bits and higher 22-bits of the alarm value, respectively. However, the configured alarm value is ineffective until the alarm is enabled by setting the TIMG_T*x*_ALARM_EN field. To avoid alarm being enabled "too late" (i.e., the timer value has already passed the alarm value when the alarm is enabled), the hardware will trigger the alarm immediately if the current timer value is:

- higher than the alarm value (within a defined range) when the up-down counter increments
- lower than the alarm value (within a defined range) when the up-down counter decrements

Table 13.3-1 and Table 13.3-2 show the relationship between the current value of the timer, the alarm value, and when an alarm is triggered. The current time value and the alarm value are defined as follows:

- TIMG_VALUE = {TIMG_T*x*HI_REG, TIMG_T*x*LO_REG}
- ALARM_VALUE = {TIMG_T*x*ALARMHI_REG, TIMG_T*x*ALARMLO_REG}

Table 13.3-1. Alarm Generation When Up-Down Counter Increments

Scenario	Range	Alarm
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	Triggered
2	$0 <$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE up to ALARM_VALUE
3	$0 \leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	Triggered
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting up from 0 after reaching the timer's maximum value and counts TIMG_VALUE up to ALARM_VALUE

Table 13.3-2. Alarm Generation When Up-Down Counter Decrements

Scenario	Range	Alarm
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	Triggered
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE down to ALARM_VALUE
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	Triggered
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting down from the timer's maximum value after reaching the minimum value and counts TIMG_VALUE down to ALARM_VALUE

When an alarm occurs, the TIMG_T*x*_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_T*x*_ALARM_EN is set next time.

13.3.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI fields that correspond to the lower 32-bits and higher 22-bits of the timer's new value, respectively. However, writing a reload value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to TIMG_T*x*LOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG_T*x*_EN is set, the timer will continue incrementing or decrementing from the new value. In this case if TIMG_T*x*_ALARM_EN is set, the timer will still trigger alarms in scenarios listed in Table 13.3-1 and 13.3-2. If TIMG_T*x*_EN is cleared, the timer will remain frozen at the new value until counting is re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs, thus allowing the timer to continue incrementing or decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the TIMG_T*x*_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment or decrement past the alarm value after an alarm.

13.3.5 Event Task Matrix Feature

The TIMG*n* on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows TIMG*n*'s ETM tasks to be triggered by any peripherals' ETM events, or TIMG*n*'s ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to TIMG*n*. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

TIMG*n* can receive the following ETM tasks:

- TIMER*n*_TASK_CNT_START_TIMER*x* (*n*:0-1): When triggered, it will enable the time-base counter of T*x*.
- TIMER*n*_TASK_CNT_STOP_TIMER*x* (*n*:0-1): When triggered, it will disable the time-base counter of T*x*.

  Note:
  The above two ETM tasks have the same function as the APB configuration TIMG_T*x*_EN. When these operations occur at the same time, the priority of each operation from high to low is as follows:

  1. TIMER*n*_TASK_CNT_START_TIMER*x*: When triggered, it will enable the time-base counter of T*x*;
  2. TIMER*n*_TASK_CNT_STOP_TIMER*x*: When triggered, it will disable the time-base counter of T*x*;
  3. APB configuration TIMG_T*x*_EN: When triggered, it will enable or disable the time-base counter of T*x*.

- TIMER*n*_TASK_ALARM_START_TIMER*x* (*n*:0-1): When triggered, it will enable the alarm generation of T*x*.

  Note:
  Alarm generation can also be enabled through APB method configuring TIMG_T*x*_ALARM_EN and hardware events. When these operations occur at the same time, the priority of each operation from high to low is as follows:

  1. TIMER*n*_TASK_ALARM_START_TIMER*x*: When triggered, it will enable the alarm generation of T*x*;
  2. Alarm events of T*x*: When triggered, it will disable the alarm generation of T*x*;
  3. APB configuration TIMG_T*x*_ALARM_EN: When triggered, it will enable or disable the alarm generation of T*x*.

- TIMER*n*_TASK_CNT_CAP_TIMER*x*(*n*:0-1): When triggered, it will update the current counter value to the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers.
- TIMER*n*_TASK_CNT_RELOAD_TIMER*x* (*n*:0-1): When triggered, it will overwrite the current counter value with the reload value stored in TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.

TIMG*n* can generate the following ETM events:

- TIMER*n*_EVT_CNT_CMP_TIMER*x* (*n*:0-1): Indicates the interrupt event of the T*x* timer in TIMG*n*.

  Note:
  All the ETM tasks and events of TIMG*n* will not take effect until the TIMG*n*_ETM_EN is set to 1.

In practical applications, timer groups' ETM events can trigger their own ETM tasks.

For example, TIMER*n*_TASK_ALARM_START_TIMER*x* (*n*:0-1) can be triggered by TIMER*n*_EVT_CNT_CMP_TIMER*x* (*n*:0-1) to realize periodic alarm. For configuration steps, please refer to

13.4.4 Timer as Periodic Alarm by ETM.

13.3.6 RTC_SLOW_CLK Frequency Calculation

Using XTAL_CLK as a reference, it is possible to calculate the frequency of clock sources for RTC_SLOW_CLK (i.e., RC_SLOW_CLK, RC_FAST_DIV_CLK, and XTAL32K_CLK) as follows:

1. Start periodic or one-shot frequency calculation (see Section 13.4.5 for details);
2. Once receiving the signal to start calculation, the counter of XTAL_CLK and the counter of RTC_SLOW_CLK begin to work at the same time. When the counter of RTC_SLOW_CLK counts to C0, the two counters stop counting simultaneously;
3. Assume the value of XTAL_CLK's counter is C1, and the frequency of RTC_SLOW_CLK would be calculated as: $f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

13.3.7 Interrupts

ESP32-P4's TIMG*n* can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- TG*n*_T*x*_INT
- TG*n*_WDT_INT

There are several internal interrupt sources from TIMG*n* that can generate the above interrupt signal(s). The interrupt sources from TIMG*n* are listed with their trigger conditions and the resulted interrupt signal(s) in Table 13.3-3.

Table 13.3-3. TIMG*n*'s Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TIMG_T*x*_INT	Alarm generated by T*x*	TG*n*_T*x*_INT
TIMG_WDT_INT	Alarm generated by WDT	TG*n*_WDT_INT

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 13.5 *Register Summary*.

13.4 Configuration and Usage

13.4.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by configuring the HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL21_REG register.

- Configure the 16-bit prescaler by setting TIMG_T*x*_DIVIDER.
- Configure the timer direction by setting or clearing TIMG_T*x*_INCREASE.
- Set the timer's starting value by writing the starting value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI, then reloading it into the timer by writing any value to TIMG_T*x*LOAD_REG.

2. Start the timer by setting TIMG_T*x*_EN.
3. Get the timer's current value.
   - Write any value to TIMG_T*x*UPDATE_REG to latch the timer's current value.
   - Wait until TIMG_T*x*UPDATE_REG is cleared by hardware.
   - Read the latched timer value from TIMG_T*x*LO_REG and TIMG_T*x*HI_REG.

13.4.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 13.4.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG.
   - Enable interrupt by setting TIMG_T*x*_INT_ENA.
3. Disable auto reload by clearing TIMG_T*x*_AUTORELOAD.
4. Start the alarm by setting TIMG_T*x*_ALARM_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

13.4.3 Timer as Periodic Alarm by APB

1. Configure the time-base counter following step 1 in Section 13.4.1.
2. Configure the alarm following step 2 in Section 13.4.2.
3. Enable auto reload by setting TIMG_T*x*_AUTORELOAD and configure the reload value via TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.
4. Start the alarm by setting TIMG_T*x*_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e., different alarm interval per iteration), then TIMG_T*x*ALARMLO_REG, TIMG_T*x*ALARMHI_REG, TIMG_T*x*_LOAD_LO, and TIMG_T*x*_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_T*x*_ALARM_EN.
6. Stop the timer (on final alarm iteration).

- Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
- Disable the timer by clearing TIMG_T*x*_EN.

13.4.4 Timer as Periodic Alarm by ETM

1. Enable the ETM module's clock
2. Map ETM event to ETM task (which means using the event to trigger the task)
   - If TIMG_T*x*_AUTORELOAD is set to 1, map TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) to the TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) by one ETM channel.
   - If TIMG_T*x*_AUTORELOAD is set to 0, in addition to mapping TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) to the TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1), the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) should also be mapped to TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1) by another ETM channel.
3. Choose to enable the one or two ETM channels.
4. Set TIMER_ETM_EN to 1 to enable timer group's ETM events and tasks.
5. Configure the time-base counter following step 1 in Section 13.4.1.
6. Configure the alarm following step 2 in Section 13.4.2.
7. Configure the reload value via TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.
8. Handle the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1).
   - When alarm generates, the TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) also generates, and the alarm generation will be disabled by the alarm.
   - If TIMG_T*x*_AUTORELOAD is 1, the current counter value is overwritten by the reloaded value. The alarm generation will be reopen by TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1).
   - If TIMG_T*x*_AUTORELOAD is 0, the current counter value is overwritten by the reloaded value because of the TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1). The alarm generation will be reopen by TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1).
9. Stop the timer (on final alarm iteration).
   - Disable the ETM channels used to map timer group's event and task
   - Set TIMER_ETM_EN to 0.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

13.4.5 RTC_SLOW_CLK Frequency Calculation

1. One-shot frequency calculation
   - Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN to enable clock gate of TIMG0_CALI_CLK.
   - Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL to select one clock as the clock source of TIMG0_CALI_CLK.

- Configure HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM to output TIMG0_CALI_CLK after integer prescaler.
- Configure the time of calculation via TIMG_RTC_CALI_MAX.
- Select one-shot frequency calculation by clearing TIMG_RTC_CALI_START_CYCLING, and enable the two counters via TIMG_RTC_CALI_START.
- Once TIMG_RTC_CALI_RDY becomes 1, read TIMG_RTC_CALI_VALUE to get the value of XTAL_CLK's counter, and calculate the frequency of RTC_SLOW_CLK according to the formula in Section 13.3.6.

2. Periodic frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of RTC_SLOW_CLK) via HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL, and configure the time of calculation via TIMG_RTC_CALI_MAX.
   - Select periodic frequency calculation by enabling TIMG_RTC_CALI_START_CYCLING.
   - When TIMG_RTC_CALI_CYCLING_DATA_VLD is 1, TIMG_RTC_CALI_VALUE is valid.
3. Timeout
   If the counter of RTC_SLOW_CLK cannot finish counting in TIMG_RTC_CALI_TIMEOUT_RST_CNT cycles, TIMG_RTC_CALI_TIMEOUT will be set to indicate a timeout.

13.5 Register Summary

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

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

Name	Description	Address	Access
T0 Control and configuration registers			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	varies
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 22 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMG_T0LO_REG or TIMG_T0HI_REG	0x000C	R/W/SC
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high 22 bits	0x0014	R/W
TIMG_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_T0LOADHI_REG	Timer 0 reload value, high 22 bits	0x001C	R/W
TIMG_T0LOAD_REG	Write to reload timer from TIMG_T0LOADLO_REG or TIMG_T0LOADHI_REG	0x0020	WT
T1 Control and configuration registers			
TIMG_T1CONFIG_REG	Timer 1 configuration register	0x0024	varies
TIMG_T1LO_REG	Timer 1 current value, low 32 bits	0x0028	RO
TIMG_T1HI_REG	Timer 1 current value, high 22 bits	0x002C	RO
TIMG_T1UPDATE_REG	Write to copy current timer value to TIMG_T1LO_REG or TIMG_T1HI_REG	0x0030	R/W/SC
TIMG_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x0034	R/W
TIMG_T1ALARMHI_REG	Timer 1 alarm value, high 22 bits	0x0038	R/W
TIMG_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x003C	R/W
TIMG_T1LOADHI_REG	Timer 1 reload value, high 22 bits	0x0040	R/W
TIMG_T1LOAD_REG	Write to reload timer from TIMG_T1LOADLO_REG or TIMG_T1LOADHI_REG	0x0044	WT
WDT Control and configuration registers			
TIMG_WDTCONFIG0_REG	Watchdog timer configuration register	0x0048	varies
TIMG_WDTCONFIG1_REG	Watchdog timer prescaler register	0x004C	varies
TIMG_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x0050	R/W
TIMG_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x0054	R/W
TIMG_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x0058	R/W
TIMG_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x005C	R/W
TIMG_WDTFEED_REG	Write to feed the watchdog timer	0x0060	WT
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
RTC CALI Control and configuration registers			
TIMG_RTCCALICFG_REG	RTC calibration configure register	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC calibration configure register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC calibration configure register 2	0x0080	varies

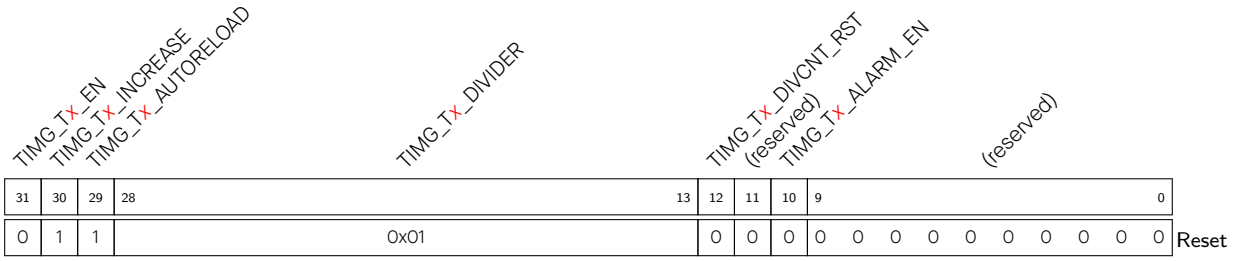
Name	Description	Address	Access
Interrupt registers			
TIMG_INT_ENA_TIMERS_REG	Interrupt enable bits	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	Raw interrupt status	0x0074	R/SS/WTC
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x007C	WT
Version register			
TIMG_NTIMERS_DATE_REG	Timer version control register	0x00F8	R/W
Clock configuration registers			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

13.6 Registers

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

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

Register 13.1. TIMG_TxCONFIG_REG (x: 0-1) (0x0000+0x24*x)



TIMG_Tx_ALARM_EN Configures whether to enable Timer Tx alarm function. This bit will be automatically cleared once an alarm occurs.

0: Disable

1: Enable

(R/W/SC)

TIMG_Tx_DIVCNT_RST Configures to reset Timer Tx's clock divider counter.

0: No effect

1: Reset

(WT)

TIMG_Tx_DIVIDER Represents Timer x clock (Tx_clk) prescaler value. (R/W)

TIMG_Tx_AUTORELOAD Configures to enable Timer Tx auto-reload function at the time of alarm.

0: No effect

1: Enable

(R/W)

TIMG_Tx_INCREASE Configures the counting direction of Timer Tx time-base counter.

0: Decrement

1: Increment

(R/W)

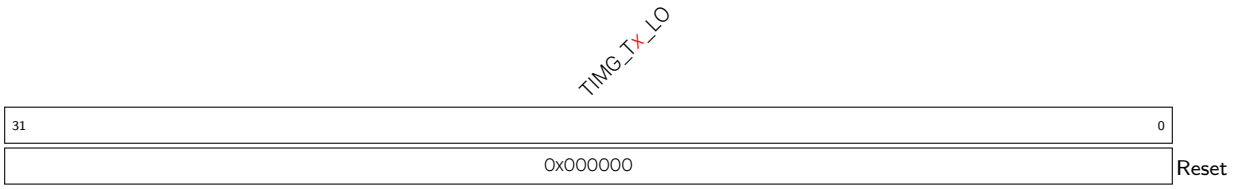
TIMG_Tx_EN Configures whether to enable Timer Tx time-base counter.

0: Disable

1: Enable

(R/W/SS/SC)

Register 13.2. TIMG_T*x*LO_REG (*x*: 0-1) (0x0004+0x24x*)**

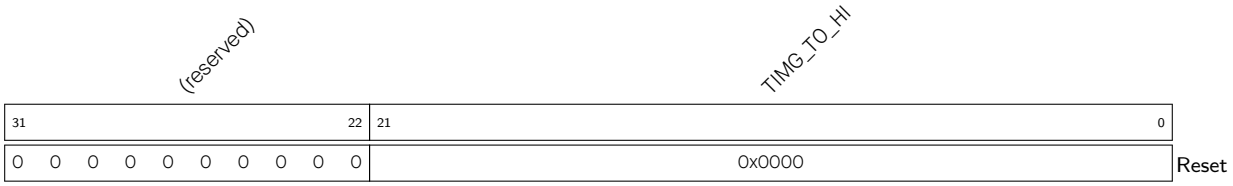


TIMG_T*x*_LO Represents the low 32 bits of the time-base counter of Timer T*x*. Valid only after writing to TIMG_T*x*UPDATE_REG.

Measurement unit: T*x*_clk.

(RO)

Register 13.3. TIMG_T*x*HI_REG (*x*: 0-1) (0x0008+0x24x*)**

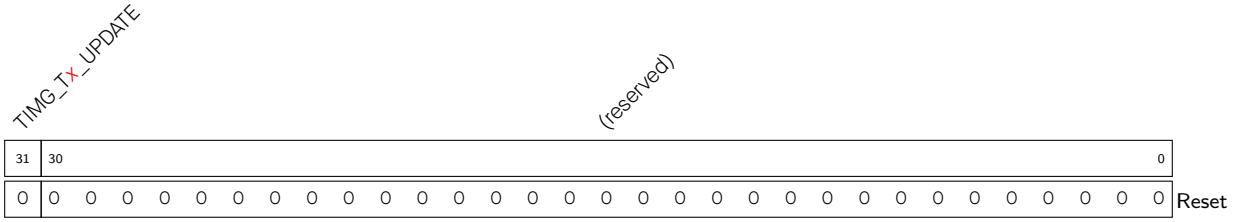


TIMG_T*x*_HI Represents the high 22 bits of the time-base counter of Timer T*x*. Valid only after writing to TIMG_T*x*UPDATE_REG.

Measurement unit: T*x*_clk.

(RO)

Register 13.4. TIMG_T*x*UPDATE_REG (*x*: 0-1) (0x000C+0x24x*)**



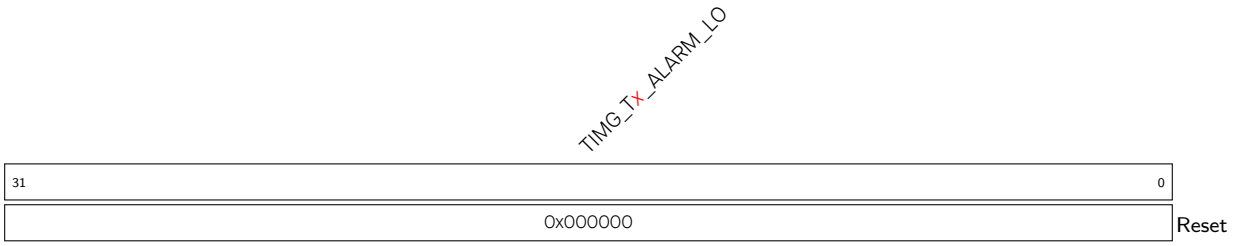
TIMG_T*x*_UPDATE Configures to latch the counter value.

0: Latch

1: Latch

(R/W/SC)

Register 13.5. TIMG_TxALARMLO_REG (x: 0-1) (0x0010+0x24*x)

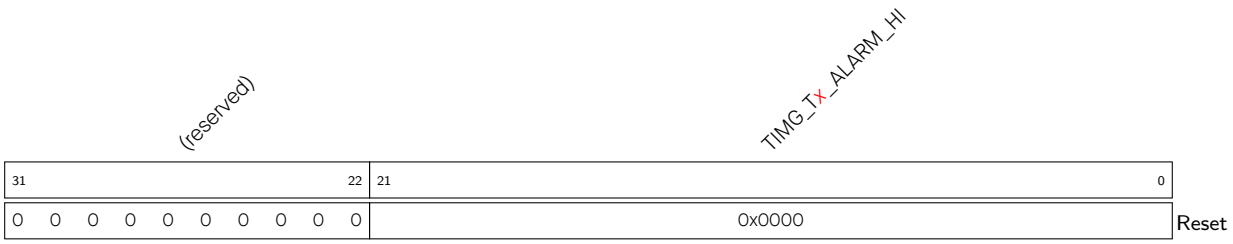


TIMG_Tx_ALARM_LO Configures the low 32 bits of Timer Tx alarm trigger time-base counter value. Valid only when TIMG_Tx_ALARM_EN is 1.

Measurement unit: Tx_clk.

(R/W)

Register 13.6. TIMG_TxALARMHI_REG (x: 0-1) (0x0014+0x24*x)

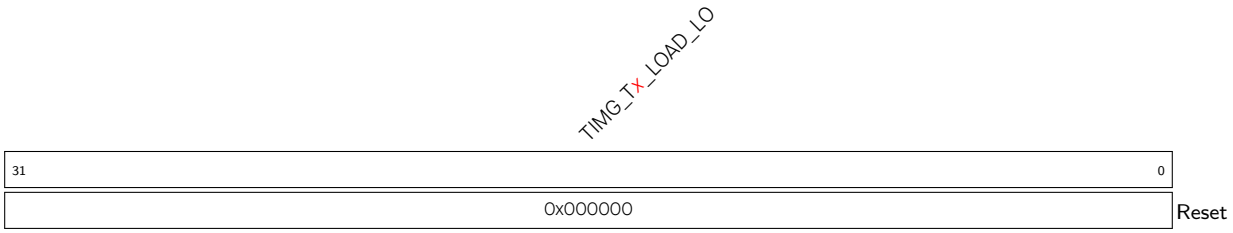


TIMG_Tx_ALARM_HI Configures the high 22 bits of Timer Tx alarm trigger time-base counter value. Valid only when TIMG_Tx_ALARM_EN is 1.

Measurement unit: Tx_clk.

(R/W)

Register 13.7. TIMG_TxLOADLO_REG (x: 0-1) (0x0018+0x24*x)

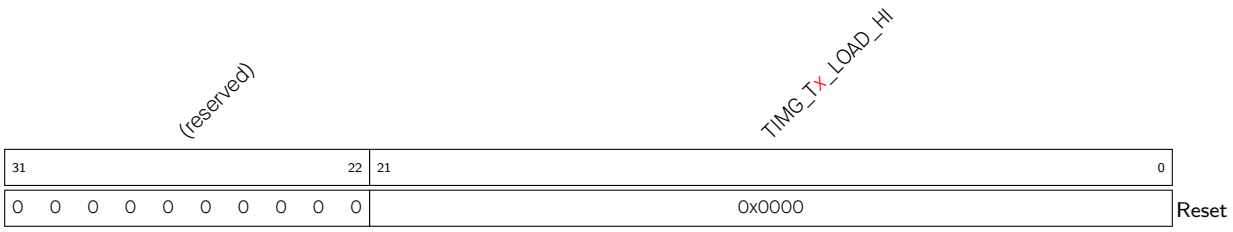


TIMG_Tx_LOAD_LO Configures low 32 bits of the value that a reload will load onto Timer Tx time-base counter.

Measurement unit: Tx_clk.

(R/W)

Register 13.8. TIMG_T*x*LOADHI_REG (*x*: 0-1) (0x001C+0x24x*)**

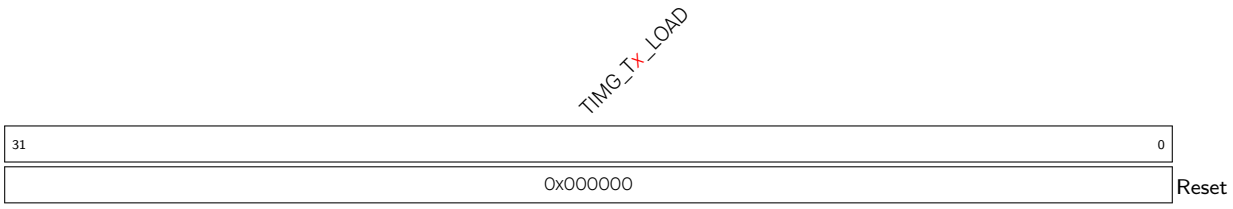


TIMG_T*x*_LOAD_HI Configures high 22 bits of the value that a reload will load onto Timer T*x* time-base counter.

Measurement unit: T*x*_clk.

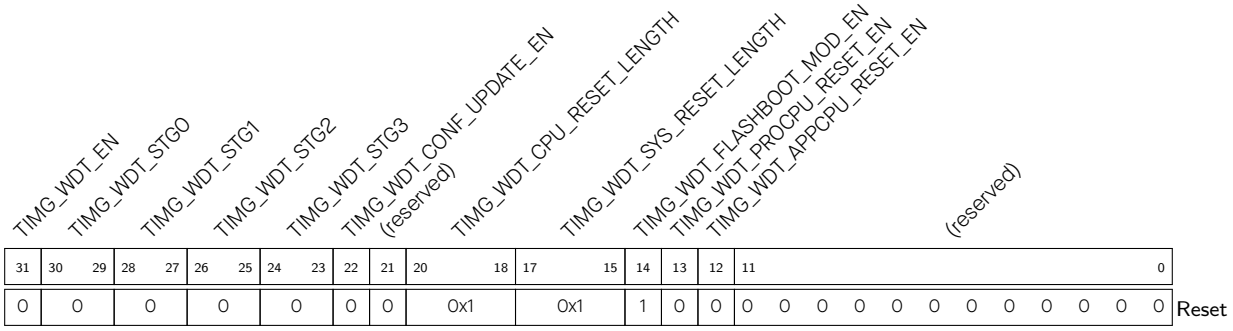
(R/W)

Register 13.9. TIMG_T*x*LOAD_REG (*x*: 0-1) (0x0020+0x24x*)**



TIMG_T*x*_LOAD Write any value to trigger Timer T*x* time-base counter reload. (WT)

Register 13.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN Configures whether to mask the APP CPU reset generated by MWDT. Valid only when write protection is disabled.

0: Mask

1: Unmask

(R/W)

TIMG_WDT_PROCPU_RESET_EN Configures whether to mask the PRO CPU reset generated by MWDT. Valid only when write protection is disabled.

0: Mask

1: Unmask

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN Configures whether to enable flash boot protection.

0: Disable

1: Enable

(R/W)

TIMG_WDT_SYS_RESET_LENGTH Configures the system reset signal length. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

0: 100 ns

1: 200 ns

2: 300 ns

3: 400 ns

4: 500 ns

5: 800 ns

6: 1.6 μs

7: 3.2 μs

(R/W)

Continued on the next page...

Register 13.10. TIMG_WDTCONFIG0_REG (0x0048)

Continued from the previous page...

TIMG_WDT_CPU_RESET_LENGTH Configures the CPU reset signal length. Valid only when write protection is disabled.
Measurement unit: mwdt_clk.

0: 100 ns
1: 200 ns
2: 300 ns
3: 400 ns
4: 500 ns
5: 800 ns
6: 1.6 μs
7: 3.2 μs

(R/W)

TIMG_WDT_CONF_UPDATE_EN Configures to update the WDT configuration registers.
0: No effect
1: Update
(WT)

TIMG_WDT_STG3 Configures the timeout action of stage 3. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_STG2 Configures the timeout action of stage 2. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_STG1 Configures the timeout action of stage 1. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

Continued on the next page...

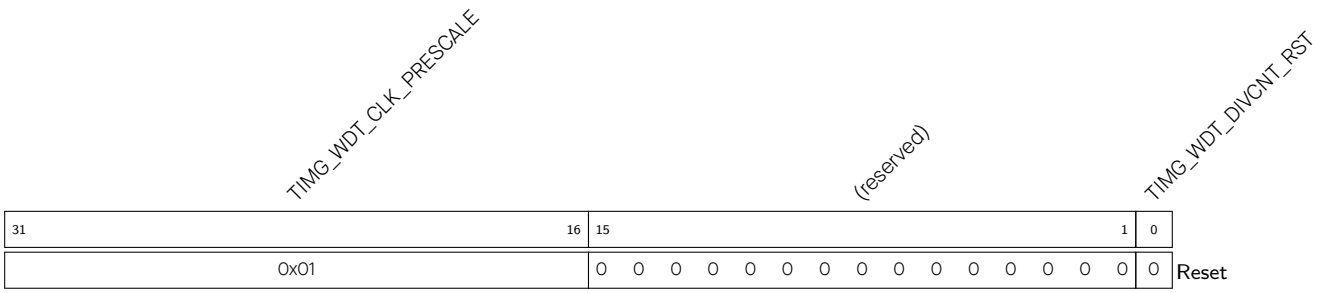
Register 13.10. TIMG_WDTCONFIG0_REG (0x0048)

Continued from the previous page...

TIMG_WDT_STG0 Configures the timeout action of stage 0. Valid only when write protection is disabled.
0: No effect
1: Interrupt
2: Reset CPU
3: Reset system
(R/W)

TIMG_WDT_EN Configures whether or not to enable the MWDT. Valid only when write protection is disabled.
0: Disable
1: Enable
(R/W)

Register 13.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST Configures whether to reset WDT's clock divider counter.

0: No effect

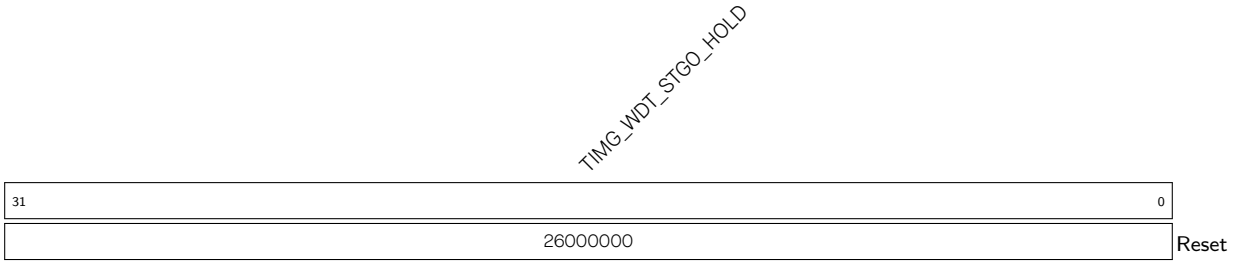
1: Reset

(WT)

TIMG_WDT_CLK_PRESCALE Configures MWDT clock prescaler value. Valid only when write protection is disabled.

MWDT clock period = 12.5 ns *TIMG_WDT_CLK_PRESCALE. (R/W)

Register 13.12. TIMG_WDTCONFIG2_REG (0x0050)

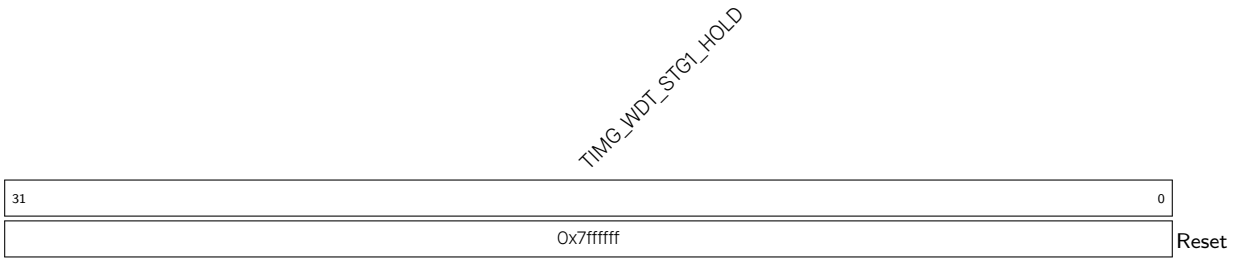


TIMG_WDT_STG0_HOLD Configures the stage 0 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 13.13. TIMG_WDTCONFIG3_REG (0x0054)

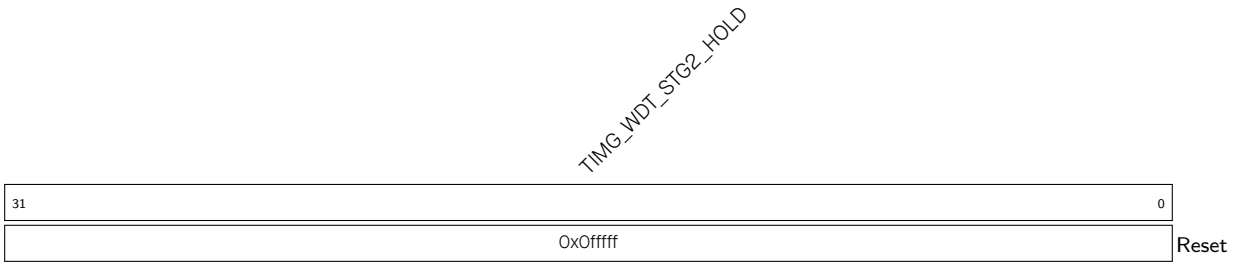


TIMG_WDT_STG1_HOLD Configures the stage 1 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 13.14. TIMG_WDTCONFIG4_REG (0x0058)

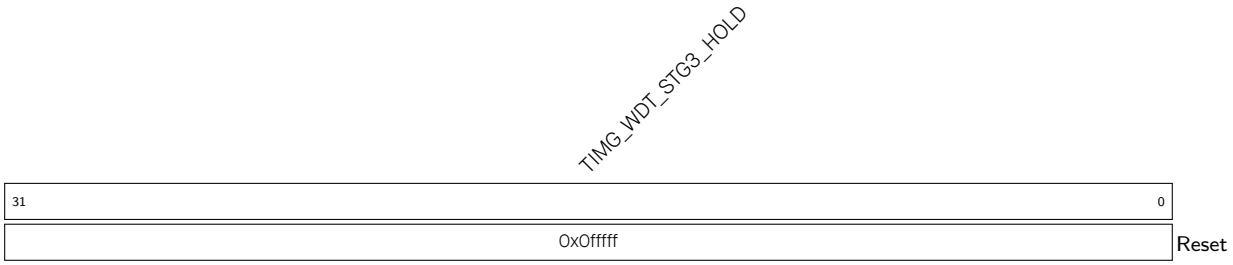


TIMG_WDT_STG2_HOLD Configures the stage 2 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

(R/W)

Register 13.15. TIMG_WDTCONFIG5_REG (0x005C)

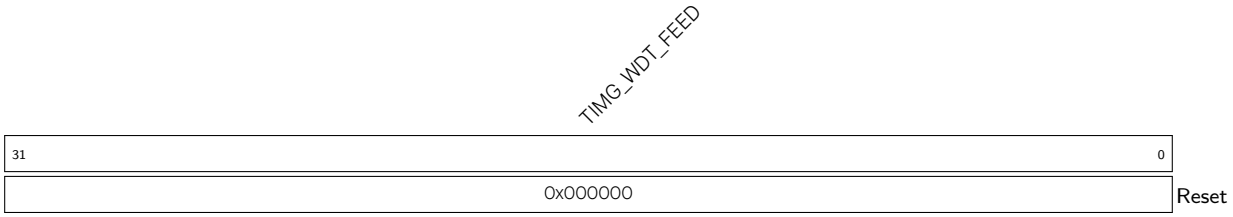


TIMG_WDT_STG3_HOLD Configures the stage 3 timeout value. Valid only when write protection is disabled.

Measurement unit: mwdt_clk.

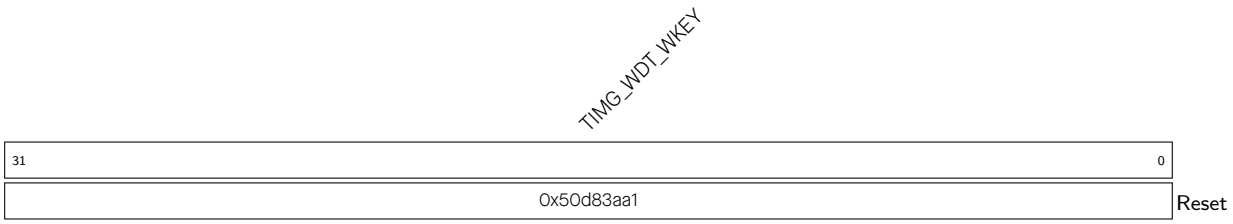
(R/W)

Register 13.16. TIMG_WDTFEED_REG (0x0060)



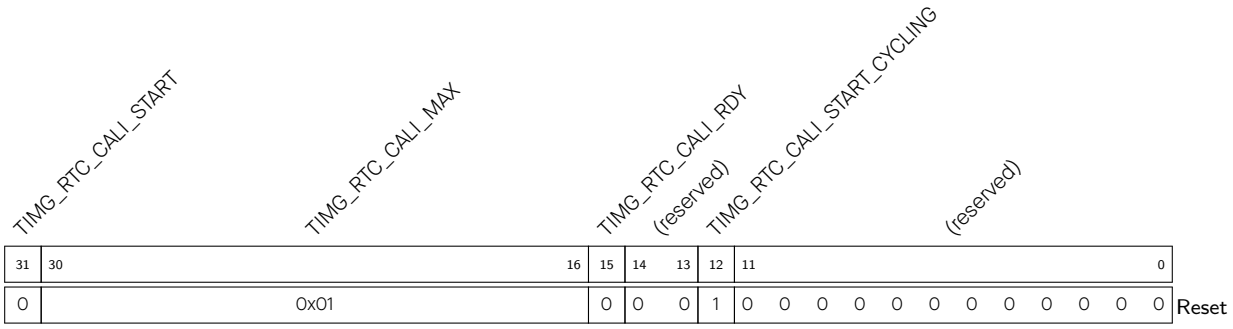
TIMG_WDT_FEED Write any value to feed the MWDT. Valid only when write protection is disabled. (WT)

Register 13.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY Configures a different value than its reset value to enable write protection. (R/W)

Register 13.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING Configures the frequency calculation mode.

0: One-shot frequency calculation

1: Periodic frequency calculation.

(R/W)

TIMG_RTC_CALI_RDY Represents whether one-shot frequency calculation is done.

0: Not done

1: Done

(RO)

TIMG_RTC_CALI_MAX Configures the time to calculate RTC slow clock's frequency.

Measurement unit: XTAL_CLK.

(R/W)

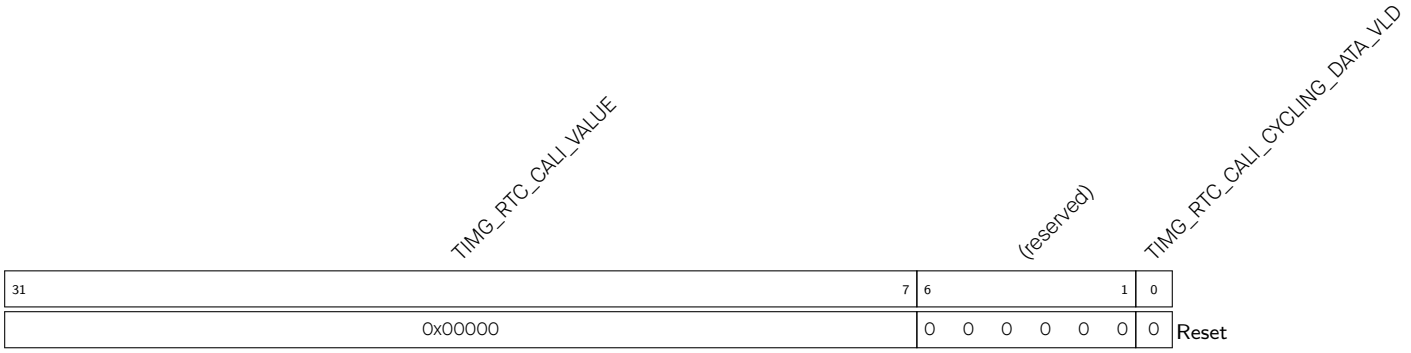
TIMG_RTC_CALI_START Configures whether to enable one-shot frequency calculation.

0: Disable

1: Enable

(R/W)

Register 13.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD Represents whether periodic frequency calculation is done.

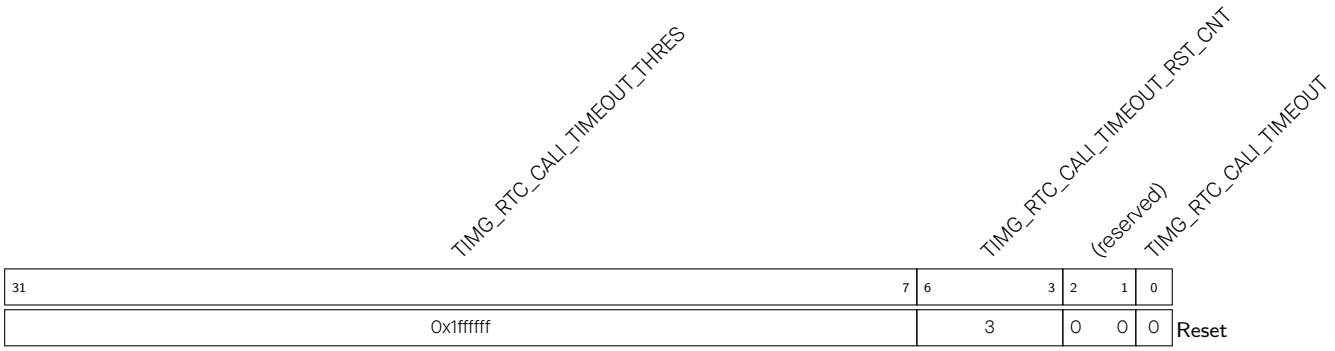
0: Not done

1: Done

(RO)

TIMG_RTC_CALI_VALUE Represents the value countered by XTAL_CLK when one-shot or periodic frequency calculation is done. It is used to calculate RTC slow clock's frequency. (RO)

Register 13.20. TIMG_RTCCALICFG2_REG (0x0080)



TIMG_RTC_CALI_TIMEOUT Represents whether RTC frequency calculation is timeout.

0: No timeout

1: Timeout

(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT Configures the cycles that reset frequency calculation timeout.

Measurement unit: XTAL_CLK.

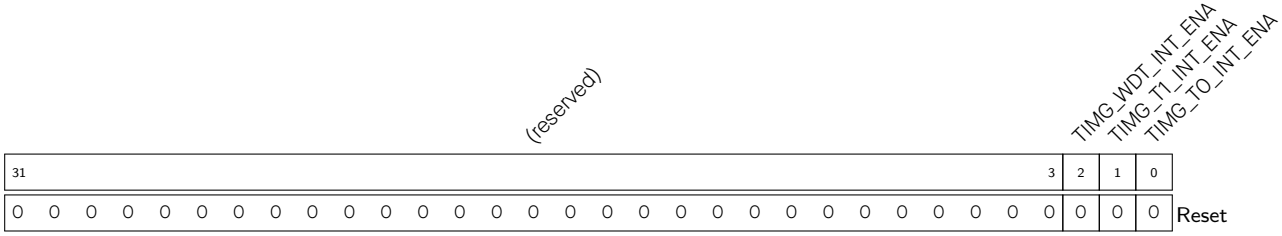
(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES Configures the threshold value for the RTC frequency calculation timer. If the timer's value exceeds this threshold, a timeout is triggered.

Measurement unit: XTAL_CLK.

(R/W)

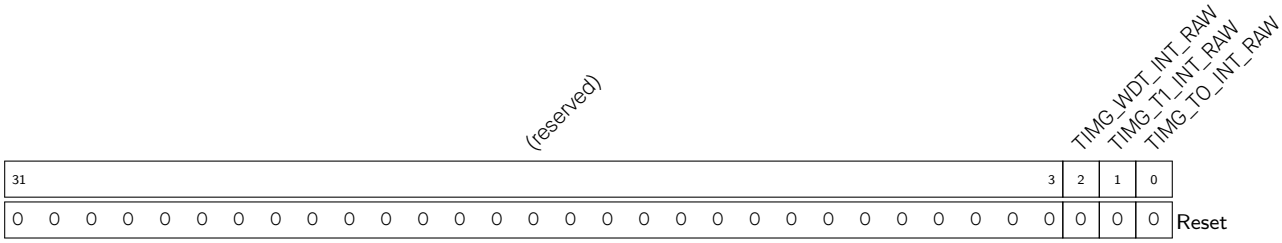
Register 13.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_T*x*_INT_ENA (*x*: 0-1) Write 1 to enable the TIMG_T*x*_INT interrupt. (R/W)

TIMG_WDT_INT_ENA Write 1 to enable the TIMG_WDT_INT interrupt. (R/W)

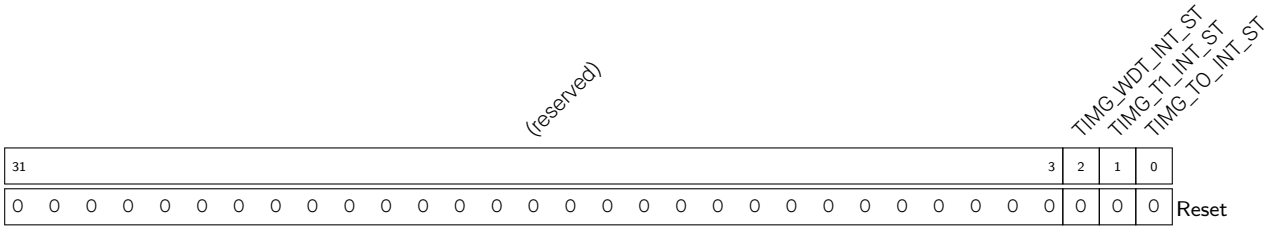
Register 13.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T*x*_INT_RAW (*x*: 0-1) The raw interrupt status bit of the TIMG_T*x*_INT interrupt. (R/SS/WTC)

TIMG_WDT_INT_RAW The raw interrupt status bit of the TIMG_WDT_INT interrupt. (R/SS/WTC)

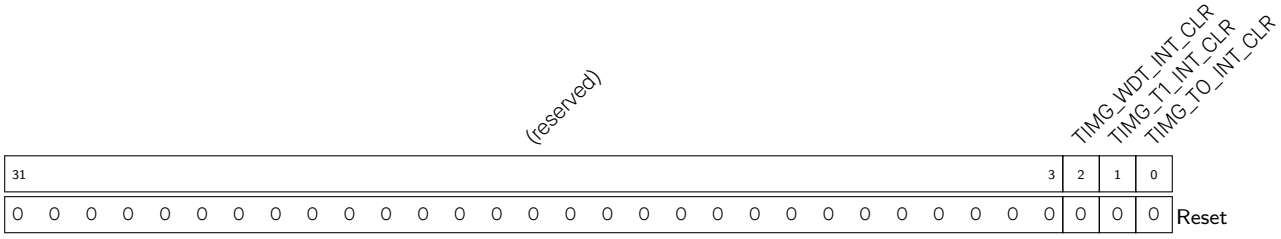
Register 13.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T*x*_INT_ST (*x*: 0-1) The masked interrupt status bit of the TIMG_T*x*_INT interrupt. (RO)

TIMG_WDT_INT_ST The masked interrupt status bit of the TIMG_WDT_INT interrupt. (RO)

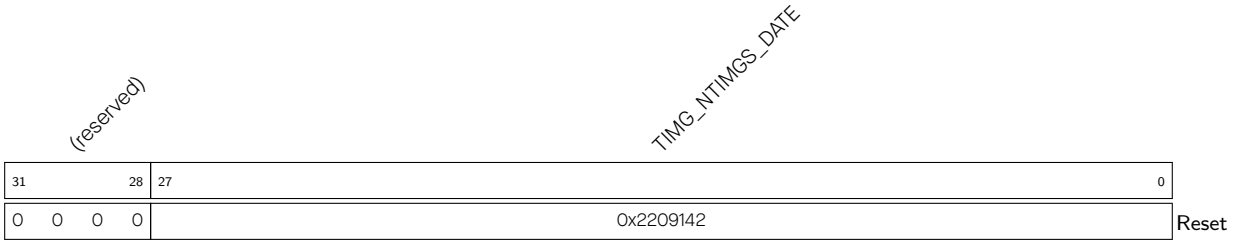
Register 13.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T*x*_INT_CLR (*x*: 0-1) Write 1 to clear the TIMG_T*x*_INT interrupt. (WT)

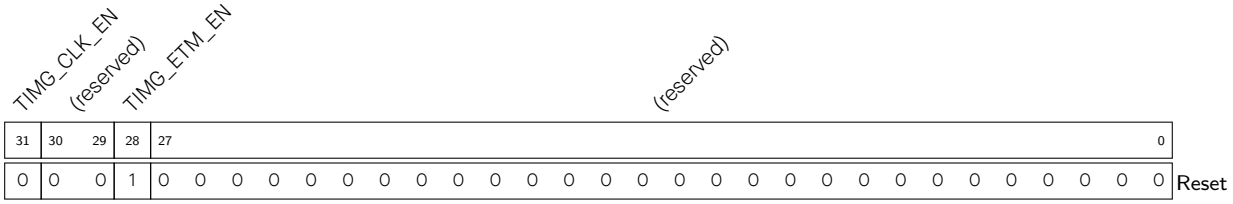
TIMG_WDT_INT_CLR Write 1 to clear the TIMG_WDT_INT interrupt. (WT)

Register 13.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE Version control register (R/W)

Register 13.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN Configures whether to enable timer's ETM task and event.

0: Disable

1: Enable

(R/W)

TIMG_CLK_EN Configures whether to enable gate clock signal for registers.

0: Force clock on for registers

1: Support clock only when registers are read or written to by software.

(R/W)

Chapter 14

Watchdog Timers (WDT)

14.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g., is stuck in a software loop or in overdue events) will fail to feed the watchdog, thus 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 14.1-1, ESP32-P4 contains three digital watchdog timers: one in each of the two timer groups in Chapter 13 *Timer Group (TIMG)* (called Main System Watchdog Timers, or MWDT) and one in the LP system (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages, and each stage can be programmed to take one action upon timeout, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, HP CPU reset, and HP core reset, while RWDT supports four timeout actions: interrupt, HP CPU reset, HP core reset, and system reset (see details in Section 14.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

In SPI Boot mode, RWDT and the MWDT in timer group 0 are enabled automatically in order to detect errors that may occur during the flash boot process and facilitate recovery.

ESP32-P4 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

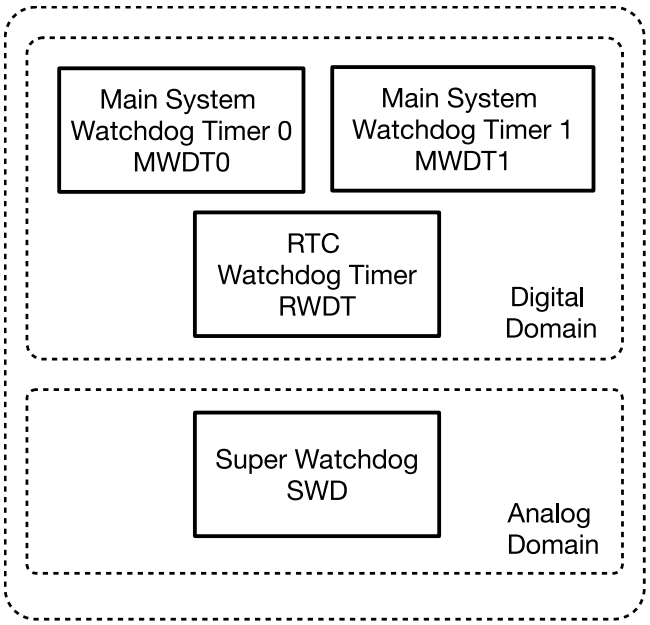


Figure 14.1-1. Watchdog Timers Overview

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)*, and the RWDT and SWD register descriptions are detailed in Section 14.5 *Register Summary*.

14.2 Digital Watchdog Timers

14.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWDT: interrupt, 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

14.2.2 Functional Description

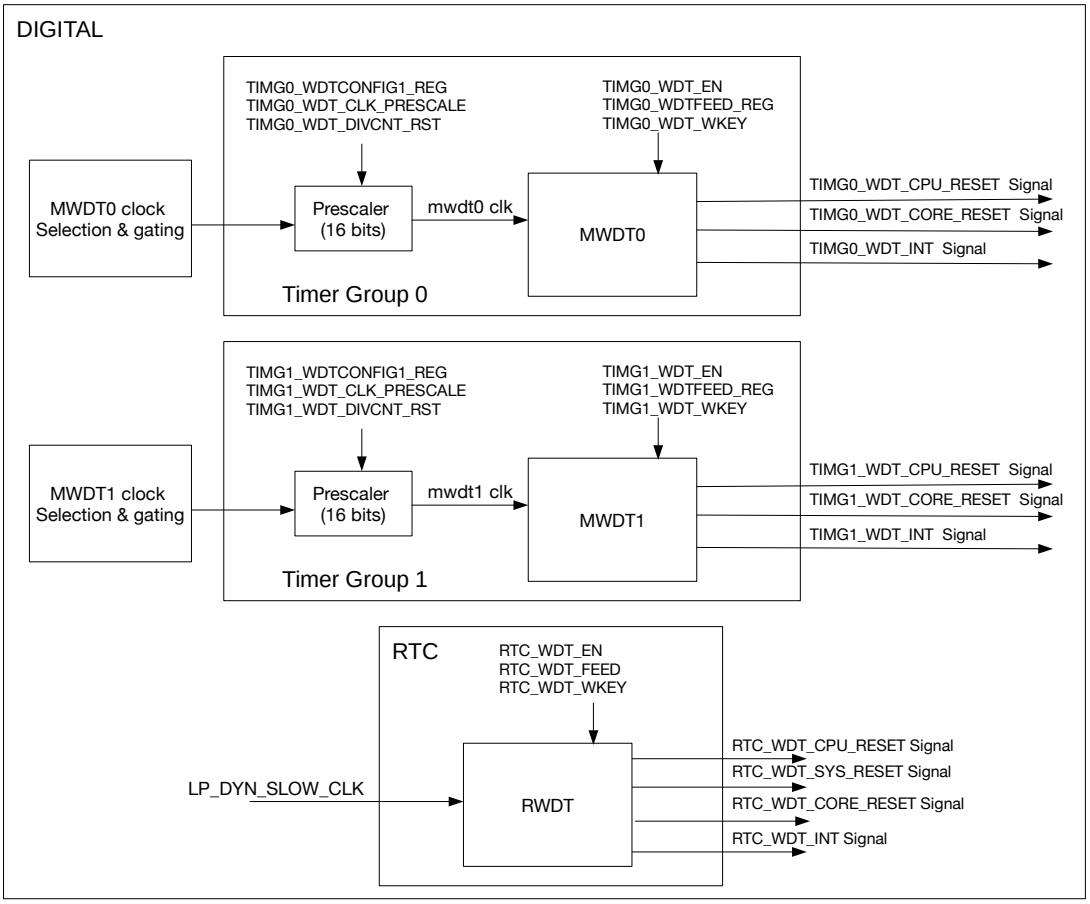


Figure 14.2-1. Digital Watchdog Timers in ESP32-P4

Figure 14.2-1 shows the three watchdog timers in ESP32-P4 digital systems.

14.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

Take MWDT0 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 8.2-1 of Chapter 8.
- 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 8

Reset and Clock).

MWDT and RWDT are enabled by setting the TIMG_WDT_EN and RTC_WDT_EN fields respectively. When enabled, the 32-bit counters of the watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e., timeout of the current stage). When this occurs, the current counter value is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to TIMG_WDTFEED_REG for MDWT and by writing 1 to RTC_WDT_FEED for RWDT.

14.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout 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 14.2-1. 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.

14.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g., due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write

protection mechanism that 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.

14.2.2.4 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.

14.3 Super Watchdog

Super watchdog (SWD) is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system (system reset) if required. SWD contains a watchdog circuit that needs to be fed for at least once during its timeout period, which is slightly less than one second. About 100 ms before watchdog timeout, it will also send out a WD_INTR signal as a request to remind the system to feed the watchdog.

If the system doesn't respond to SWD feed request and watchdog finally times out, SWD will generate a system level signal SWD_RSTB to reset whole digital circuits on the chip (system reset) .

The source of the clock for SWD is constant and can not be selected.

14.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD is about to time out
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

14.3.2 Super Watchdog Controller

14.3.2.1 Structure

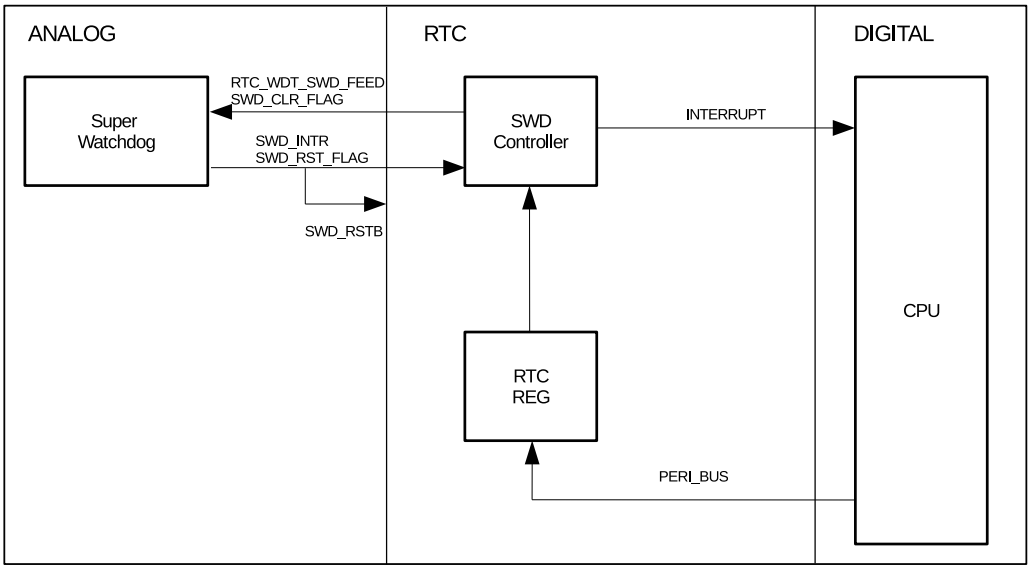


Figure 14.3-1. Super Watchdog Controller Structure

14.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU.
- Main CPU can feed SWD directly by setting RTC_WDT_SWD_FEED.
- When trying to feed SWD, CPU needs to disable SWD controller's write protection by writing 0x50D83AA1 to RTC_WDT_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_WDT_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU.

After reset:

- Check 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.

14.4 Interrupts

For watchdog timer interrupts, please refer to Section 13.3.7 *Interrupts* in Chapter 13 *Timer Group (TIMG)*.

14.5 Register Summary

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

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

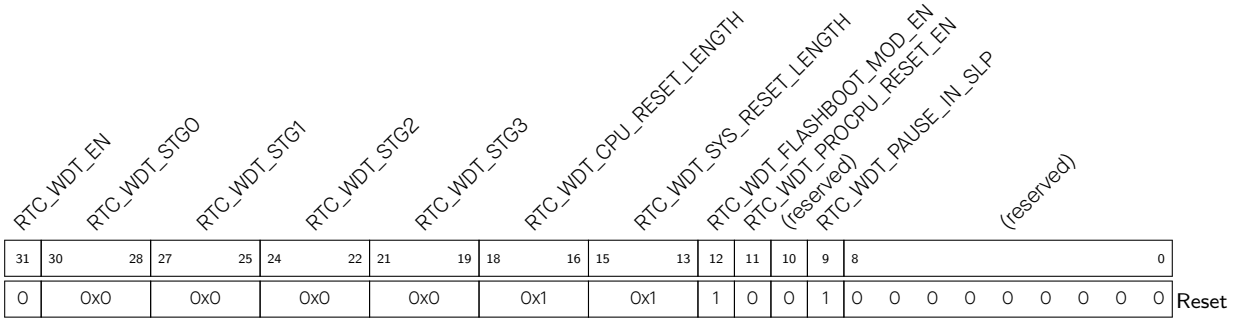
Name	Description	Address	Access
configuration register			
RTC_WDT_CONFIG0_REG	Configure the RWDT operation	0x0000	R/W
RTC_WDT_CONFIG1_REG	Configure the RWDT timeout time of stage0	0x0004	R/W
RTC_WDT_CONFIG2_REG	Configure the RWDT timeout time of stage1	0x0008	R/W
RTC_WDT_CONFIG3_REG	Configure the RWDT timeout time of stage2	0x000C	R/W
RTC_WDT_CONFIG4_REG	Configure the RWDT timeout time of stage3	0x0010	R/W
RTC_WDT_FEED_REG	Configure the feed function of RWDT	0x0014	WT
RTC_WDT_WPROTECT_REG	Configure the lock function of RWDT	0x0018	R/W
RTC_WDT_SWD_CONFIG_REG	Configure the SWD operation	0x001C	varies
RTC_WDT_SWD_WPROTECT_REG	Configure the lock function of SWD	0x0020	R/W
RTC_WDT_INT_RAW_REG	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

14.6 Registers

MWDT registers are part of the timer submodule and are described in Section 13.5 *Register Summary* in Chapter 13 *Timer Group (TIMG)*.

The addresses of RWDT and SWD registers in this section are relative to RTC_WDT base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Register 14.1. 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 14.1. 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)

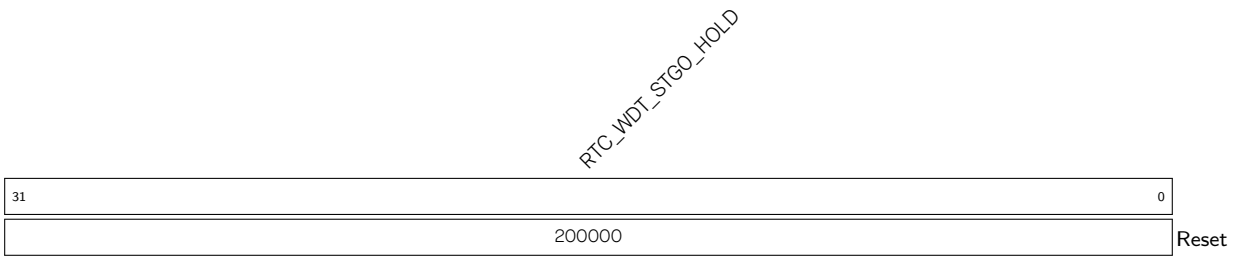
RTC_WDT_EN Configure whether or not enable RWDT.

0: Disable RWDT

1: Enable RWDT

(R/W)

Register 14.2. RTC_WDT_CONFIG1_REG (0x0004)

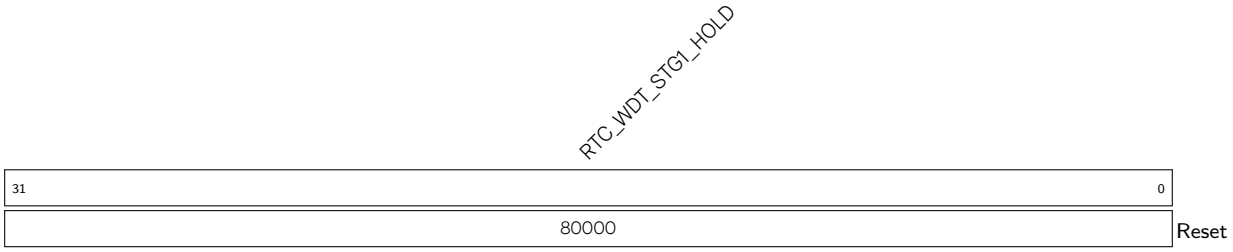


RTC_WDT_STG0_HOLD Configure the timeout time for stage0.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 14.3. RTC_WDT_CONFIG2_REG (0x0008)

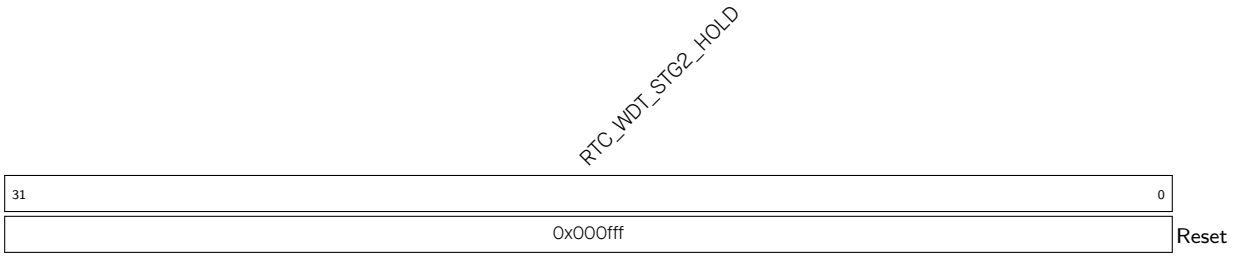


RTC_WDT_STG1_HOLD Configure the timeout time for stage1.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 14.4. RTC_WDT_CONFIG3_REG (0x000C)

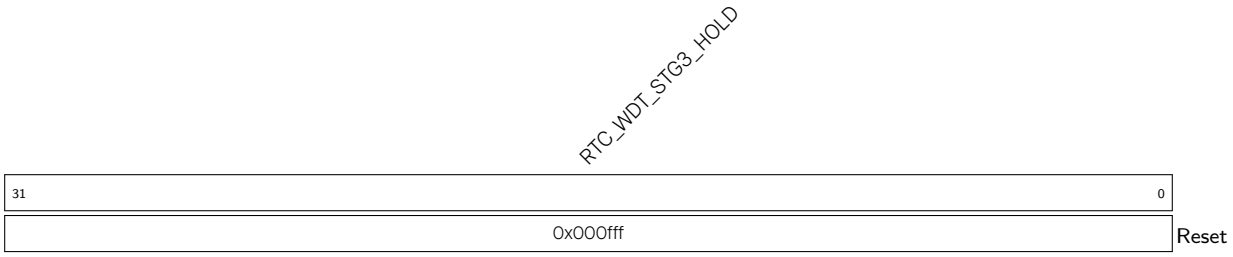


RTC_WDT_STG2_HOLD Configure the timeout time for stage2.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 14.5. RTC_WDT_CONFIG4_REG (0x0010)

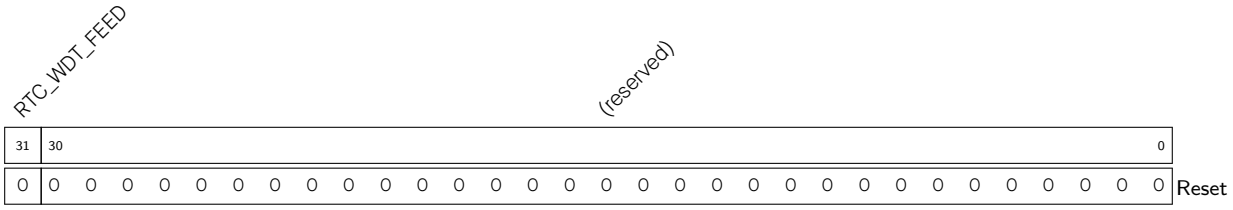


RTC_WDT_STG3_HOLD Configure the timeout time for stage3.

Measurement unit: LP_DYN_SLOW_CLK

(R/W)

Register 14.6. RTC_WDT_FEED_REG (0x0014)



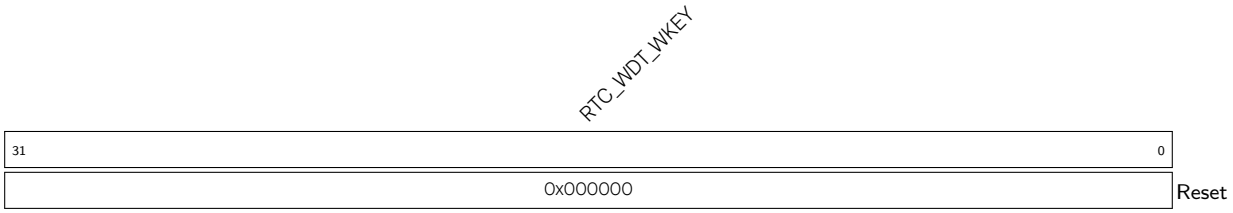
RTC_WDT_FEED Configure this bit to feed the RWDT.

0: Invalid

1: Feed RWDT

(WT)

Register 14.7. RTC_WDT_WPROTECT_REG (0x0018)



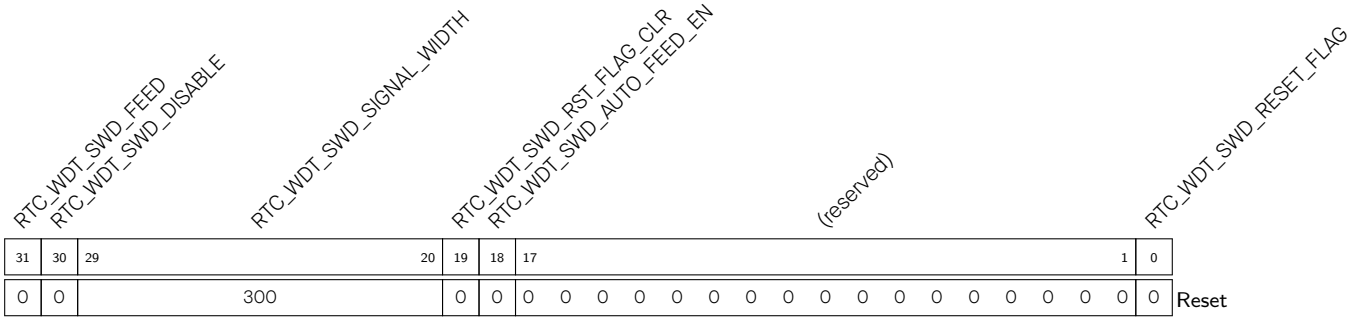
RTC_WDT_WKEY Configure this field to lock or unlock RWDT's configuration registers.

0x50D83AA1: unlock the RWDT configuration register

Others value: lock the RWDT configuration register which can't be modified by software.

(R/W)

Register 14.8. 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 Confgure 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)

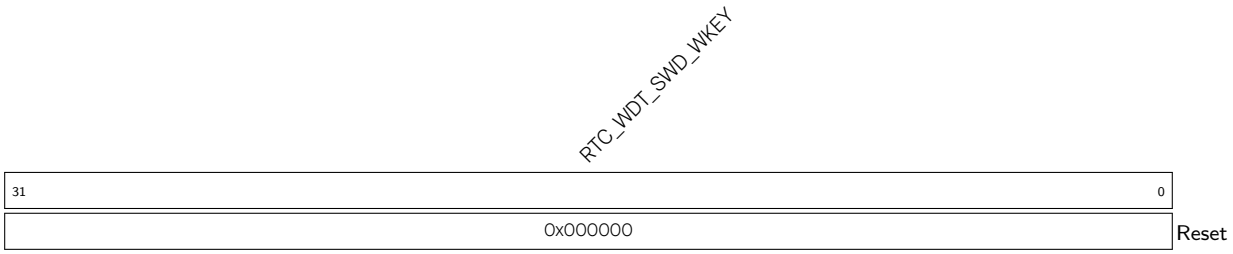
RTC_WDT_SWD_FEED Configure this bit to feed the SWD.

0: Invalid

1: Feed SWD

(WT)

Register 14.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)



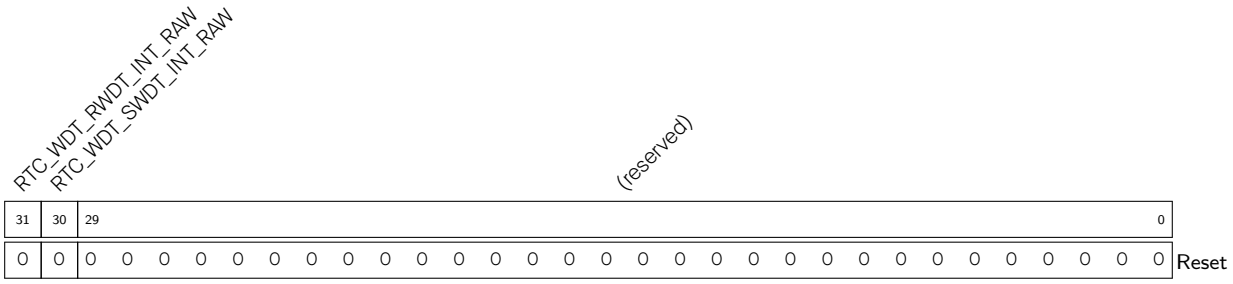
RTC_WDT_SWD_WKEY Configure this field to lock or unlock SWD's configuration registers.

0x50D83AA1: unlock the SWD configuration register.

Others value: lock the SWD configuration register which can't be modified by the software.

(R/W)

Register 14.10. 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)

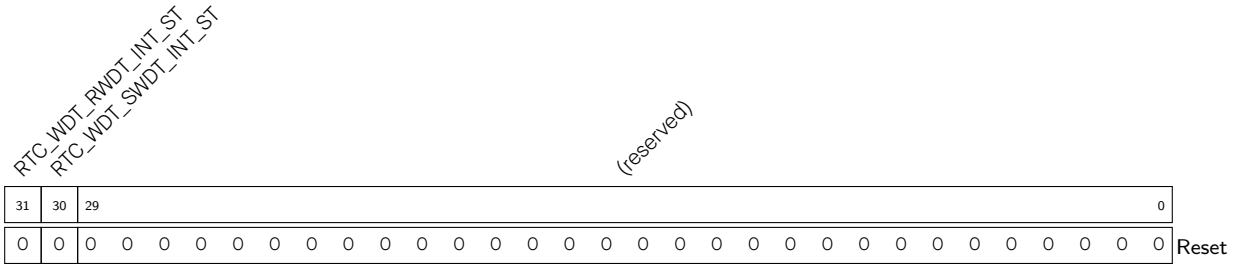
RTC_WDT_RWDT_INT_RAW Represents the RWDT whether or not generates timeout interrupt.

0: No

1: Yes

(R/WTC/SS)

Register 14.11. 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)

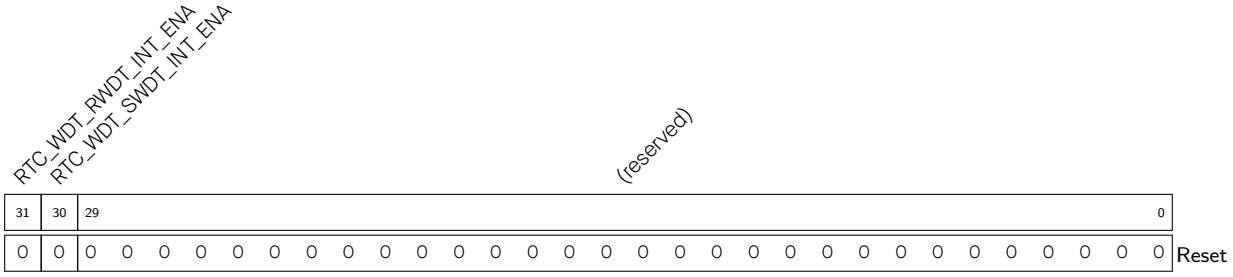
RTC_WDT_RWDT_INT_ST Represents the RWDT whether or not generates and sends timeout interrupt to CPU.

0: No

1: Yes

(RO)

Register 14.12. 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)

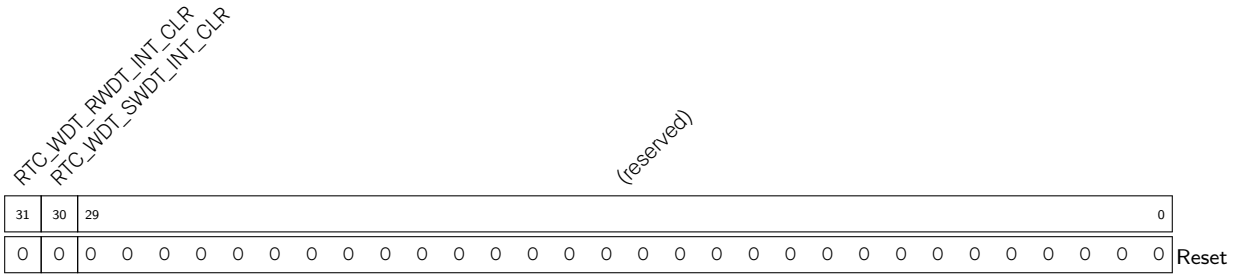
RTC_WDT_RWDT_INT_ENA Configure whether or not to enable the RWDT to send timeout interrupt.

0: Disable

1: Enable

(R/W)

Register 14.13. 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)

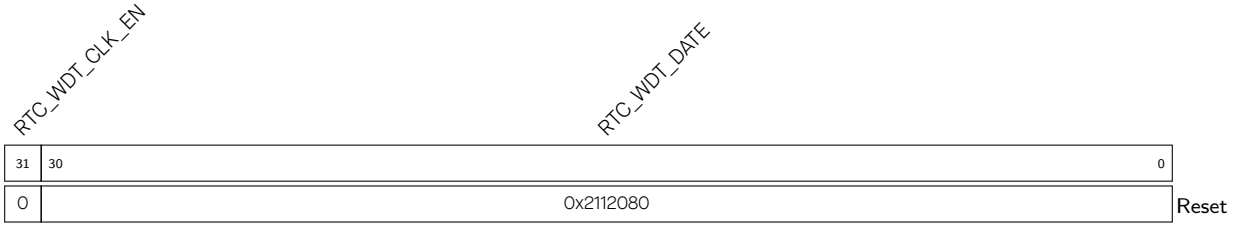
RTC_WDT_RWDT_INT_CLR Configure whether to clear the timeout interrupt signal sent by RWDT to CPU.

0: No

1: Yes

(WT)

Register 14.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE Version control register. (R/W)

RTC_WDT_CLK_EN Reserved. (R/W)

Chapter 15

RTC Timer

15.1 Introduction

RTC Timer is an important module for implementing low power management of ESP32-P4. Based on a 48-bit readable counter, RTC Timer is mainly used as a system timer in low power mode when the timer peripheral in the HP system is unavailable. It also allows for configuring timer interrupts and logging the time when specific events happen in the system. The supported events are described in the 15.3 Functional Description.

15.2 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.

15.3 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 15.3-1. 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 15.3-2. 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.

15.4 Event Task Matrix Feature

The RTC Timer on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows RTC Timer's ETM tasks to be triggered by any peripherals' ETM events, or RTC Timer's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to RTC Timer. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

RTC Timer doesn't receive any ETM tasks:

RTC Timer can generate the following ETM events:

- RTC_EVT_CMP: Indicate that the RTC Timer reaches the target time 0 configured by RTC_TIMER_MAIN_TIMER_TAR_LOW0 and RTC_TIMER_MAIN_TIMER_TAR_HIGH0.
- RTC_EVT_OVF: Indicate the 48-bit counter overflow.

15.5 Interrupts

ESP32-P4's RTC Timer can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- LP_TIMER_REG_0_INTR
- LP_TIMER_REG_1_INTR

There are several internal interrupt sources from RTC Timer that can generate the above interrupt signal(s). The interrupt sources from RTC Timer are listed with their trigger conditions and the resulted interrupt signal(s) in Table 15.5-1.

Table 15.5-1. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 15.6 *Register Summary*.

15.6 Register Summary

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

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

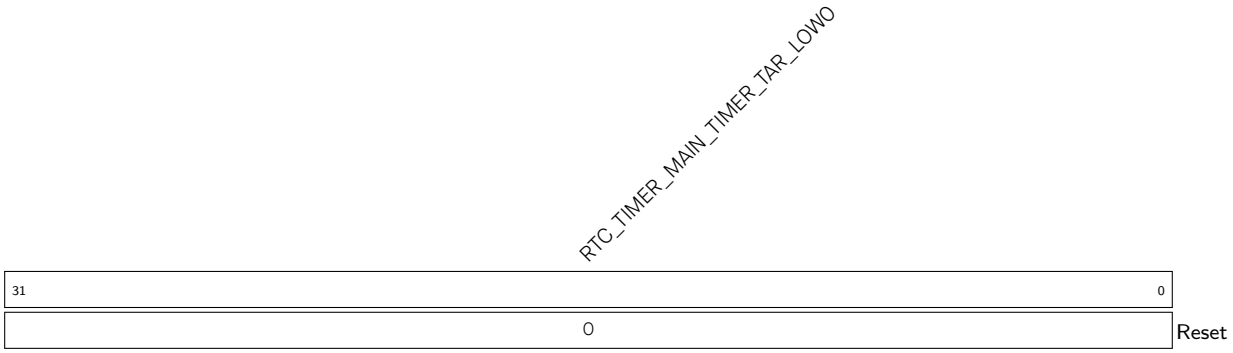
Name	Description	Address	Access
Configuration Registers			
RTC_TIMER_TAR0_LOW_REG	Configure the low 32 bits of the target time 0	0x0000	R/W
RTC_TIMER_TAR0_HIGH_REG	Configure the high 16 bits of the target time 0	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	Configure the low 32 bits of the target time 1	0x0008	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

15.7 Registers

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

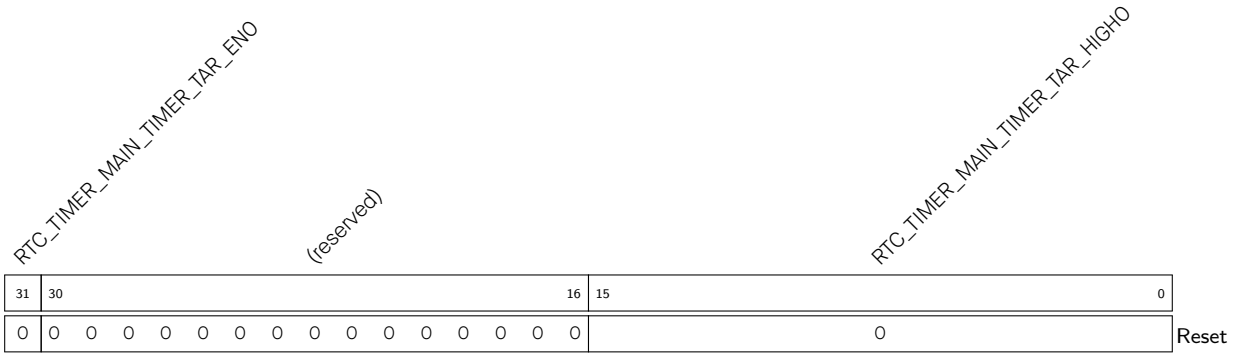
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 15.1. 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)

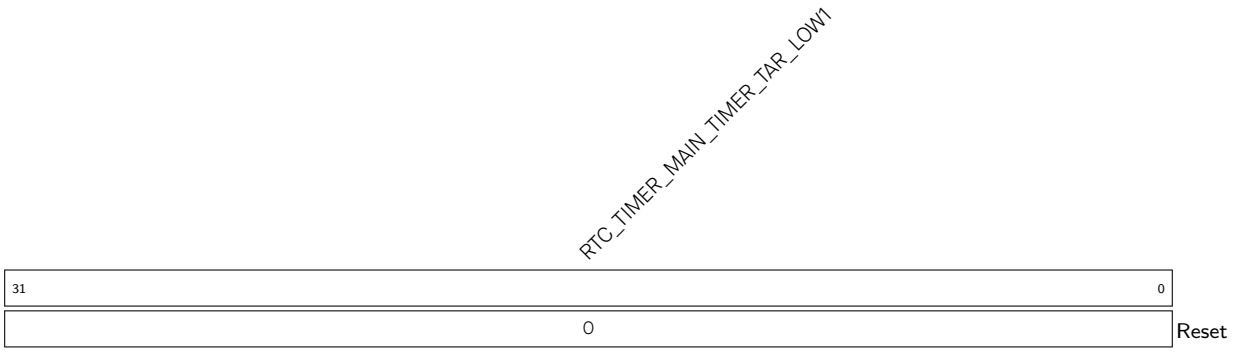
Register 15.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_HIGH0 Configure the high 16 bits of the target time 0. (R/W)

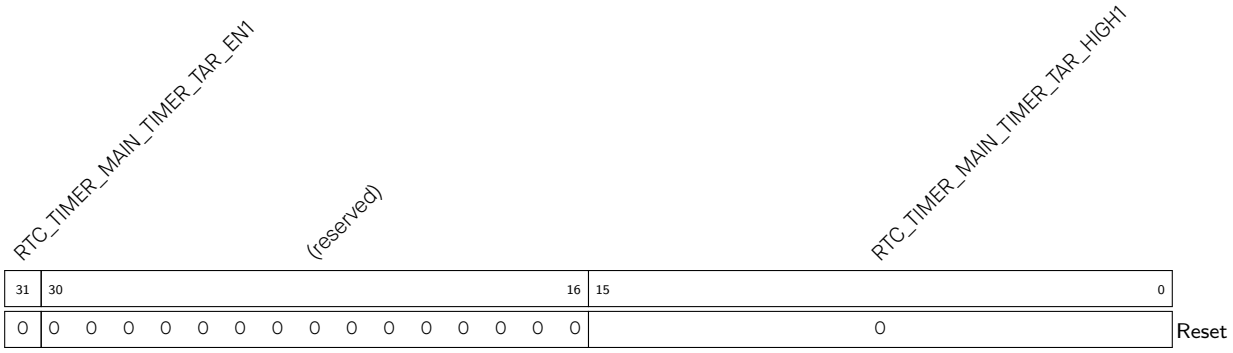
RTC_TIMER_MAIN_TIMER_TAR_EN0 Configure to enable the target time 0 of the RTC Timer. (WO)

Register 15.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 Configure the low 32 bits of the target time 1. (R/W)

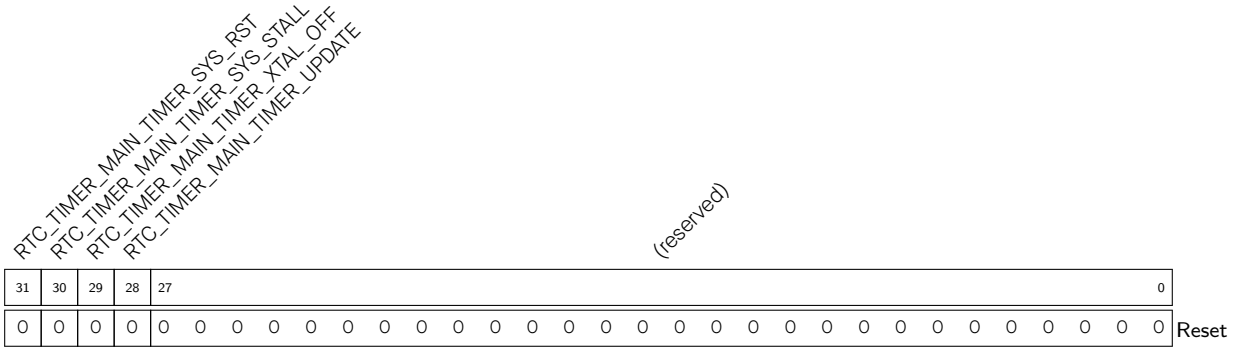
Register 15.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)



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 15.5. RTC_TIMER_UPDATE_REG (0x0010)



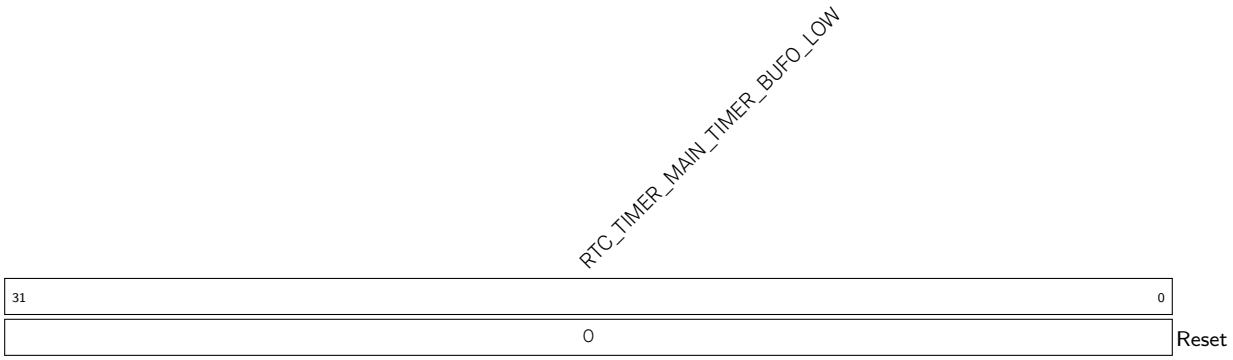
RTC_TIMER_MAIN_TIMER_UPDATE Configure to log the current time through software configuration. (WO)

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)

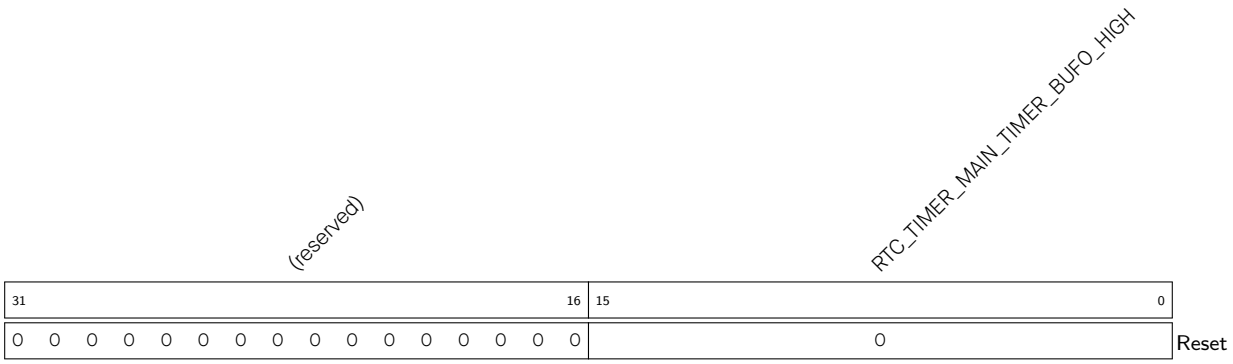
RTC_TIMER_MAIN_TIMER_SYS_RST Configure to enable RTC Timer to log the time of system reset. (R/W)

Register 15.6. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



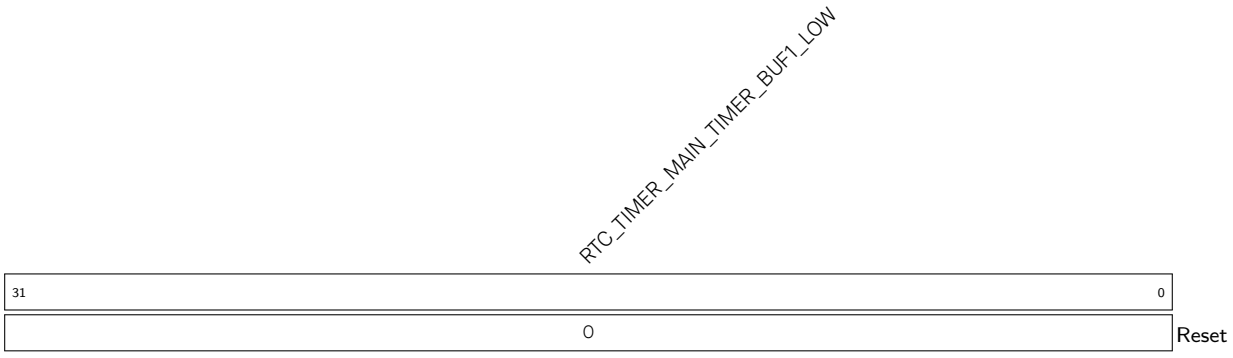
RTC_TIMER_MAIN_TIMER_BUF0_LOW Represent the low 32 bits of the cached value 0 in RTC Timer. (RO)

Register 15.7. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



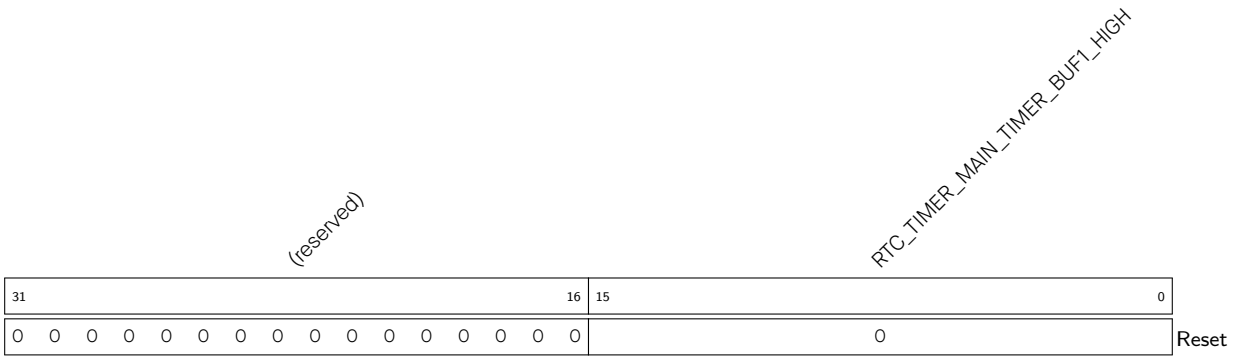
RTC_TIMER_MAIN_TIMER_BUF0_HIGH Represent the high 16 bits of the cached value 0 in RTC Timer. (RO)

Register 15.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



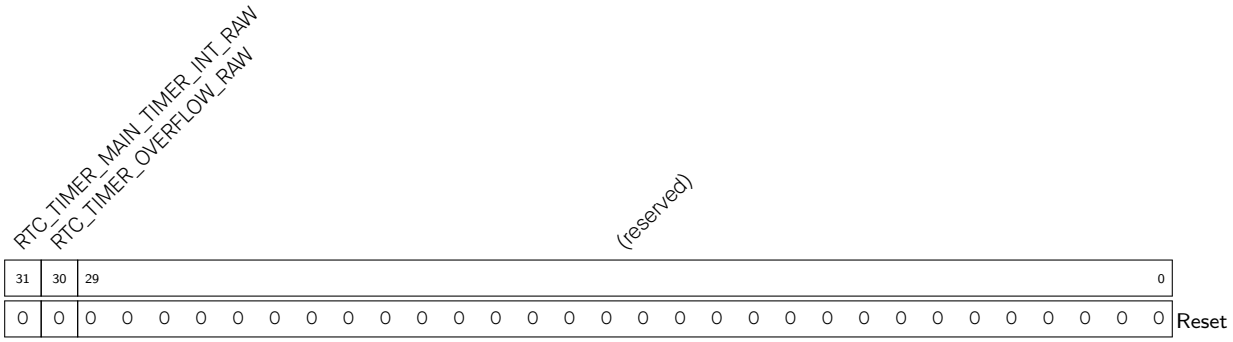
RTC_TIMER_MAIN_TIMER_BUF1_LOW Represent the low 32 bits of the cached value 1 in RTC Timer. (RO)

Register 15.9. 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)

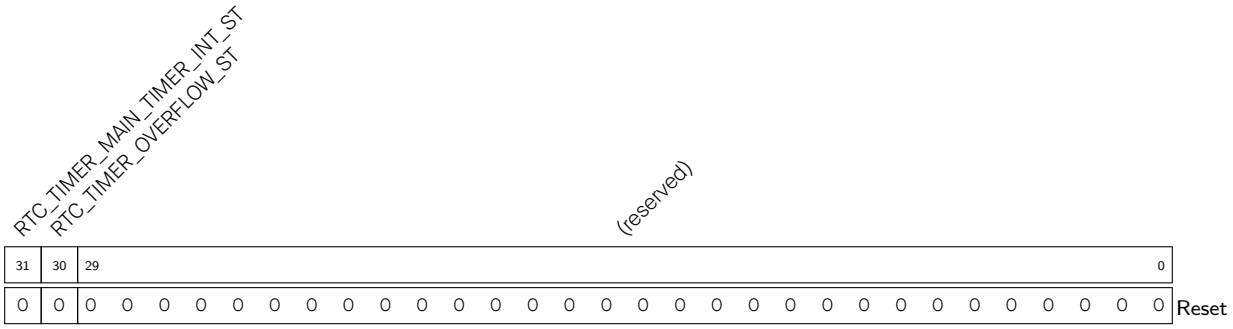
Register 15.10. RTC_TIMER_INT_RAW_REG (0x0028)



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)

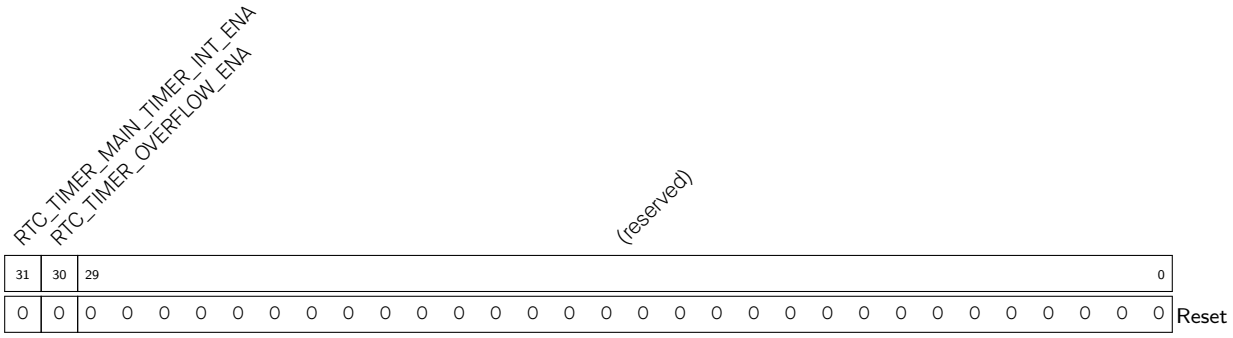
Register 15.11. RTC_TIMER_INT_ST_REG (0x002C)



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)

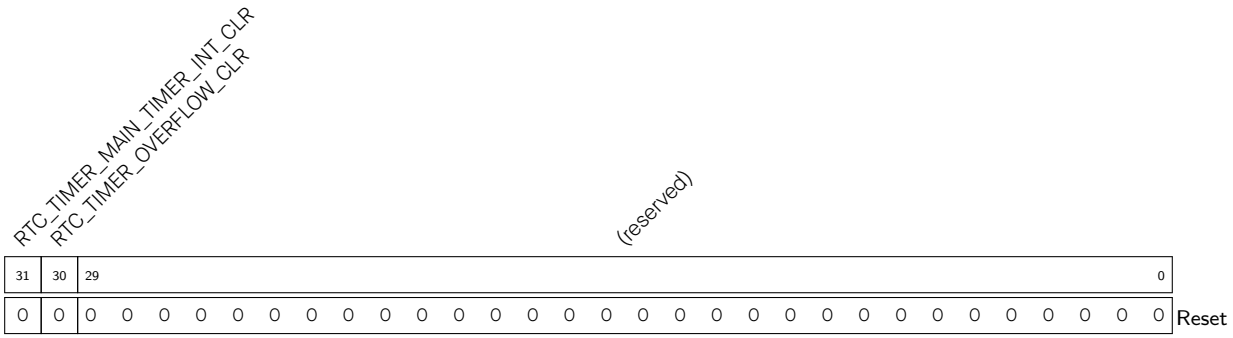
Register 15.12. RTC_TIMER_INT_ENA_REG (0x0030)



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)

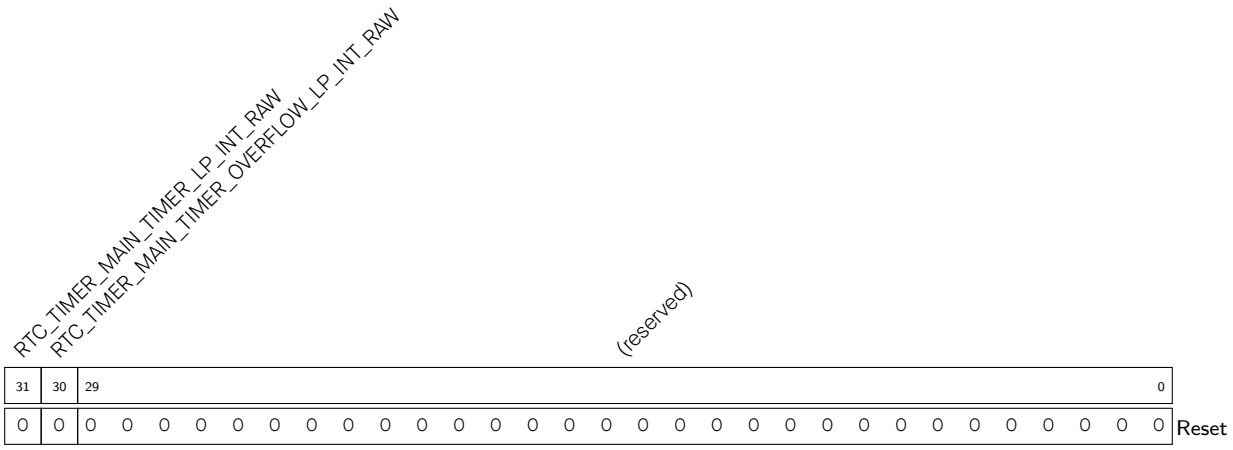
Register 15.13. RTC_TIMER_INT_CLR_REG (0x0034)



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)

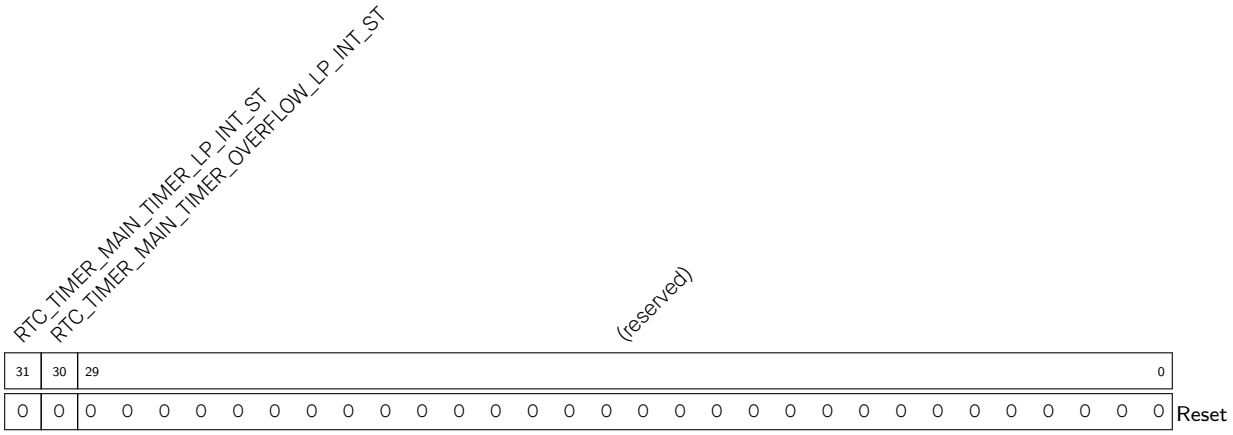
Register 15.14. RTC_TIMER_LP_INT_RAW_REG (0x0038)



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)

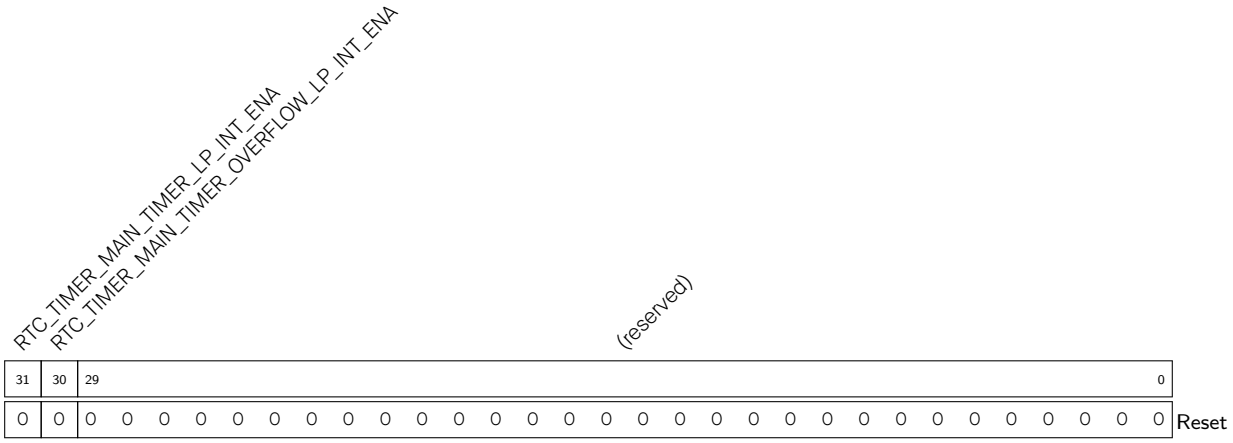
Register 15.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



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)

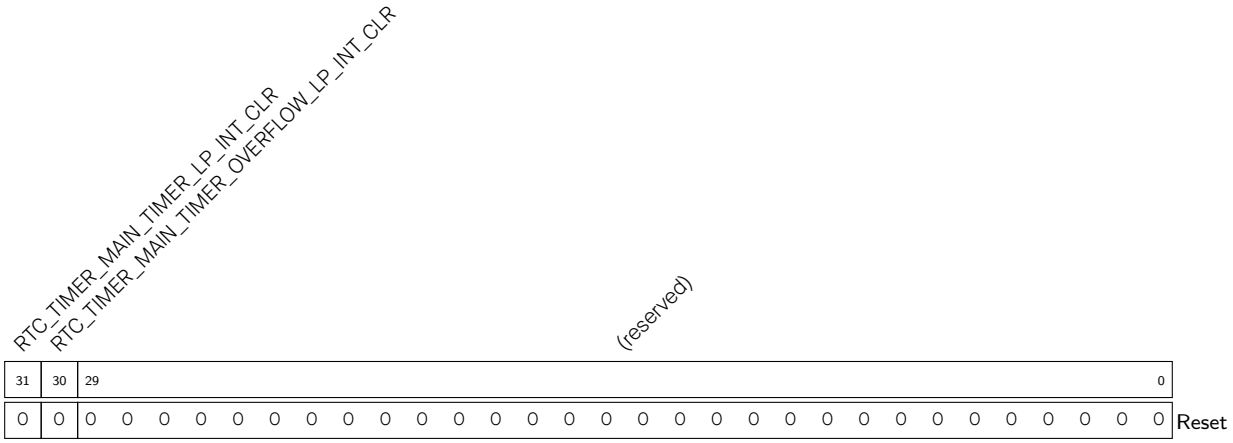
Register 15.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



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)

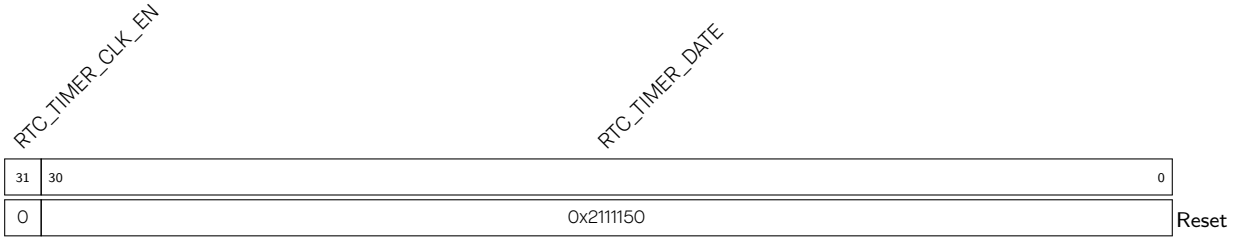
Register 15.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)



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 15.18. 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 16

Permission Control (PMS)

16.1 Overview

The permission control of ESP32-P4 consists of two parts: PMP (Physical Memory Protection) and APM (Access Permission Management).

PMP is located inside the HP CPUs and can control the HP CPUs' access to all address spaces. APM is an access permission management module located at the bus port, which can manage the permissions for the CPU (HP CPU0/1, LP CPU) to access the registers of all on-chip peripherals (see HP CPU PERI, HP PERI, and LP PERI in Table 16.1-1) and part of internal and external memory. The APM can also manage DMA masters like DW-GDMA, USB OTG 2.0, to access parts of internal and external memory.

The APM module compares the configurable address range or the fixed address range, the access permissions of each address range, and the information carried on the bus like master ID, security mode, access address, access type, etc, to determine whether the access is allowed. With APM, users can precisely control the access of all masters to the memory and peripheral registers.

The distribution of management areas by PMP and APM is shown in Table 16.1-1. For HP CPU0/1, the control relationship between PMP and APM is shown in Figure 16.1-1.

Table 16.1-1. Management Areas by PMP and APM

Master / Slave	HP CPU0	HP CPU1	LP CPU	DMA Masters[1]
HP ROM[2]	PMP	PMP	HP APM	DMA APM
HP TCM	PMP	PMP	N/A	N/A
HP L2MEM[2]	PMP	PMP	HP APM	DMA APM
EXT MEM[2]	PMP	PMP	HP APM	DMA APM
Direct Access HP ROM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access HP L2MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access EXT MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
HP CPU PERI[4]	PMP + HP APM	PMP + HP APM	HP APM	N/A
HP PERI[5]	PMP + HP APM	PMP + HP APM	HP APM	N/A
LP PERI[6]	PMP + LP APM	PMP + LP APM	LP APM	N/A
LP ROM	N/A	N/A	N/A	N/A
LP SRAM	PMP + LP APM	PMP + LP APM	N/A	N/A

[1] DMA masters that support DMA transfers, such as DW-GDMA, USB 2.0 OTG, MEM_MONITOR, etc.

[2]
- HP ROM (0x4FC0_0000 - 0x4FC1_FFFF)
- HP L2MEM (0x4FF0_0000 - 0x4FFB_FFFF)
- EXT MEM, including external flash (0x4000_0000 - 0x43FF_FFFF) and external RAM (0x4800_0000 - 0x4BFF_FFFF)

[3] "Direct access" means HP CPU0/1 accesses internal and external memory mapped into the following address spaces without going through the cache:
- HP ROM (0x8FC0_0000 - 0x8FC1_FFFF)
- HP L2MEM (0x8FF0_0000 - 0x8FFB_FFFF)
- EXT MEM, including external flash (0x8000_0000 - 0x83FF_FFFF) and external RAM (0x8800_0000 - 0x8BFF_FFFF)

[4] HP CPU peripheral registers (0x3FF0_0000 - 0x3FF1_FFFF)

[5] HP system peripheral registers (0x5000_0000 - 0x500F_FFFF)

[6] LP system peripheral registers (0x5011_0000 - 0x5012_FFFF)

[7] As can be seen from the table, APM includes HP APM, LP APM, and DMA APM, which will be described in the following sections.

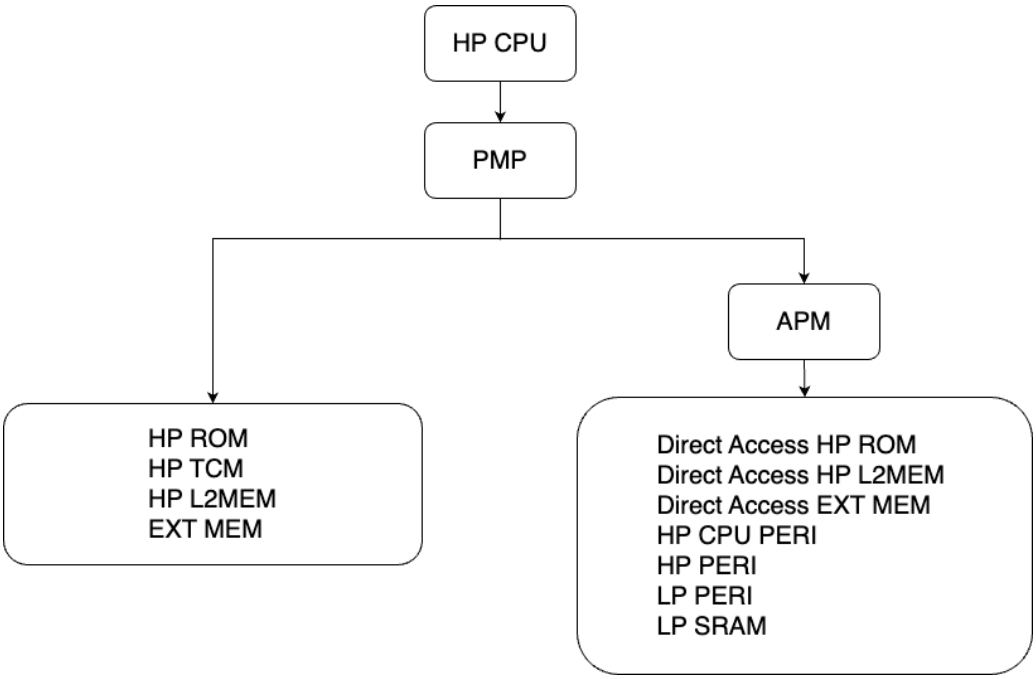


Figure 16.1-1. PMP-APM Management Relation

The diagram illustrates that when the HP CPUs access HP ROM, HP TCM, HP L2MEM, and EXT MEM, the access paths are solely managed by PMP. On the other hand, when the HP CPUs access the peripheral registers, HP ROM, HP L2MEM, and EXT MEM without going through the cache, the access paths are controlled by both PMP and APM. If the PMP check fails, the APM permission control will not be triggered.

PMP-related registers are located inside the HP CPUs and can be read or configured with special instructions. For how to configure PMP, please refer to Chapter 1 *High-Performance CPU [to be added later]* > *Physical Memory Protection*.

The following sections will provide a detailed introduction to the features, functions, and configurations of the APM module. When referring to HP CPU0/1, accessing HP ROM/HP L2MEM/EXT MEM always means direct access to HP ROM/HP L2MEM/EXT MEM.

16.2 Features

The APM module has the following features:

- Up to 32 configurable address ranges for each DMA master
- Access permission management for each CPU core to access internal memory, external memory, and peripheral registers
- Support for interrupts
- Support for exception information record

16.3 Functional Description

16.3.1 Architecture

The APM module contains three parts and five register modules as follows:

- **DMA APM**, containing register group HP_DMA_PMS_REG, responsible for:
  - configuring up to 32 address regions for DMA masters
  - managing the access permission to each address region for each DMA master
- **HP APM**, containing register groups HP_PERI_PMS_REG and LP2HP_PERI_PMS_REG, responsible for managing the access permissions to peripheral registers (HP CPU PERI, HP PERI), internal memory (HP ROM, HP L2MEM), and external memory (EXT MEM) in the HP system, specifically,
  - managing the access permissions for HP CPU0/1 in user mode to access all the above-mentioned slaves
  - managing the access permissions for HP CPU0/1 in machine mode to access all the above-mentioned slaves
  - managing the access permissions for LP CPU in machine mode to access all the above-mentioned slaves
- **LP APM**, containing register groups LP_PERI_PMS_REG and HP2LP_PERI_PMS_REG, responsible for managing the access permissions to peripheral registers (LP PERI) and internal memory (LP SRAM) in the LP system, specifically,
  - managing the access permissions for HP CPU0/1 in user mode to access LP PERI and LP SRAM
  - managing the access permissions for HP CPU0/1 in machine mode to access LP PERI and LP SRAM
  - managing the access permissions for LP CPU in machine mode to access LP PERI

Figure 16.3-1 shows the access paths managed by APM and the corresponding register groups.

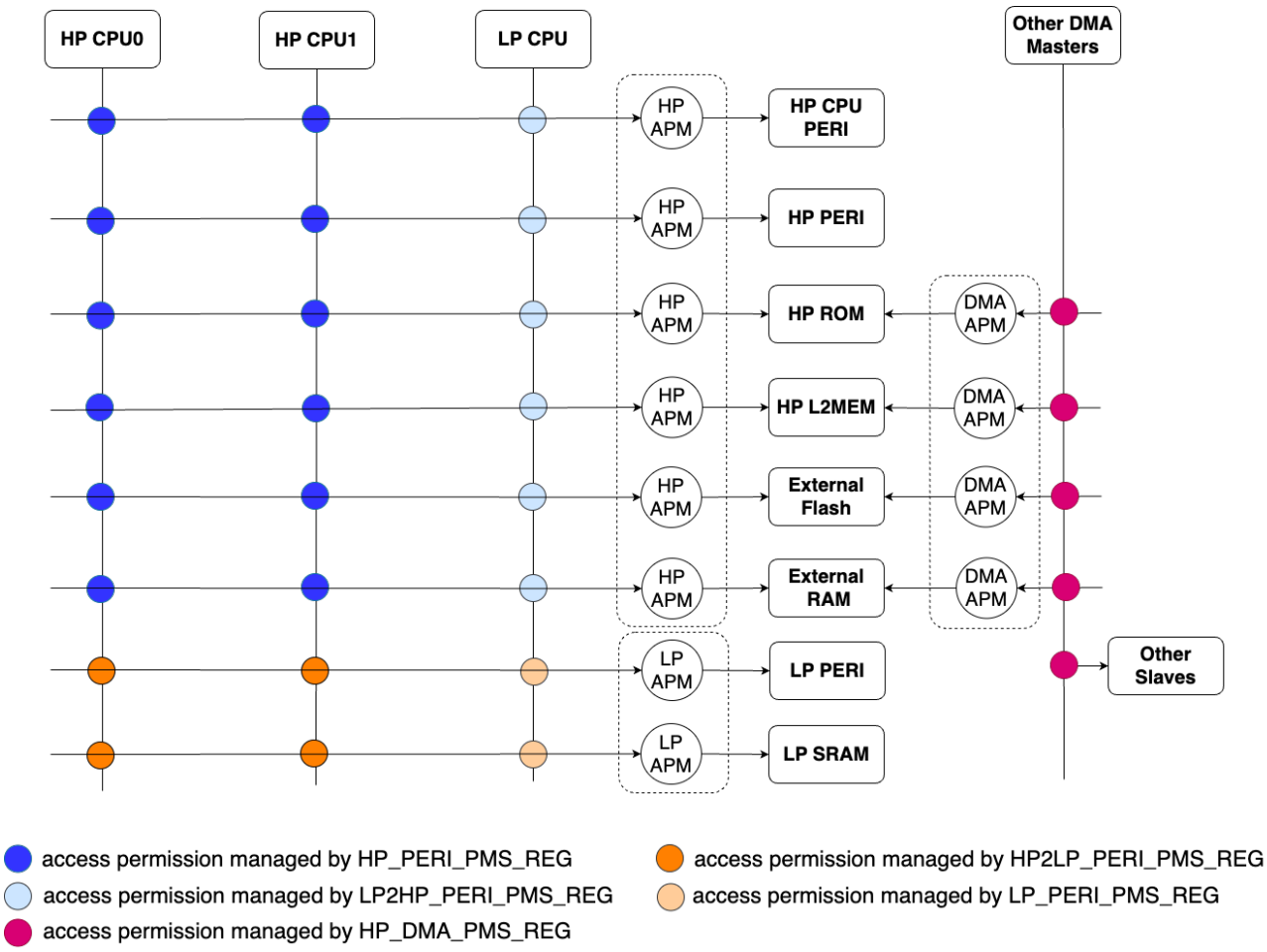


Figure 16.3-1. APM Architecture

Note:

- For register group names "HP2LP" and "LP2HP", the "HP" refers to HP CPU0/1, and "LP" refers to LP CPU.
- From Table 16.1-1, it can be seen that HP CPU and LP CPU access different address spaces of HP ROM, HP L2MEM, external flash, and external RAM. For HP CPU0/1, HP APM manages their direct access to internal and external memory without going through the cache.
- LP_PERI_PMS_REG includes two additional configurable address ranges for HP APM and LP APM to manage the access permissions to peripheral registers, including HP CPU PERI, HP PERI, and LP PERI.

16.3.2 Address Ranges and Permissions Management

APM manages address ranges, which can be fixed or configurable. DMA APM can manage up to 32 configurable address ranges. In addition to the fixed address ranges, HP APM and LP APM can also manage in total two configurable address ranges.

16.3.2.1 Address Ranges Managed by HP APM and LP APM

In the access paths controlled by HP APM and LP APM, the internal memory and external memory use fixed address ranges for access control, while peripheral registers not only support fixed address ranges permission control but also support two configurable address ranges.

- Fixed address ranges
  - The fixed address ranges are the valid address spaces of peripheral registers, internal memory, and external memory. See the table notes under Table 16.1-1 for the address ranges of internal and external memory. Refer to Chapter 5 *System and Memory* > Table 5.3-2 for the address ranges of each peripheral.
  - To manage access to the fixed address ranges, please refer to Figure 16.3-1 for the access paths and their corresponding register groups.
- Configurable Address Range
  - APM can configure two address ranges with PMS_PERI_REGION*n*_LOW_REG and PMS_PERI_REGION*n*_HIGH_REG (*n* = 0 ~ 1) in LP_PERI_PMS_REG. These two configurable address ranges can be used by HP CPUs and LP CPU to access all on-chip peripheral registers (HP CPU PERI, HP PERI, LP PERI).
  - The access permissions are managed by PMS_PERI_REGION_PMS_REG in LP_PERI_PMS_REG.

When managing access permissions to peripheral registers, the permission management for the configurable address range should precede that of the fixed address range. Specifically:

- If the access address is within the configurable address range, then APM will check if this access is permitted to access the configurable address range.
  - If permitted, APM will continue to check if the access is permitted to the fixed address range.
  - If not permitted, the access will be denied and skip the permission check to the fixed address range.
- If the access address is outside the configurable address range, then it will skip the configurable address range check, but will continue to the fixed address range check.
- The management of configurable address ranges is a supplement to the management of fixed address ranges. For example, users can configure access permissions for a fixed address range, and deny access to two configurable ranges within the fixed range, resulting in a fixed address range containing two access-disabled address ranges.

16.3.2.2 Address Ranges Managed by DMA APM

The register group HP_DMA_PMS_REG provides 32 sets of registers (PMS_DMA_REGION*n*_LOW_REG ~ PMS_DMA_REGION*n*_HIGH_REG, *n* = 0 ~ 31) to configure address ranges for DMA masters to access. Note that since the address must be aligned to 4K bytes, the lowest 12 bits are set to 0.

HP_DMA_PMS_REG provides separate read and write permissions for each DMA master to access each address range.

16.4 Programming Procedure

As HP CPUs can work in either user mode or machine mode, HP APM and LP APM grant specific access permissions for each work mode. Therefore, users need to configure the work mode for HP CPUs first in order for the HP CPUs to access internal memory, external memory, and peripheral registers.

Since LP CPU only supports machine mode, HP APM and LP APM grant access permissions for the machine mode of LP CPU.

DMA masters can access slaves regardless of the CPUs' work mode. DMA APM provides separate read and write permissions for each DMA master.

16.4.1 Configuring Access Permissions for HP CPU0/1

The configuration process for HP CPU0/1 accessing internal memory, external memory, or peripheral registers is as follows:

1. Configure user mode or machine mode for HP CPU0/1. For how to configure the work mode, please refer to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.
2. If accessing peripheral registers, two additional address ranges can be configured, please refer to Section 16.3.2 for details.
3. Configure access permissions:
   - Configure the access permissions to internal memory (HP L2MEM and HP ROM), external memory (External RAM and flash), and peripheral registers (HP PERI and HP CPU PERI) in the HP system through HP_PERI_PMS_REG:
     - Configure the permission of HP CPU*n* accessing the *MODULE_NAME* module in *MODE_NAME* mode using PMS_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW, for example PMS_CORE*n*_MM_PSRAM_ALLOW.
   - Configure the access permissions to LP SRAM and LP PERI through HP2LP_PERI_PMS_REG:
     - Configure the permission of HP CPU*n* accessing the *MODULE_NAME* module in *MODE_NAME* mode using PMS_HP_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW, for example PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW.

16.4.2 Configuring Access Permissions for LP CPU

LP CPU only supports machine mode. The configuration process for LP CPU accessing internal memory, external memory, or peripheral registers is as follows:

1. If accessing peripheral registers, two additional address ranges can be configured, please refer to Section 16.3.2 for details.
2. Configure access permissions:
   - Configure the access permissions to internal memory (HP L2MEM and HP ROM), external memory (External RAM and flash), and peripheral registers (HP PERI and HP CPU PERI) in the HP system through LP2HP_PERI_PMS_REG:
     - Configure the permission of LP CPU accessing the *MODULE_NAME* module in machine mode using PMS_LP_MM_*MODULE_NAME*_ALLOW, for example PMS_LP_MM_PSRAM_ALLOW.
   - Configure the access permissions to peripheral registers (LP PERI) in the LP system through LP_PERI_PMS_REG:
     - Configure the permission of LP CPU accessing the *MODULE_NAME* module in machine mode using PMS_LP_MM_*MODULE_NAME*_ALLOW, for example PMS_LP_MM_LP_SYSREG_ALLOW.

16.4.3 Configuring Access Permission for DMA Masters

The configuration process for DMA masters accessing internal memory, external memory, and other address spaces is as follows:

1. Configure the address ranges accessible to all DMA masters:
   - Configure the high 20 bits of the starting address of region*n* using PMS_DMA_REGION*n*_LOW_REG.
   - Configure the high 20 bits of the ending address of region*n* using PMS_DMA_REGION*n*_HIGH_REG.
2. Configure the read and write permissions for each DMA master (*DMA_MASTER*) to access each address range:
   - Configure the read permissions using PMS_*DMA_MASTER*_PMS_R_REG, such as PMS_DMA_GMAC_PMS_R_REG.
   - Configure the write permissions using PMS_*DMA_MASTER*_PMS_W_REG, such as PMS_DMA_GMAC_PMS_W_REG.

16.5 Illegal Access and Interrupts

When the APM module manages access paths, it parses the information carried on the bus (e.g., master ID, access direction (read or write), access address, and working mode) and then determines the access permission. When the APM module confirms that the access lacks permission, ESP32-P4 will consider it as an unauthorized access and will take the following actions:

- Reject the access request and provide an error response.
- HP APM will trigger ICM_SYS_REG_INT interrupt signal.
- DMA APM will trigger ICM_SYS_REG_INT interrupt signal.
- LP APM will trigger LP_SYSREG_INT interrupt signal.

These interrupt signals are generated by the internal interrupt sources of the APM module. Table 18.3-1 lists the internal interrupt sources, interrupt trigger conditions, and the generated interrupt signals.

Table 16.5-1. APM Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
ICM_CPU_ADDRHOLE_INTR	Triggered when HP APM detects unauthorized access; or when DMA APM detects unauthorized access from high-speed USB 2.0 OTG, full-speed USB 2.0 OTG, EMAC, SDMMC, Trace0/1, GDMA-AHB, L2MEM Monitor, or TCM Monitor.	ICM_SYS_REG_INT
ICM_SYS_ADDRHOLE_INTR	Triggered when DMA APM detects unauthorized access from DW-GDMA, GDMA-AXI, 2D-DMA, or H264 DMA.	ICM_SYS_REG_INT
LP_ADDRHOLE_INTR	Triggered when LP APM detects unauthorized access.	LP_SYSREG_INTR

APM automatically records relevant information about illegal accesses, including master ID, access direction (read or write), access address, illegal access flags, etc. All this information can be retrieved from the relevant registers described in Chapter 6 *System Registers [to be added later]*.

16.6 Register Summary

16.6.1 HP_DMA_PMS_REG

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

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

Name	Description	Address	Access
Version Control Registers			
PMS_DMA_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_DMA_CLK_EN_REG	Clock gating register	0x0004	R/W
Region Configuration Registers			
PMS_DMA_REGION0_LOW_REG	Region0 start address configuration register	0x0008	R/W
PMS_DMA_REGION0_HIGH_REG	Region0 end address configuration register	0x000C	R/W
PMS_DMA_REGION1_LOW_REG	Region1 start address configuration register	0x0010	R/W
PMS_DMA_REGION1_HIGH_REG	Region1 end address configuration register	0x0014	R/W
PMS_DMA_REGION2_LOW_REG	Region2 start address configuration register	0x0018	R/W
PMS_DMA_REGION2_HIGH_REG	Region2 end address configuration register	0x001C	R/W
PMS_DMA_REGION3_LOW_REG	Region3 start address configuration register	0x0020	R/W
PMS_DMA_REGION3_HIGH_REG	Region3 end address configuration register	0x0024	R/W
PMS_DMA_REGION4_LOW_REG	Region4 start address configuration register	0x0028	R/W
PMS_DMA_REGION4_HIGH_REG	Region4 end address configuration register	0x002C	R/W
PMS_DMA_REGION5_LOW_REG	Region5 start address configuration register	0x0030	R/W
PMS_DMA_REGION5_HIGH_REG	Region5 end address configuration register	0x0034	R/W
PMS_DMA_REGION6_LOW_REG	Region6 start address configuration register	0x0038	R/W
PMS_DMA_REGION6_HIGH_REG	Region6 end address configuration register	0x003C	R/W
PMS_DMA_REGION7_LOW_REG	Region7 start address configuration register	0x0040	R/W

Name	Description	Address	Access
PMS_DMA_REGION7_HIGH_REG	Region7 end address configuration register	0x0044	R/W
PMS_DMA_REGION8_LOW_REG	Region8 start address configuration register	0x0048	R/W
PMS_DMA_REGION8_HIGH_REG	Region8 end address configuration register	0x004C	R/W
PMS_DMA_REGION9_LOW_REG	Region9 start address configuration register	0x0050	R/W
PMS_DMA_REGION9_HIGH_REG	Region9 end address configuration register	0x0054	R/W
PMS_DMA_REGION10_LOW_REG	Region10 start address configuration register	0x0058	R/W
PMS_DMA_REGION10_HIGH_REG	Region10 end address configuration register	0x005C	R/W
PMS_DMA_REGION11_LOW_REG	Region11 start address configuration register	0x0060	R/W
PMS_DMA_REGION11_HIGH_REG	Region11 end address configuration register	0x0064	R/W
PMS_DMA_REGION12_LOW_REG	Region12 start address configuration register	0x0068	R/W
PMS_DMA_REGION12_HIGH_REG	Region12 end address configuration register	0x006C	R/W
PMS_DMA_REGION13_LOW_REG	Region13 start address configuration register	0x0070	R/W
PMS_DMA_REGION13_HIGH_REG	Region13 end address configuration register	0x0074	R/W
PMS_DMA_REGION14_LOW_REG	Region14 start address configuration register	0x0078	R/W
PMS_DMA_REGION14_HIGH_REG	Region14 end address configuration register	0x007C	R/W
PMS_DMA_REGION15_LOW_REG	Region15 start address configuration register	0x0080	R/W
PMS_DMA_REGION15_HIGH_REG	Region15 end address configuration register	0x0084	R/W
PMS_DMA_REGION16_LOW_REG	Region16 start address configuration register	0x0088	R/W
PMS_DMA_REGION16_HIGH_REG	Region16 end address configuration register	0x008C	R/W
PMS_DMA_REGION17_LOW_REG	Region17 start address configuration register	0x0090	R/W
PMS_DMA_REGION17_HIGH_REG	Region17 end address configuration register	0x0094	R/W
PMS_DMA_REGION18_LOW_REG	Region18 start address configuration register	0x0098	R/W

Name	Description	Address	Access
PMS_DMA_REGION18_HIGH_REG	Region18 end address configuration register	0x009C	R/W
PMS_DMA_REGION19_LOW_REG	Region19 start address configuration register	0x00A0	R/W
PMS_DMA_REGION19_HIGH_REG	Region19 end address configuration register	0x00A4	R/W
PMS_DMA_REGION20_LOW_REG	Region20 start address configuration register	0x00A8	R/W
PMS_DMA_REGION20_HIGH_REG	Region20 end address configuration register	0x00AC	R/W
PMS_DMA_REGION21_LOW_REG	Region21 start address configuration register	0x00B0	R/W
PMS_DMA_REGION21_HIGH_REG	Region21 end address configuration register	0x00B4	R/W
PMS_DMA_REGION22_LOW_REG	Region22 start address configuration register	0x00B8	R/W
PMS_DMA_REGION22_HIGH_REG	Region22 end address configuration register	0x00BC	R/W
PMS_DMA_REGION23_LOW_REG	Region23 start address configuration register	0x00C0	R/W
PMS_DMA_REGION23_HIGH_REG	Region23 end address configuration register	0x00C4	R/W
PMS_DMA_REGION24_LOW_REG	Region24 start address configuration register	0x00C8	R/W
PMS_DMA_REGION24_HIGH_REG	Region24 end address configuration register	0x00CC	R/W
PMS_DMA_REGION25_LOW_REG	Region25 start address configuration register	0x00D0	R/W
PMS_DMA_REGION25_HIGH_REG	Region25 end address configuration register	0x00D4	R/W
PMS_DMA_REGION26_LOW_REG	Region26 start address configuration register	0x00D8	R/W
PMS_DMA_REGION26_HIGH_REG	Region26 end address configuration register	0x00DC	R/W
PMS_DMA_REGION27_LOW_REG	Region27 start address configuration register	0x00E0	R/W
PMS_DMA_REGION27_HIGH_REG	Region27 end address configuration register	0x00E4	R/W
PMS_DMA_REGION28_LOW_REG	Region28 start address configuration register	0x00E8	R/W
PMS_DMA_REGION28_HIGH_REG	Region28 end address configuration register	0x00EC	R/W
PMS_DMA_REGION29_LOW_REG	Region29 start address configuration register	0x00F0	R/W

Name	Description	Address	Access
PMS_DMA_REGION29_HIGH_REG	Region29 end address configuration register	0x00F4	R/W
PMS_DMA_REGION30_LOW_REG	Region30 start address configuration register	0x00F8	R/W
PMS_DMA_REGION30_HIGH_REG	Region30 end address configuration register	0x00FC	R/W
PMS_DMA_REGION31_LOW_REG	Region31 start address configuration register	0x0100	R/W
PMS_DMA_REGION31_HIGH_REG	Region31 end address configuration register	0x0104	R/W
DMA Masters Read and Write Permission Control Registers			
PMS_DMA_GDMA_CH0_R_PMS_REG	GDMA ch0 read permission control register	0x0108	R/W
PMS_DMA_GDMA_CH0_W_PMS_REG	GDMA ch0 write permission control register	0x010C	R/W
PMS_DMA_GDMA_CH1_R_PMS_REG	GDMA ch1 read permission control register	0x0110	R/W
PMS_DMA_GDMA_CH1_W_PMS_REG	GDMA ch1 write permission control register	0x0114	R/W
PMS_DMA_GDMA_CH2_R_PMS_REG	GDMA ch2 read permission control register	0x0118	R/W
PMS_DMA_GDMA_CH2_W_PMS_REG	GDMA ch2 write permission control register	0x011C	R/W
PMS_DMA_GDMA_CH3_R_PMS_REG	GDMA ch3 read permission control register	0x0120	R/W
PMS_DMA_GDMA_CH3_W_PMS_REG	GDMA ch3 write permission control register	0x0124	R/W
PMS_DMA_AHB_PDMA_ADC_R_PMS_REG	GDMA-AHB ADC read permission control register	0x0128	R/W
PMS_DMA_AHB_PDMA_ADC_W_PMS_REG	GDMA-AHB ADC write permission control register	0x012C	R/W
PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG	GDMA-AHB I2S0 read permission control register	0x0130	R/W
PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG	GDMA-AHB I2S0 write permission control register	0x0134	R/W
PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG	GDMA-AHB I2S1 read permission control register	0x0138	R/W
PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG	GDMA-AHB I2S1 write permission control register	0x013C	R/W
PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG	GDMA-AHB I2S2 read permission control register	0x0140	R/W
PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG	GDMA-AHB I2S2 write permission control register	0x0144	R/W

Name	Description	Address	Access
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG	GDMA-AHB I3C MST read permission control register	0x0148	R/W
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG	GDMA-AHB I3C MST write permission control register	0x014C	R/W
PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG	GDMA-AHB UHCI read permission control register	0x0150	R/W
PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG	GDMA-AHB UHCI write permission control register	0x0154	R/W
PMS_DMA_AHB_PDMA_RMT_R_PMS_REG	GDMA-AHB RMT read permission control register	0x0158	R/W
PMS_DMA_AHB_PDMA_RMT_W_PMS_REG	GDMA-AHB RMT write permission control register	0x0170	R/W
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG	GDMA-AXI LCD_CAM read permission control register	0x0174	R/W
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG	GDMA-AXI LCD_CAM write permission control register	0x0178	R/W
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG	GDMA-AXI GPSPI2 read permission control register	0x017C	R/W
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG	GDMA-AXI GPSPI2 write permission control register	0x0180	R/W
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG	GDMA-AXI GPSPI3 read permission control register	0x0184	R/W
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG	GDMA-AXI GPSPI3 write permission control register	0x0188	R/W
PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG	GDMA-AXI PARLIO read permission control register	0x018C	R/W
PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG	GDMA-AXI PARLIO write permission control register	0x0190	R/W
PMS_DMA_AXI_PDMA_AES_R_PMS_REG	GDMA-AXI AES read permission control register	0x0194	R/W
PMS_DMA_AXI_PDMA_AES_W_PMS_REG	GDMA-AXI AES write permission control register	0x0198	R/W
PMS_DMA_AXI_PDMA_SHA_R_PMS_REG	GDMA-AXI SHA read permission control register	0x019C	R/W
PMS_DMA_AXI_PDMA_SHA_W_PMS_REG	GDMA-AXI SHA write permission control register	0x01A0	R/W
PMS_DMA_DMA2D_JPEG_PMS_R_REG	2D-DMA JPEG read permission control register	0x01A4	R/W
PMS_DMA_DMA2D_JPEG_PMS_W_REG	2D-DMA JPEG write permission control register	0x01A8	R/W
PMS_DMA_USB_PMS_R_REG	High-speed USB 2.0 OTG read permission control register	0x01AC	R/W
PMS_DMA_USB_PMS_W_REG	High-speed USB 2.0 OTG write permission control register	0x01B0	R/W

Name	Description	Address	Access
PMS_DMA_GMAC_PMS_R_REG	EMAC read permission control register	0x01B4	R/W
PMS_DMA_GMAC_PMS_W_REG	EMAC write permission control register	0x01B8	R/W
PMS_DMA_SDMMC_PMS_R_REG	SDMMC read permission control register	0x01BC	R/W
PMS_DMA_SDMMC_PMS_W_REG	SDMMC write permission control register	0x01C0	R/W
PMS_DMA_USBOTG11_PMS_R_REG	Full-speed USB 2.0 OTG read permission control register	0x01C4	R/W
PMS_DMA_USBOTG11_PMS_W_REG	Full-speed USB 2.0 OTG write permission control register	0x01C8	R/W
PMS_DMA_TRACE0_PMS_R_REG	TRACE0 read permission control register	0x01CC	R/W
PMS_DMA_TRACE0_PMS_W_REG	TRACE0 write permission control register	0x01D0	R/W
PMS_DMA_TRACE1_PMS_R_REG	TRACE1 read permission control register	0x01D4	R/W
PMS_DMA_TRACE1_PMS_W_REG	TRACE1 write permission control register	0x01D8	R/W
PMS_DMA_L2MEM_MON_PMS_W_REG	L2MEM Monitor write permission control register	0x01E0	R/W
PMS_DMA_TCM_MON_PMS_W_REG	TCM Monitor write permission control register	0x01E8	R/W
PMS_DMA_H264_PMS_R_REG	H264 DMA read permission control register	0x01FC	R/W
PMS_DMA_H264_PMS_W_REG	H264 DMA write permission control register	0x0200	R/W
PMS_DMA_DMA2D_PPA_PMS_R_REG	2D-DMA PPA read permission control register	0x0204	R/W
PMS_DMA_DMA2D_PPA_PMS_W_REG	2D-DMA PPA write permission control register	0x0208	R/W
PMS_DMA_DMA2D_DUMMY_PMS_R_REG	2D-DMA Dummy read permission control register	0x020C	R/W
PMS_DMA_DMA2D_DUMMY_PMS_W_REG	2D-DMA Dummy write permission control register	0x0210	R/W
PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG	GDMA-AHB Dummy read permission control register	0x0214	R/W
PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG	GDMA-AHB Dummy write permission control register	0x0218	R/W
PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG	GDMA-AXI Dummy read permission control register	0x021C	R/W
PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG	GDMA-AXI Dummy write permission control register	0x0220	R/W

16.6.2 HP_PERI_PMS_REG

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

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

Name	Description	Address	Access
Version Control Registers			
PMS_HP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_HP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
HP CPU Permission Control Registers			
PMS_CORE0_MM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU0 in machine mode	0x0008	R/W
PMS_CORE0_MM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU0 in machine mode	0x000C	R/W
PMS_CORE0_MM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU0 in machine mode	0x0010	R/W
PMS_CORE0_MM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU0 in machine mode	0x0014	R/W
PMS_CORE0_UM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU0 in user mode	0x0018	R/W
PMS_CORE0_UM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU0 in user mode	0x001C	R/W
PMS_CORE0_UM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU0 in user mode	0x0020	R/W
PMS_CORE0_UM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU0 in user mode	0x0024	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU1 in machine mode	0x0028	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU1 in machine mode	0x002C	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU1 in machine mode	0x0030	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU1 in machine mode	0x0034	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG0_REG	Permission control register0 for HP CPU1 in user mode	0x0038	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG1_REG	Permission control register1 for HP CPU1 in user mode	0x003C	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG2_REG	Permission control register2 for HP CPU1 in user mode	0x0040	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG3_REG	Permission control register3 for HP CPU1 in user mode	0x0044	R/W

16.6.3 LP2HP_PERI_PMS_REG

The addresses in this section are relative to the LP2HP_PERI_PMS base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Version Control Registers			
PMS_LP2HP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_LP2HP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
LP CPU Permission Control Registers			
PMS_LP_MM_PMS_REG0_REG	Permission control register0 for the LP CPU in machine mode	0x0008	R/W
PMS_LP_MM_PMS_REG1_REG	Permission control register1 for the LP CPU in machine mode	0x0030	R/W
PMS_LP_MM_PMS_REG2_REG	Permission control register2 for the LP CPU in machine mode	0x00A4	R/W
PMS_LP_MM_PMS_REG3_REG	Permission control register3 for the LP CPU in machine mode	0x011C	R/W

16.6.4 LP_PERI_PMS_REG

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

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

Name	Description	Address	Access
Version Control Registers			
PMS_LP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_LP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
LP CPU Permission Control Registers			
PMS_LP_MM_LP_PERI_PMS_REG0_REG	Permission control register0 for LP CPU in machine mode	0x0008	R/W
Configurable Address Range Configuration Registers			
PMS_PERI_REGION0_LOW_REG	Region0 start address configuration register	0x000C	R/W
PMS_PERI_REGION0_HIGH_REG	Region0 end address configuration register	0x0010	R/W
PMS_PERI_REGION1_LOW_REG	Region1 start address configuration register	0x0014	R/W
PMS_PERI_REGION1_HIGH_REG	Region1 end address configuration register	0x0018	R/W

16.6.5 HP2LP_PERI_PMS_REG

The addresses in this section are relative to the HP2LP_PERI_PMS base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Version Control Registers			
PMS_HP2LP_PERI_PMS_DATE_REG	Version control register	0x0000	R/W
Clock Gating Registers			
PMS_HP2LP_PERI_PMS_CLK_EN_REG	Clock gating register	0x0004	R/W
HP CPU Permission Control Registers			
PMS_HP_COREO_MM_PMS_REGO_REG	Permission control register0 for HP CPU0 in machine mode	0x0008	R/W
PMS_HP_COREO_UM_PMS_REGO_REG	Permission control register0 for HP CPU0 in user mode	0x000C	R/W
PMS_HP_CORE1_MM_PMS_REGO_REG	Permission control register0 for HP CPU1 in machine mode	0x0010	R/W
PMS_HP_CORE1_UM_PMS_REGO_REG	Permission control register0 for HP CPU1 in user mode	0x0014	R/W

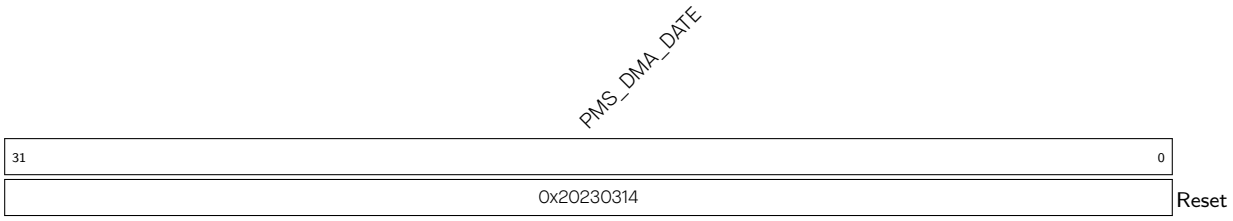
16.7 Registers

16.7.1 HP_DMA_PMS_REG

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

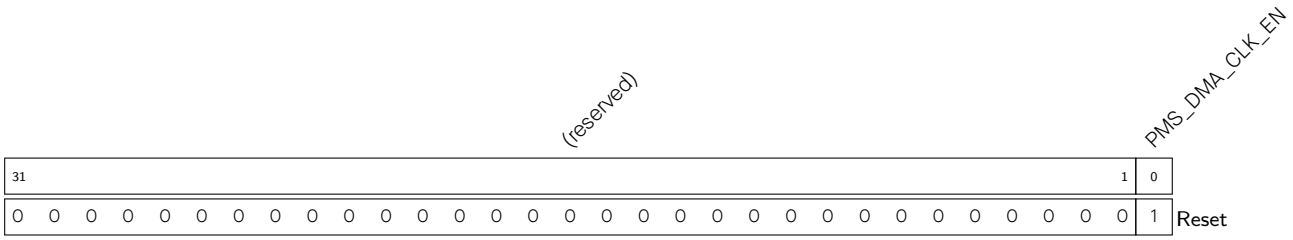
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 16.1. PMS_DMA_DATE_REG (0x0000)



PMS_DMA_DATE Version control register. (R/W)

Register 16.2. PMS_DMA_CLK_EN_REG (0x0004)



PMS_DMA_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating.

1: Keep the clock always on.

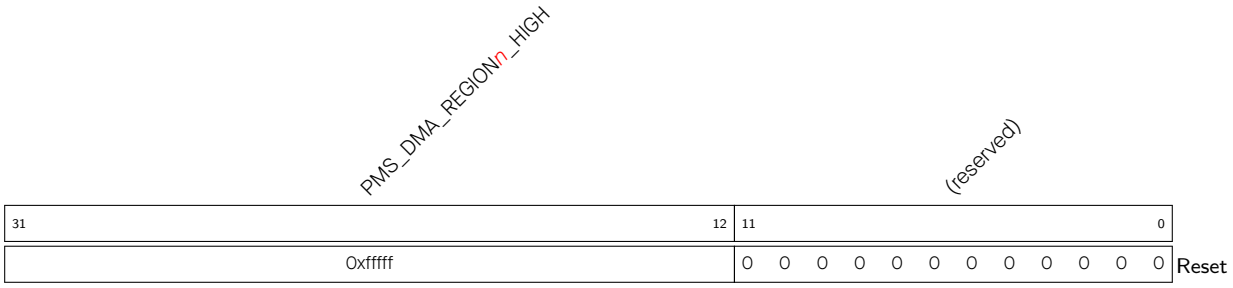
(R/W)

Register 16.3. PMS_DMA_REGION*n*_LOW_REG (*n*: 0-31) (0x0008+0x8n*)**



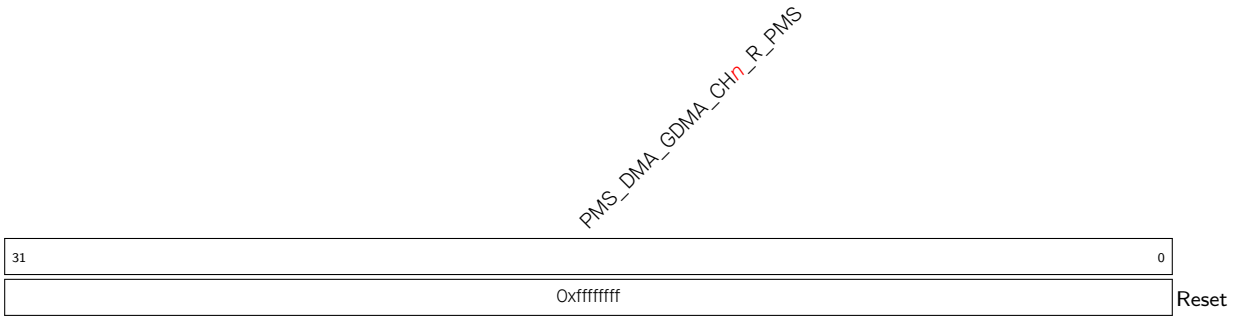
PMS_DMA_REGION*n*_LOW Configures the high 20 bits of the start address for region*n*. (R/W)

Register 16.4. PMS_DMA_REGION*n*_HIGH_REG (*n*: 0-31) (0x000C+0x8n*)**



PMS_DMA_REGION*n*_HIGH Configures the high 20 bits of the end address for region*n*. (R/W)

Register 16.5. PMS_DMA_GDMA_CH*n*_R_PMS_REG (*n*: 0-3) (0x0108+0x8n*)**



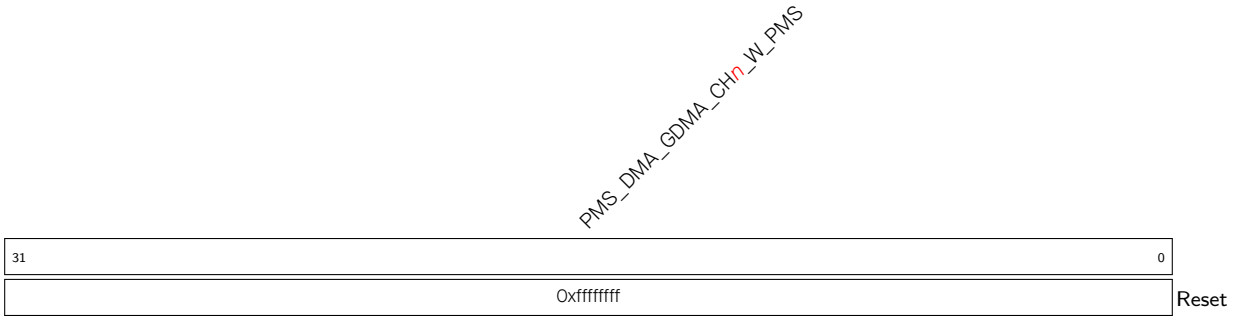
PMS_DMA_GDMA_CH*n*_R_PMS Configures the permission for GDMA ch*n* to read 32 address regions. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.6. PMS_DMA_GDMA_CH*n*_W_PMS_REG (*n*: 0-3) (0x010C+0x8n*)**



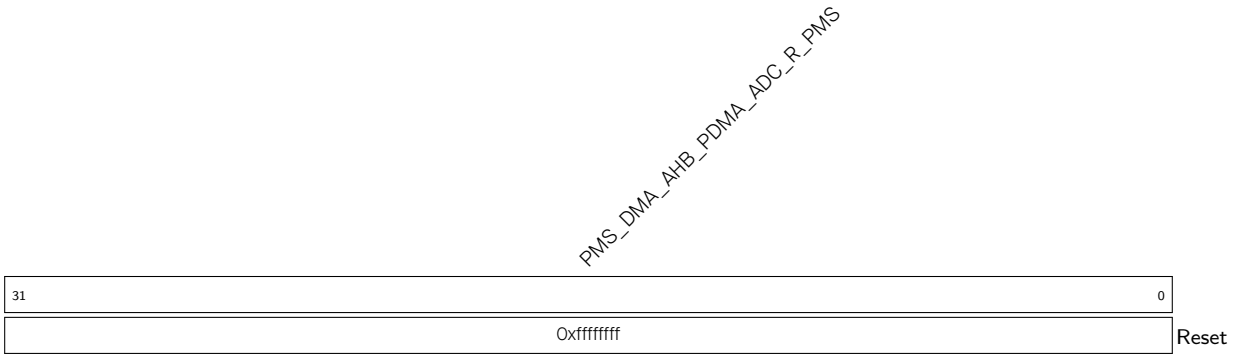
PMS_DMA_GDMA_CH*n*_W_PMS Configures the permission for GDMA ch*n* to write 32 address regions. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.7. PMS_DMA_AHB_PDMA_ADC_R_PMS_REG (0x0128)



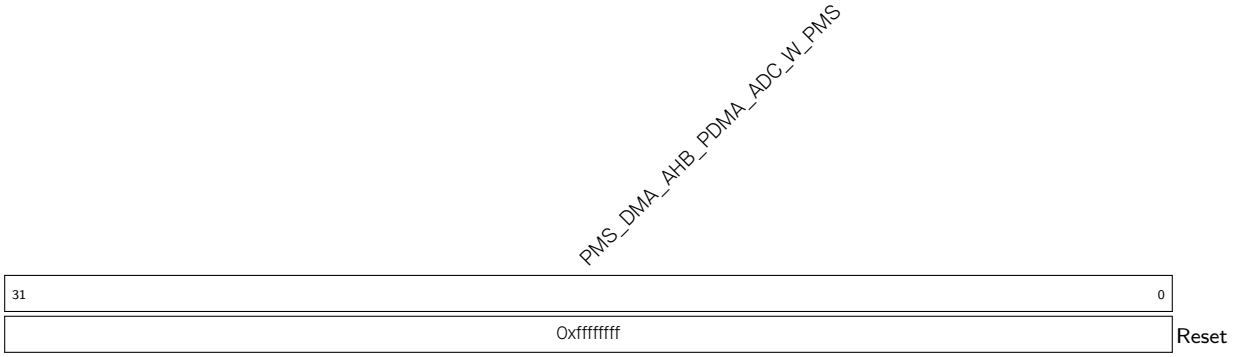
PMS_DMA_AHB_PDMA_ADC_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by ADC. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.8. PMS_DMA_AHB_PDMA_ADC_W_PMS_REG (0x012C)



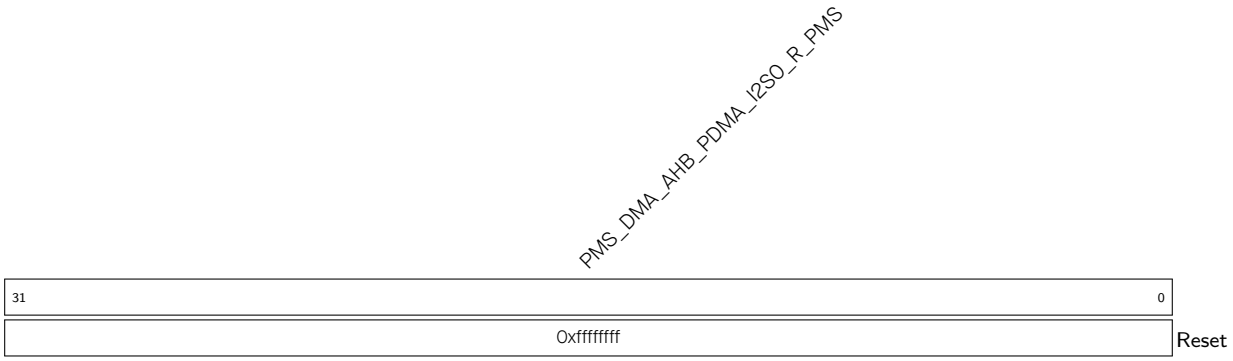
PMS_DMA_AHB_PDMA_ADC_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by ADC. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.9. PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG (0x0130)



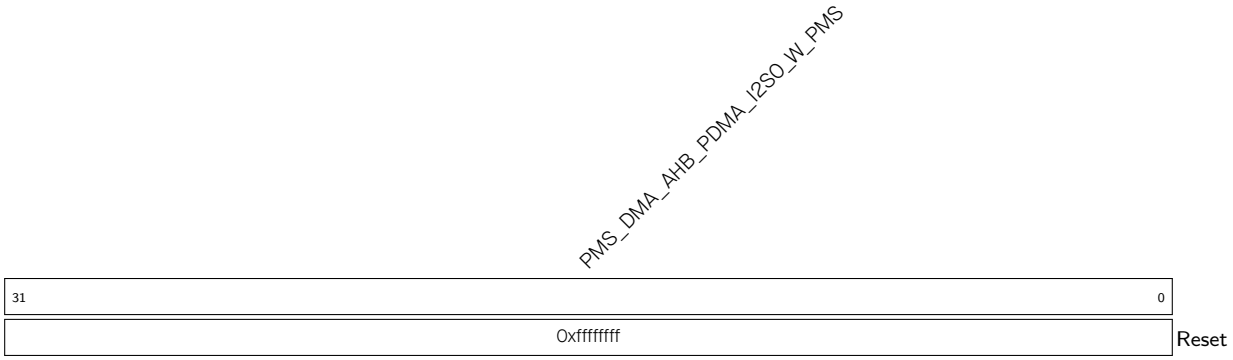
PMS_DMA_AHB_PDMA_I2S0_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S0. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.10. PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG (0x0134)



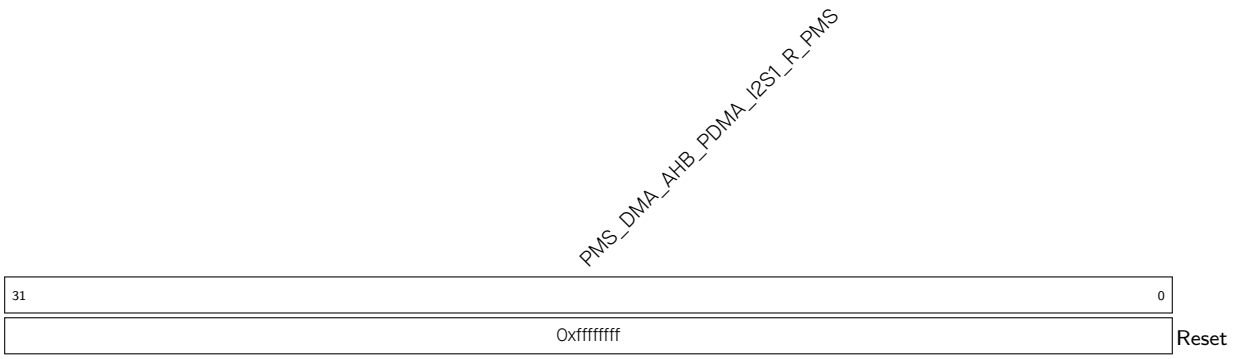
PMS_DMA_AHB_PDMA_I2S0_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S0. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.11. PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG (0x0138)



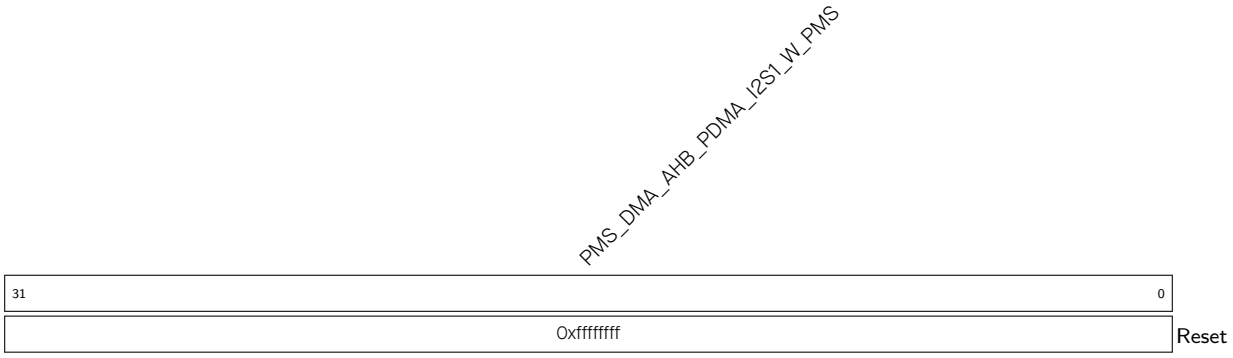
PMS_DMA_AHB_PDMA_I2S1_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S1. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.12. PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG (0x013C)



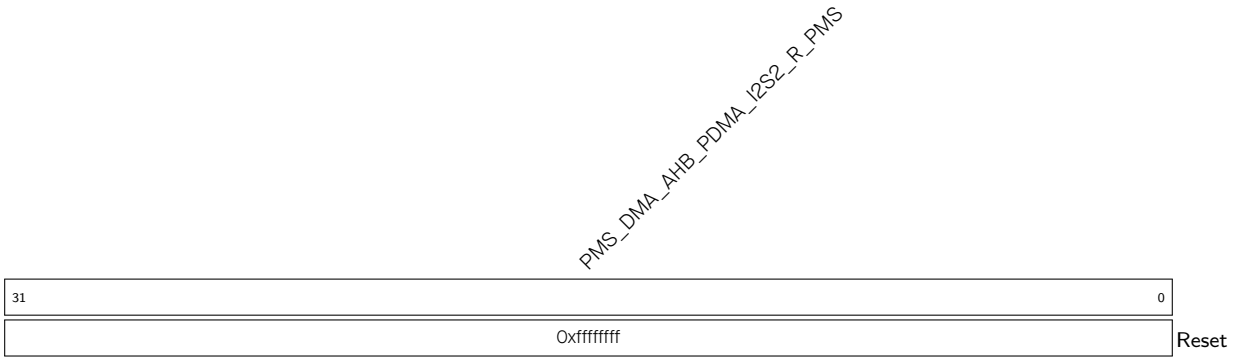
PMS_DMA_AHB_PDMA_I2S1_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S1. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.13. PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG (0x0140)



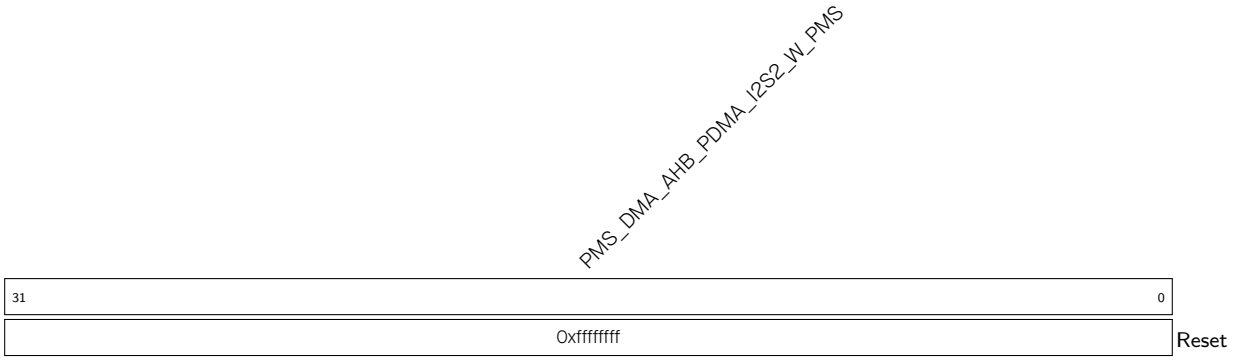
PMS_DMA_AHB_PDMA_I2S2_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I2S2. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.14. PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG (0x0144)



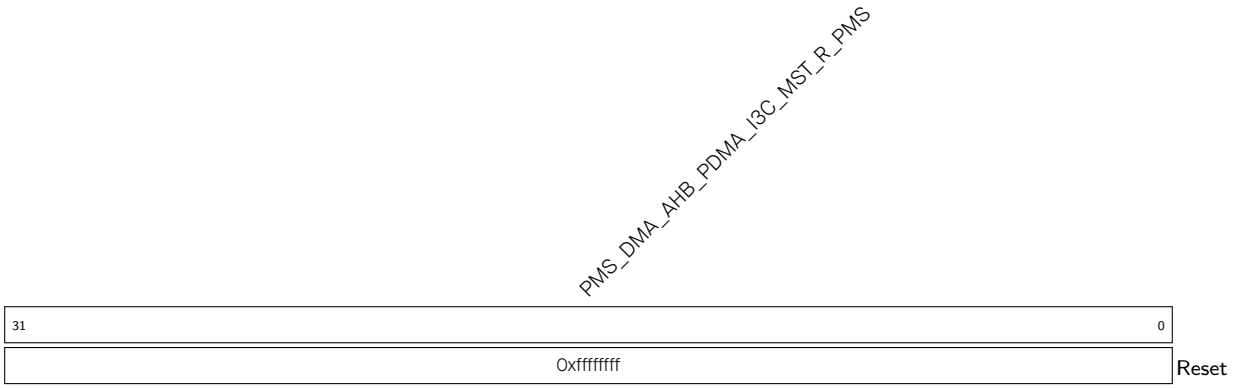
PMS_DMA_AHB_PDMA_I2S2_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I2S2. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.15. PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG (0x0148)



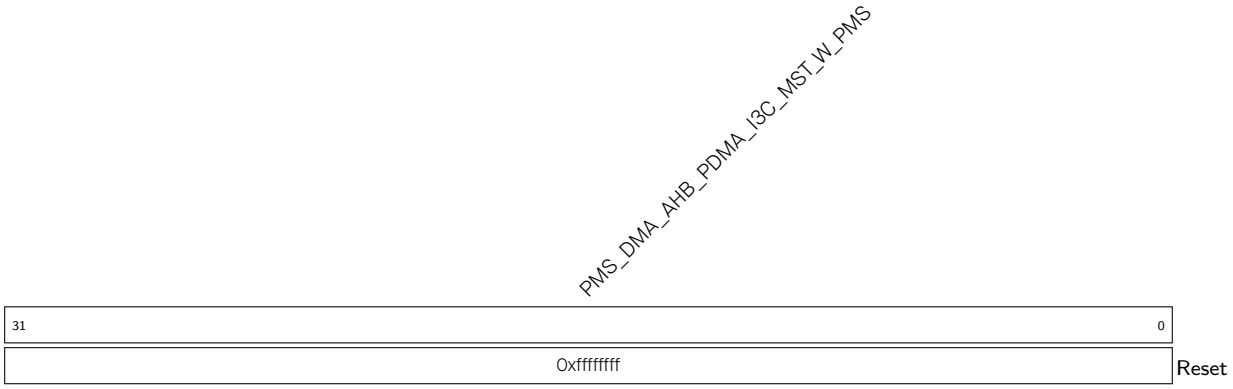
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by I3C master. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.16. PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG (0x014C)



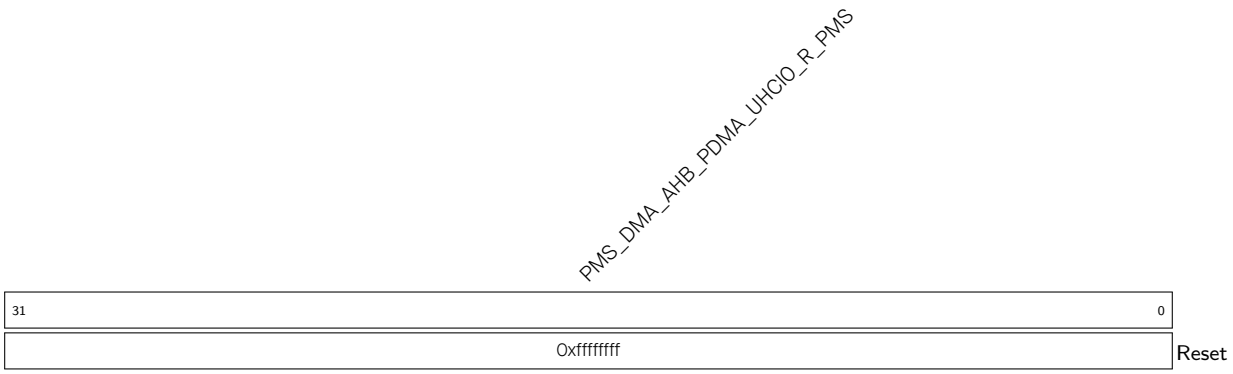
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by I3C master. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.17. PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG (0x0150)



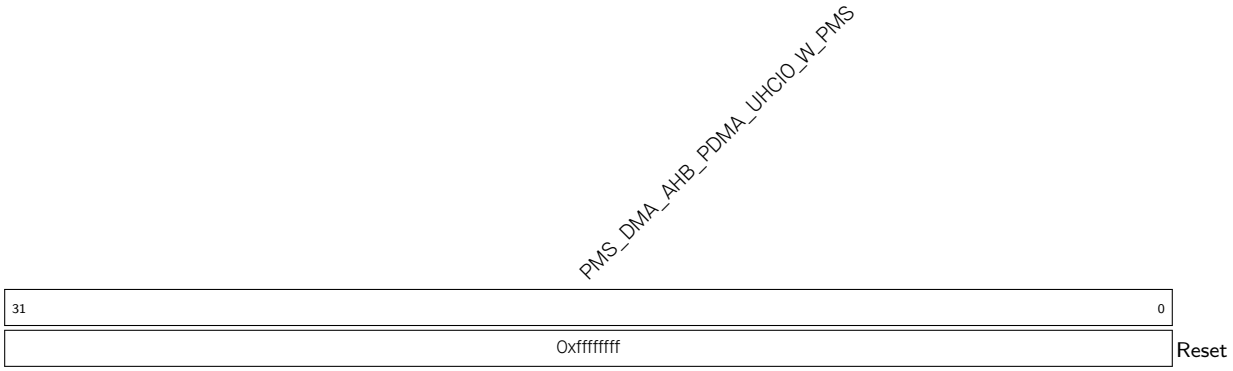
PMS_DMA_AHB_PDMA_UHCI0_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by UHCI. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.18. PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG (0x0154)



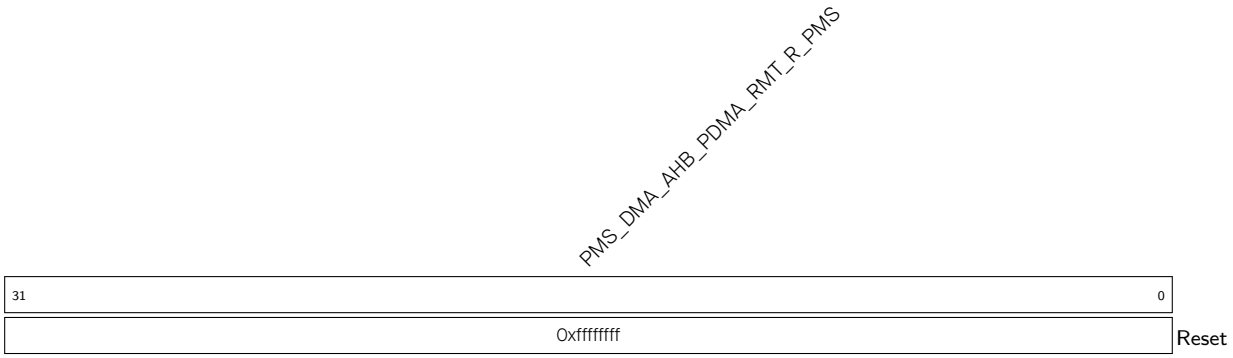
PMS_DMA_AHB_PDMA_UHCI0_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by UHCI. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.19. PMS_DMA_AHB_PDMA_RMT_R_PMS_REG (0x0158)



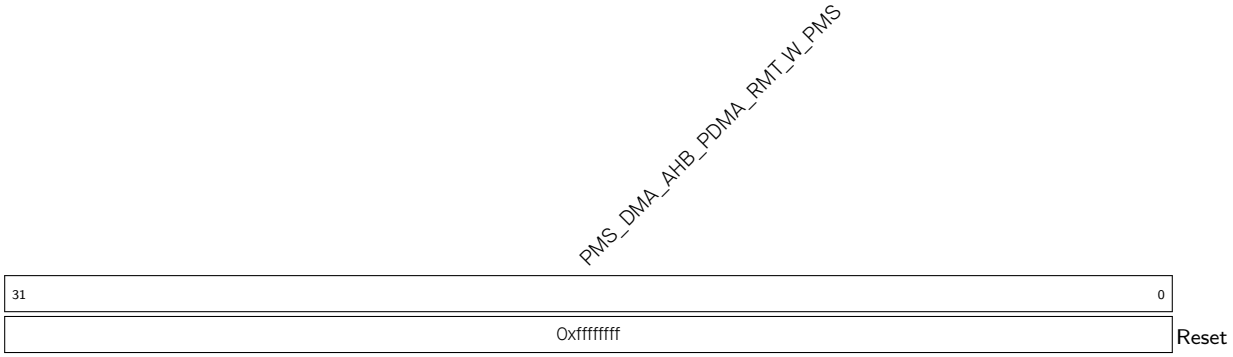
PMS_DMA_AHB_PDMA_RMT_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by RMT. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.20. PMS_DMA_AHB_PDMA_RMT_W_PMS_REG (0x0170)



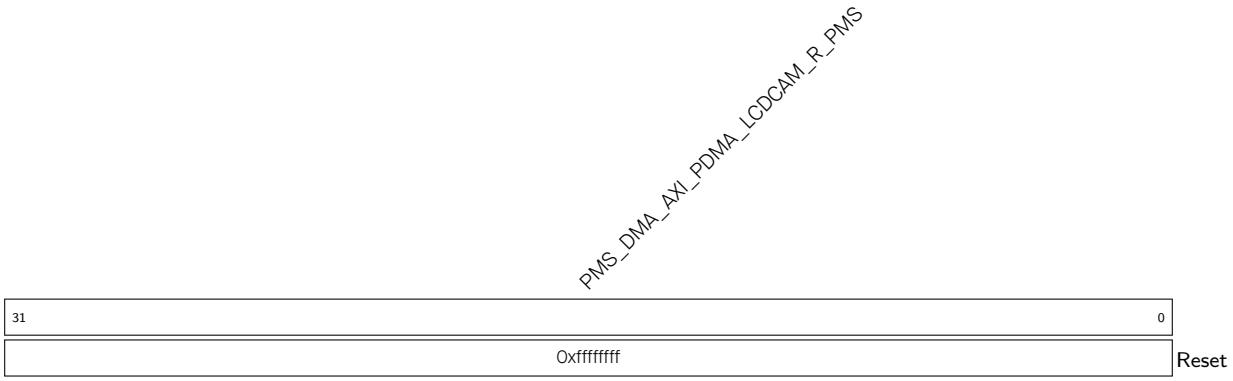
PMS_DMA_AHB_PDMA_RMT_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by RMT. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.21. PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG (0x0174)



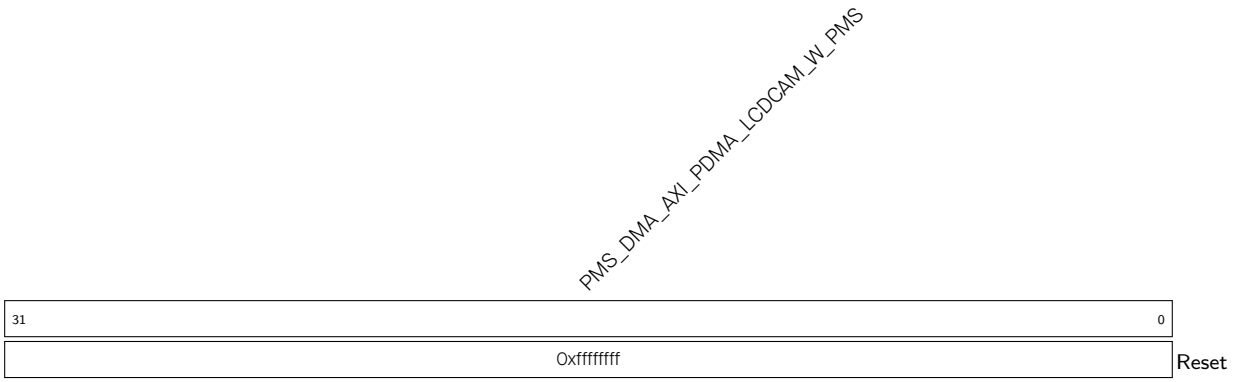
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by LCD_CAM. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.22. PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG (0x0178)



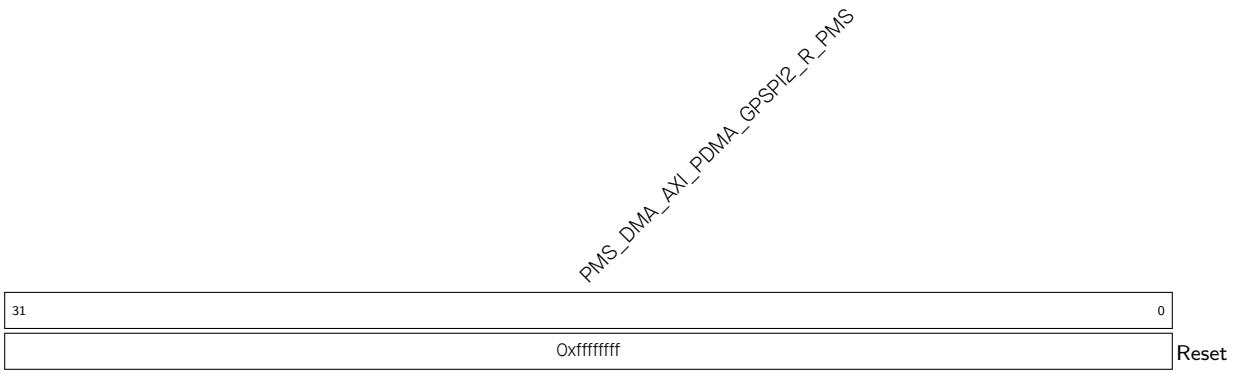
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by LCD_CAM. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.23. PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG (0x017C)



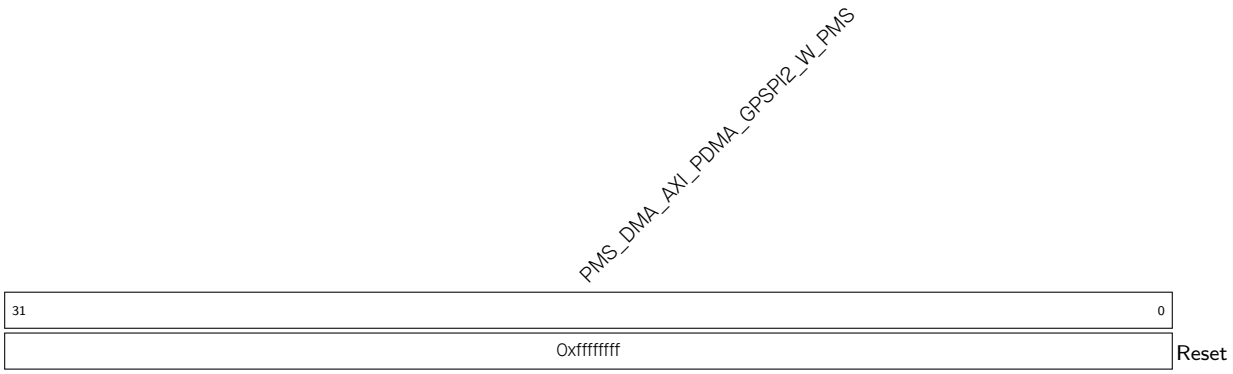
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by GP-SPI2. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.24. PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG (0x0180)



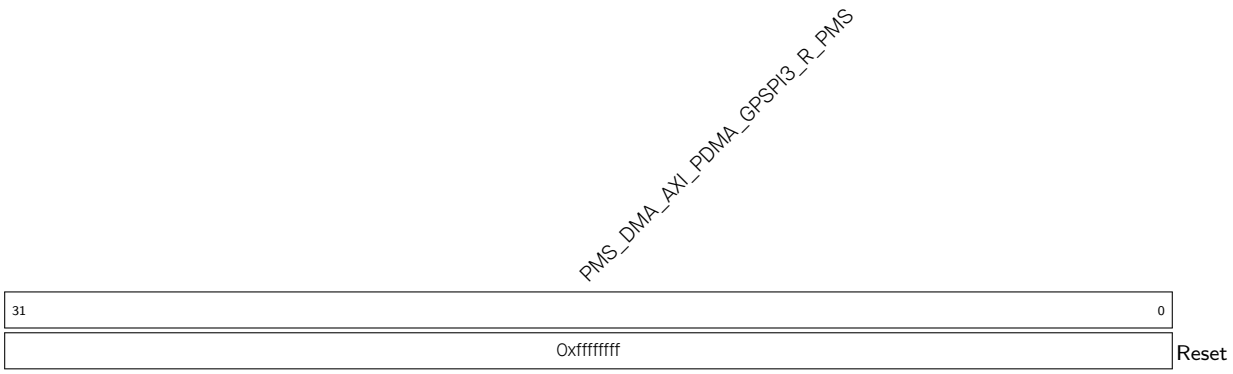
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by GP-SPI2. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.25. PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG (0x0184)



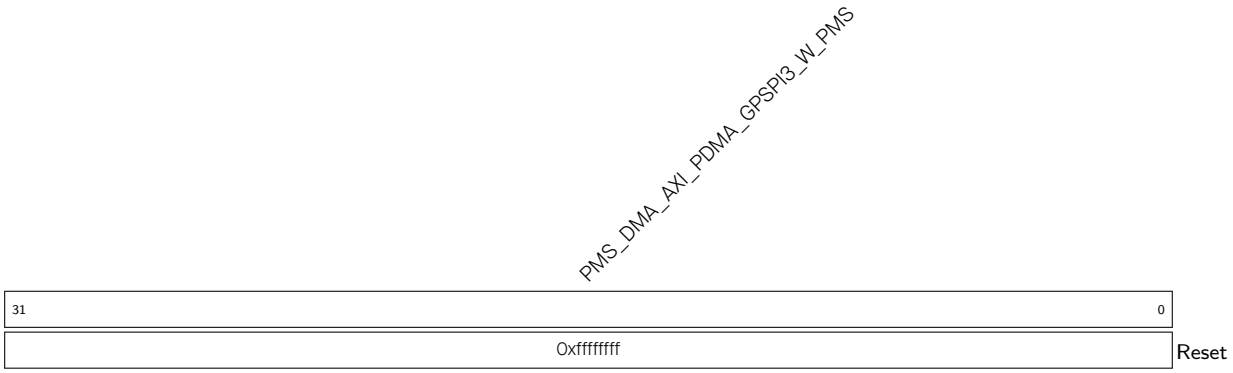
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by GP-SPI3. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.26. PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG (0x0188)



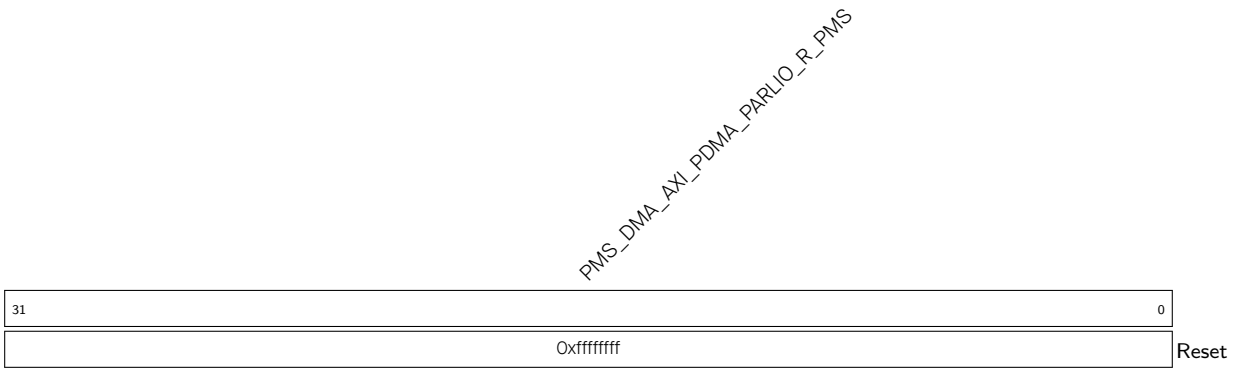
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by GP-SPI3. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.27. PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG (0x018C)



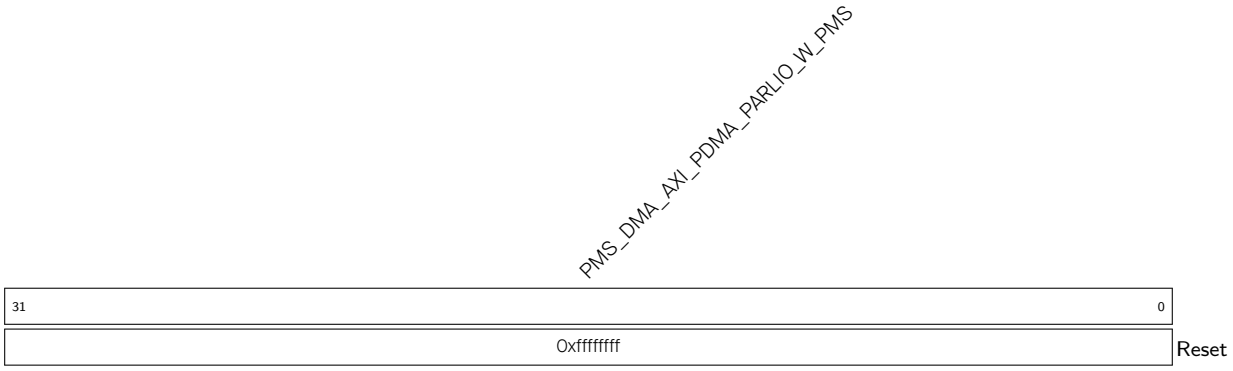
PMS_DMA_AXI_PDMA_PARLIO_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by PARLIO (Parallel IO Controller). Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.28. PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG (0x0190)



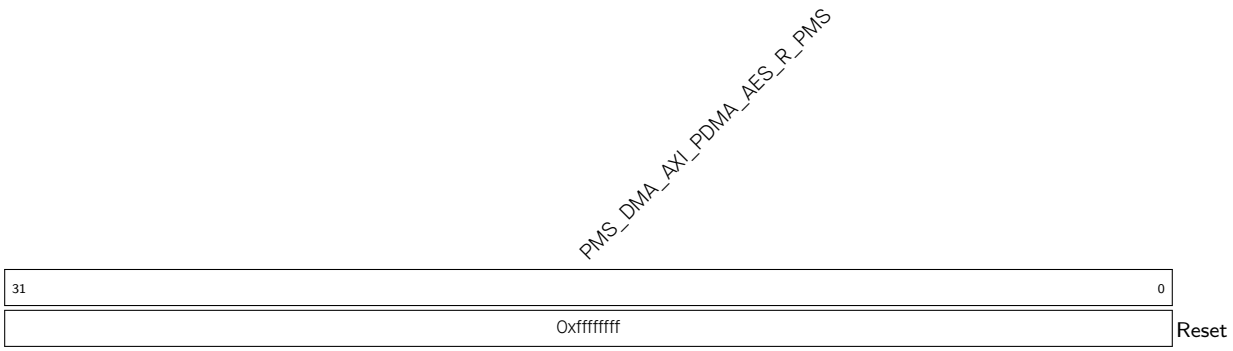
PMS_DMA_AXI_PDMA_PARLIO_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by PARLIO. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.29. PMS_DMA_AXI_PDMA_AES_R_PMS_REG (0x0194)



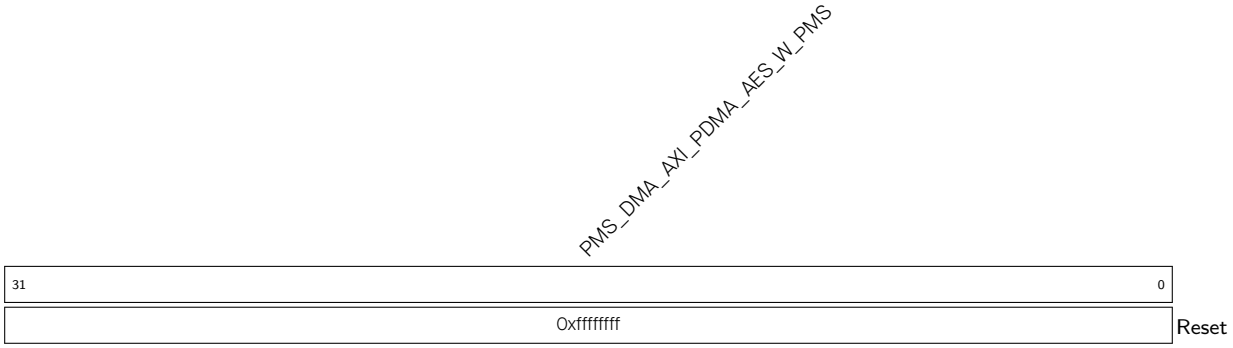
PMS_DMA_AXI_PDMA_AES_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by AES. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.30. PMS_DMA_AXI_PDMA_AES_W_PMS_REG (0x0198)



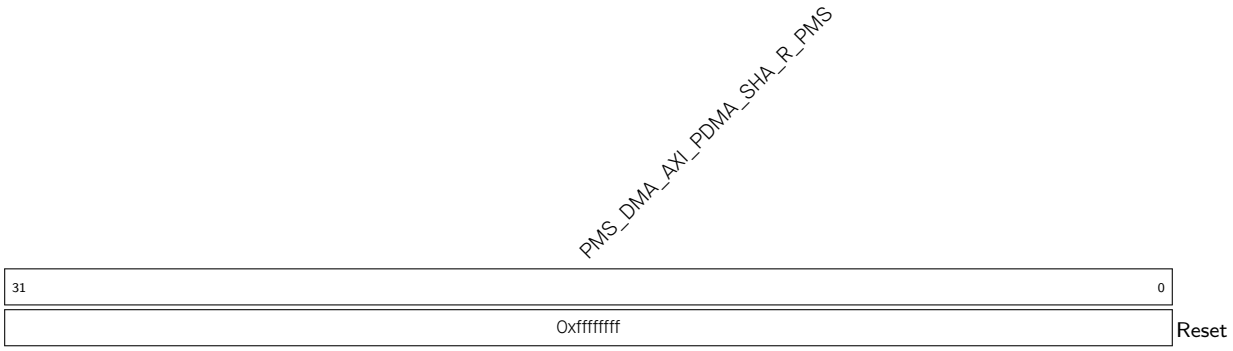
PMS_DMA_AXI_PDMA_AES_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by AES. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.31. PMS_DMA_AXI_PDMA_SHA_R_PMS_REG (0x019C)



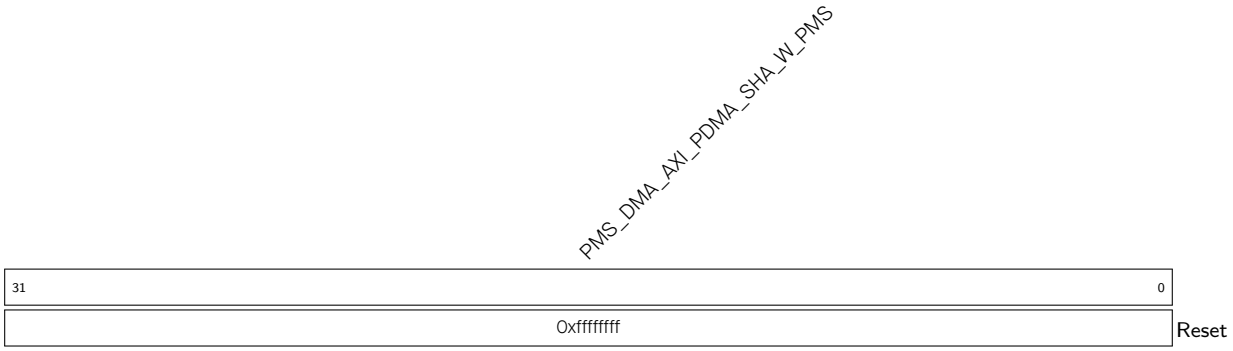
PMS_DMA_AXI_PDMA_SHA_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by SHA. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.32. PMS_DMA_AXI_PDMA_SHA_W_PMS_REG (0x01A0)



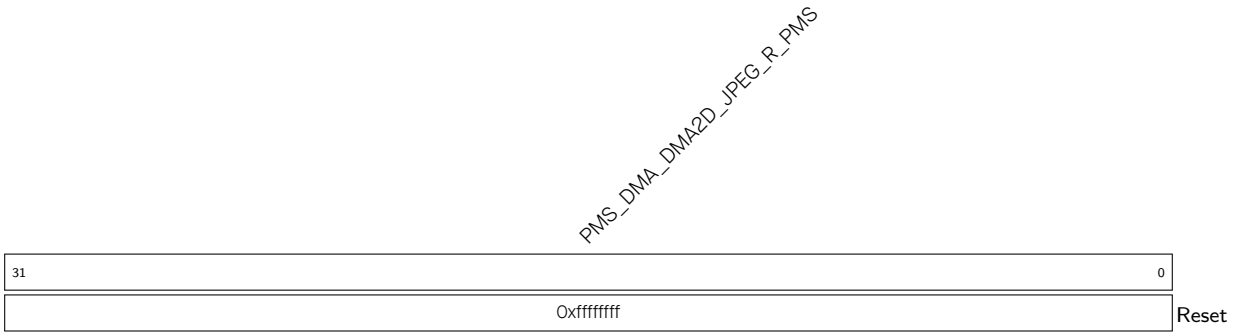
PMS_DMA_AXI_PDMA_SHA_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by SHA. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.33. PMS_DMA_DMA2D_JPEG_PMS_R_REG (0x01A4)



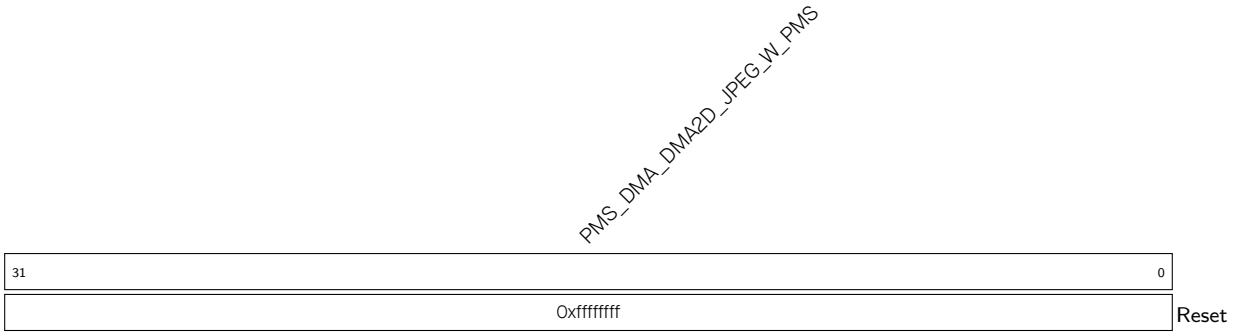
PMS_DMA_DMA2D_JPEG_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by JPEG. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.34. PMS_DMA_DMA2D_JPEG_PMS_W_REG (0x01A8)



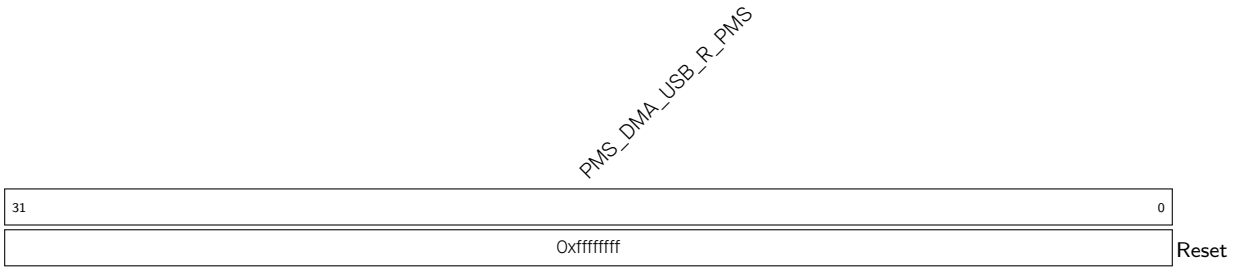
PMS_DMA_DMA2D_JPEG_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by JPEG. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.35. PMS_DMA_USB_PMS_R_REG (0x01AC)



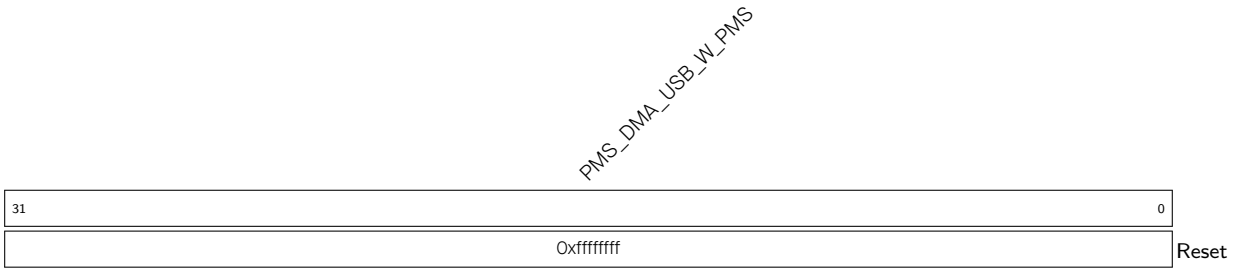
PMS_DMA_USB_R_PMS Configures read permission for high-speed USB 2.0 OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.36. PMS_DMA_USB_PMS_W_REG (0x01B0)



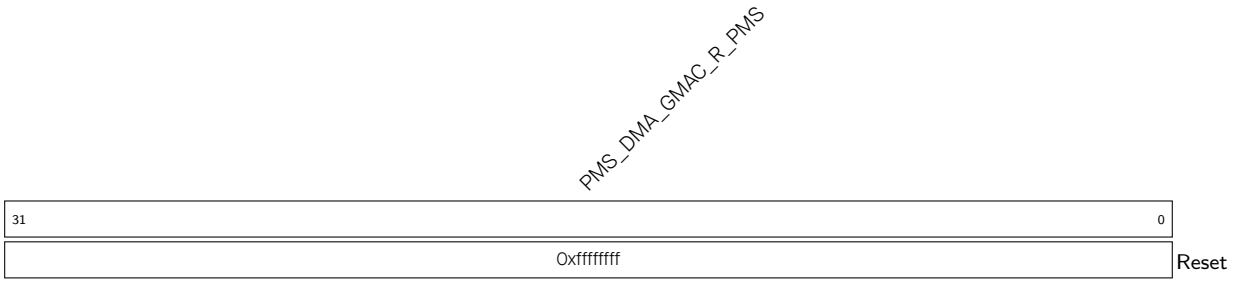
PMS_DMA_USB_W_PMS Configures write permission for high-speed USB 2.0 OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.37. PMS_DMA_GMAC_PMS_R_REG (0x01B4)



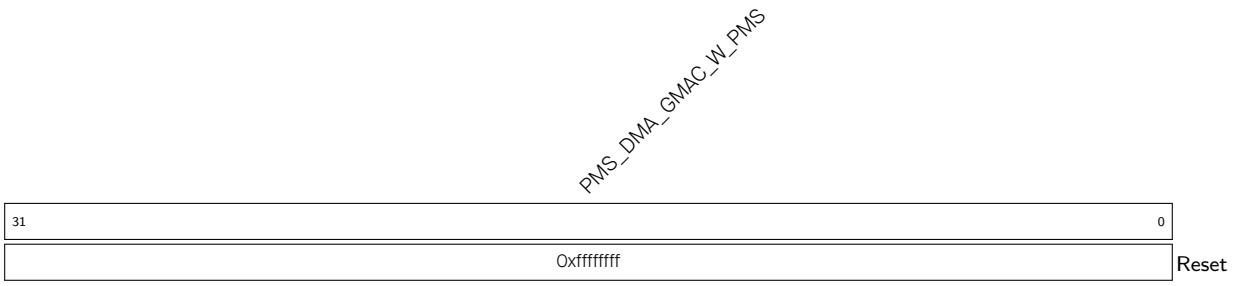
PMS_DMA_GMAC_R_PMS Configures read permission for EMAC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.38. PMS_DMA_GMAC_PMS_W_REG (0x01B8)



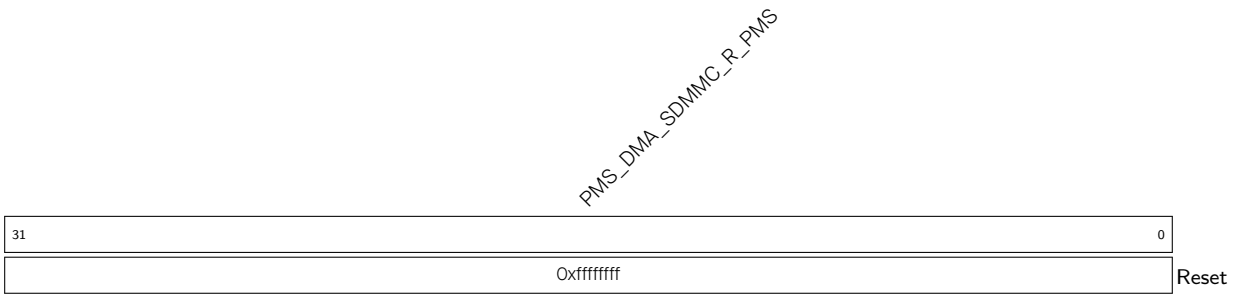
PMS_DMA_GMAC_W_PMS Configures write permission for EMAC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.39. PMS_DMA_SDMMC_PMS_R_REG (0x01BC)



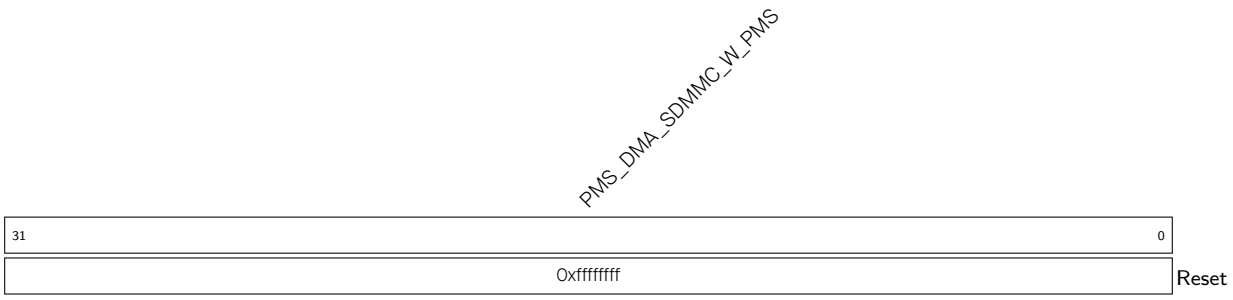
PMS_DMA_SDMMC_R_PMS Configures read permission for SDMMC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.40. PMS_DMA_SDMMC_PMS_W_REG (0x01C0)



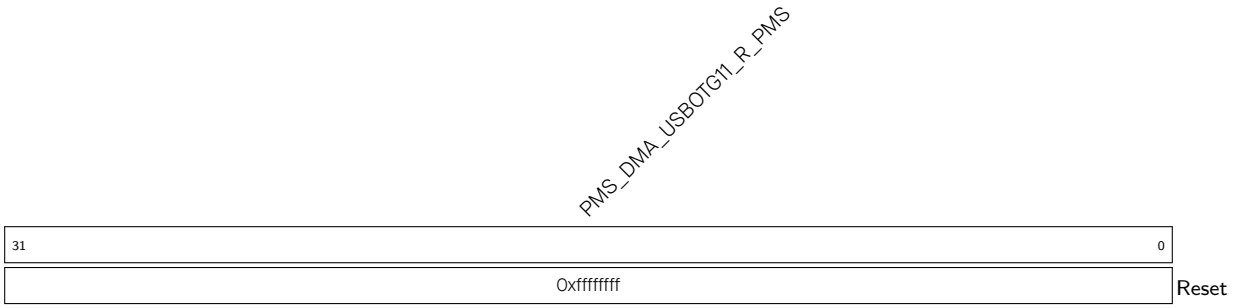
PMS_DMA_SDMMC_W_PMS Configures write permission for SDMMC to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.41. PMS_DMA_USBOTG11_PMS_R_REG (0x01C4)



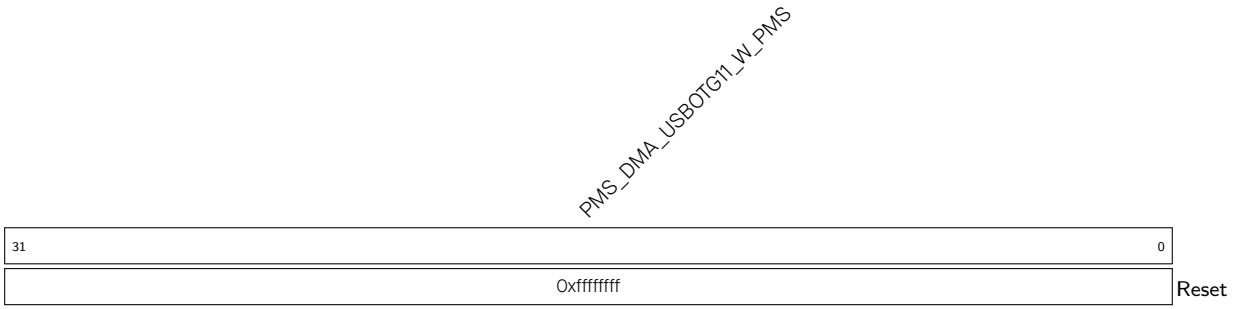
PMS_DMA_USBOTG11_R_PMS Configures read permission for full-speed USB 2.0 OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.42. PMS_DMA_USBOTG11_PMS_W_REG (0x01C8)



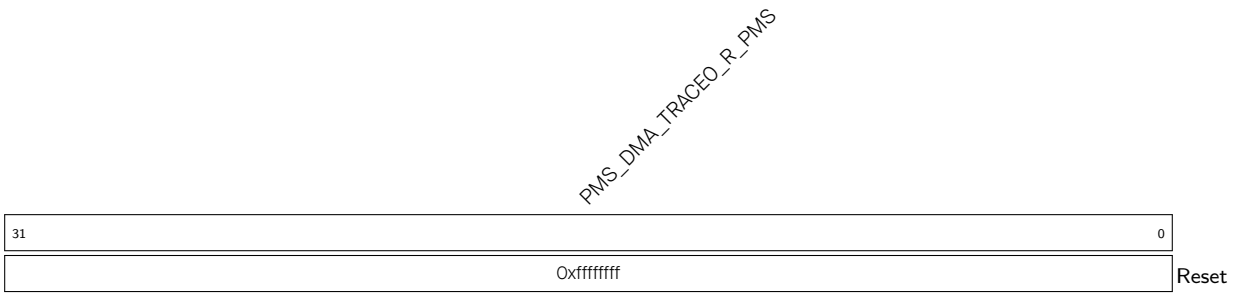
PMS_DMA_USBOTG11_W_PMS Configures write permission for full-speed USB 2.0 OTG to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.43. PMS_DMA_TRACE0_PMS_R_REG (0x01CC)



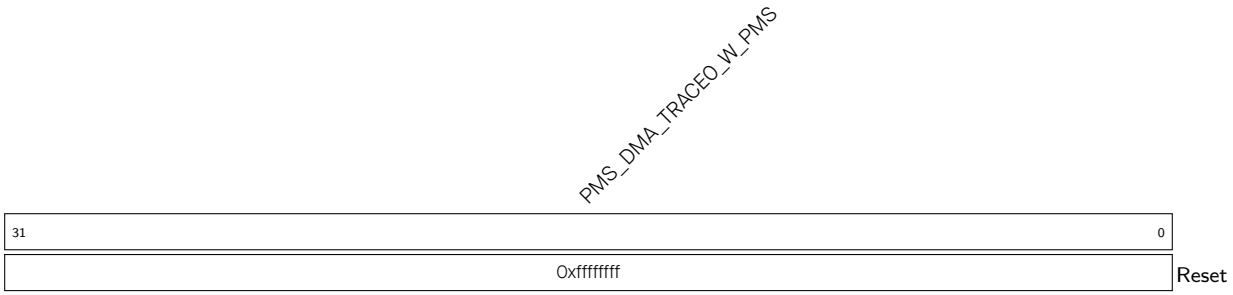
PMS_DMA_TRACE0_R_PMS Configures read permission for TRACE0 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.44. PMS_DMA_TRACE0_PMS_W_REG (0x01D0)



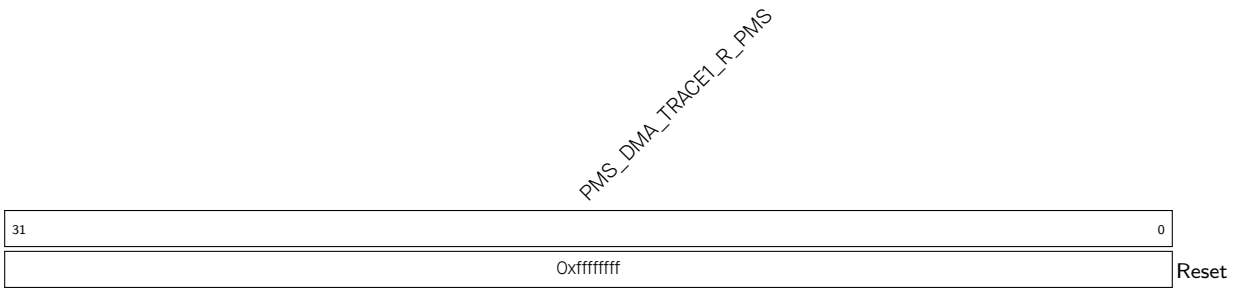
PMS_DMA_TRACE0_W_PMS Configures write permission for TRACE0 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.45. PMS_DMA_TRACE1_PMS_R_REG (0x01D4)



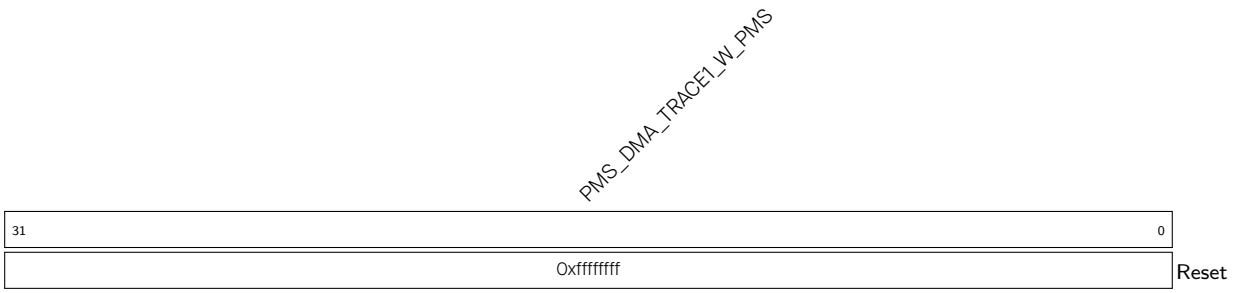
PMS_DMA_TRACE1_R_PMS Configures read permission for TRACE1 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.46. PMS_DMA_TRACE1_PMS_W_REG (0x01D8)



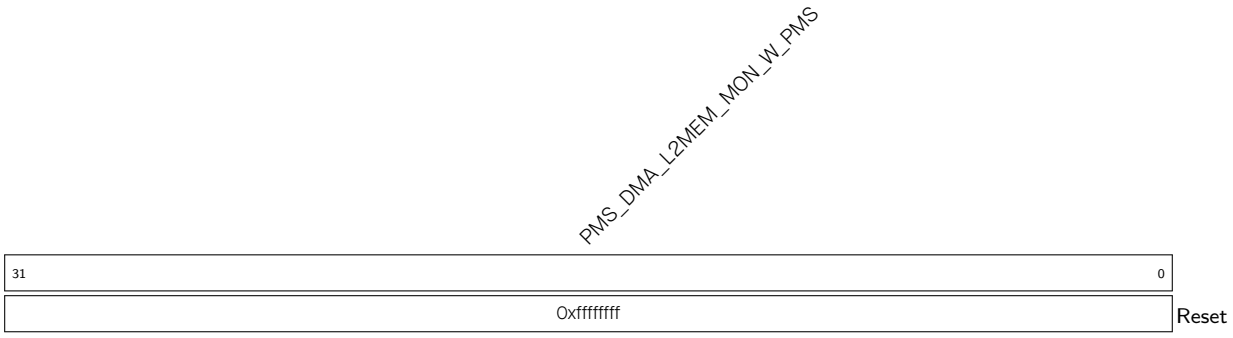
PMS_DMA_TRACE1_W_PMS Configures write permission for TRACE1 to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.47. PMS_DMA_L2MEM_MON_PMS_W_REG (0x01E0)



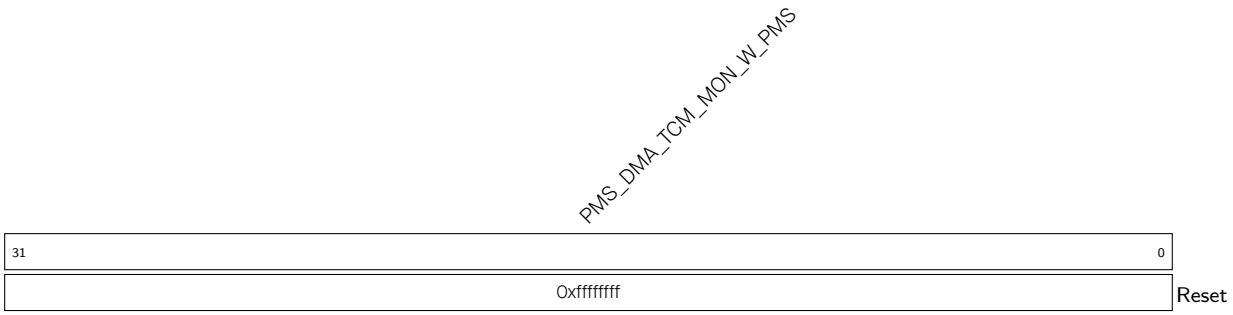
PMS_DMA_L2MEM_MON_W_PMS Configures write permission for L2MEM monitor to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.48. PMS_DMA_TCM_MON_PMS_W_REG (0x01E8)



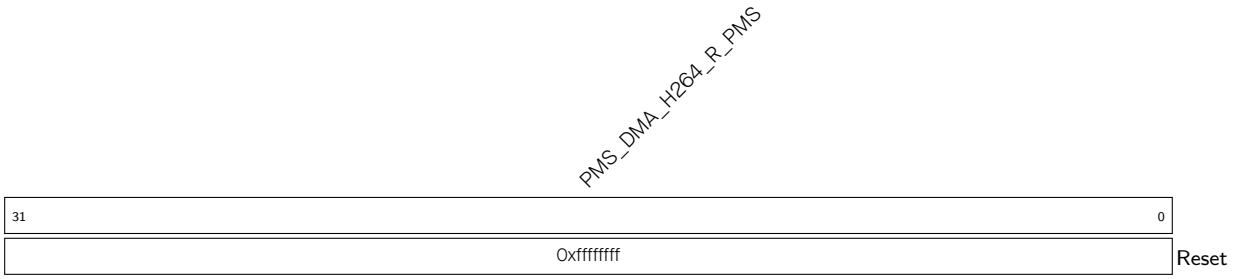
PMS_DMA_TCM_MON_W_PMS Configures write permission for TCM monitor to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.49. PMS_DMA_H264_PMS_R_REG (0x01FC)



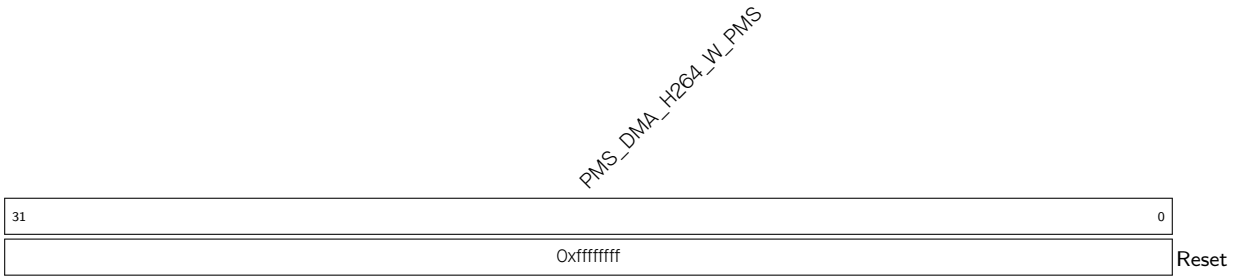
PMS_DMA_H264_R_PMS Configures read permission for H264 DMA to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.50. PMS_DMA_H264_PMS_W_REG (0x0200)



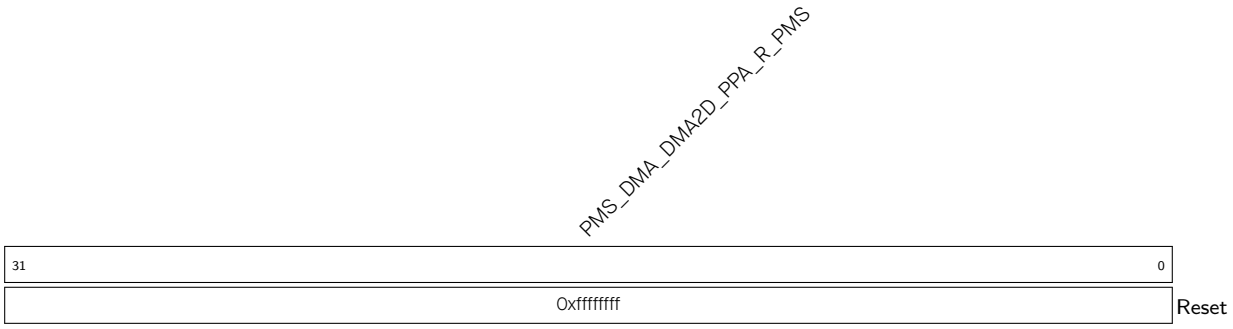
PMS_DMA_H264_W_PMS Configures write permission for H264 DMA to access 32 address ranges. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.51. PMS_DMA_DMA2D_PPA_PMS_R_REG (0x0204)



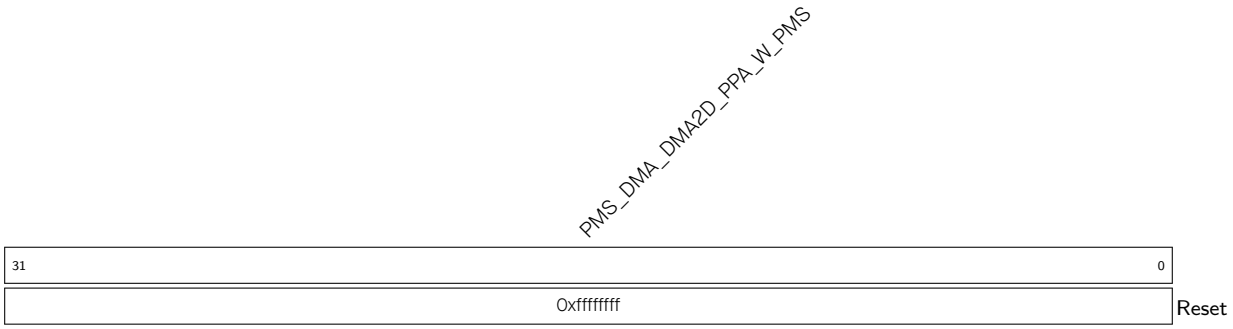
PMS_DMA_DMA2D_PPA_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by PPA (Pixel-Processing Accelerator). Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.52. PMS_DMA_DMA2D_PPA_PMS_W_REG (0x0208)



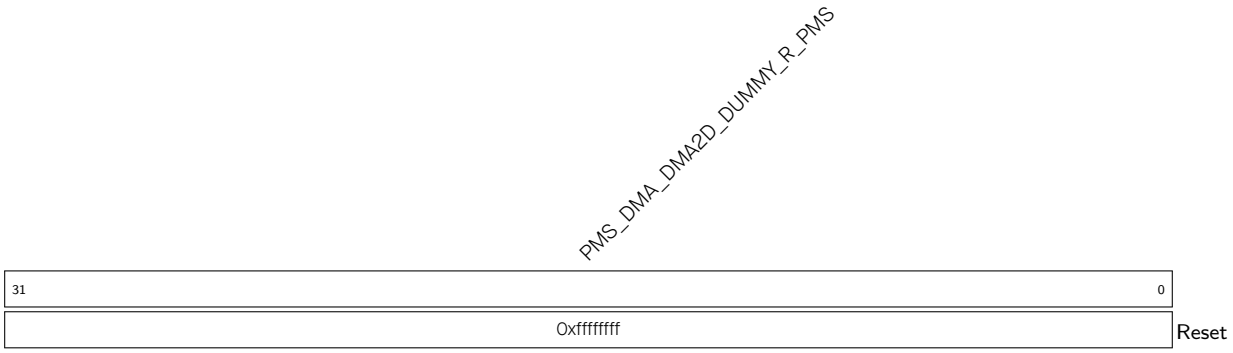
PMS_DMA_DMA2D_PPA_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by PPA. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.53. PMS_DMA_DMA2D_DUMMY_PMS_R_REG (0x020C)



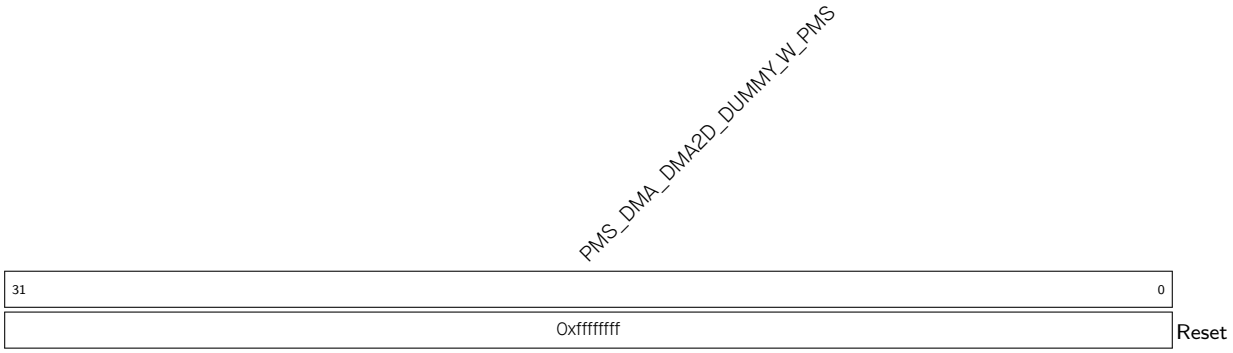
PMS_DMA_DMA2D_DUMMY_R_PMS Configures 2D-DMA permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.54. PMS_DMA_DMA2D_DUMMY_PMS_W_REG (0x0210)



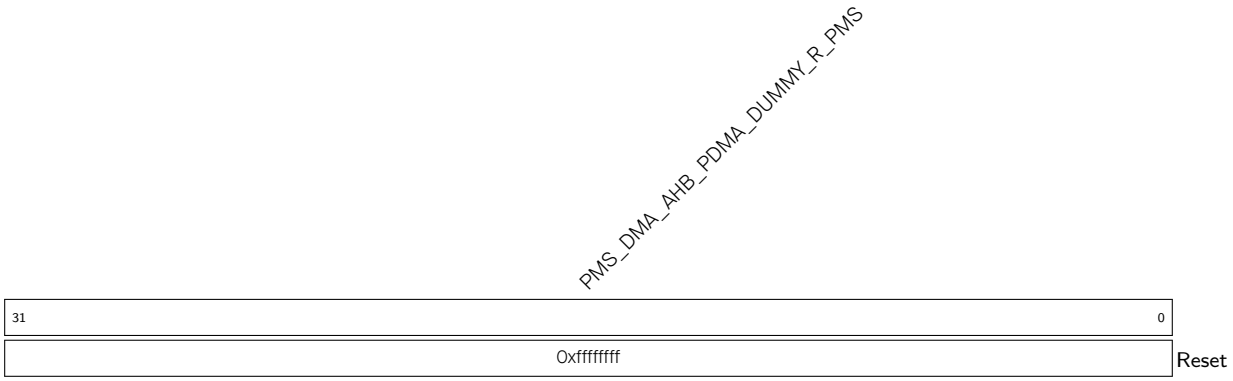
PMS_DMA_DMA2D_DUMMY_W_PMS Configures 2D-DMA permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.55. PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG (0x0214)



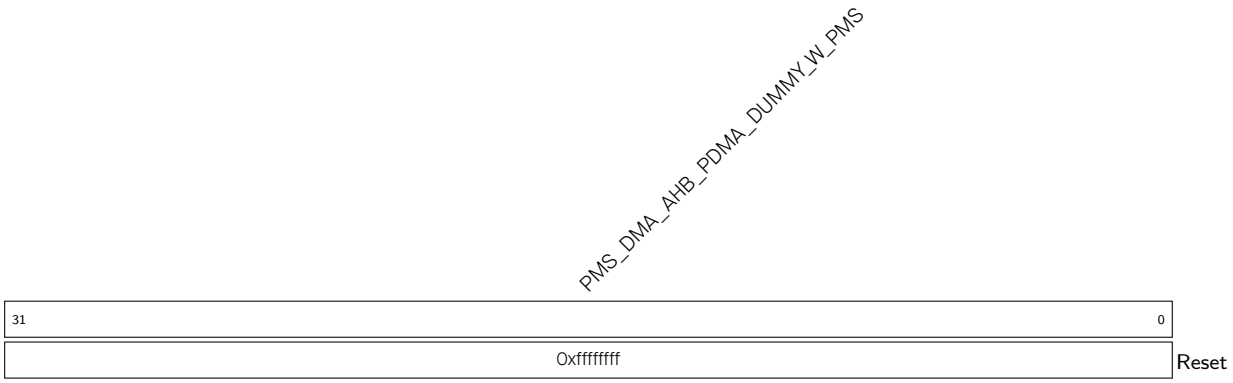
PMS_DMA_AHB_PDMA_DUMMY_R_PMS Configures GDMA-AHB permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.56. PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG (0x0218)



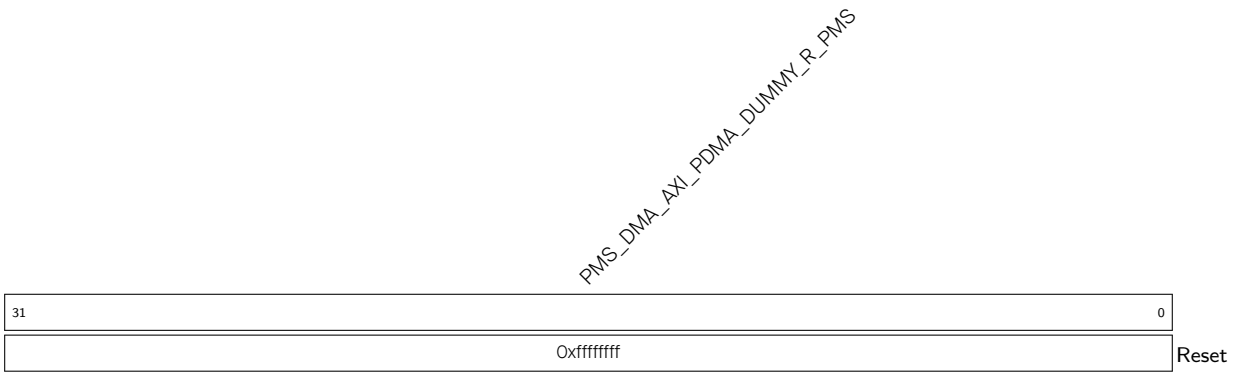
PMS_DMA_AHB_PDMA_DUMMY_W_PMS Configures GDMA-AHB permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

(R/W)

Register 16.57. PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG (0x021C)



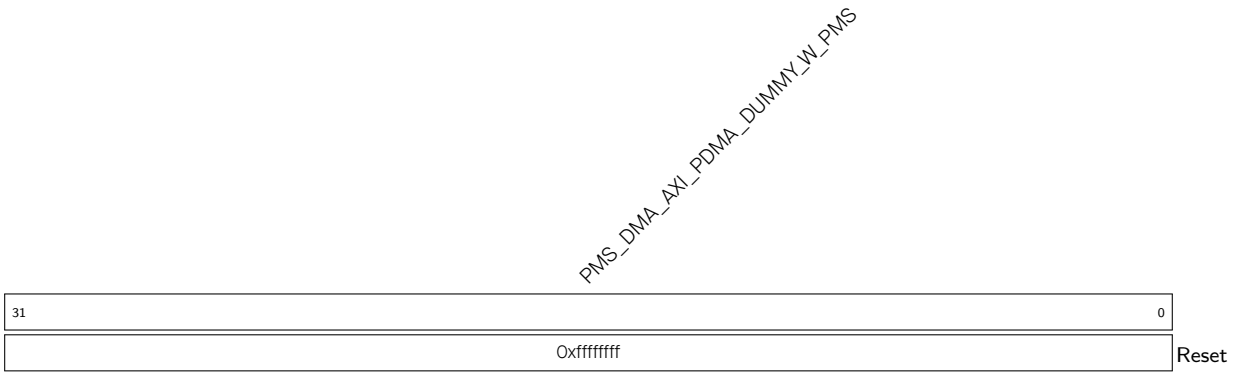
PMS_DMA_AXI_PDMA_DUMMY_R_PMS Configures GDMA-AXI permission to read 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable read permission.

1: Enable read permission.

(R/W)

Register 16.58. PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG (0x0220)



PMS_DMA_AXI_PDMA_DUMMY_W_PMS Configures GDMA-AXI permission to write 32 address ranges requested by Dummy. Bit 0 corresponds to region0, and so on.

0: Disable write permission.

1: Enable write permission.

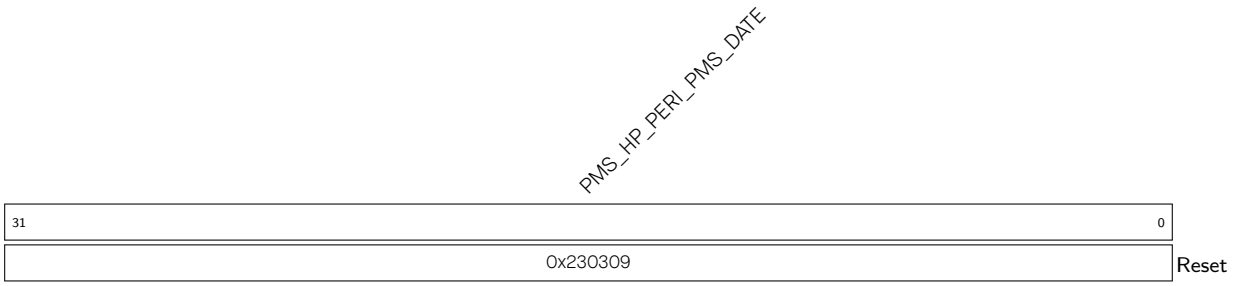
(R/W)

16.7.2 HP_PERI_PMS_REG

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

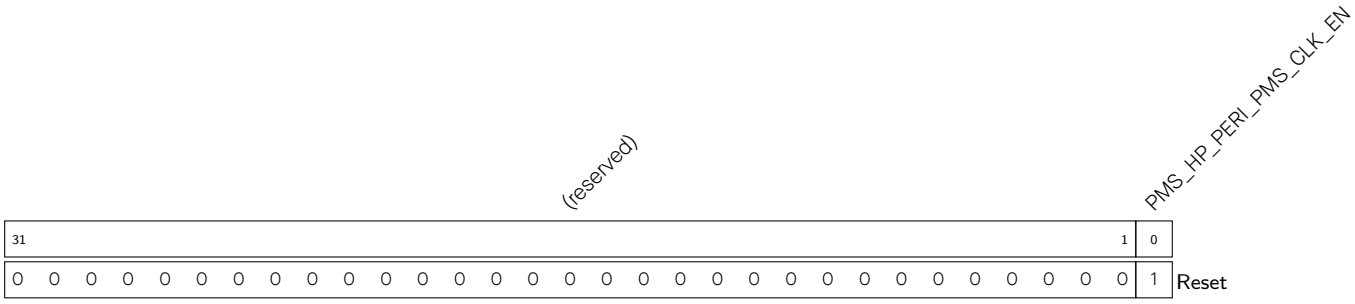
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 16.59. PMS_HP_PERI_PMS_DATE_REG (0x0000)



PMS_HP_PERI_PMS_DATE Version control register. (R/W)

Register 16.60. PMS_HP_PERI_PMS_CLK_EN_REG (0x0004)



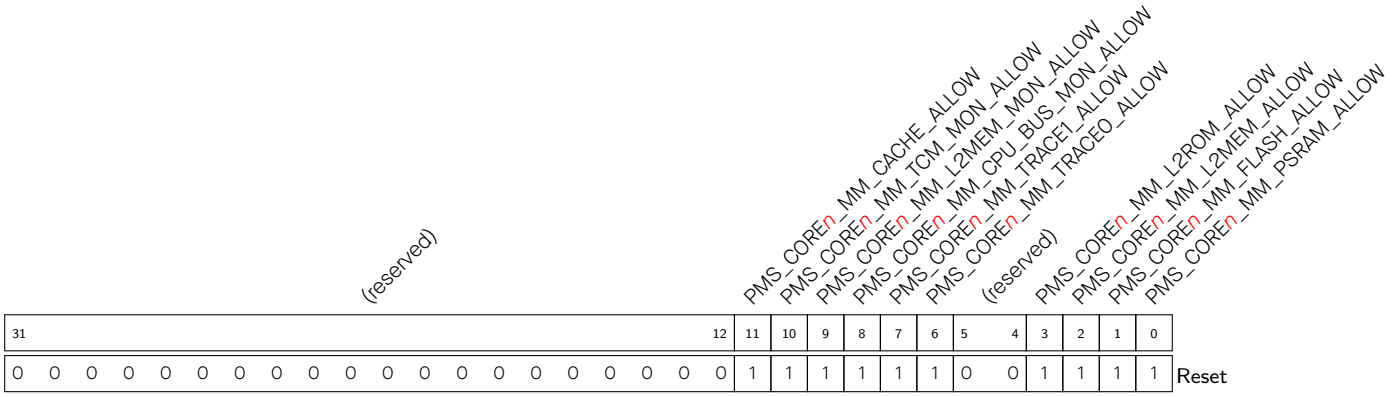
PMS_HP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 16.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20**n*)



PMS_CORE*n*_MM_PSRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_FLASH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_L2MEM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP L2MEM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_L2ROM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_TRACE0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_TRACE1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20**n*)

Continued from the previous page...

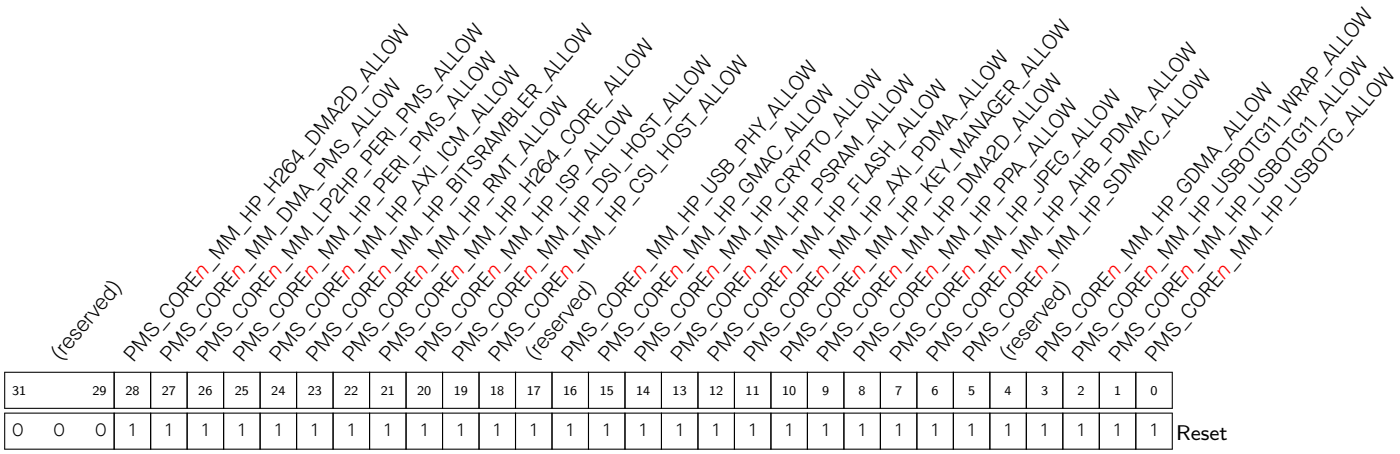
PMS_CORE*n*_MM_CPU_BUS_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access CPU bus monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_L2MEM_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access L2MEM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_TCM_MON_ALLOW Configures whether HP CPU*n* in machine mode has permission to access TCM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_CACHE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access cache.
0: Not allowed
1: Allowed
(R/W)

Register 16.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20n*)**



PMS_CORE*n*_MM_HP_USBOTG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP high-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP full-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_WRAP_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP full-speed USB 2.0 OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_GDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP DW-GDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SDMMC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_AHB_PDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access GDMA-AHB.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_JPEG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PPA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_DMA2D_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_KEY_MANAGER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP key manager.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_AXI_PDMA_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_FLASH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_PSRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_CRYPTO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP CRYPTO (including AES/SHA/RSA/HMAC Accelerators).
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GMAC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_USB_PHY_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP high-speed USB 2.0 OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_CSI_HOST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_DSI_HOST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_ISP_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ISP (Image Signal Processor).
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_H264_CORE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_RMT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_BITSRAMBLER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_AXI_ICM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_LP2HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_DMA_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

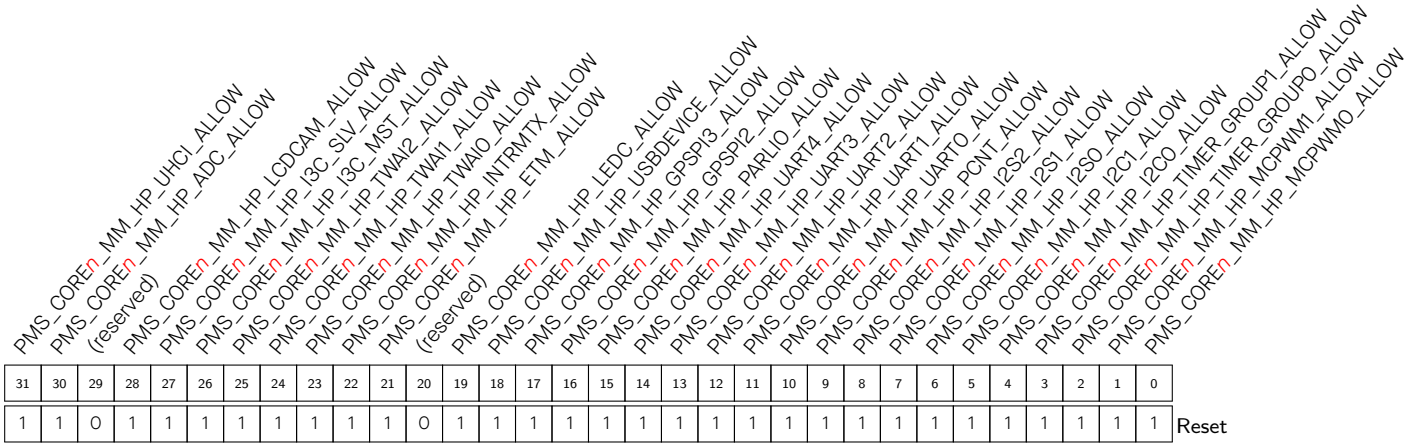
PMS_CORE*n*_MM_HP_H264_DMA2D_ALLOW Configures whether HP CPU*n* in machine mode has permission to access 2D-DMA.

0: Not allowed

1: Allowed

(R/W)

Register 16.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0010+0x20**n*)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_CORE*n*_MM_HP_MCPWM0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_MCPWM1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_I2C0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_I2C1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_I2S0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I2S1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I2S2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PCNT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_UART3_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_UART4_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_PARLIO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GPSPI2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_GPSPI3_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_USBDEVICE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP USB Serial/JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_LEDC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_ETM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ETM (Event Task Matrix).
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_INTRMTX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI0_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI1_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_TWAI2_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I3C_MST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_MM_HP_I3C_SLV_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_MM_HP_LCDCAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP LCD_CAM.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_ADC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP ADC.

0: Not allowed

1: Allowed

(R/W)

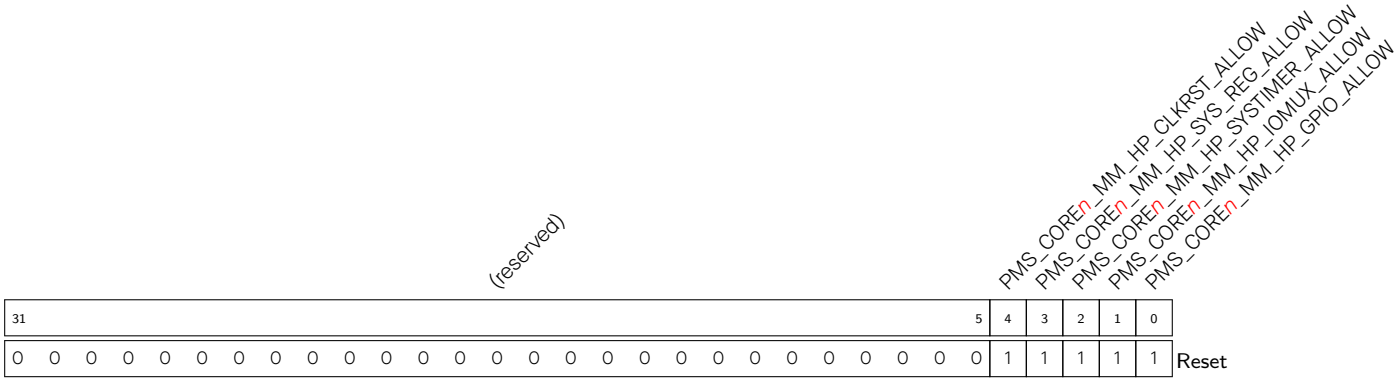
PMS_CORE*n*_MM_HP_UHCI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP UHCI.

0: Not allowed

1: Allowed

(R/W)

Register 16.64. PMS_CORE*n*_MM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0014+0x20n*)**



PMS_CORE*n*_MM_HP_GPIO_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP GPIO Matrix.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_IOMUX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP IO MUX.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SYSTIMER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP system timer.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_MM_HP_SYS_REG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP system register.

0: Not allowed

1: Allowed

(R/W)

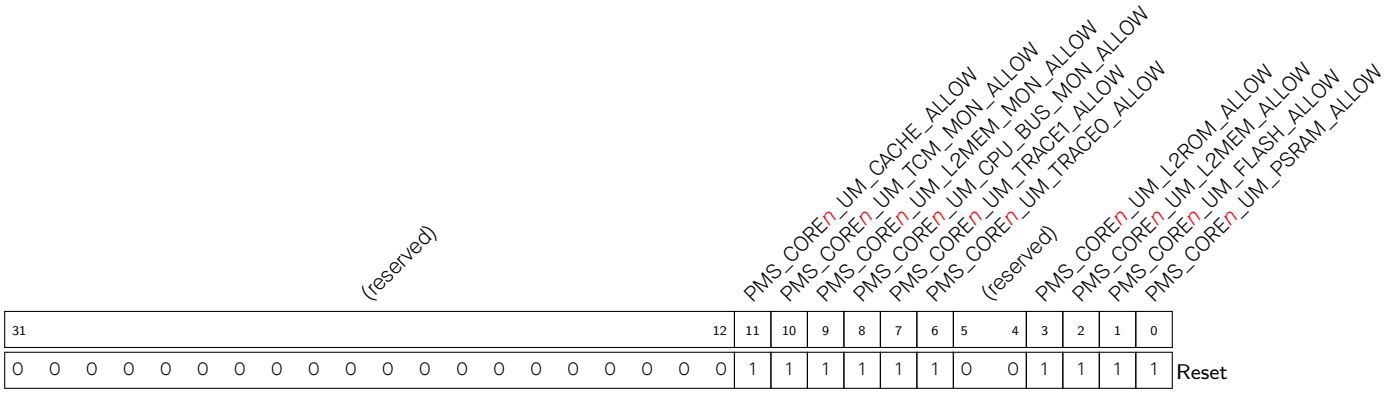
PMS_CORE*n*_MM_HP_CLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP_SYS_CLKRST.

0: Not allowed

1: Allowed

(R/W)

Register 16.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0018+0x20*n)



PMS_CORE*n*_UM_PSRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_FLASH_ALLOW Configures whether HP CPU*n* in user mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_L2MEM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP L2MEM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_L2ROM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_TRACE0_ALLOW Configures whether HP CPU*n* in user mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_TRACE1_ALLOW Configures whether HP CPU*n* in user mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_CPU_BUS_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access CPU bus monitor.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_L2MEM_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access L2MEM monitor.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_TCM_MON_ALLOW Configures whether HP CPU*n* in user mode has permission to access TCM monitor.

0: Not allowed

1: Allowed

(R/W)

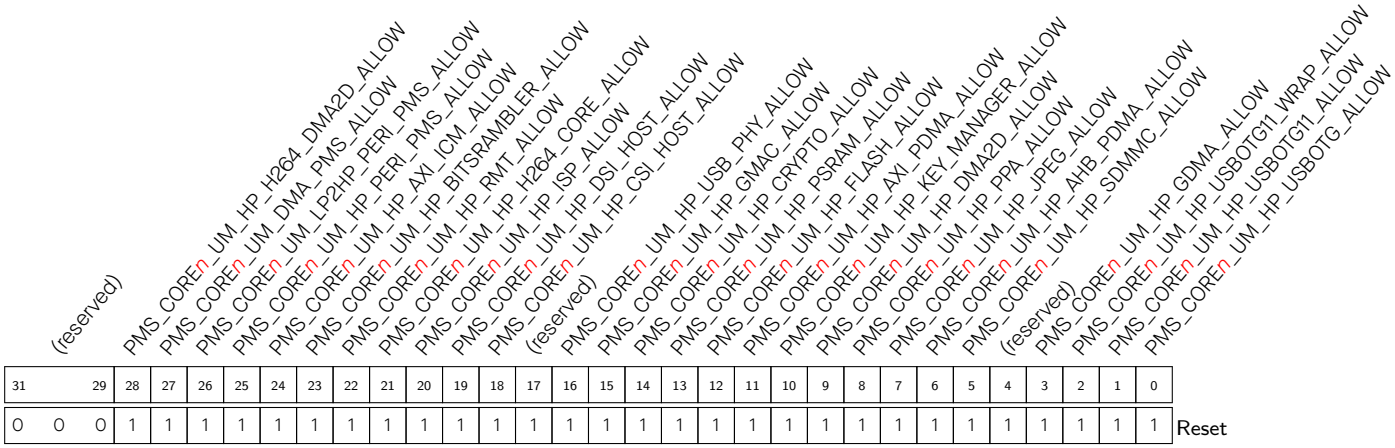
PMS_CORE*n*_UM_CACHE_ALLOW Configures whether HP CPU*n* in user mode has permission to access cache.

0: Not allowed

1: Allowed

(R/W)

Register 16.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x001C+0x20n*)**



PMS_CORE*n*_UM_HP_USBOTG_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP high-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP full-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_WRAP_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP full-speed USB 2.0 OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_GDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP DW-GDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_SDMMC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_AHB_PDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access GDMA-AHB.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_JPEG_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PPA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_DMA2D_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_KEY_MANAGER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP key manager.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_AXI_PDMA_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_FLASH_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_PSRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_CRYPTO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP CRYPTO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GMAC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_USB_PHY_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP high-speed USB 2.0 OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_CSI_HOST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_DSI_HOST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_ISP_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ISP.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_H264_CORE_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_RMT_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_BITSRAMBLER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_AXI_ICM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_LP2HP_PERI_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_DMA_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

Continued from the previous page...

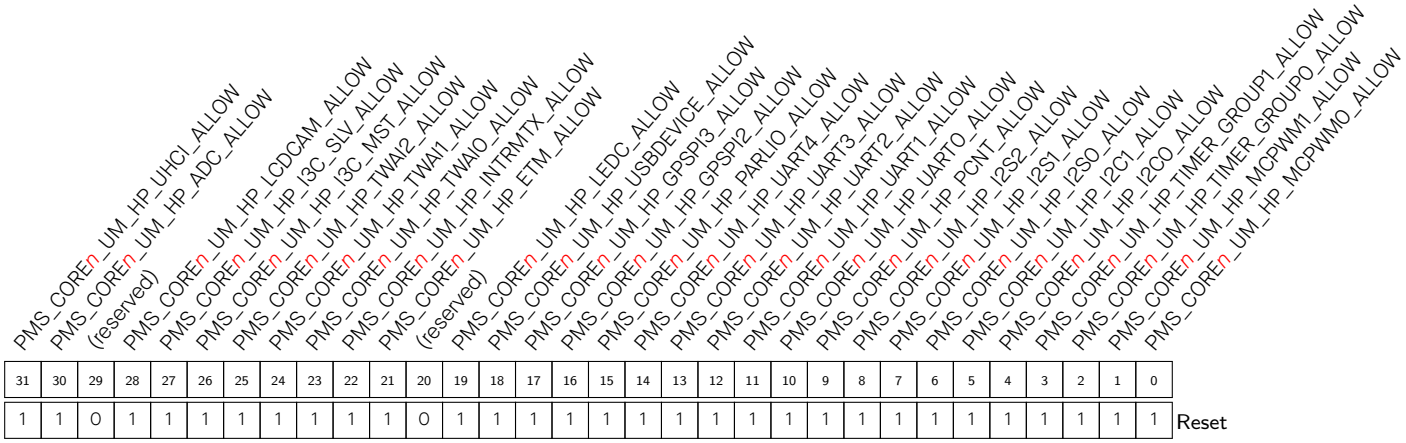
PMS_CORE*n*_UM_HP_H264_DMA2D_ALLOW Configures whether HP CPU*n* in user mode has permission to access 2D-DMA.

0: Not allowed

1: Allowed

(R/W)

Register 16.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20n*)**



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_CORE*n*_UM_HP_MCPWM0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_MCPWM1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_I2C0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_CORE*n*_UM_HP_I2C1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_I2S0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I2S1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I2S2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PCNT_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_UART3_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UART4_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_PARLIO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GPSPI2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_GPSPI3_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_USBDEVICE_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP USB/Serial JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_LEDC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

Continued from the previous page...

PMS_CORE*n*_UM_HP_ETM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ETM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_INTRMTX_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI0_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI1_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_TWAI2_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I3C_MST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_I3C_SLV_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

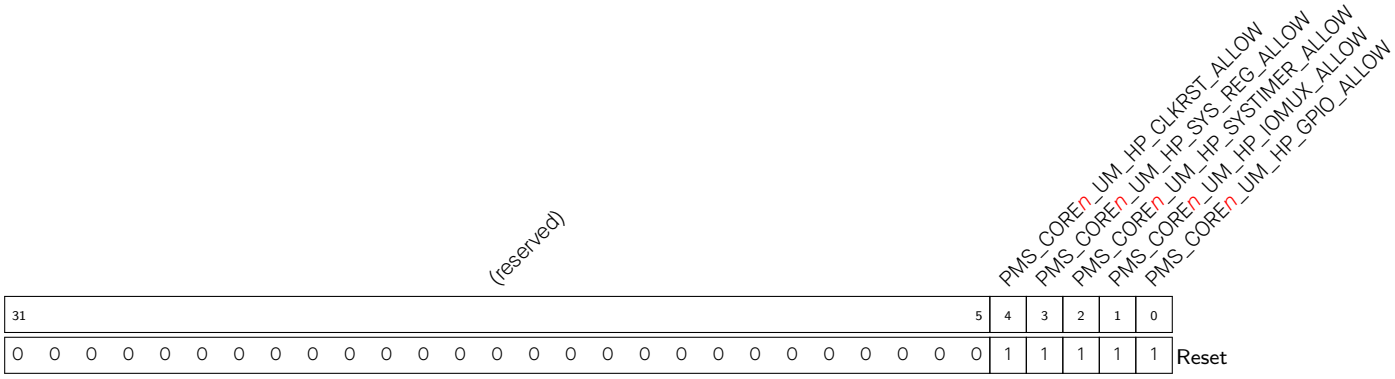
Continued from the previous page...

PMS_CORE*n*_UM_HP_LCDCAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP LCD_CAM.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_ADC_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_UHCI_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP UHCI.
0: Not allowed
1: Allowed
(R/W)

Register 16.68. PMS_CORE*n*_UM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0024+0x20*n)



PMS_CORE*n*_UM_HP_GPIO_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP GPIO Matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_IOMUX_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_SYSTIMER_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP system timer.
0: Not allowed
1: Allowed
(R/W)

PMS_CORE*n*_UM_HP_SYS_REG_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP system register.
0: Not allowed
1: Allowed
(R/W)

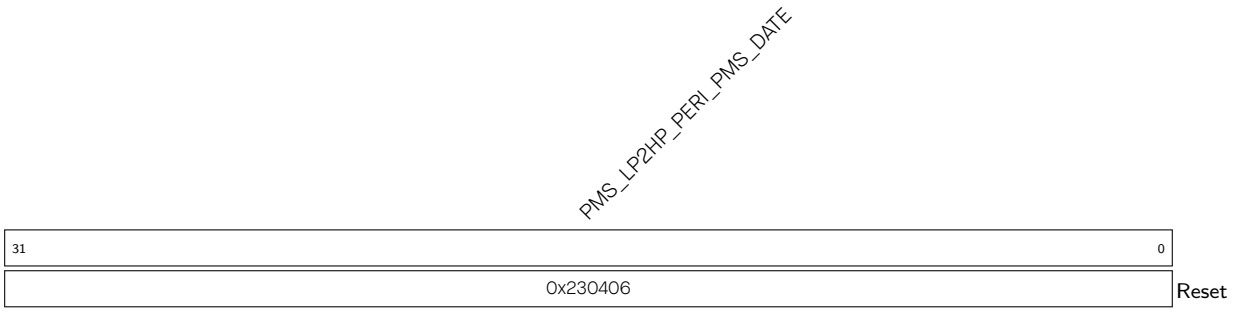
PMS_CORE*n*_UM_HP_CLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP_SYS_CLKRST.
0: Not allowed
1: Allowed
(R/W)

16.7.3 LP2HP_PERI_PMS_REG

The addresses in this section are relative to the LP2HP_PERI_PMS base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

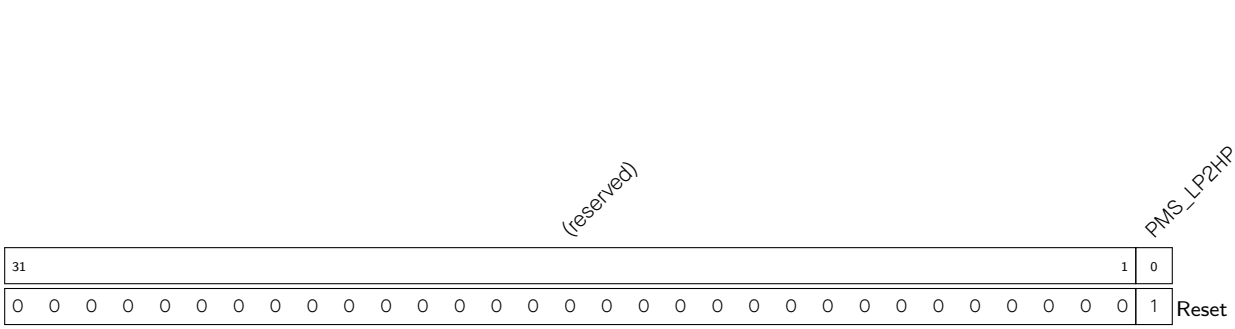
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 16.69. PMS_LP2HP_PERI_PMS_DATE_REG (0x0000)



PMS_LP2HP_PERI_PMS_DATE Version control register. (R/W)

Register 16.70. PMS_LP2HP_PERI_PMS_CLK_EN_REG (0x0004)



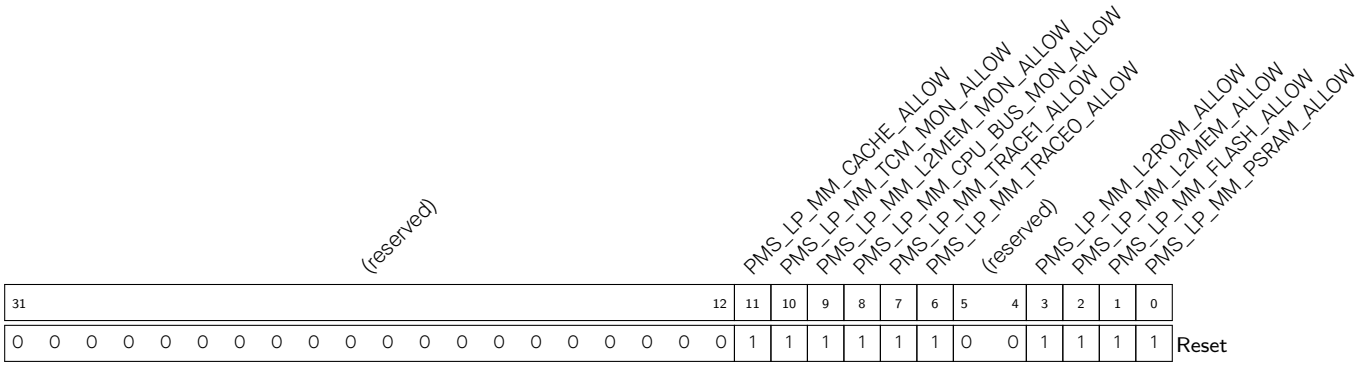
PMS_LP2HP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating.

1: Keep the clock always on.

(R/W)

Register 16.71. PMS_LP_MM_PMS_REG0_REG (0x0008)



PMS_LP_MM_PSRAM_ALLOW Configures whether the LP CPU in machine mode has permission to access external RAM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_FLASH_ALLOW Configures whether the LP CPU in machine mode has permission to access external flash without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_L2MEM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP L2M2M without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_L2ROM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ROM without going through cache.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_TRACE0_ALLOW Configures whether the LP CPU in machine mode has permission to access TRACE0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_TRACE1_ALLOW Configures whether the LP CPU in machine mode has permission to access TRACE1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.71. PMS_LP_MM_PMS_REG0_REG (0x0008)

Continued from the previous page...

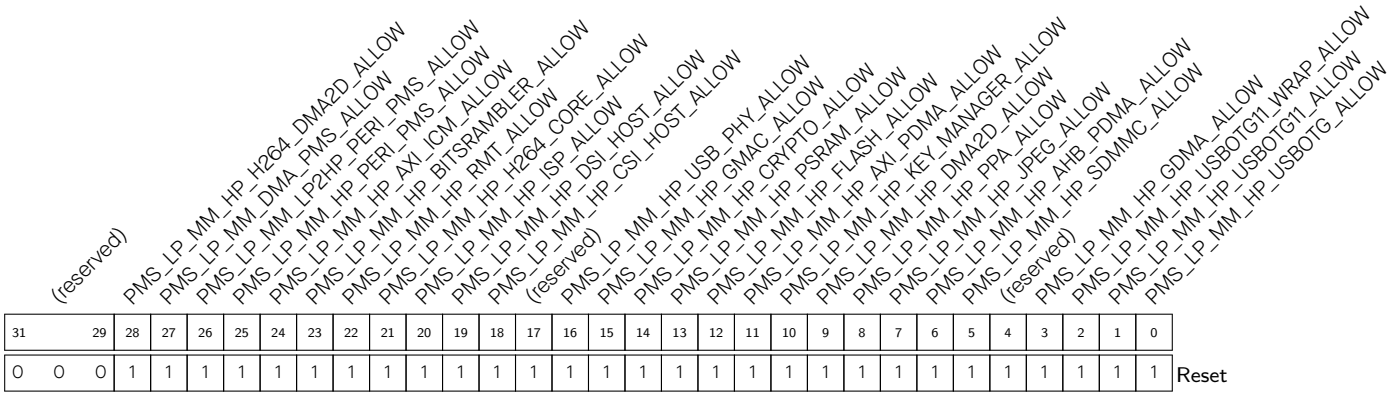
PMS_LP_MM_CPU_BUS_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access CPU bus monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_L2MEM_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access L2MEM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_TCM_MON_ALLOW Configures whether the LP CPU in machine mode has permission to access TCM monitor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_CACHE_ALLOW Configures whether the LP CPU in machine mode has permission to access cache.
0: Not allowed
1: Allowed
(R/W)

Register 16.72. PMS_LP_MM_PMS_REG1_REG (0x0030)



PMS_LP_MM_HP_USBOTG_ALLOW Configures whether the LP CPU in machine mode has permission to access HP high-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_USBOTG11_ALLOW Configures whether the LP CPU in machine mode has permission to access HP full-speed USB 2.0 OTG.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_USBOTG11_WRAP_ALLOW Configures whether the LP CPU in machine mode has permission to access HP full-speed USB 2.0 OTG's wrap.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_GDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP DW-GDMA.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SDMMC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP SDMMC.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_AHB_PDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access GDMA-AHB.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_JPEG_ALLOW Configures whether the LP CPU in machine mode has permission to access HP JPEG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PPA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PPA.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_DMA2D_ALLOW Configures whether the LP CPU in machine mode has permission to access HP 2D-DMA.
0: Not allowed 1: Allowed
(R/W)

PMS_LP_MM_HP_KEY_MANAGER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP key manager.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_AXI_PDMA_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GDMA-AXI.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_FLASH_ALLOW Configures whether the LP CPU in machine mode has permission to access HP flash MSPI controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PSRAM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PSRAM MSPI controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_CRYPTO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP CRYPTO.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_GMAC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP EMAC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_USB_PHY_ALLOW Configures whether the LP CPU in machine mode has permission to access HP high-speed USB 2.0 OTG PHY.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_CSI_HOST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MIPI CSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_DSI_HOST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MIPI DSI host.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_ISP_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ISP.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_H264_CORE_ALLOW Configures whether the LP CPU in machine mode has permission to access HP H264 Encoder.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

Continued from the previous page...

PMS_LP_MM_HP_RMT_ALLOW Configures whether the LP CPU in machine mode has permission to access HP RMT.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_BITSRAMBLER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP bit scrambler.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_AXI_ICM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP AXI ICM.
0: Not allowed
1: Allowed
(R/W)

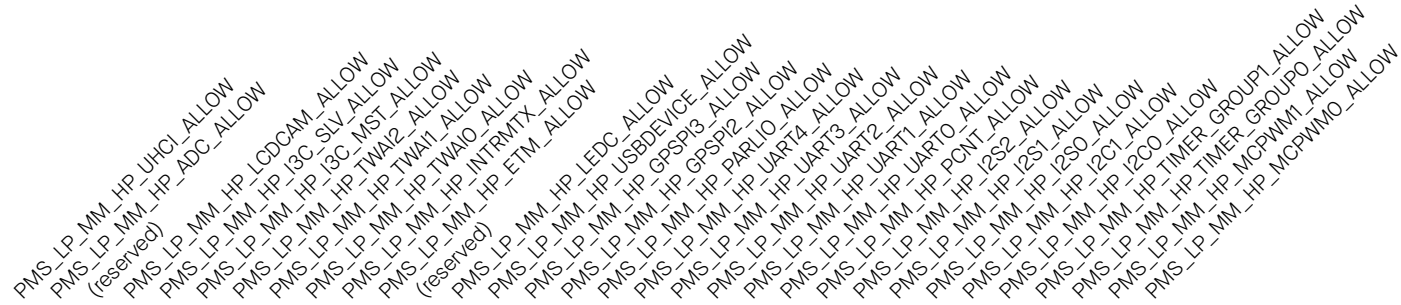
PMS_LP_MM_HP_PERI_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP2HP_PERI_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access LP2HP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_DMA_PMS_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_DMA_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_H264_DMA2D_ALLOW Configures whether the LP CPU in machine mode has permission to access 2D-DMA.
0: Not allowed
1: Allowed
(R/W)

Register 16.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Reset

PMS_LP_MM_HP_MCPWM0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MCPWM0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_MCPWM1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP MCPWM1.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_TIMER_GROUP0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP timer group0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_TIMER_GROUP1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP timer group1.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_I2C0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2C0.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_I2C1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2C1.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_I2S0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I2S1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I2S2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I2S2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PCNT_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PCNT.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART2.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_UART3_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART3.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UART4_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UART4.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_PARLIO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP PARLIO.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_GPSPI2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GP-SPI2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_GPSPI3_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GP-SPI3.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_USBDEVICE_ALLOW Configures whether the LP CPU in machine mode has permission to access HP USB/Serial JTAG Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_LEDC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP LEDC.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

Continued from the previous page...

PMS_LP_MM_HP_ETM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ETM.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_INTRMTX_ALLOW Configures whether the LP CPU in machine mode has permission to access HP interrupt matrix.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI0_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI0.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI1_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI1.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_TWAI2_ALLOW Configures whether the LP CPU in machine mode has permission to access HP TWAI2.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I3C_MST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I3C master controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_I3C_SLV_ALLOW Configures whether the LP CPU in machine mode has permission to access HP I3C slave controller.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

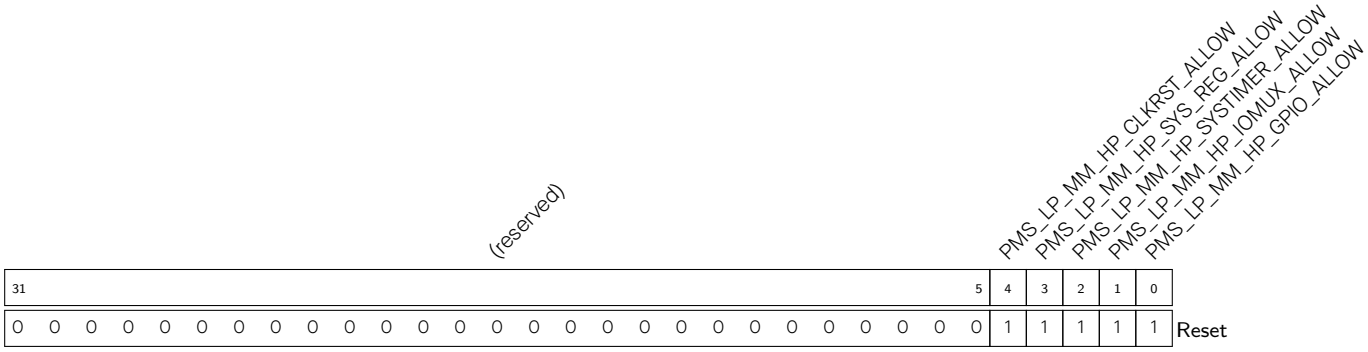
Continued from the previous page...

PMS_LP_MM_HP_LCDCAM_ALLOW Configures whether the LP CPU in machine mode has permission to access HP LCD_CAM.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_ADC_ALLOW Configures whether the LP CPU in machine mode has permission to access HP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP_UHCI_ALLOW Configures whether the LP CPU in machine mode has permission to access HP UHCI.
0: Not allowed
1: Allowed
(R/W)

Register 16.74. PMS_LP_MM_PMS_REG3_REG (0x011C)



PMS_LP_MM_HP_GPIO_ALLOW Configures whether the LP CPU in machine mode has permission to access HP GPIO Matrix.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_IOMUX_ALLOW Configures whether the LP CPU in machine mode has permission to access HP IO MUX.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SYSTIMER_ALLOW Configures whether the LP CPU in machine mode has permission to access HP system timer.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_SYS_REG_ALLOW Configures whether the LP CPU in machine mode has permission to access HP system register.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_HP_CLKRST_ALLOW Configures whether the LP CPU in machine mode has permission to access HP_SYS_CLKRST.

0: Not allowed

1: Allowed

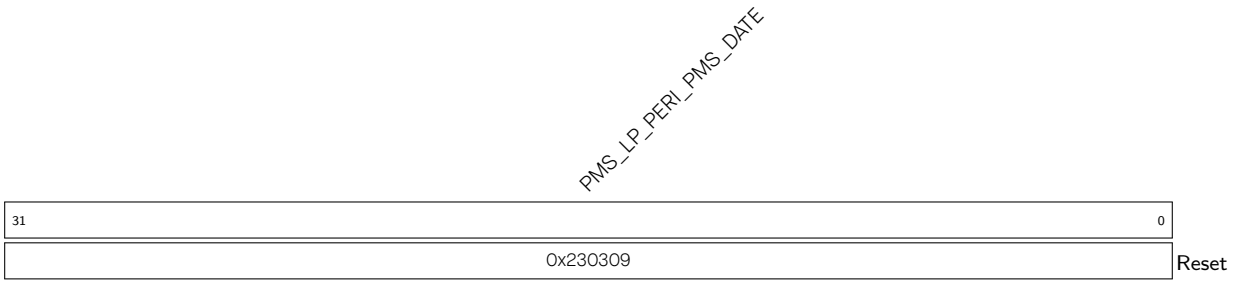
(R/W)

16.7.4 LP_PERI_PMS_REG

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

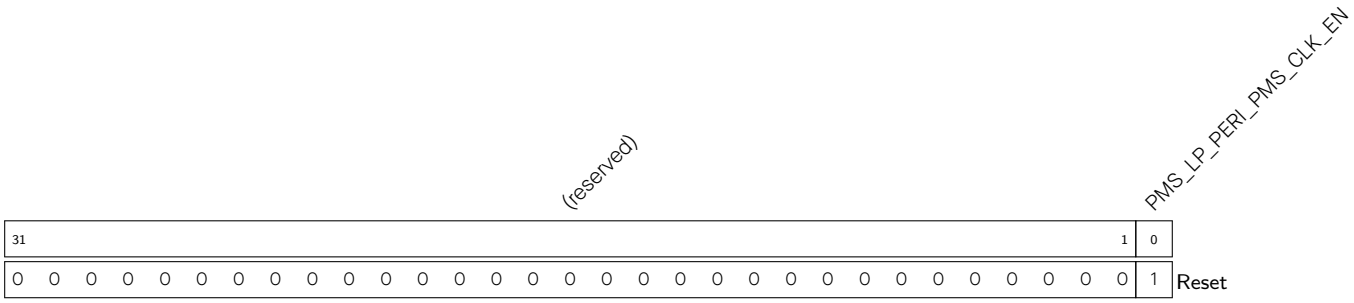
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 16.75. PMS_LP_PERI_PMS_DATE_REG (0x0000)



PMS_LP_PERI_PMS_DATE Version control register (R/W)

Register 16.76. PMS_LP_PERI_PMS_CLK_EN_REG (0x0004)



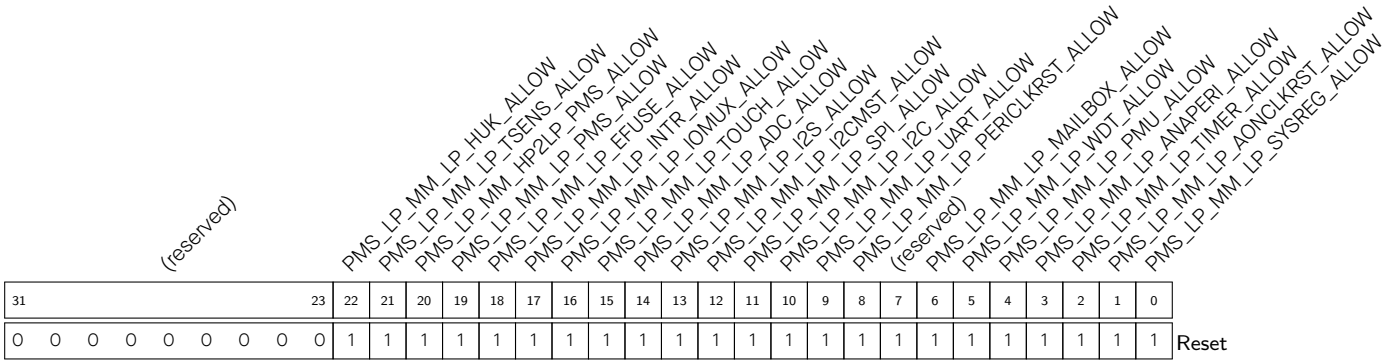
PMS_LP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 16.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)



PMS_LP_MM_LP_SYSREG_ALLOW Configures whether LP CPU in machine mode has permission to access LP system registers.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_AONCLKRST_ALLOW Configures whether LP CPU in machine mode has permission to access LP_AONCLKRST (LP always-on clock and reset).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_TIMER_ALLOW Configures whether LP CPU in machine mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_ANAPERI_ALLOW Configures whether LP CPU in machine mode has permission to access LP ANAPERI (analog peripherals).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_PMU_ALLOW Configures whether LP CPU in machine mode has permission to access LP PMU (Power Management Unit).

0: Not allowed

1: Allowed

(R/W)

PMS_LP_MM_LP_WDT_ALLOW Configures whether LP CPU in machine mode has permission to access LP WDT (watchdog timer).

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

Continued from the previous page...

PMS_LP_MM_LP_MAILBOX_ALLOW Configures whether LP CPU in machine mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_PERICLKRST_ALLOW Configures whether LP CPU in machine mode has permission to access LP PREICLKRST (peripheral clock and reset).
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_UART_ALLOW Configures whether LP CPU in machine mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2C_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_SPI_ALLOW Configures whether LP CPU in machine mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2CMST_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_I2S_ALLOW Configures whether LP CPU in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

Continued from the previous page...

PMS_LP_MM_LP_ADC_ALLOW Configures whether LP CPU in machine mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_TOUCH_ALLOW Configures whether LP CPU in machine mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_IOMUX_ALLOW Configures whether LP CPU in machine mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_INTR_ALLOW Configures whether LP CPU in machine mode has permission to access LP INTR (interrupt).
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_EFUSE_ALLOW Configures whether LP CPU in machine mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_LP_PMS_ALLOW Configures whether LP CPU in machine mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_LP_MM_HP2LP_PMS_ALLOW Configures whether LP CPU in machine mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

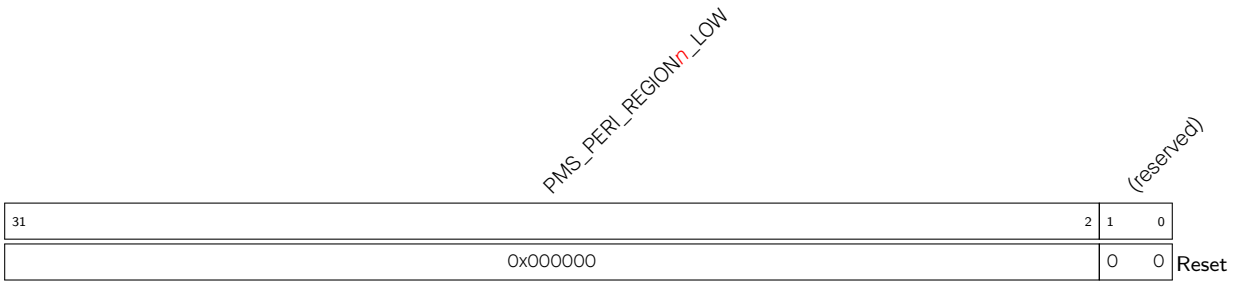
Register 16.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

Continued from the previous page...

PMS_LP_MM_LP_TSENS_ALLOW Configures whether LP CPU in machine mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

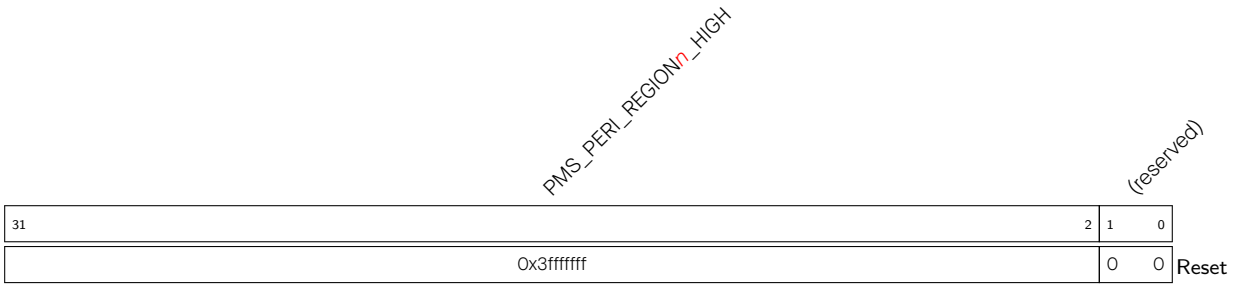
PMS_LP_MM_LP_HUK_ALLOW Configures whether LP CPU in machine mode has permission to access LP HUK (Hardware Unique Key).
0: Not allowed
1: Allowed
(R/W)

Register 16.78. PMS_PERI_REGION*n*_LOW_REG (*n*: 0-1) (0x000C+0x8**n*)



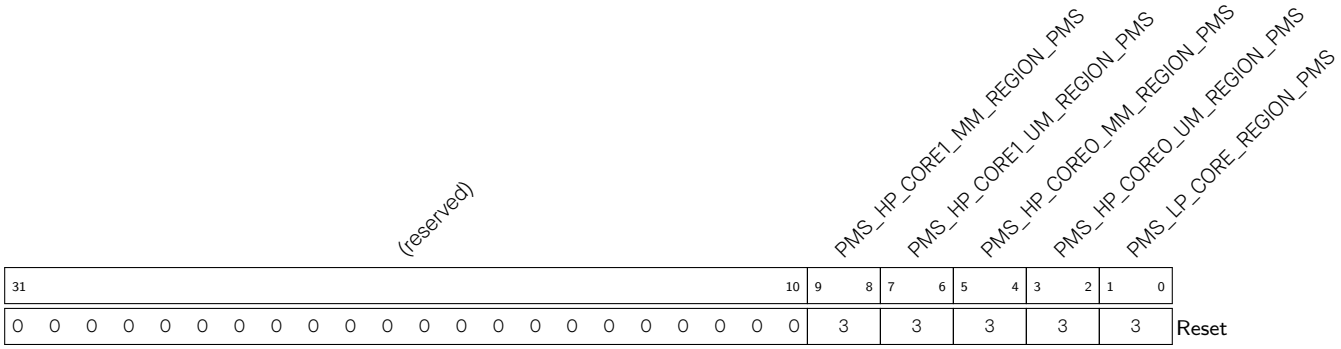
PMS_PERI_REGION*n*_LOW Configures the high 30 bits of the start address of peripheral register's region*n*. (R/W)

Register 16.79. PMS_PERI_REGION*n*_HIGH_REG (*n*: 0-1) (0x0010+0x8**n*)



PMS_PERI_REGION*n*_HIGH Configures the high 30 bits of the end address of peripheral register's region*n*. (R/W)

Register 16.80. PMS_PERI_REGION_PMS_REG (0x001C)



PMS_LP_CORE_REGION_PMS Configures whether LP core in machine mode has permission to access address region0 and address region1. Bit0 corresponds to region0 and bit1 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_UM_REGION_PMS Configures whether HP CPU0 in user mode has permission to access address region0 and address region1. Bit2 corresponds to region0 and bit3 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE0_MM_REGION_PMS Configures whether HP CPU0 in machine mode has permission to access address region0 and address region1. Bit4 corresponds to region0 and bit5 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE1_UM_REGION_PMS Configures whether HP CPU1 in user mode has permission to access address region0 and address region1. Bit6 corresponds to region0 and bit7 corresponds to region1.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE1_MM_REGION_PMS Configures whether HP CPU1 in machine mode has permission to access address region0 and address region1. Bit8 corresponds to region0 and bit9 corresponds to region1.

0: Not allowed

1: Allowed

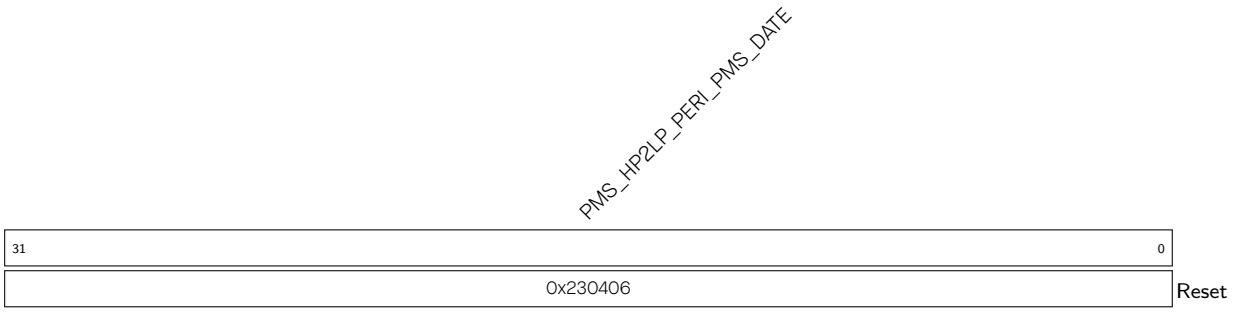
(R/W)

16.7.5 HP2LP_PERI_PMS_REG

The addresses in this section are relative to the H2LP_PERI_PMS base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

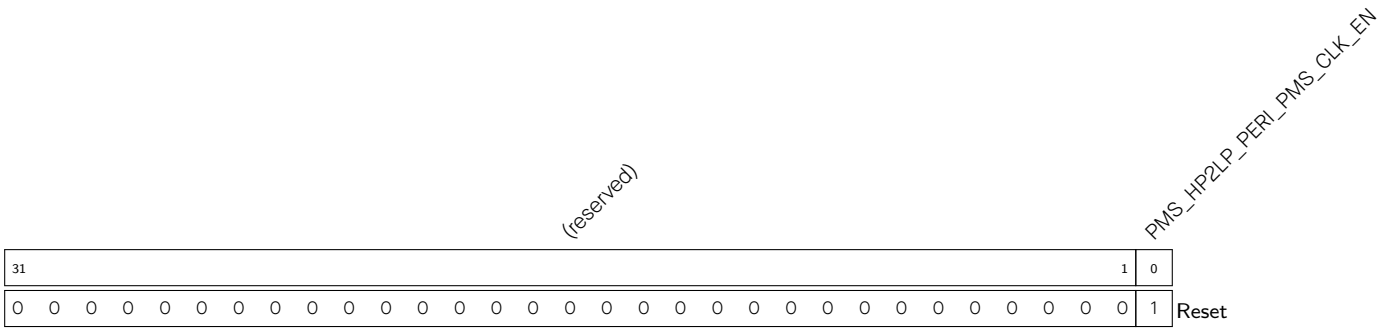
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 16.81. PMS_HP2LP_PERI_PMS_DATE_REG (0x0000)



PMS_HP2LP_PERI_PMS_DATE Version control register (R/W)

Register 16.82. PMS_HP2LP_PERI_PMS_CLK_EN_REG (0x0004)



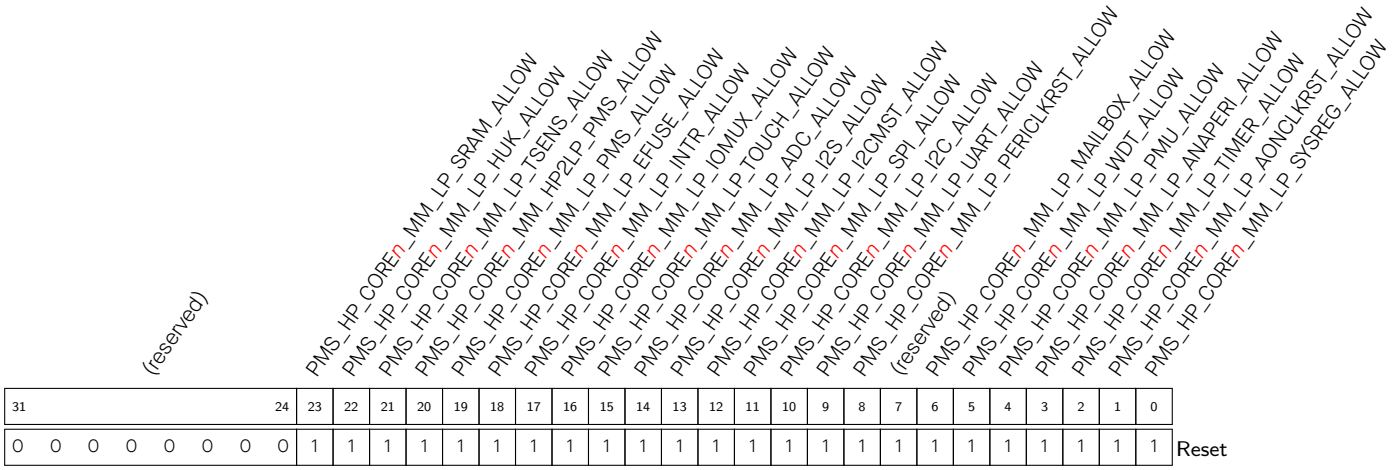
PMS_HP2LP_PERI_PMS_CLK_EN Configures whether to keep the clock always on.

0: Enable automatic clock gating

1: Keep the clock always on

(R/W)

Register 16.83. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8*n*)



PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP System Registers.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_AONCLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP_AONCLKRST.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_TIMER_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_ANAPERI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP ANAPERI.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_MM_LP_PMU_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP PMU.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.83. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_MM_LP_WDT_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP WDT.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_MAILBOX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_PERICLKRST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP PERICLKRST.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_UART_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_I2C_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2C.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_SPI_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_I2CMST_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.83. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_MM_LP_I2S_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_ADC_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_TOUCH_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_IOMUX_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_INTR_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP INTR.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_EFUSE_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.83. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

Continued from the previous page...

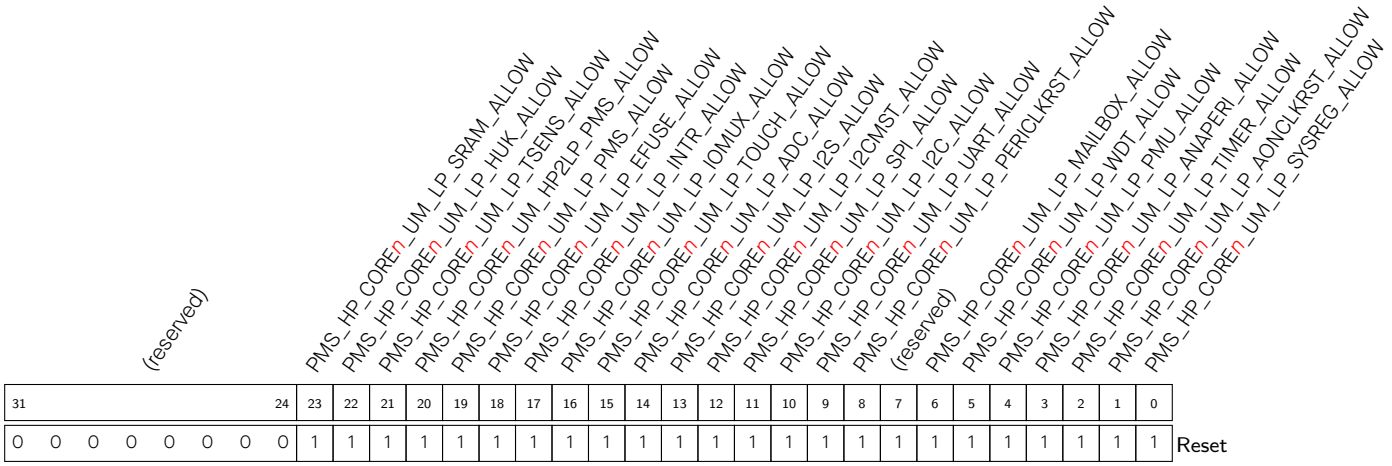
PMS_HP_CORE*n*_MM_HP2LP_PMS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_TSENS_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_HUK_ALLOW Configures whether HP CPU*n* in machine mode has permission to LP HUK.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_MM_LP_SRAM_ALLOW Configures whether HP CPU*n* in machine mode has permission to access LP SRAM.
0: Not allowed
1: Allowed
(R/W)

Register 16.84. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8n*)**



PMS_HP_CORE*n*_UM_LP_SYSREG_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP System Registers.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_AONCLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP_AONCLKRST.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_TIMER_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP timer.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_ANAPERI_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP ANAPERI.

0: Not allowed

1: Allowed

(R/W)

PMS_HP_CORE*n*_UM_LP_PMU_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP PMU.

0: Not allowed

1: Allowed

(R/W)

Continued on the next page...

Register 16.84. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_LP_WDT_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP WDT.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_MAILBOX_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP Mailbox Controller.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_PERICLKRST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP PERICLKRST.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_UART_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP UART.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_I2C_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2C.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_SPI_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP SPI.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_I2CMST_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2C master.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.84. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_LP_I2S_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP I2S.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_ADC_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP ADC.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_TOUCH_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP touch sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_IOMUX_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP IO MUX.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_INTR_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP INTR.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_EFUSE_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP eFuse.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

Continued on the next page...

Register 16.84. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

Continued from the previous page...

PMS_HP_CORE*n*_UM_HP2LP_PMS_ALLOW Configures whether HP CPU*n* in user mode has permission to access HP2LP_PERI_PMS_REG.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_TSENS_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP temperature sensor.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_HUK_ALLOW Configures whether HP CPU*n* in user mode has permission to LP HUK.
0: Not allowed
1: Allowed
(R/W)

PMS_HP_CORE*n*_UM_LP_SRAM_ALLOW Configures whether HP CPU*n* in user mode has permission to access LP SRAM.
0: Not allowed
1: Allowed
(R/W)

Chapter 17

Debug Assistant

17.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

17.2 Features

- **Read/write monitoring**: Monitors whether the High-Performance dual-core CPU (HP CPU0 and HP CPU1) bus reads from or writes to a specified memory address space. A detected read or write in the monitored address space will trigger an interrupt.
- **Stack pointer (SP) monitoring**: Monitors whether the SP exceeds the specified address space. A bounds violation will trigger an interrupt.
- **Program counter (PC) logging**: Records the PC value. The developer can get the last PC value at the most recent reset of HP CPU0 or HP CPU1.
- **Bus access logging**: Records the information about bus access. When the HP CPU0, HP CPU1, or the Direct Memory Access controller (DMA) writes a specified value, the Debug Assistant module will record the data type, address of this write operation, and additionally the PC value when the write is performed by HP CPU0 or HP CPU1, and push such information to the HP L2MEM.

17.3 Functional Description

17.3.1 Region Read/Write Monitoring

The Debug Assistant module can monitor reads/writes performed by HP CPU0 bus in a certain address space, i.e., memory region. Whenever the bus reads or writes in the specified address space, an interrupt will be triggered. Please refer to Table **??** in Chapter 1 *High-Performance CPU [to be added later]*. In this chapter, when HP CPU0 bus accesses the data space, it is referred to as the Data bus, and when HP CPU0 bus accesses the AHB peripheral space, it is referred to as the Peripheral bus. The Data bus can monitor two memory regions (assuming they are HP CPU0 region 0 and HP CPU0 region 1, defined by developers' needs) at the same time, and so can Peripheral bus.

This is also the case for HP CPU1.

17.3.2 SP Monitoring

The Debug Assistant module can monitor the SP so as to prevent stack overflow or erroneous push/pop in HP CPU0. When the HP CPU0 SP goes below or beyond the lower or upper bound of the HP CPU0 SP monitored region, the module will record the PC pointer and generate an interrupt. The bound is configured by software.

This is also the case for HP CPU1.

17.3.3 PC Logging

In some cases, software developers want to know the PC at the last reset of HP CPU0. For instance, when the program is stuck and can only be reset, the developer may want to know where the program got stuck in order to debug. The Debug Assistant module can record the PC at the last reset of HP CPU0, which can be then read for software debugging.

This is also the case for HP CPU1.

17.3.4 CPU/DMA Bus Access Logging

The Debug Assistant module can record the information about the HP CPU0 Data bus's, HP CPU1 bus's, and DMA bus's write behaviors in real time. When a write operation occurs in or a specific value is written to a specified address space, the module will record the bus type, the address, PC (only when the write is performed by the HP CPU0 or HP CPU1, will PC be recorded), and other information, and then store the data in the HP L2MEM in a certain format. The specified address range must fall within the permitted memory regions of HP TCM or HP L2MEM.

17.4 Interrupts

The following interrupt sources from the Debug Assistant module can generate the interrupt signal ASSIST_DEBUG_INT.

- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT: Triggered when the HP CPU0 Data bus reads in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT: Triggered when the HP CPU0 Data bus writes in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT: Triggered when the HP CPU0 Data bus reads in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT: Triggered when the HP CPU0 Data bus writes in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT: Triggered when the HP CPU0 Peripheral bus reads in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT: Triggered when the HP CPU0 Peripheral bus writes in HP CPU0 region 0.
- ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT: Triggered when the HP CPU0 Peripheral bus reads in HP CPU0 region 1.

- ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT: Triggered when the HP CPU0 Peripheral bus writes in HP CPU0 region 1.
- ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT: Triggered when the HP CPU0 SP goes below the lower bound of the HP CPU0 SP monitored region.
- ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT: Triggered when the HP CPU0 SP goes beyond the upper bound of the HP CPU0 SP monitored region.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT: Triggered when the HP CPU1 Data bus reads in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT: Triggered when the HP CPU1 Data bus writes in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT: Triggered when the HP CPU1 Data bus reads in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT: Triggered when the HP CPU1 Data bus writes in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT: Triggered when the HP CPU1 Peripheral bus reads in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT: Triggered when the HP CPU1 Peripheral bus writes in HP CPU1 region 0.
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT: Triggered when the HP CPU1 Peripheral bus reads in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT: Triggered when the HP CPU1 Peripheral bus writes in HP CPU1 region 1.
- ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT: Triggered when the HP CPU1 SP goes below the lower bound of the HP CPU1 SP monitored region.
- ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT: Triggered when the HP CPU1 SP goes beyond the upper bound of the HP CPU1 SP monitored region.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 17.6 *Register Summary*.

17.5 Recommended Operation

17.5.1 Region Monitoring and SP Monitoring Configuration

The Debug Assistant module can monitor reads and writes performed by the Data bus and Peripheral bus of HP CPU0 and HP CPU1. Two memory regions on each bus can be monitored at the same time. All the

monitoring modes supported by the Debug Assistant module are listed below:

- Monitoring the read/write operations performed by Data bus
  - HP CPU0 Data bus reads in HP CPU0 region 0
  - HP CPU0 Data bus writes in HP CPU0 region 0
  - HP CPU0 Data bus reads in HP CPU0 region 1
  - HP CPU0 Data bus writes in HP CPU0 region 1
  - HP CPU1 Data bus reads in HP CPU1 region 0
  - HP CPU1 Data bus writes in HP CPU1 region 0
  - HP CPU1 Data bus reads in HP CPU1 region 1
  - HP CPU1 Data bus writes in HP CPU1 region 1
- Monitoring the read/write operations performed by Peripheral bus
  - HP CPU0 Peripheral bus reads in HP CPU0 region 0
  - HP CPU0 Peripheral bus writes in HP CPU0 region 0
  - HP CPU0 Peripheral bus reads in HP CPU0 region 1
  - HP CPU0 Peripheral bus writes in HP CPU0 region 1
  - HP CPU1 Peripheral bus reads in HP CPU1 region 0
  - HP CPU1 Peripheral bus writes in HP CPU1 region 0
  - HP CPU1 Peripheral bus reads in HP CPU1 region 1
  - HP CPU1 Peripheral bus writes in HP CPU1 region 1
- Monitoring SP bounds violations
  - HP CPU0 SP goes beyond the upper bound of the HP CPU0 SP monitored region
  - HP CPU0 SP goes below the lower bound of the HP CPU0 SP monitored region
  - HP CPU1 SP goes beyond the upper bound of the HP CPU1 SP monitored region
  - HP CPU1 SP goes below the lower bound of the HP CPU1 SP monitored region

The configuration process for region monitoring and SP monitoring is as follows:

1. Configure the monitored region and SP bounds.
   - Configure HP CPU0 Data bus region 0 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG.
   - Configure HP CPU0 Data bus region 1 with ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG.
   - Configure HP CPU0 Peripheral bus region 0 with ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG.
   - Configure HP CPU0 Peripheral bus region 1 with ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG and ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG.

- Configure HP CPU0 SP bounds with ASSIST_DEBUG_CORE_0_SP_MIN_REG and ASSIST_DEBUG_CORE_0_SP_MAX_REG.
- Configure HP CPU1 Data bus region 0 with ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG.
- Configure HP CPU1 Data bus region 1 with ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG.
- Configure HP CPU1 Peripheral bus region 0 with ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG.
- Configure HP CPU1 Peripheral bus region 1 with ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG and ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG.
- Configure HP CPU1 SP bounds with ASSIST_DEBUG_CORE_1_SP_MIN_REG and ASSIST_DEBUG_CORE_1_SP_MAX_REG.

2. Configure interrupts.
   - Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG or ASSIST_DEBUG_CORE_1_INTR_ENA_REG to enable the interrupt of a monitoring mode.
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG or ASSIST_DEBUG_CORE_1_INTR_RAW_REG to get the raw interrupt status of a monitoring mode.
   - Configure ASSIST_DEBUG_CORE_0_INTR_CLR_REG or ASSIST_DEBUG_CORE_1_INTR_CLR_REG to clear the interrupt of a monitoring mode.
3. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG or ASSIST_DEBUG_CORE_1_MONTR_ENA_REG to enable the monitoring mode(s). Various monitoring modes can be enabled at the same time.

Assuming that Debug Assistant module needs to monitor whether HP CPU0 Data bus has written to [A ~ B] address space, the user can enable monitoring in either HP CPU0 Data bus region 0 or region 1. The following configuration process is based on region 0:

1. Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable HP CPU0 to update the PC signals to the Debug Assistant module.
2. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG to Address A.
3. Configure ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG to Address B.
4. Configure ASSIST_DEBUG_CORE_0_INTR_ENA_REG bit[1] to enable the interrupt for write operations by HP CPU0 Data bus in region 0.
5. Configure ASSIST_DEBUG_CORE_0_MONTR_ENA_REG bit[1] to enable monitoring write operations by HP CPU0 Data bus in region 0.
6. Configure interrupt matrix to map ASSIST_DEBUG_INTR into HP CPU0/1 interrupt (please refer to Chapter 10 *Interrupt Matrix*).
7. After the interrupt is triggered:
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG and ASSIST_DEBUG_CORE_1_INTR_RAW_REG to learn which operation triggered the interrupt.

- If the interrupt is triggered by HP CPU0 region monitoring, read ASSIST_DEBUG_CORE_0_AREA_PC_REG for the HP CPU0 PC value, and ASSIST_DEBUG_CORE_0_AREA_SP_REG for the HP CPU0 SP.
- If the interrupt is triggered by SP monitoring, read ASSIST_DEBUG_CORE_0_SP_PC_REG for the HP CPU0 PC value.
- Write 1 to the corresponding bits of ASSIST_DEBUG_CORE_0_INTR_RAW_REG and ASSIST_DEBUG_CORE_1_INTR_CLR_REG to clear the interrupts.

17.5.2 PC Logging Configuration

Configure ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN to 1 to enable HP CPU0 to update the PC signals to the Debug Assistant module. If ASSIST_DEBUG_CORE_0_RCD_RECORDEN is also configured to 1, ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG will record the HP CPU0's PC signal and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will record the HP CPU0 SP value. Otherwise, the two registers keep the original values.

When the HP CPU0 resets, ASSIST_DEBUG_CORE_0_RCD_EN_REG will reset, while ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG and ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG will not. Therefore, the two registers will keep the PC value and SP value at the HP CPU0 reset.

This is also the case for HP CPU1.

17.5.3 CPU/DMA Bus Access Logging Configuration

17.5.3.1 HP CPU0/1 Bus Access Logging Configuration

The configuration process for HP CPU0/1 bus access logging (TCM Monitor) is described below.

1. Configure monitored address space.
   - Configure TCM_MEM_MONITOR_LOG_MIN_REG and TCM_MEM_MONITOR_LOG_MAX_REG to specify monitored address space. The monitored address space should range from 0x3010_0000 to 0x3010_1FFF.
2. Configure the monitoring mode with TCM_MEM_MONITOR_LOG_MODE:
   - Write monitoring (whether the bus has write operations)
   - Word monitoring (whether the bus writes a specific word)
   - Halfword monitoring (whether the bus writes a specific halfword)
   - Byte monitoring (whether the bus writes a specific byte)
3. Configure the specific values to be monitored.
   - In word monitoring mode, TCM_MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, TCM_MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, TCM_MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.

- TCM_MEM_MONITOR_LOG_DATA_MASK_REG is used to mask the byte specified in TCM_MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if TCM_MEM_MONITOR_LOG_CHECK_DATA_REG is configured to 0x01020304 and TCM_MEM_MONITOR_LOG_DATA_MASK_REG is configured to 0x1, then any writes of the data matching the 0x010203XX pattern by the bus will be recorded.

4. Configure the storage space for recorded data.
   - TCM_MEM_MONITOR_LOG_MEM_START_REG and TCM_MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x4080_0000 ~ 0x4087_FFFF.
   - Set TCM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to TCM_MEM_MONITOR_LOG_MEM_START_REG.
   - Configure the permission for the TCM Monitor function of Debug Assistant module to access the HP L2MEM. Only when the access permission is enabled can the Debug Assistant module access the HP L2MEM. For more information, please refer to TEE_DMA_TCM_MON_PMS_W_REG and other relevant registers in Chapter 16 *Permission Control (PMS)*.
5. Configure the writing mode for the recorded data: loop mode or non-loop mode.
   - In loop mode, writing to the specified address space is performed in loops. When writing reaches the end address, it will return to the starting address and continue, overwriting the previously recorded data. Set TCM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the previous data written by the 1th to 5th operations.
   - In non-loop mode, when writing reaches the end address, it will stop at the end address and dump the remaining data, not overwriting the previously recorded data. Clear TCM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to use non-loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th to 10th write operations will stop at address 4 and will not be performed any more. Therefore, the address 0 ~ 4 stores the values written by the 1 ~ 5 operations and the values of the 6 ~ 10 operations are dumped.
6. Configure bus enable registers.
   - Enable HP CPU0 or HP CPU1 bus access logging with TCM_MEM_MONITOR_LOG_CORE_ENA. They can be enabled at the same time.

The Debug Assistant module first writes the recorded data of HP CPU0 or HP CPU1 to the internal buffer A, and then fetches the data from the buffer and writes it to the configured memory space. When the monitored behaviors are triggered continuously, the generated recording packets may fully occupy the buffer, making it unable to take any incoming packets. At this time, the module dumps these incoming packets and buffers a HP CPU LOST packet instead before the buffer reaches its capacity. However, it is only possible to know the bus type of the dumped packets when the HP CPU LOST packet was generated, but not the number of the dumped packets before the generation.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data is in three packet formats, namely HP CPU0 packet (corresponding to HP CPU0 Data bus), HP

CPU1 packet (corresponding to HP CPU1 Data bus), and HP CPU LOST packet. The packet formats are shown in Table 17.5-1 and 17.5-2.

Table 17.5-1. HP CPU0/1 Packet Format

Bit[63:33]	Bit[32]	Bit[31:3]	Bit[2:1]	Bit[0]
pc_offset	anchored (1)	addr_offset	format	anchored (0)

Table 17.5-2. HP CPU LOST Packet Format

Bit[31:7]	Bit[6:3]	Bit[2:1]	Bit[0]
reserved	lost_source	format	anchored (0)

It can be seen from the data packet formats that the size of the HP CPU0/1 packet is 64 bits and that of the HP CPU LOST packet is 32 bits. These packets contain the following fields:

- **format** – the packet type. 0: HP CPU0 packet; 1: HP CPU1 packet; 2: HP CPU LOST packet; 3: Reserved.
- **pc_offset** - the offset of the HP CPU0/1 PC register at the time of access. Actual PC = pc_offset + 0x0000_0000.
- **addr_offset** - the address offset of a write operation. Actual address = addr_offset + TCM_MEM_MONITOR_LOG_MIN_REG[31:4], 4'h0.
- **lost_source** - the dumped packets when the HP CPU LOST packet was generated.
  - Bit[4:3]: 0 indicates HP CPU0 packets were not dumped. 3 indicates HP CPU0 packets were dumped. Other values are reserved.
  - Bit[6:5]: 0 indicates HP CPU1 packets were not dumped. 3 indicates HP CPU1 packets were dumped. Other values are reserved.
- **anchored** - the location of the 32 bits in the data packet. 0: Lower 32 bits. 1: Higher 32 bits.

The internal buffer of the module is 32 bits wide. When the HP CPU0 and HP CPU1 bus access loggings are both enabled and the record data is generated at the same time, the HP CPU0 data packets are first buffered, then the HP CPU1 packets. The Debug Assistant will automatically fetch the buffered data and store it in 32-bit data width into the specified memory space.

In loop mode, data looping several times in the storage memory may cause residual data, which can interfere with packet parsing. For example, the lower 32 bits of a HP CPU0/1 packet are overwritten, thus making its higher 32 bits residual data. Therefore, users need to filter out the possible residual data in order to determine the starting position of the first valid packet with TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG. Once the starting position of the packet is identified, check the anchored bit value of the packet. If it is 0, the data will be retained. If it is 1, it will be dumped.

The process of packet parsing is described below:

- Determine whether there is a data overflow with TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG.
  - If no, the address space to read is TCM_MEM_MONITOR_LOG_MEM_START_REG ~ TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.

  - If yes and the loop mode is enabled, the address space is TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ TCM_MEM_MONITOR_LOG_MEM_END_REG and TCM_MEM_MONITOR_LOG_MEM_START_REG ~ TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and loop mode is not enabled, the address space is TCM_MEM_MONITOR_LOG_MEM_START_REG ~ TCM_MEM_MONITOR_LOG_MEM_END_REG.
- Read and parse data from the starting address. Read 32 bits each time.

After packet parsing is completed, clear the TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting TCM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG.

17.5.3.2 DMA Bus Access Logging Configuration

The Debug Assistant module supports a total of four DMA bus channel groups, DMA_0, DMA_1, DMA_2, and DMA_3 channel groups. DMA_0 and DMA_1 channel groups are reserved and cannot be used. The DMA bus for the DMA_2 channel group is the bus on which the AXI matrix accesses the HP L2MEM. The DMA bus for the DMA_3 channel group is the bus on which the AHB matrix accesses the HP L2MEM. For more information, see Chapter 5 *System and Memory* > Section 5.3.2. The configuration process for DMA bus access logging (L2MEM Monitor) is described below.

1. Configure monitored address space.
   - Configure L2_MEM_MONITOR_LOG_MIN_REG and L2_MEM_MONITOR_LOG_MAX_REG to specify monitored address space. The monitored address space should range from 0x4FF0_0000 to 0x4FFB_FFFF or from 0x8FF0_0000 to 0x8FFB_FFFF.
2. Configure the monitoring mode with L2_MEM_MONITOR_LOG_MODE:
   - Write monitoring (whether the bus has write operations)
   - Word monitoring (whether the bus writes a specific word)
   - Halfword monitoring (whether the bus writes a specific halfword)
   - Byte monitoring (whether the bus writes a specific byte)
3. Configure the specific values to be monitored.
   - In word monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG specifies the monitored word.
   - In halfword monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] specifies the monitored halfword.
   - In byte monitoring mode, L2_MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] specifies the monitored byte.
   - L2_MEM_MONITOR_LOG_DATA_MASK_REG is used to mask the byte specified in L2_MEM_MONITOR_LOG_CHECK_DATA_REG. A masked byte can be any value. For example, in word monitoring, if L2_MEM_MONITOR_LOG_CHECK_DATA_REG is configured to 0x01020304 and L2_MEM_MONITOR_LOG_DATA_MASK_REG is configured to 0x1, then any writes of the data matching the 0x010203XX pattern by the bus will be recorded.
4. Configure the storage space for recorded data.

- L2_MEM_MONITOR_LOG_MEM_START_REG and L2_MEM_MONITOR_LOG_MEM_END_REG specify the storage space for recorded data. The storage space must be in the range of 0x4080_0000 ~ 0x4087_FFFF.
- Set L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG to update the value in L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG to L2_MEM_MONITOR_LOG_MEM_START_REG.
- Configure the permission for the L2MEM Monitor function of Debug Assistant module to access the HP L2MEM. Only when the access permission is enabled can the Debug Assistant module access the HP L2MEM. For more information, please refer to TEE_DMA_L2MEM_MON_PMS_W_REG and other relevant registers in Chapter 16 *Permission Control (PMS)*.

5. Configure the writing mode for the recorded data: loop mode or non-loop mode.
   - In loop mode, writing to the specified address space is performed in loops. When writing reaches the end address, it will return to the starting address and continue, overwriting the previously recorded data. Set L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to enable loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th operation will start writing from address 0. The 6th to 10th operations will overwrite the previous data written by the 1th to 5th operations.
   - In non-loop mode, when writing reaches the end address, it will stop at the end address and dump the remaining data, not overwriting the previously recorded data. Clear L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE to use non-loop mode.
     For example, there are 10 write operations (1 ~ 10) to address space 0 ~ 4 during bus access. After the 5th operation writes to address 4, the 6th to 10th write operations will stop at address 4 and will not be performed any more. Therefore, the address 0 ~ 4 stores the values written by the 1 ~ 5 operations and the values of the 6 ~ 10 operations are dumped.
   - See the example in 17.5.3.1 > step 5.
6. Configure bus enable registers.
   - Enable DMA_2 and DMA_3 channel groups bus access logging with L2_MEM_MONITOR_LOG_DMA_2_ENA and L2_MEM_MONITOR_LOG_DMA_3_ENA respectively. They can be enabled at the same time.

The Debug Assistant module first writes the DMA recorded data to the internal buffer B, and then fetches the data from the buffer and writes it to the configured memory space. When the monitored behaviors are triggered continuously, the generated recording packets may fully occupy the buffer, making it unable to take any incoming packets. At this time, the module dumps these incoming packets and buffers a DMA LOST packet instead before the buffer reaches its capacity. However, it is only possible to know the bus type of the dumped packets when the DMA LOST packet was generated, but not the number of the dumped packets before the generation.

When bus access logging is finished, the recorded data can be read from memory for decoding. The recorded data is in three packet formats, namely DMA_2 packet (corresponding to DMA_2 Data bus), DMA_3 packet (corresponding to DMA_3 Data bus), and DMA LOST packet. The packet formats are shown in Table 17.5-3, 17.5-4, and 17.5-5.

Table 17.5-3. DMA_2 Packet Format

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset0	reserved	format	anchored (0)

Table 17.5-4. DMA_3 Packet Format

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset1	reserved	format	anchored (0)

Table 17.5-5. DMA LOST Packet Format

Bit[31:10]	Bit[9:8]	Bit[7:4]	Bit[3:1]	Bit[0]
reserved	lost_source	reserved	format	anchored (0)

It can be seen from the data packet formats that the size of the DMA_2, DMA_3, and DMA LOST packet is 32 bits. These packets contain the following fields:

- **format** – the packet type. 4: DMA_2 packet; 5: DMA_3 packet; 6: DMA LOST packet; other values: Reserved.
- **addr_offset0** - the address offset of a write operation. Actual address = addr_offset0 + L2_MEM_MONITOR_LOG_MIN_REG[31:5],5'h0.
- **addr_offset1** - the address offset of a write operation. Actual address = addr_offset1 + L2_MEM_MONITOR_LOG_MIN_REG[31:2],2'h0.
- **lost_source** - the dumped packets when the DMA LOST packet was generated.
  - Bit[8]: 0 indicates DMA_2 packets were not dumped. 1 indicates DMA_2 packets were dumped.
  - Bit[9]: 0 indicates DMA_3 packets were not dumped. 1 indicates DMA_3 packets were dumped.
- **anchored** - the location of the 32 bits in the data packet. 0: Lower 32 bits. 1: Higher 32 bits.

The internal buffer of the module is 32 bits wide. When the bus access loggings of DMA_2 and DMA_3 channel groups are both enabled and the record data is generated at the same time, the DMA_2 data packets are first buffered, then the DMA_3 packets. The Debug Assistant will automatically fetch the buffered data and store it in 32-bit data width into the specified memory space.

The process of packet parsing is described below:

- Determine whether there is a data overflow with L2_MEM_MONITOR_LOG_MEM_FULL_FLAG.
  - If no, the address space to read is L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and the loop mode is enabled, the address space is L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG and L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4.
  - If yes and loop mode is not enabled, the address space is L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG.

- Read and parse data from the starting address. Read 32 bits each time.

After packet parsing is completed, clear the L2_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit by setting L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG.

17.6 Register Summary

The addresses of **HP CPU bus logging configuration registers** (see 17.6.1) in this section are relative to the **TCM Monitor** base address. The addresses of **DMA bus logging configuration registers** (see 17.6.2) in this section are relative to the **L2MEM Monitor** base address. The addresses of other registers (see 17.6.3) are relative to the **Bus Monitor** base address. All base addresses are provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

17.6.1 HP CPU Bus Logging Configuration Register Summary

Name	Description	Address	Access
HP CPU bus logging configuration registers			
TCM_MEM_MONITOR_LOG_SETTING_REG	Configures bus access logging	0x0000	R/W
TCM_MEM_MONITOR_LOG_CHECK_DATA_REG	Configures monitored data in bus access logging	0x0008	R/W
TCM_MEM_MONITOR_LOG_DATA_MASK_REG	Configures masked data in bus access logging	0x000C	R/W
TCM_MEM_MONITOR_LOG_MIN_REG	Configures monitored address space in bus access logging	0x0010	R/W
TCM_MEM_MONITOR_LOG_MAX_REG	Configures monitored address space in bus access logging	0x0014	R/W
TCM_MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the memory for recorded data	0x0018	R/W
TCM_MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the memory for recorded data	0x001C	R/W
TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x0020	RO
TCM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write to the starting address for the recorded data	0x0024	WT
TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Represents the logging overflow status	0x0028	varies
Clock control register			
TCM_MEM_MONITOR_CLOCK_GATE_REG	Clock control register	0x002C	R/W
Version control register			
TCM_MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

17.6.2 DMA Bus Logging Configuration Register Summary

Name	Description	Address	Access
DMA bus logging configuration registers			

Name	Description	Address	Access
L2_MEM_MONITOR_LOG_SETTING_REG	Configures bus access logging	0x0000	R/W
L2_MEM_MONITOR_LOG_SETTING1_REG	Configures bus access logging	0x0004	R/W
L2_MEM_MONITOR_LOG_CHECK_DATA_REG	Configures monitored data in bus access logging	0x0008	R/W
L2_MEM_MONITOR_LOG_DATA_MASK_REG	Configures masked data in bus access logging	0x000C	R/W
L2_MEM_MONITOR_LOG_MIN_REG	Configures monitored address space in bus access logging	0x0010	R/W
L2_MEM_MONITOR_LOG_MAX_REG	Configures monitored address space in bus access logging	0x0014	R/W
L2_MEM_MONITOR_LOG_MEM_START_REG	Configures the starting address of the memory for recorded data	0x0018	R/W
L2_MEM_MONITOR_LOG_MEM_END_REG	Configures the end address of the memory for recorded data	0x001C	R/W
L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	Represents the address for the next write	0x0020	RO
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	Updates the address for the next write to the starting address for the recorded data	0x0024	WT
L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	Represents the logging overflow status	0x0028	varies
Clock control register			
L2_MEM_MONITOR_CLOCK_GATE_REG	Clock control register	0x002C	R/W
Version control register			
L2_MEM_MONITOR_DATE_REG	Version control register	0x03FC	R/W

17.6.3 Summary of Other Registers

Name	Description	Address	Access
Monitor configuration registers			
ASSIST_DEBUG_CORE_0_MONTR_ENA_REG	Configures whether to enable HP CPU0 monitoring	0x0000	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU0 Data bus	0x0010	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU0 Data bus	0x0014	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU0 Data bus	0x0018	R/W

Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU0 Data bus	0x001C	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU0 Peripheral bus	0x0020	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU0 Peripheral bus	0x0024	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU0 Peripheral bus	0x0028	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU0 Peripheral bus	0x002C	R/W
ASSIST_DEBUG_CORE_0_AREA_PC_REG	Represents the PC status under HP CPU0 region monitoring	0x0030	RO
ASSIST_DEBUG_CORE_0_AREA_SP_REG	Represents the SP status under HP CPU0 region monitoring	0x0034	RO
ASSIST_DEBUG_CORE_0_SP_MIN_REG	Configures the lower bound address of the HP CPU0 SP monitored region	0x0038	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	Configures the upper bound address of the HP CPU0 SP monitored region	0x003C	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	Represents the PC status under HP CPU0 SP monitoring	0x0040	RO
ASSIST_DEBUG_CORE_1_MONTR_ENA_REG	Configures whether to enable HP CPU1 monitoring	0x0080	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU1 Data bus	0x0090	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU1 Data bus	0x0094	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU1 Data bus	0x0098	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU1 Data bus	0x009C	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG	Configures the lower bound address of region 0 monitored on HP CPU1 Peripheral bus	0x00A0	R/W

Name	Description	Address	Access
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG	Configures the upper bound address of region 0 monitored on HP CPU1 Peripheral bus	0x00A4	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG	Configures the lower bound address of region 1 monitored on HP CPU1 Peripheral bus	0x00A8	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG	Configures the upper bound address of region 1 monitored on HP CPU1 Peripheral bus	0x00AC	R/W
ASSIST_DEBUG_CORE_1_AREA_PC_REG	Represents the PC status under HP CPU1 region monitoring	0x00B0	RO
ASSIST_DEBUG_CORE_1_AREA_SP_REG	Represents the SP status under HP CPU1 region monitoring	0x00B4	RO
ASSIST_DEBUG_CORE_1_SP_MIN_REG	Configures the lower bound address of the HP CPU1 SP monitored region	0x00B8	R/W
ASSIST_DEBUG_CORE_1_SP_MAX_REG	Configures the upper bound address of the HP CPU1 SP monitored region	0x00BC	R/W
ASSIST_DEBUG_CORE_1_SP_PC_REG	Represents the PC status under HP CPU1 SP monitoring	0x00C0	RO
Interrupt configuration registers			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	HP CPU0 raw interrupt status register	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_ENA_REG	HP CPU0 interrupt enable register	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	HP CPU0 interrupt clear register	0x000C	WT
ASSIST_DEBUG_CORE_1_INTR_RAW_REG	HP CPU1 raw interrupt status register	0x0084	RO
ASSIST_DEBUG_CORE_1_INTR_ENA_REG	HP CPU1 interrupt enable register	0x0088	R/W
ASSIST_DEBUG_CORE_1_INTR_CLR_REG	HP CPU1 interrupt clear register	0x008C	WT
PC logging configuration register			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	HP CPU0 PC logging enable register	0x0044	R/W
ASSIST_DEBUG_CORE_1_RCD_EN_REG	HP CPU1 PC logging enable register	0x00C4	R/W
PC logging status registers			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	HP CPU0 PC logging register	0x0048	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	HP CPU0 SP logging register	0x004C	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG	HP CPU1 PC logging register	0x00C8	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG	HP CPU1 SP logging register	0x00CC	RO
CPU status registers			
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU0 enters exception	0x0070	RO

Name	Description	Address	Access
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	HP CPU0 debug mode status register	0x0074	RO
ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG	PC of the last command before HP CPU1 enters exception	0x00F0	RO
ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG	HP CPU1 debug mode status register	0x00F4	RO
Clock control register			
ASSIST_DEBUG_CLOCK_GATE_REG	Clock control register	0x0108	R/W
Version control register			
ASSIST_DEBUG_DATE_REG	Version control register	0x03FC	R/W

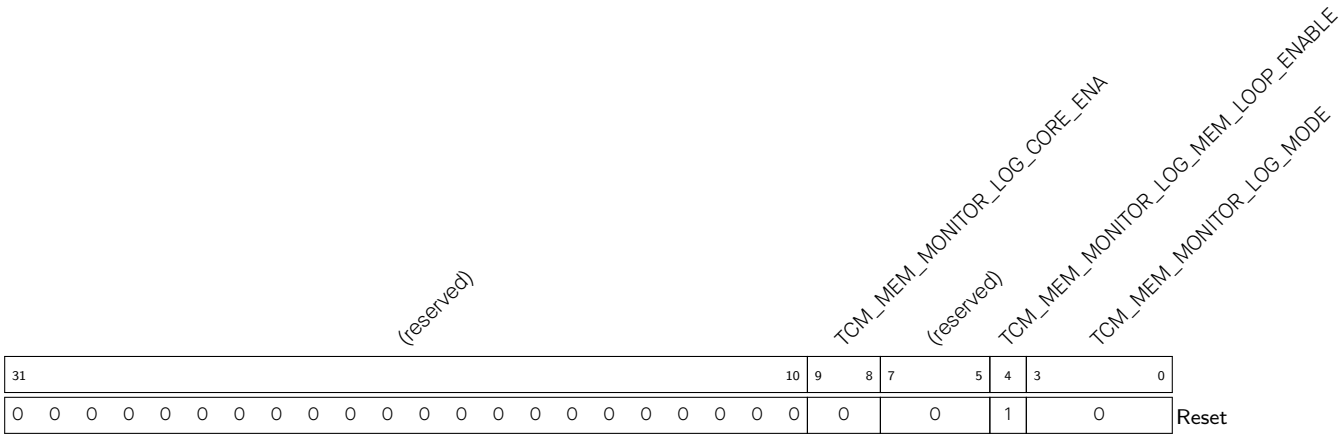
17.7 Registers

The addresses of **HP CPU bus logging configuration registers** (see 17.6.1) in this section are relative to the **TCM Monitor** base address. The addresses of **DMA bus logging configuration registers** (see 17.6.2) in this section are relative to the **L2MEM Monitor** base address. The addresses of other registers (see 17.7.3) are relative to the **Bus Monitor** base address. All base addresses are provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

17.7.1 HP CPU Bus Logging Configuration Registers

Register 17.1. TCM_MEM_MONITOR_LOG_SETTING_REG (0x0000)



TCM_MEM_MONITOR_LOG_MODE Configures monitoring modes.

bit[0]: Configures write monitoring.
0: Disable
1: Enable
bit[1]: Configures word monitoring.
0: Disable
1: Enable
bit[2]: Configures halfword monitoring.
0: Disable
1: Enable
bit[3]: Configures byte monitoring.
0: Disable
1: Enable
(R/W)

TCM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for recorded data.

1: Loop mode
0: Non-loop mode
(R/W)

Continued on the next page...

Register 17.1. TCM_MEM_MONITOR_LOG_SETTING_REG (0x0000)

Continued from the previous page...

TCM_MEM_MONITOR_LOG_CORE_ENA Configures whether to enable HP CPU0 or HP CPU1 bus access logging.

bit[0]: Configures whether to enable HP CPU0 bus access logging.

0: Disable

1: Enable

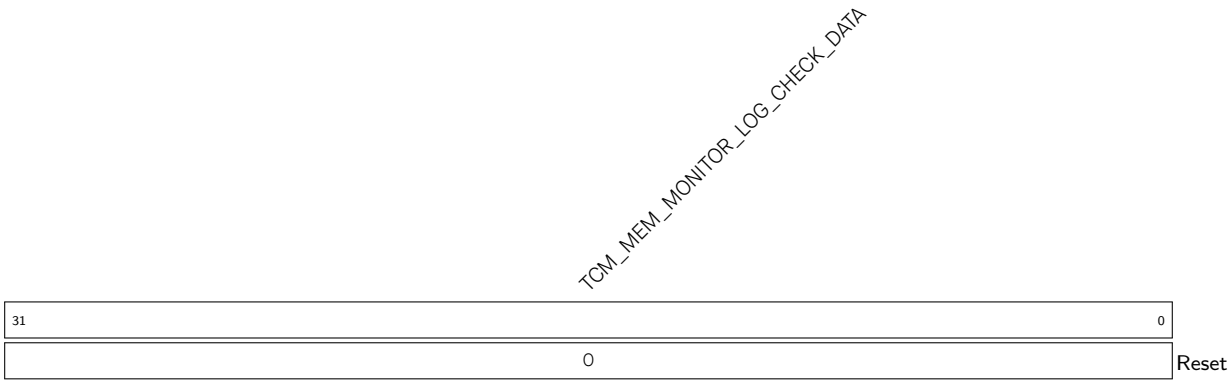
bit[1]: Configures whether to enable HP CPU1 bus access logging.

0: Disable

1: Enable

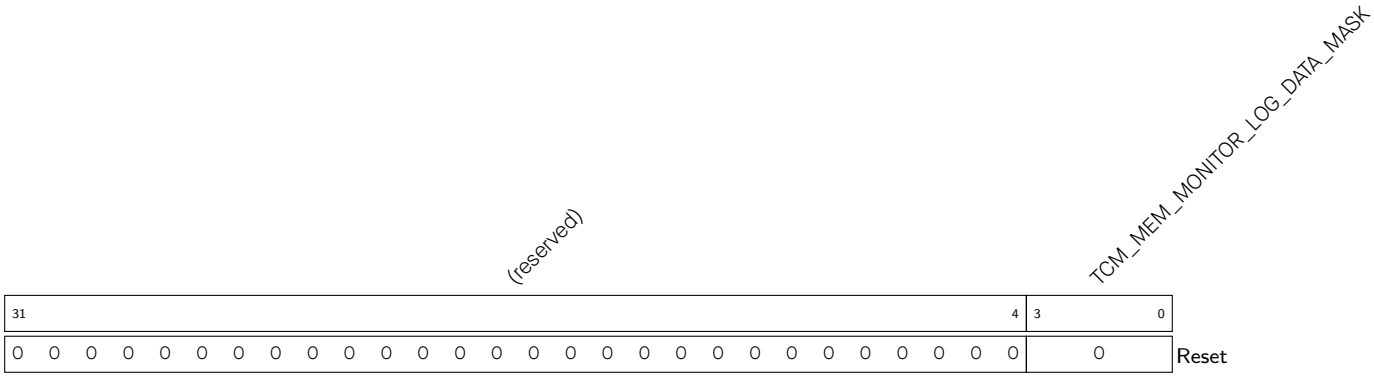
(R/W)

Register 17.2. TCM_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



TCM_MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 17.3. TCM_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



TCM_MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in TCM_MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

bit[0]: Configures whether to mask the least significant byte of TCM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[1]: Configures whether to mask the second least significant byte of TCM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[2]: Configures whether to mask the second most significant byte of TCM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

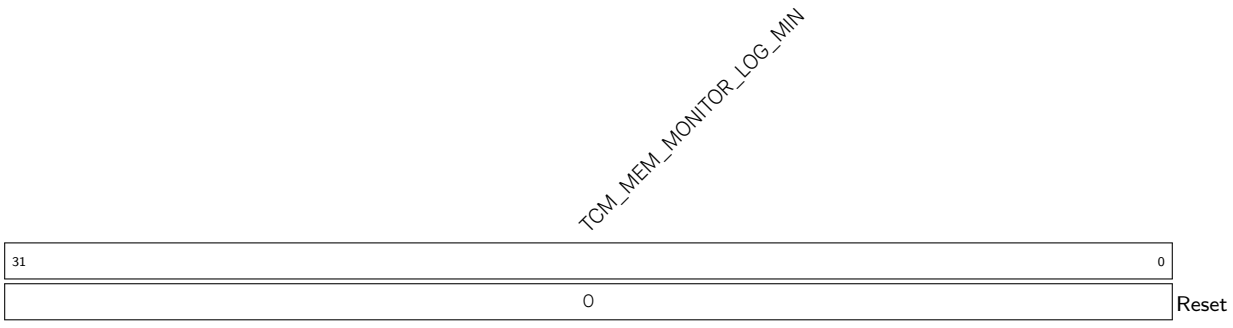
bit[3]: Configures whether to mask the most significant byte of TCM_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

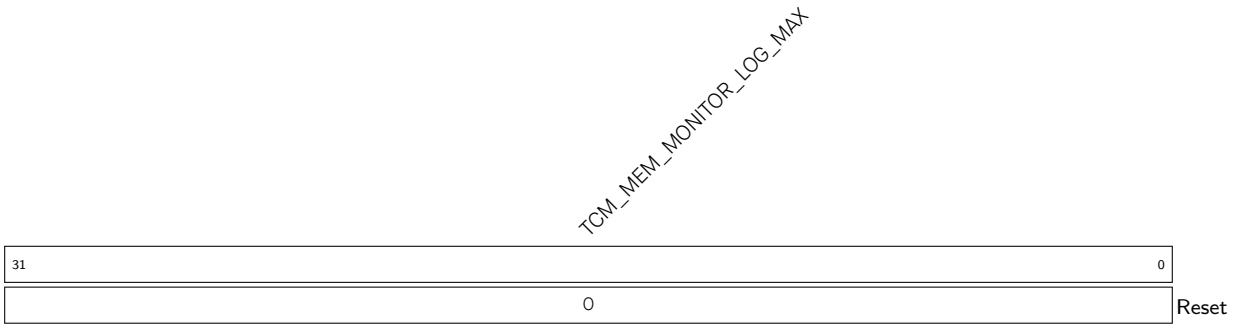
(R/W)

Register 17.4. TCM_MEM_MONITOR_LOG_MIN_REG (0x0010)



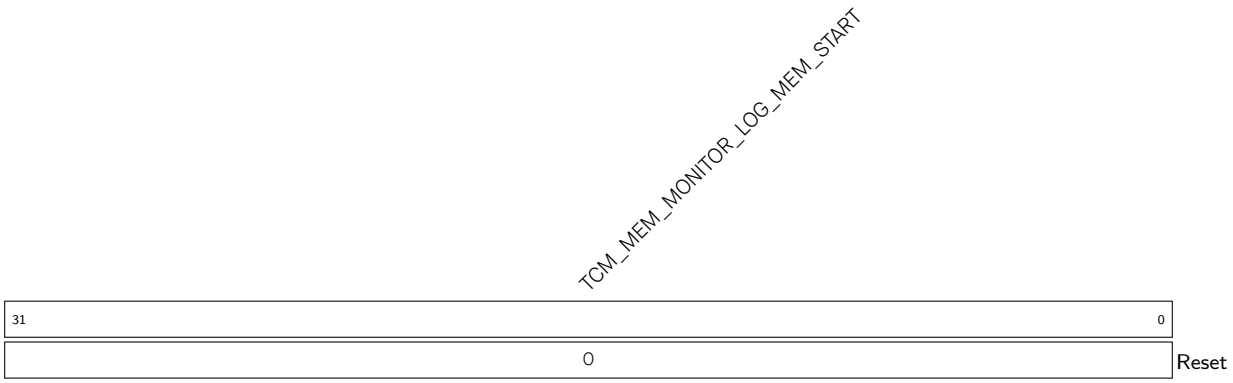
TCM_MEM_MONITOR_LOG_MIN Configures the lower bound address of the monitored address space. (R/W)

Register 17.5. TCM_MEM_MONITOR_LOG_MAX_REG (0x0014)



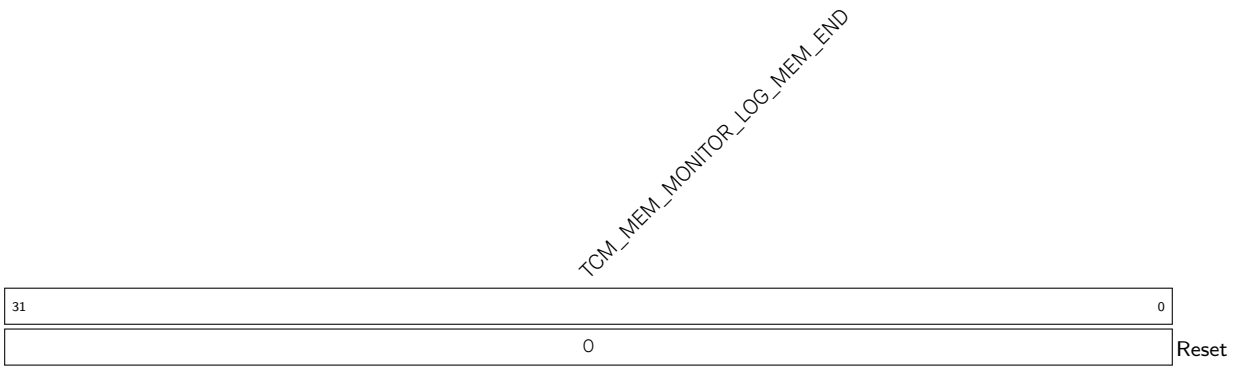
TCM_MEM_MONITOR_LOG_MAX Configures the upper bound address of the monitored address space. (R/W)

Register 17.6. TCM_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



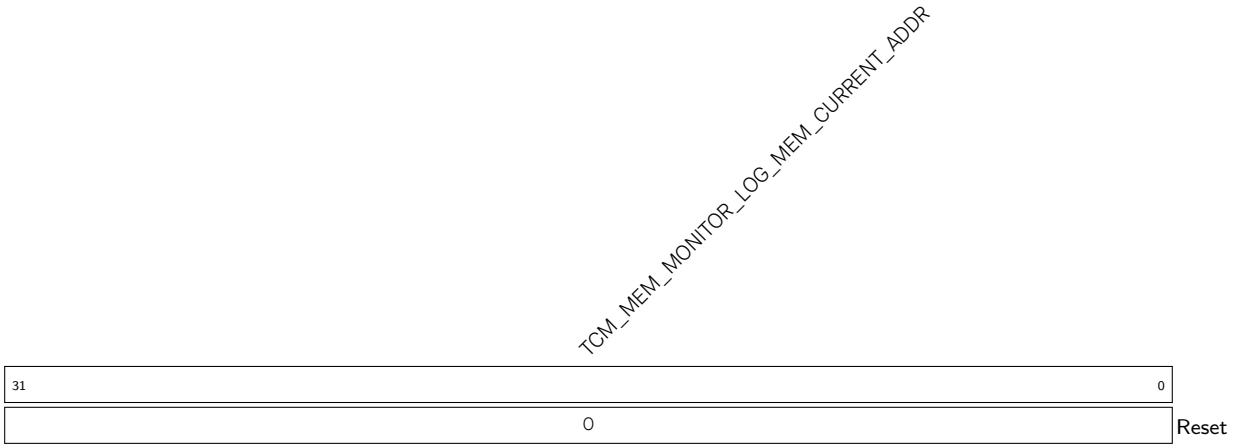
TCM_MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for recorded data. (R/W)

Register 17.7. TCM_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



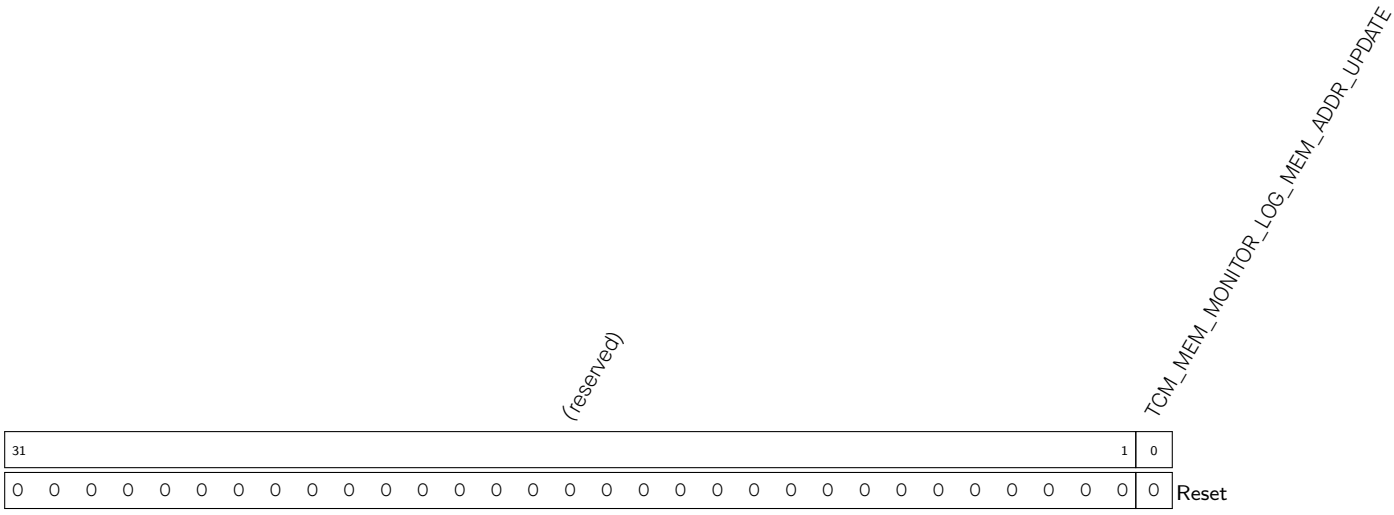
TCM_MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for recorded data. (R/W)

Register 17.8. TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 17.9. TCM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



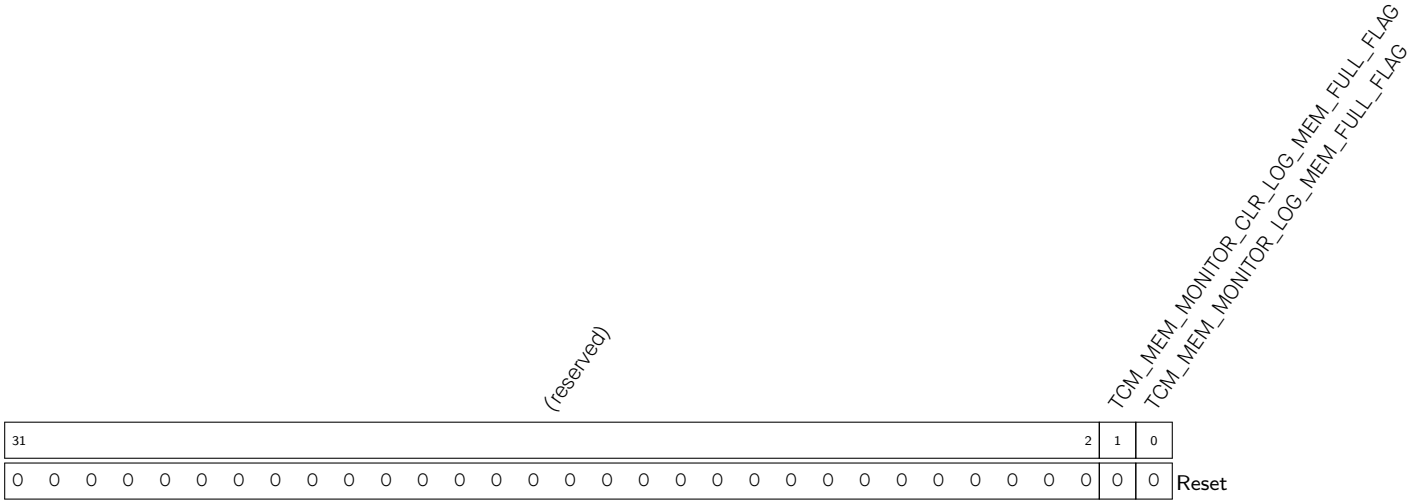
TCM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in TCM_MEM_MONITOR_LOG_MEM_START_REG to TCM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

1: Update

0: Not update

(WT)

Register 17.10. TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space.

0: Not overflow

1: Overflow

(RO)

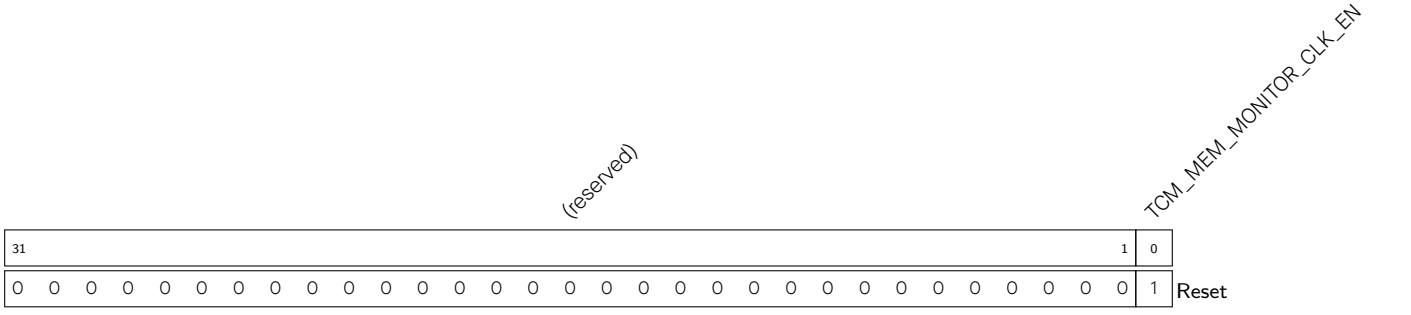
TCM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the TCM_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear (default)

1: Clear

(WT)

Register 17.11. TCMMEM_MONITOR_CLOCK_GATE_REG (0x002C)



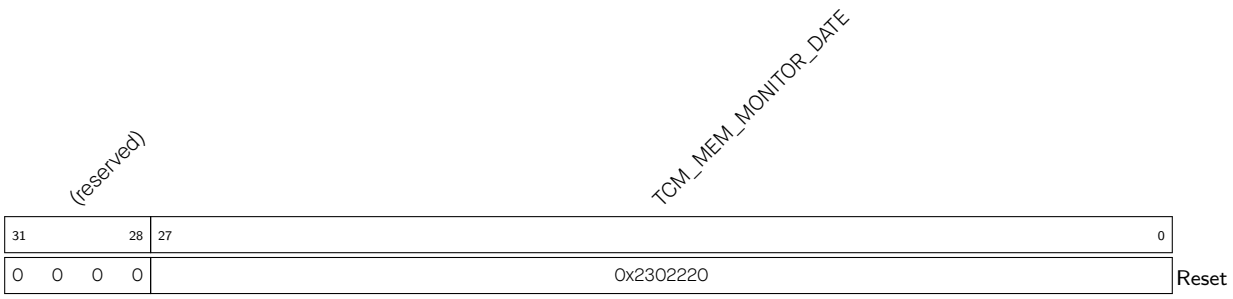
TCM_MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

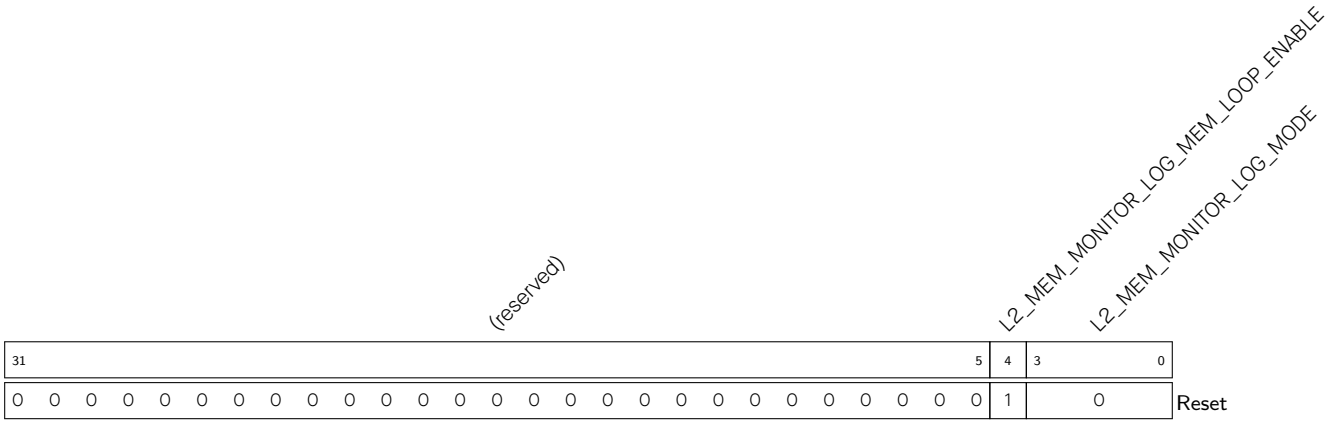
Register 17.12. TCM_MEM_MONITOR_DATE_REG (0x03FC)



TCM_MEM_MONITOR_DATE Version control register. (R/W)

17.7.2 DMA Bus Logging Configuration Registers

Register 17.13. L2_MEM_MONITOR_LOG_SETTING_REG (0x0000)



L2_MEM_MONITOR_LOG_MODE Configures monitoring modes.

bit[0]: Configures write monitoring.

0: Disable

1: Enable

bit[1]: Configures word monitoring.

0: Disable

1: Enable

bit[2]: Configures halfword monitoring.

0: Disable

1: Enable

bit[3]: Configures byte monitoring.

0: Disable

1: Enable

(R/W)

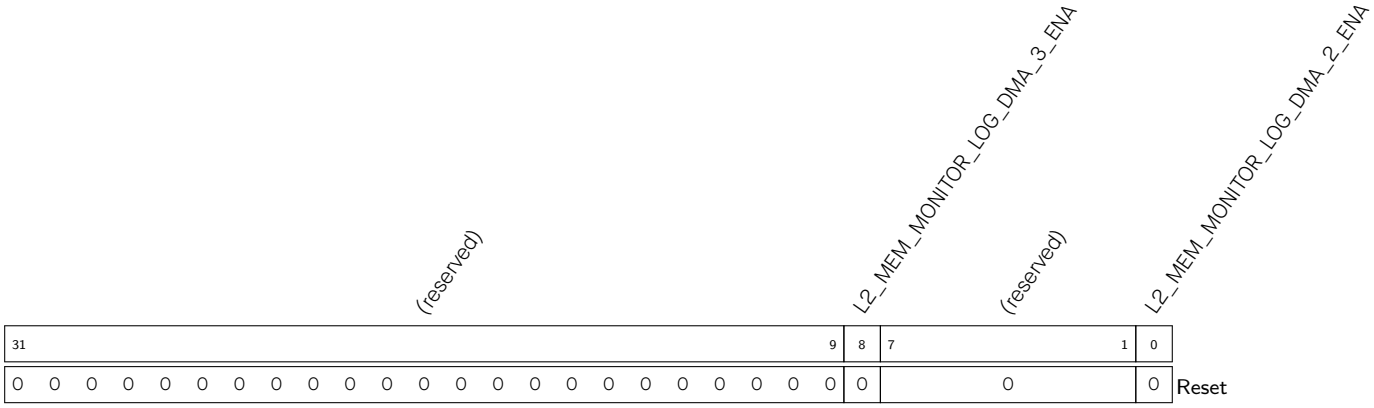
L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE Configures the writing mode for recorded data.

1: Loop mode

0: Non-loop mode

(R/W)

Register 17.14. L2_MEM_MONITOR_LOG_SETTING1_REG (0x0004)



L2_MEM_MONITOR_LOG_DMA_2_ENA Configures whether to enable DMA_2 bus access logging.

0: Disable

1: Enable

(R/W)

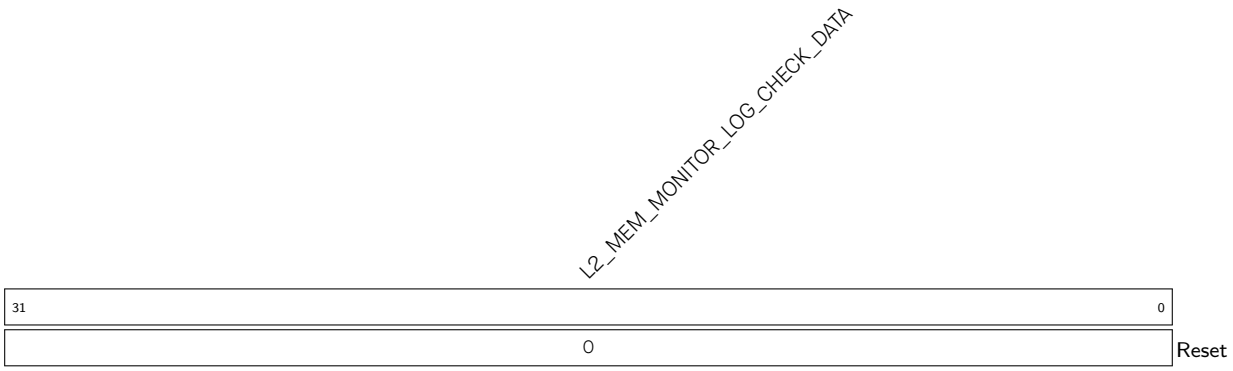
L2_MEM_MONITOR_LOG_DMA_3_ENA Configures whether to enable DMA_3 bus access logging.

0: Disable

1: Enable

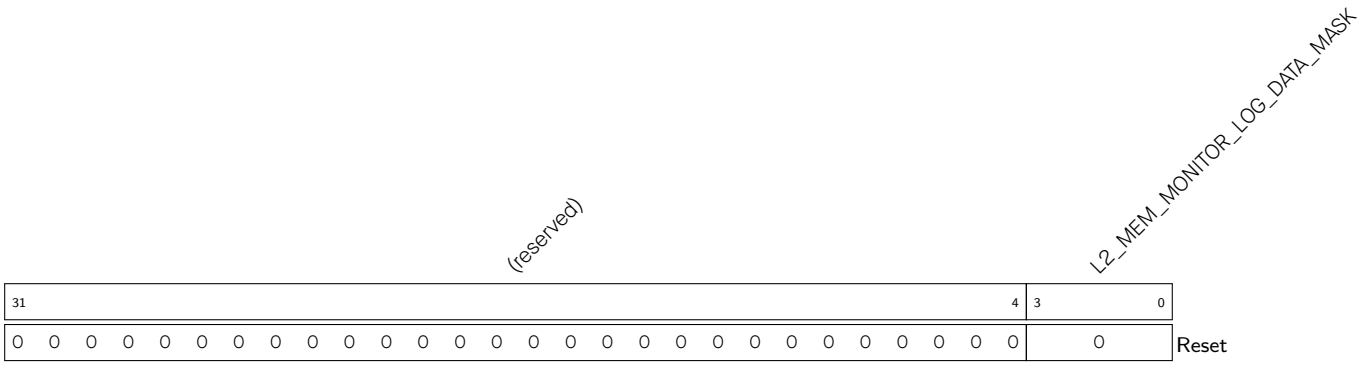
(R/W)

Register 17.15. L2_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



L2_MEM_MONITOR_LOG_CHECK_DATA Configures the data to be monitored during bus accessing. (R/W)

Register 17.16. L2_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



L2_MEM_MONITOR_LOG_DATA_MASK Configures which byte(s) in L2_MEM_MONITOR_LOG_CHECK_DATA_REG to mask.

bit[0]: Configures whether to mask the least significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[1]: Configures whether to mask the second least significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

bit[2]: Configures whether to mask the second most significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

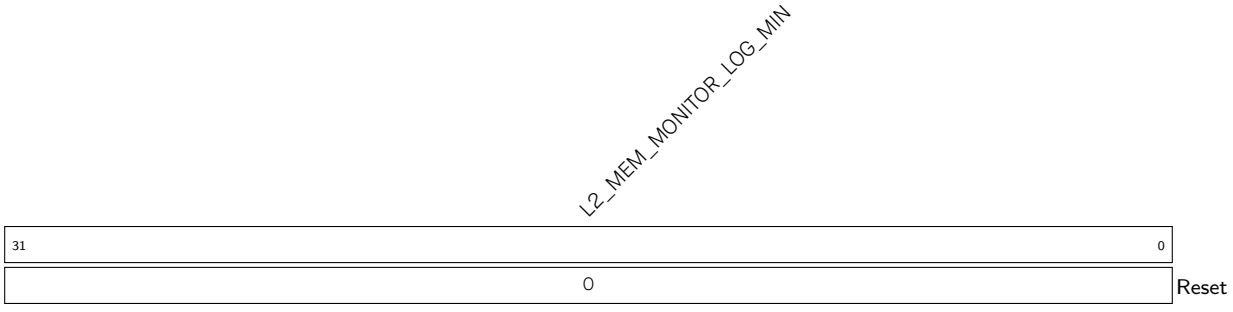
bit[3]: Configures whether to mask the most significant byte of L2_MEM_MONITOR_LOG_CHECK_DATA_REG.

0: Not mask

1: Mask

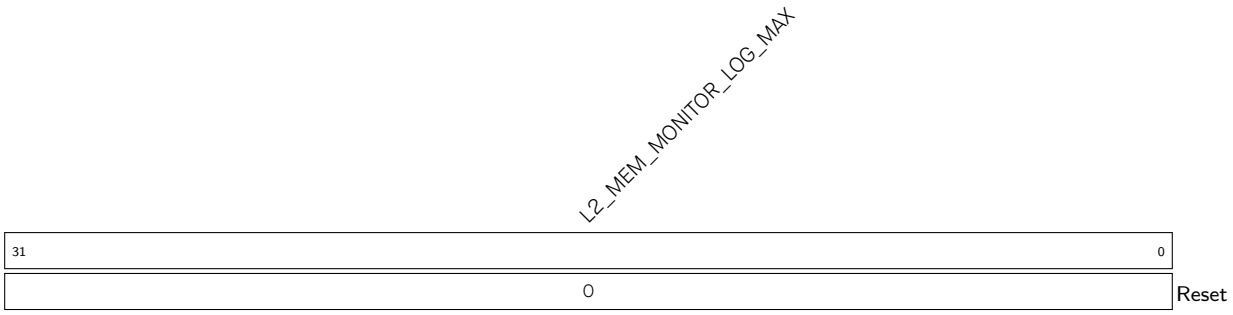
(R/W)

Register 17.17. L2_MEM_MONITOR_LOG_MIN_REG (0x0010)



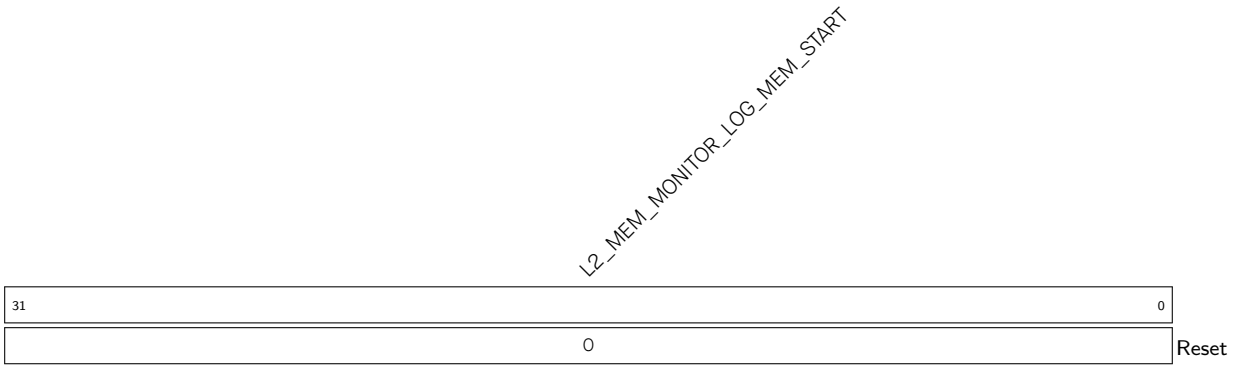
L2_MEM_MONITOR_LOG_MIN Configures the lower bound address of the monitored address space. (R/W)

Register 17.18. L2_MEM_MONITOR_LOG_MAX_REG (0x0014)



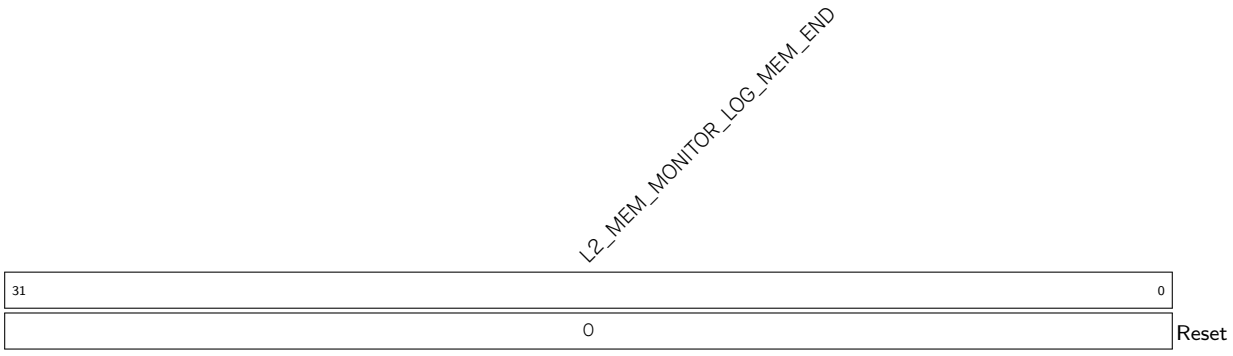
L2_MEM_MONITOR_LOG_MAX Configures the upper bound address of the monitored address space. (R/W)

Register 17.19. L2_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



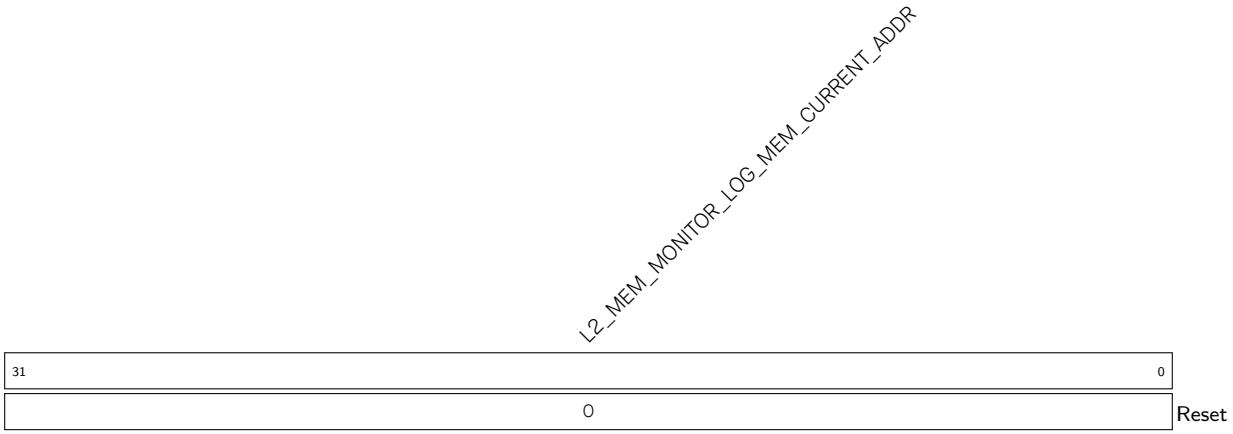
L2_MEM_MONITOR_LOG_MEM_START Configures the starting address of the storage space for recorded data. (R/W)

Register 17.20. L2_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



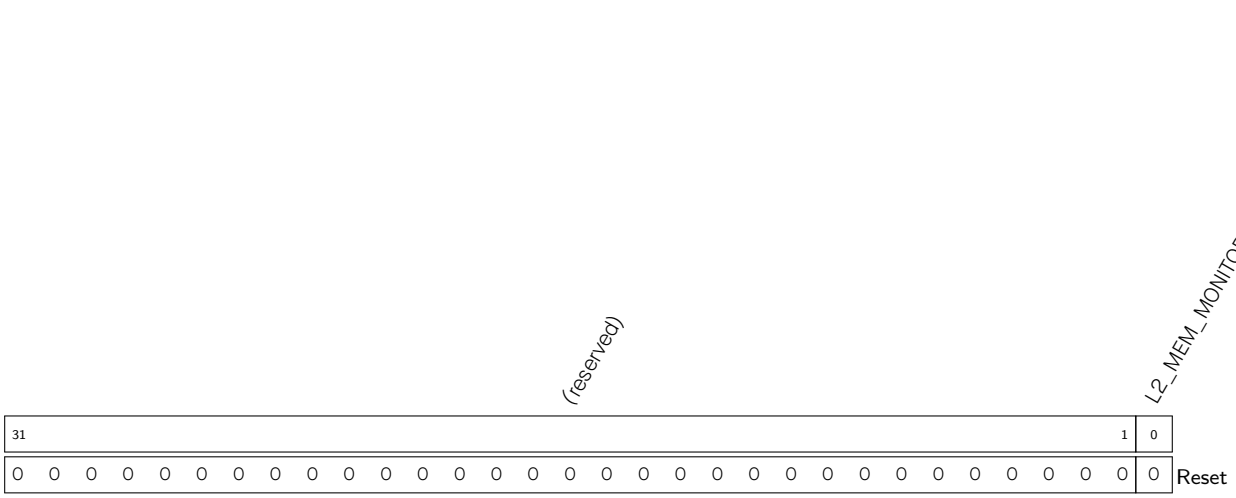
L2_MEM_MONITOR_LOG_MEM_END Configures the ending address of the storage space for recorded data. (R/W)

Register 17.21. L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR Represents the address of the next write. (RO)

Register 17.22. L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



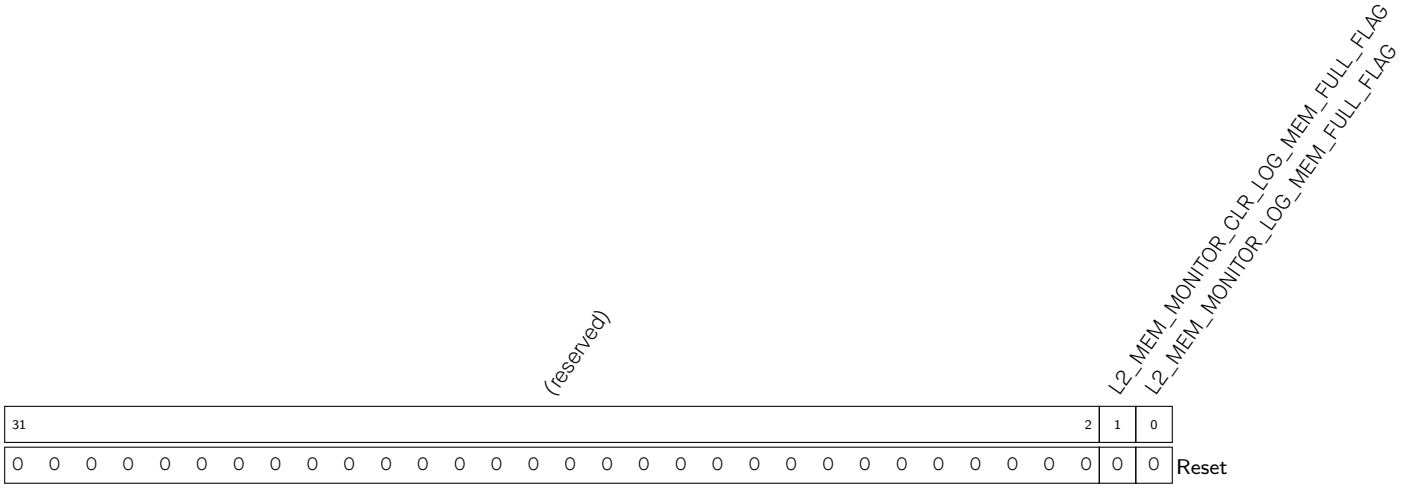
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE Configures whether to update the value in L2_MEM_MONITOR_LOG_MEM_START_REG to L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG.

1: Update

0: Not update (default)

(WT)

Register 17.23. L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



L2_MEM_MONITOR_LOG_MEM_FULL_FLAG Represents whether data overflows the storage space.

0: Not overflow

1: Overflow

(RO)

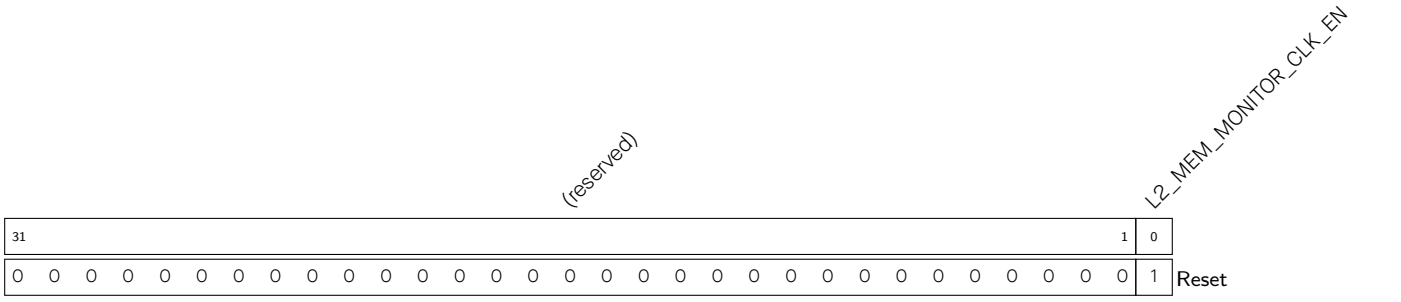
L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG Configures whether to clear the L2_MEM_MONITOR_LOG_MEM_FULL_FLAG flag bit.

0: Not clear (default)

1: Clear

(WT)

Register 17.24. L2_MEM_MONITOR_CLOCK_GATE_REG (0x002C)



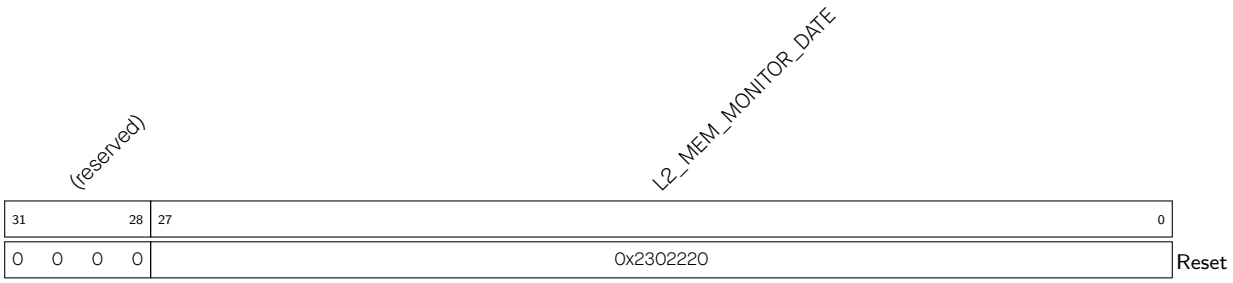
L2_MEM_MONITOR_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

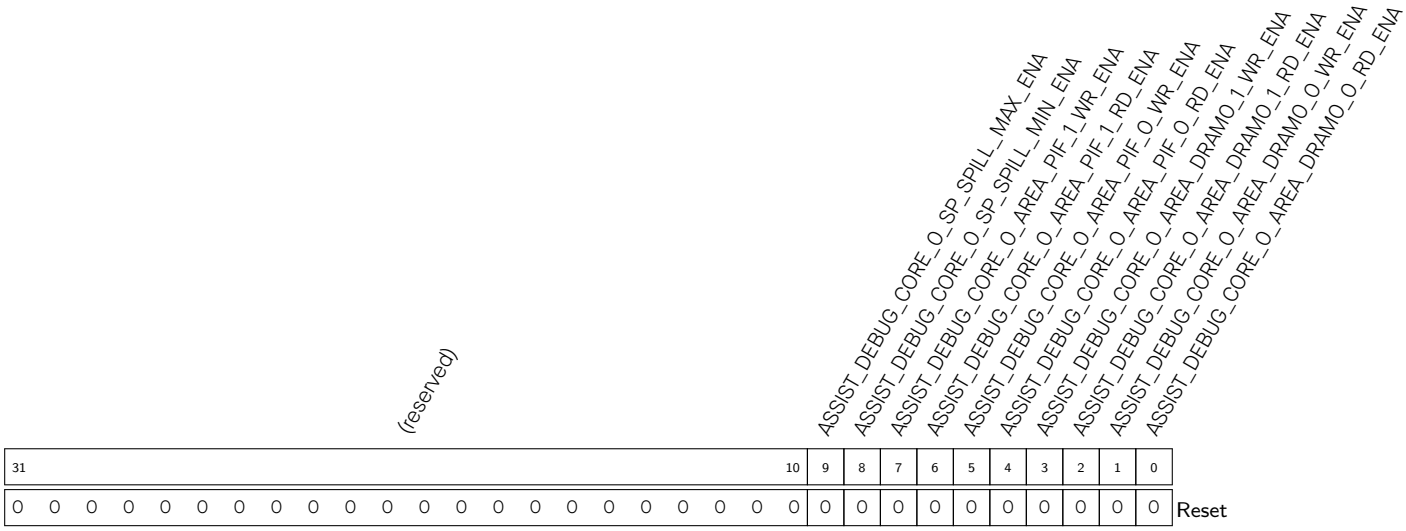
Register 17.25. L2_MEM_MONITOR_DATE_REG (0x03FC)



L2_MEM_MONITOR_DATE Version control register. (R/W)

17.7.3 Other Registers

Register 17.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_ENA Configures whether to monitor read operations in HP CPU0 region 0 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_ENA Configures whether to monitor write operations in HP CPU0 region 0 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_ENA Configures whether to monitor read operations in HP CPU0 region 1 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_ENA Configures whether to monitor write operations in HP CPU0 region 1 by the HP CPU0 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_ENA Configures whether to monitor read operations in region 0 by the Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

Continued on the next page...

Register 17.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)

Continued from the previous page...

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in HP CPU0 region 0 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in HP CPU0 region 1 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in HP CPU0 region 1 by the HP CPU0 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_ENA Configures whether to monitor HP CPU0 SP going below the lower bound address of HP CPU0 SP monitored region.

0: Not monitor

1: Monitor

(R/W)

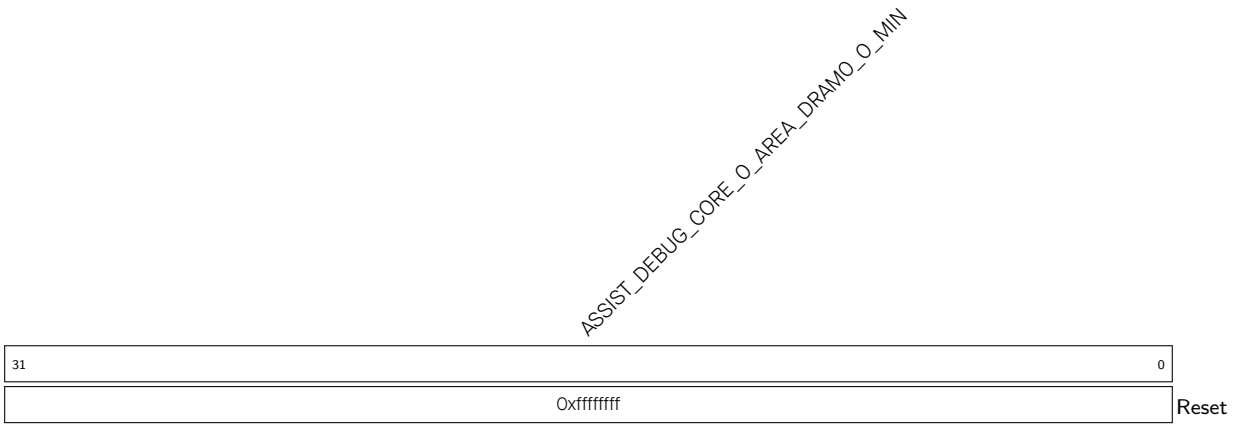
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_ENA Configures whether to monitor HP CPU0 SP going beyond the upper bound address of HP CPU0 SP monitored region.

0: Not monitor

1: Monitor

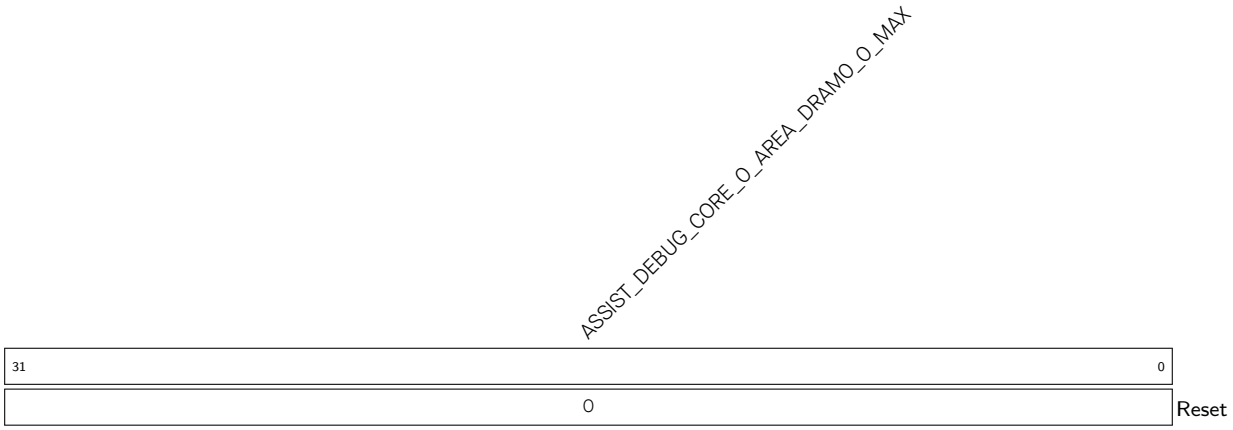
(R/W)

Register 17.27. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN Configures the lower bound address of HP CPU0 Data bus region 0. (R/W)

Register 17.28. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)



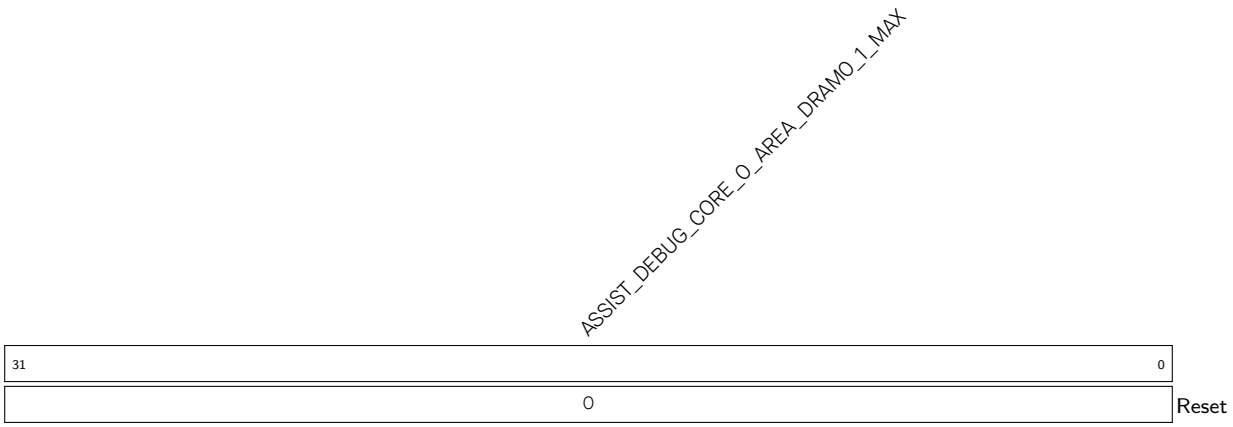
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX Configures the upper bound address of HP CPU0 Data bus region 0. (R/W)

Register 17.29. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN Configures the lower bound address of HP CPU0 Data bus region 1. (R/W)

Register 17.30. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX Configures the upper bound address of HP CPU0 Data bus region 1. (R/W)

Register 17.31. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG (0x0020)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN Configures the lower bound address of HP CPU0 Peripheral bus region 0. (R/W)

Register 17.32. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG (0x0024)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX Configures the upper bound address of HP CPU0 Peripheral bus region 0. (R/W)

Register 17.33. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



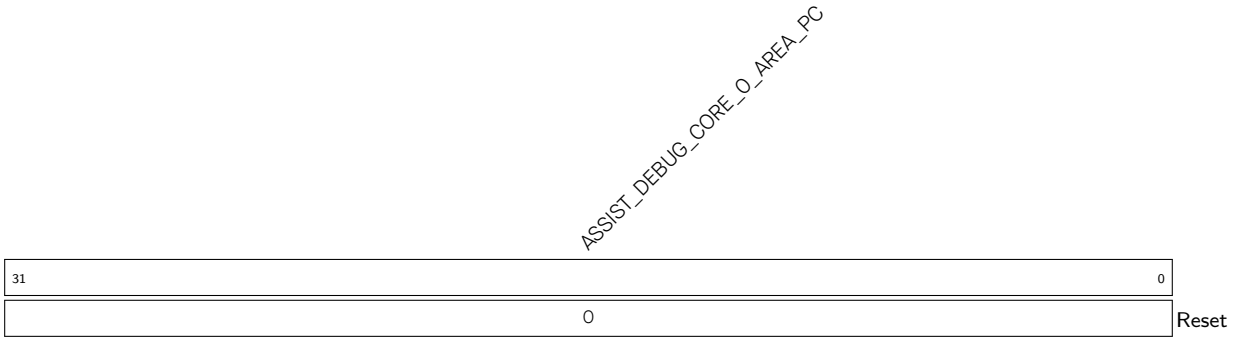
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN Configures the lower bound address of HP CPU0 Peripheral bus region 1. (R/W)

Register 17.34. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



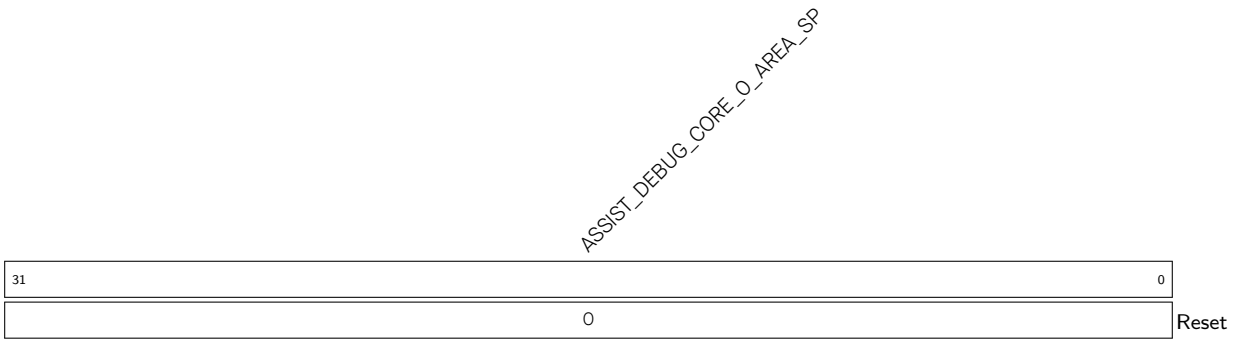
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX Configures the upper bound address of HP CPU0 Peripheral bus region 1. (R/W)

Register 17.35. ASSIST_DEBUG_CORE_0_AREA_PC_REG (0x0030)



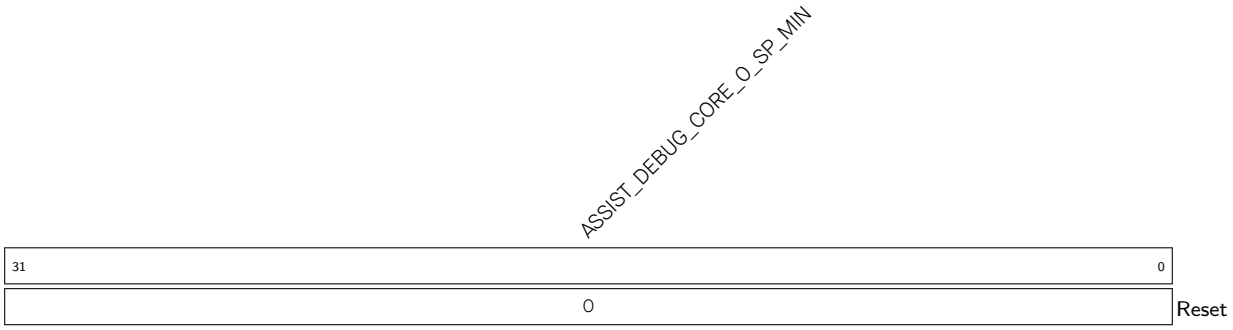
ASSIST_DEBUG_CORE_0_AREA_PC Represents the HP CPU0 PC value when an interrupt is triggered during region monitoring. (RO)

Register 17.36. ASSIST_DEBUG_CORE_0_AREA_SP_REG (0x0034)



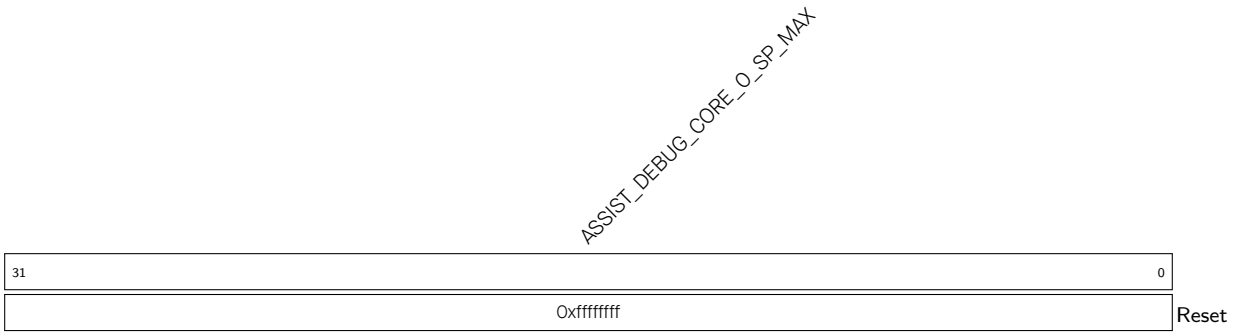
ASSIST_DEBUG_CORE_0_AREA_SP Represents the HP CPU0 SP value when an interrupt is triggered during region monitoring. (RO)

Register 17.37. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0038)



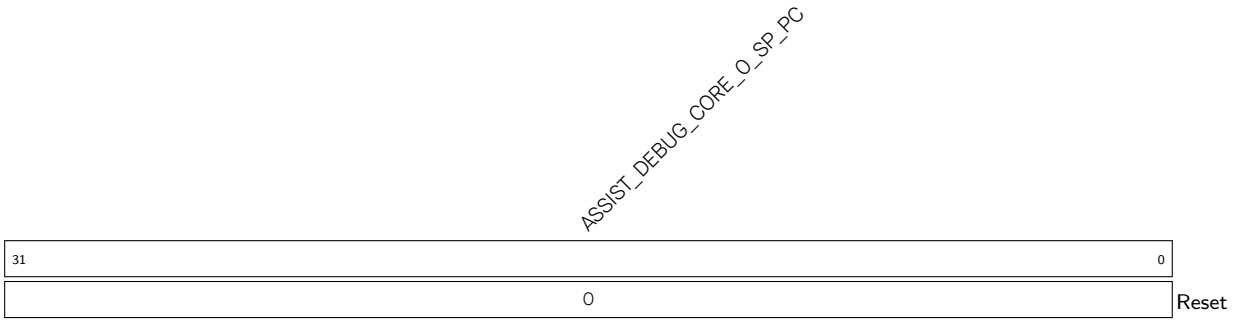
ASSIST_DEBUG_CORE_0_SP_MIN Configures the lower bound address of the HP CPU0 SP monitored region. (R/W)

Register 17.38. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x003C)



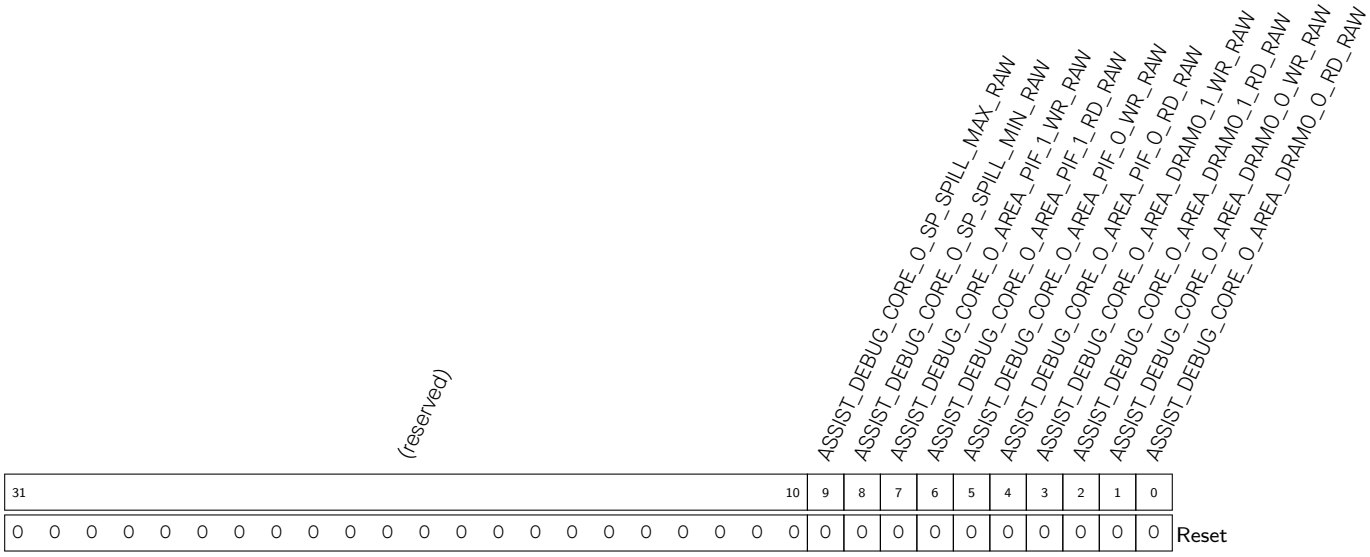
ASSIST_DEBUG_CORE_0_SP_MAX Configures the upper bound address of the HP CPU0 SP monitored region. (R/W)

Register 17.39. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0040)



ASSIST_DEBUG_CORE_0_SP_PC Represents the HP CPU0 PC value during SP monitoring. (RO)

Register 17.40. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (RO)

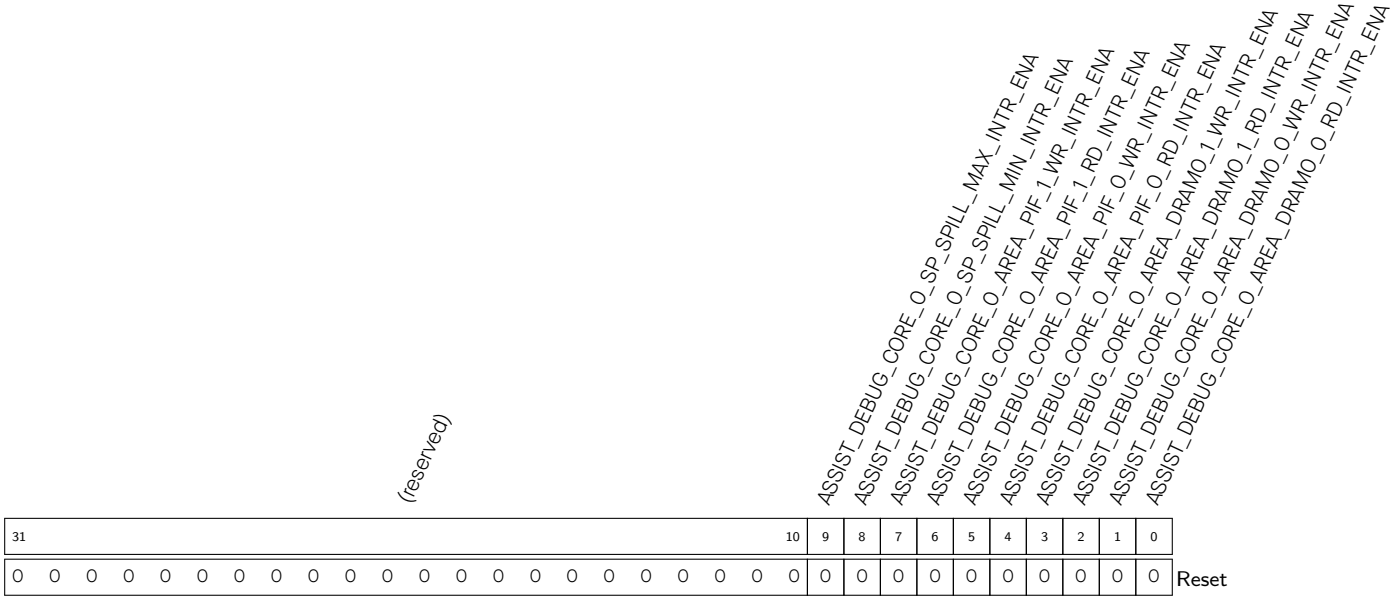
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW The raw interrupt status of ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (RO)

Register 17.41. ASSIST_DEBUG_CORE_0_INTR_ENA_REG (0x0008)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (R/W)

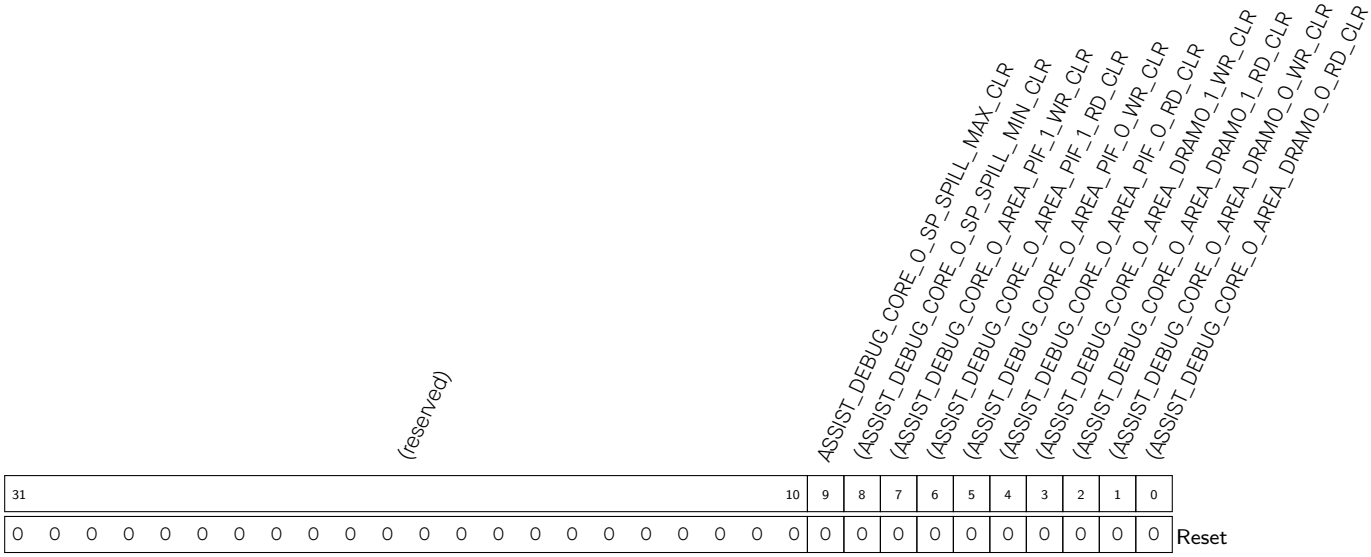
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (R/W)

Register 17.42. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT. (WT)

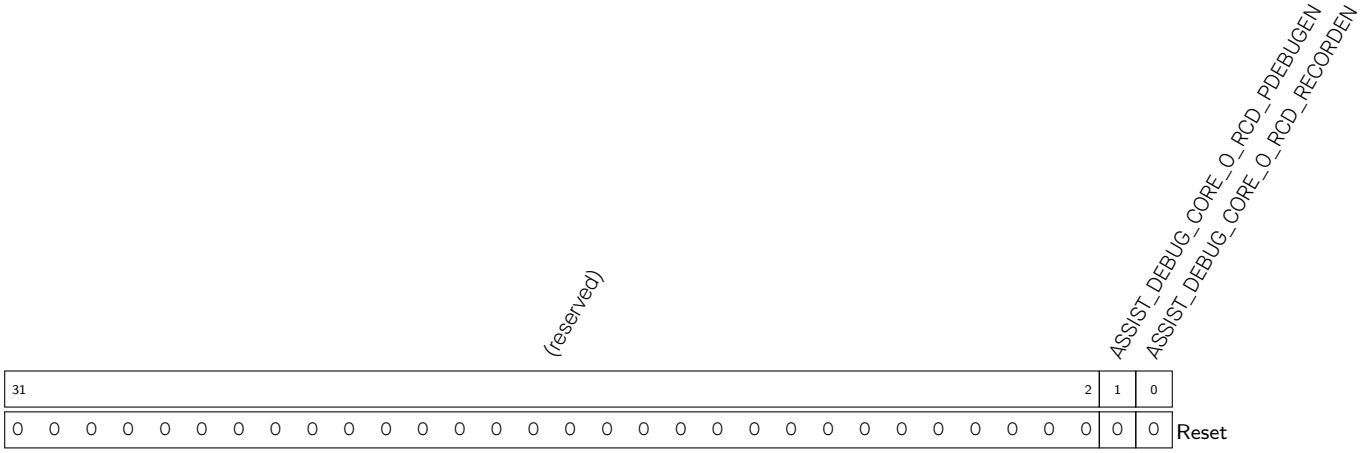
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_SP_SPILL_MIN_INT. (WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR Write 1 to clear AS-SIST_DEBUG_CORE_0_SP_SPILL_MAX_INT. (WT)

Register 17.43. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x0044)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN Configures whether to enable HP CPU0 PC logging.

0: Disable

1: ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG starts to record HP CPU0 PC in real time

(R/W)

ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN Configures whether to enable HP CPU0 debugging.

0: Disable

1: HP CPU0 outputs HP CPU0 PC

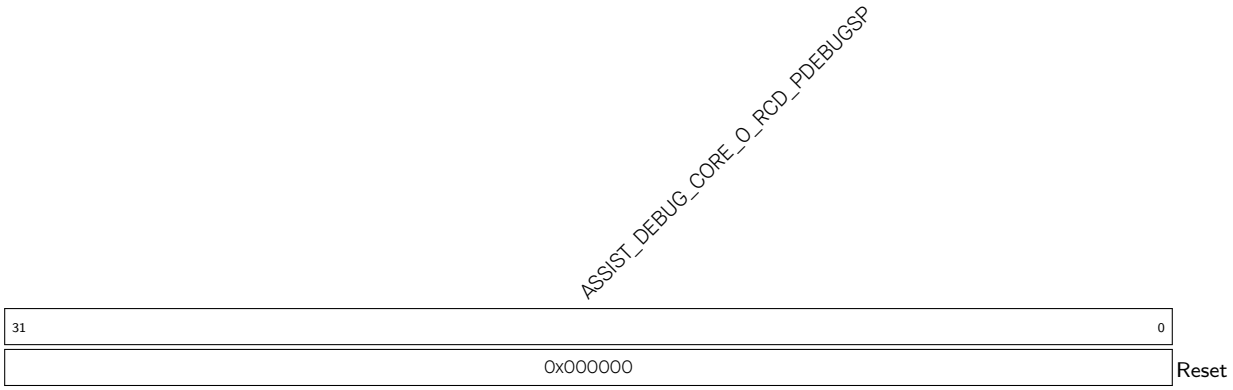
(R/W)

Register 17.44. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0048)



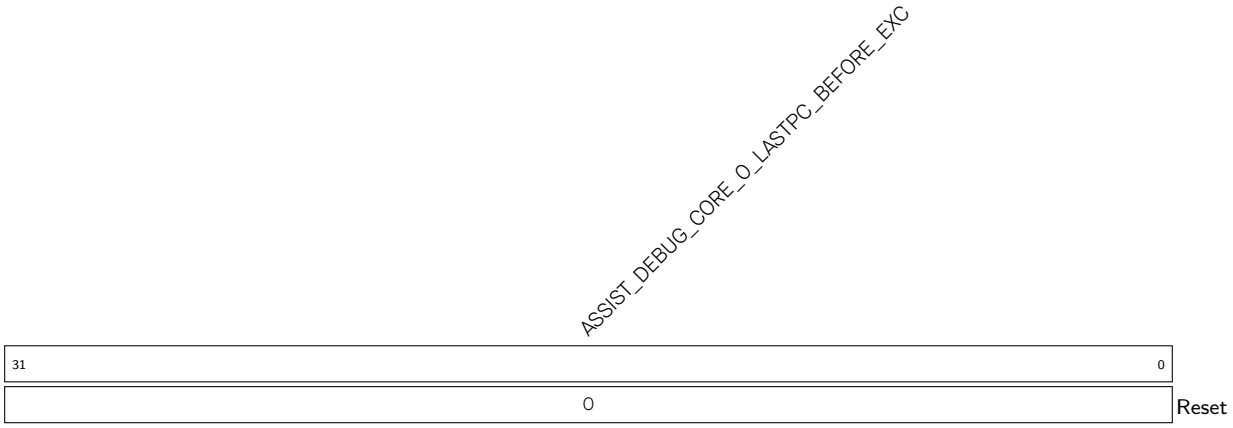
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC Represents the PC value at HP CPU0 reset. (RO)

Register 17.45. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x004C)



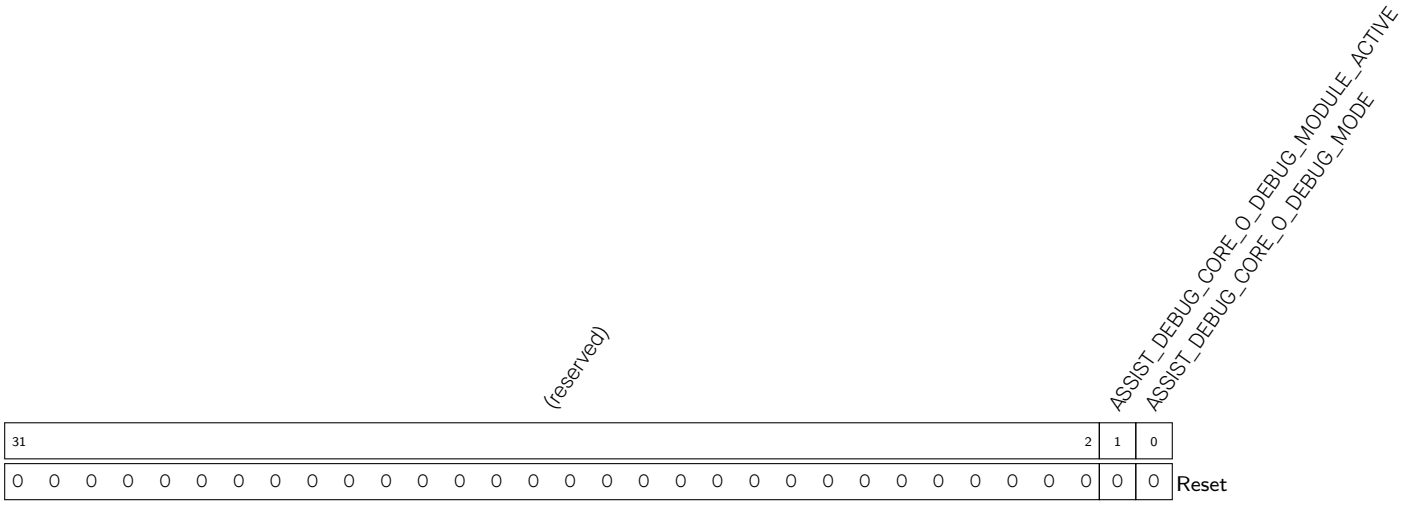
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP Represents the HP CPU0 SP. (RO)

Register 17.46. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC Represents the HP CPU0 PC of the last command before the HP CPU0 enters exception. (RO)

Register 17.47. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x0074)



ASSIST_DEBUG_CORE_0_DEBUG_MODE Represents whether HP CPU0 is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

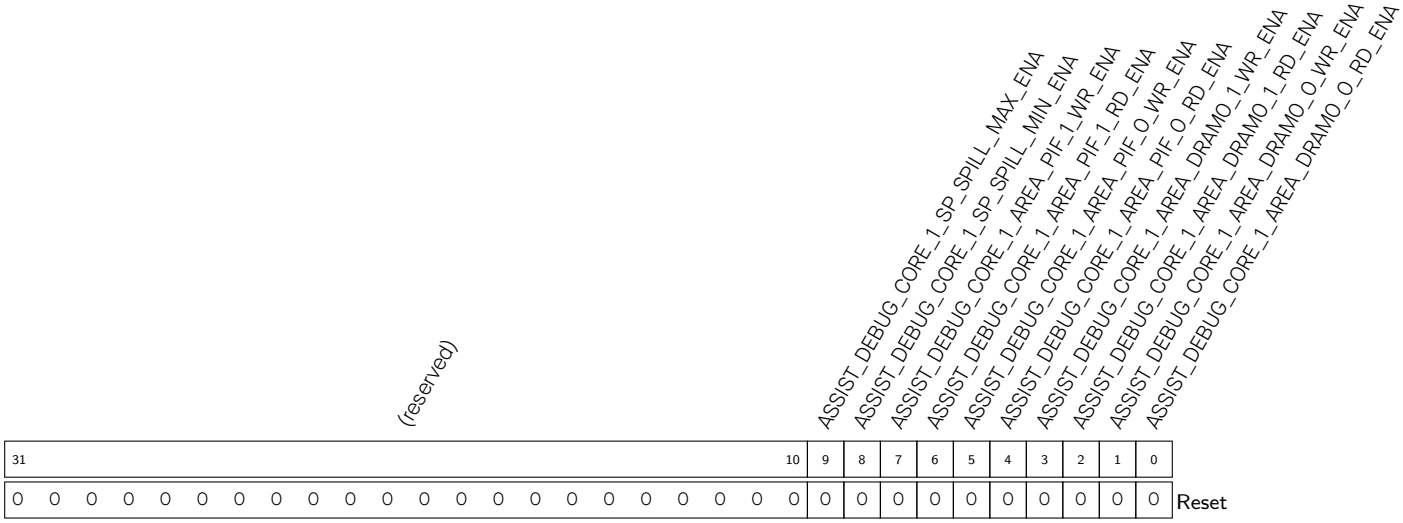
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE Represents the status of the HP CPU0 debug module.

1: Active status

Other: Inactive status

(RO)

Register 17.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0080)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_ENA Configures whether to monitor read operations in HP CPU1 region 0 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_ENA Configures whether to monitor write operations in HP CPU1 region 0 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_ENA Configures whether to monitor read operations in HP CPU1 region 1 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_ENA Configures whether to monitor write operations in HP CPU1 region 1 by the HP CPU1 Data bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_ENA Configures whether to monitor read operations in HP CPU1 region 0 by the HP CPU1 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

Continued on the next page...

Register 17.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0000)

Continued from the previous page...

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_ENA Configures whether to monitor write operations in HP CPU1 region 0 by the HP CPU1 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_ENA Configures whether to monitor read operations in HP CPU1 region 1 by the HP CPU1 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_ENA Configures whether to monitor write operations in HP CPU1 region 1 by the HP CPU1 Peripheral bus.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_ENA Configures whether to monitor HP CPU1 SP going below the lower bound address of HP CPU1 SP monitored region.

0: Not monitor

1: Monitor

(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_ENA Configures whether to monitor HP CPU1 SP going beyond the upper bound address of HP CPU1 SP monitored region.

0: Not monitor

1: Monitor

(R/W)

Register 17.49. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG (0x0090)



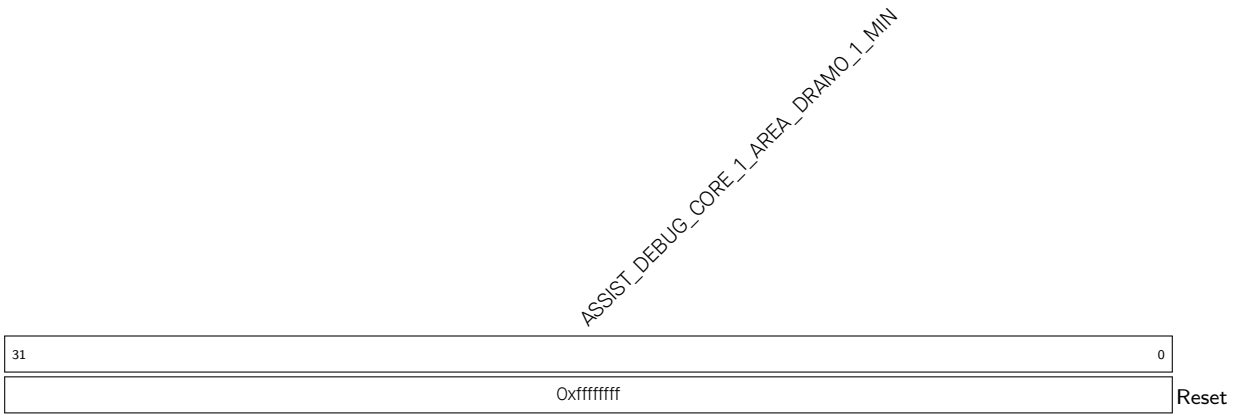
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN Configures the lower bound address of HP CPU1 Data bus region 0. (R/W)

Register 17.50. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG (0x0094)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX Configures the upper bound address of HP CPU1 Data bus region 0. (R/W)

Register 17.51. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG (0x0098)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN Configures the lower bound address of HP CPU1 Data bus region 1. (R/W)

Register 17.52. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG (0x009C)



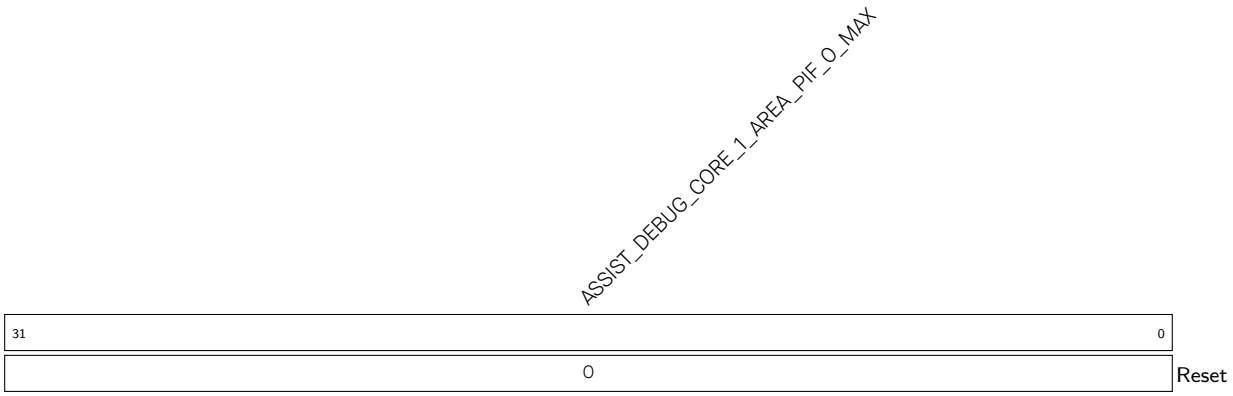
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX Configures the upper bound address of HP CPU1 Data bus region 1. (R/W)

Register 17.53. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG (0x00A0)



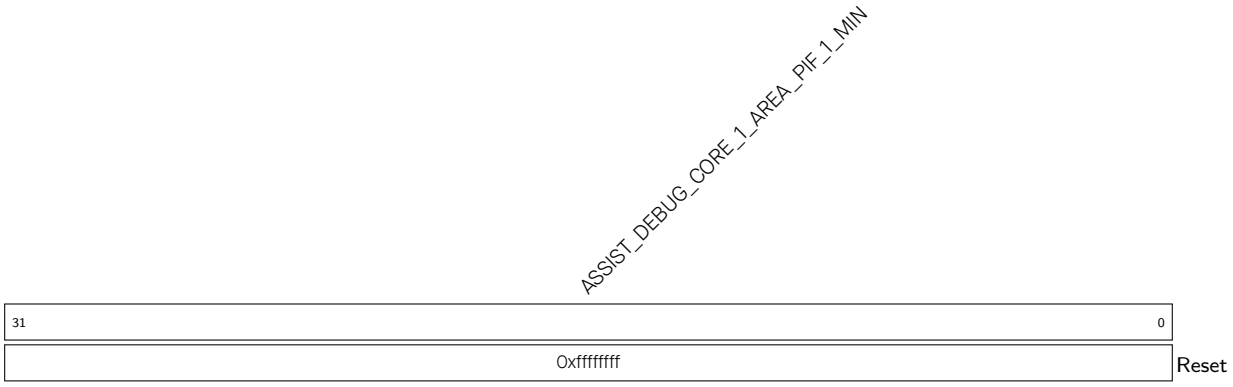
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN Configures the lower bound address of HP CPU1 Peripheral bus region 0. (R/W)

Register 17.54. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG (0x00A4)



ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX Configures the upper bound address of HP CPU1 Peripheral bus region 0. (R/W)

Register 17.55. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG (0x00A8)



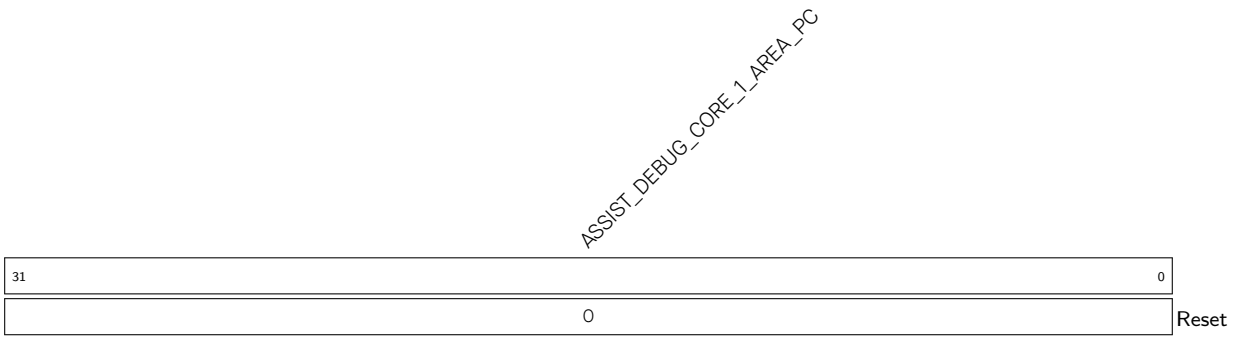
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN Configures the lower bound address of HP CPU1 Peripheral bus region 1. (R/W)

Register 17.56. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG (0x00AC)



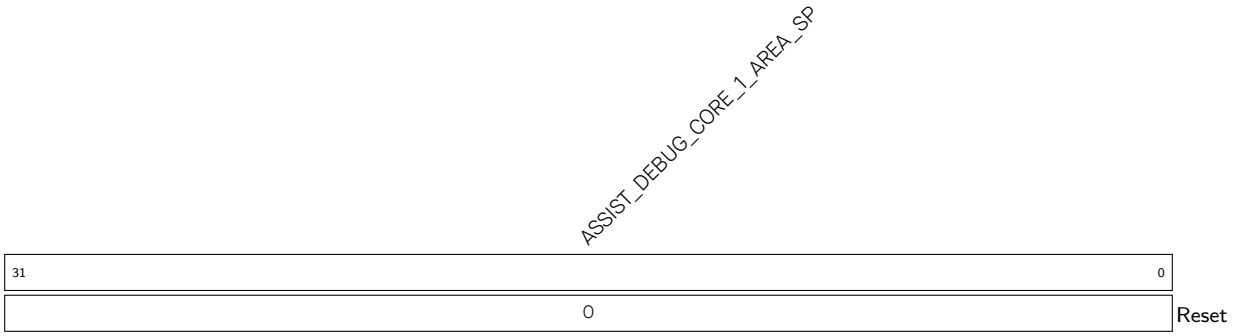
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX Configures the upper bound address of HP CPU1 Peripheral bus region 1. (R/W)

Register 17.57. ASSIST_DEBUG_CORE_1_AREA_PC_REG (0x00B0)



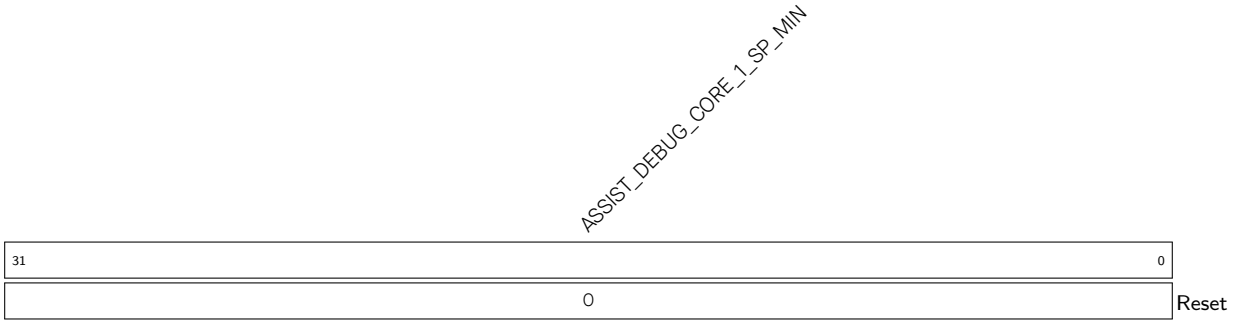
ASSIST_DEBUG_CORE_1_AREA_PC Represents the HP CPU1 PC value when an interrupt is triggered during region monitoring. (RO)

Register 17.58. ASSIST_DEBUG_CORE_1_AREA_SP_REG (0x00B4)



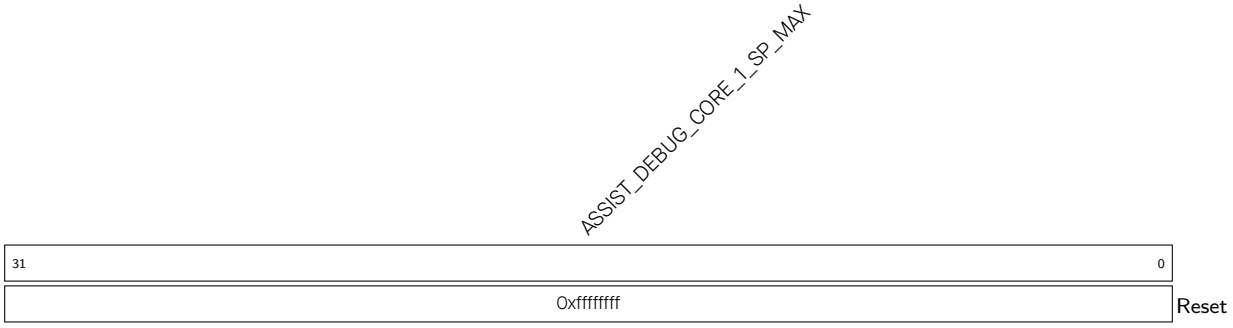
ASSIST_DEBUG_CORE_1_AREA_SP Represents the HP CPU1 SP value when an interrupt is triggered during region monitoring. (RO)

Register 17.59. ASSIST_DEBUG_CORE_1_SP_MIN_REG (0x00B8)



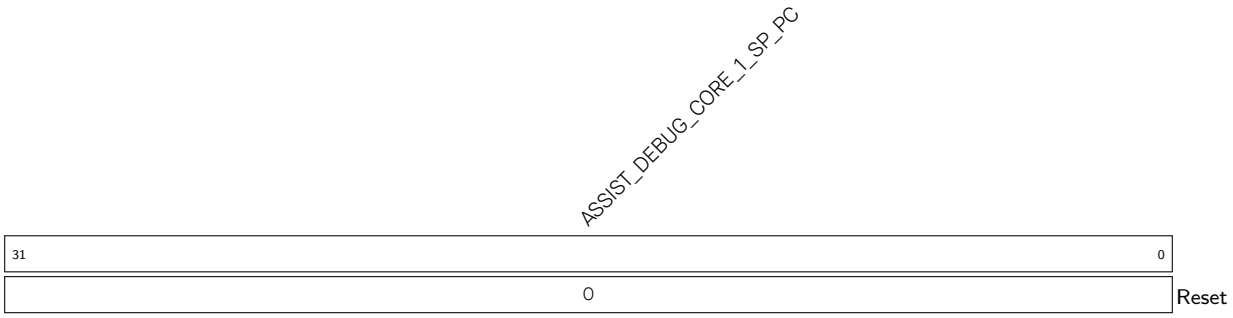
ASSIST_DEBUG_CORE_1_SP_MIN Configures the lower bound address of the HP CPU1 SP monitored region. (R/W)

Register 17.60. ASSIST_DEBUG_CORE_1_SP_MAX_REG (0x00BC)



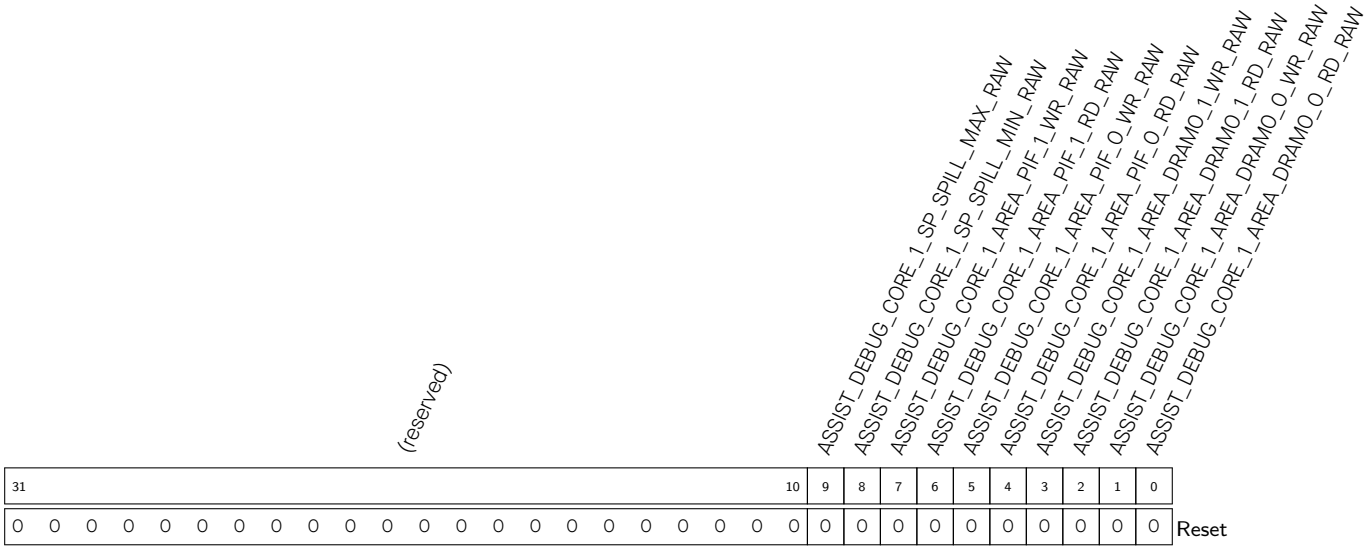
ASSIST_DEBUG_CORE_1_SP_MAX Configures the upper bound address of the HP CPU1 SP monitored region. (R/W)

Register 17.61. ASSIST_DEBUG_CORE_1_SP_PC_REG (0x00C0)



ASSIST_DEBUG_CORE_1_SP_PC Represents the HP CPU1 PC value during SP monitoring. (RO)

Register 17.62. ASSIST_DEBUG_CORE_1_INTR_RAW_REG (0x0084)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (RO)

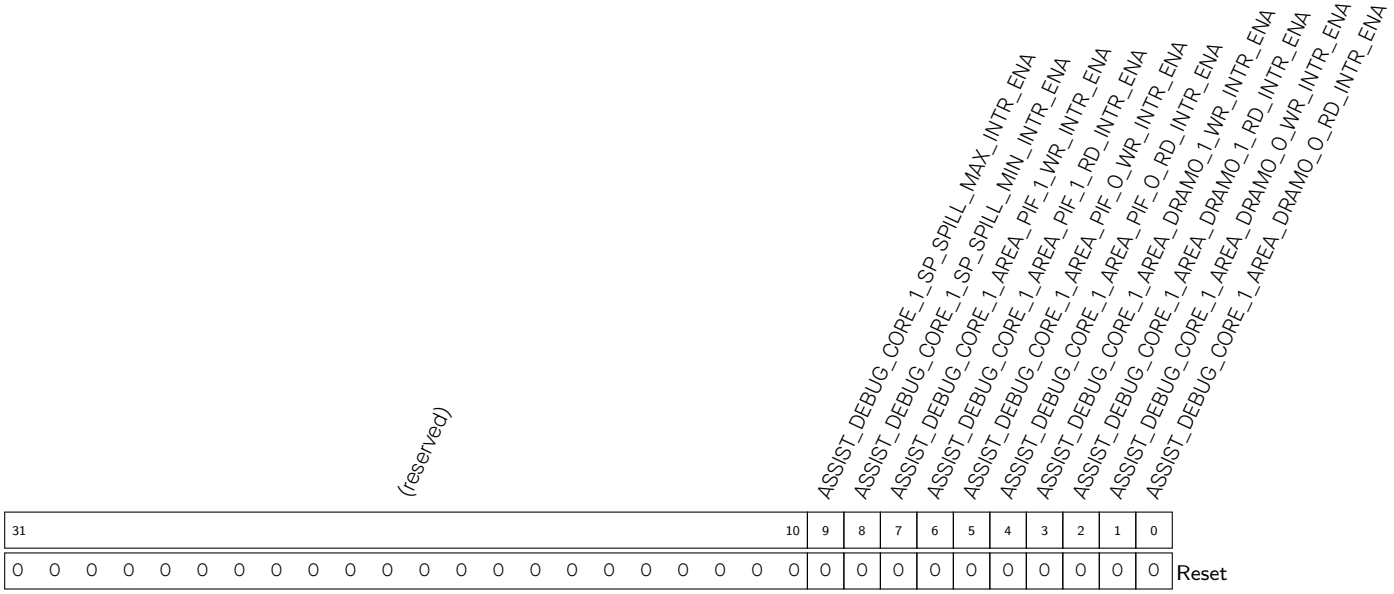
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_RAW The raw interrupt status of ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (RO)

Register 17.63. ASSIST_DEBUG_CORE_1_INTR_ENA_REG (0x0088)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (R/W)

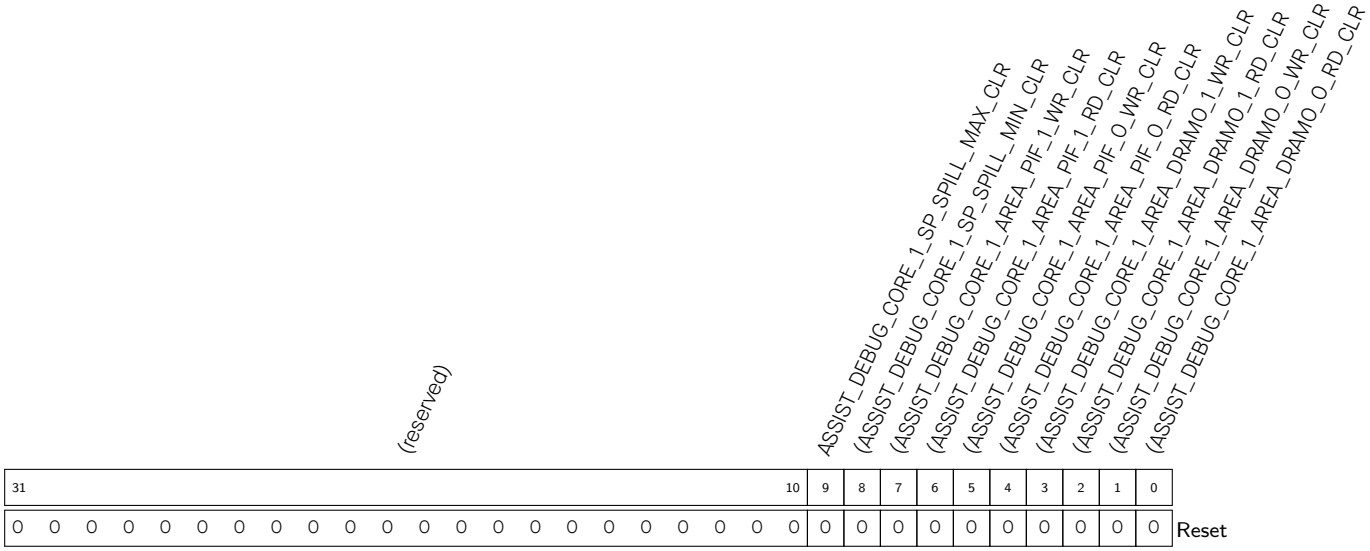
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INTR_ENA Write 1 to enable ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (R/W)

Register 17.64. ASSIST_DEBUG_CORE_1_INTR_CLR_REG (0x008C)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT. (WT)

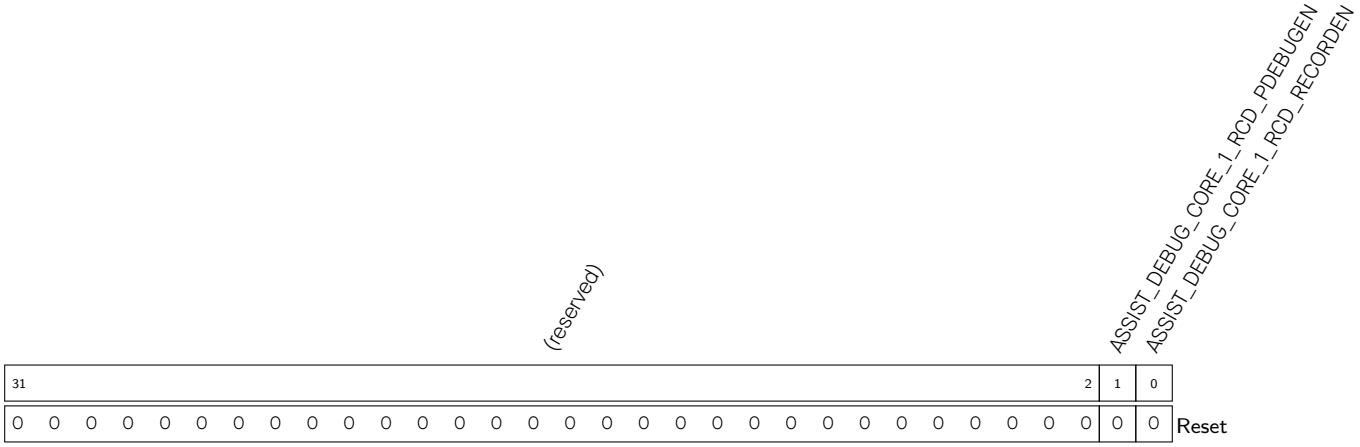
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT. (WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT. (WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_SP_SPILL_MIN_INT. (WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_CLR Write 1 to clear AS-SIST_DEBUG_CORE_1_SP_SPILL_MAX_INT. (WT)

Register 17.65. ASSIST_DEBUG_CORE_1_RCD_EN_REG (0x00C4)



ASSIST_DEBUG_CORE_1_RCD_RECORDEN Configures whether to enable HP CPU1 PC logging.

0: Disable

1: ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG starts to record HP CPU1 PC in real time

(R/W)

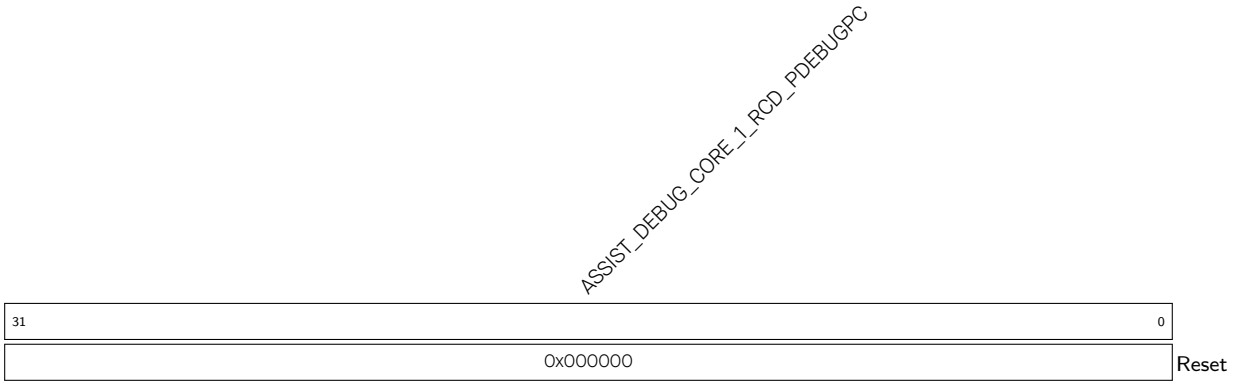
ASSIST_DEBUG_CORE_1_RCD_PDEBUGEN Configures whether to enable HP CPU1 debugging.

0: Disable

1: HP CPU1 outputs HP CPU1 PC

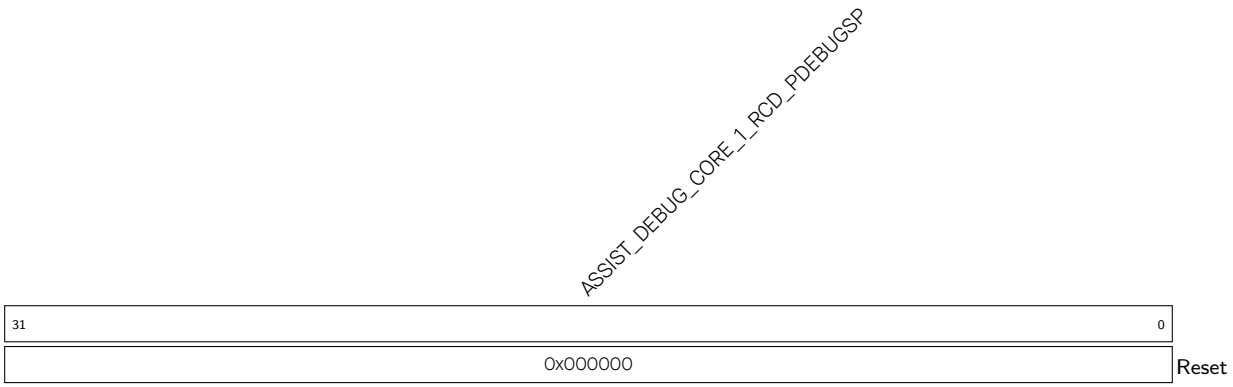
(R/W)

Register 17.66. ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG (0x00C8)



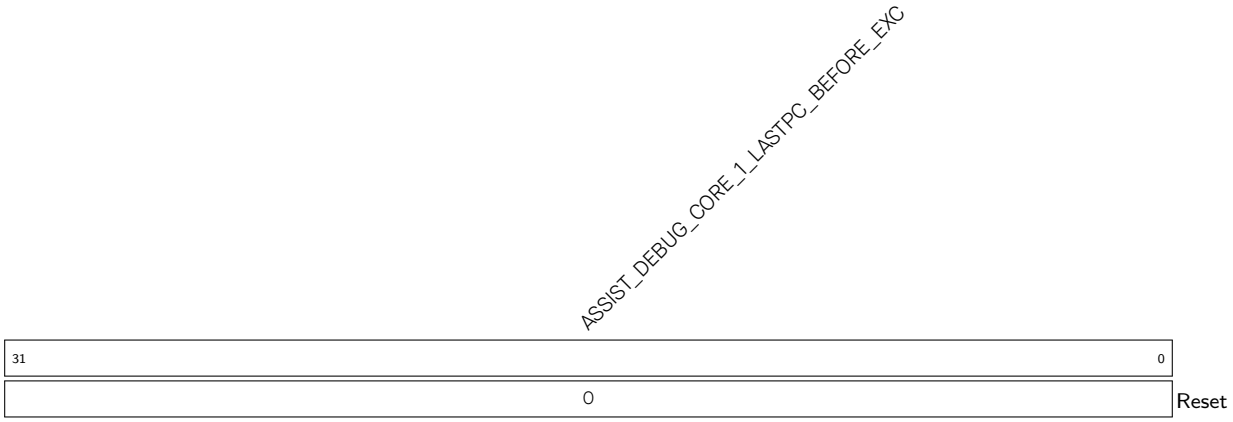
ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC Represents the PC value at HP CPU1 reset. (RO)

Register 17.67. ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG (0x00CC)



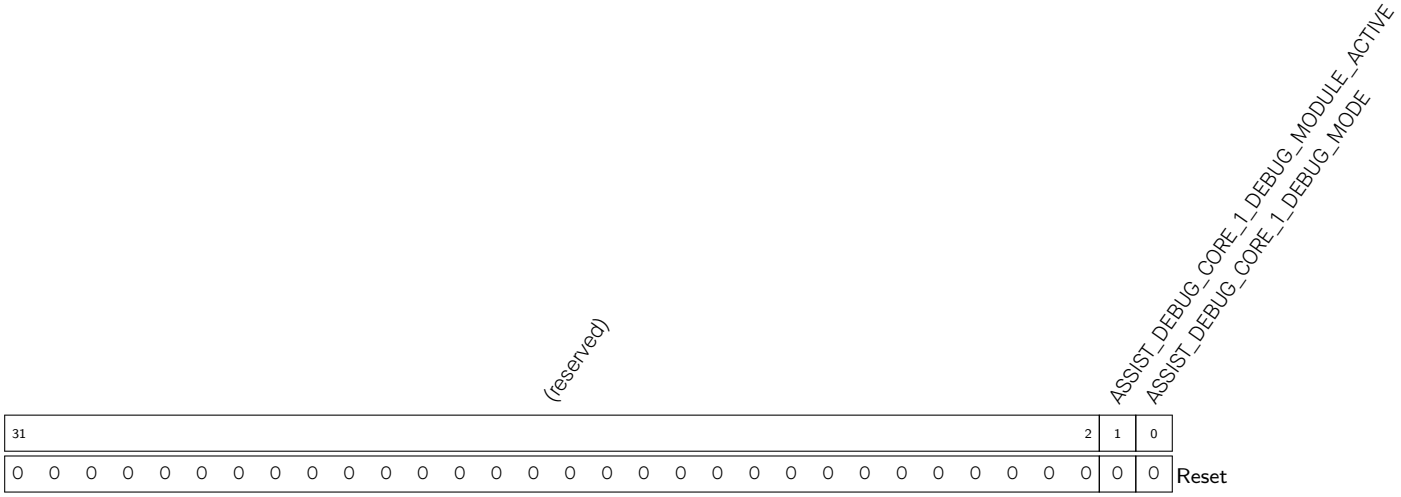
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP Represents the HP CPU1 SP. (RO)

Register 17.68. ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG (0x00F0)



ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXC Represents the HP CPU1 PC of the last command before the HP CPU1 enters exception. (RO)

Register 17.69. ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG (0x00F4)



ASSIST_DEBUG_CORE_1_DEBUG_MODE Represents whether HP CPU1 is in debugging mode.

1: In debugging mode

0: Not in debugging mode

(RO)

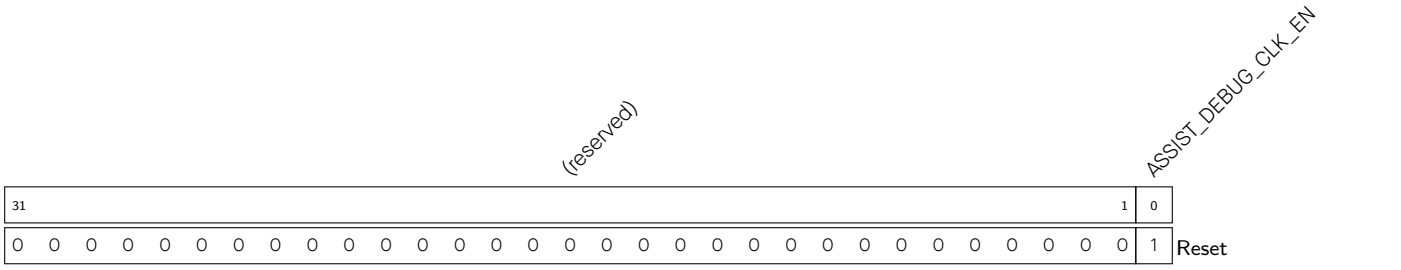
ASSIST_DEBUG_CORE_1_DEBUG_MODULE_ACTIVE Represents the status of the HP CPU1 debug module.

1: Active status

Other: Inactive status

(RO)

Register 17.70. ASSIST_DEBUG_CLOCK_GATE_REG (0x0108)



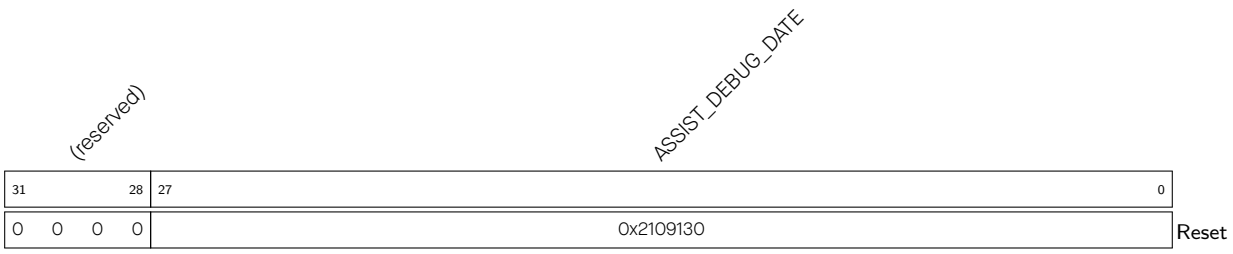
ASSIST_DEBUG_CLK_EN Configures whether to enable the register clock gating.

0: Disable

1: Enable

(R/W)

Register 17.71. ASSIST_DEBUG_DATE_REG (0x03FC)



ASSIST_DEBUG_DATE Version control register. (R/W)

Chapter 18

LP Mailbox

18.1 Overview

ESP32-P4 integrates an LP Mailbox module which provides an efficient inter-core communication mechanism between the LP CPU and HP CPU0/1. The LP Mailbox module comprises of sixteen 32-bit message registers that the LP CPU and HP CPU0/1 can use to store and exchange message. Inter-core communication between LP CPU and HP CPU0/1 is achieved through an interrupt mechanism implemented within the LP Mailbox module.

18.2 Features

ESP32-P4 LP Mailbox has the following features:

- Sixteen 32-bit message registers for inter-core communication
- LP CPU external interrupt signal: MB_LP_INTR
- HP CPU*x* (*x*: 0 ~ 1) external interrupt signal: MB_HP_INTR

18.3 Functional Description

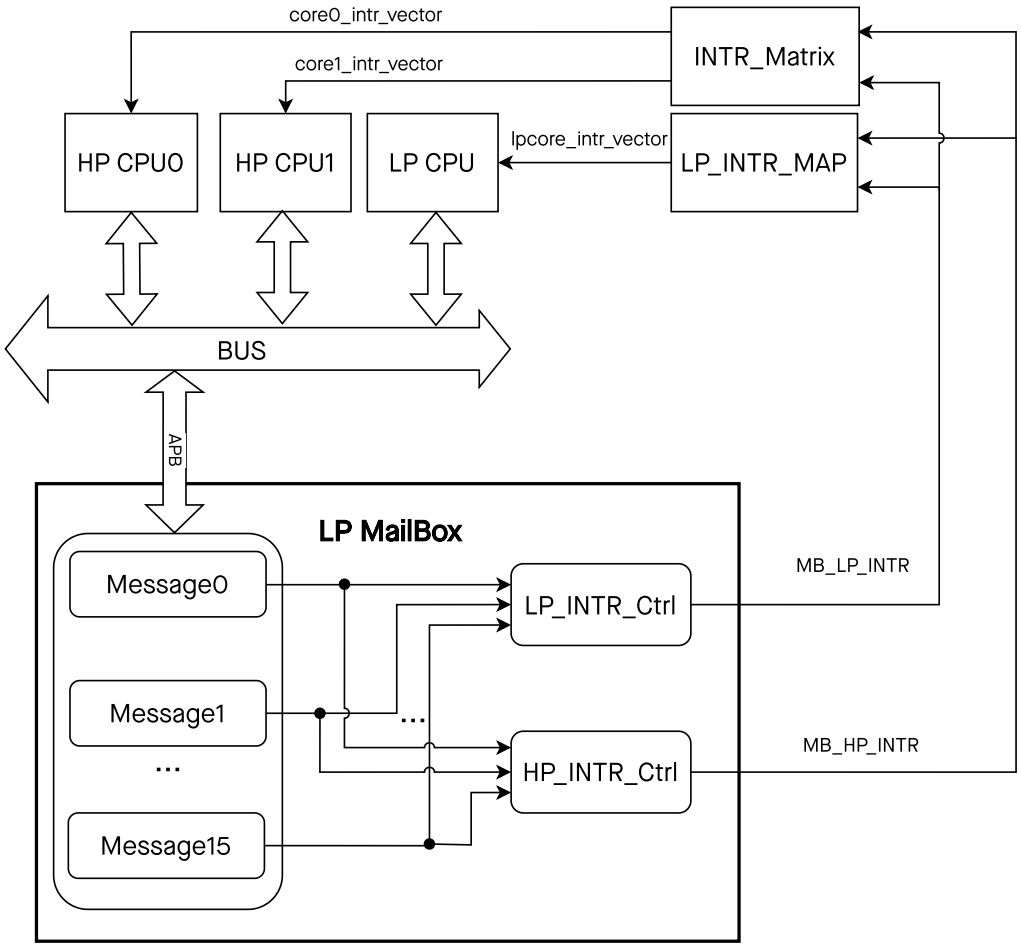


Figure 18.3-1. LP Mailbox Block Diagram

18.3.1 Message Registers

The LP Mailbox provides sixteen 32-bit message registers in total, denoted as MB_MESSAGE_0_REG ~ MB_MESSAGE_15_REG, for caching inter-core communication messages. Both LP CPU and HP CPU0/1 can read and write these message registers to pass or retrieve message.

Read and write message registers is done through the APB bus, using APB_CLK, which is the only one clock in LP Mailbox. For more information about the clock, please refer to Chapter 8 *Reset and Clock*.

18.3.2 Interrupts

ESP32-P4 LP Mailbox can generate the following external interrupt signals:

- MB_LP_INTR
- MB_HP_INTR

MB_LP_INTR is directly mapped to LP CPU (see *Low-Power CPU*), and MB_HP_INTR can be mapped to HP CPU0/1 through the Interrupt Matrix. These interrupt signals are generated by internal interrupt sources of the

LP Mailbox module. Table 18.3-1 lists the internal interrupt sources, interrupt trigger conditions, and generated interrupt signals of the LP Mailbox.

Table 18.3-1. LP Mailbox's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
MB_HP_*m*_INT	Write to MB_MESSAGE_*m*_REG	MB_HP_INTR
MB_LP_*n*_INT	Write to MB_MESSAGE_*n*_REG	MB_LP_INTR

Note:

- *m* and *n* can be 0 ~ 15.
- MB_HP_INTR can also serve as an external interrupt source for LP CPU and be mapped to LP CPU by LP_INTR_MAP as illustrated in Figure 18.3-1. 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.

18.3.3 Inter-Core Communication

When communication is required between LP CPU and HP CPU*x*, subset of or all message registers in the LP Mailbox can be utilized to share message as needed. The synchronization of message can be achieved through the corresponding interrupts. During communication, software must ensure that only the sender is allowed to write to the message registers to prevent data corruption.

- Example of initiating communication from HP CPU0 to LP CPU using MB_MESSAGE_*n*_REG:
  - Set MB_LP_*n*_INT_ENA to enable the MB_LP_*n*_INT interrupt.
  - HP CPU0 writes to MB_MASSEGE_*n*_REG, caching shared message.
  - LP CPU checks MB_LP_INT_ST_REG interrupt status register, finding the MB_LP_*n*_ST status bit set.
  - LP CPU reads MB_MASSEGE_*n*_REG to retrieve shared message.
- Example of initiating communication from LP CPU to HP CPU0:
  - Set MB_HP_*m*_INT_ENA to enable the MB_HP_*m*_INT interrupt.
  - Configure the Interrupt Matrix to map MB_HP_INTR to HP CPU0. See Chapter 10 *Interrupt Matrix*.
  - LP CPU writes to MB_MASSEGE_*m*_REG, caching shared messages.
  - HP CPU0 checks the MB_HP_INT_ST_REG interrupt status register, finding the MB_HP_*m*_ST status bit set.
  - HP CPU0 reads MB_MASSEGE_*m*_REG to retrieve shared messages.

18.4 Register Summary

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

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

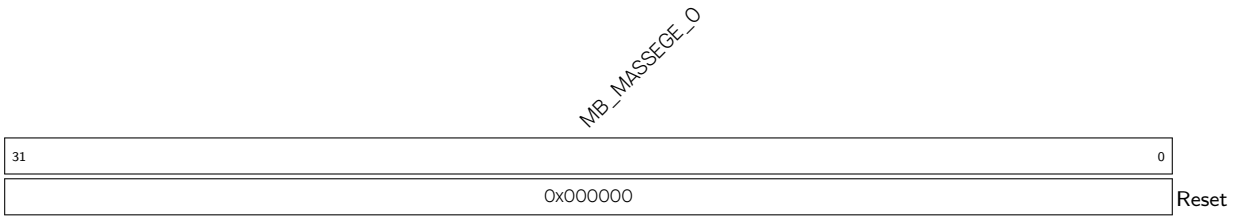
Name	Description	Address	Access
Configuration Registers			
MB_MASSEGE_0_REG	Message register 0	0x0000	R/W
MB_MASSEGE_1_REG	Message register 1	0x0004	R/W
MB_MASSEGE_2_REG	Message register 2	0x0008	R/W
MB_MASSEGE_3_REG	Message register 3	0x000C	R/W
MB_MASSEGE_4_REG	Message register 4	0x0010	R/W
MB_MASSEGE_5_REG	Message register 5	0x0014	R/W
MB_MASSEGE_6_REG	Message register 6	0x0018	R/W
MB_MASSEGE_7_REG	Message register 7	0x001C	R/W
MB_MASSEGE_8_REG	Message register 8	0x0020	R/W
MB_MASSEGE_9_REG	Message register 9	0x0024	R/W
MB_MASSEGE_10_REG	Message register 10	0x0028	R/W
MB_MASSEGE_11_REG	Message register 11	0x002C	R/W
MB_MASSEGE_12_REG	Message register 12	0x0030	R/W
MB_MASSEGE_13_REG	Message register 13	0x0034	R/W
MB_MASSEGE_14_REG	Message register 14	0x0038	R/W
MB_MASSEGE_15_REG	Message register 15	0x003C	R/W
MB_REG_CLK_EN_REG	Clock gate register	0x0060	R/W
Interrupt Registers			
MB_LP_INT_RAW_REG	MB_LP_INTR raw bits	0x0040	RO
MB_LP_INT_ST_REG	MB_LP_INTR status bits	0x0044	RO
MB_LP_INT_ENA_REG	MB_LP_INTR enable bits	0x0048	R/W
MB_LP_INT_CLR_REG	MB_LP_INTR clear bits	0x004C	WO
MB_HP_INT_RAW_REG	MB_HP_INTR raw bits	0x0050	RO
MB_HP_INT_ST_REG	MB_HP_INTR status bits	0x0054	RO
MB_HP_INT_ENA_REG	MB_HP_INTR enable bits	0x0058	R/W
MB_HP_INT_CLR_REG	MB_HP_INTR clear bits	0x005C	WO

18.5 Registers

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

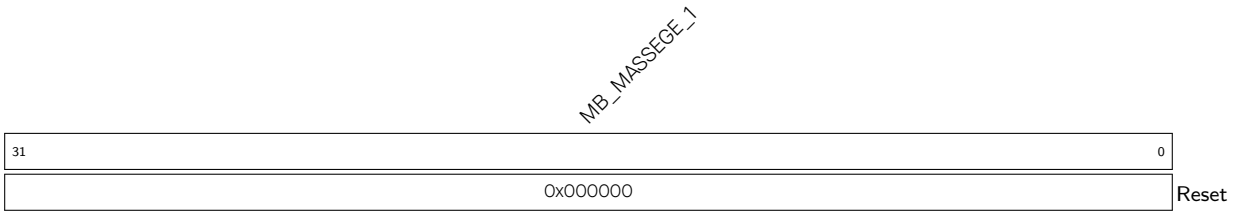
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 18.1. MB_MASSEGE_0_REG (0x0000)



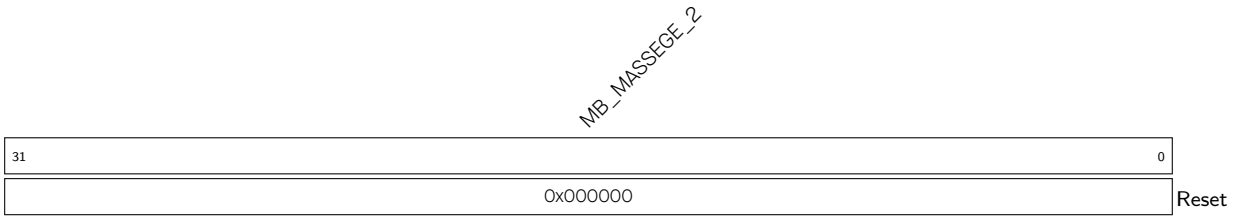
MB_MASSEGE_0 Message register 0. (R/W)

Register 18.2. MB_MASSEGE_1_REG (0x0004)



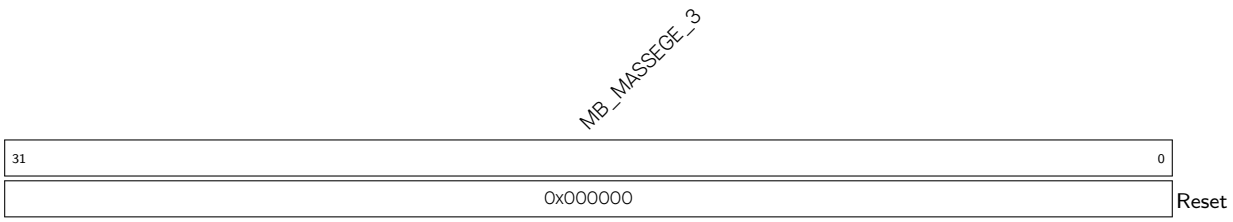
MB_MASSEGE_1 Message register 1. (R/W)

Register 18.3. MB_MASSEGE_2_REG (0x0008)



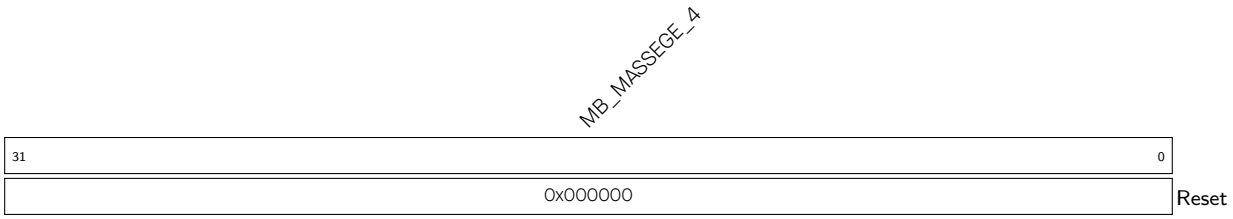
MB_MASSEGE_2 Message register 2. (R/W)

Register 18.4. MB_MASSEGE_3_REG (0x000C)



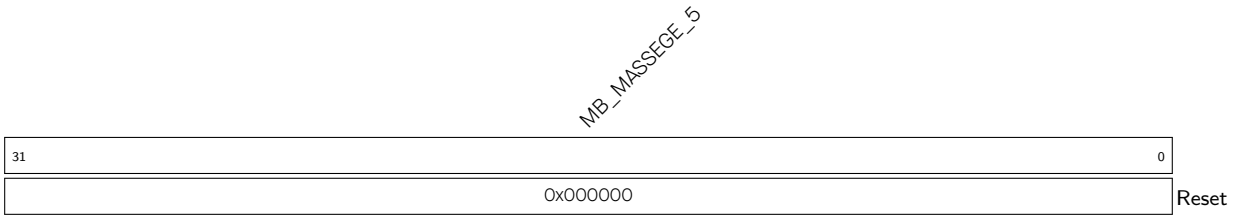
MB_MASSEGE_3 Message register 3. (R/W)

Register 18.5. MB_MASSEGE_4_REG (0x0010)



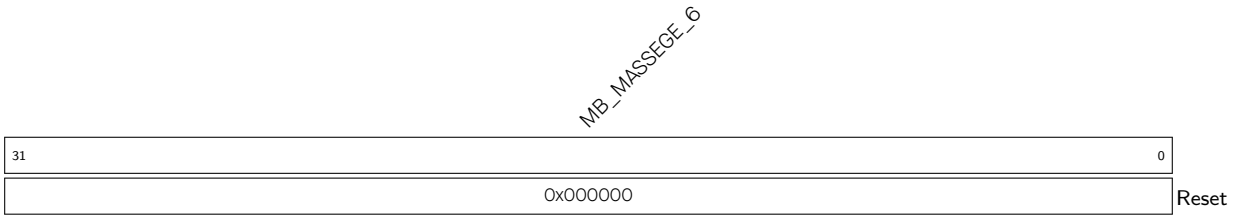
MB_MASSEGE_4 Message register 4. (R/W)

Register 18.6. MB_MASSEGE_5_REG (0x0014)



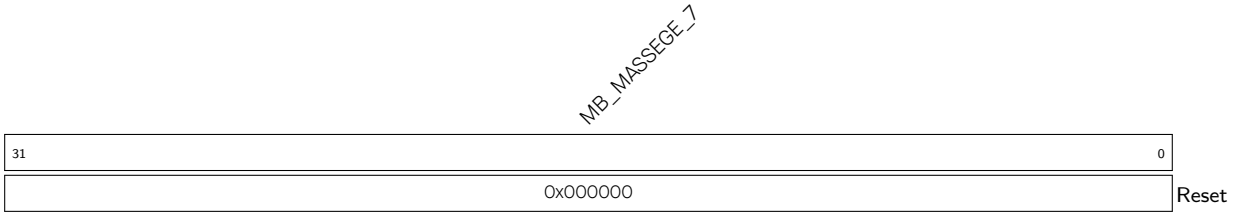
MB_MASSEGE_5 Message register 5. (R/W)

Register 18.7. MB_MASSEGE_6_REG (0x0018)



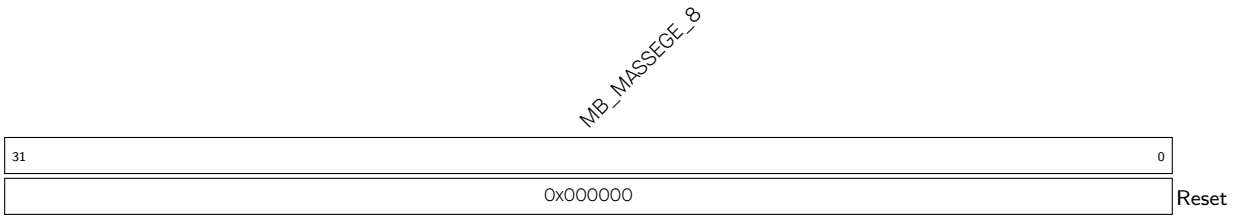
MB_MASSEGE_6 Message register 6. (R/W)

Register 18.8. MB_MASSEGE_7_REG (0x001C)



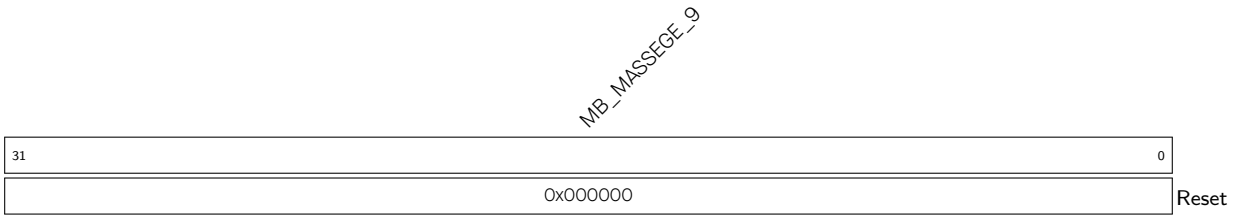
MB_MASSEGE_7 Message register 7. (R/W)

Register 18.9. MB_MASSEGE_8_REG (0x0020)



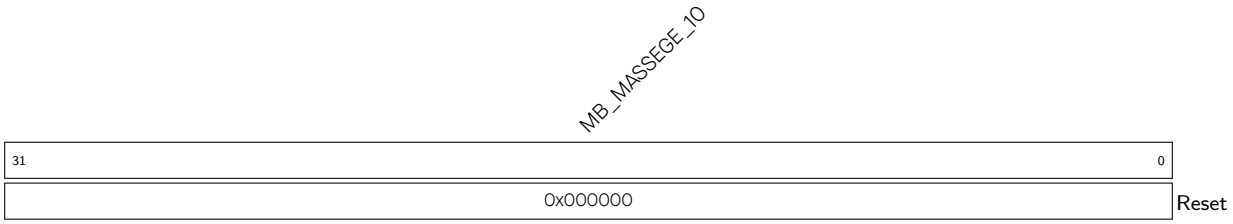
MB_MASSEGE_8 Message register 8. (R/W)

Register 18.10. MB_MASSEGE_9_REG (0x0024)



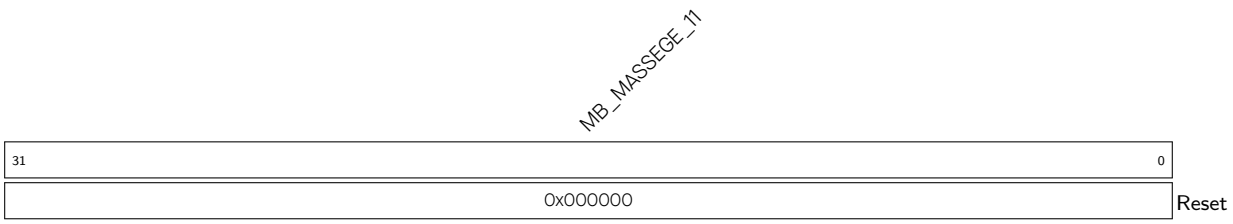
MB_MASSEGE_9 Message register 9. (R/W)

Register 18.11. MB_MASSEGE_10_REG (0x0028)



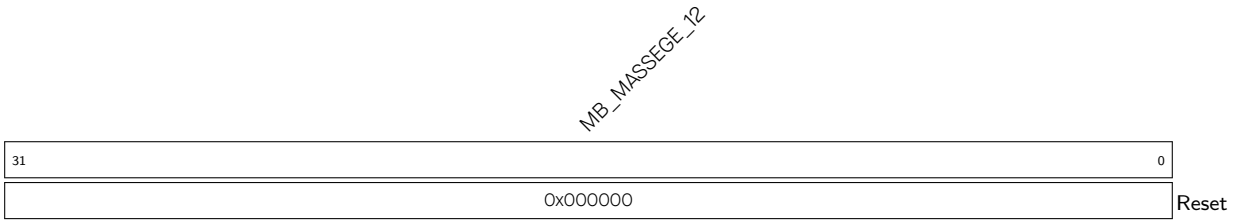
MB_MASSEGE_10 Message register 10. (R/W)

Register 18.12. MB_MASSEGE_11_REG (0x002C)



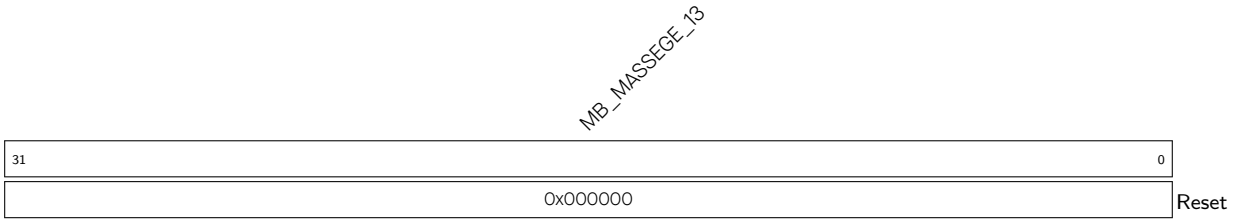
MB_MASSEGE_11 Message register 11. (R/W)

Register 18.13. MB_MASSEGE_12_REG (0x0030)



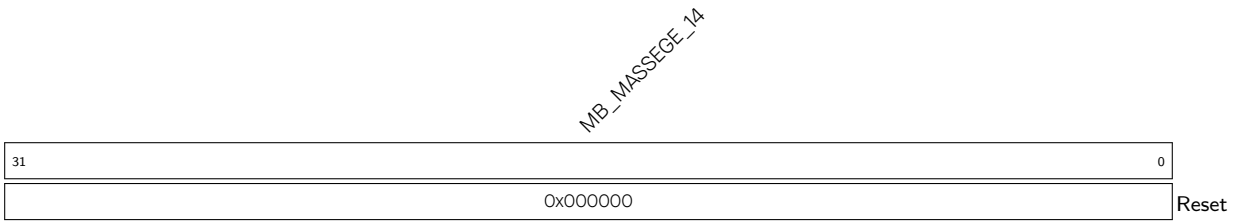
MB_MASSEGE_12 Message register 12. (R/W)

Register 18.14. MB_MASSEGE_13_REG (0x0034)



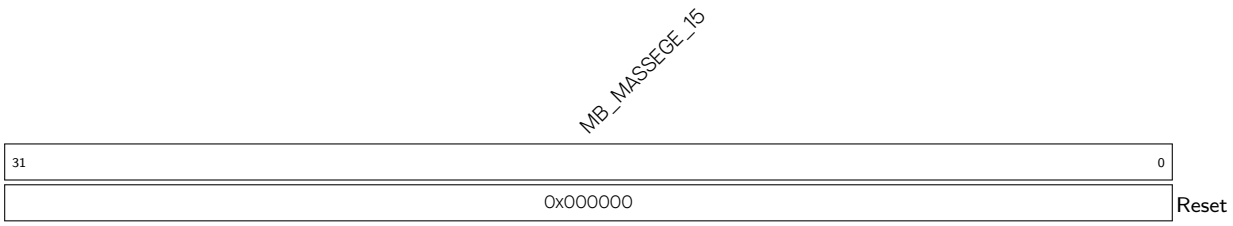
MB_MASSEGE_13 Message register 13. (R/W)

Register 18.15. MB_MASSEGE_14_REG (0x0038)



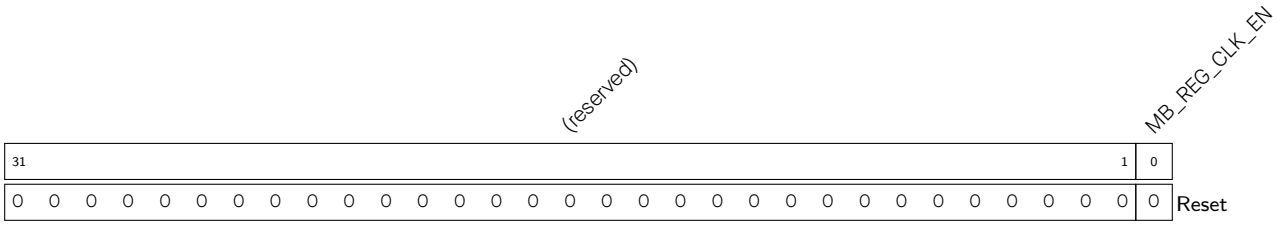
MB_MASSEGE_14 Message register 14. (R/W)

Register 18.16. MB_MASSEGE_15_REG (0x003C)



MB_MASSEGE_15 Message register 15. (R/W)

Register 18.17. MB_REG_CLK_EN_REG (0x0060)



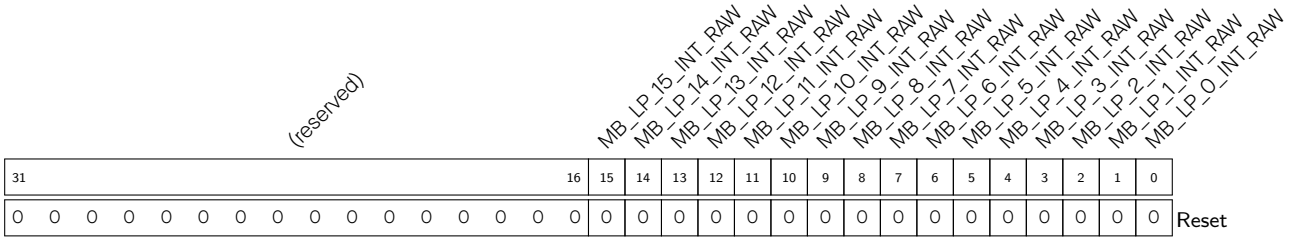
MB_REG_CLK_EN Configures register clock gating.

1: Register clock is always enabled for read and write operations.

0: Only enable clock for register read or write operations.

(R/W)

Register 18.18. 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)

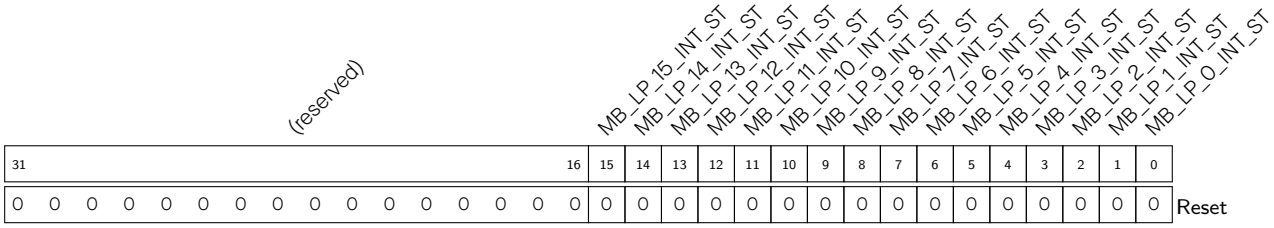
MB_LP_12_INT_RAW The raw interrupt bit of the MB_LP_12_INT interrupt. (RO)

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 18.19. 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)

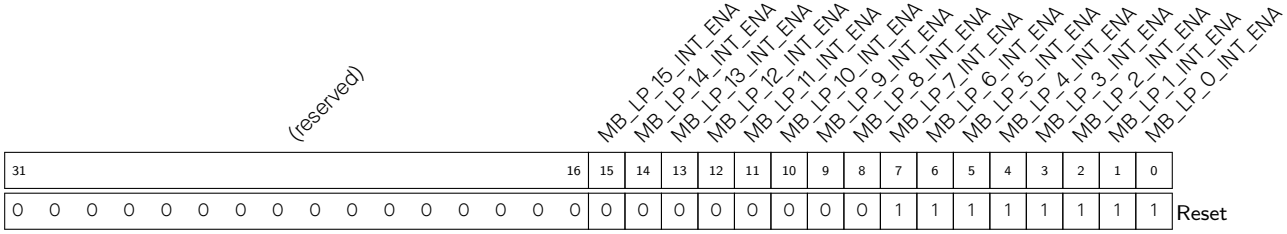
MB_LP_12_INT_ST The masked interrupt status bit of the MB_LP_12_INT interrupt. (RO)

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 18.20. 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)

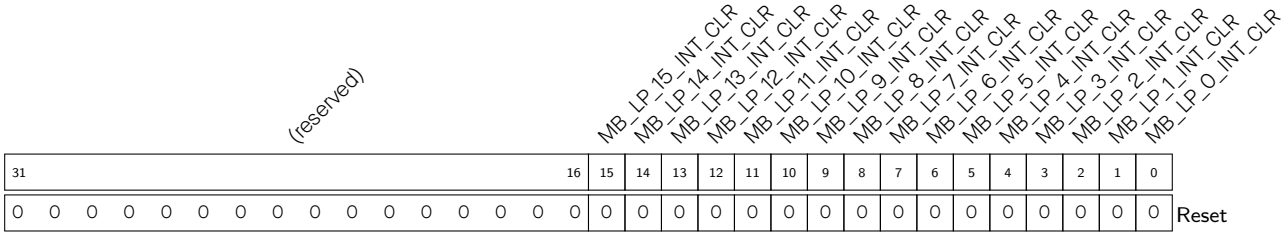
MB_LP_12_INT_ENA Write 1 to enable the MB_LP_12_INT interrupt. (R/W)

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 18.21. 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)

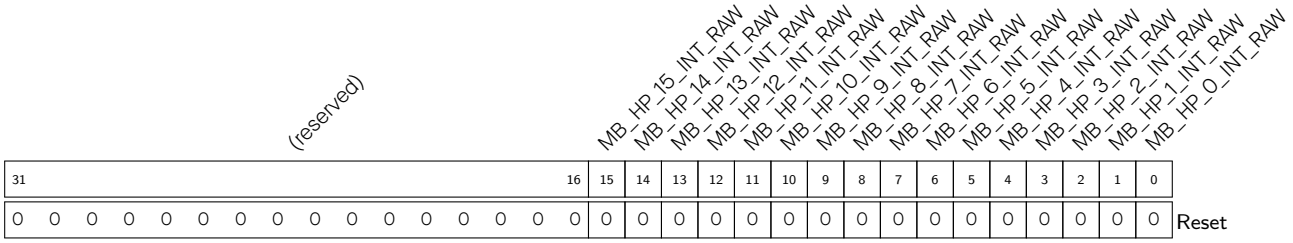
MB_LP_12_INT_CLR Write 1 to clear the MB_LP_12_INT interrupt. (WO)

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 18.22. 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)

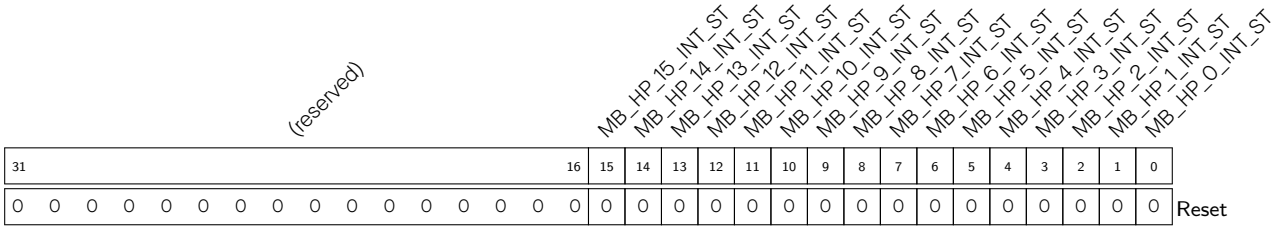
MB_HP_12_INT_RAW The raw interrupt bit of the MB_HP_12_INT interrupt. (RO)

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 18.23. 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)

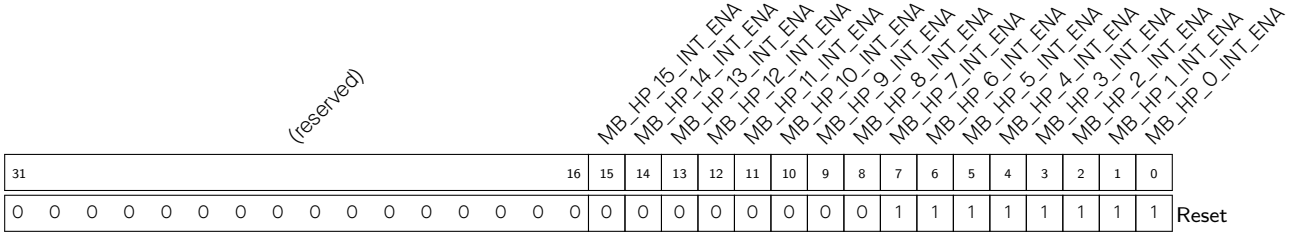
MB_HP_12_INT_ST The masked interrupt status bit of the MB_HP_12_INT interrupt. (RO)

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 18.24. 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)

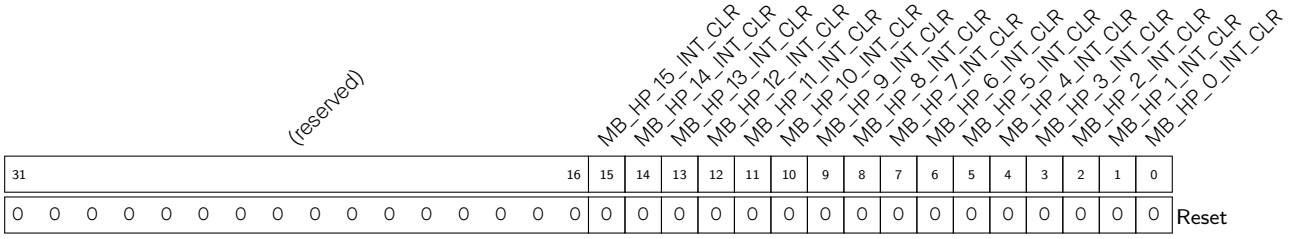
MB_HP_12_INT_ENA Write 1 to enable the MB_HP_12_INT interrupt. (R/W)

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 18.25. MB_HP_INT_CLR_REG (0x005C)



MB_HP_0_INT_CLR Write 1 to clear the MB_HP_0_INT interrupr. (WO)

MB_HP_1_INT_CLR Write 1 to clear the MB_HP_1_INT interrupr. (WO)

MB_HP_2_INT_CLR Write 1 to clear the MB_HP_2_INT interrupr. (WO)

MB_HP_3_INT_CLR Write 1 to clear the MB_HP_3_INT interrupr. (WO)

MB_HP_4_INT_CLR Write 1 to clear the MB_HP_4_INT interrupr. (WO)

MB_HP_5_INT_CLR Write 1 to clear the MB_HP_5_INT interrupr. (WO)

MB_HP_6_INT_CLR Write 1 to clear the MB_HP_6_INT interrupr. (WO)

MB_HP_7_INT_CLR Write 1 to clear the MB_HP_7_INT interrupr. (WO)

MB_HP_8_INT_CLR Write 1 to clear the MB_HP_8_INT interrupr. (WO)

MB_HP_9_INT_CLR Write 1 to clear the MB_HP_9_INT interrupr. (WO)

MB_HP_10_INT_CLR Write 1 to clear the MB_HP_10_INT interrupr. (WO)

MB_HP_11_INT_CLR Write 1 to clear the MB_HP_11_INT interrupr. (WO)

MB_HP_12_INT_CLR Write 1 to clear the MB_HP_12_INT interrupr. (WO)

MB_HP_13_INT_CLR Write 1 to clear the MB_HP_13_INT interrupr. (WO)

MB_HP_14_INT_CLR Write 1 to clear the MB_HP_14_INT interrupr. (WO)

MB_HP_15_INT_CLR Write 1 to clear the MB_HP_15_INT interrupr. (WO)

Chapter 19

Brown-out Detector

19.1 Introduction

The brown-out detector of ESP32-P4 monitors the voltage levels of pins VDDA and VBAT. If the voltage on these pins drops below the predefined threshold (defaulting to 2.7 V), the detector triggers signals to shut down certain power-consuming blocks (e.g., flash), ensuring that the digital module has sufficient time to save and transfer important data.

19.2 Feature List

- Two monitored sources
  - Monitors the voltage level of pin VDDA, which supplies power to the analog circuit.
  - Monitors the voltage level of pin VBAT, which connects the external battery to the chip.
- Two configurable monitoring modes
  - Mode 0: The brown-out detector triggers interrupts when the brown-out counter reaches the predefined threshold and selects the reset mode according to the configuration.
  - Mode 1: The brown-out detector triggers a system reset when the voltage falls below the threshold.
- Configurable voltage-monitoring thresholds and noise tolerance
- Configurable handling modes for under-voltage events

19.3 Architectural Overview

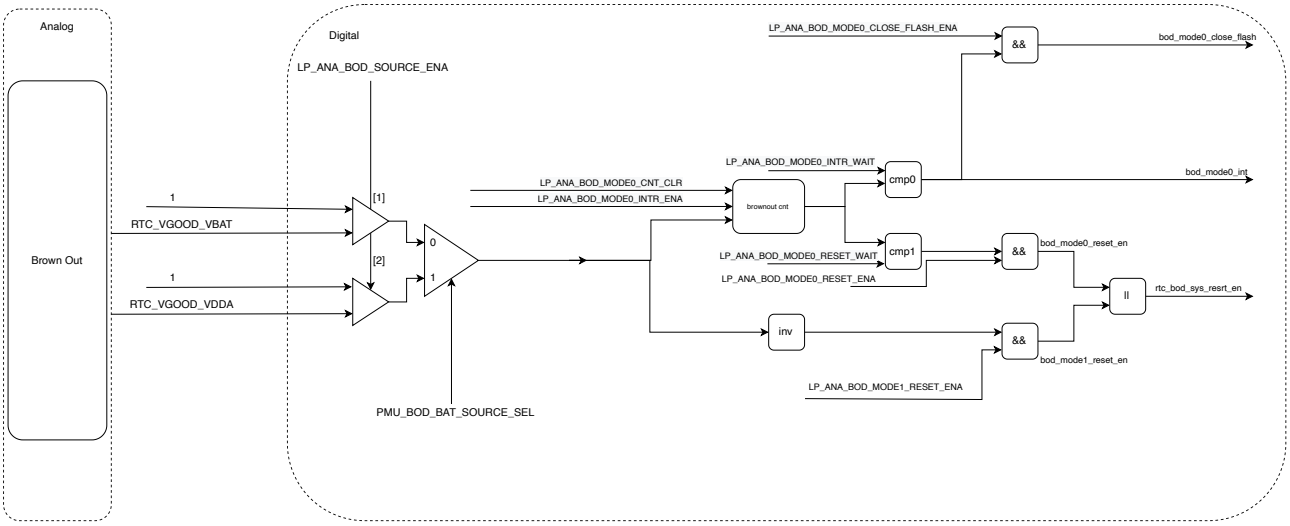


Figure 19.3-1. Brown-out Detector Overview

- Source selection for detection: The brown-out detector can select which power source to monitor by configuring LP_ANA_BOD_SOURCE_ENA. RTC_VGOOD_VBAT indicates the detection result of VBAT, and RTC_VGOOD_VDDA indicates the detection result of VDDA.
- Brown-out detector (Brown Out): An analog circuit designed to monitor the voltage of power pins and generate brown-out signals. It compares the voltage with the predefined threshold. When the voltage falls below this threshold, the detector outputs brown-out signals.
- Brown-out counter (brownout cnt): A counter used for filtering voltage noise on the detected power pins. When the brown-out signal is received, the counter starts counting based on the clock source LP_DYN_FAST_CLK. If the brown-out signal ceases during the counting process, the counter resets to 0. This counter can be cleared by configuring LP_ANA_BOD_MODE0_CNT_CLR.
- Interrupt comparator (cmp0): The comparator generates interrupts by comparing the counter value with the threshold. When the counter value exceeds the threshold, the comparator generates the interrupt signal bod_mode0_int. If LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA is enabled, bod_mode0_close_flash will be triggered, causing flash to enter the SUSPEND state.
- Reset comparator (cmp1): The comparator triggers reset signals by comparing the counter value with the threshold. When the counter value exceeds the threshold, the comparator triggers reset signals.

19.4 Functional Description

- Monitored sources
  - VDDA: Power pin for the analog circuit, supporting both Mode 0 and Mode 1.
  - VBAT: Power pin connected to the battery, supporting both Mode 0 and Mode 1. In addition, it supports two configurable voltage thresholds:
    * VBAT_BOD: Used for brown-out detection of VBAT

    * VBAT_CHARGER: Indicates whether the battery connected to VBAT needs charging.

  These thresholds trigger different interrupts.

- Monitoring modes
  There are two monitoring modes based on the way the brown-out detector handles the brown-out signal:
  - Mode 0: When Mode 0 is enabled, the brown-out counter starts counting after detecting a brown-out signal. Interrupt and reset signals are generated when the counter reaches the thresholds defined by LP_ANA_BOD_MODE0_RESET_WAIT and LP_ANA_BOD_MODE0_INTR_WAIT, respectively. This helps filter out noise on the monitored pins. Configure LP_ANA_BOD_MODE0_RESET_SEL to select the reset options:
    * 0: Reset the chip.
    * 1: Reset the system.
  - Mode 1: Mode 1 is the default mode, which resets the system immediately upon detecting under-voltage.
- Voltage-monitoring thresholds
  The voltage-monitoring threshold is configured in the analog circuit by the analog I2C master. The threshold configuration registers are as follows. BIAS is the slave of the analog I2C master. The address of BIAS is 0x6a. REGXX indicates the register address of BIAS, and [x:x] indicates the bits of this register.
  - BIAS_OR_DREFL_VBAT (REG08[7:5]): threshold for releasing the under-voltage state of VBAT
  - BIAS_OR_DREFH_VBAT (REG08[4:2]): under-voltage threshold of VBAT
  - BIAS_OR_DREFL_VDDA (REG09[7:5]): threshold for releasing the under-voltage state of VDDA
  - BIAS_OR_DREFH_VDDA (REG09[4:2]): under-voltage threshold of VDDA
  - BIAS_OR_DREFL_VBAT_CHARGER (REG10[7:5]): threshold for releasing the under-voltage state of battery connected to VBAT
  - BIAS_OR_DREFH_VBAT_CHARGER (REG10[4:2]): under-voltage threshold of battery connected to VBAT

  The relationship between the threshold configuration and voltage is:
  - 0: 2.52 V
  - 1: 2.57 V
  - 2: 2.63 V
  - 3: 2.68 V
  - 4: 2.74 V
  - 5: 2.78 V
  - 6: 2.83 V
  - 7: 2.89 V
- Voltage-monitoring filter
  A dedicated glitch filter is used to process the detection results of VBAT, as shown in the figure below.

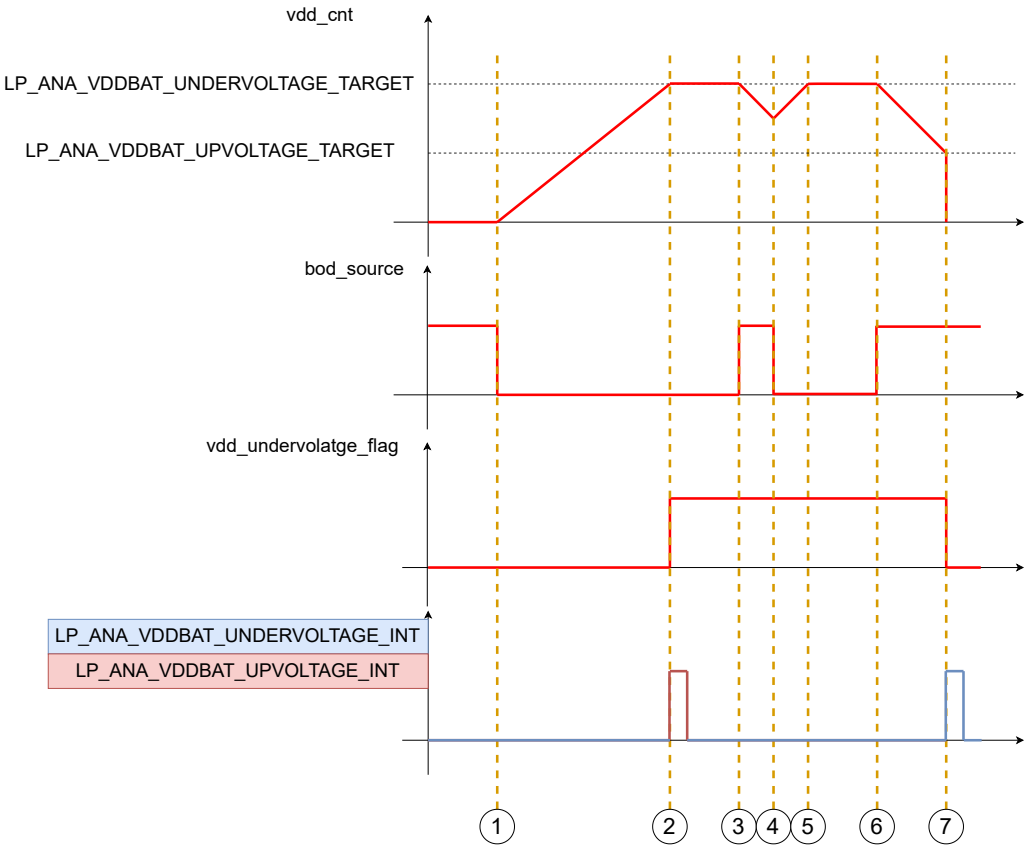


Figure 19.4-1. Structure of Brown-out Detector

- vdd_cnt: the value of the brown-out counter.
- bod_source: the monitored source, which is VBAT in the figure.
- vdd_undervoltage_flag: the under-voltage flag.
- Stage 0 ~ 1: When bod_source is 1, it indicates the voltage is normal, and the brown-out counter (vdd_cnt) remains 0.
- Stage 1 ~ 2: When bod_source is 0, it indicates under-voltage happens, and the brown-out counter will start to count. When the counter value reaches the corresponding threshold, vdd_undervoltage_flag will be set to 1, and the interrupt is generated.
- Stage 2 ~ 3: When vdd_underboltage_flag is 1, the brown-out counter will stop counting, and stabilize at LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET.
- Stage 3 ~ 4 ~ 5: When bod_source changes back to 0 for a short period, the value in the brown-out counter will decrease. However, as the value does not reach to LP_ANA_VDDBAT_UPVOLTAGE_TARGET, vdd_underboltage_flag will not be toggled.
- Stage 6 ~ 7: After bod_source remains 0 for a certain period, i.e., the counter value decreases to LP_ANA_VDDBAT_UPVOLTAGE_TARGET, vdd_underboltage_flag is reset to 0. Then the counter is cleared to 0.

19.5 Interrupts

ESP32-P4's brown-out detector can generate the following interrupt and send it to Interrupt Matrix.

- LP_ANAPERI_INT

The interrupt is generated by the internal interrupt sources of the brown-out detector.

- LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT: Triggered when the voltage of VBAT increases to the predefined charging threshold.
- LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT: Triggered when the voltage of VBAT decreases to the predefined charging threshold.
- LP_ANA_VDDBAT_UPVOLTAGE_INT: Triggered when the voltage of VBAT increases to the predefined brown-out threshold.
- LP_ANA_VDDBAT_UNDERVOLTAGE_INT: Triggered when the voltage of VBAT reduces to the predefined brown-out threshold.
- LP_ANA_BOD_MODE0_INT: Triggered when the value of the brown-out counter reaches the predefined threshold in Mode 0.

Each interrupt source can be configured by a common set of registers that are described in Chapter *Interrupt Configuration Registers*. The specific registers can be found in Section 19.7 *Register Summary*.

19.6 Programming Procedures

- Mode 0
  - Configure LP_ANA_BOD_MODE0_RESET_WAIT to set the counter threshold for triggering the reset signal.
  - Configure LP_ANA_BOD_MODE0_INTR_WAIT to set the counter threshold for triggering interrupts.
  - Set LP_ANA_BOD_MODE0_RESET_SEL to select the reset way in Mode 0.
  - Configure LP_ANA_BOD_SOURCE_ENA to enable the power pin to be detected.
  - Set LP_ANA_BOD_MODE0_RESET_ENA to enable the brown-out reset funtionality.
  - Set LP_ANA_BOD_MODE0_INTR_ENA to enable the noise-filtering counter and interrupts.
- Mode 1
  - Configure LP_ANA_BOD_SOURCE_ENA to enable the power pin to be detected.

19.7 Register Summary

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

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

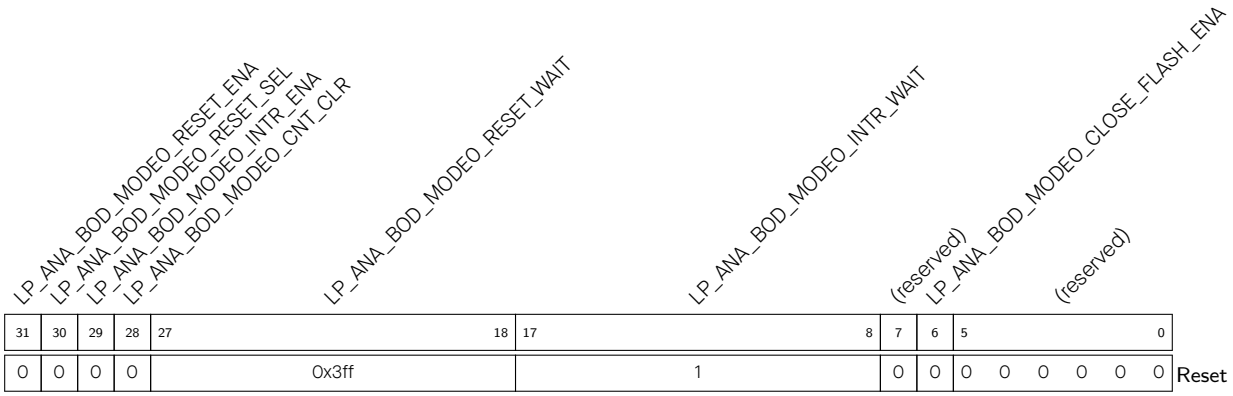
Name	Description	Address	Access
LP_ANA_BOD_MODE0_CNTL_REG	Control register of Mode 0	0x0000	R/W
LP_ANA_VDD_SOURCE_CNTL_REG	Control register of monitoring sources	0x0008	varies
LP_ANA_VDDBAT_BOD_CNTL_REG	Control register of brown-out detecting for VBAT	0x000C	varies
LP_ANA_VDDBAT_CHARGE_CNTL_REG	Pre-charge control register for under-voltage of VBAT	0x0010	varies
LP_ANA_INT_RAW_REG	Raw interrupt status	0x0020	R/WTC/SS
LP_ANA_INT_ST_REG	Interrupt state register	0x0024	RO
LP_ANA_INT_ENA_REG	Interrupt enable register	0x0028	R/W
LP_ANA_INT_CLR_REG	Interrupt clear register	0x002C	WT

19.8 Registers

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

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

Register 19.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)



LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA Configure to close the flash functionality when under-voltage happens in Mode 0.

0: Disable

1: Enable

(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT Configure the counter threshold that triggers interrupts in Mode 0. (R/W)

LP_ANA_BOD_MODE0_RESET_WAIT Configure the counter threshold that triggers reset in Mode 0. (R/W)

LP_ANA_BOD_MODE0_CNT_CLR Clear counter value in Mode 0. (R/W)

LP_ANA_BOD_MODE0_INTR_ENA Configure to enable the counter and interrupts in Mode 0.

0: Disable

1: Enable

(R/W)

LP_ANA_BOD_MODE0_RESET_SEL Configure the reset way when under-voltage happens in Mode 0.

0: Reset the chip

1: Reset the system

(R/W)

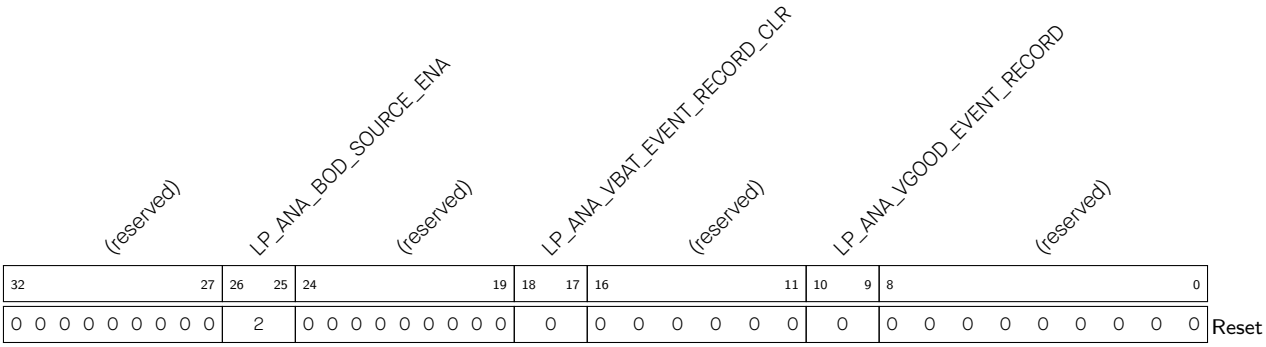
LP_ANA_BOD_MODE0_RESET_ENA Configure to enable reset in Mode 0.

0: Disable

1: Enable

(R/W)

Register 19.2. LP_ANA_VDD_SOURCE_CNTL_REG (0x0008)



LP_ANA_VGOOD_EVENT_RECORD Flags for indicating pins whose voltage is lower than the threshold.

0: Under-voltage does not happen on either VBAT or VDDA;

1: Under-voltage happens on VBAT;

2: Under-voltage happens on VDDA;

3: Under-voltage happens on both VBAT and VDDA.

(RO)

LP_ANA_VBAT_EVENT_RECORD_CLR Clear the flags.

0: Not clear

1: Clear the brown-out flag of VBAT

2: Clear the brown-out flag of VDDA

3: No effect

(WT)

LP_ANA_BOD_SOURCE_ENA Configure to detect monitoring sources.

0: Not detecting any pins;

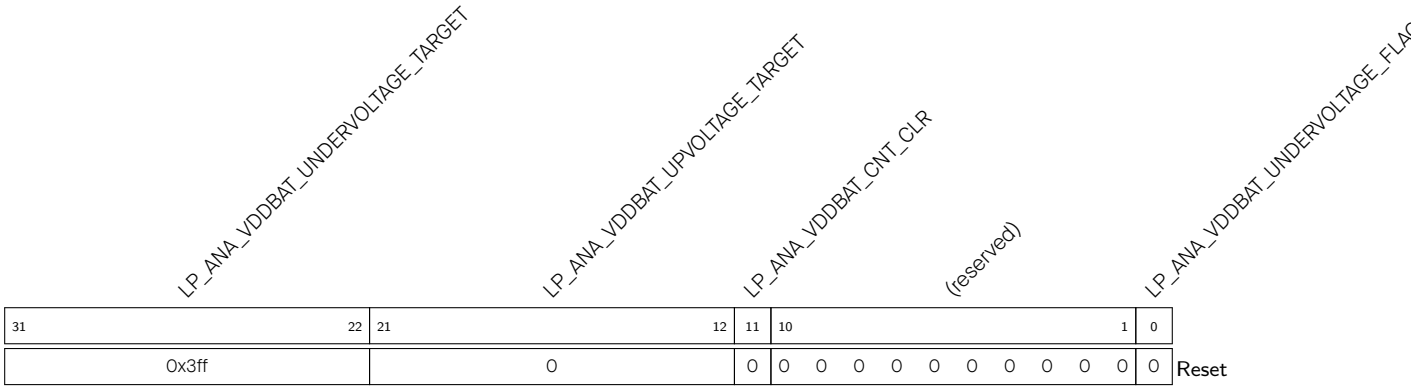
1: Only detect VBAT;

2: Only detect VDDA;

3: Detect both VBAT and VDDA.

(R/W)

Register 19.3. LP_ANA_VDDBAT_BOD_CNTL_REG (0x000C)



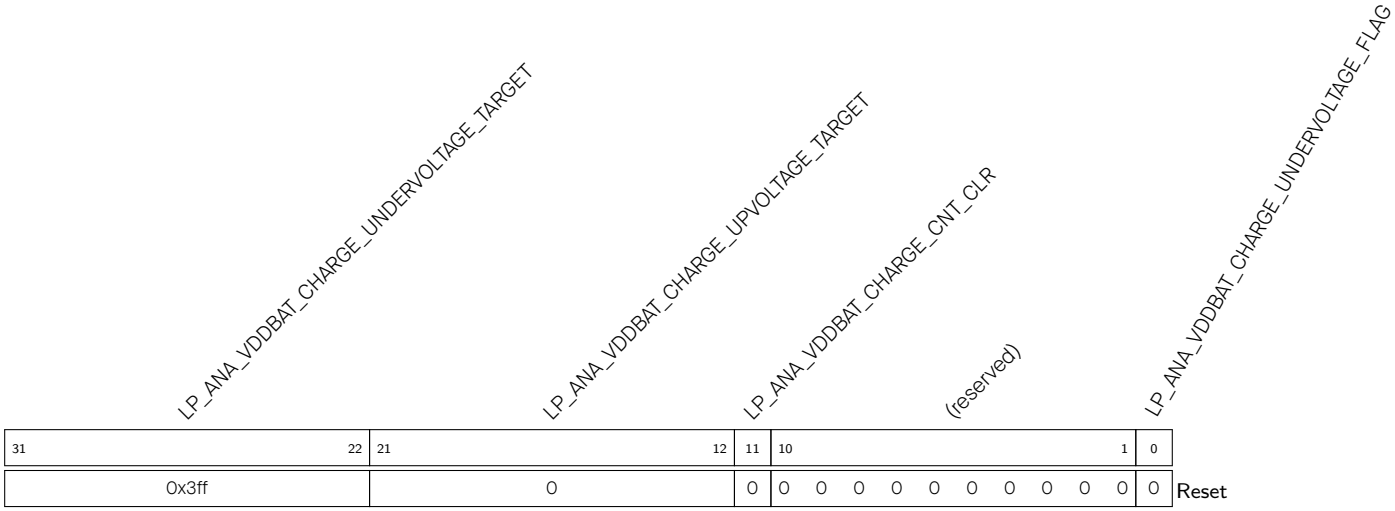
LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG Flag indicating that under-voltage occurs on VBAT. (RO)

LP_ANA_VDDBAT_CNT_CLR Clear the brown-out counter of VBAT. (WT)

LP_ANA_VDDBAT_UPVOLTAGE_TARGET Configure the voltage-recovery threshold for the brown-out counter of VBAT. When the counter decreases to this threshold, LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG is cleared to 0. (R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET Configure the brown-out threshold for the brown-out counter of VBAT. When the counter increases to this threshold, LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG is set to 1. (R/W)

Register 19.4. LP_ANA_VDDBAT_CHARGE_CNTL_REG (0x0010)



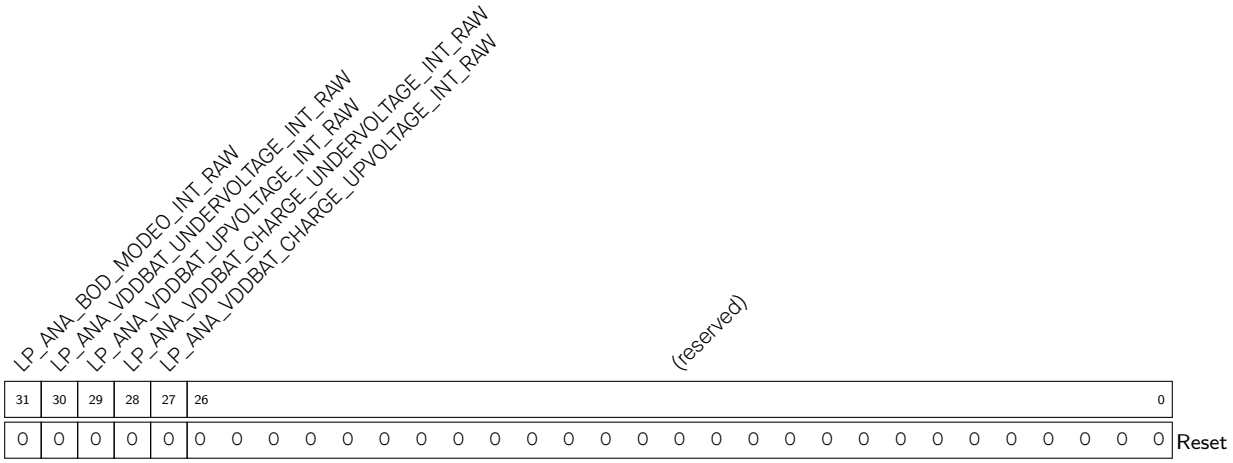
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG Flag indicating that VBAT is charging. (RO)

LP_ANA_VDDBAT_CHARGE_CNT_CLR Clear the brown-out counter of VBAT. (WT)

LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_TARGET Configure the voltage-recovery threshold for the brown-out counter of VBAT. When the counter decreases to this threshold, LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG is cleared to 0. (R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_TARGET Configure the brown-out threshold for the brown-out counter of VBAT. When the counter increases to this threshold, LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG is set to 1. (R/W)

Register 19.5. LP_ANA_INT_RAW_REG (0x0020)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VBAT increases to the charging threshold. (R/WTC/SS)

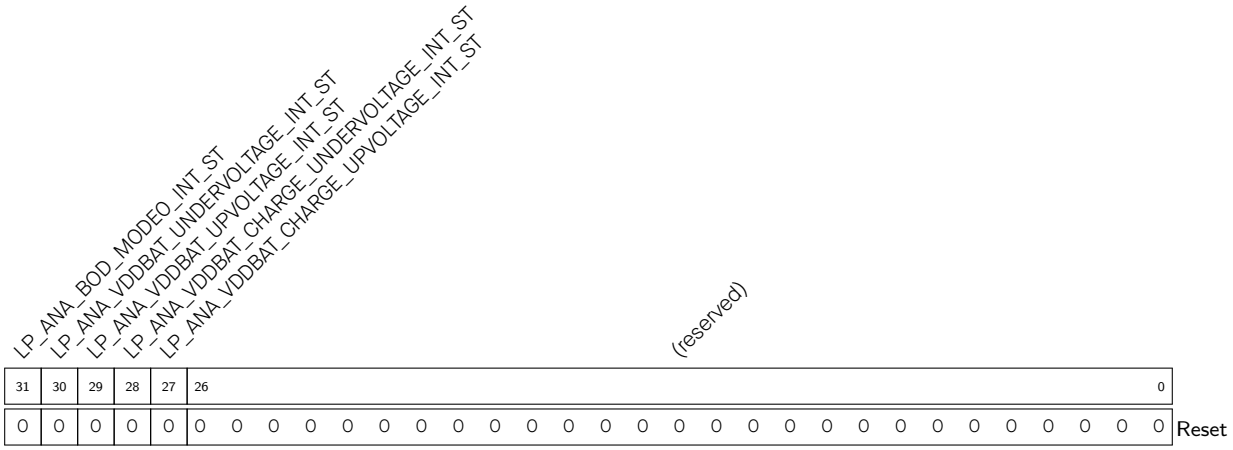
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VBAT decreases to the charging threshold. (R/WTC/SS)

LP_ANA_VDDBAT_UPVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VBAT increases to the brown-out threshold. (R/WTC/SS)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_RAW The interrupt is triggered when the voltage of VBAT decreases to the brown-out threshold. (R/WTC/SS)

LP_ANA_BOD_MODE0_INT_RAW Brown-out interrupt in Mode 0. (R/WTC/SS)

Register 19.6. LP_ANA_INT_ST_REG (0x0024)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VBAT increases to the charging threshold. (RO)

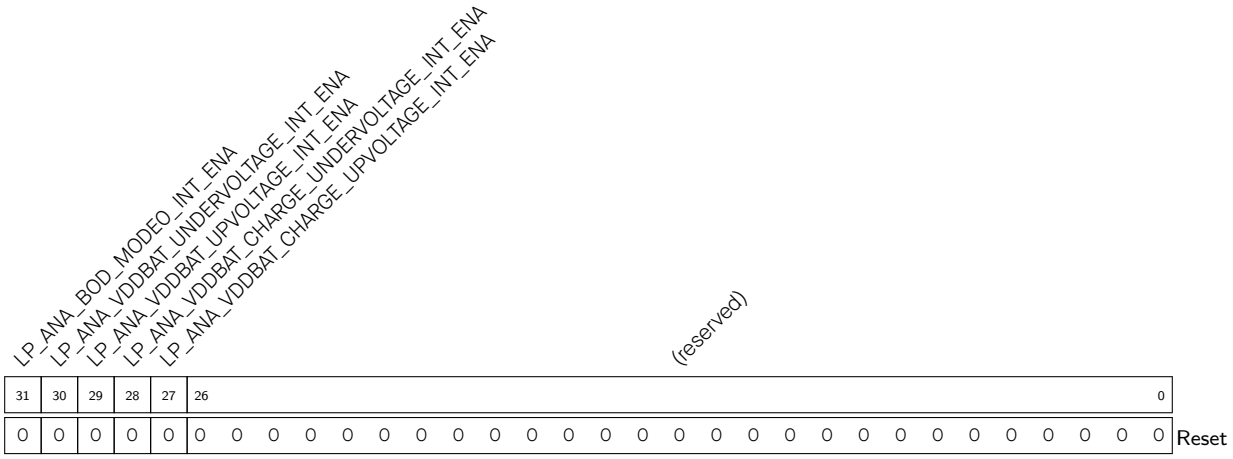
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VBAT decreases to the charging threshold. (RO)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VBAT increases to the brown-out threshold. (RO)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ST The interrupt is triggered and sent to CPU when the voltage of VBAT decreases to the brown-out threshold. (RO)

LP_ANA_BOD_MODE0_INT_ST The brown-out interrupt in Mode 0 is triggered and sent to CPU. (RO)

Register 19.7. LP_ANA_INT_ENA_REG (0x0028)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ENA Configure whether to enable the interrupt which is triggered when the voltage of VBAT increases to the charging threshold.

0: No

1: Yes

(R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VBAT reduces to the charging threshold.

0: Yes

1: No

(R/W)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VBAT increases to the brown-out threshold.

0: Yes

1: No

(R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ENA Configures whether to enable the interrupt which is triggered when the voltage of VBAT reduces to the brown-out threshold.

0: No

1: Yes

(R/W)

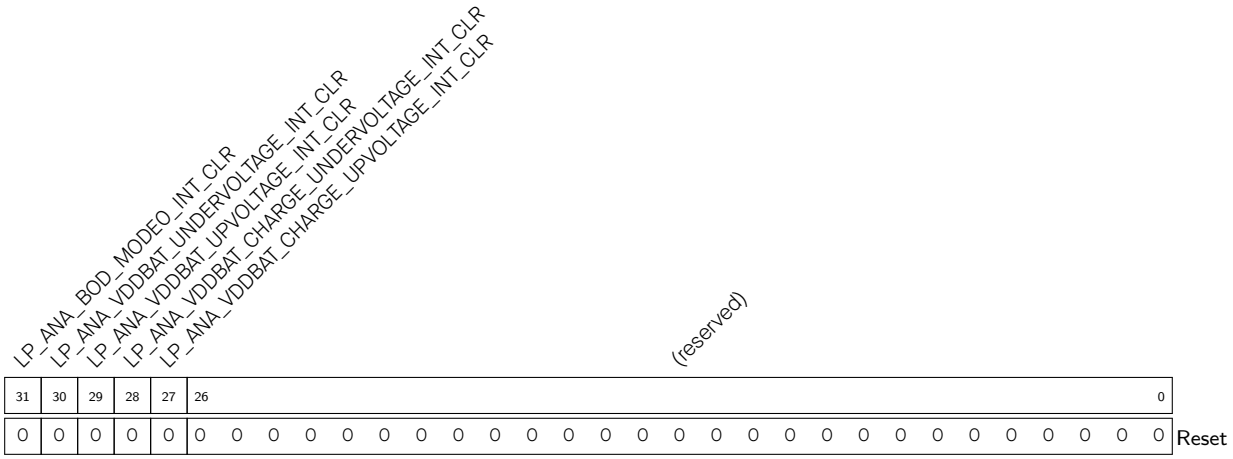
LP_ANA_BOD_MODE0_INT_ENA Configures whether to enable the brown-out interrupt in Mode 0.

0: No

1: Yes

(R/W)

Register 19.8. LP_ANA_INT_CLR_REG (0x002C)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VBAT voltage increasing to the charging threshold. (WT)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VBAT voltage reducing to the charging threshold. (WT)

LP_ANA_VDDBAT_UPVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VBAT voltage increasing to the brown-out threshold. (WT)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_CLR Configures whether to clear the interrupt triggered by the VBAT voltage reducing to the brown-out threshold. (WT)

LP_ANA_BOD_MODE0_INT_ST Configures whether to clear the brown-out interrupt in Mode 0. (WT)

Part V

Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like AES and ECC. It also covers digital signatures, random number generation, key management, and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

Chapter 20

AES Accelerator (AES)

20.1 Introduction

ESP32-P4 integrates an Advanced Encryption Standard (AES) accelerator, which is a hardware device that speeds up computation using AES algorithm significantly, compared to AES algorithms implemented solely in software. The AES accelerator integrated in ESP32-P4 has two working modes, which are Typical AES and DMA-AES.

20.2 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

20.3 Clock and Reset

The AES accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_AES_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_RST_EN_AES bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the HP_SYS_CLKRST_RST_EN_DS bit and the HP_SYS_CLKRST_RST_EN_KM bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register.

20.4 AES Working Modes

The AES accelerator integrated in ESP32-P4 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197.

  In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197;
  - Supports block cipher modes ECB, CBC, OFB, CTR, CFB8, and CFB128 under NIST SP 800-38A.
  - Supports GCM under NIST SP 800-38D.

  In this working mode, the plaintext and ciphertext are written and read via DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 20.4-1 below.

Table 20.4-1. 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 20.4-2 below.

Table 20.4-2. 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 20.5 and Section 20.6 below.

Notice:

ESP32-P4's Digital Signature Algorithm (DSA) module will call the AES accelerator. Therefore, users cannot access the AES accelerator when Digital Signature Algorithm (DSA) module is working.

20.5 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 20.5-1 below.

Table 20.5-1. 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.

20.5.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption or decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.
- For AES-256 encryption or decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_*m*_REG and AES_TEXT_OUT_*m*_REG, which are two sets of four 32-bit registers.

- For AES-128 or AES-256 encryption, the AES_TEXT_IN_*m*_REG registers are initialized with plaintext. Then, the AES accelerator stores the ciphertext into AES_TEXT_OUT_*m*_REG after operation.
- For AES-128 or AES-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

20.5.2 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 20.5-2.

Table 20.5-2. Text Endianness Type for Typical AES

Plaintext/Ciphertext					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]

Table 20.5-2. Text Endianness Type for Typical AES

		Plaintext/Ciphertext			
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] The definition of "State (including c and r)" is described in Section 3.4 *The State* in NIST FIPS 197.

[2] Where *x* = IN or OUT.

Key Endianness

In Typical AES working mode, when filling keys into AES_KEY_*m*_REG registers, users should follow the key endianness type specified in Table 20.5-3 and Table 20.5-4.

Table 20.5-3. 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 20.5-4. 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.

20.5.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_TEXT_IN_*m*_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers (*m*: 0-3) are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, and AES_KEY_*n*_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG and AES_KEY_*n*_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

20.6 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 20.6-1 below.

Table 20.6-1. 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 20.6-2 below.

Table 20.6-2. 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.

20.6.1 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 20.6-3 below.

Table 20.6-3. 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**($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

20.6.2 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 20.6-4 below.

Table 20.6-4. 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

20.6.3 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.

20.6.4 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.

20.6.5 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 20.6-4, 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.

20.6.6 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.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 20.6-1.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 20.6.4.
   - 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_CLR_REG register, if any AES interrupt occurred during the computation.
7. Release the AES accelerator by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 4 is completed.

20.6.7 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.
   - Write 6 to the AES_BLOCK_MODE_REG register.
   - Initialize the AES_BLOCK_NUM_REG register. Details about this register are described in Section 20.6.4 *Block Number*.
   - Initialize the AES_AAD_BLOCK_NUM_REG register. Details about this register are described in Section 20.7.4 *AAD Block Number*.
   - Initialize the AES_REMAINDER_BIT_NUM_REG register. Details about this register is described in Section 20.7.5 *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 20.6-2 *DMA-AES Working Mode*. At this step, no interrupt occurs.
5. Obtain the H value from the AES_H_MEM memory.
6. Generate J_0 and write it to the AES_J0_MEM memory.
7. Continue operating by writing 1 to the AES_CONTINUE_OP_REG register.
8. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs. For details on the working status of AES Accelerator, please refer to Table 20.6-2 *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_CLR_REG register, if any AES interrupt occurred during the computation.
12. Exit DMA by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG becomes 0. Note that, you can exit DMA earlier, but only after Step 8 is completed.

20.7 GCM Algorithm

ESP32-P4's AES accelerator fully supports GCM Algorithm. In reality, the AAD, C and P that are longer than 2^{32}-1 bits are seldom used. Therefore, we specify that the length of AAD, C and P should be no longer than 2^{32}-1 here. Figure 20.7-1 below demonstrates how GCM encryption is implemented in the AES Accelerator of ESP32-P4.

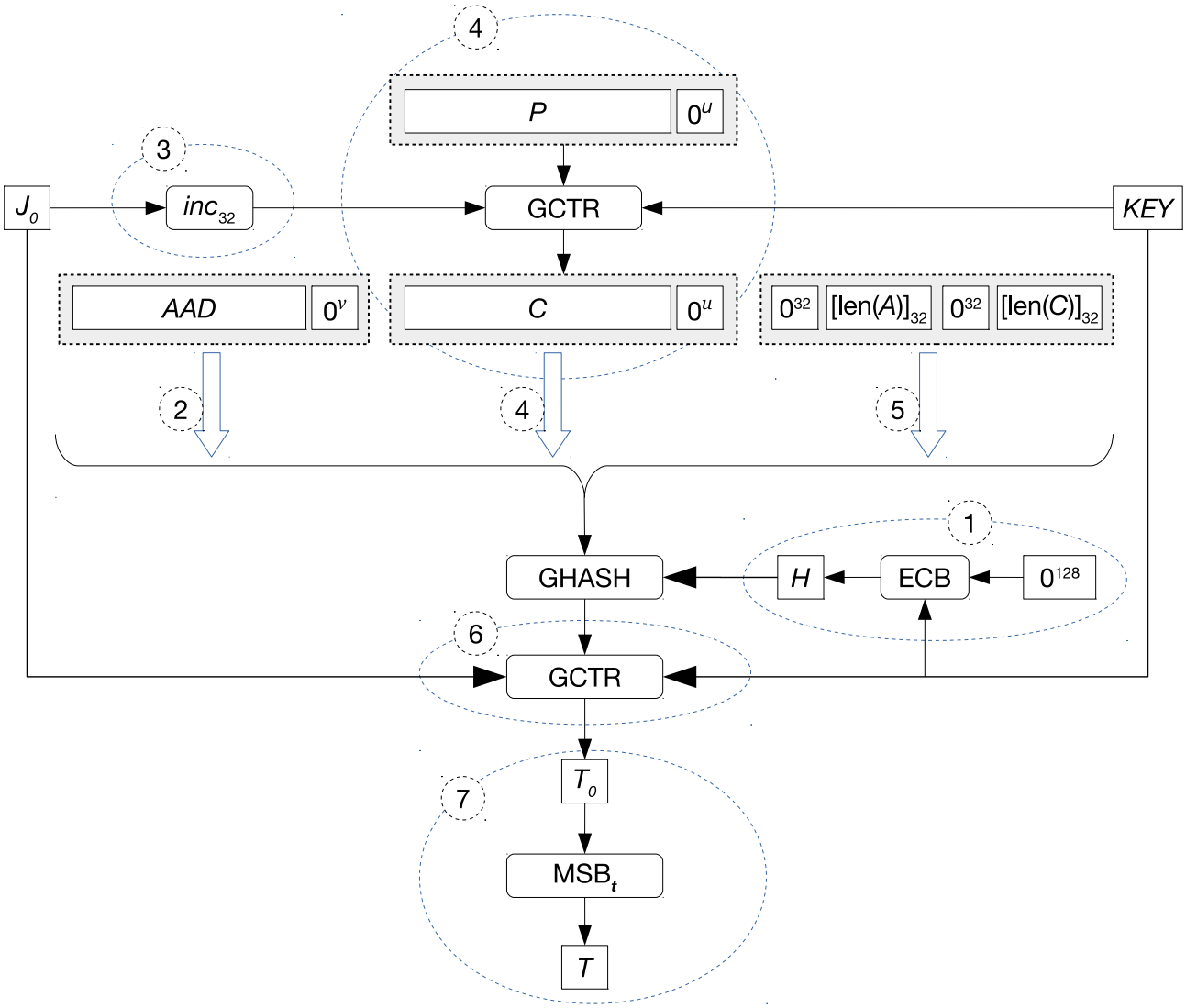


Figure 20.7-1. GCM Encryption Process

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 cipertext 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 20.7-1. 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 cipertext in GCM decryption. For details, please see NIST SP 800-38D.

20.7.1 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 20.7-1. 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 20.5-2.

20.7.2 J_0

J_0 is a 128-bit value computed by hardware, which is required during Step 3 and Step 6 in Figure 20.7-1. 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 20.5-2.

20.7.3 Authentication Tag

Authentication Tag (Tag for short) is one of the key results of GCM computation, which is demonstrated in Step 7 of Figure 20.7-1. 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 20.5-2.
- 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.

20.7.4 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.

20.7.5 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/cipertext. 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 cipertext, but does impact the value of T_0, therefore the Tag value too.

The GCM Algorithm can be viewed as the combination of GCTR operation and GHASH operation, among which, the GCTR performs the encryption and decryption, while the GHASH sloves the Tag.

Note that the AES_REMAINDER_BIT_NUM_REG register is only effective for GCM encryption. To be more specific:

- For GCM encryption, the Hardware firstly computes C, then passes it in the form of **TEXT-PADDING**(C) as the input of GHASH operation. In this case, hardware determines how many trailing "0" should be added based on the content of AES_REMAINDER_BIT_NUM_REG.
- For GCM decrpytion, the padding is completed with the **TEXT-PADDING**(C) function. In this case, the AES_REMAINDER_BIT_NUM_REG register is not effective.

20.8 Interrupts

ESP32-P4's AES can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- AES_INTR

There is one internal interrupt source from AES that can generate the above interrupt signal(s). The interrupt source from AES are listed with its trigger condition and the resulted interrupt signal(s) in Table 20.8-1.

Table 20.8-1. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

20.9 Memory Summary

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

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

Name	Description	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

20.10 Register Summary

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

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

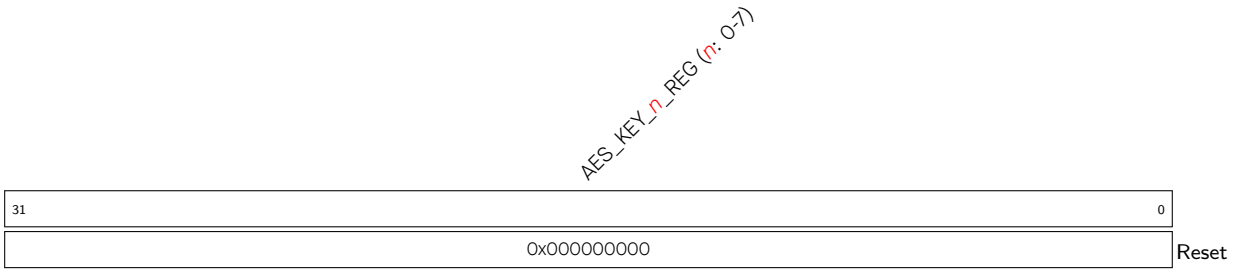
Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key data register 0	0x0000	R/W
AES_KEY_1_REG	AES key data register 1	0x0004	R/W
AES_KEY_2_REG	AES key data register 2	0x0008	R/W
AES_KEY_3_REG	AES key data register 3	0x000C	R/W
AES_KEY_4_REG	AES key data register 4	0x0010	R/W
AES_KEY_5_REG	AES key data register 5	0x0014	R/W
AES_KEY_6_REG	AES key data register 6	0x0018	R/W
AES_KEY_7_REG	AES key data register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source text data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source text data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source text data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source text data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result text data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result text data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result text data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result text data register 3	0x003C	RO
Control or Configuration Registers			
AES_MODE_REG	Defines key length and encryption/decryption	0x0040	R/W
AES_DMA_ENABLE_REG	Selects the working mode of the AES accelerator	0x0090	R/W
AES_BLOCK_MODE_REG	Defines the block cipher mode	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
AES_AAD_BLOCK_NUM_REG	AAD block number configuration register	0x00A0	R/W
AES_REMAINDER_BIT_NUM_REG	Remainder bit number of plaintext/ciphertext	0x00A4	R/W
AES_TRIGGER_REG	Operation start controlling register	0x0048	WT
AES_CONTINUE_OP_REG	Operation continue controlling register	0x00A8	WO
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
Status Register			
AES_STATE_REG	Operation status register	0x004C	RO
Interrupt Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WT
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

20.11 Registers

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

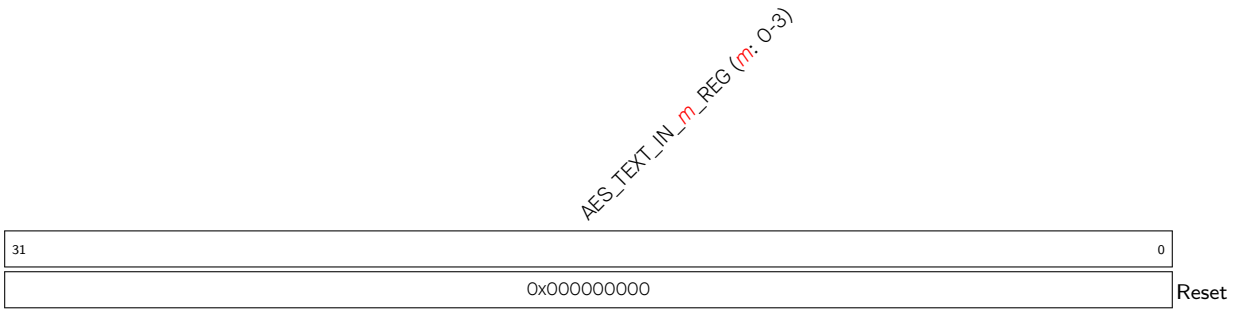
For how to program reserved fields, please refer to Section IX *Programming Reserved Register Field*.

Register 20.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4*n)



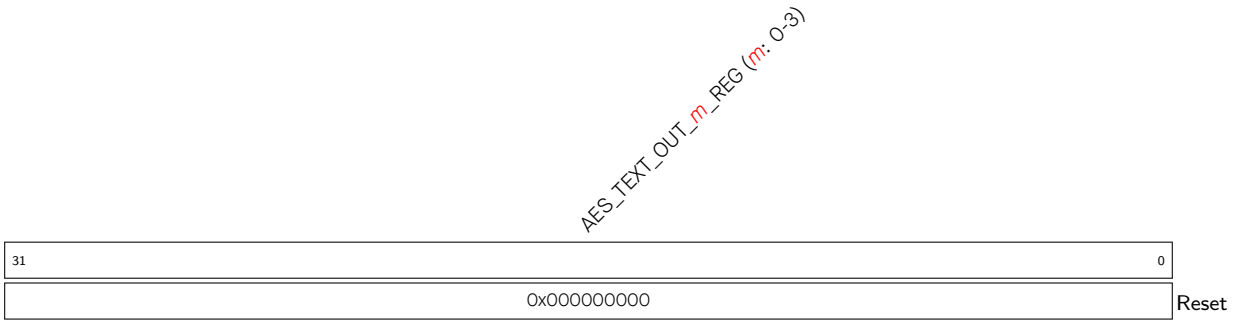
AES_KEY_*n*_REG (*n*: 0-7) Represents AES key data. (R/W)

Register 20.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4*m)



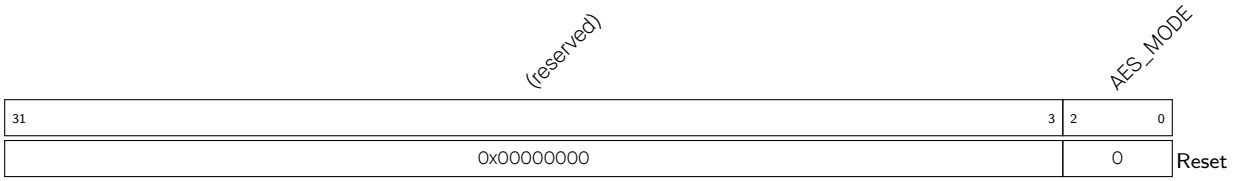
AES_TEXT_IN_*m*_REG (*m*: 0-3) Represents the source text data when the AES accelerator operates in the Typical AES working mode. (R/W)

Register 20.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4*m)



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 20.4. AES_MODE_REG (0x0040)



AES_MODE Configures the key length and encryption/decryption of the AES accelerator.

0: AES-128 encryption

1: Reserved

2: AES-256 encryption

3: Reserved

4: AES-128 decryption

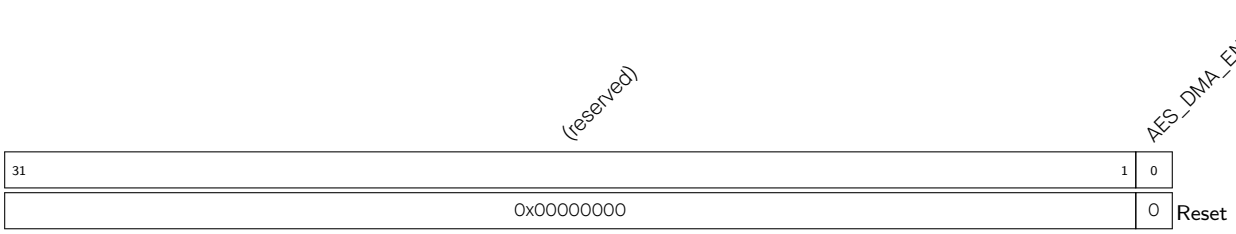
5: Reserved

6: AES-256 decryption

7: Reserved

(R/W)

Register 20.5. AES_DMA_ENABLE_REG (0x0090)



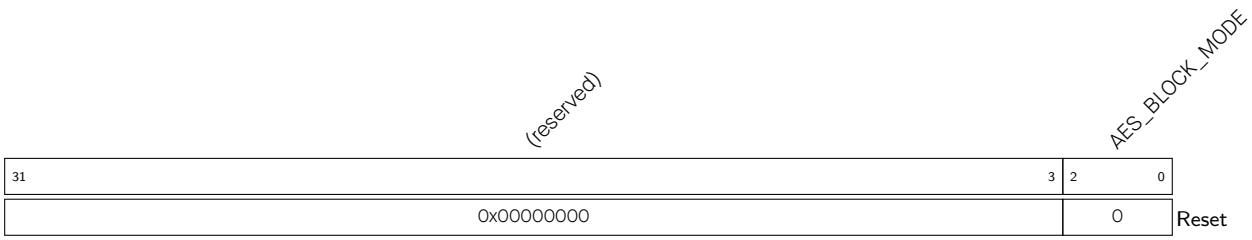
AES_DMA_ENABLE Configures the working mode of the AES accelerator.

0: Typical AES

1: DMA-AES

(R/W)

Register 20.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE Configures the block cipher mode of the AES accelerator operating under the DMA-AES working mode.

0: ECB (Electronic Code Block)

1: CBC (Cipher Block Chaining)

2: OFB (Output FeedBack)

3: CTR (Counter)

4: CFB8 (8-bit Cipher FeedBack)

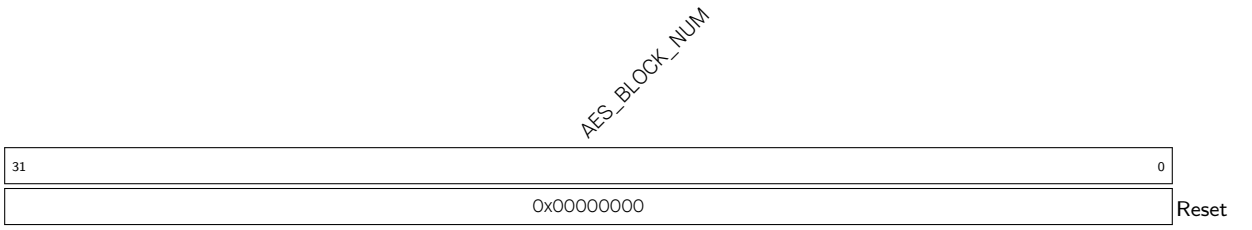
5: CFB128 (128-bit Cipher FeedBack)

6: GCM (Galois/Counter Mode)

7: Reserved

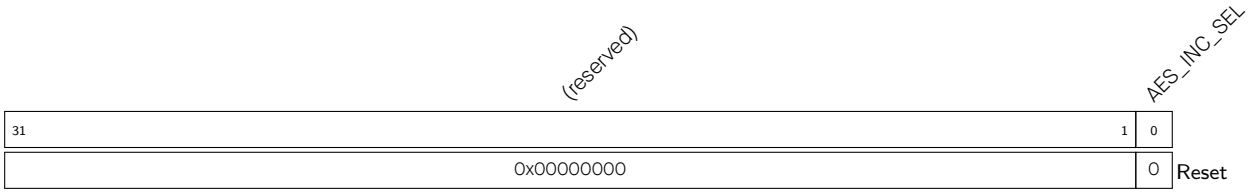
(R/W)

Register 20.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM Represents the Block Number of plaintext or ciphertext when the AES accelerator operates under the DMA-AES working mode. For details, see Section 20.6.4. (R/W)

Register 20.8. AES_INC_SEL_REG (0x009C)



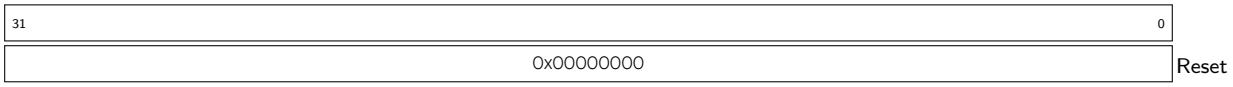
AES_INC_SEL Configures the Standard Incrementing Function for CTR block operation.

0: INC_{32}

1: INC_{128}

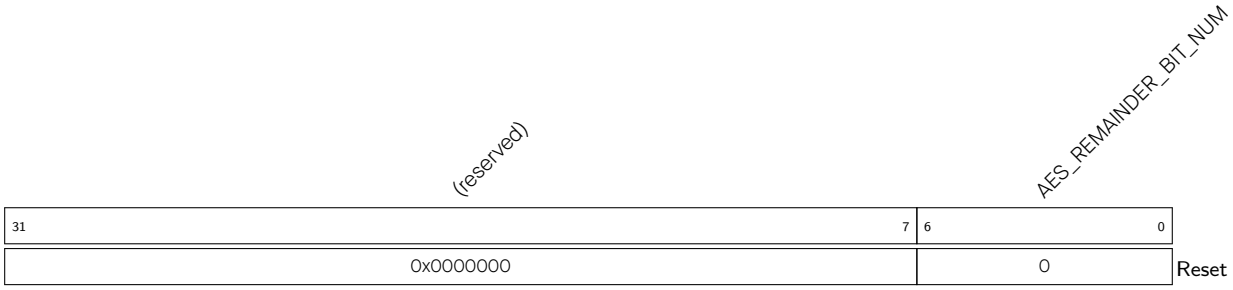
(R/W)

Register 20.9. AES_AAD_BLOCK_NUM_REG (0x00A0)



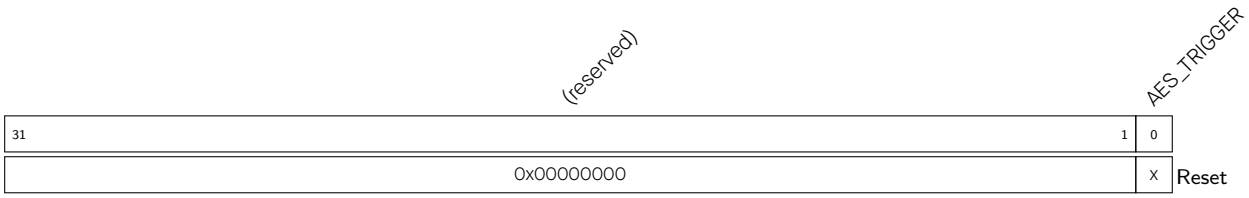
AES_AAD_BLOCK_NUM Stores the ADD block number for the GCM operation. For details, see Section 20.7.4. (R/W)

Register 20.10. AES_REMAINDER_BIT_NUM_REG (0x00A4)



AES_REMAINDER_BIT_NUM Stores the Remainder Bit Number for the GCM operation. For details, see Section 20.7.5. (R/W)

Register 20.11. AES_TRIGGER_REG (0x0048)



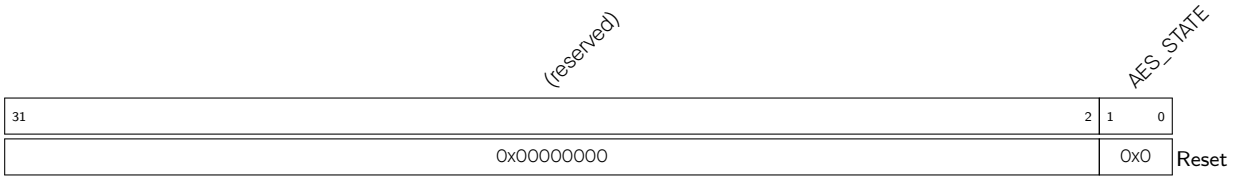
AES_TRIGGER Configures whether to start AES operation.

0: No effect

1: Start

(WT)

Register 20.12. AES_STATE_REG (0x004C)



AES_STATE Represents the working status of the AES accelerator.

In Typical AES working mode:

0: IDLE

1: WORK

2: No effect

3: No effect

In DMA-AES working mode:

0: IDLE

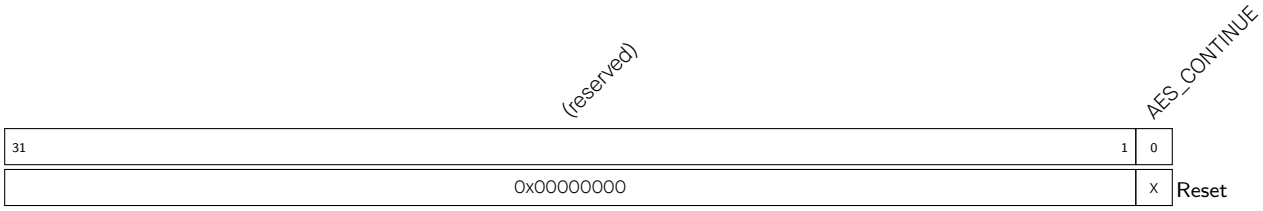
1: WORK

2: DONE

3: No effect

(RO)

Register 20.13. AES_CONTINUE_OP_REG (0x00A8)



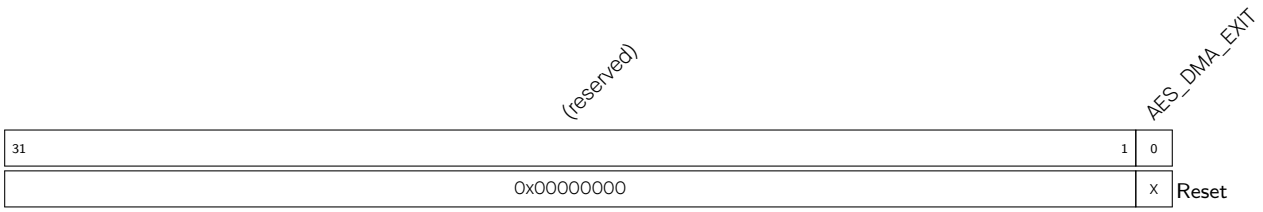
AES_CONTINUE_OP Configures whether to continue AES operation.

0: No effect

1: Continue

(WO)

Register 20.14. AES_DMA_EXIT_REG (0x00B8)



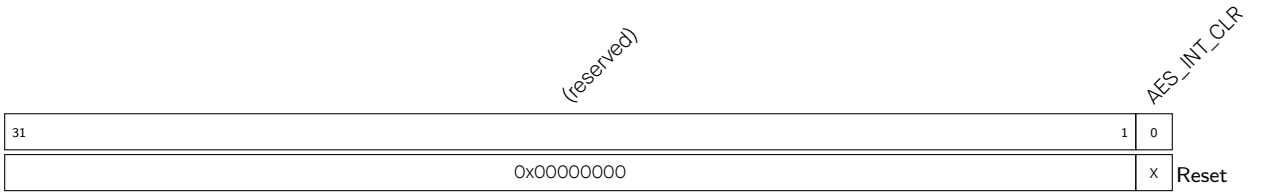
AES_DMA_EXIT Configures whether to exit AES operation.

0: No effect

1: Exit

Only valid for DMA-AES operation. (WO)

Register 20.15. AES_INT_CLR_REG (0x00AC)



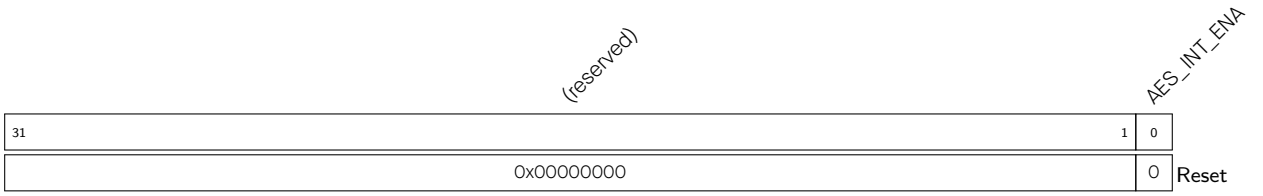
AES_INT_CLR Configures whether to clear AES interrupt.

0: No effect

1: Clear

(WT)

Register 20.16. AES_INT_ENA_REG (0x00B0)



AES_INT_ENA Configures whether to enable AES interrupt.

0: Disable

1: Enable

(R/W)

Chapter 21

ECC Accelerator (ECC)

21.1 Introduction

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves. ECC uses smaller keys compared to RSA cryptography while providing equivalent security.

ESP32-P4's ECC accelerator can complete various calculations based on different elliptic curves, thus accelerating the ECC algorithm and ECC-derived algorithms (such as ECDSA).

21.2 Features

ESP32-P4's ECC accelerator has the following features:

- 2 different elliptic curves, namely P-192 and P-256 defined in FIPS 186-3
- 11 working modes
- Interrupt upon completion of calculation

21.3 ECC Basics

To better illustrate the functionality of the ECC accelerator, the basic knowledge and terminology used in this chapter are introduced in this section.

21.3.1 Elliptic Curve and Points on the Curves

The ECC algorithm is based on elliptic curves over prime fields, which can be represented as:

$$y^2 = x^3 + ax + b \bmod p$$

where,

- p is a prime number,
- a and b are two non-negative integers smaller than p,
- and (x, y) is a point on the curve satisfying the representation.

21.3.2 Affine Coordinates and Jacobian Coordinates

An elliptic curve can be represented as below:

- In affine coordinates:

$$y^2 = x^3 + ax + b \bmod p$$

- In Jacobian coordinates:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

To convert affine coordinates (x, y) to/from Jacobian coordinates (X, Y, Z):

- From Jacobian to Affine coordinates:

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- From Affine to Jacobian coordinates:

$$X = x$$

$$Y = y$$

$$Z = 1$$

21.3.3 Memory Blocks

ECC's memory blocks store the input and output data of the ECC operation.

Table 21.3-1. ECC Accelerator Memory Blocks

Memory Block	Size (byte)	Starting Address *	Ending Address *	Access
ECC_Mem_k	32	0x100	0x11F	R/W
ECC_Mem_Px	32	0x120	0x13F	R/W
ECC_Mem_Py	32	0x140	0x15F	R/W
ECC_Mem_Qx	32	0x160	0x17F	R/W
ECC_Mem_Qy	32	0x180	0x19F	R/W
ECC_Mem_Qz	32	0x1A0	0x1BF	R/W

* Address offset related to the ECC accelerator base address is provided in Table 5.3-2 in Chapter 5 *System and Memory*.

21.3.4 Data and Data Block

ESP32-P4's ECC operates on 256-bit data. The data ($D[255:0]$) can be divided into eight 32-bit data blocks $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with smaller indexes correspond to lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

21.3.5 Writing Data

Write data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be specific, write data to an ECC memory block means writing $D[n][31:0](n = 0, 1, \cdots, 7)$ to the "starting address of this ECC memory block + $4 \times n$" successively:

- write $D[0]$ to "starting address"

- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

Note:
When the key size of 192 bits is used, append 0 before 192 bits of data to ensure 256-bit data is written.

21.3.6 Reading Data

Read data means reading data from the starting address of an ECC memory block and using this data as the output from the ECC algorithm. To be specific, read data from an ECC memory block means reading $D[n][31:0](n = 0, 1, \cdots, 7)$ from the "starting address of this ECC memory block + $4 \times n$" successively:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[7]$ from "starting address + 28"

Note:
When the key size of 192 bits is used, only read the low 192 bits (6 blocks) of data.

21.3.7 Standard Calculation and Jacobian Calculation

ESP32-P4's ECC performs Affine Point Calculation (including Affine Point Verification, Affine Point Add, and Affine Point Multiplication) using the affine coordinates and Jacobian Calculation (including Jacobian Point Verification, Jacobian Point Add, and Jacobian Point Multiplication) using the Jacobian coordinates.

21.4 Function Description

21.4.1 Key Size

ESP32-P4's ECC supports acceleration based on two key sizes, each corresponding to an elliptic curve. By configuring the ECC_KEY_LENGTH field, users can select the desired key size. For details, see Table 21.4-1 below.

Table 21.4-1. ECC Accelerator Key Size Selection

ECC_KEY_LENGTH	Elliptic Curves*
0	FIPS P-192
1	FIPS P-256

* See definition of FIPS P-192 and P-256 in FIPS 186-3.

21.4.2 Working Modes

ESP32-P4's ECC accelerator supports 11 working modes based on two elliptic curves described in the above section. By configuring the ECC_WORK_MODE field, users can select the desired working mode. For details, see Table 21.4-2.

Table 21.4-2. Working Modes of ECC Accelerator

ECC_WORK_MODE	Working Modes
0	Affine Point Multi
1	**Reserved**
2	Affine Point Verif
3	Affine Point Verif + Multi
4	Jacobian Point Multi
5	Point Add
6	Jacobian Point Verif
7	Affine Point Verif + Jacobian Point Multi
8	Mod Add
9	Mod Sub
10	Mod Multi
11	Mod Div

Note:
Note that the calculation of Jacobian Point Multi mode is about 10% faster than that of the Affine Point Multi mode.

Detailed descriptions about different working modes are provided in the following sections.

21.4.2.1 Affine Point Multiplication (Affine Point Multi)

Affine Point Multiplication can be represented as:

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

where,

- (Q_x, Q_y) is the affine expression of point Q.
- (J_x, J_y, J_z) is the Jacobian expression of point Q.
- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output: Q_x and Q_y are stored in ECC_Mem_Px and ECC_Mem_Py respectively.

21.4.2.2 Affine Point Verification (Affine Point Verif)

Affine Point Verification can be used to verify if a point (P_x, P_y) is on a selected elliptic curve.

- Input: P_x and P_y are stored in ECC_Mem_Px and ECC_Mem_Py respectively.
- Output: The verification result is stored in the ECC_VERIFICATION_RESULT bit.

21.4.2.3 Affine Point Verification + Affine Point Multiplication (Affine Point Verif + Multi)

In this mode, ECC first verifies if point (P_x, P_y) is on the selected elliptic curve. If so, the following multiplication is performed:

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

where,

- (Q_x, Q_y) is the affine expression of point Q.
- (J_x, J_y, J_z) is the Jacobian expression of point Q.
- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output:
  - The verification result is stored in the ECC_VERIFICATION_RESULT bit.
  - Q_x and Q_y are stored in ECC_Mem_Px and ECC_Mem_Py respectively.
  - J_x, J_y, and J_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz.

21.4.2.4 Jacobian Point Multiplication (Jacobian Point Multi)

Jacobian Point Multiplication can be represented as:

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output: Q_x, Q_y, and Q_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz respectively.

21.4.2.5 Point Addition (Point Add)

Point Addition can be represented as:

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

where,

- (R_x, R_y) is the affine expression of point R.
- (J_x, J_y, J_z) is the Jacobian expression of point R.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input:
  - P_x and P_y are stored in ECC_Mem_Px and ECC_Mem_Py.
  - Q_x, Q_y, and Q_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz.
- Output:
  - R_x and R_y are stored in ECC_Mem_Px and ECC_Mem_Py.
  - J_x, J_y and J_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz.

21.4.2.6 Jacobian Point Verification (Jacobian Point Verif)

Jacobian Point Verification can be used to verify if point (Q_x, Q_y, Q_z) is on a selected elliptic curve.

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- Input: Q_x, Q_y, and Q_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz respectively.
- Output: The verification result is stored in the ECC_VERIFICATION_RESULT bit.

21.4.2.7 Affine Point Verification + Jacobian Point Multiplication (Affine Point Verif + Jacobian Point Multi)

In this mode, ECC first verifies if point (P_x, P_y) is on the selected elliptic curve. If so, the following multiplication is performed:

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is the Jacobian expression of point Q.
- 1 in the point's Jacobian coordinates is automatically completed by hardware.
- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k.
- Output:
  - The verification result is stored in the ECC_VERIFICATION_RESULT bit.
  - Q_x, Q_y, and Q_z are stored in ECC_Mem_Qx, ECC_Mem_Qy, and ECC_Mem_Qz.

21.4.2.8 Mod Addition (Mod Add)

Mod Addition can be represented as:

$$R = A + B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_Mem_Px and ECC_Mem_Py.
  - The value of N is related to the register fields below:
    - ECC_KEY_LENGTH to select the related curve.
    - ECC_MOD_BASE to choose using mod base or order of the curve.
- Output: R is stored in ECC_Mem_Px.

21.4.2.9 Mod Subtraction (Mod Sub)

Mod Subtraction can be represented as:

$$R = A - B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_Mem_Px and ECC_Mem_Py.

  - The value of N is related to the register fields below:
    * ECC_KEY_LENGTH to select the related curve.
    * ECC_MOD_BASE to choose using mod base or order of the curve.
- Output: R is stored in ECC_Mem_Px.

21.4.2.10 Mod Multiplication (Mod Multi)

Mod Multiplication can be represented as:

$$R = A \cdot B \bmod N$$

where,

- Input:
  - A and B are stored in ECC_Mem_Px and ECC_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_KEY_LENGTH to select the related curve.
    * ECC_MOD_BASE to choose using mod base or order of the curve.
- Output: R is stored in ECC_Mem_Py.

21.4.2.11 Mod Division (Mod Div)

Mod Division can be represented as:

$$R = A \cdot B^{-1} \bmod N$$

where,

- Input:
  - A and B are stored in ECC_Mem_Px and ECC_Mem_Py.
  - The value of N is related to the register fields below:
    * ECC_KEY_LENGTH to select the related curve.
    * ECC_MOD_BASE to choose using mod base or order of the curve.
- Output: R is stored in ECC_Mem_Py.

21.5 Clock and Reset

ESP32-P4's ECC only has one clock module (CRYPTO_ECC_CLK) and one reset module (CRYPTO_ECC_RST).

ECC's clock and reset is handled by the the Power/Clock/Reset (PCR) module (see Chapter 8 *Reset and Clock* for more information). Users should enable the ECC clock by setting HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN and release the ECC reset by clearing HP_SYS_CLKRST_RST_EN_ECC before starting the ECC accelerator. Besides, due to resource reuse between cryptography accelerator modules, users also need to additionally clear the HP_SYS_CLKRST_RST_EN_ECDSA bit.

21.6 Interrupts

ESP32-P4's ECC accelerator can generate the ECC_INTR interrupt signal that will be sent to the Interrupt Matrix.

ECC_INTR has only one interrupt source to generate the ECC_INTR interrupt signal, i.e., ECC_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_CALC_DONE_INT interrupt source is configured by the following registers (refer to Section 21.8 *Register Summary* for more information):

- ECC_CALC_DONE_INT_RAW: Stores the raw interrupt status of ECC_CALC_DONE_INT.
- ECC_CALC_DONE_INT_ST: Indicates the status of the ECC_CALC_DONE_INT interrupt. This bit is generated by enabling or disabling the ECC_CALC_DONE_INT_RAW bit via ECC_CALC_DONE_INT_ENA.
- ECC_CALC_DONE_INT_ENA: Enables or disables the ECC_CALC_DONE_INT interrupt.
- ECC_CALC_DONE_INT_CLR: Set this bit to clear the ECC_CALC_DONE_INT interrupt status. By setting this bit to 1, the ECC_CALC_DONE_INT_RAW and ECC_CALC_DONE_INT_ST bits will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

21.7 Programming Procedures

The programming procedure for configuring ECC is described below:

1. Configure the ECC clock and reset. Refer to Section 21.5 for detailed information.
2. Select the key size and working mode as described in Section 21.4.
3. Enable the ECC_CALC_DONE_INT interrupt as described in Section 21.6.
4. Set the ECC_START field to start ECC calculation.
5. Wait for the ECC_CALC_DONE_INT interrupt, which indicates the completion of the ECC calculation.
6. Check the result as described in Section 21.4.

21.8 Register Summary

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

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

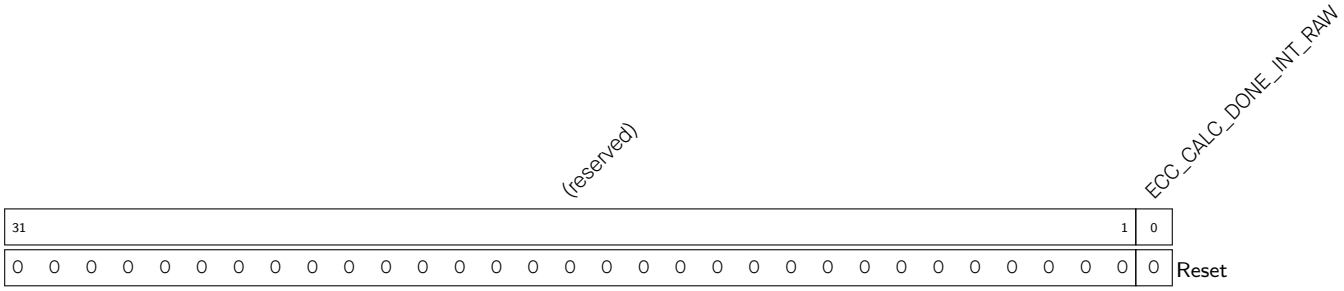
Name	Description	Address	Access
Interrupt Registers			
ECC_MULT_INT_RAW_REG	ECC raw interrupt status register	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	ECC masked interrupt status register	0x0010	RO
ECC_MULT_INT_ENA_REG	ECC interrupt enable register	0x0014	R/W
ECC_MULT_INT_CLR_REG	ECC interrupt clear register	0x0018	WT
Configuration Register			
ECC_MULT_CONF_REG	ECC configuration register	0x001C	varies
Version Register			
ECC_MULT_DATE_REG	Version control register	0x00FC	R/W

21.9 Registers

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

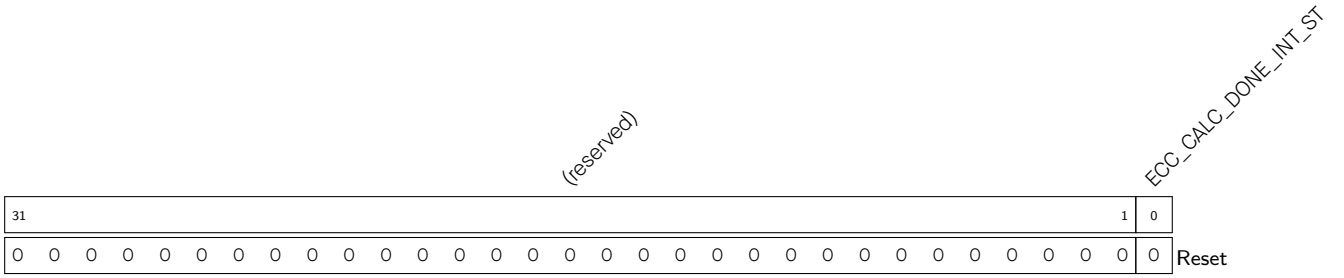
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 21.1. ECC_MULT_INT_RAW_REG (0x000C)



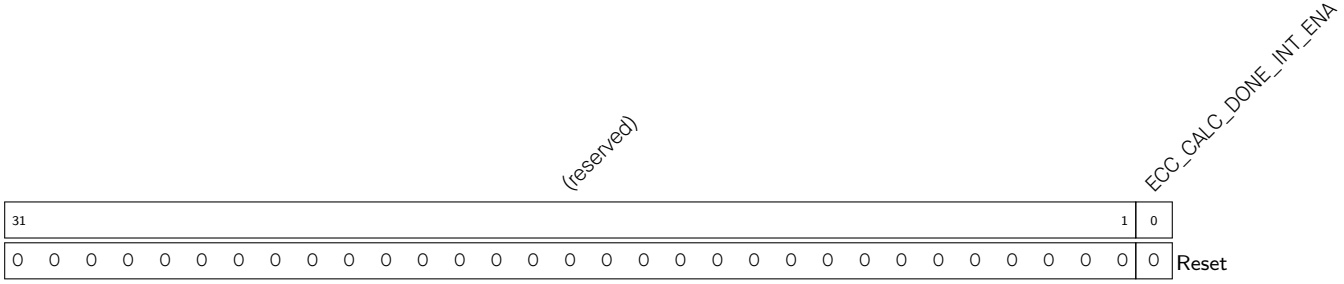
ECC_CALC_DONE_INT_RAW The raw interrupt status of the ECC_CALC_DONE_INT interrupt. (R/SS/WTC)

Register 21.2. ECC_MULT_INT_ST_REG (0x0010)



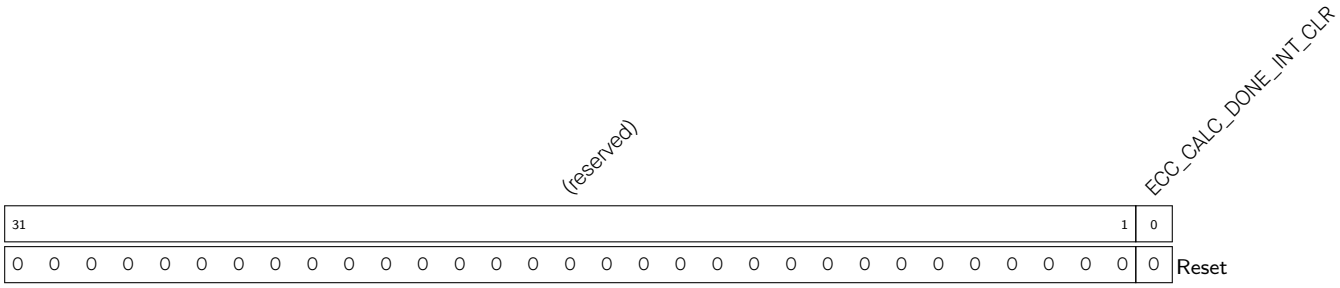
ECC_CALC_DONE_INT_ST The masked interrupt status of the ECC_CALC_DONE_INT interrupt. (RO)

Register 21.3. ECC_MULT_INT_ENA_REG (0x0014)



ECC_CALC_DONE_INT_ENA Write 1 to enable the ECC_CALC_DONE_INT interrupt. (R/W)

Register 21.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_CALC_DONE_INT_CLR Write 1 to clear the ECC_CALC_DONE_INT interrupt. (WT)

Register 21.5. ECC_MULT_CONF_REG (0x001C)

Field	ECC_MEM_CLOCK_GATE_FORCE_ON	ECC_CLK_EN	ECC_VERIFICATION_RESULT	(reserved)	ECC_WORK_MODE	ECC_MOD_BASE	ECC_KEY_LENGTH	ECC_RESET	ECC_START
Bits	31	30	29	28 ... 9	7 ... 4	3	2	1	0
Reset	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0

ECC_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_RESET Configures whether to reset the ECC accelerator.

0: No effect

1: Reset

(WT)

ECC_KEY_LENGTH Configures the key length of the ECC accelerator.

0: P-192

1: P-256

(R/W)

ECC_MOD_BASE Configures whether to choose using the mod base or order of curve in the mod operation. Only valid in working modes 8-11.

0: n (order of curve)

1: p (mod base of curve)

(R/W)

ECC_WORK_MODE Configures the working mode of the ECC accelerator.

0: Affine Point Multi mode

1: Reserved

2: Affine Point Verif mode

3: Affine Point Verif + Multi mode

4: Jacobian Point Multi mode

5: Point Add mode

6: Jacobian Point Verif mode

7: Affine Point Verif + Jacobian Point Multi mode

8: Mod Add mode

9. Mod Sub mode

10: Mod Multi mode

11: Mod Div mode

(R/W)

Continued on the next page...

Register 21.5. ECC_MULT_CONF_REG (0x001C)

Continued from the previous page...

ECC_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_CLK_EN Configures whether to force on register clock gate.

0: No effect

1: Force on

(R/W)

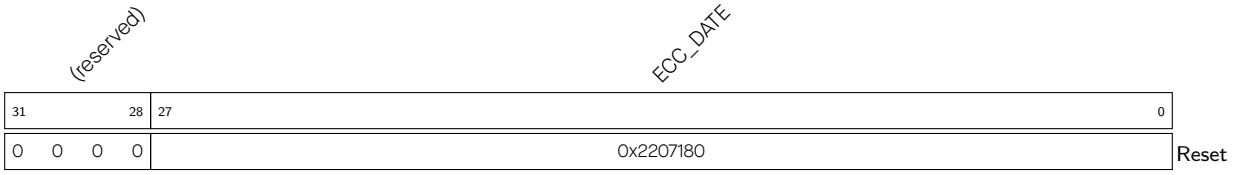
ECC_MEM_CLOCK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.

0: No effect

1: Force on

(R/W)

Register 21.6. ECC_MULT_DATE_REG (0x00FC)



ECC_DATE Version control register. (R/W)

Chapter 22

HMAC Accelerator (HMAC)

The Hash-based Message Authentication Code (HMAC) module computes Message Authentication Codes (MACs) using hash algorithm SHA-256 and keys as described in RFC 2104. The 256-bit HMAC key is stored in an eFuse key block and can be set as read-protected, i.e., the key is not accessible from outside the HMAC accelerator.

22.1 Main Features

- Standard HMAC-SHA-256 algorithm
- HMAC-SHA-256 calculation based on key in eFuse,
  - whose result cannot be accessed by software in downstream mode for high security
  - whose result can be accessed by software in upstream mode
- Generates required keys for the Digital Signature Algorithm (DSA) peripheral in downstream mode
- Re-enables soft-disabled JTAG in downstream mode

22.2 Functional Description

The HMAC module operates in two modes: upstream mode and downstream mode. In upstream mode, users provide the HMAC message and read back the calculation result. In downstream mode, the HMAC module provides input to two possible other internal hardware modules: On the one hand, an HMAC can be used to enable JTAG after JTAG has been temporarily disabled before. On the other hand, an HMAC can be used as decryption key for Digital Signature parameters stored in flash memory for the DSA peripheral. Furthermore, the calculations happen internally and automatically in downstream mode, so that confidentiality of any key and derived key material is ensured, given correct configuration.

Note:
After the reset signal being released, the HMAC module will check whether the DSA key exists in the eFuse. If the key exists, the HMAC module will enter downstream digital signature algorithm mode and finish the DSA key calculation automatically. 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.

22.2.1 Upstream Mode

To calculate the HMAC value in upstream mode, users should perform the following steps:

1. Initialize the HMAC module and enter upstream mode.

2. Write the correctly padded message to the HMAC, one block at a time.
3. Read back the result from HMAC.

For details of this process, please see Section 22.2.5.

Note:
Common use cases for the upstream mode are challenge-response protocols supporting HMAC-SHA-256. Assume the two entities in the challenge-response protocol are A and B respectively, and the data message they expect to exchange is M. The general authentication process of this protocol is as follows:

- A calculates a unique random number M.
- A sends M to B.
- B calculates the HMAC (through M and KEY) and sends the result to A.
- A calculates the HMAC (through M and KEY) internally.
- A compares the two results. If the results are the same, then the identity of B is authenticated.

22.2.2 Downstream Mode - JTAG Enable Feature

JTAG debugging can be disabled by eFuse in a way which allows later re-enabling using the HMAC module. (For more details, please see Chapter 6 *eFuse Controller*.) 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 22.2.5 for more details.
2. Write 1 to the HMAC_SOFT_JTAG_CTRL_REG register to enter JTAG re-enable mode.
3. Write the 256-bit HMAC value to register HMAC_WR_JTAG_REG. This value is obtained by performing a local HMAC calculation from the 32-byte 0x00 using SHA-256 and the key that has been written to the eFuse. It needs to be written 8 times and 32-bit each time in big-endian word order.
4. If the HMAC result calculated from the key in the eFuse matches the value that users wrote in step 3, then JTAG is re-enabled. Otherwise, JTAG remains disabled.
5. After writing 1 to HMAC_SET_INVALIDATE_JTAG_REG or resetting the chip, JTAG will be disabled. If users want to re-enable JTAG again, they need to repeat the above steps again.

22.2.3 Downstream Mode - Digital Signature Algorithm and Key Derivation Feature

The Digital Signature Algorithm (DSA) 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 DSA module, users need to obtain the parameter decryption key for the DSA module through HMAC calculation. For more information, please see Chapter 25 *Digital Signature Algorithm (DSA)*. 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.

22.2.4 HMAC eFuse Configuration

Each HMAC key burned into an eFuse block has a key purpose, specifying for which functionality the key can be used. The HMAC module will not accept a key with a non-matching purpose for any functionality. The HMAC module provides three different functionalities: re-enabling JTAG, DSA KDF in downstream mode, and pure HMAC calculation in upstream mode. For each functionality, there exists a corresponding key purpose, listed in Table 22.2-1. Additionally, another purpose specifies a key which may be used for re-enabling JTAG as well as for serving as DSA KDF.

Before enabling HMAC to do calculations, user should make sure the key to be used has been burned in eFuse by reading the registers EFUSE_KEY_PURPOSE_x (We have a total of 6 keys in eFuse, so the value of x is 0 ~ 5. Among which, EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 belong to the register EFUSE_RD_REPEAT_DATA1_REG, and EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 belong to the register EFUSE_RD_REPEAT_DATA2_REG) from 6 *eFuse Controller*. Take upstream mode as an example, if there is no EFUSE_KEY_PURPOSE_HMAC_UP in EFUSE_KEY_PURPOSE_0 ~ 5, it means there is no key in eFuse that can be used for the HMAC upstream mode. Users can burn a key to eFuse as follows:

1. Prepare a secret 256-bit HMAC key and burn the key to an empty eFuse block y. As there are 6 blocks for storing a key in eFuse and the numbers of those blocks range from 4 to 9, the value of y is 4 ~ 9. Hence, when talking about key0, it means eFuse block4. Then, program the purpose to EFUSE_KEY_PURPOSE_$(y-4)$. Take upstream mode as an example: after programming the key, the user should program EFUSE_KEY_PURPOSE_HMAC_UP (corresponding value is 8) to EFUSE_KEY_PURPOSE_$(y-4)$. Please see Chapter 6 *eFuse Controller* 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 DSA.

Table 22.2-1. HMAC Purposes and Configuration Value

Purpose	Mode	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
DSA KDF	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and DSA KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Select eFuse Key Blocks and HMAC Purposes

The eFuse controller provides six key blocks, i.e., KEY0 ~ 5. To select a particular KEYn for an HMAC calculation, write the key number n to register HMAC_SET_PARA_KEY_REG.

Write a correct purpose to register HMAC_SET_PARA_PURPOSE_REG (see Section 22.2.5). Note that the purpose of the key has also been programmed to eFuse memory. Only when the configured HMAC purpose matches the defined purpose of KEYn, the HMAC module will execute the configured calculation. Otherwise, it will return a matching error and stop the current calculation.

For example, suppose a user selects KEY3 for HMAC calculation, and the value programmed to EFUSE_KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 22.2-1, KEY3 can be used to re-enable JTAG. If the value written to register HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC module will start the process to re-enable JTAG.

22.2.5 HMAC Process (Detailed)

The process for users to call HMAC in ESP32-P4 is as follows:

22.2.5.1 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_RST_EN_HMAC. For information on those registers, please see Chapter 8 *Reset and Clock*.
2. Write 1 to register HMAC_SET_START_REG.

22.2.5.2 Configure HMAC Keys and Key Purposes

1. Write the key purpose m to register HMAC_SET_PARA_PURPOSE_REG. The possible key purpose values are shown in Table 22.2-1. For more information, please refer to Section 22.2.4.
2. Select KEYn in eFuse memory as the key by writing n (ranges from 0 to 5) to register HMAC_SET_PARA_KEY_REG. For more information, please refer to Section 22.2.4.
3. Write 1 to register HMAC_SET_PARA_FINISH_REG to complete the configuration.
4. Read register HMAC_QUERY_ERROR_REG. If its value is 1, it means the purpose of the selected block does not match the configured key purpose and the calculation will not proceed. If its value is 0, it means the purpose of the selected block matches the configured key purpose, and then the calculation can proceed.
5. When the value of HMAC_SET_PARA_PURPOSE_REG is not 8, it means the HMAC module is in downstream mode, proceed with 22.2.5.3. When the value is 8, it means the HMAC module is in upstream mode, proceed with 22.2.5.4.

22.2.5.3 Downstream Mode Process

1. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
2. To clear the result and make further usage of the dependent hardware (JTAG or DSA) impossible, write 1 to either register HMAC_SET_INVALIDATE_JTAG_REG to clear the result generated by the JTAG key; or to register HMAC_SET_INVALIDATE_DS_REG to clear the result generated by DSA key. Afterwards, the HMAC process needs to be restarted to re-enable any of the dependent peripherals.

22.2.5.4 Upstream Mode Process

1. Write message in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Apply SHA padding to message as described in Section 22.3.1, and get message block: Block_n ($n >= 0$)
   (c) Perform one of the following operations according to the block number n:
   - If there is only one message block in total which has included all padding bits:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_ONE_BLOCK_REG, and finish message transmission.
   - If Block_n is the last padded block:
     i. Write the 512-bit Block_0 to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Message transmission finished automatically.
   - If Block_n is the second last padded block:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_SET_MESSAGE_PAD_REG, and continue message transmission (jump back to step (c)).
   - If Block_n is neither the last nor the second last message block:
     i. Write the 512-bit Block_n to register HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15). Write 1 to register HMAC_SET_MESSAGE_ONE_REG, to trigger the processing of this message block.
     ii. Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
     iii. Write 1 to register HMAC_SET_MESSAGE_ING_REG, and continue message transmission (jump back to step (c)).
2. Read hash result in upstream mode:
   (a) Poll Status register HMAC_QUERY_BUSY_REG until it reads 0.
   (b) Read hash result from register HMAC_RD_RESULT_*n*_REG (*n*: 0-7).
   (c) Write 1 to register HMAC_SET_RESULT_FINISH_REG to finish calculation. The result will be cleared at the same time.
   (d) Upstream mode operation is completed.

Note:
The SHA accelerator can be called directly, or used internally by the DSA module and the HMAC module. However, they can not share the hardware resources simultaneously. Therefore, the SHA module must not be called neither by the CPU nor by the DSA module when the HMAC module is in use.

22.3 HMAC Algorithm Details

22.3.1 Padding Bits

The HMAC module uses SHA-256 as hash algorithm. If the input message is not a multiple of 512 bits, the user must apply a SHA-256 padding algorithm in software. The SHA-256 padding algorithm is the same as described in Section *Padding the Message* of FIPS PUB 180-4. In downstream mode, users do not need to input any message or apply padding. The HMAC module uses a default 32-byte pattern of 0x00 for re-enabling JTAG and a 32-byte pattern of 0xff for deriving the AES key for the DSA module.

For the convenience of reading, here we will briefly describe the process of message padding. As shown in Figure 22.3-1, suppose the length of the unpadded message is m bits. Padding steps are as follows:

1. Append one bit of value "1" to the end of the unpadded message.
2. Append k bits of value "0", where k is the smallest non-negative number which satisfies $m+1+k \equiv 448 (mod 512)$.
3. Append a 64-bit integer value as a binary block. This block consists of the length of the unpadded message as a big-endian binary integer value m.

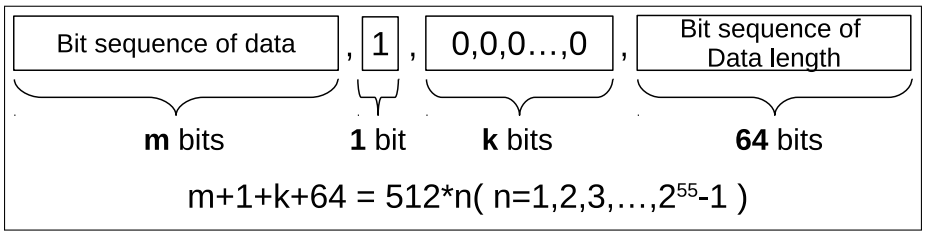


Figure 22.3-1. HMAC SHA-256 Padding Diagram

In upstream mode, if the length of the unpadded message is a multiple of 512 bits, users can configure hardware to apply SHA padding by writing 1 to HMAC_SET_MESSAGE_END_REG or do padding work themselves by writing 1 to HMAC_SET_MESSAGE_PAD_REG. If the length is not a multiple of 512 bits, SHA padding must be manually applied by the user. After the user prepared the padding data, they should complete the subsequent configuration according to the Section 22.2.5.

22.3.2 HMAC Algorithm Structure

The structure of the implemented algorithm in the HMAC module is shown in Figure 22.3-2. This is the standard HMAC algorithm as described in RFC 2104.

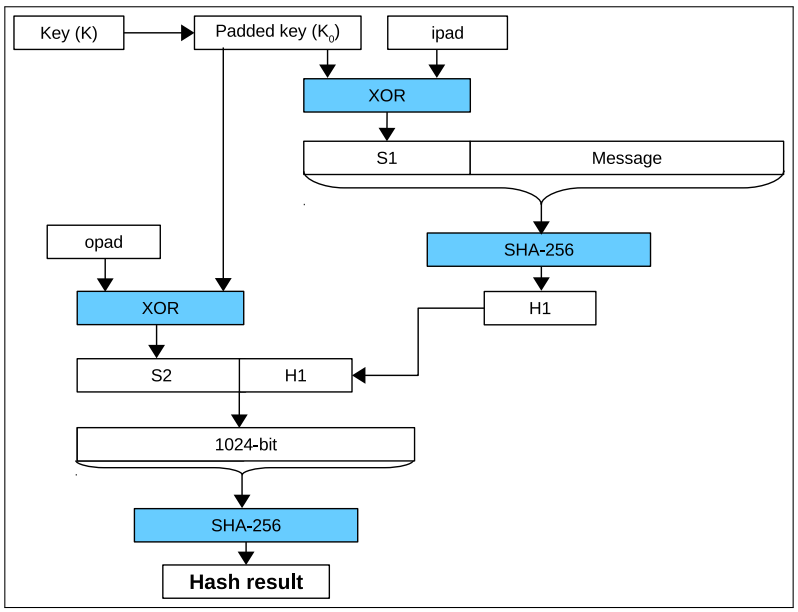


Figure 22.3-2. HMAC Structure Schematic Diagram

In Figure 22.3-2:

1. ipad is a 512-bit message block composed of 64 bytes of 0x36.
2. opad is a 512-bit message block composed of 64 bytes of 0x5c.

The HMAC module appends a 256-bit 0 sequence after the bit sequence of the 256-bit key K in order to get a 512-bit K_0. Then, the HMAC module XORs K_0 with ipad to get the 512-bit S1. Afterwards, the HMAC module appends the input message (multiple of 512 bits) after the 512-bit S1, and exercises the SHA-256 algorithm to get the 256-bit H1.

The HMAC module appends the 256-bit SHA-256 hash result H1 to the 512-bit S2 value, which is calculated using the XOR operation of K_0 and opad. A 768-bit sequence will be generated. Then, the HMAC module uses the SHA padding algorithm described in Section 22.3.1 to pad the 768-bit sequence to a 1024-bit sequence, and applies the SHA-256 algorithm to get the final hash result (256-bit).

22.4 Register Summary

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

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

Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC message control register	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC result reading finish register	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	Invalidate digital signature result register	0x0064	WO
HMAC_QUERY_ERROR_REG	Stores matching results between keys generated by users and corresponding purposes	0x0068	RO
HMAC_QUERY_BUSY_REG	Busy state of HMAC module	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	Software padding register	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block message register	0x00F4	WO
Configuration Registers			
HMAC_SET_PARA_PURPOSE_REG	HMAC parameter configuration register	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC parameters configuration register	0x0048	WO
HMAC_SOFT_JTAG_CTRL_REG	Re-enable JTAG register 0	0x00F8	WO
HMAC_WR_JTAG_REG	Re-enable JTAG register 1	0x00FC	WO
HMAC Message Block			
HMAC_WR_MESSAGE_0_REG	Message register 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	Message register 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	Message register 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	Message register 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	Message register 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	Message register 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	Message register 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	Message register 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	Message register 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	Message register 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	Message register 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	Message register 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	Message register 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	Message register 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	Message register 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	Message register 15	0x00BC	WO
HMAC Upstream Result			

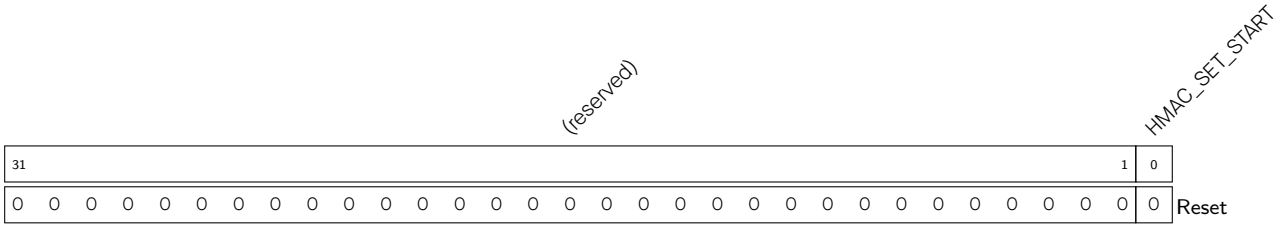
Name	Description	Address	Access
HMAC_RD_RESULT_0_REG	Hash result register 0	0x00C0	RO
HMAC_RD_RESULT_1_REG	Hash result register 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash result register 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash result register 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash result register 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash result register 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash result register 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash result register 7	0x00DC	RO
Version Register			
HMAC_DATE_REG	Version control register	0x00F8	R/W

22.5 Registers

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

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

Register 22.1. HMAC_SET_START_REG (0x0040)



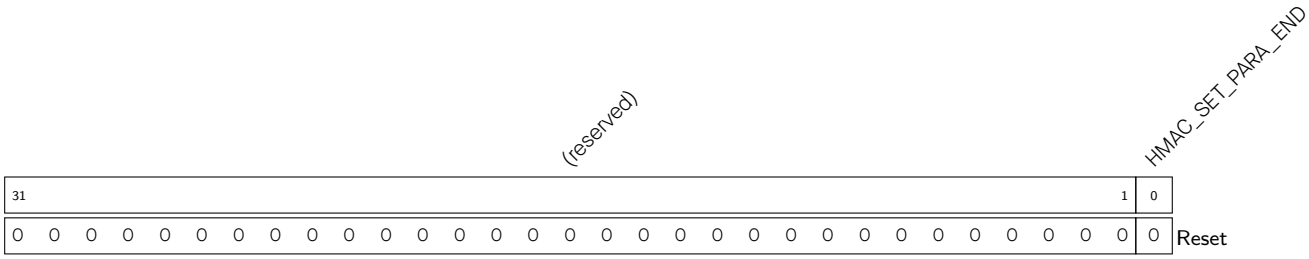
HMAC_SET_START Configures whether or not to enable HMAC.

0: Disable HMAC

1: Enable HMAC

(WO)

Register 22.2. HMAC_SET_PARA_FINISH_REG (0x004C)



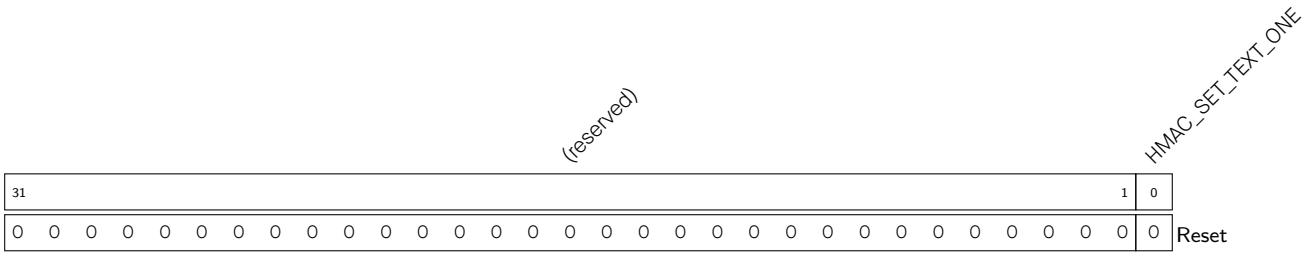
HMAC_SET_PARA_END Configures whether to finish HMAC configuration.

0: No effect

1: Finish configuration

(WO)

Register 22.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



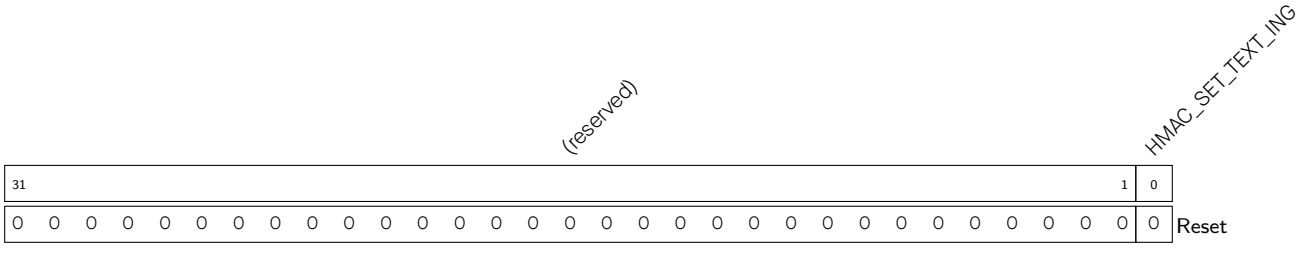
HMAC_SET_TEXT_ONE Calls SHA to calculate one message block.

0: No effect

1: Calls SHA to calculate one message block.

(WO)

Register 22.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



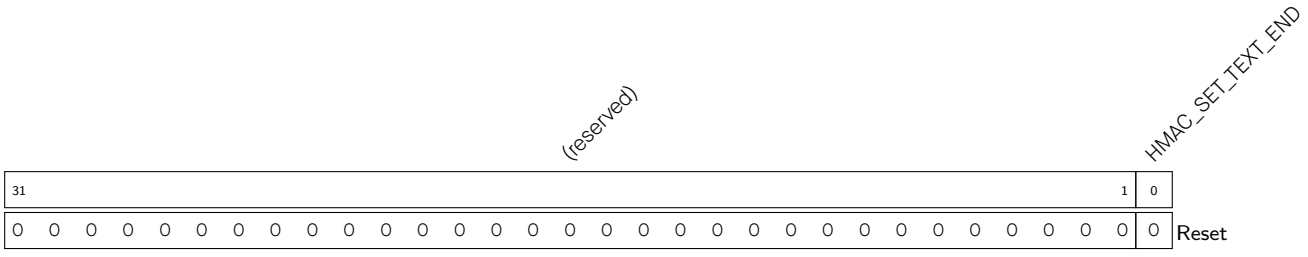
HMAC_SET_TEXT_ING Configures whether or not there are unprocessed message blocks.

0: No unprocessed message block

1: There are still some message blocks to be processed.

(WO)

Register 22.5. HMAC_SET_MESSAGE_END_REG (0x0058)



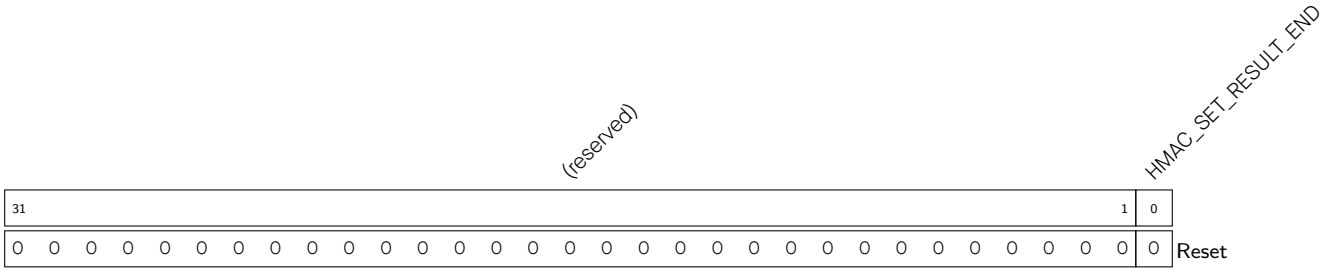
HMAC_SET_TEXT_END Configures whether to start hardware padding.

0: No effect

1: Start hardware padding

(WO)

Register 22.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



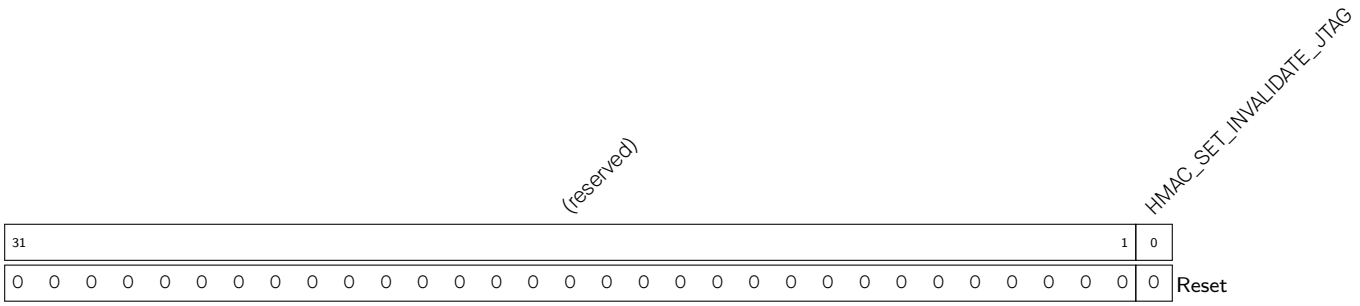
HMAC_SET_RESULT_END Configures whether to exit upstream mode and clear calculation results.

0: Not exit

1: Exit upstream mode and clear calculation results.

(WO)

Register 22.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



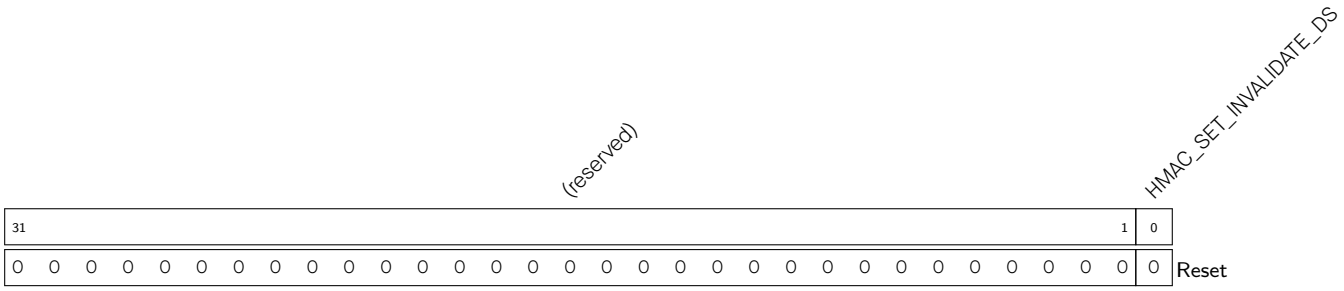
HMAC_SET_INVALIDATE_JTAG Configures whether or not to clear calculation results when re-enabling JTAG in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 22.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



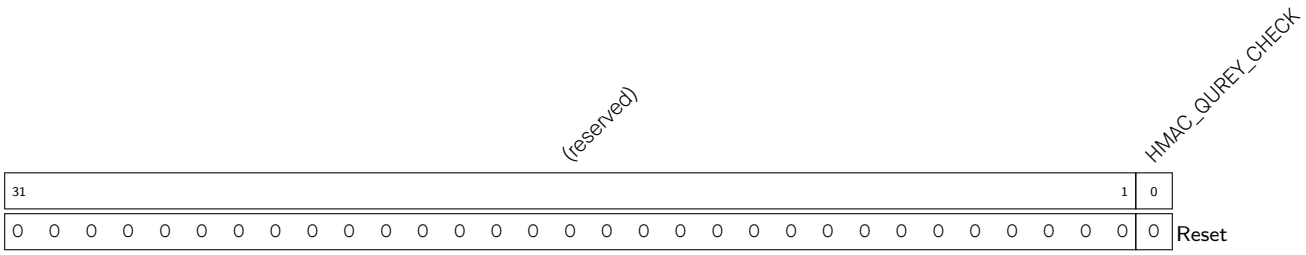
HMAC_SET_INVALIDATE_DS Configures whether or not to clear calculation results of the DS module in downstream mode.

0: Not clear

1: Clear calculation results

(WO)

Register 22.9. HMAC_QUERY_ERROR_REG (0x0068)



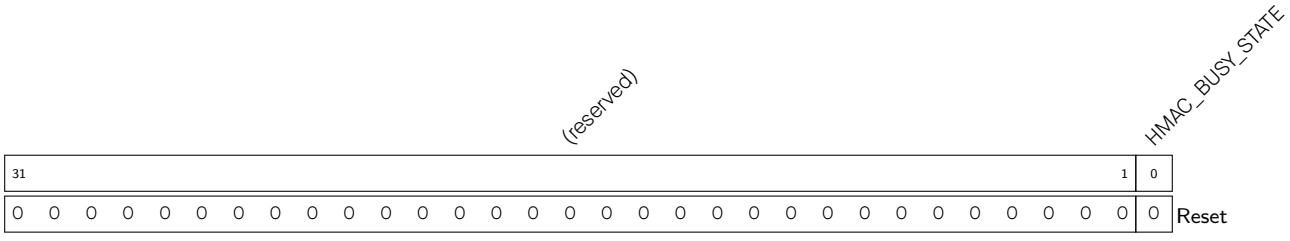
HMAC_QUREY_CHECK Represents whether or not an HMAC key matches the purpose.

0: Match

1: Error

(RO)

Register 22.10. HMAC_QUERY_BUSY_REG (0x006C)



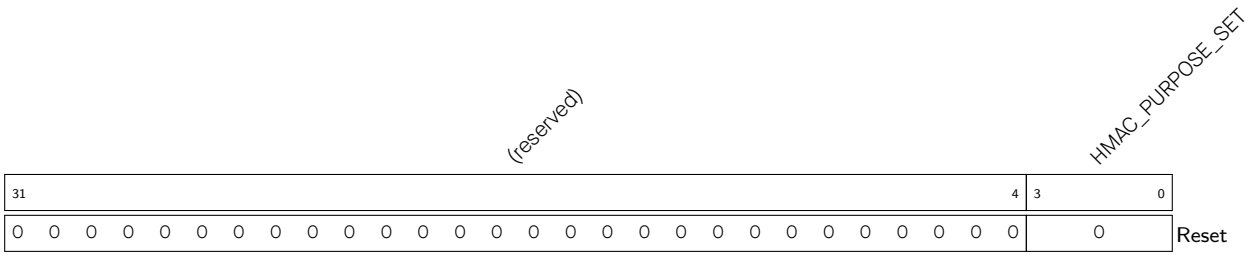
HMAC_BUSY_STATE Represents whether or not HMAC is in a busy state. Before configuring HMAC, please make sure HMAC is in an IDLE state.

0: Idle

1: HMAC is still working on the calculation

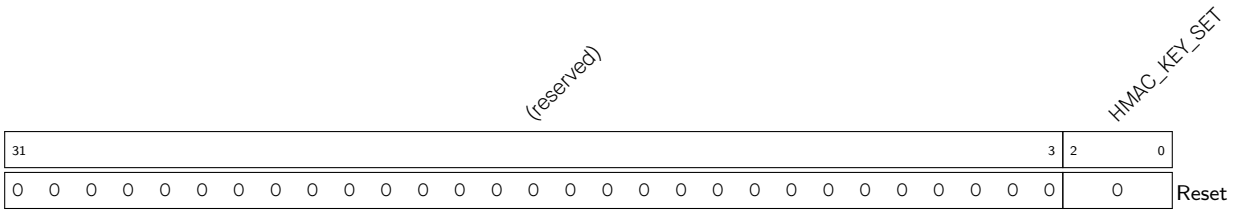
(RO)

Register 22.11. HMAC_SET_PARA_PURPOSE_REG (0x0044)



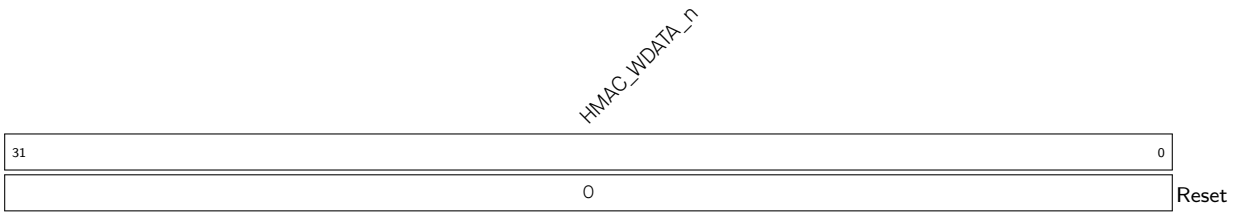
HMAC_PURPOSE_SET Configures the HMAC purpose, refer to the Table 22.2-1. (WO)

Register 22.12. HMAC_SET_PARA_KEY_REG (0x0048)



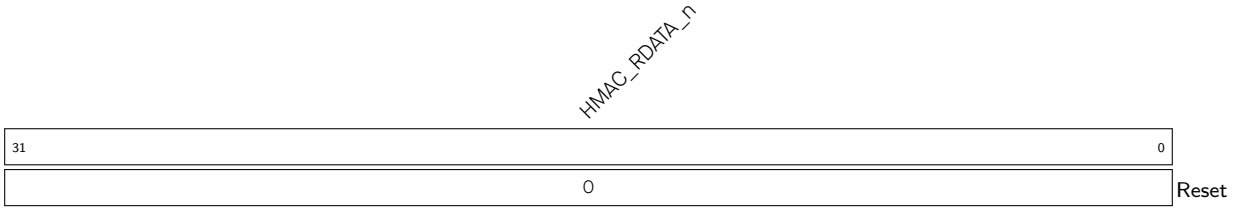
HMAC_KEY_SET Configures HMAC key. There are six keys with index 0~5. Write the index of the selected key to this field. (WO)

Register 22.13. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4**n*)



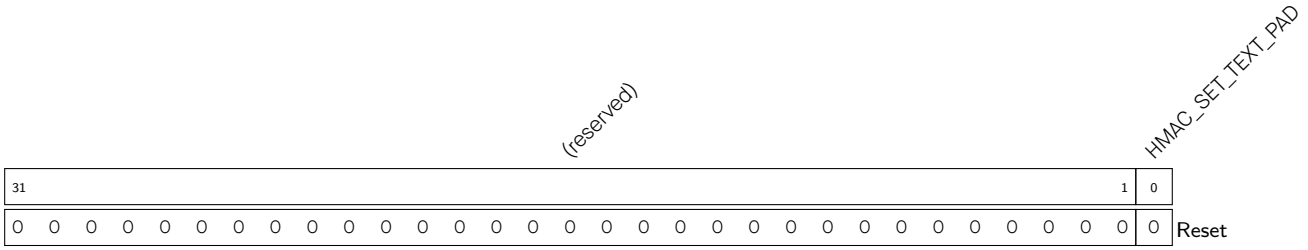
HMAC_WDATA_*n* Represents the *n*th 32-bit of message. (WO)

Register 22.14. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4**n*)



HMAC_RDATA_*n* Represents the *n*th 32-bit of hash result. (RO)

Register 22.15. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



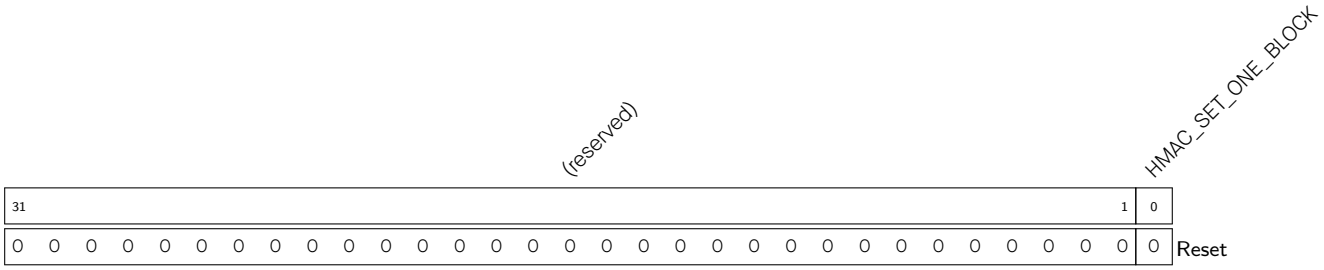
HMAC_SET_TEXT_PAD Configures whether or not the padding is applied by software.

0: Not applied by software

1: Applied by software

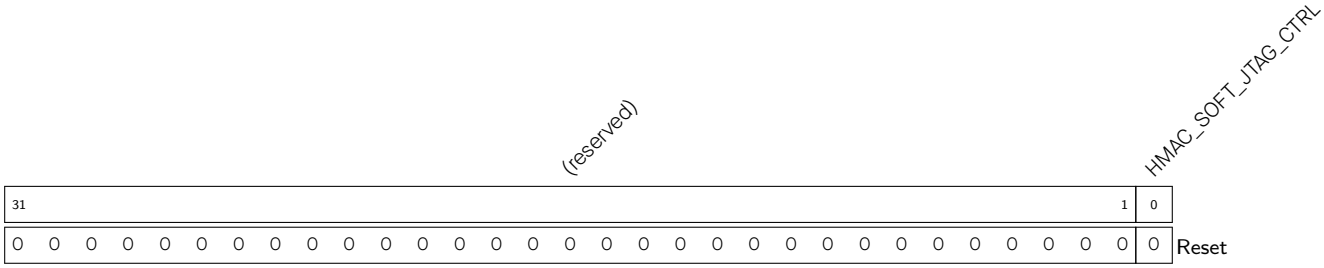
(WO)

Register 22.16. HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK Write 1 to indicate there is only one block which already contains padding bits and there is no need for padding. (WO)

Register 22.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



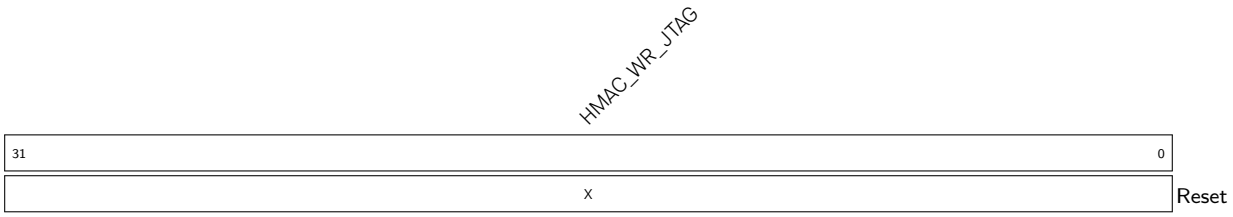
HMAC_SOFT_JTAG_CTRL Configures whether or not to enable JTAG authentication mode.

0: Disable

1: Enable

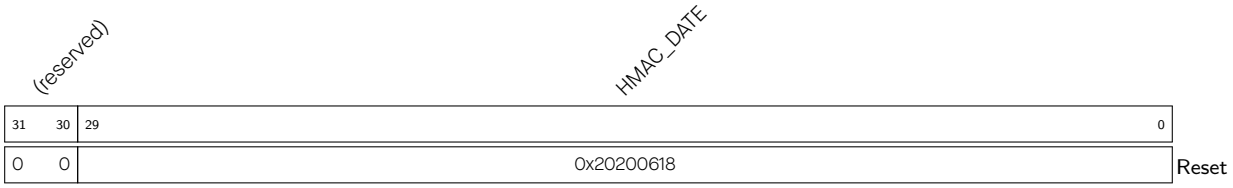
(WO)

Register 22.18. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG Writes the comparing input used for re-enabling JTAG. (WO)

Register 22.19. HMAC_DATE_REG (0x00F8)



HMAC_DATE Version control register. (R/W)

Chapter 23

RSA Accelerator (RSA)

23.1 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.

23.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two additional acceleration options to further increase the calculation speed
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

23.3 Functional Description

The RSA accelerator is activated by setting the HP_SYS_CLKRST_REG_CRYPTO_RSA_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_REG_RST_EN_RSA bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. Additionally, users also need to clear HP_SYS_CLKRST_REG_RST_EN_DS and HP_SYS_CLKRST_REG_RST_EN_ECDSA bits to reset the Digital Signature Algorithm (DSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA) accelerators.

The RSA accelerator is only available after the RSA-related memories are initialized. The content of the RSA_QUERY_CLEAN_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, wait until RSA_QUERY_CLEAN_REG becomes 1 before using the RSA accelerator.

The RSA_INT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this field to enable or disable the interrupt. By default, the interrupt function of the RSA accelerator is enabled.

Notice:

ESP32-P4's Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) modules

also call the RSA accelerator when working. Therefore, users cannot access the RSA accelerator when the Digital Signature Algorithm (DSA) or the Elliptic Curve Digital Signature Algorithm (ECDSA) module is working.

23.3.1 Definitions and Representations

ESP32-P4's RSA accelerator supports operands of different lengths $N = 32 \times n$.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the values in $Z_{n-1}\cdots Z_0$, $X_{n-1}\cdots X_0$, $Y_{n-1}\cdots Y_0$, $M_{n-1}\cdots M_0$, $\overline{r}_{n-1}\cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional argument $\overline{r}$ can be calculated as $\overline{r} = R^2 \bmod M$.

Also, argument M' can be calculated using the formula below:

$$M' = -M^{-1} \bmod b$$

where, M^{-1} is the modular multiplicative inverse of M, and it can be calculated with the extended binary GCD algorithm.

23.3.2 Large-Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

Large-number modular exponentiation on the ESP32-P4 can be implemented as follows:

1. Write 1 or 0 to the RSA_INT_ENA field to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.

(b) Write M' to the RSA_M_PRIME_REG register.

(c) Configure registers related to the acceleration options, which are described later in Section 23.3.5.

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.

23.3.3 Large-Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. Therefore, similar to the large-number modular exponentiation, two additional arguments are needed – $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA accelerator supports large-number modular multiplication with operands of length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 128\}$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant configuration registers:

   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.

   (b) Write M' to the RSA_M_PRIME_REG register.

3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM, and RSA_Z_MEM, respectively. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.

4. Write 1 to the RSA_SET_START_MODMULT field.

5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt, .

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

23.3.4 Large-Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA accelerator only supports large-number multiplication with operand length $N = 32 \times n$, where $n \in \{1, 2, 3, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e., $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n-1\}$ to the address of the i words of the RSA_X_MEM memory block. Note that Y_i for $i \in \{0, 1, \ldots, n-1\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e., the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length is ignored.
4. Write 1 to the RSA_SET_START_MULT register.
5. Wait for the completion of computation, which happens when the content of RSA_QUERY_IDLE becomes 1 or the RSA interrupt occurs, if enabled.
6. Read the result Z_i for $i \in \{0, 1, \ldots, \hat{n}-1\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. If you have the interrupt enabled, write 1 to RSA_CLEAR_INTERRUPT to clear the interrupt.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

23.3.5 Options for Additional Acceleration

The ESP32-P4 RSA accelerator also provides SEARCH and CONSTANT_TIME options that can be configured to further accelerate the large-number modular exponentiation. By default, both options are configured as no additional acceleration.

Users can choose to use one or two of these options to further accelerate the computation. Note that, even when none of these two options is configured, using the hardware RSA accelerator is still much faster than implementing the RSA algorithm in software.

To be more specific, when neither of these two options are configured for additional acceleration, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. When either or both of these two options are configured for additional acceleration, the time required is also correlated with the 0 and 1 bit distribution in Y.

To better illustrate how these two options work, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e., the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When either of these two options is configured for additional acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for additional acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. Set α to a number smaller than N-1, which otherwise leads to the same result as if this option is not used for additional acceleration. The best acceleration performance can be achieved by setting α to t, in which case all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0 s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
  - Note that this option compromises the security because it ignores some bits, which essentially shortens the key length, thus should not be enabled for applications with high security requirement.
- CONSTANT_TIME Option (Configuring RSA_CONSTANT_TIME_REG to 0 for additional acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better is the acceleration performance.
  - Note that this option also compromises the security because its time cost correlates with the 0/1 distribution of the key, which can be used in a Side Channel Attack (SCA), thus should not be enabled for applications with high security requirement.

Below is an example to demonstrate the performance of the RSA accelerator under different combinations of SEARCH and CONSTANT_TIME configuration. In this example:

- We perform $Z = X^Y \bmod M$
- N = 3072

- Y = 65537
- X and M are taken at random
- When the SEARCH option is enabled, α, i.e., the position register RSA_SEARCH_POS_REG, is set to 16.

Table 23.3-1 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 23.3-1. Acceleration Performance

SEARCH Option	CONSTANT_TIME Option	Time Cost (clock cycle)
No acceleration	No acceleration	174.7×10^6
Accelerated	No acceleration	1.023×10^6
No acceleration	Acceleration	0.546×10^6
Acceleration	Acceleration	0.540×10^6

As shown in Table 23.3-1:

- 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.

23.4 Interrupts

ESP32-P4's RSA accelerator can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- RSA_INTR

There is one internal interrupt source from the RSA accelerator that can generate the above interrupt signal. The interrupt source from the RSA accelerator is listed with its trigger condition and the resulting interrupt signal in Table 23.4-1.

Table 23.4-1. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

23.5 Memory Summary

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

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

Table 23.5-1. 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

23.6 Register Summary

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

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

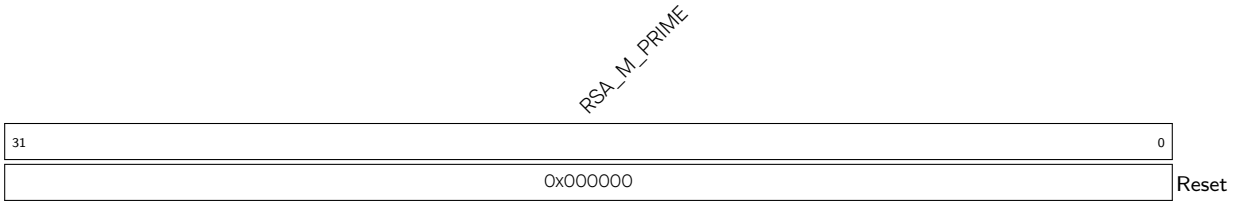
Name	Description	Address	Access
Control or Configuration Registers			
RSA_M_PRIME_REG	Represents M'	0x0800	R/W
RSA_MODE_REG	Configures RSA length	0x0804	R/W
RSA_SET_START_MODEXP_REG	Starts modular exponentiation	0x080C	WT
RSA_SET_START_MODMULT_REG	Starts modular multiplication	0x0810	WT
RSA_SET_START_MULT_REG	Starts multiplication	0x0814	WT
RSA_QUERY_IDLE_REG	Represents the RSA status	0x0818	RO
RSA_CONSTANT_TIME_REG	Configures the constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	Configures the search option	0x0824	R/W
RSA_SEARCH_POS_REG	Configures the search position	0x0828	R/W
Status Register			
RSA_QUERY_CLEAN_REG	RSA initialization status	0x0808	RO
Interrupt Registers			
RSA_INT_CLR_REG	Clears RSA interrupt	0x081C	WT
RSA_INT_ENA_REG	Enables the RSA interrupt	0x082C	R/W
Version Control Register			
RSA_DATE_REG	Version control register	0x0830	R/W

23.7 Registers

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

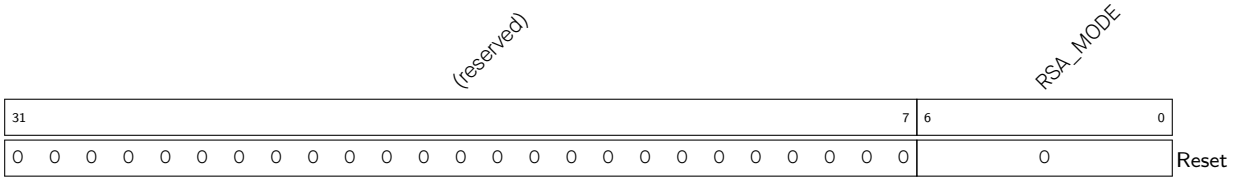
For how to program reserved fields, please refer to Section IX *Programming Reserved Register Field*.

Register 23.1. RSA_M_PRIME_REG (0x0800)



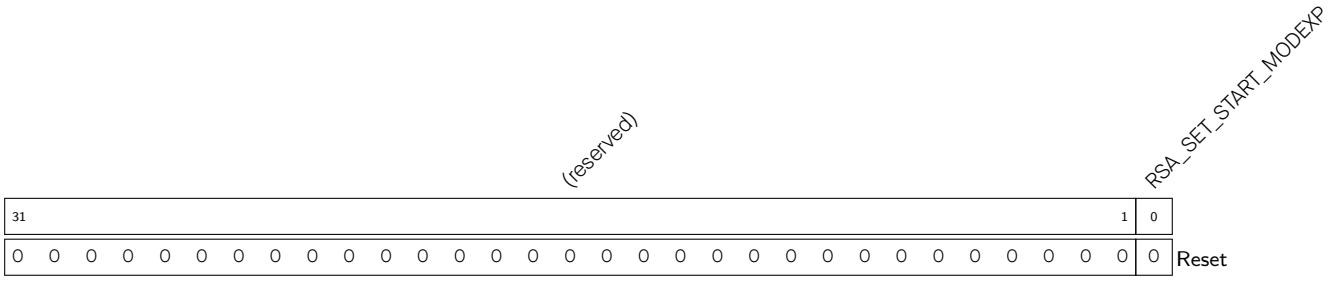
RSA_M_PRIME Represents M'. (R/W)

Register 23.2. RSA_MODE_REG (0x0804)



RSA_MODE Configures the RSA length. (R/W)

Register 23.3. RSA_SET_START_MODEXP_REG (0x080C)



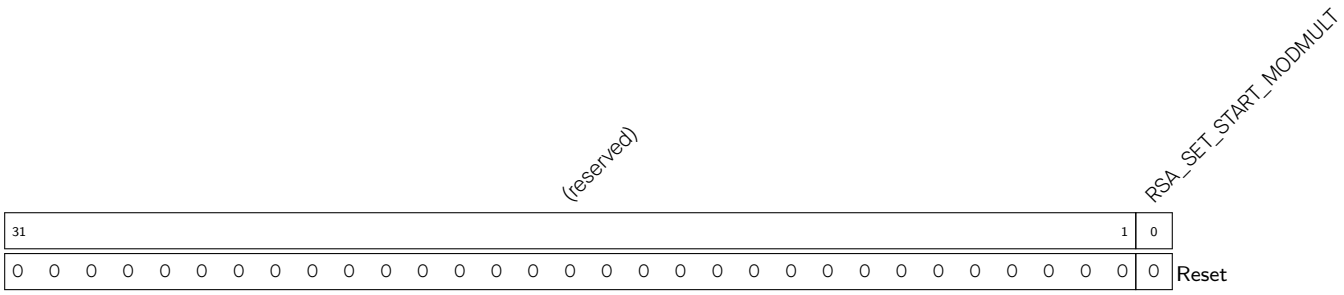
RSA_SET_START_MODEXP Configures whether or not to starts the modular exponentiation.

0: No effect

1: Start

(WT)

Register 23.4. RSA_SET_START_MODMULT_REG (0x0810)



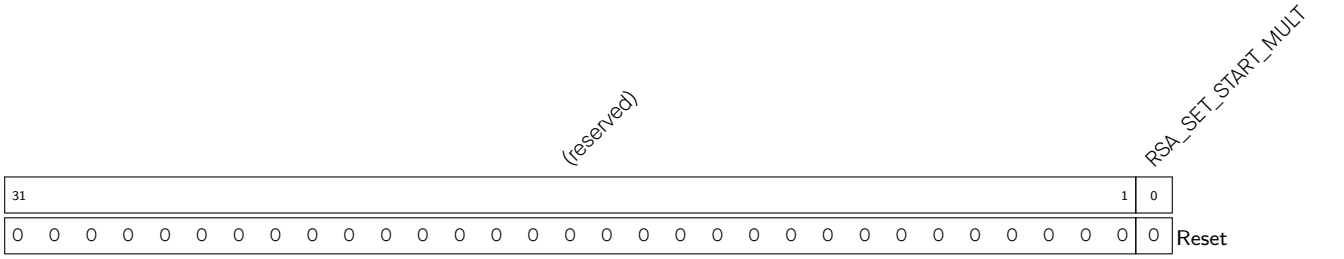
RSA_SET_START_MODMULT Configures whether or not to start the modular multiplication.

0: No effect

1: Start

(WT)

Register 23.5. RSA_SET_START_MULT_REG (0x0814)



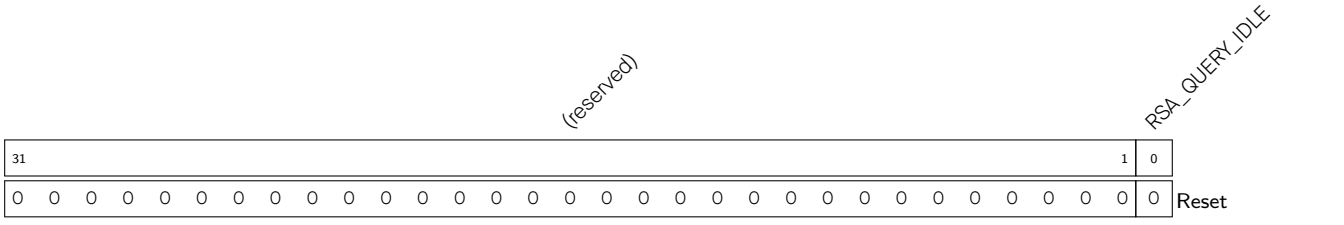
RSA_SET_START_MULT Configures whether or not to start the multiplication.

0: No effect

1: Start

(WT)

Register 23.6. RSA_QUERY_IDLE_REG (0x0818)



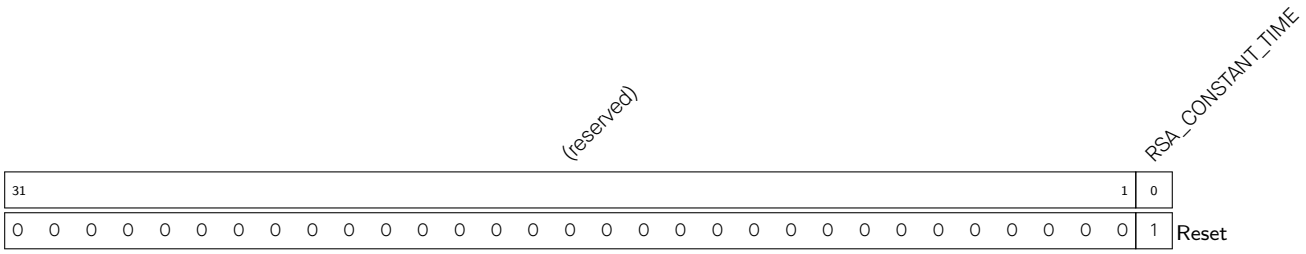
RSA_QUERY_IDLE Represents the RSA status.

0: Busy

1: Idle

(RO)

Register 23.7. RSA_CONSTANT_TIME_REG (0x0820)



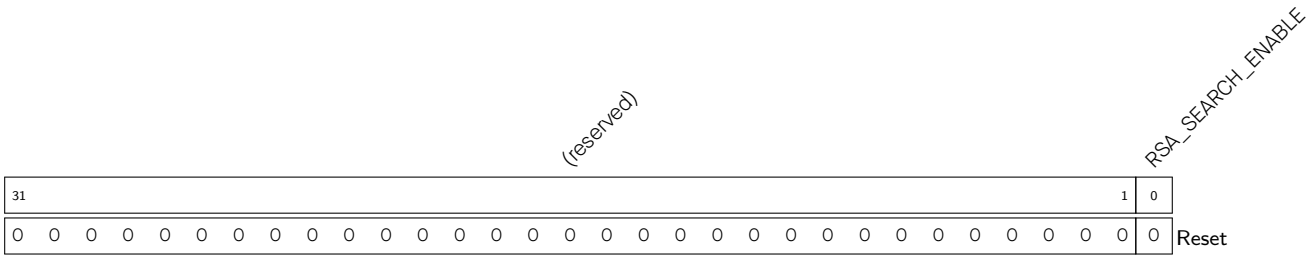
RSA_CONSTANT_TIME Configures the constant_time option.

0: Acceleration

1: No acceleration (default)

(R/W)

Register 23.8. RSA_SEARCH_ENABLE_REG (0x0824)



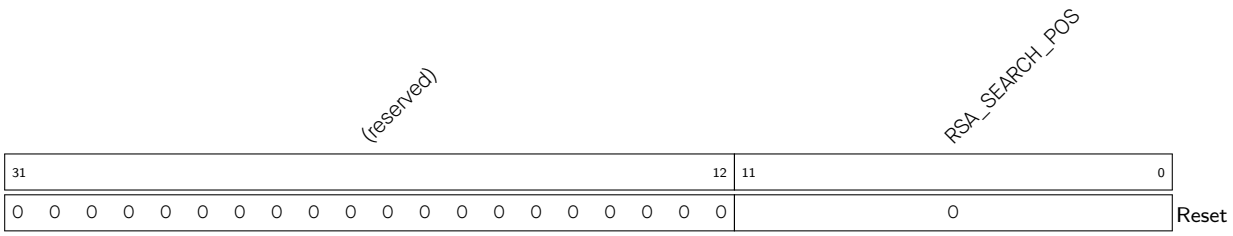
RSA_SEARCH_ENABLE Configures the search option.

0: No acceleration (default)

1: Acceleration

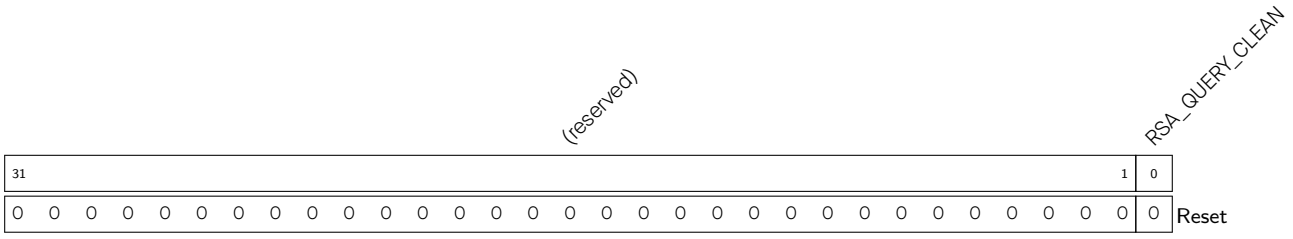
This option should be used together with RSA_SEARCH_POS_REG. (R/W)

Register 23.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS Configures the starting address to start search. This field should be used together with RSA_SEARCH_ENABLE_REG. The field is only valid when RSA_SEARCH_ENABLE is high. (R/W)

Register 23.10. RSA_QUERY_CLEAN_REG (0x0808)



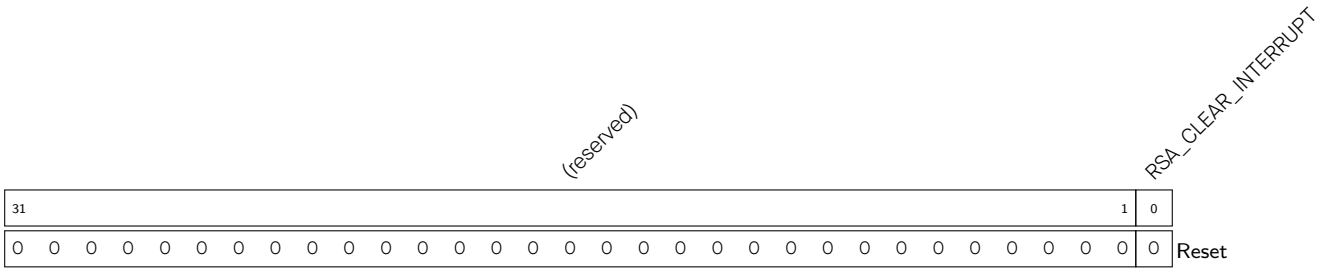
RSA_QUERY_CLEAN Represents whether or not the RSA memory completes initialization.

0: Not complete

1: Completed

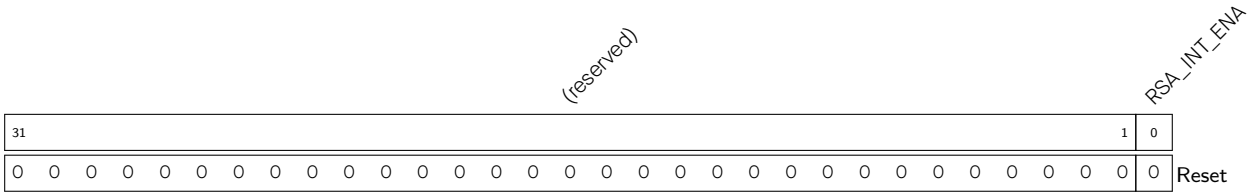
(RO)

Register 23.11. RSA_INT_CLR_REG (0x081C)



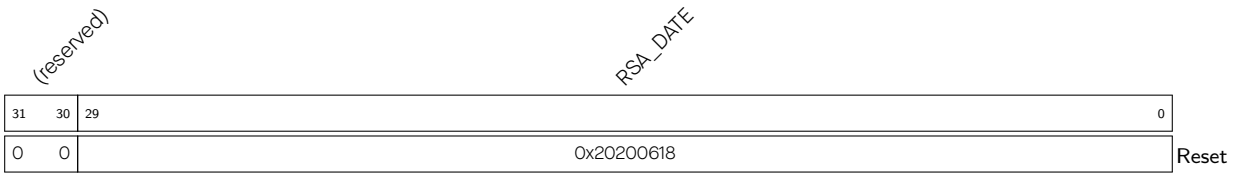
RSA_CLEAR_INTERRUPT Write 1 to clear the RSA interrupt. (WT)

Register 23.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA Write 1 to enable the RSA interrupt. (R/W)

Register 23.13. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

Chapter 24

SHA Accelerator (SHA)

24.1 Introduction

ESP32-P4 integrates an SHA accelerator, which is a hardware device that speeds up the SHA algorithm significantly, compared to an SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-P4 has two working modes, which are Typical SHA and DMA-SHA.

24.2 Features

The following functionality is supported:

- The following hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/256
  - SHA-512/*t*
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleaved function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

24.3 Working Modes

The SHA accelerator integrated in ESP32-P4 has two working modes.

- Typical SHA Working Mode: all the data is written and read via CPU directly.
- DMA-SHA Working Mode: all the data is read via DMA. That is, users can configure the GDMA-AXI controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

The SHA accelerator is activated by setting the HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN register field and clearing the HP_SYS_CLKRST_RST_EN_SHA register field. Additionally, due to the hardware reuse relationship between cryptography modules, set the related reset register field of Digital Signature Algorithm (DSA), HMAC Accelerator (HMAC), and Elliptic Curve Digital Signature Algorithm (ECDSA) to 1 can reset the SHA module. Therefore, users also need to clear HP_SYS_CLKRST_RST_EN_DS, HP_SYS_CLKRST_RST_EN_HMAC, and HP_SYS_CLKRST_RST_EN_ECDSA fields.

Users can start the SHA accelerator with different working modes by configuring registers SHA_START_REG and SHA_DMA_START_REG. For details, please see Table 24.3-1.

Table 24.3-1. 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 24.3-2.

Table 24.3-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/*t*	7

Notice:

ESP32-P4's Digital Signature Algorithm (DSA) and HMAC Accelerator (HMAC) modules also call the SHA accelerator when working. Therefore, users cannot access the SHA accelerator when these modules are working.

24.4 Function Description

The SHA accelerator generates the message digest via two steps: Preprocessing and Hash operation.

24.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

24.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 or 1024 bits, depending on the algorithm. Thus, all the messages should be padded to a multiple of 512 bits or 1024 bits before the hash operation.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

- **SHA-1**, **SHA-224** and **SHA-256**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k bits of zeros, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
  3. Last, append the 64-bit block of value equal to the number m expressed using a binary representation.
- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896 \ mod \ 1024$;
  3. Last, append the 128-bit block of value equal to the number m expressed using a binary representation.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

24.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit or 1024-bit blocks. During the task, all the message blocks should be written into the SHA_M_*n*_REG.

- For **SHA-1**, **SHA-224** and **SHA-256**:

  The message and its padding are parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, $M_0^{(i)}$ should be stored in SHA_M_0_REG, $M_1^{(i)}$ stored in SHA_M_1_REG, ..., and $M_{15}^{(i)}$ stored in SHA_M_15_REG.

- For **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**:
  The message and its padding are parsed into N 1024-bit blocks. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

  During the task, the most significant 32 bits and the least significant 32 bits of $M_0^{(i)}$ should be stored in SHA_M_0_REG and SHA_M_1_REG, respectively, ..., the most significant 32 bits and the least significant 32 bits of $M_{15}^{(i)}$ should be stored in SHA_M_30_REG and SHA_M_31_REG, respectively.

Note:

For more information about "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

24.4.1.3 Setting the Initial Hash Value

Before hash task begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t\_length = 7'h48$ and t_string is padded in the following format:

$8'h30 + 8'ht_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and t_string = $32'h38800000$.

   - If $10 <= t <= 99$, then $t\_length = 7'h50$ and t_string is padded in the following format:

$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and t_string = $32'h35368000$.

   - If $100 <= t < 512$, then $t\_length = 7'h58$ and t_string is padded in the following format:

$8'h30 + 8'ht_2$	$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and t_string = $32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, which is performing SHA-512 operation (with its initial hash value set to the result of 8-bitwise XOR operation of C and 0xa5) from the hexadecimal representation of the string "SHA-512/t".

24.4.2 Hash Operation

After the preprocessing, the ESP32-P4 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP32-P4 SHA accelerator supports two working modes, which are Typical SHA and DMA-SHA. The operation process for the SHA accelerator under two working modes is described in the following subsections.

24.4.2.1 Typical SHA Mode Process

Usually, the SHA accelerator will process all blocks of a message and produce a message digest before starting the computation of the next message digest.

However, ESP32-P4 SHA also supports optional "interleaved" message digest calculation in Typical SHA mode, which means before SHA completes all blocks of the current message, users are given a chance to insert new computation of another message digest upon the completion of each individual block of the current message.

Specifically, users can read out the message digest from registers SHA_H_*n*_REG after completing part of a message digest calculation, and use the SHA accelerator for a different calculation. After the different calculation completes, users can restore the previous message digest to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused calculation.

Typical SHA Process (except for SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 24.3-2.
2. Process the current message block.
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator[1].
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;
   - If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register to 7 for SHA-512/*t*.
2. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 24.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
3. Process the current message block[1].
   - Write the message block in registers SHA_M_*n*_REG.
4. Start the SHA accelerator
   - Set the SHA_CONTINUE_REG register to 1. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
5. Check the progress of the calculation.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status[3].
6. Decide if you have more message blocks to process:
   - If yes, please go back to Step 3.
   - Otherwise, please continue.
7. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Note:

1. In this step, the software can also write the next message block (to be processed) in registers SHA_M_*n*_REG, if any, while the hardware starts SHA calculation, to save time.
2. You are resuming the SHA accelerator with the previously paused calculation.
3. Here you can decide if you want to insert other calculations. If yes, please go to the process for interleaved calculations for details.

As mentioned above, ESP32-P4 SHA accelerator supports **"interleaving" calculation under the Typical SHA working mode**.

The process to implement interleaved calculation is described below.

1. Prepare to hand the SHA accelerator over for an interleaved calculation by storing the following data of the previous calculation.
   - The selected hash algorithm configured in the SHA_MODE_REG register.
   - The message digest stored in registers SHA_H_*n*_REG.

2. Perform the interleaved calculation. For the detailed process of the interleaved calculation, please refer to Typical SHA process or DMA-SHA process, depending on the working mode of your interleaved calculation.
3. Prepare to hand the SHA accelerator back to the previously paused calculation by restoring the following data of the previous calculation.
   - Write the previously stored hash algorithm back to register SHA_MODE_REG.
   - Write the previously stored message digest back to registers SHA_H_*n*_REG.
4. Write the next message block from the previous paused calculation in registers SHA_M_*n*_REG, and set the SHA_CONTINUE_REG register to 1 to restart the SHA accelerator with the previously paused calculation.

24.4.2.2 DMA-SHA Mode Process

ESP32-P4 SHA accelerator does not support "interleaving" message digest calculation at the level of individual message blocks when using DMA, which means you cannot insert new calculation before a complete DMA-SHA process (of one or more message blocks) completes. In this case, users who need interleaved operation are recommended to divide the message blocks and perform several DMA-SHA calculations, instead of trying to compute all the messages in one go.

Single DMA-SHA calculation supports up to 63 message blocks.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via GDMA-AXI. Therefore, users are required to configure the GDMA-AXI controller following the description in Chapter 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 24.3-2.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
4. Start the DMA-SHA calculation.
   - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.

6. Obtain the message digest:
    - Read the message digest from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/*t*

1. Select a hash algorithm.
    - Select SHA-512/*t* algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
    (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 24.4.1.3.
    (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.

24.4.3 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 24.4-4 below:

Table 24.4-4. The Storage and Length of Message Digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t*[2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG

[1] The message digest is stored in registers from most significant bits to the least significant bits, with the first word stored in register SHA_H_0_REG and the second word stored in register SHA_H_1_REG... For details, please see subsection 24.4.1.2.

[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.

24.5 Interrupt

ESP32-P4's SHA accelerator can generate the following interrupt signal(s) that will be sent to the Interrupt Matrix.

- SHA_INTR

There are several internal interrupt sources from SHA that can generate the above interrupt signal(s). The interrupt sources from SHA are listed with their trigger conditions and the resulted interrupt signal(s) in Table 24.5-1.

Table 24.5-1. 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 10

Interrupt Matrix > Section 10.2 *Interrupt Terminology in ESP32-P4*.

- Different from the standard interrupt register group, the interrupt register group of SHA only contains the INT_ENA (SHA_INT_ENA_REG) and INT_CLR (SHA_INT_CLEAR_REG) fields, and does not support users to read the INT_RAW and INT_ST fields.

24.6 Register Summary

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

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

Name	Description	Address	Access
Control/Configuration Registers			
SHA_MODE_REG	Configures SHA algorithm	0x0000	R/W
SHA_T_STRING_REG	Initial Hash Value calculation factor t_string (for SHA-512/t)	0x0004	R/W
SHA_T_LENGTH_REG	Initial Hash Value calculation factor t_length (for SHA-512/t)	0x0008	R/W
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
SHA_DMA_CONTINUE_REG	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	Block number register (only effective in DMA-SHA mode)	0x000C	R/W
Status Registers			
SHA_BUSY_REG	Represents if SHA Accelerator is busy or not	0x0018	RO
Interrupt Registers			
SHA_INT_CLEAR_REG	DMA-SHA interrupt clear register	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA interrupt enable register	0x0028	R/W
Data Registers			
SHA_H_0_REG	Hash value	0x0040	R/W
SHA_H_1_REG	Hash value	0x0044	R/W
SHA_H_2_REG	Hash value	0x0048	R/W
SHA_H_3_REG	Hash value	0x004C	R/W
SHA_H_4_REG	Hash value	0x0050	R/W
SHA_H_5_REG	Hash value	0x0054	R/W
SHA_H_6_REG	Hash value	0x0058	R/W
SHA_H_7_REG	Hash value	0x005C	R/W
SHA_H_8_REG	Hash value	0x0060	R/W
SHA_H_9_REG	Hash value	0x0064	R/W
SHA_H_10_REG	Hash value	0x0068	R/W
SHA_H_11_REG	Hash value	0x006C	R/W
SHA_H_12_REG	Hash value	0x0070	R/W
SHA_H_13_REG	Hash value	0x0074	R/W
SHA_H_14_REG	Hash value	0x0078	R/W
SHA_H_15_REG	Hash value	0x007C	R/W
SHA_M_0_REG	Message	0x0080	R/W
SHA_M_1_REG	Message	0x0084	R/W
SHA_M_2_REG	Message	0x0088	R/W
SHA_M_3_REG	Message	0x008C	R/W
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W

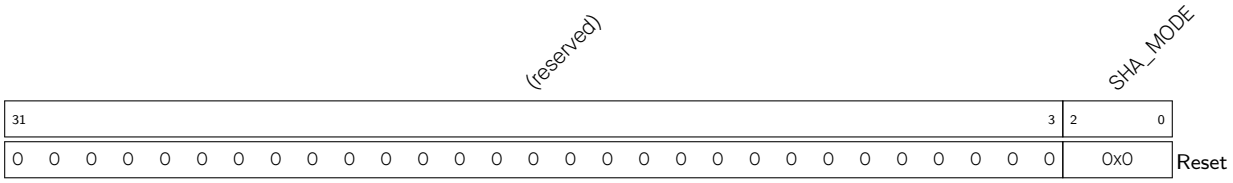
Name	Description	Address	Access
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	0x00AC	R/W
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W
SHA_M_16_REG	Message	0x00C0	R/W
SHA_M_17_REG	Message	0x00C4	R/W
SHA_M_18_REG	Message	0x00C8	R/W
SHA_M_19_REG	Message	0x00CC	R/W
SHA_M_20_REG	Message	0x00D0	R/W
SHA_M_21_REG	Message	0x00D4	R/W
SHA_M_22_REG	Message	0x00D8	R/W
SHA_M_23_REG	Message	0x00DC	R/W
SHA_M_24_REG	Message	0x00E0	R/W
SHA_M_25_REG	Message	0x00E4	R/W
SHA_M_26_REG	Message	0x00E8	R/W
SHA_M_27_REG	Message	0x00EC	R/W
SHA_M_28_REG	Message	0x00F0	R/W
SHA_M_29_REG	Message	0x00F4	R/W
SHA_M_30_REG	Message	0x00F8	R/W
SHA_M_31_REG	Message	0x00FC	R/W
Version Register			
SHA_DATE_REG	Version control register	0x002C	R/W

24.7 Registers

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

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

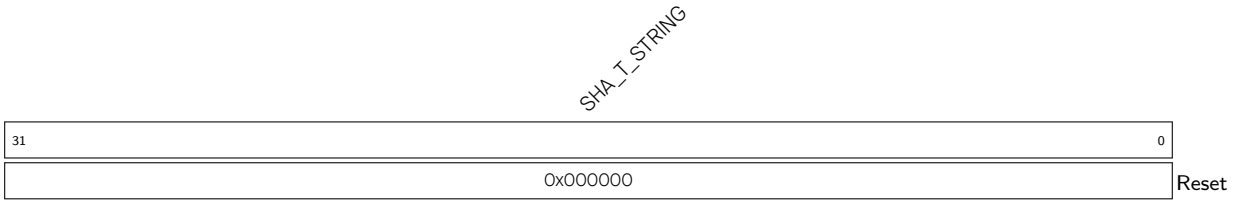
Register 24.1. SHA_MODE_REG (0x0000)



SHA_MODE Configures the SHA algorithm.

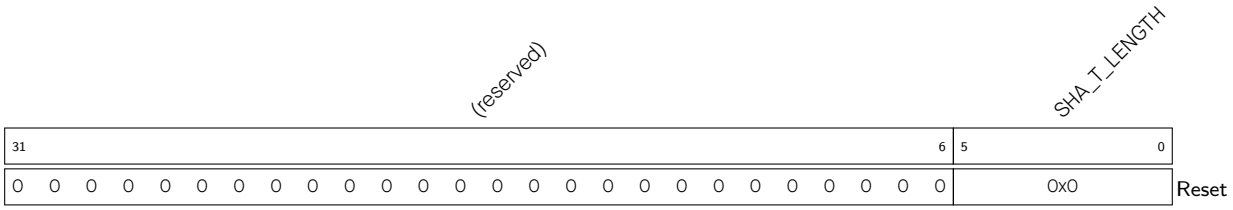
0: SHA-1
1: SHA-224
2: SHA-256
3: SHA-384
4: SHA-512
5: SHA-512/224
6: SHA-512/256
7: SHA-512/t
(R/W)

Register 24.2. SHA_T_STRING_REG (0x0004)



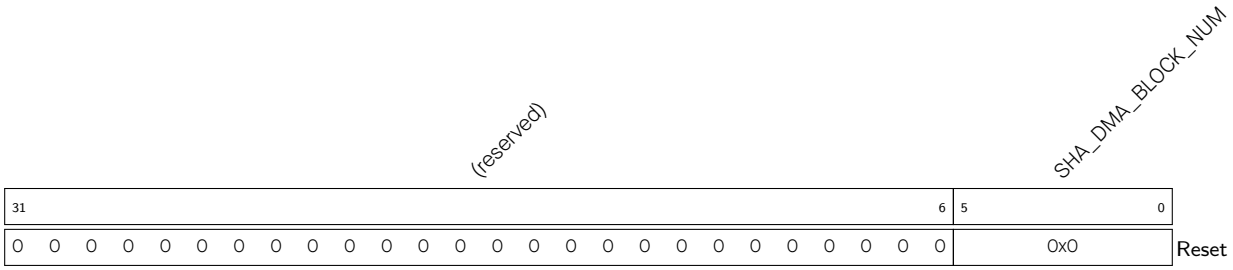
SHA_T_STRING Configures t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 24.3. SHA_T_LENGTH_REG (0x0008)



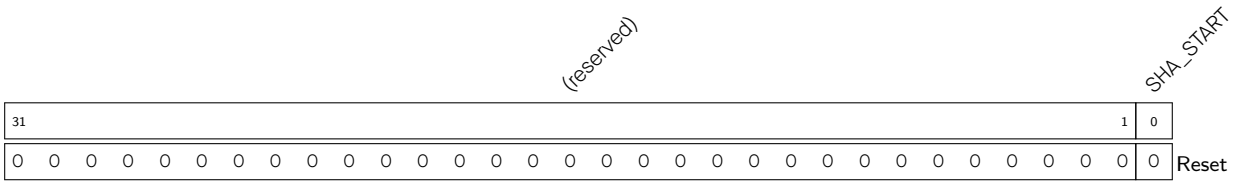
SHA_T_LENGTH Configures t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 24.4. SHA_DMA_BLOCK_NUM_REG (0x000C)



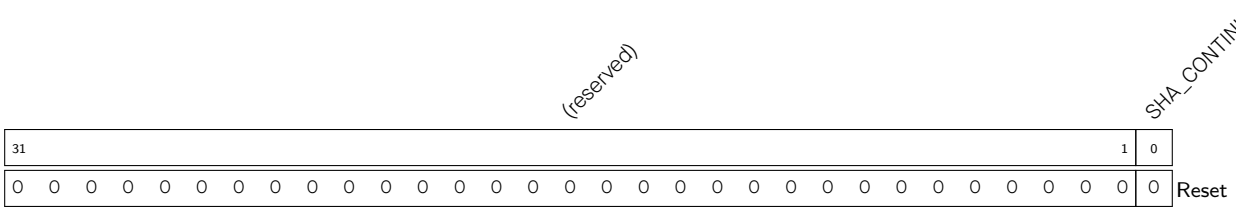
SHA_DMA_BLOCK_NUM Configures the DMA-SHA block number. (R/W)

Register 24.5. SHA_START_REG (0x0010)



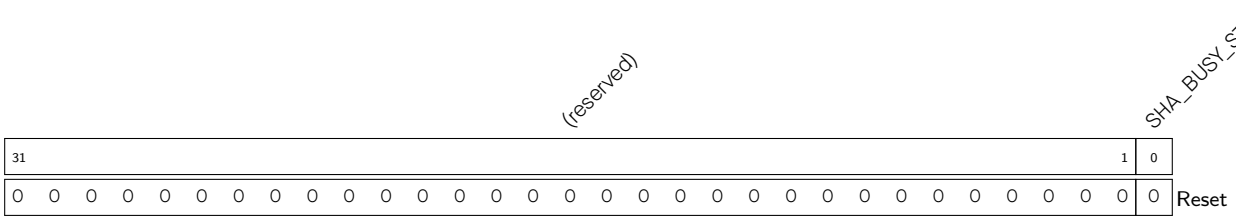
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 24.6. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 24.7. SHA_BUSY_REG (0x0018)



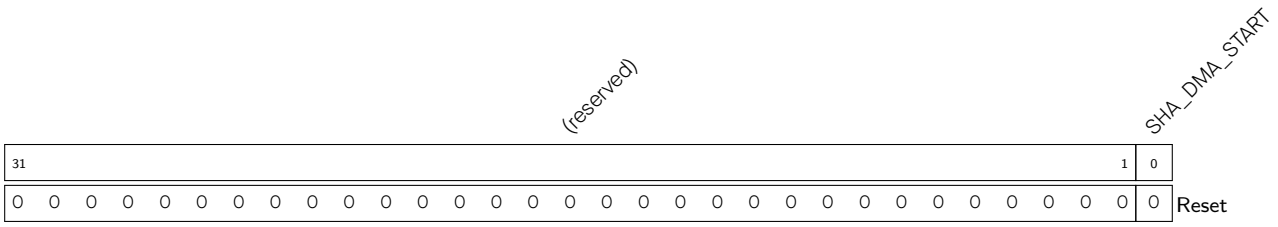
SHA_BUSY_STATE Represents the states of SHA accelerator.

0: idle

1: busy

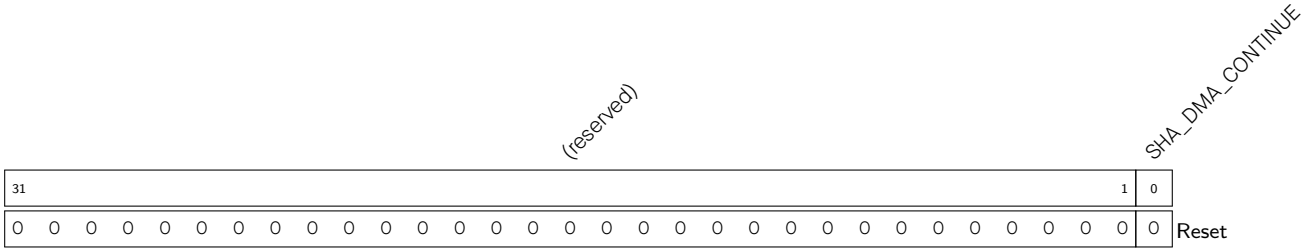
(RO)

Register 24.8. SHA_DMA_START_REG (0x001C)



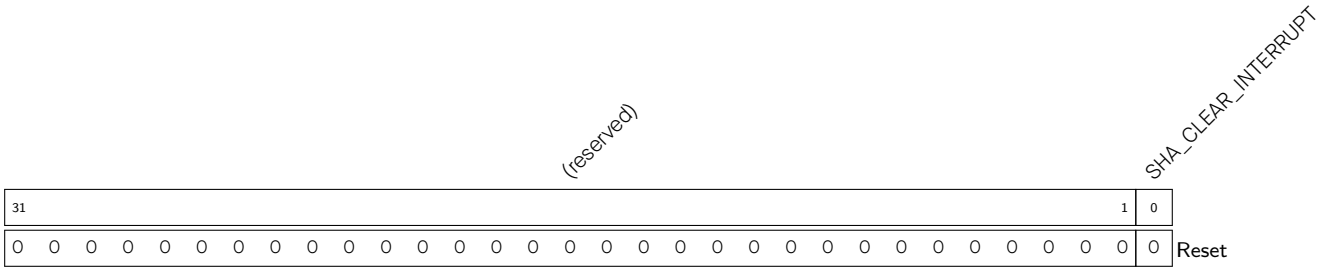
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 24.9. SHA_DMA_CONTINUE_REG (0x0020)



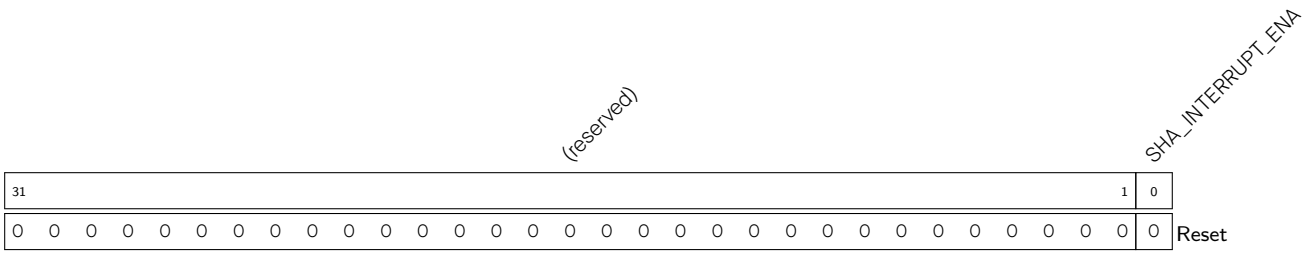
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 24.10. SHA_INT_CLEAR_REG (0x0024)



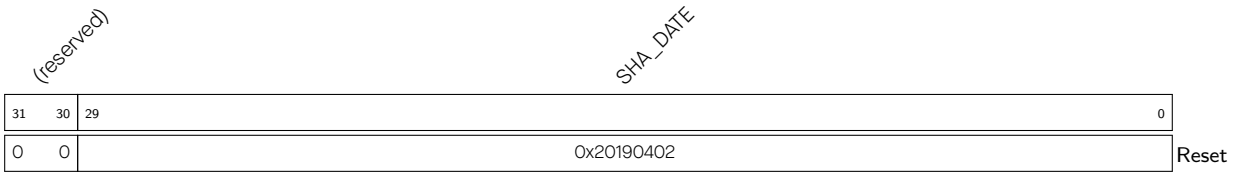
SHA_CLEAR_INTERRUPT Write 1 to clear DMA-SHA interrupt. (WO)

Register 24.11. SHA_INT_ENA_REG (0x0028)



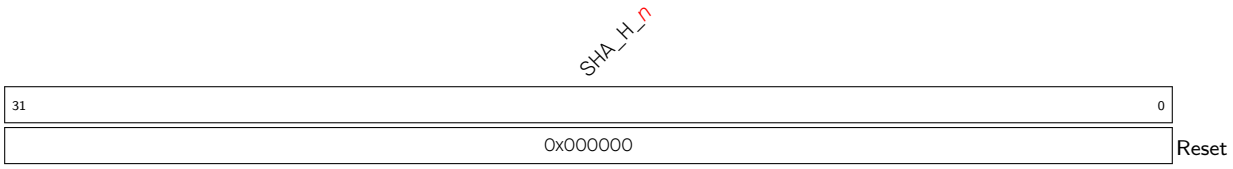
SHA_INTERRUPT_ENA Write 1 to enable DMA-SHA interrupt. (R/W)

Register 24.12. SHA_DATE_REG (0x002C)



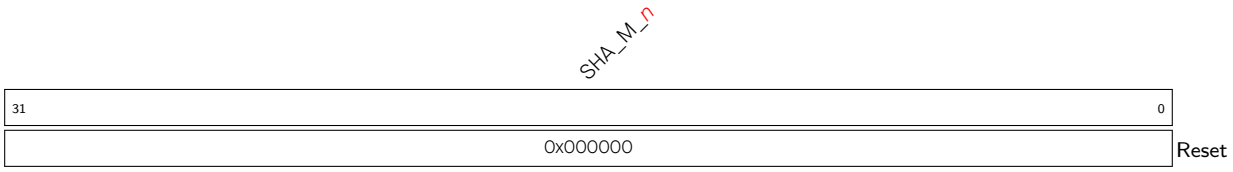
SHA_DATE Version control register. (R/W)

Register 24.13. SHA_H_*n*_REG (*n*: 0-15) (0x0040+4**n*)



SHA_H_*n* Represents the *n*th 32-bit piece of the Hash value. (R/W)

Register 24.14. SHA_M_*n*_REG (*n*: 0-31) (0x0080+4**n*)



SHA_M_*n* Represents the *n*th 32-bit piece of the message. (R/W)

Chapter 25

Digital Signature Algorithm (DSA)

25.1 Overview

The Digital Signature Algorithm (DSA) is used to verify the authenticity and integrity of a message using a cryptographic algorithm. This can be used to validate a device's identity to a server or to check the integrity of a message.

ESP32-P4 includes a Digital Signature Algorithm (DSA) module providing hardware acceleration of messages' signatures based on RSA. HMAC is used as the key derivation function (KDF) to output the DSA_KEY key using a key stored in eFuse as the input key. Subsequently, the DSA module uses DSA_KEY to decrypt the pre-encrypted parameters and calculate the signature. The whole process happens in hardware so that all the keys involved during the calculating process cannot be seen by users, guaranteeing the security of the operation.

25.2 Features

- RSA digital signatures with key length up to 4096 bits
- Encrypted private key data, only decryptable by the DSA module
- SHA-256 digest to protect private key data against tampering by an attacker

25.3 Functional Description

25.3.1 Overview

The DSA 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 (DSA_KEY) which can only be calculated by the DSA peripheral via the HMAC peripheral. The required inputs ($HMAC\_KEY$) to generate the key are only stored in eFuse and can only be accessed by the HMAC peripheral. That is to say, the DSA 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 6 *eFuse Controller* and Chapter *22 HMAC Accelerator (HMAC)*.

The input message X will be sent directly to the DSA peripheral by the software each time a signature is needed. After the RSA signature operation, the signature Z is read back by the software.

For better understanding, we define some symbols and functions here, which are only applicable to this chapter:

- $\mathbf{1}^s$: A bit string consisting of s bits with the value of "1".
- $\mathbf{[}x\mathbf{]}_s$: A bit string of s bits, in which s is an integer multiple of 8 bits. If x is a number ($x < 2^s$), it is represented in little-endian byte order in the bit string. x may be a variable such as $[Y]_{4096}$ or a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]_t$ can be right-padded with $(s-t)$ number of zeros to reach s bits in length, and finally get $[x]_s$. For example, $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$.
- $||$: A bit string concatenation operator for joining multiple-bit strings into a longer bit string.

25.3.2 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 DSA 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 25.3.3.

The DSA 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 23.3.2 *Large-Number Modular Exponentiation* in Chapter 23 *RSA Accelerator (RSA)*.

25.3.3 Software Prerequisites

If you want to use the DSA module, the software needs a series of preparations, as shown in Figure 25.3-1. 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.

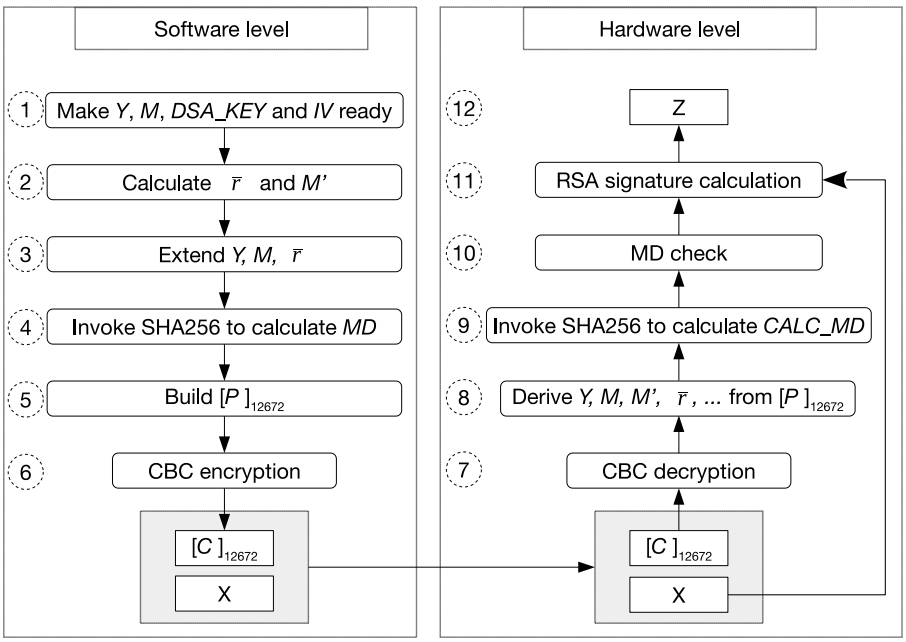


Figure 25.3-1. Software Preparations and Hardware Working Process

Note:

1. The software preparation (left side in Figure 25.3-1) is a one-time operation before any signature is calculated, while the hardware calculation (right side in Figure 25.3-1) repeats for every signature calculation.
2. Software preparation requires configuring the clock reset. For more information, please refer to Chapter 8 *Reset and Clock*.

Users need to follow the steps shown in the left part of Figure 25.3-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare operands Y and M whose lengths should meet the requirements in Section 25.3.2. Define $[L]_{32} = \frac{N}{32} - 1$ (i.e., for RSA 4096, $[L]_{32}$ == $[0x80\text{-}1]_{32}$). Prepare $[HMAC\_KEY]_{256}$ and calculate $[DSA\_KEY]_{256}$ based on DSA_KEY = HMAC-SHA256 ($[HMAC\_KEY]_{256}$, 1^{256}). Generate a random $[IV]_{128}$ which should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 20 *AES Accelerator (AES)*.
- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$ in order to get $[Y]_{4096}$, $[M]_{4096}$, and $[\overline{r}]_{4096}$, respectively. This step is only required for Y, M, and $\overline{r}$ whose length are less than 4096 bits, since their largest length are 4096 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256:
  $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[Box]_{384}$), where $[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$) and $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a $[0x0808080808080808]_{64}$ string composed of 8 bytes (0x80). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.
- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[DSA\_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext with a length of 1200 bytes. C can also be calculated as C = $[C]_{12672}$ = ($[\widehat{Y}]_{4096}||[\widehat{M}]_{4096}||[\widehat{\overline{r}}]_{4096}||[\widehat{Box}]_{384}$), where $[\widehat{Y}]_{4096}$, $[\widehat{M}]_{4096}$, $[\widehat{\overline{r}}]_{4096}$, $[\widehat{Box}]_{384}$ are the four sub-parameters

of C, and correspond to the ciphertext of $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[Box]_{384}$ respectively.

25.3.4 DSA Operation at the Hardware Level

The hardware operation is triggered each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV.

The DSA operation at the hardware level can be divided into the following three stages:

1. **Decryption: Step 7 and 8 in Figure 25.3-1**

   The decryption process is the inverse of Step 6 in Figure 25.3-1. The DSA module 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, DSA_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by the user. $[DSA\_KEY]_{256}$ is provided by the HMAC module, derived from $HMAC\_KEY$ stored in eFuse. $[DSA\_KEY]_{256}$, as well as $[HMAC\_KEY]_{256}$ are not readable by users.

   With P, the DSA module 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 25.3-1**

   The DSA module will perform two checks: MD check and padding check. The padding check is not shown in Figure 25.3-1, as it happens at the same time as the MD check.

   - MD check: The DSA module calls SHA-256 to calculate the hash value $[CALC\_MD]_{256}$ ($[CALC\_MD]_{256}$ is calculated the same way and with same parameters as $[MD]_{256}$, see step 4). Then, $[CALC\_MD]_{256}$ is compared against the MD authentication code $[MD]_{256}$ from step 4. Only when the two match does the MD check pass.
   - Padding check: The DSA module 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 DSA module 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 25.3-1**

   The DSA module 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 DSA module will calculate the signed result Z by calling RSA to perform $Z = X^Y$ mod M.

25.3.5 DSA Operation at the Software Level

The software steps below should be followed each time a digital signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV. These software steps trigger the hardware steps described in Section 25.3.4.

We assume that the software has called the HMAC peripheral and the HMAC peripheral has calculated DSA_KEY based on $HMAC\_KEY$.

1. **Prerequisites**: Prepare operands C, X, IV according to Section 25.3.3.
2. **Activate the DSA peripheral**: Write 1 to DSA_SET_START_REG.

3. **Check if DSA_KEY is ready**: Poll DSA_QUERY_BUSY_REG until the software reads 0.

   If the software does not read 0 in DSA_QUERY_BUSY_REG after approximately 1 ms, it indicates a problem with HMAC initialization. In such a case, the software can read register DSA_QUERY_KEY_WRONG_REG to get more information:

   - If the software reads 0 in DSA_QUERY_KEY_WRONG_REG, it indicates that the HMAC peripheral has not been called.
   - If the software reads any value from 1 to 15 in DSA_QUERY_KEY_WRONG_REG, it indicates that HMAC was called, but the DSA module did not successfully get the DSA_KEY value from the HMAC peripheral. This may indicate that the HMAC operation has been interrupted due to a software concurrency problem.

4. **Configure register**: Write the content in the IV block to register DSA_IV_*m*_REG (*m*: 0 ~ 3). For more information on the IV block, please refer to Chapter 20 *AES Accelerator (AES)*.

5. **Write X to memory block DSA_X_MEM**: Write X_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, to memory block DSA_X_MEM whose capacity is 96 words. Each word can store one base-b digit. The memory block uses the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address. Words in DSA_X_MEM block after the configured length of X (N bits, as described in Section 25.3.2), are ignored.

6. **Write C to corresponding memory blocks**: Write the four sub-parameters of C to corresponding memory blocks:

   - Write $\widehat{Y_i}$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_Y_MEM.
   - Write $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_M_MEM.
   - Write $\widehat{r}_i$ ($i \in \{0, 1, \ldots, 127\}$) to DSA_RB_MEM.
   - write $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) to DSA_BOX_MEM.

     The capacity of DSA_Y_MEM, DSA_M_MEM, and DSA_RB_MEM is 128 words, whereas the capacity of DSA_BOX_MEM is only 12 words. Each word can store one base-b digit. The memory blocks use the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address.

7. **Start DSA operation**: Write 1 to register DSA_SET_ME_REG.

8. **Wait for the operation to be completed**: Poll register DSA_QUERY_BUSY_REG until the software reads 0.

9. **Query check result**: Read register DSA_QUERY_CHECK_REG and conduct subsequent operations as illustrated below based on the return value:

   - If the value is 0, it indicates that both the padding check and MD check pass. Users can continue to get the signed result Z.
   - If the value is 1, it indicates that the padding check passes but MD check fails. The signed result Z is invalid. The operation will resume directly from Step 11.
   - If the value is 2, it indicates that the padding check fails but the MD check passes. Users can continue to get the signed result Z. But please note that the data does not comply with the aforementioned PKCS#7 padding format, which may not be what you want.

- If the value is 3, it indicates that both the padding check and MD check fail. In this case, some fatal errors have occurred and the signed result Z is invalid. The operation will resume directly from Step 11.

10. **Read the signed result**: Read the signed result Z_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, from memory block DSA_Z_MEM. The memory block stores Z in little-endian byte order.
11. **Exit the operation**: Write 1 to DSA_SET_FINISH_REG, and then poll DSA_QUERY_BUSY_REG until the software reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

25.4 Memory Summary

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

Name	Description	Size (byte)	Starting Address	Ending Address	Access
DSA_Y_MEM	Memory block Y	512	0x0000	0x01FF	WO
DSA_M_MEM	Memory block M	512	0x0200	0x03FF	WO
DSA_RB_MEM	Memory block $\overline{r}$	512	0x0400	0x05FF	WO
DSA_BOX_MEM	Memory block Box	48	0x0600	0x062F	WO
DSA_X_MEM	Memory block X	512	0x0800	0x09FF	WO
DSA_Z_MEM	Memory block Z	512	0x0A00	0x0BFF	RO

25.5 Register Summary

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

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

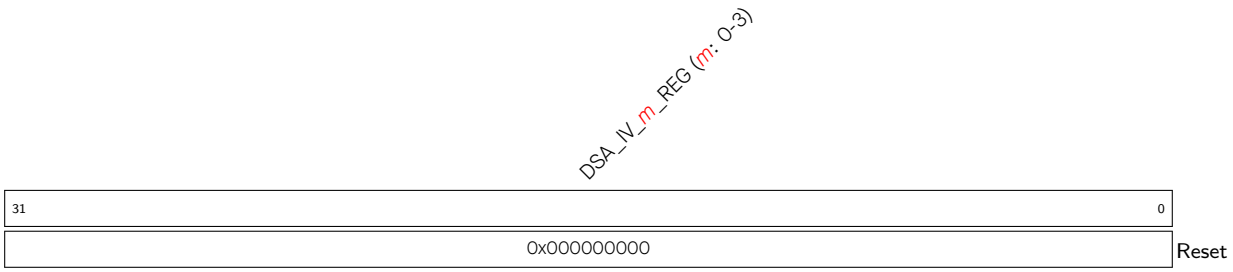
Name	Description	Address	Access
Configuration Registers			
DSA_IV_0_REG	IV block data	0x0630	WO
DSA_IV_1_REG	IV block data	0x0634	WO
DSA_IV_2_REG	IV block data	0x0638	WO
DSA_IV_3_REG	IV block data	0x063C	WO
Status/Control Registers			
DSA_SET_START_REG	Activates the DSA module	0x0E00	WO
DSA_SET_ME_REG	Starts DSA operation	0x0E04	WO
DSA_SET_FINISH_REG	Ends DSA operation	0x0E08	WO
DSA_QUERY_BUSY_REG	Status of the DSA module	0x0E0C	RO
DSA_QUERY_KEY_WRONG_REG	Checks the reason why DSA_KEY is not ready	0x0E10	RO
DSA_QUERY_CHECK_REG	Queries DSA check result	0x0E14	RO
Version control register			
DSA_DATE_REG	Version control register	0x0E20	W/R

25.6 Registers

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

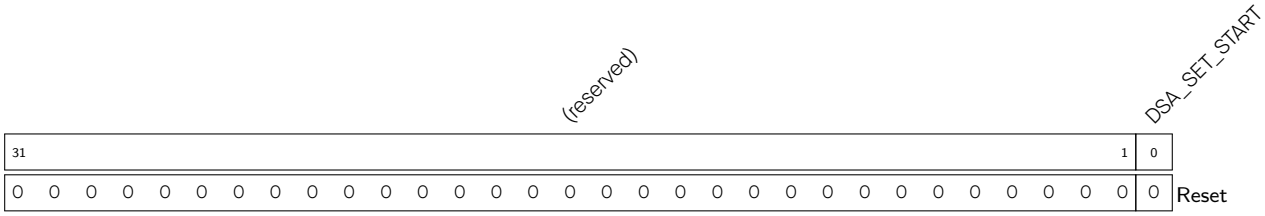
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 25.1. DSA_IV_*m*_REG (*m*: 0-3) (0x0630+4m*)**



DSA_IV_*m*_REG (*m*: 0-3) Writes IV block data. (WO)

Register 25.2. DSA_SET_START_REG (0x0E00)



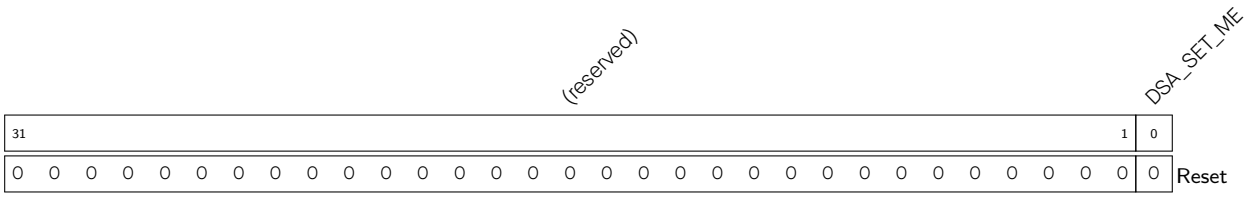
DSA_SET_START Configures whether to activate the DSA peripheral.

0: No effect

1: Activate the DSA peripheral

(WO)

Register 25.3. DSA_SET_ME_REG (0x0E04)



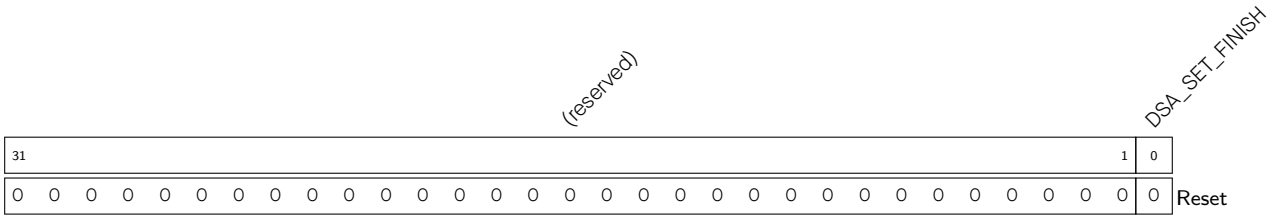
DSA_SET_ME Configures whether to start the DSA operation.

0: No effect

1: Start the DSA operation

(WO)

Register 25.4. DSA_SET_FINISH_REG (0x0E08)



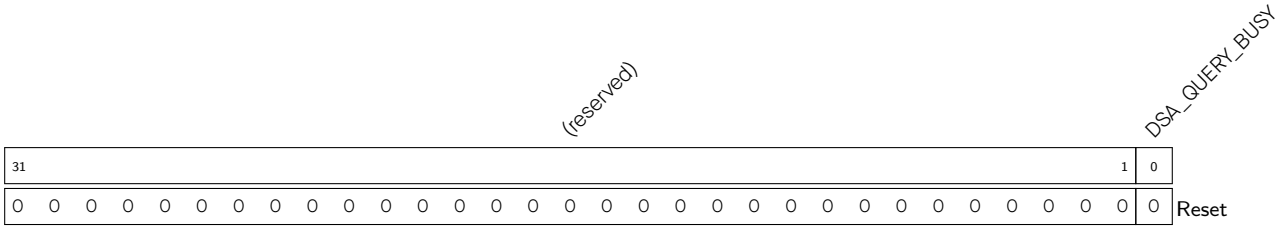
DSA_SET_FINISH Configures whether to end the DSA operation.

0: No effect

1: End the DSA operation

(WO)

Register 25.5. DSA_QUERY_BUSY_REG (0x0E0C)



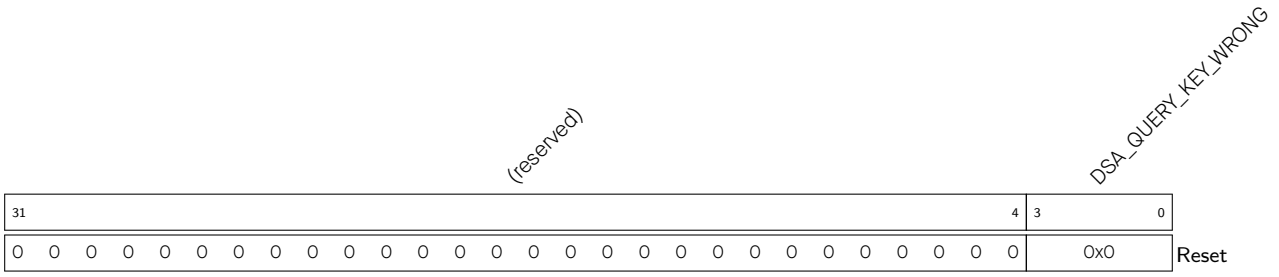
DSA_QUERY_BUSY Represents whether the DSA module is idle.

0: The DSA module is idle.

1: The DSA module is busy.

(RO)

Register 25.6. DSA_QUERY_KEY_WRONG_REG (0x0E10)



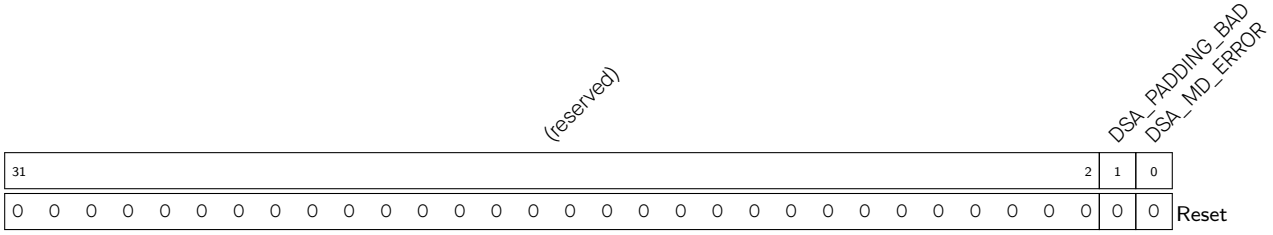
DSA_QUERY_KEY_WRONG Represents the specific problem with HMAC initialization.

0: HMAC is not called.

1-15: HMAC was activated, but the DSA peripheral did not successfully receive the DSA_KEY from the HMAC peripheral.

(RO)

Register 25.7. DSA_QUERY_CHECK_REG (0x0E14)



DSA_PADDING_BAD Represents whether the padding check passes.

0: The padding check passes.

1: The padding check fails.

(RO)

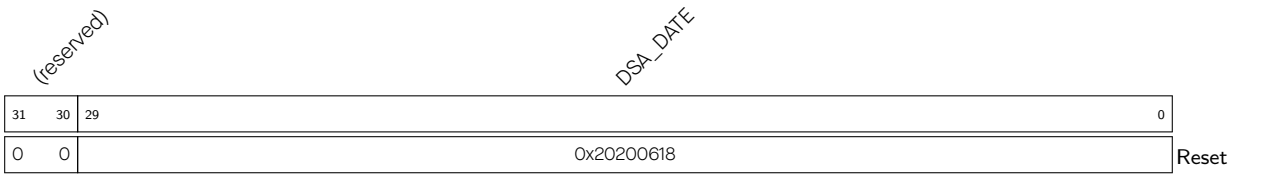
DSA_MD_ERROR Represents whether the MD check passes.

0: The MD check passes.

1: The MD check fails.

(RO)

Register 25.8. DSA_DATE_REG (0x0E20)



DSA_DATE Version control register. (R/W)

Chapter 26

Elliptic Curve Digital Signature Algorithm (ECDSA)

26.1 Introduction

In cryptography, the Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic-curve cryptography.

ESP32-P4's ECDSA accelerator 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 the ECDSA accelerator, users can be confident that their data is being protected without sacrificing performance.

26.2 Features

ESP32-P4's ECDSA accelerator supports:

- Digital signature generation and verification
- Two different elliptic curves, namely P-192 and P-256, defined in FIPS 186-3 Spec
- Two hash algorithms for message hash in the ECDSA operation, namely SHA-224 and SHA-256, defined in FIPS PUB 180-4 Spec
- Dynamic access permission in different operation statuses to ensure information security

26.3 ECDSA Basics

26.3.1 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 curve. 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 followings:

- 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 base point G on the curve, which can be used to generate public keys.

  - The order n of the curve, 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 and are critical for the secure use of ECDSA.

26.3.2 Key Generation

The process of key generation in ECDSA is described below:

1. Select an elliptic curve domain by defining:
   - the prime modulus p
   - the coefficients a and b
   - the base point G and its order n
2. Generate a private key:
   - A private key d is a randomly chosen integer between 1 and n-1. It is essential to use a secure *Random Number Generator (RNG)* to ensure that the private key is truly random and cannot be predicted or reproduced.
   - The private key is used for signature generation. This key must be kept secret and secure.
   - Once a private key is chosen, it should be burned into the eFuse in ESP32-P4. (Please refer to Chapter 6 *eFuse Controller*).
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.

26.3.3 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 curve 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.

26.3.4 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 curve 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.

If the signature is valid,

26.4 Functional Description

This section describes the details of ESP32-P4's ECDSA accelerator.

26.4.1 ECDSA Working Modes

The ECDSA accelerator integrated in the ESP32-P4 has three working modes, which are Signature Generation, Signature Verification and Public Key Export modes.

Users can choose the working mode for the ECDSA accelerator by configuring the ECDSA_WORK_MODE according to Table 26.4-1 below.

Table 26.4-1. ECDSA Working Mode

ECDSA_WORK_MODE	Working Mode
0	Signature Verification
1	Signature Generation
2	Public Key Export

Users can select the elliptic curves used by configuring the ECDSA_ECDSA_CURVE according to Table 26.4-2.

Table 26.4-2. ECDSA Elliptic Curves Selection

ECDSA_ECC_CURVE	Elliptic Curve
0	P-192
1	P-256

Users can select the SHA algorithms for message hash by configuring the ECDSA_SHA_MODE according to Table 26.4-3.

Table 26.4-3. ECDSA SHA Algorithm

ECDSA_SHA_MODE	SHA Algorithm
1	SHA-224
2	SHA-256
Others	Invalid

Additionally, users can check the working status of the ECDSA accelerator by inquiring the ECDSA_STATE_REG register and comparing the return value against the Table 26.4-4 below.

Table 26.4-4. ECDSA Working Status

ECDSA_STATE_REG	Status	Description
0	IDLE	Idle or completed operation. Corresponding to IDLE Stage.
1	LOAD	Waiting for users to load information into ECDSA. Corresponding to LOAD Stage.
2	GAIN	Waiting for users to gain information from ECDSA. Corresponding to GAIN Stage.
3	BUSY	In the middle of a hardware operation. Corresponding to PREP, PROC & POST Stage.

26.4.2 Data and Data Block

ESP32-P4's ECDSA accelerator operates on data of 256 or 512 bits. This data ($D[255:0]$) can be divided into 32-bit data blocks.

Take 256-bit long data as an example, $D[n][31:0](n = 0, 1, \cdots, 7)$. Data blocks with the smaller serial number correspond to the lower binary bits. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

26.4.2.1 Writing Data

Writing data means writing data to an ECDSA memory block and using this data as the input to the ECDSA algorithm. To be specific, writing data to an ECDSA memory block means writing $D[n][31:0]$ to the "starting address of this ECDSA memory block + $4 \times n$" For a 256-bit long data example:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- $\cdots$
- write $D[7]$ to "starting address + 28"

Note:
When the data size of 192 bits is used, you need to append 0 after 192 bits of data and write 256 bits of data.

26.4.2.2 Reading Data

Reading data means reading data from the starting address of an ECDSA memory block and using this data as the output from the ECDSA algorithm. To be specific, reading data from an ECDSA memory block means reading $D[n][31:0]$ from the "starting address of this ECDSA memory block + $4 \times n$". For a 256-bit long data example:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- $\cdots$
- read $D[7]$ from "starting address + 28"

Note:
When the data size of 192 bits is used, only use the low 192 bits (6 blocks) of data.

26.4.2.3 Padding the Message

The SHA accelerator can only process message blocks of 512 bits. Thus, all the messages should be padded to a multiple of 512 bits before the hash operation.

Suppose that the length of the message M is L_M bits. Then M shall be padded as introduced below:

1. First, append the bit "1" to the end of the message;
2. Second, append L_A bits of zeros, where L_A is the smallest, non-negative solution to the equation $L_M + 1 + L_A \equiv 448 \ mod \ 512$;
3. Last, append the 64-bit block of value equal to the number L_M expressed using a binary representation.

For more details, please refer to FIPS PUB 180-4 Spec > Section "Padding the Message".

26.4.2.4 Parsing the Message

The message and its padding must be parsed into N 512-bit message blocks: $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$.

Note:

1. For more details about "parsing the message", please refer to FIPS PUB 180-4 Spec > Section "Parsing the Message".
2. For more information on "message block", please refer to FIPS PUB 180-4 Spec > Section "Glossary of Terms and Acronyms".

26.4.3 Security Features

To ensure the security of the ECDSA operation process, the ECDSA accelerator implements a variety of security functions.

26.4.3.1 Dynamic Access Permission

ESP32-P4's ECDSA accelerator has implemented a dynamic access permission mechanism to prevent any possibility of key theft by tampering with the configuration or accessing the data during the operation.

By implementing this dynamic access permission mechanism, the accesses for ECDSA registers are designed to vary in different statues. For example, ECDSA_CONF_REG is only available for reading and writing when the accelerator is in the IDLE status. In this way, the configuration information is protected from reading or writing when the accelerator is in other statues, such as LOAD, GAIN and BUSY. For details about all ECDSA working statues, please refer to Table 26.4-4.

For detailed information on the dynamic access permission of each ECDSA register, please refer to Section Register Summary.

26.4.3.2 Hardware Occupation

During the ECDSA operation, the following hardware modules will be occupied by ESP32-P4's ECDSA accelerator:

- SHA Accelerator
- RSA Accelerator
- ECC Accelerator

Among them, the SHA accelerator will be released when the ECDSA_SHA_RELEASE_INT is triggered. While, the RSA accelerator and the ECC accelerator will be occupied during the whole ECDSA operation.

Note:

Hardware occupation is a mechanism to protect multiplexed modules and storage space. When a module is hardware occupied, the user will fail to:

- read or write data to the module's registers or memories.
- disable the module clock.

- reset the module.

At the end of the hardware occupation, the occupied module will be automatically reset. In addition, when the user performs a software reset to the master module, all the occupied modules will be reset at the same time.

26.5 Programming Procedures

26.5.1 ECDSA Process

The overall ECDSA process consists of the following six stages.

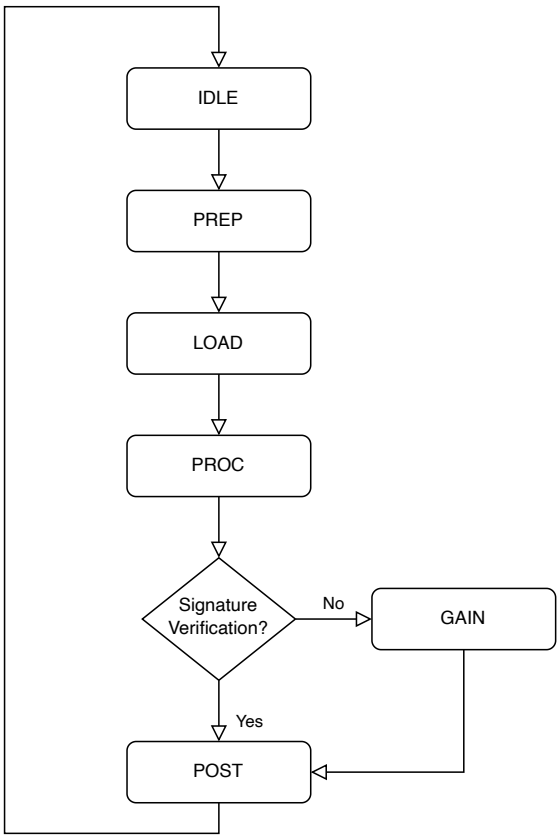


Figure 26.5-1. ECDSA Process

The detailed programming procedures of each stage are described in the following sections.

26.5.1.1 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. To correctly configure the key value in eFuse, user need to write the key value in KEY*n* (n = 0 ~ 5), and set the corresponding EFUSE_KEY_PURPOSE_*n* as ECDSA_KEY. Please refer to Chapter 6 *eFuse Controller* for more detailed configuration steps.

2. Configure the configuration register, including the following fields:
   (a) ECDSA_WORK_MODE: Choose the working mode of the ECDSA accelerator.
   (b) ECDSA_ECC_CURVE: Choose the elliptic curve of the ECDSA accelerator.
   (c) ECDSA_SOFTWARE_SET_Z: Choose to use direct input z.
   (d) ECDSA_DETERMINISTIC_K: Choose to use deterministic k.
   (e) ECDSA_DETERMINISTIC_LOOP: Choose the loop number of deterministic k.
3. Configure the register field ECDSA_START to enter the PREP stage.

Note:
For more details about Deterministic ECDSA, please refer to RFC6979.

26.5.1.2 PREP Stage

In the PREP stage, the ECDSA_STATE_REG is BUSY, and the ECDSA accelerator performs preparation.

1. Wait till the PREP stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA accelerator will automatically enter the LOAD stage.

26.5.1.3 LOAD Stage

In the LOAD stage:

1. Provide input z into the ECDSA accelerator using one of the following options:
   - Direct input z: write z to ECDSA_Z_MEM.
   - ECDSA-SHA interface: generate z from the message. For details, please refer to Section 26.5.1.7.
2. According to the selected ECDSA_WORK_MODE, configure as follows:
   - Signature Verification:
     (a) write signature (r, s) to ECDSA_R_MEM and ECDSA_S_MEM.
     (b) write public key (Qx, Qy) to ECDSA_QX_MEM and ECDSA_QY_MEM.
   - Signature Generation:
     (a) no other configuration is required.
   - Public Key Export:
     (a) no other configuration is required.
3. Write 1 to ECDSA_LOAD_DONE, indicating the configuration is done. Then the accelerator will automatically enter the PROC stage.

26.5.1.4 PROC Stage

In the PROC stage, the ECDSA_STATE_REG is BUSY, and the ECDSA accelerator performs ECDSA operation based on the selected working mode.

1. Wait till the PROC stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA accelerator will automatically enter the either the GAIN stage or the POST stage depending on the selected working mode:
   - Signature generation: GAIN stage
   - Signature verification: POST stage

26.5.1.5 GAIN Stage

When the Signature Generation mode or the Public Key Export mode is selected, the ECDSA accelerator enters the GAIN stage after PROC stage:

1. Read data from the ECDSA memory:
   - read signature (r, s) from ECDSA_R_MEM and ECDSA_S_MEM only in the Signature Generation mode.
   - read public key (Qx, Qy) from ECDSA_QX_MEM and ECDSA_QY_MEM both in the Signature Generation mode and the Public Key Export mode.
2. Write 1 to ECDSA_GAIN_DONE, indicating the GAIN stage is done. Then the accelerator will automatically enter the POST stage.

26.5.1.6 POST Stage

In the POST stage, the ECDSA_STATE_REG is BUSY, and the ECDSA accelerator performs some wrap-up work of ECDSA operation.

1. Wait till the POST stage to end by polling ECDSA_BUSY until it is not BUSY. Then the ECDSA accelerator will automatically return to the IDLE stage.

26.5.1.7 ECDSA SHA Interface

ESP32-P4's ECDSA accelerator can automatically executes hash operation and generates z based on a direct input message.

For message hash, ECDSAS accelerator supports SHA algorithms SHA-224 (only valid when P-192 is selected as the elliptic curve) and SHA-256.

To use the ECDSA SHA interface, complete the following steps:

1. Pad the message by following the steps described in Section 26.4.2.3.
2. Parse the message and its padding into message blocks. See details in Section 26.4.2.4.
3. Process the current message block.
   - Write the current message block into ECDSA_MEM_M.
4. Start the ECDSA SHA interface[1].
   - If this is the first time to execute this step, write 1 to ECDSA_SHA_START to start the ECDSA SHA interface;
   - If this is not the first time to execute this step[2], write 1 to ECDSA_SHA_CONTINUE to continue the operation.

5. Check the progress of the current message block processing by polling.
    - Poll register ECDSA_SHA_BUSY until it's IDLE, indicating the interface has completed the operation for the current message block.
6. Process the next message block:
    - If yes, go back to Step 3.
    - If no more message block, exit.

Note:

1. In this step, the software can also write the next message block (to be processed) in register ECDSA_MEM_M, if any, while the interface starts SHA operation, to save time.
2. You are resuming the ECDSA SHA interface with the previously paused operation.

26.5.2 Clocks and Resets

ESP32-P4's ECDSA accelerator has one clock module and one reset module. It is activated by setting the HP_SYS_CLKRST_REG_CRYPTO_ECDSA_CLK_EN bit in the HP_SYS_CLKRST_PERI_CLK_CTRL25_REG register and clearing the HP_SYS_CLKRST_REG_RST_EN_ECDSA bit in the HP_SYS_CLKRST_HP_RST_EN2_REG register. For details on how to configure the ECDSA clock and reset, please refer to Chapter 8 *Reset and Clock*.

26.5.3 Interrupts

ESP32-P4's ECDSA accelerator can generate one interrupt signal ECDSA_INTR and send it to Interrupt Matrix.

The ECDSA accelerator has two interrupt sources that can generate the ECDSA_INTR interrupt signal:

- ECDSA_CALC_DONE_INT: triggered on the completion of an ECC operation. This interrupt source is configured by the following registers:
    - ECDSA_CALC_DONE_INT_RAW: stores the raw interrupt status of ECDSA_CALC_DONE_INT.
    - ECDSA_CALC_DONE_INT_ST: indicates the status of the ECDSA_CALC_DONE_INT interrupt. This field is generated by enabling/disabling the ECDSA_CALC_DONE_INT_RAW field.
    - ECDSA_CALC_DONE_INT_ENA: enables/disables the ECDSA_CALC_DONE_INT interrupt.
    - ECDSA_CALC_DONE_INT_CLR: set this bit to clear the ECDSA_CALC_DONE_INT interrupt status. By setting this bit to 1, fields ECDSA_CALC_DONE_INT_RAW and ECDSA_CALC_DONE_INT_ST will be cleared.
- ECDSA_SHA_RELEASE_INT: triggered when SHA is released.This interrupt source is configured by the following registers:
    - ECDSA_SHA_RELEASE_INT_RAW: stores the raw interrupt status of ECDSA_SHA_RELEASE_INT.
    - ECDSA_SHA_RELEASE_INT_ST: indicates the status of the ECDSA_SHA_RELEASE_INT interrupt. This field is generated by enabling/disabling the ECDSA_SHA_RELEASE_INT_RAW field via ECDSA_SHA_RELEASE_INT_ENA.

- ECDSA_SHA_RELEASE_INT_ENA: enables/disables the ECDSA_SHA_RELEASE_INT interrupt.
- ECDSA_SHA_RELEASE_INT_CLR: set this bit to clear the ECDSA_SHA_RELEASE_INT interrupt status. By setting this bit to 1, fields ECDSA_SHA_RELEASE_INT_RAW and ECDSA_SHA_RELEASE_INT_ST will be cleared.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

26.6 Memory Blocks

ECDSA's memory blocks store input data and output data of the ECDSA operation.

Table 26.6-1. ECDSA Memory Blocks

Memory	Size (byte)	Starting Address*	Ending Address *	Access
ECDSA_MEM_M	64	0x280	0x2BF	R/W
ECDSA_MEM_R	32	0xA00	0xA1F	R/W
ECDSA_MEM_S	32	0xA20	0xA3F	R/W
ECDSA_MEM_Z	32	0xA40	0xA5F	R/W
ECDSA_MEM_Qx	32	0xA60	0xA7F	R/W
ECDSA_MEM_Qy	32	0xA80	0xA9F	R/W

* Address offset related to the ECDSA accelerator base address is provided in Table 5.3-2 in Chapter 5 *System and Memory*.

26.7 Register Summary

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

To enhance security, ECDSA registers have different read and write permissions in different operating statuses. The following is the abbreviation and corresponding relationship of each state:

- PI: IDLE status, corresponding to IDLE Stage.
- PL: LOAD status, corresponding to LOAD Stage.
- PG: GAIN status, corresponding to GAIN Stage.
- PB: BUSY status, corresponding to PREP, PROC and POST Stage.

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

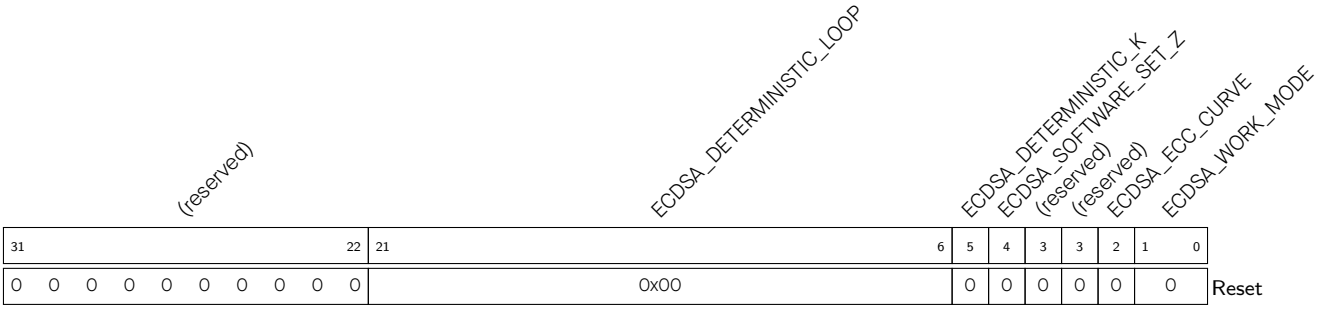
Name	Description	Address	Access			
			PI	PL	PG	PB
Data Memory	See Table 26.6-1.					
Configuration Registers						
ECDSA_CONF_REG	ECDSA configuration register	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA start register	0x001C	WT			N/A
Clock and Reset Register						
ECDSA_CLK_REG	ECDSA clock gate register	0x0008	R/W	N/A		
Interrupt Registers						
ECDSA_INT_RAW_REG	ECDSA interrupt raw register	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA interrupt status register	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA interrupt enable register	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA interrupt clear register	0x0018	WT			
Status Registers						
ECDSA_STATE_REG	ECDSA status register	0x0020	RO			
Result Register						
ECDSA_RESULT_REG	ECDSA result register	0x0024	RO/SS			N/A
SHA Registers						
ECDSA_SHA_MODE_REG	ECDSA controlling SHA register (Hash algrithm)	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA controlling SHA register (operation)	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA controlling SHA register (operation)	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA controlling SHA status register	0x0218	N/A	RO	N/A	
Version Register						
ECDSA_DATE_REG	Version control register	0x00FC	R/W	N/A		

26.8 Registers

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

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

Register 26.1. ECDSA_CONF_REG (0x0004)



ECDSA_WORK_MODE Configures the working mode of ECDSA accelerator.

0: Signature Verification Mode

1: Signature Generation Mode

2: Public Key Export Mode

3: Invalid

(R/W)

ECDSA_ECC_CURVE Configures the elliptic curve used.

0: P-192

1: P-256

(R/W)

ECDSA_SOFTWARE_SET_Z Configures how the parameter z is set.

0: generated from SHA result

1: written by software

(R/W)

ECDSA_DETERMINISTIC_K Configures how the parameter k is set.

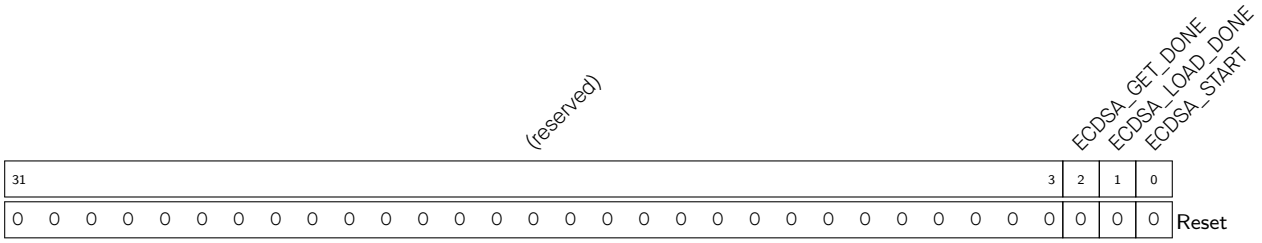
0: automatically generated by TRNG

1: generated by the deterministic derivation algorithm

(R/W)

ECDSA_DETERMINISTIC_LOOP Configures the (loop number - 1) value used in the deterministic derivation algorithm to derive k. (R/W)

Register 26.2. ECDSA_START_REG (0x001C)



ECDSA_START Configures whether to start the ECDSA operation. This bit will be self-cleared after configuration.

0: no effect

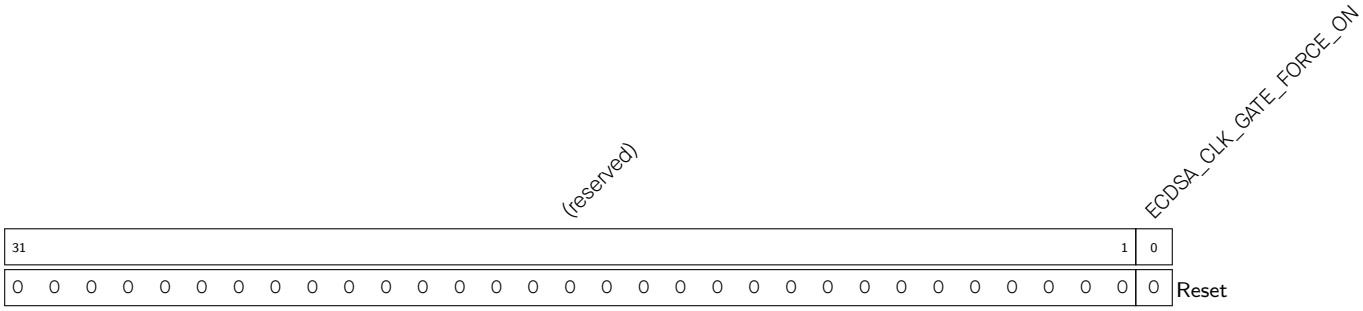
1: start the ECDSA operation

(WT)

ECDSA_LOAD_DONE Write 1 to generate a signal indicating the ECDSA accelerator's LOAD operation is done. This bit will be self-cleared after configuration. (WT)

ECDSA_GAIN_DONE Write 1 to generate a signal indicating the ECDSA accelerator's GAIN operation is done. This bit will be self-cleared after configuration. (WT)

Register 26.3. ECDSA_CLK_REG (0x0008)



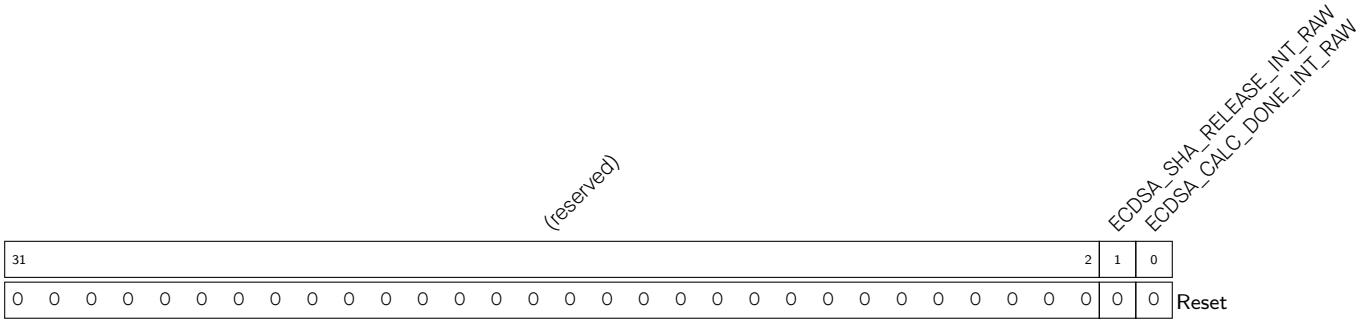
ECDSA_CLK_GATE_FORCE_ON Configures whether to force on ECC memory clock gate.

0: no effect

1: force on

(R/W)

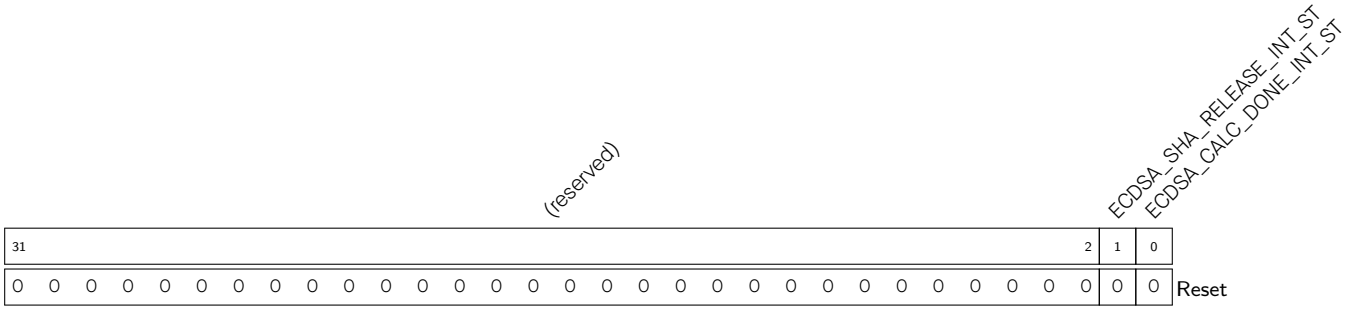
Register 26.4. ECDSA_INT_RAW_REG (0x000C)



ECDSA_CALC_DONE_INT_RAW The raw interrupt status of the ECDSA_CALC_DONE_INT interrupt. (RO/WTC/SS)

ECDSA_SHA_RELEASE_INT_RAW The raw interrupt status of the ECDSA_SHA_REKEASE_INT interrupt. (RO/WTC/SS)

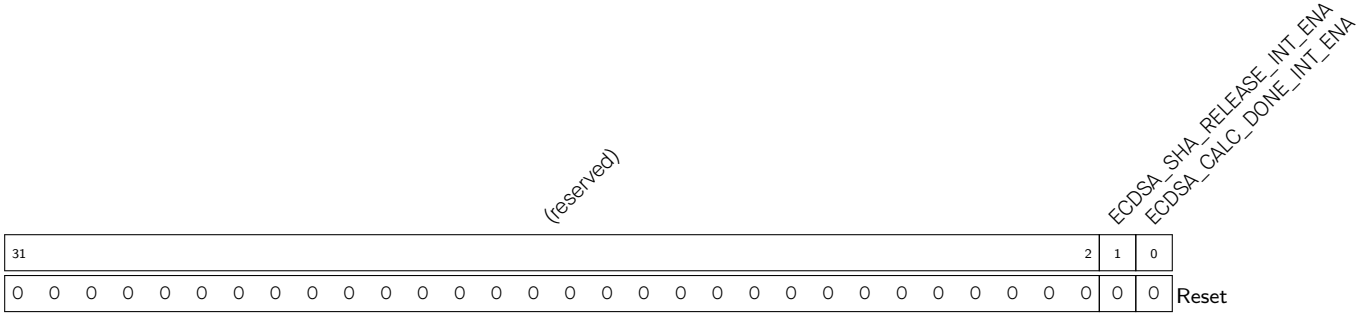
Register 26.5. ECDSA_INT_ST_REG (0x0010)



ECDSA_CALC_DONE_INT_ST The masked interrupt status of the ECDSA_CALC_DONE_INT interrupt. (RO)

ECDSA_SHA_RELEASE_INT_ST The masked interrupt status of the ECDSA_SHA_REKEASE_INT interrupt. (RO)

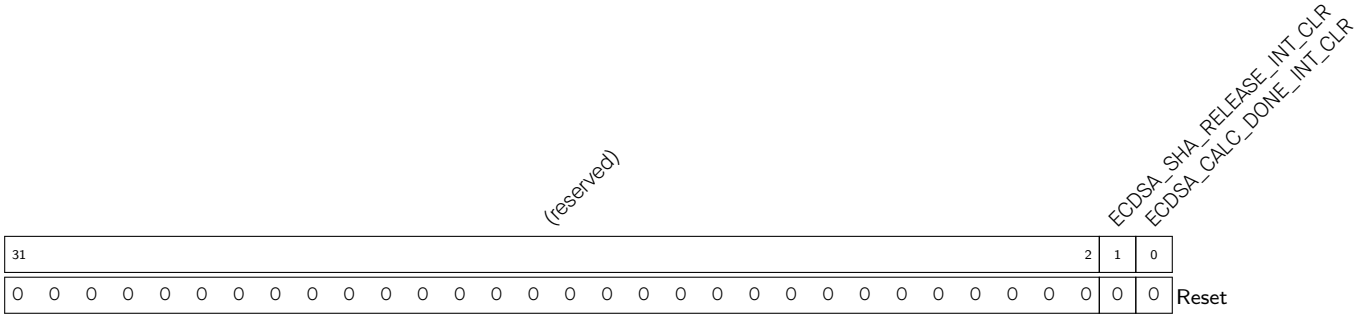
Register 26.6. ECDSA_INT_ENA_REG (0x0014)



ECDSA_CALC_DONE_INT_ENA Write 1 to enable the ECDSA_CALC_DONE_INT interrupt. (R/W)

ECDSA_SHA_RELEASE_INT_ENA Write 1 to enable the ECDSA_SHA_REKEASE_INT interrupt. (R/W)

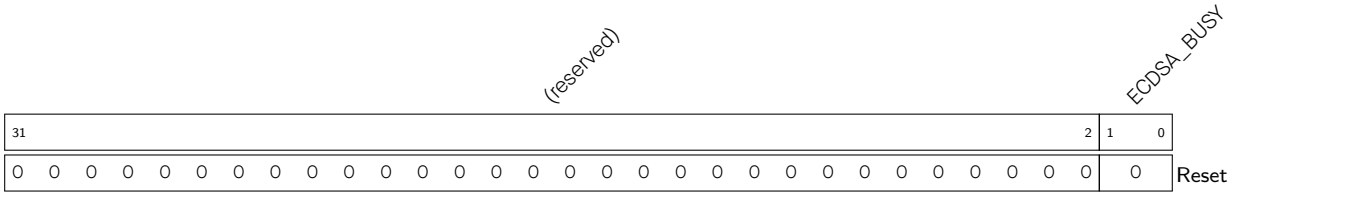
Register 26.7. ECDSA_INT_CLR_REG (0x0018)



ECDSA_CALC_DONE_INT_CLR Write 1 to clear the ECDSA_CALC_DONE_INT interrupt. (WT)

ECDSA_SHA_RELEASE_INT_CLR Write 1 to clear the ECDSA_SHA_REKEASE_INT interrupt. (WT)

Register 26.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY Represents the working status of the ECDSA accelerator.

0: IDLE

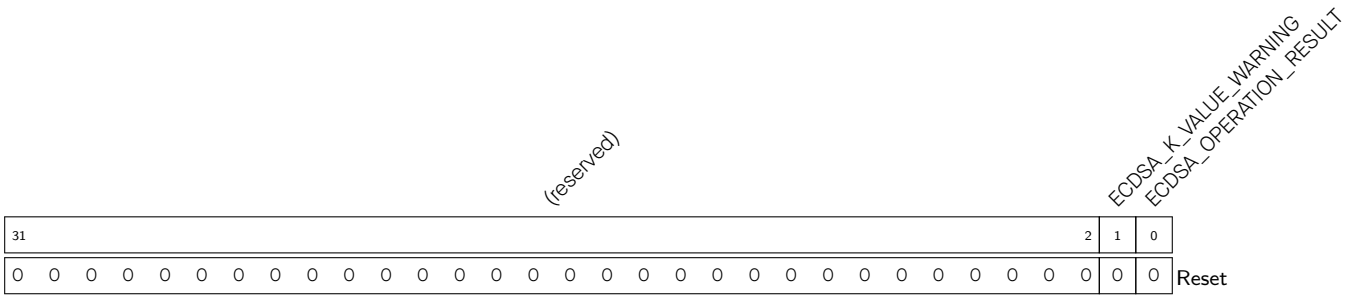
1: LOAD

2: GET

3: BUSY

(RO)

Register 26.9. ECDSA_RESULT_REG (0x0024)



ECDSA_OPERATION_RESULT Indicates if the ECDSA operation is successful.

0: not successful

1: successful

Only valid when the ECDSA operation is done. (RO/SS)

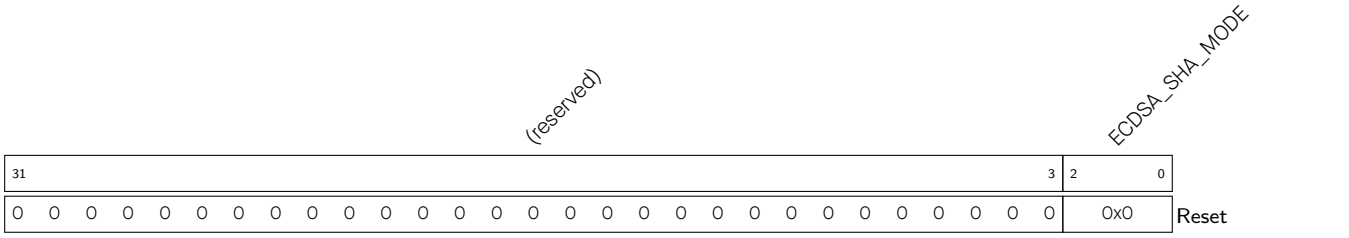
ECDSA_K_VALUE_WARNING Indicates if the k value is greater than the curve order.

0: k value is not greater than the curve order. In this case, the k value is the set k value.

1: k value is greater than than the curve order. In this case, the k value is the set k mod n.

Only valid when the ECDSA operation is done. (RO/SS)

Register 26.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE Configures SHA algorithms for message hash.

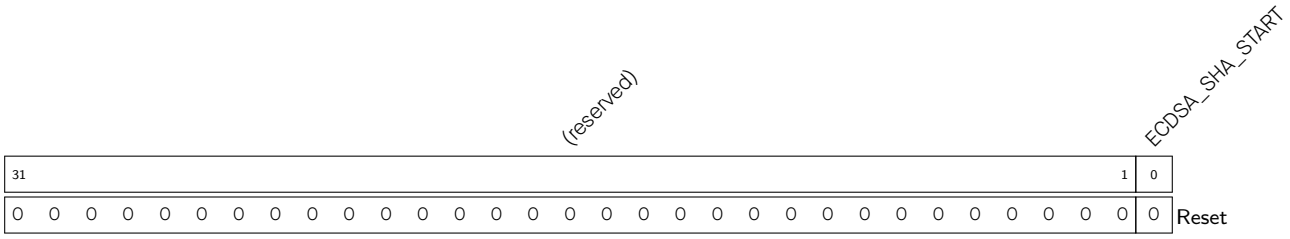
1: SHA-224

2: SHA-256

Others: invalid

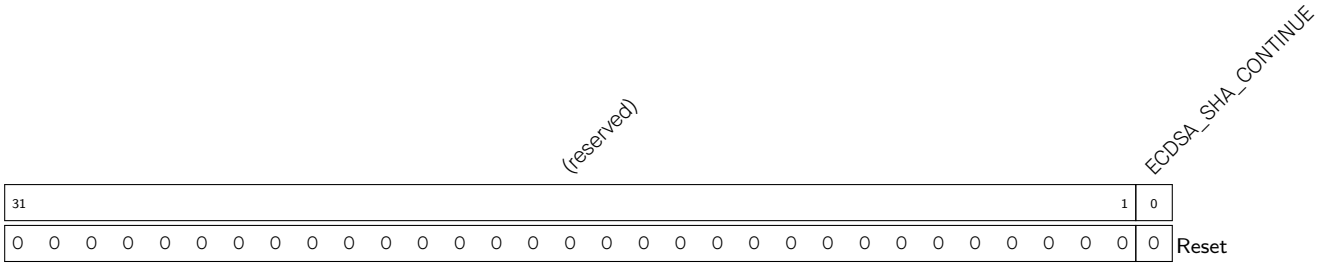
(R/W)

Register 26.11. ECDSA_SHA_START_REG (0x0210)



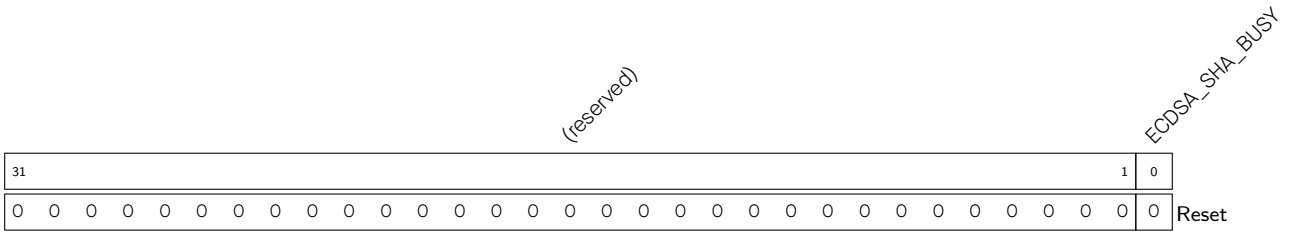
ECDSA_SHA_START Write 1 to start the first SHA operation in the ECDSA process. This bit will be self-cleared after configuration. (WT)

Register 26.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE Write 1 to start the latter SHA operation in the ECDSA process. This bit will be self-cleared after configuration. (WT)

Register 26.13. ECDSA_SHA_BUSY_REG (0x0218)



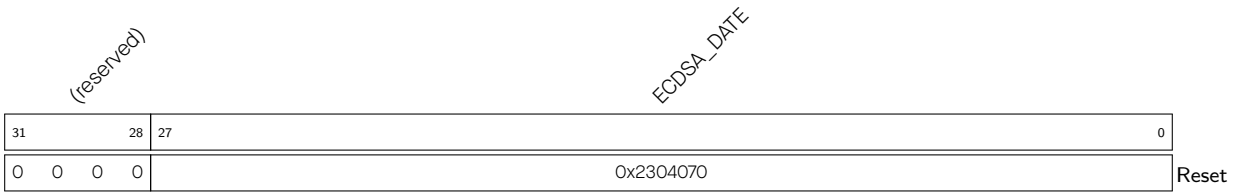
ECDSA_SHA_BUSY Represents the working status of the SHA accelerator in the ECDSA process.

0: IDLE

1: BUSY

(RO)

Register 26.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE The ECDSA version control register. (R/W)

Chapter 27

Random Number Generator (RNG)

27.1 Introduction

The ESP32-P4 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

27.2 Feature List

The random number generator in ESP32-P4 generates true random numbers, which means random numbers generated from a physical process, rather than by means of an algorithm. No number generated within the specified range is more or less likely to appear than any other number.

27.3 Functional Description

Every 32-bit value that the system reads from the LPSYSREG_RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the **thermal noise** in the system and the **asynchronous clock mismatch**.

- **Thermal noise** comes from the SAR ADC. Whenever the SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.
- RC_FAST_CLK is an **asynchronous clock** source and it increases the RNG entropy by introducing circuit metastability. Please refer to the Note about the RC_FAST_CLK.

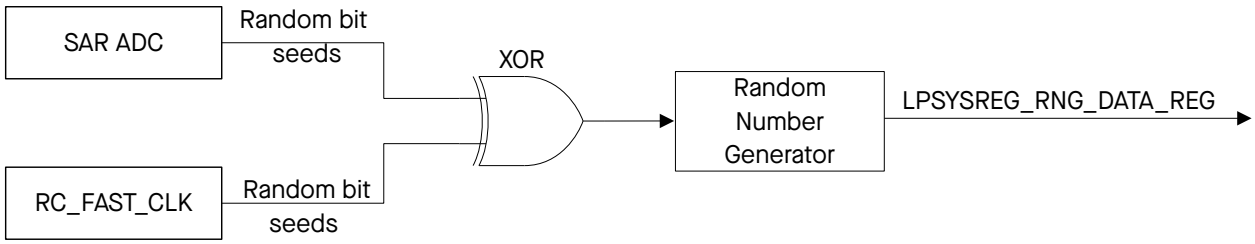


Figure 27.3-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 1-bit entropy in one clock cycle of RC_FAST_CLK, which is generated from an internal RC oscillator (see Chapter 8 *Reset and Clock* for details). Thus, it is advisable to read the LPSYSREG_RNG_DATA_REG register at a maximum rate of 1 MHz to obtain the maximum entropy.

27.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC or RC_FAST_CLK is enabled. Otherwise, pseudo-random numbers will be returned.

- For SAR ADC, please refer to Chapter 45 *ADC Controller (ADC)*.
- RC_FAST_CLK is enabled by setting the PMU_HP_SLEEP_XPD_FOSC_CLK bit in the PMU_HP_SLEEP_LP_CK_POWER_REG register. Please refer to the Note about the RC_FAST_CLK.
- RC_FAST_CLK is gated by setting the LPPERI_CK_EN_RNG bit in the LPPERI_CLK_EN_REG register.

Note:
Enabling RC_FAST_CLK increases the RNG entropy. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

When using the random number generator, read the LPSYSREG_RNG_DATA_REG register multiple times until sufficient random numbers have been generated. Ensure the rate at which the register is read does not exceed the frequencies described in section 27.3 above.

27.5 Register Summary

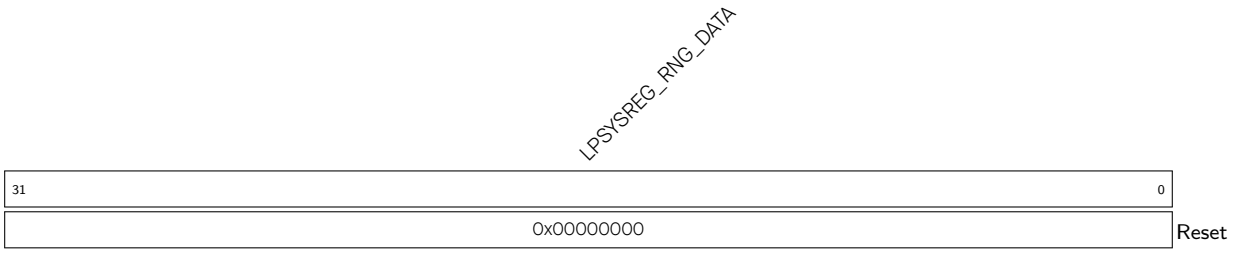
The abbreviations given in Column **Access** are explained in Section *Access Types for Registers*.

Name	Description	Address	Access
LPSYSREG_RNG_DATA_REG	Random source data	0x5011_01A4	RO
LPPERI_CLK_EN_REG	Integrated clock gating enable signal of RC_FAST_CLK	0x5012_0000	R/W

27.6 Registers

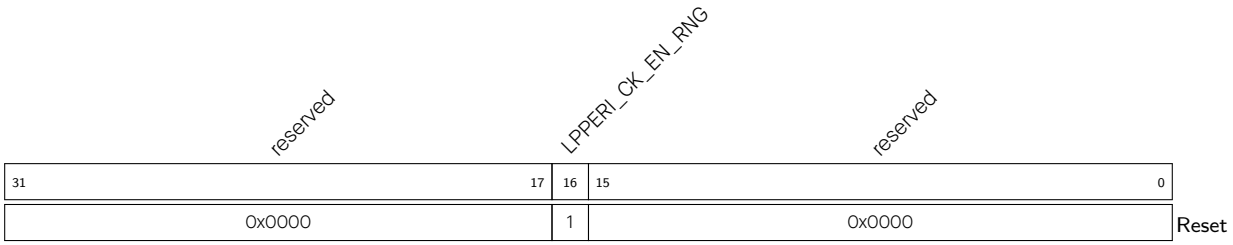
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 27.1. LPSYSREG_RNG_DATA_REG (0x5011_01A4)



LPSYSREG_RNG_DATA Random number source. (RO)

Register 27.2. LPPERI_CLK_EN_REG (0x5012_0000)



LPPERI_CK_EN_RNG Clock gating control signal of RC_FAST_CLK. (R/W)

Part VI

Image and Voice Processing

Focusing on multimedia processing, this part classifies components related to image and voice processing, including camera controllers, video encoders, image signal processors, and voice activity detection.

Chapter 28

JPEG Codec

ESP32-P4's JPEG codec is an image codec, which is based on the JPEG baseline standard, for compressing (encoding) and decompressing (decoding) images to reduce the bandwidth required to transmit images or the space required to store images, making it possible to process large-resolution images. The image encoding process mainly includes discrete cosine transform (DCT), zigzag scanning, quantization, and Huffman coding. The image decoding process, which is the inverse operation of the encoding process, mainly includes Huffman decoding, inverse quantization, inverse zigzag scanning, and inverse discrete cosine transform (IDCT).

28.1 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 28.1-1 shows the schematic diagram of JPEG bitstream format. In this diagram, the dashed lines mark the structural composition of the corresponding parts of the bitstream, and the orange blocks indicate the marker codes.

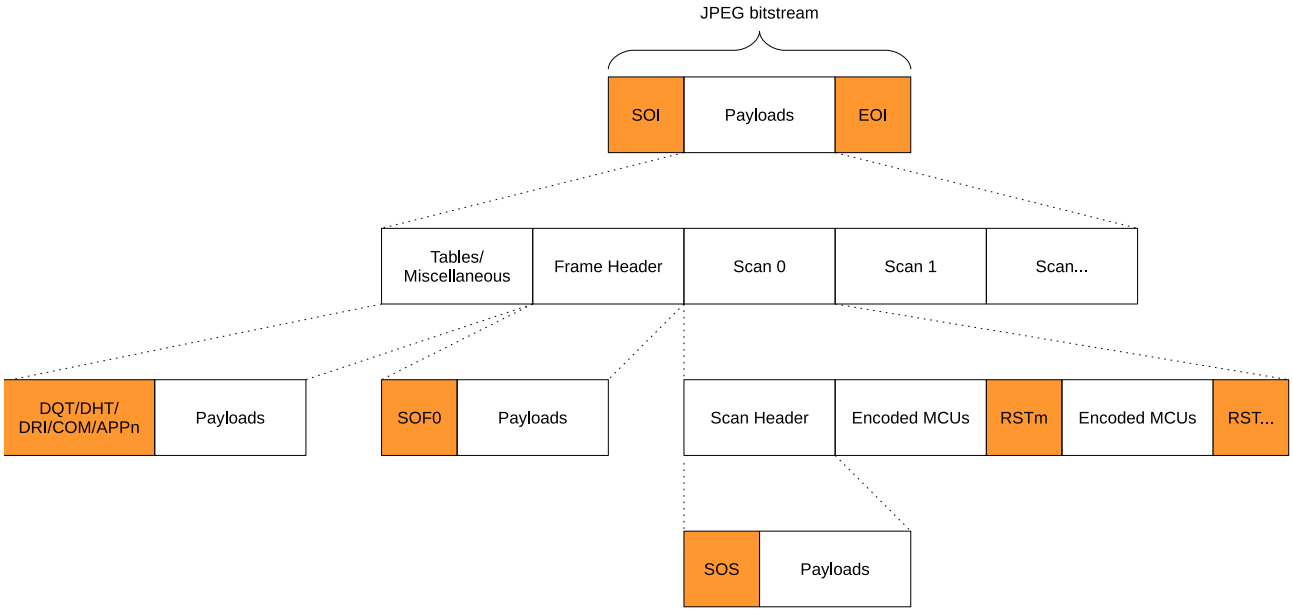


Figure 28.1-1. JPEG Bitstream Format

The marker code assignments are shown in Table 28.1-2.

Table 28.1-2. 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

28.2 Introduction

The JPEG codec is based on the JPEG baseline standard, which specifies that an encoding process is completed through one or more scans. An scan contains one or more MCUs. MCU is the sequence of data units defined by the sampling factors of the image components in the scan, which means MCU of different image formats contains different sequence of data units, as shown in Table 28.2-1.

Table 28.2-1. The Number of Data Units in the MCU of Different Image Format

Image Format	MCU[1]		
	Luminance Component Y	Chrominance Component U	Chrominance Component V
YUV444	1 x 1	1 x 1	1 x 1

YUV422	2 x 1	1 x 1	1 x 1
YUV420	2 x 2	1 x 1	1 x 1
GRAY	1 x 1	none	none

[1] The MCU of each image format is represented by the number of data units in the horizontal direction x the number of data units in the vertical direction of each component.

At a time, the JPEG codec can only be configured as an encoder or a decoder, that is, it cannot work as an encoder and a decoder at the same time.

- When used as an encoder, it obtains the image to be compressed through 2D DMA and sends the encoded bitstream to the 2D DMA.
- When used as a decoder, it obtains the coded bitstream through 2D DMA and sends the decoded image to 2D DMA.

The workflow of JPEG codec is shown in Figure 28.2-1.

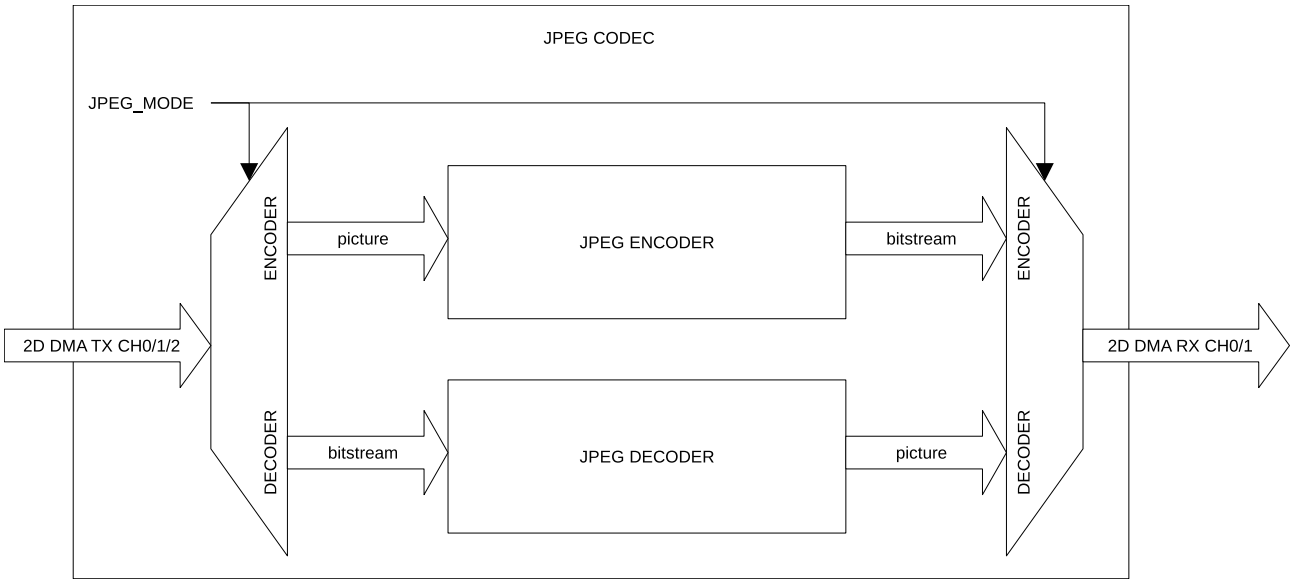


Figure 28.2-1. JPEG codec system connection block diagram

The function clock of the JPEG codec is SYS_CLK.

28.3 Features

When used as an encoder, the JPEG codec has the following features:

- integrated discrete cosine transform algorithm
- integrated canonical Huffman coding
- supported input image formats:
  - YUV444, YUV422, YUV420, and GRAY
  - RGB888 and RGB565 (after conversion into YUV444, YUV422 or YUV420)

- 4 x configurable quantization coefficient tables with 8-bit or 16-bit precision
- performance:
  - still image compression: up to 4K resolution
  - dynamic image compression: up to 1080P@40fps,720P@70fps (excluding header encoding time)
- automatically added stuffed zero byte
- automatically added EOI marker

When used as a decoder, the JPEG codec has the following features:

- integrated inverse discrete cosine transform algorithm
- integrated Huffman decoding
- supported image formats for compressed bitstream decoding: YUV444, YUV422, YUV420, and GRAY.
- 4 x configurable quantization coefficient tables with 8-bit or 16-bit precision
- 2 x DC and 2 x AC Huffman tables
- supports image decoding of any resolution. However, the resolution of output decoded image differs from the format of the input image:
  - YUV444, GRAY: both the horizontal and vertical resolutions of the output decoded image are multiples of 8, i.e., 150 × 150 images with an output resolution of 152 × 152
  - YUV422: the horizontal resolution of the output decoded image is the multiples of 16 and the vertical resolution is multiples of 8, i.e., 150 × 150 images with an output resolution of 160 × 152
  - YUV420: both the horizontal and vertical resolutions of the output decoded image are multiples of 16, i.e., 150 × 150 images with an output resolution of 160 × 160
- performance:
  - still image decoding: up to 4K resolution
  - dynamic image decoding: up to 1080P@40fps,720P@70fps (excluding header parsing time)

28.4 Architectural Overview

When the JPEG codec is configured as an encoder, its architecture is shown in Figure 28.4-1.

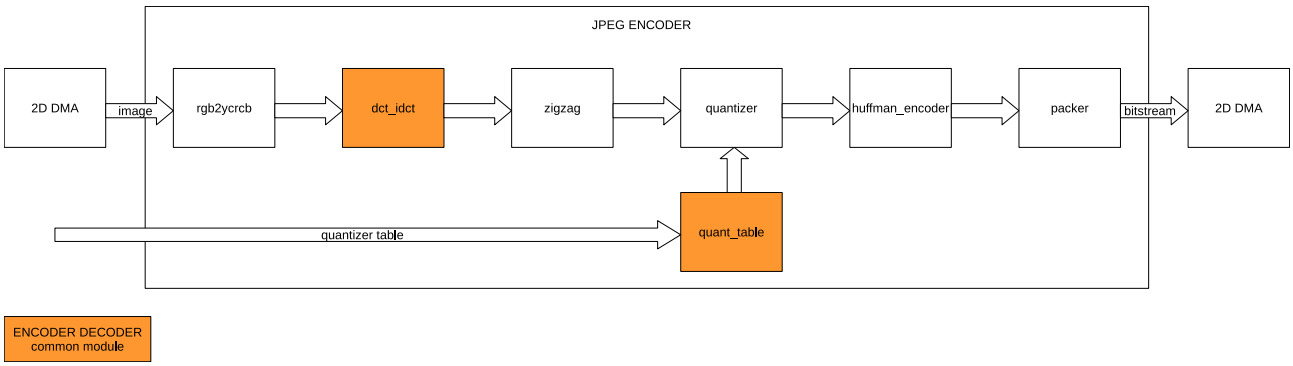


Figure 28.4-1. JPEG encoder architecture

- **rgb2ycrcb** implements the color space conversion, and converts the original image into an image format supported by the codec for compressing.
- **dct_idct** implements two-dimensional DCT transformation of data units.
- **zigzag** performs zigzag scanning and sorting on the DCT transformed data units.
- **quant_table** stores the quantization coefficient table delivered by the software.
- **quantizer** quantifies the zigzag sorted data units based on the quantization coefficient table stored in **quant_table**.
- **huffman_encoder** performs Huffman encoding on the quantized result of the data units.
- **packer** packs the Huffman-encoded data into groups of 32 bits, and outputs the final encoded bitstream.

When the JPEG codec is configured as a decoder, its architecture is shown in figure 28.4-2.

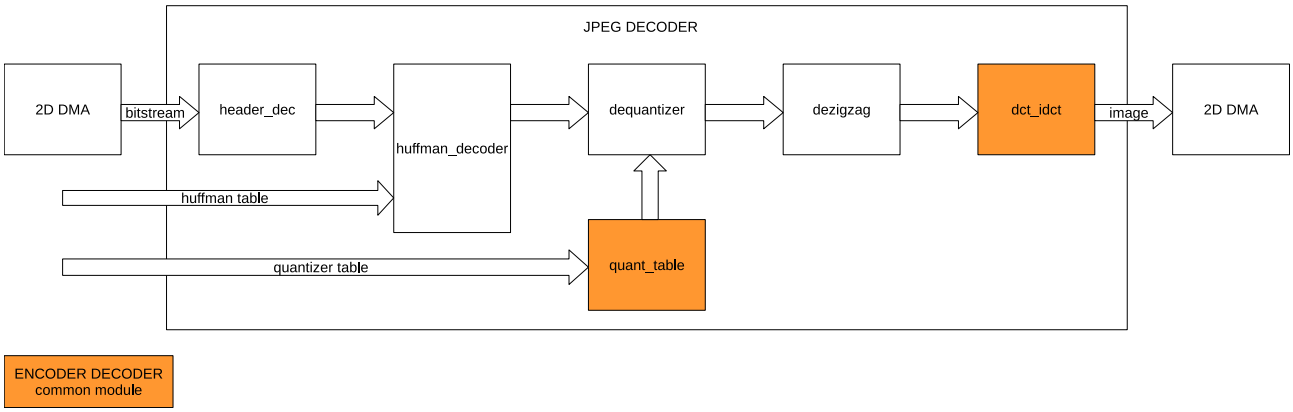


Figure 28.4-2. JPEG decoder architecture

- **header_dec** implements bitstream extraction and marker SOS and EOI parsing, and controls the decoding process.
- **huffman_decoder** stores the Huffman table delivered by the software, and implements Huffman decoding of the bitstream.
- **quant_table** stores the quantization coefficient table delivered by the software.
- **dequantizer** dequantizes the Huffman decoded data based on the quantization coefficient table stored in **quant_table**.
- **dezigzag** implements reverse zigzag scanning and sorting on the dequantized data units, and combines them into dequantized data units
- **dct_idct** implements the two-dimensional inverse DCT transformation of the dequantized data units, and outputs the final decoded data units.

Since the JPEG codec cannot be used as an encoder and decoder at the same time, the encoder and decoder share the **quant_table** and **dct_idct** modules to save hardware resources.

28.5 Functional Description

The JPEG Codec can be configured to work as an encoder or a decoder.

28.5.1 JPEG Encoder

28.5.1.1 Pause

The JPEG encoder can be paused during the operation for other higher priority tasks. This feature is controlled by register field JPEG_PAUSE_EN.

- Set JPEG_PAUSE_EN to 1, the JPEG encoder will no longer output the bitstream to the 2D DMA RX channel.
- Set JPEG_PAUSE_EN back to 0, the JPEG encoder will be resumed, and output the bitstream normally.

28.5.1.2 Color Space Conversion

According to the standard, the image formats supported by the JPEG encoder are YUV444, YUV422, YUV420, and GRAY. However, ESP32-P4's JPEG encoder has integrated a color space conversion function to convert RGB888 and RGB565 formats into the above-mentioned image formats supported by the encoder before encoding. In this way, the range of supported image formats is expanded.

The color space conversion formulas for converting RGB to YUV is:

$$Y = (0.299 * R) + (0.587 * G) + (0.114 * B)$$
$$U = (-0.1687 * R) + (-0.3313 * G) + (0.5 * B) + 128$$
$$V = (0.5 * R) + (-0.4187 * G) + (-0.0813 * B) + 128$$

The image format before conversion is represented by its format as well as pixel order. Table 28.5-1 demonstrates the representations of the supported image formats for color space conversion:

Table 28.5-1. 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

[1] The image formats can be converted into other formats supported by the encoder via configuring register field JPEG_SAMPLE_SEL:
- 0: YUV444
- 1: YUV422
- 2: YUV420

[2] Image formats YUV422 and GRAY will not be converted, regardless of the value of JPEG_SAMPLE_SEL.

28.5.1.3 Configurable Quantization Coefficient Table

ESP32-P4 JEPG encoder has implemented four 8 x 8 quantization coefficient tables in its hardware, which allows 64 software-configurable coefficient values per table.

The precision of each quantization table is configurable via JPEG_QNR_PRESITION:

- 0: low 8-bit effective
- 1: low 16-bit effective

The coefficient values for each table can be configured in FIFO mode or non-FIFO mode depending on the value of JPEG_QNR_FIFO_EN:

- 0: non-FIFO mode, in which each coefficient is written to a specific address
- 1: FIFO mode, in which all coefficients are written to the same address

Table 28.5-2. Position Number of Coefficients in a Quantization Coefficient Table (Example)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
...							
56	57	58	59	60	61	62	63

In different modes, the configuration of the quantization coefficient table is different. Note, *n* (*n* = 0, 1, 2, 3) indicates the number of each quantization coefficient table:

- non-FIFO mode: each quantization coefficient in a specific position of a quantization coefficient table *n* must be written to a specific address, which is (JPEG codec base address + *n* * 0x100 + *quantization coefficient position number* * 0x4), in which the *quantization coefficient position number* can be 0, 1, 2, ..., 63 (see Table 28.5-2). In this mode, since each coefficient is written to a specific address, configuring the 64 quantization coefficients in sequence is not required.
- FIFO mode: all 64 quantization coefficients of the quantization table *n* are written to the same address, which is JPEG_T*n*QNR_REG. In this mode, since all coefficients are written to the same address, configuring the 64 quantization coefficients one by one in sequence is required. Afterwards, the hardware will automatically store these 64 quantization coefficients in sequence as shown in Table 28.5-2.

If there are less than 4 quantization tables, the corresponding quantization table will not be written.

To use a specific quantization coefficient table *n* for the luminance component or the chrominance component, just set the corresponding register fields to *n*:

- luminance component: JPEG_LQNR_TBL_SEL
- chrominance component: JPEG_CQNR_TBL_SEL

The JPEG baseline standard recommended 8-bit precision quantization coefficient tables for the luminance component and the chrominance component are shown in Table 28.5-3 and Table 28.5-4.

Table 28.5-3. 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 28.5-4. Recommended 8-bit Precision Quantization Coefficient Table for the Chrominance Component

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

More information can be found in ITU T.81 Recommendation. Adjusting the quantization table can change the compression ratio (the ratio between the uncompressed image size and compressed image size) and the quality of encoded images (the larger the quantization coefficient, the greater the compression ratio and the worse the image quality).

28.5.1.4 Stuffed Zero Byte

As required in the JPEG baseline standard to avoid any marker within a Huffman-coded segment, any 0xFF byte generated by the Huffman coding should be followed by a "stuffed" zero byte.

JPEG encoder has implemented a mechanism to automatically add this "stuffed" 0x00 byte after each 0xFF byte in the Huffman-coded segment of bitstream.

Users can configure whether to add 0x00 byte after each 0xFF byte via JPEG_FF_CHECK_EN:

- 0: not add
- 1: add (by default)

28.5.1.5 EOI Marker

The end of Image (EOI) marker is used to mark the end of the bitstream of a compressed image. The EOI marker code is 0xFFD9.

JPEG encoder has implemented a mechanism to automatically add the EOI marker at the end of the bitstream.

Users can configure whether to add EOF marker via JPEG_TAILER_EN:

- 0: not add
- 1: add

28.5.2 JPEG Decoder

The information in the bitstream must be parsed by the software, then sent to the hardware through the APB configuration bus. Note that the hardware does not support parsing any marker segments, except the SOS, RST and EOI marker segments.

28.5.2.1 Multiple Chrominance Components

The frame header contains information about the number of chrominance components of the image, the ID of each component, and the sampling factors in the horizontal and vertical directions of each component.

JPEG decoder's software must parse such information and pass it to respective registers JPEG_COMPONENT_NUM, JPEG_C*n*_ID, JPEG_C*n*_X_FACTOR, and JPEG_C*n*_Y_FACTOR. Here, *n* can be 0, 1, 2, indicating that up to 3 chrominance components are supported by hardware. Note that if less than 3 components are involved, just do not configure the rest of the registers.

According to the value of register JPEG_C*n*_ID of each chrominance component, the hardware will check whether it is consistent with the ID in the scan header information. The sampling factor indicates the number of data units of this component that an MCU contains in both the horizontal and vertical directions. So according to the value of register JPEG_COMPONENT_NUM, JPEG_C*n*_X_FACTOR and JPEG_C*n*_Y_FACTOR, the hardware can obtain the proportional relationship of chrominance components, that is, the format of the image to be decoded. See Table 28.5-5.

Table 28.5-5. Sampling Factor and Image Format

JPEG_COMPONENT_NUM	JPEG_C*n*_X/Y_FACTOR	Format of the Image to be Decoded
0	-	invalid
1	C0_X_FACTOR is 1 C0_Y_FACTOR is 1	GRAY
2	-	invalid
3	C0_X_FACTOR is 1 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV444
	C0_X_FACTOR is 2 C0_Y_FACTOR is 1 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV422
	C0_X_FACTOR is 2 C0_Y_FACTOR is 2 C1_X_FACTOR is 1 C1_Y_FACTOR is 1 C2_X_FACTOR is 1 C2_Y_FACTOR is 1	YUV420

Regardless of the format of the image to be decoded, the decoder only supports one scan, which contains all components. After decoding, the image will be sent to 2D DMA, so please refer to Chapter 4 *2D-DMA Controller (2D-DMA)* for the layout of the decoded image in each format.

28.5.2.2 Parsing RST Marker

The bitstream may have a restart (RST) marker in the scan segment. The RST marker code is 0xFFD*x* (*x* = 0, 1, ..., 7).

JPEG decoder's hardware can parse RST markers and check whether the actual number of MCUs between two RST markers is the same as the restart interval defined in the DRI segment of the Tables/Miscellaneous segment.

Firstly, the software parses the DRI segment and pass the defined restart interval to register JPEG_RESTART_INTERVAL. Then, the hardware will check whether the subsequent bitstream has an RST marker every time JPEG_RESTART_INTERVAL MCUs are decoded.

Configuring JPEG_RESTART_INTERVAL to 0 indicates the bitstream does not contain any RST marker. In this case, the hardware will not check RST markers.

28.5.2.3 Configurable Quantization Coefficient Table

ESP32-P4 JPEG decoder also supports 4 configurable quantization coefficient tables. Since the JPEG codec cannot be used as an encoder and decoder at the same time, decoder can share the same four configurable quantization coefficient tables with the encoder to save hardware resources.

Therefore, the configuration of decoder's quantization coefficient tables is similar to that of the encoder (see Section 28.5.1.3) with some exceptions described below:

- The software needs to parse the DQT segment in the Tables/Miscellaneous segment to obtain the quantization coefficient tables of the decoder. The recommended quantization coefficient tables are the same as the encoder shown in Table 28.5-3 and Table 28.5-4.
- The precision of the quantization table*n* is configured via higher 4 bits of register JPEG_T*n*_DQT_INFO:
  - 0: low 8-bit effective
  - 1: low 16-bit effective

  The precision is also obtained from the DQT segment in the Tables/Miscellaneous segment.
- For each chrominance component, a different quantization coefficient table can be selected. The lower 4 bits of register JPEG_T*n*_DQT_INFO specify the ID of each quantization coefficient table, and each chrominance component selects the quantization coefficient table whose ID is the value of register JPEG_C*n*_DQT_TBL_SEL. The ID of each quantization coefficient table is also obtained from the DQT segment in the Tables/Miscellaneous segment.

28.5.2.4 Configurable Huffman Table

ESP32-P4 supports up to 2 DC Huffman tables and 2 AC Huffman tables. Users can configure as many Huffman tables as needed within the limits.

Each Huffman table can be established with three types of information, which can be obtained from the DHT segment in the Tables/Miscellaneous segment:

- the number of codeword with a codeword length of 1 to 16 bits,
- the symbols corresponding to the decoded codeword
  - DC table: up to 16 symbols
  - AC table: up to 256 symbols
- the minimum codeword with a codeword length of 1 to 16 bits.

Both of the DC and AC Huffman tables can be configured in a FIFO fashion or a non-FIFO fashion depending on the value of JPEG_DHT_FIFO_EN:

- 0: non-FIFO mode
- 1: FIFO mode

In different modes, the configuration of the Huffman table is different. Here is the configuration of DC tables*n* (*n* = 0, 1):

- non-FIFO mode: each piece in different type of information used to establish the DC*n* Huffman table must be written to a specific address.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0x500 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.
  - the symbols corresponding to the decoded codeword: JPEG codec base address + 0x600 + *n* * 0x40 + *symbol number* * 0x4, in which *symbol number* = 0, 1, ..., 15.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0xF00 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.

  In this mode, since each piece of different type of information is written to a specific address, configuring each piece of information in sequence is not required.
- FIFO mode: all pieces of the same type of information used to establish the DC*n* Huffman table must be written to the same address, respectively.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG_DHT_TOTLEN_DC*n*_REG.
  - all symbols corresponding to the decoded codeword (DC table has up to 16 symbols): JPEG_DHT_VAL_DC*n*_REG.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG_DHT_CODEMIN_DC*n*_REG.

  In this mode, since all pieces of the same type of information are written to the same address, configuring each piece of same type of information one by one in sequence is required. Afterwards, the hardware will automatically store such information in sequence.

The configuration of AC*n* (*n* = 0, 1) tables is very similar to that of the DC*n* (*n* = 0, 1) tables. Here below is the configuration of AC*n* (*n* = 0, 1) tables:

- non-FIFO mode: each piece in different type of information used to establish the AC*n* Huffman table must be written to a specific address.

  - the number of codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0x540 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.
  - the symbols corresponding to the decoded codeword: JPEG codec base address + 0x680 + *n* * 0x400 + *symbol number* * 0x4, in which *symbol number* = 0, 1, ..., 256.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG codec base address + 0xF40 + *n* * 0x80 + (*codeword length* - 1) * 0x4, in which *codeword length* = 1, 2, ..., 16.

  In this mode, since each piece of different type of information is written to a specific address, configuring each piece of information in sequence is not required.

- FIFO mode: all pieces of the same type of information used to establish the AC*n* Huffman table must be written to the same address, respectively.
  - the number of codeword with a codeword length of 1 to 16 bits: JPEG_DHT_TOTLEN_AC*n*_REG.
  - all symbols corresponding to the decoded codeword (AC table has up to 256 symbols): JPEG_DHT_VAL_AC*n*_REG.
  - the minimum codeword with a codeword length of 1 to 16 bits: JPEG_DHT_CODEMIN_AC*n*_REG.

  In this mode, since all pieces of the same type of information are written to the same address, configuring each piece of same type of information one by one in sequence is required. Afterwards, the hardware will automatically store such information in sequence.

For the minimum codeword with a codeword length of 1 to 16 bits, it should be moved left to the high bit of the 16-bit configuration data. If a codeword length does not exist, the configuration value is 0xFFFF. For example, if a minimum codeword is 0x7C, the 16-bit configuration data should be 0x7C00.

The JPEG baseline standard recommended DC and AC Huffman tables can be found in ITU T.81 Recommendation.

The software can obtain the IDs of DC*n* and AC*n* Huffman tables by parsing the DHT segment in the Tables/Miscellaneous segment, then pass these ID to the corresponding register JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

The hardware will automatically use the DC*n* and AC*n* Huffman tables with the same ID from JPEG_DC*n*_DHT_ID and JPEG_AC*n*_DHT_ID.

28.5.2.5 Timeout Detection

Additionally, timeout detection is supported when the JPEG codec is working as a decoder. The hardware triggers the timeout interrupt JPEG_DECODE_TIMEOUT_INT:

- when the decoder stops reading the bitstream for more than $(2^{JPEG\_DECODE\_TIMEOUT\_THRES} - 1)$ clock cycles,
- or when the number of clock cycles consumed for decoding an image exceeds the value of $(2^{JPEG\_DECODE\_TIMEOUT\_THRES} - 1)$

In this case, the decoder will be reset depending on the configuration of JPEG_DECODE_TIMEOUT_TASK_SEL:

- 0: the hardware will not perform any operation, and the software is required to configure the JPEG_SOFT_RST or JPEG_FSM_RST to perform a reset operation.

- 1: the hardware will automatically reset the decoder to exit the timeout state and enter the initial state.

28.6 Interrupts

ESP32-P4's JPEG Codec can generate the JPEG_INTR interrupt signal that will be sent to the Interrupt Matrix.

Interrupt signal JPEG_INTR can be generated by the following internal interrupt sources:

- JPEG_DONE_INT: Triggered when a frame of image encoding or decoding ends.
- JPEG_RLE_PARALLEL_ERR_INT: Triggered when a run-length encoding error occurs.
- JPEG_CID_ERR_INT: Triggered when the decoded component ID is different from the component ID configured by the software.
- JPEG_C_DHT_DC_ID_ERR_INT: Triggered when the decoded DC Huffman table ID of each component is not the software configured DC0 Huffman table ID or DC1 Huffman table ID.
- JPEG_C_DHT_AC_ID_ERR_INT: Triggered when the decoded AC Huffman table ID of each component is not the software configured AC0 Huffman table ID or AC1 Huffman table ID.
- JPEG_C_DQT_ID_ERR_INT: Triggered when the decoded quantization table ID of each component is different from the software configured quantization table ID.
- JPEG_RST_UXP_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is 0 but the RST marker is parsed by the decoder.
- JPEG_RST_CHECK_NONE_ERR_INT: Triggered when the JPEG_RESTART_INTERVAL configured by the software is non-0 but the RST marker cannot be parsed by the decoder.
- JPEG_RST_CHECK_POS_ERR_INT: Triggered when the MCU number between two parsed RST markers is not equal to the JPEG_RESTART_INTERVAL configured by the software.
- JPEG_OUT_EOF_INT: Triggered when the EOF marker is read from the TX channel of 2D DMA.
- JPEG_SR_COLOR_MODE_ERR_INT: This interrupt is invalid.
- JPEG_DCT_DONE_INT: Triggered when the DCT or IDCT calculation of one data unit is completed.
- JPEG_BS_LAST_BLOCK_EOF_INT: Triggered when the encoding of the last data unit is completed.
- JPEG_SCAN_CHECK_NONE_ERR_INT: Triggered when an image frame has multiple scans to be decoded and the SOS marker is not parsed within (JPEG_SOS_CHECK_BYTE_NUM + 1) bytes in any scan header information.
- JPEG_SCAN_CHECK_POS_ERR_INT: Triggered when the position of the scan header information parsed by the decoder is wrong.
- JPEG_UXP_DET_INT: Triggered when the marker parsed by the decoder is not supported by the hardware.
- JPEG_EN_FRAME_EOF_ERR_INT: Triggered when the number of data units to be encoded read from 2D DMA is less than the number of data units calculated based on the image resolution configured by the software.

- JPEG_EN_FRAME_EOF_LACK_INT: Triggered when a frame of image to be encoded is completely read from 2D DMA but the EOF marker is not read.
- JPEG_DE_FRAME_EOF_ERR_INT: Triggered when the number of data units obtained after decoding a frame of image is different from the number of data units calculated based on the image resolution configured by the software.
- JPEG_DE_FRAME_EOF_LACK_INT: Triggered when the bitstream of a image is completely read from 2D DMA but the EOF marker or EOI marker is not read.
- JPEG_SOS_UNMATCH_ERR_INT: Triggered when the number of components in the scan header information parsed by the decoder is 0 or the header length in the scan header information parsed by the decoder does not match the actual header length.
- JPEG_MARKER_ERR_FST_SCAN_INT: Triggered when there is an error in the first scan header information parsed by the decoder. The errors include errors triggering JPEG_CID_ERR_INT, errors triggering JPEG_C_DHT_DC_ID_ERR_INT, errors triggering JPEG_C_DHT_AC_ID_ERR_INT, errors triggering JPEG_C_DQT_ID_ERR_INT and errors triggering JPEG_SOS_UNMATCH_ERR_INT.
- JPEG_MARKER_ERR_OTHER_SCAN_INT: Triggered when there is an error in the non-first scan header information parsed by the decoder. The error type is the same as the errors that trigger JPEG_MARKER_ERR_FST_SCAN_INT.
- JPEG_UNDET_INT: Triggered when the bitstream of an image is completely read from 2D DMA but the SOS marker is not read.
- JPEG_DECODE_TIMEOUT_INT: Triggered when the decoder is timeout. Section 28.7.2 describes how this interrupt will occur.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 28.8 *Register Summary*.

28.7 Programming Procedures

28.7.1 JPEG Encoder

When the JPEG codec is working as an encoder, the configuration process is:

1. Activate the JPEG codec:
   - Set HP_SYS_CLKRST_JPEG_SYS_CLK_EN to 1 to enable the JPEG codec clock;
   - Set HP_SYS_CLKRST_RST_EN_JPEG to 0 to release the JPEG codec system level reset.
2. Configure the JPEG codec:

- Set JPEG_SOFT_RST to 1 to reset the JPEG codec, and then set JPEG_SOFT_RST to 0 to release the JPEG codec reset.
- Set JPEG_MODE to 0 to configure the JPEG codec to be used as an encoder.
- Select the original image format by configuring the following two registers:
  - Format: JPEG_COLOR_SPACE
  - Pixel order: JPEG_PIXEL_REV

  See details in Table 28.5-1.
- Select whether to add the EOI marker (0xFFD9) at the end of the bitstream by configuring JPEG_TAILER_EN:
  - 0: disable
  - 1: enable
- Select the image format to be compressed by configuring JPEG_SAMPLE_SEL:
  - 0: YUV444
  - 1: YUV422
  - 2: YUV420

  Note that this register is valid only when the original image format is RGB. See details in Table 28.5-1.
- Configure JPEG_HA as the original image width and JPEG_VA as the original image height.
- Configure the quantization coefficient table by following the instructions in Section 28.5.1.3.

3. Configure 2D DMA (For detailed instructions or introduction to related concepts, please refer to 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. Among them, according to the original image format and the format of the image to be compressed, there are restrictions on hb and vb, as shown in the table 28.7-1:

Table 28.7-1. 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, which means one-time reading of image blocks of hb × vb size.

(c) Connect 2D DMA channels to JPEG codec:

- Connect the TX channel*n* to the JPEG codec by configuring DMA2D_OUT_PERI_SEL_CH*n* to 0;
- Connect the RX channel*n* to the JPEG codec by configuring DMA2D_IN_PERI_SEL_CH*n* to 0.

(d) Reset 2D DMA channels:

- 2D DMA TX channel*n*: Set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.
- 2D DMA RX channel*n*: Set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*.
- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*.

(f) Select the 2D DMA output image block size by configuring DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* according to the format of original image and the format of the image to be compressed. See details in Table 28.7-2:

- 0: 8x8
- 1: 16x8
- 2: 16x16

Table 28.7-2. 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. Note that, this function is only available for 2D DMA TX channel0, not for any other TX channels.

(h) Configure 2D DMA channel linked list address:

- 2D DMA TX channel: DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX channel: DMA2D_INLINK_ADDR_CH*n*

4. Enable interrupt sources:
   - DMA2D_OUT_INT_ENA_CH*n*_REG
   - DMA2D_IN_INT_ENA_CH*n*_REG
   - JPEG_INT_ENA_REG
5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0
   - 2D DMA RX channel: set DMA2D_INLINK_START_CH*n* to 1 and then to 0
6. Start the JPEG codec encoding by setting the JPEG_JPEG_START field.
7. Wait till DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the encoded bitstream has been completely written to the memory.

28.7.2 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 28.5.2.2, if required.
   - Configure the quantization coefficient tables by following the instructions in Section 28.5.2.3.
   - Configure the Huffman tables by following the instructions in Section 28.5.2.4.
   - Configure the width and height of the decoded image:
     - Width: write width of the decoded image to JPEG_HA.
     - Height: write the height of the decoded image to JPEG_VA.

– According to different decoded image formats, JPEG_HA and JPEG_VA must meet the requirements of Table 28.7-3.

Table 28.7-3. 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 detailed instructions or introduction to related concepts, please refer to 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 the length and size as the number of bytes in the space to store the bitstream to be decoded.
   - configure mod as 0, which means one-time reading of image blocks of hb x vb size.

   (b) Configure the linked list of 2D DMA RX channel.

   - Set 2DEN to 1 to enable 2D function.
   - Configure the Buffer address pointer as the starting address where the decoded image will be stored.
   - Configure pbyte to select the number of bytes of one pixel of the image which is decoded and has undergone 2D DMA color space conversion.
   - Configure HA as the width of the output decoded image, configure VA as the height of the output decoded image, configure hb as the number of pixels in the horizontal direction of an image block, configure vb as the number of pixels in the vertical direction of an image block.
   - Configure mod as 1, which means continuous reading of image blocks of hb × vb size.
   - Configure (X, Y) as the coordinates of the first pixel in the starting image block. Among them, according to the decoded image format, there are restrictions on hb and vb, as shown in the table 28.7-4:

Table 28.7-4. JPEG Decoder hb and vb Configuration

Decoded image format	hb	vb
YUV444	Multiples of 8	Multiples of 8
YUV422	Multiples of 16	Multiples of 8
YUV420	Multiples of 16	Multiples of 16
GRAY	Multiples of 8	Multiples of 8

(c) Connect 2D DMA channels to JPEG codec:

- Connect the TX channel*n* to the JPEG codec by configuring DMA2D_OUT_PERI_SEL_CH*n* to 0;
- Connect the RX channel*n* to the JPEG codec by configuring DMA2D_IN_PERI_SEL_CH*n* to 0.

(d) Reset 2D DMA channels:

- 2D DMA TX channel*n*: set DMA2D_OUT_RST_CH*n* to 1 to reset 2D DMA TX channel*n*, then set DMA2D_OUT_RST_CH*n* to 0 to release the reset of 2D DMA TX channel*n*.
- 2D DMA RX channel*n*: set DMA2D_IN_RST_CH*n* to 1 to reset 2D DMA RX channel*n*, then set DMA2D_IN_RST_CH*n* to 0 to release the reset of 2D DMA RX channel*n*.

(e) Select the burst length of 2D DMA channels:

- Configure DMA2D_OUT_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA TX channel*n*
- Configure DMA2D_IN_MEM_BURST_LENGTH_CH*n* to select the burst length of 2D DMA RX channel*n*

(f) Select 2D DMA receive image block size by configuring DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* according to the format of the decoded image. See details in Table 28.7-5:

- 0: 8x8
- 1: 16x8
- 2: 16x16

Table 28.7-5. DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* Configuration

Format of Decoded Image	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*
YUV444	0
YUV422	1
YUV420	2
GRAY	0

(g) Enable the reorder function to improve bandwidth utilization by setting set DMA2D_IN_REORDER_EN_CH0 to 1. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(h) Select whether to scramble the pixel order of the decoded image before 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1

- 5: BYTE0-1-2

Configuring DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 to 5 means swapping the pixel order, for example, swapping YUV444 output by the JPEG decoder to VUY444. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(i) Select the color conversion methods by configuring DMA2D_IN_COLOR_INPUT_SEL_CH0, DMA2D_IN_COLOR_3B_PROC_EN_CH0, and DMA2D_IN_COLOR_OUTPUT_SEL_CH0. See details in Table 28.7-6:

Table 28.7-6. 2D DMA Color Space Conversion Configuration

Color Space Conversion Method	INPUT_SEL_CH0	3B_PROC_EN_CH0	OUTPUT_SEL_CH0
YUV420 → RGB	0	1	1
YUV422 → RGB	0	1	1
YUV444 → RGB	2	1	1
YUV444 (only reverse the pixel order)	2	0	1
No color space conversion	7	N/A	N/A

Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(j) Configure parameters required for the color space conversion formula, including DMA2D_IN_COLOR_PARAM_H0_CH0, DMA2D_IN_COLOR_PARAM_H1_CH0, DMA2D_IN_COLOR_PARAM_M0_CH0, DMA2D_IN_COLOR_PARAM_M1_CH0, DMA2D_IN_COLOR_PARAM_L0_CH0, and DMA2D_IN_COLOR_PARAM_L1_CH0. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(k) Select whether to reverse the pixel order of the image after 2D DMA color space conversion by configuring DMA2D_IN_SCRAMBLE_SEL_POST_CH0:

- 0: BYTE2-1-0
- 1: BYTE2-0-1
- 2: BYTE1-0-2
- 3: BYTE1-2-0
- 4: BYTE0-2-1
- 5: BYTE0-1-2

Configuration of 5 means reversing the pixel order, for example, reversing the RGB after 2D DMA color space conversion to BGR. Note that, this function is only available for 2D DMA RX channel0, not for any other RX channels.

(l) Configure DMA2D_OUTLINK_ADDR_CH*n* as the 2D DMA TX channel linked list address, configure DMA2D_INLINK_ADDR_CH*n* as the 2D DMA RX channel linked list address.

4. Enable interrupt sources:

- DMA2D_OUT_INT_ENA_CH*n*_REG
- DMA2D_IN_INT_ENA_CH*n*_REG
- JPEG_INT_ENA_REG

5. Start the transmission of 2D DMA channels:
   - 2D DMA TX channel*n*: set DMA2D_OUTLINK_START_CH*n* to 1 and then to 0
   - 2D DMA RX channel*n*: set DMA2D_INLINK_START_CH*n* to 1 and then to 0
6. Start JPEG codec decoding by setting the JPEG_JPEG_START field.
7. Wait till DMA2D_IN_SUC_EOF_CH*n*_INT becomes 1, indicating that the decoded image has been completely written to the memory.

28.7.3 Reset

Users can **reset** JPEG codec entirely or partially, depending on the actual scenarios, at any time during the encoding or decoding process:

- The entire JPEG codec, including all the RAMs and FIFOs as well as all the state machines: set JPEG_SOFT_RST to 1. In this case, the register configuration value will not be reset.
- Only all the RAMs and FIFOs inside the JPEG codec, for example, changing the quantization coefficient tables or Huffman tables: set JPEG_FIFO_RST to 1.
- Only all the state machines inside the JPEG codec, for example, only stop the current encoding or decoding process: set JPEG_FSM_RST to 1.

28.8 Register Summary

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

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

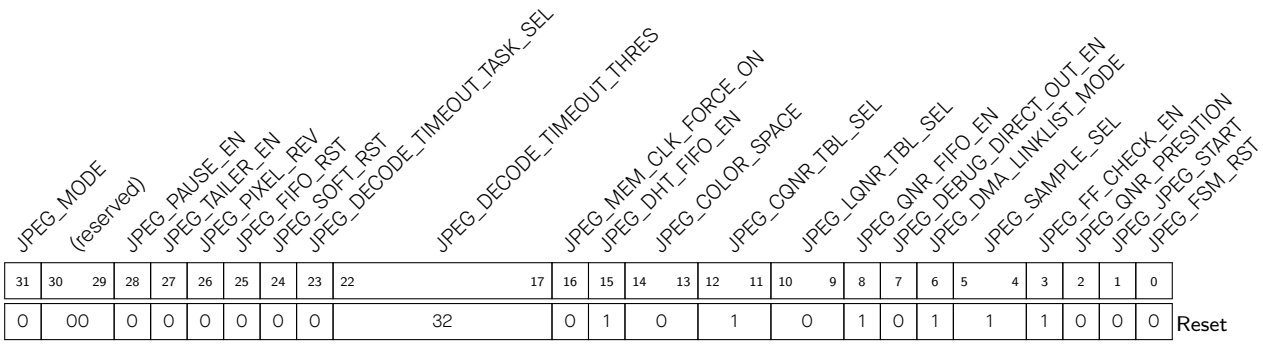
Name	Description	Address	Access
Control and Configuration registers			
JPEG_CONFIG_REG	System level control and configuration register	0x0000	varies
JPEG_DQT_INFO_REG	Quantization coefficient table information register	0x0004	R/W
JPEG_PIC_SIZE_REG	Image size register	0x0008	R/W
JPEG_T0QNR_REG	Quantization coefficient table0 register	0x0010	HRO
JPEG_T1QNR_REG	Quantization coefficient table1 register	0x0014	HRO
JPEG_T2QNR_REG	Quantization coefficient table2 register	0x0018	HRO
JPEG_T3QNR_REG	Quantization coefficient table3 register	0x001C	HRO
JPEG_DECODE_CONF_REG	Decoder configuration register	0x0020	varies
JPEG_C0_REG	Color component0 register	0x0024	R/W
JPEG_C1_REG	Color component1 register	0x0028	R/W
JPEG_C2_REG	Color component2 register	0x002C	R/W
JPEG_C3_REG	Color component3 register	0x0030	R/W
JPEG_DHT_INFO_REG	Huffman table information register	0x0034	R/W
JPEG_DHT_TOTLEN_DC0_REG	DC0 Huffman table codeword length register	0x0058	HRO
JPEG_DHT_VAL_DC0_REG	DC0 Huffman table symbol register	0x005C	HRO
JPEG_DHT_TOTLEN_AC0_REG	AC0 Huffman table codeword length register	0x0060	HRO
JPEG_DHT_VAL_AC0_REG	AC0 Huffman table symbol register	0x0064	HRO
JPEG_DHT_TOTLEN_DC1_REG	DC1 Huffman table codeword length register	0x0068	HRO
JPEG_DHT_VAL_DC1_REG	DC1 Huffman table symbol register	0x006C	HRO
JPEG_DHT_TOTLEN_AC1_REG	AC1 Huffman table codeword length register	0x0070	HRO
JPEG_DHT_VAL_AC1_REG	AC1 Huffman table symbol register	0x0074	HRO
JPEG_DHT_CODEMIN_DC0_REG	DC0 Huffman table minimum codeword register	0x0078	HRO
JPEG_DHT_CODEMIN_AC0_REG	AC0 Huffman table minimum codeword register	0x007C	HRO
JPEG_DHT_CODEMIN_DC1_REG	DC1 Huffman table minimum codeword register	0x0080	HRO
JPEG_DHT_CODEMIN_AC1_REG	AC1 Huffman table minimum codeword register	0x0084	HRO
JPEG_SYS_REG	System configuration register	0x00F8	R/W
Interrupt registers			
JPEG_INT_RAW_REG	Interrupt raw status register	0x0038	R/WTC/SS
JPEG_INT_ENA_REG	Interrupt enable register	0x003C	R/W
JPEG_INT_ST_REG	Interrupt masked status register	0x0040	RO
JPEG_INT_CLR_REG	Interrupt clear register	0x0044	WT
Version Register			
JPEG_VERSION_REG	Version control register	0x00FC	R/W

28.9 Registers

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

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

Register 28.1. JPEG_CONFIG_REG (0x0000)



JPEG_FSM_RST Configures whether or not to reset the state machine of JPEG Codec.

0: Invalid. No effect

1: Reset the state machine

(WT)

JPEG_JPEG_START Configures whether or not to start compressing a new image.

0: Invalid. No effect

1: Start compressing a new image

(WT)

JPEG_QNR_PRESITION Configures the quantization coefficient table precision for the encoder.

0: 8-bit precision

1: 16-bit precision

(R/W)

JPEG_FF_CHECK_EN Configures whether or not to add “0x00” after “0xFF”.

0: Not add

1: Add

(R/W)

JPEG_SAMPLE_SEL Configures the format of the image to be compressed.

0: YUV444

1: YUV422

2: YUV420

3: Invalid. No effect

(R/W)

Continued on the next page...

Register 28.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_DMA_LINKLIST_MODE Represents 2D DMA linked list mode.
0: Invalid. No effect
1: 2D DMA uses linked lists to configure
(RO)

JPEG_DEBUG_DIRECT_OUT_EN Configures whether out not to enable debug mode for encoder.
0: Disable debug mode, the encoder will work in normal mode
1: Enable debug mode, the input image will be directly output by the encoder
(R/W)

JPEG_QNR_FIFO_EN Configures whether or not to enable FIFO mode when configuring quantization coefficient tables.
0: Disable. Use non-FIFO mode
1: Enable
(R/W)

JPEG_LQNR_TBL_SEL Configures the luminance quantization table ID for the encoder. (R/W)

JPEG_CQNR_TBL_SEL Configures the chrominance quantization table ID for the encoder. (R/W)

JPEG_COLOR_SPACE Configures the original image's color space.
0: RGB888
1: YUV422
2: RGB565
3: GRAY
(R/W)

JPEG_DHT_FIFO_EN Configures whether or not to enable FIFO mode when configuring Huffman tables.
0: Disable. Use non-FIFO mode
1: Enable
Note that to read Huffman tables only non-FIFO mode is supported.
(R/W)

JPEG_MEM_CLK_FORCE_ON Configures whether or not to force on memory's clock gate.
0: Not force on
1: Force on
(R/W)

JPEG_DECODE_TIMEOUT_THRES Configures decode timeout period to trigger JPEG_DECODE_TIMEOUT_INT_RAW. The clock cycles of timeout period = $2^{JPEG\_DECODE\_TIMEOUT\_THRES} - 1$. (R/W)

Continued on the next page...

Register 28.1. JPEG_CONFIG_REG (0x0000)

Continued from the previous page...

JPEG_DECODE_TIMEOUT_TASK_SEL Configures the reset mode upon the decoder timeout.

0: Software uses reset to abort the decoding process

1: Hardware automatically aborts decoding process

(R/W)

JPEG_SOFT_RST Configures whether or not to apply soft reset to the JPEG Codec.

0: Release the soft reset

1: Apply soft reset to the JPEG Codec (the register configuration value will not be reset)

(R/W)

JPEG_FIFO_RST Configures whether or not to apply FIFO reset to the JPEG Codec.

0: Release the FIFO reset

1: Apply FIFO reset to the JPEG Codec

(R/W)

JPEG_PIXEL_REV Configures whether or not to reverse the original image's pixel order.

0: Not reverse

1: Reverse

(R/W)

JPEG_TAILER_EN Configures whether or not to add EOI marker "0xFFD9" at the end of bitstream.

0: Not add

1: Add

(R/W)

JPEG_PAUSE_EN Configures whether or not to pause the JPEG Codec.

0: Not pause

1: Pause

(R/W)

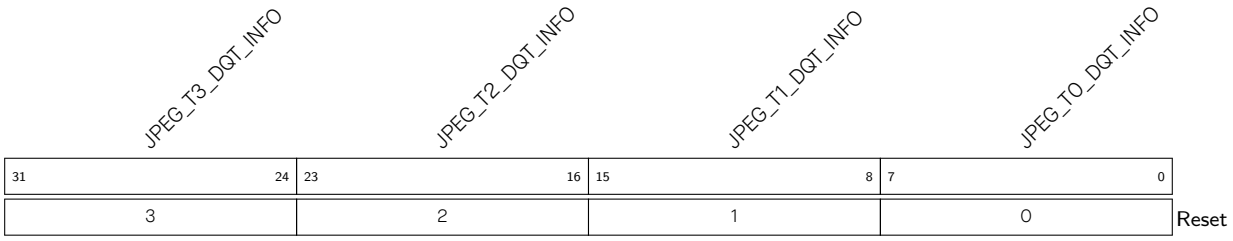
JPEG_MODE Configures if the JPEG Codec is working as an encoder or a decoder.

0: Encoder

1: Decoder

(R/W)

Register 28.2. JPEG_DQT_INFO_REG (0x0004)



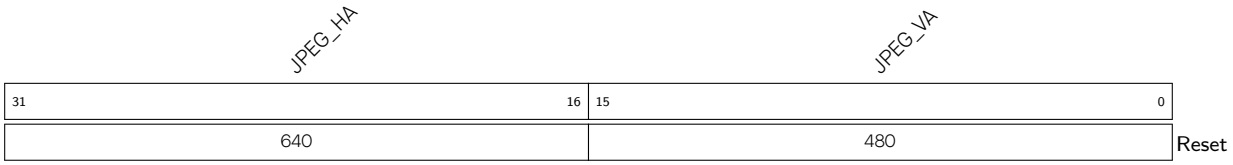
JPEG_T0_DQT_INFO Configures quantization coefficient table0's precision and ID for decoder.
Bit[7:4]: Configures precision
Bit[3:0]: Configures ID
(R/W)

JPEG_T1_DQT_INFO Configures quantization coefficient table1's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T2_DQT_INFO Configures quantization coefficient table2's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

JPEG_T3_DQT_INFO Configures quantization coefficient table3's precision and ID for decoder. See details in JPEG_T0_DQT_INFO. (R/W)

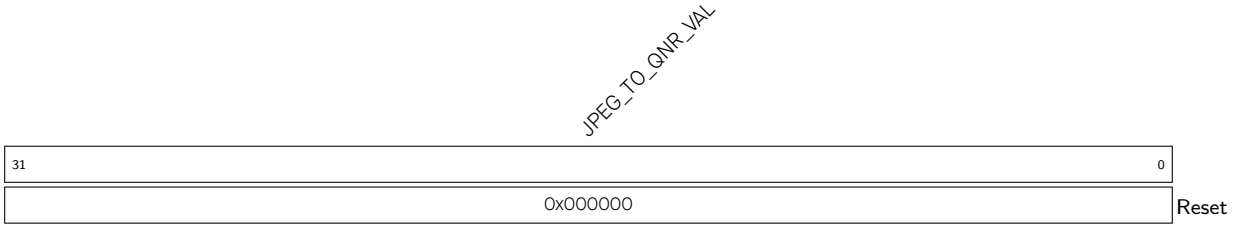
Register 28.3. JPEG_PIC_SIZE_REG (0x0008)



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)

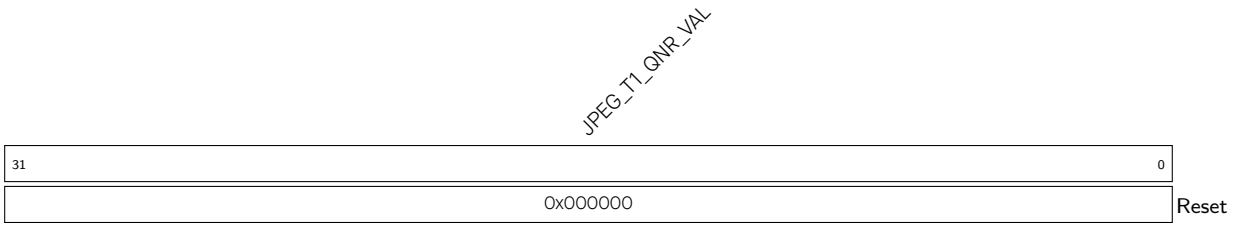
JPEG_HA Configures the image's width.
When JPEG Codec works as an encoder, the maximum configurable bits is 14.
When JPEG Codec works as a decoder, the maximum configurable bits is 16.
(R/W)

Register 28.4. JPEG_T0QNR_REG (0x0010)



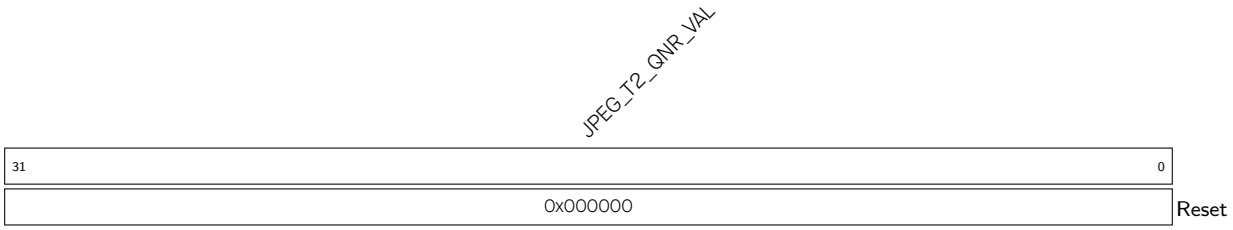
JPEG_T0_QNR_VAL Configures the quantization coefficients of the quantization table 0 in FIFO mode. (HRO)

Register 28.5. JPEG_T1QNR_REG (0x0014)



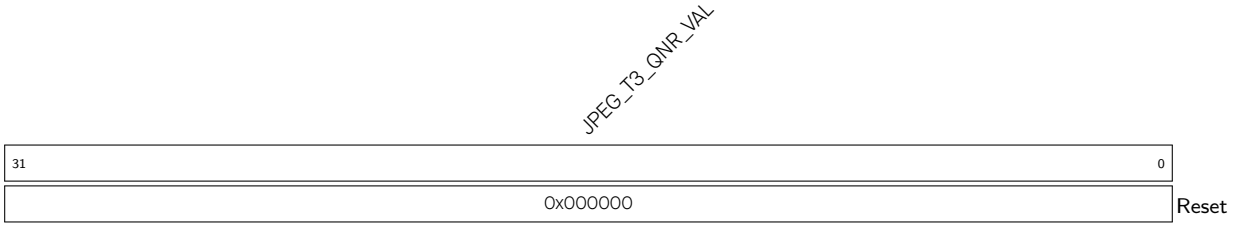
JPEG_T1_QNR_VAL Configures the quantization coefficients of the quantization table 1 in FIFO mode. (HRO)

Register 28.6. JPEG_T2QNR_REG (0x0018)



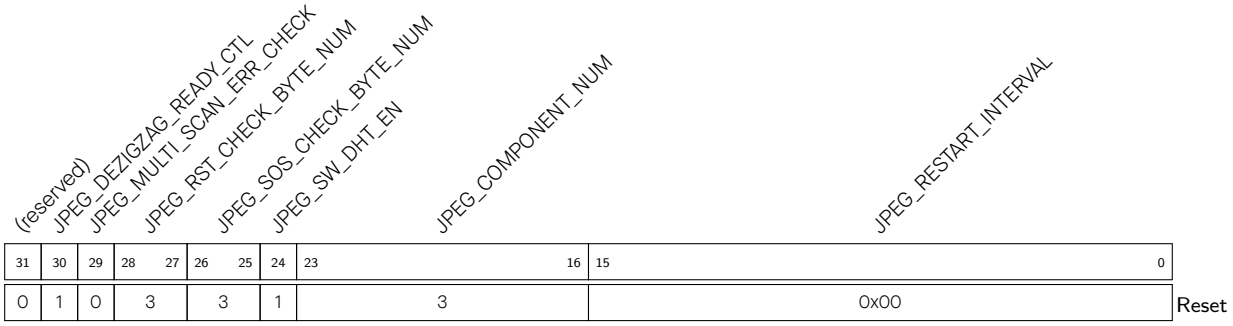
JPEG_T2_QNR_VAL Configures the quantization coefficients of the quantization table 2 in FIFO mode. (HRO)

Register 28.7. JPEG_T3QNR_REG (0x001C)



JPEG_T3_QNR_VAL Configures the quantization coefficients of the quantization table 3 in FIFO mode. (HRO)

Register 28.8. JPEG_DECODE_CONF_REG (0x0020)



JPEG_RESTART_INTERVAL Configures the RST interval in DRI segment when decoding. (R/W)

JPEG_COMPONENT_NUM Configures the number of chrominance components in the image when decoding. (R/W)

JPEG_SW_DHT_EN Represents the Huffman table configured by the software for the decoder. The value is always 1. (RO)

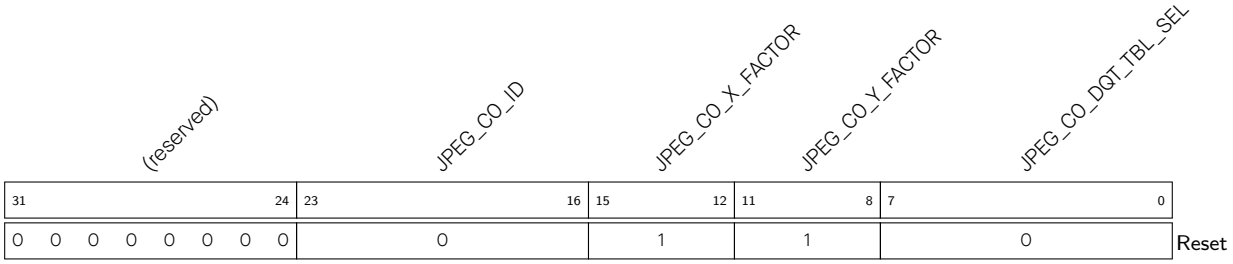
JPEG_SOS_CHECK_BYTE_NUM Configures the byte number to check the next SOS marker in the multi-scan image after one scan is decoded down. The actual check number is JPEG_SOS_CHECK_BYTE_NUM + 1 (R/W)

JPEG_RST_CHECK_BYTE_NUM Configures the byte number to check the next RST marker after one RST interval is decoded down. The actual check number is JPEG_RST_CHECK_BYTE_NUM + 1. (R/W)

JPEG_MULTI_SCAN_ERR_CHECK Reserved for decoder and should not be configured. (R/W)

JPEG_DEZIGZAG_READY_CTL Reserved for decoder and should not be configured. (R/W)

Register 28.9. JPEG_C0_REG (0x0024)



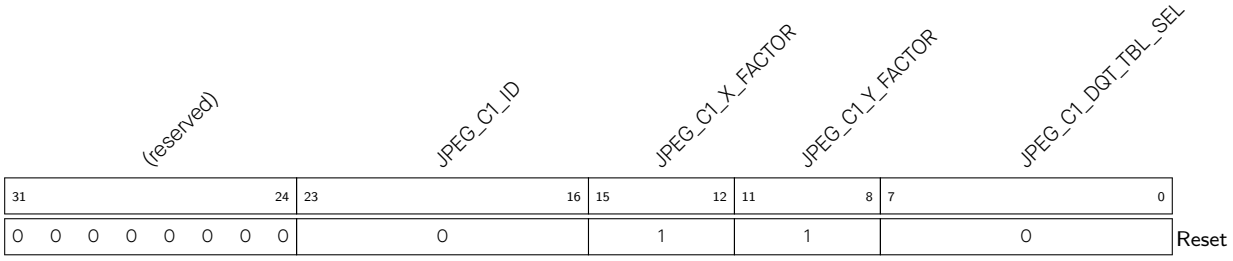
JPEG_C0_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component0 in decoder mode. (R/W)

JPEG_C0_Y_FACTOR Configures the vertical sampling factor of component0 in decoder mode. (R/W)

JPEG_C0_X_FACTOR Configures the horizontal sampling factor of component0 in decoder mode. (R/W)

JPEG_C0_ID Configures the ID of component0 in decoder mode. (R/W)

Register 28.10. JPEG_C1_REG (0x0028)



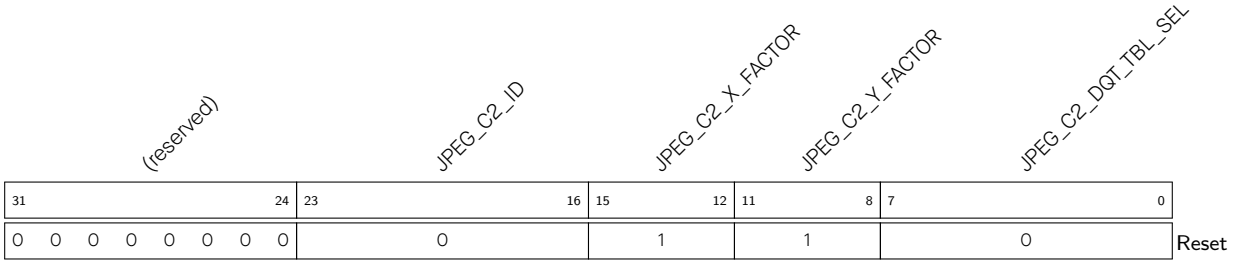
JPEG_C1_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component1 in decoder mode. (R/W)

JPEG_C1_Y_FACTOR Configures the vertical sampling factor of component1 in decoder mode. (R/W)

JPEG_C1_X_FACTOR Configures the horizontal sampling factor of component1 in decoder mode. (R/W)

JPEG_C1_ID Configures the ID of component1 in decoder mode. (R/W)

Register 28.11. JPEG_C2_REG (0x002C)



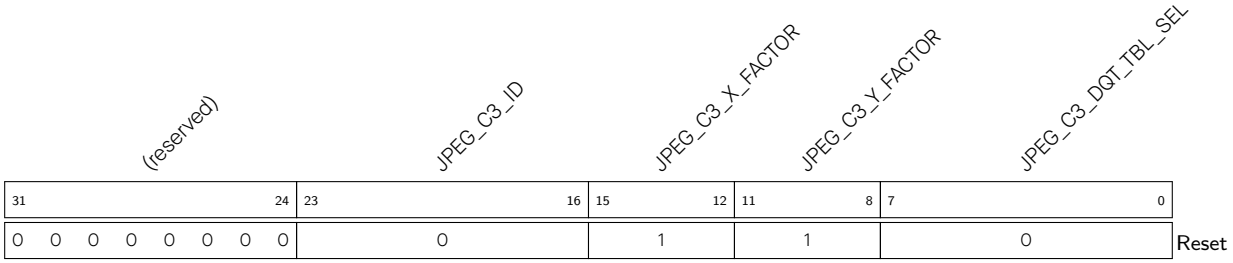
JPEG_C2_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component2 in decoder mode. (R/W)

JPEG_C2_Y_FACTOR Configures the vertical sampling factor of component2 in decoder mode. (R/W)

JPEG_C2_X_FACTOR Configures the horizontal sampling factor of component2 in decoder mode. (R/W)

JPEG_C2_ID Configures the ID of component2 in decoder mode. (R/W)

Register 28.12. JPEG_C3_REG (0x0030)



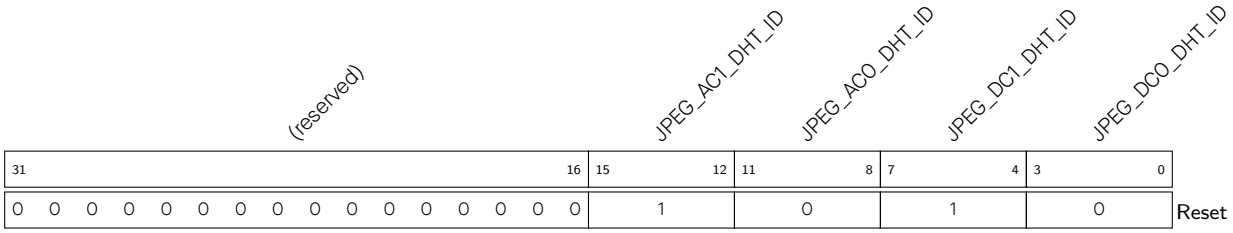
JPEG_C3_DQT_TBL_SEL Configures the selected quantization coefficient table ID for component3 in decoder mode. (R/W)

JPEG_C3_Y_FACTOR Configures the vertical sampling factor of component3 in decoder mode. (R/W)

JPEG_C3_X_FACTOR Configures the horizontal sampling factor of component3 in decoder mode. (R/W)

JPEG_C3_ID Configures the ID of component3 in decoder mode. (R/W)

Register 28.13. JPEG_DHT_INFO_REG (0x0034)



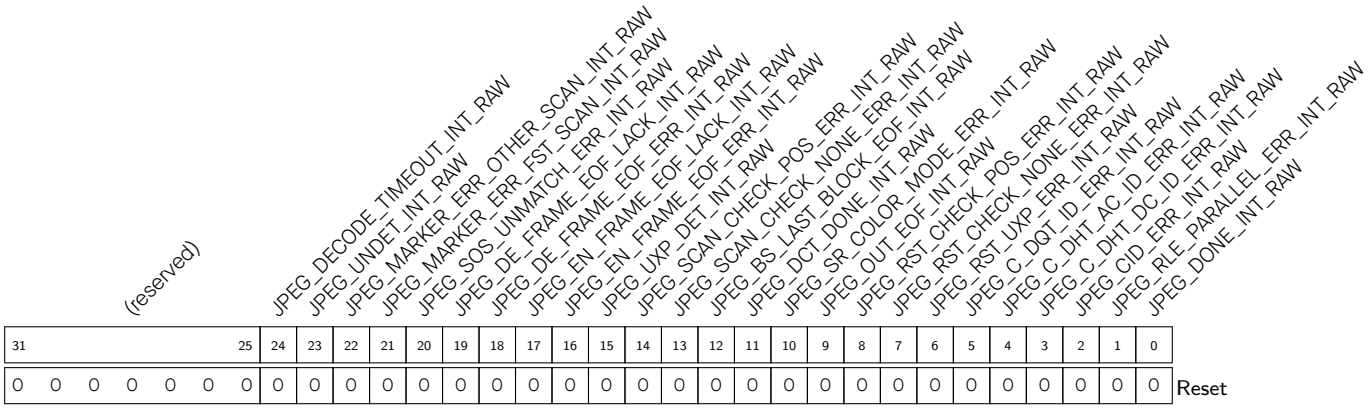
JPEG_DC0_DHT_ID Configures the ID of DC0 Huffman table in decoder mode. (R/W)

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 28.14. JPEG_INT_RAW_REG (0x0038)



JPEG_DONE_INT_RAW The raw status of JPEG_DONE_INT. (R/WTC/SS)

JPEG_RLE_PARALLEL_ERR_INT_RAW The raw status of JPEG_RLE_PARALLEL_ERR_INT. (R/WTC/SS)

JPEG_CID_ERR_INT_RAW The raw status of JPEG_CID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_DC_ID_ERR_INT_RAW The raw status of JPEG_C_DHT_DC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DHT_AC_ID_ERR_INT_RAW The raw status of JPEG_C_DHT_AC_ID_ERR_INT. (R/WTC/SS)

JPEG_C_DQT_ID_ERR_INT_RAW The raw status of JPEG_C_DQT_ID_ERR_INT. (R/WTC/SS)

JPEG_RST_UXP_ERR_INT_RAW The raw status of JPEG_RST_UXP_ERR_INT. (R/WTC/SS)

JPEG_RST_CHECK_NONE_ERR_INT_RAW The raw status of JPEG_RST_CHECK_NONE_ERR_INT. (R/WTC/SS)

Continued on the next page...

Register 28.14. JPEG_INT_RAW_REG (0x0038)

Continued from the previous page...

JPEG_RST_CHECK_POS_ERR_INT_RAW The raw status of JPEG_RST_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_OUT_EOF_INT_RAW The raw status of JPEG_OUT_EOF_INT. (R/WTC/SS)

JPEG_SR_COLOR_MODE_ERR_INT_RAW The raw status of JPEG_SR_COLOR_MODE_ERR_INT. This interrupt is invalid. (R/WTC/SS)

JPEG_DCT_DONE_INT_RAW The raw status of JPEG_DCT_DONE_INT. (R/WTC/SS)

JPEG_BS_LAST_BLOCK_EOF_INT_RAW The raw status of JPEG_BS_LAST_BLOCK_EOF_INT. (R/WTC/SS)

JPEG_SCAN_CHECK_NONE_ERR_INT_RAW The raw status of JPEG_SCAN_CHECK_NONE_ERR_INT. (R/WTC/SS)

JPEG_SCAN_CHECK_POS_ERR_INT_RAW The raw status of JPEG_SCAN_CHECK_POS_ERR_INT. (R/WTC/SS)

JPEG_UXP_DET_INT_RAW The raw status of JPEG_UXP_DET_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_ERR_INT_RAW The raw status of JPEG_EN_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_EN_FRAME_EOF_LACK_INT_RAW The raw status of JPEG_EN_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_ERR_INT_RAW The raw status of JPEG_DE_FRAME_EOF_ERR_INT. (R/WTC/SS)

JPEG_DE_FRAME_EOF_LACK_INT_RAW The raw status of JPEG_DE_FRAME_EOF_LACK_INT. (R/WTC/SS)

JPEG_SOS_UNMATCH_ERR_INT_RAW The raw status of JPEG_SOS_UNMATCH_ERR_INT. (R/WTC/SS)

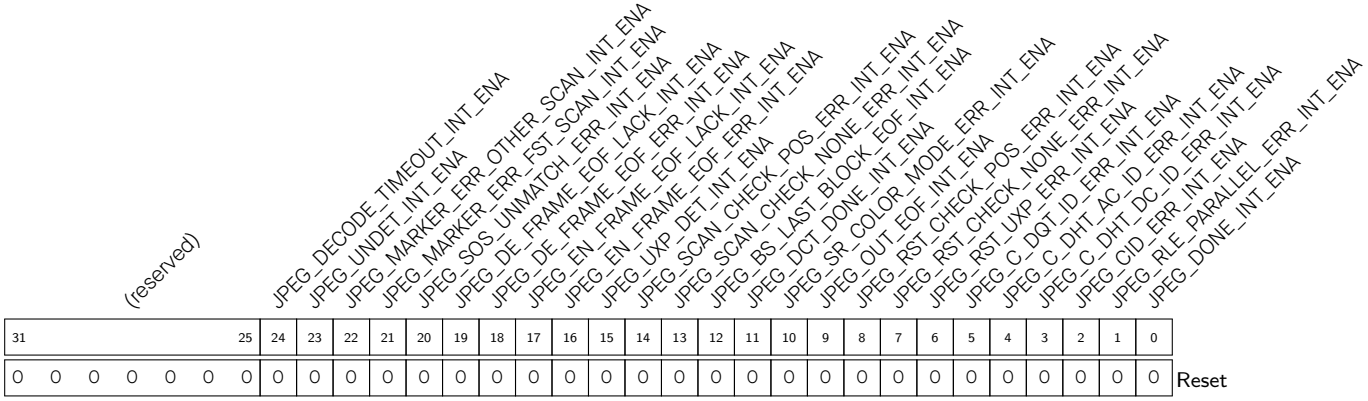
JPEG_MARKER_ERR_FST_SCAN_INT_RAW The raw status of JPEG_MARKER_ERR_FST_SCAN_INT. (R/WTC/SS)

JPEG_MARKER_ERR_OTHER_SCAN_INT_RAW The raw status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/WTC/SS)

JPEG_UNDET_INT_RAW The raw status of JPEG_UNDET_INT. (R/WTC/SS)

JPEG_DECODE_TIMEOUT_INT_RAW The raw status of JPEG_DECODE_TIMEOUT_INT. (R/WTC/SS)

Register 28.15. JPEG_INT_ENA_REG (0x003C)



JPEG_DONE_INT_ENA Write 1 to enable JPEG_DONE_INT_ENA. (R/W)

JPEG_RLE_PARALLEL_ERR_INT_ENA Write 1 to enable JPEG_RLE_PARALLEL_ERR_INT. (R/W)

JPEG_CID_ERR_INT_ENA Write 1 to enable JPEG_CID_ERR_INT. (R/W)

JPEG_C_DHT_DC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_DC_ID_ERR_INT. (R/W)

JPEG_C_DHT_AC_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DHT_AC_ID_ERR_INT. (R/W)

JPEG_C_DQT_ID_ERR_INT_ENA Write 1 to enable JPEG_C_DQT_ID_ERR_INT. (R/W)

JPEG_RST_UXP_ERR_INT_ENA Write 1 to enable JPEG_RST_UXP_ERR_INT. (R/W)

JPEG_RST_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_NONE_ERR_INT. (R/W)

JPEG_RST_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_RST_CHECK_POS_ERR_INT. (R/W)

JPEG_OUT_EOF_INT_ENA Write 1 to enable JPEG_OUT_EOF_INT. (R/W)

JPEG_SR_COLOR_MODE_ERR_INT_ENA Write 1 to enable JPEG_SR_COLOR_MODE_ERR_INT. (R/W)

JPEG_DCT_DONE_INT_ENA Write 1 to enable JPEG_DCT_DONE_INT. (R/W)

JPEG_BS_LAST_BLOCK_EOF_INT_ENA Write 1 to enable JPEG_BS_LAST_BLOCK_EOF_INT. (R/W)

JPEG_SCAN_CHECK_NONE_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_NONE_ERR_INT. (R/W)

JPEG_SCAN_CHECK_POS_ERR_INT_ENA Write 1 to enable JPEG_SCAN_CHECK_POS_ERR_INT. (R/W)

JPEG_UXP_DET_INT_ENA Write 1 to enable JPEG_UXP_DET_INT. (R/W)

Continued on the next page...

Register 28.15. JPEG_INT_ENA_REG (0x003C)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_ERR_INT. (R/W)

JPEG_EN_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_EN_FRAME_EOF_LACK_INT. (R/W)

JPEG_DE_FRAME_EOF_ERR_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_ERR_INT. (R/W)

JPEG_DE_FRAME_EOF_LACK_INT_ENA Write 1 to enable JPEG_DE_FRAME_EOF_LACK_INT. (R/W)

JPEG_SOS_UNMATCH_ERR_INT_ENA Write 1 to enable JPEG_SOS_UNMATCH_ERR_INT. (R/W)

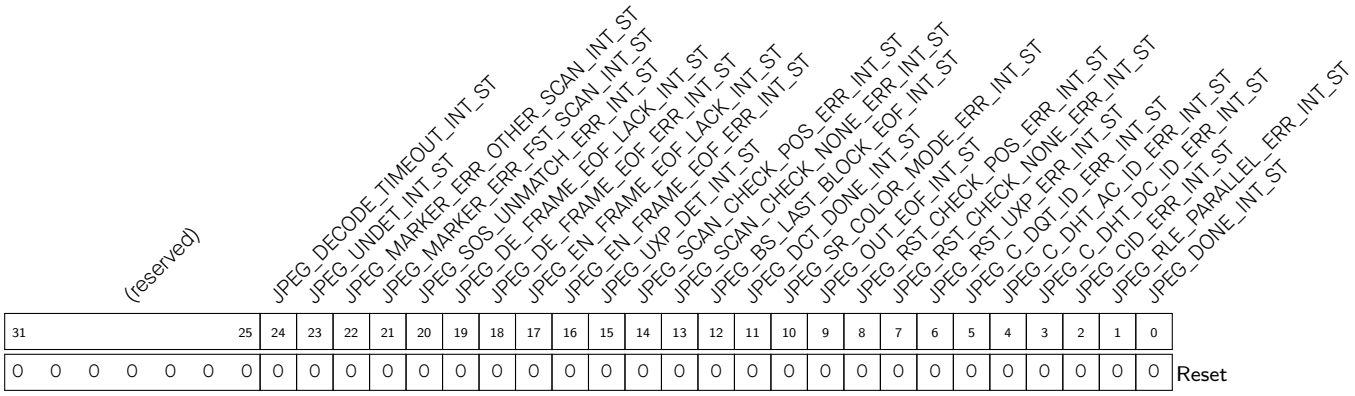
JPEG_MARKER_ERR_FST_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_FST_SCAN_INT. (R/W)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ENA Write 1 to enable JPEG_MARKER_ERR_OTHER_SCAN_INT. (R/W)

JPEG_UNDET_INT_ENA Write 1 to enable JPEG_UNDET_INT. (R/W)

JPEG_DECODE_TIMEOUT_INT_ENA Write 1 to enable JPEG_DECODE_TIMEOUT_INT. (R/W)

Register 28.16. 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 28.16. JPEG_INT_ST_REG (0x0040)

Continued from the previous page...

JPEG_EN_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_ERR_INT. (RO)

JPEG_EN_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_EN_FRAME_EOF_LACK_INT. (RO)

JPEG_DE_FRAME_EOF_ERR_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_ERR_INT. (RO)

JPEG_DE_FRAME_EOF_LACK_INT_ST The masked interrupt status of JPEG_DE_FRAME_EOF_LACK_INT. (RO)

JPEG_SOS_UNMATCH_ERR_INT_ST The masked interrupt status of JPEG_SOS_UNMATCH_ERR_INT. (RO)

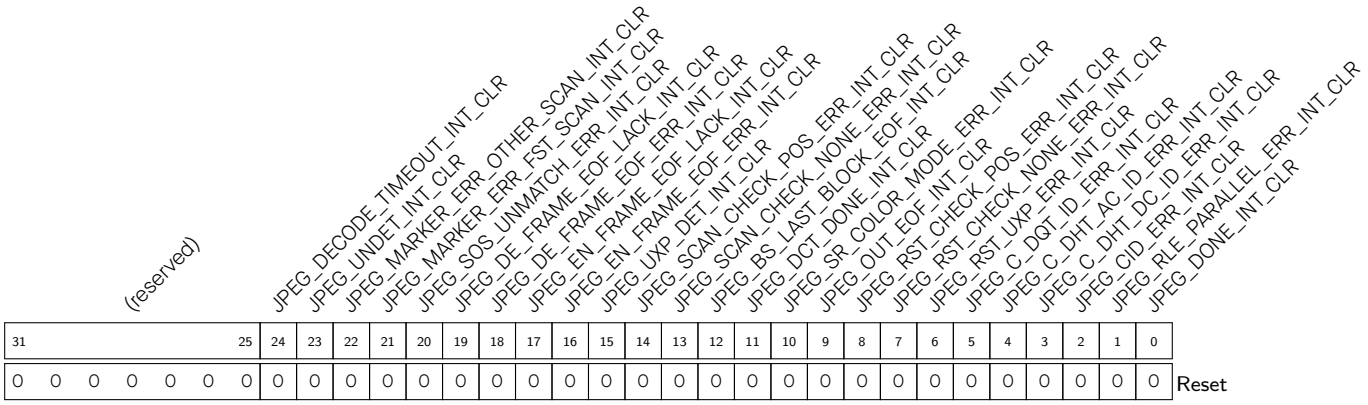
JPEG_MARKER_ERR_FST_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_FST_SCAN_INT. (RO)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ST The masked interrupt status of JPEG_MARKER_ERR_OTHER_SCAN_INT. (RO)

JPEG_UNDET_INT_ST The masked interrupt status of JPEG_UNDET_INT. (RO)

JPEG_DECODE_TIMEOUT_INT_ST The masked interrupt status of JPEG_DECODE_TIMEOUT_INT. (RO)

Register 28.17. JPEG_INT_CLR_REG (0x0044)



JPEG_DONE_INT_CLR Write 1 to clear JPEG_DONE_INT. (WT)

JPEG_RLE_PARALLEL_ERR_INT_CLR Write 1 to clear JPEG_RLE_PARALLEL_ERR_INT. (WT)

JPEG_CID_ERR_INT_CLR Write 1 to clear JPEG_CID_ERR_INT. (WT)

JPEG_C_DHT_DC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_DC_ID_ERR_INT. (WT)

JPEG_C_DHT_AC_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DHT_AC_ID_ERR_INT. (WT)

JPEG_C_DQT_ID_ERR_INT_CLR Write 1 to clear JPEG_C_DQT_ID_ERR_INT. (WT)

JPEG_RST_UXP_ERR_INT_CLR Write 1 to clear JPEG_RST_UXP_ERR_INT. (WT)

JPEG_RST_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_NONE_ERR_INT. (WT)

JPEG_RST_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_RST_CHECK_POS_ERR_INT. (WT)

JPEG_OUT_EOF_INT_CLR Write 1 to clear JPEG_OUT_EOF_INT. (WT)

JPEG_SR_COLOR_MODE_ERR_INT_CLR Write 1 to clear JPEG_SR_COLOR_MODE_ERR_INT. (WT)

JPEG_DCT_DONE_INT_CLR Write 1 to clear JPEG_DCT_DONE_INT. (WT)

JPEG_BS_LAST_BLOCK_EOF_INT_CLR Write 1 to clear JPEG_BS_LAST_BLOCK_EOF_INT. (WT)

JPEG_SCAN_CHECK_NONE_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_NONE_ERR_INT. (WT)

JPEG_SCAN_CHECK_POS_ERR_INT_CLR Write 1 to clear JPEG_SCAN_CHECK_POS_ERR_INT. (WT)

Continued on the next page...

Register 28.17. JPEG_INT_CLR_REG (0x0044)

Continued from the previous page...

JPEG_UXP_DET_INT_CLR Write 1 to clear JPEG_UXP_DET_INT. (WT)

JPEG_EN_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_ERR_INT. (WT)

JPEG_EN_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_EN_FRAME_EOF_LACK_INT. (WT)

JPEG_DE_FRAME_EOF_ERR_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_ERR_INT. (WT)

JPEG_DE_FRAME_EOF_LACK_INT_CLR Write 1 to clear JPEG_DE_FRAME_EOF_LACK_INT. (WT)

JPEG_SOS_UNMATCH_ERR_INT_CLR Write 1 to clear JPEG_SOS_UNMATCH_ERR_INT. (WT)

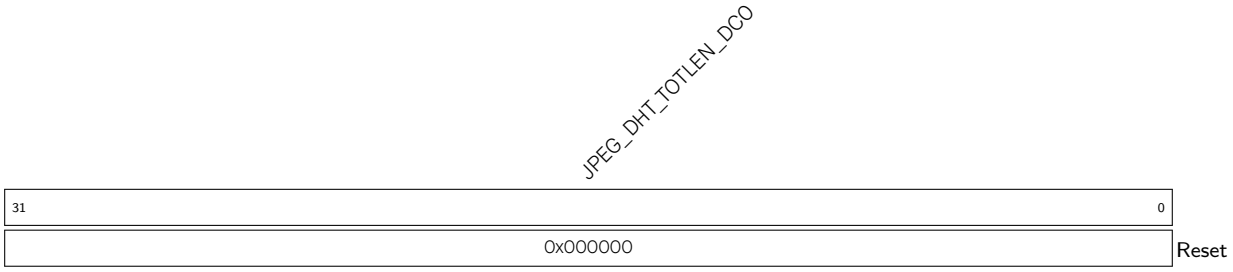
JPEG_MARKER_ERR_FST_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_FST_SCAN_INT. (WT)

JPEG_MARKER_ERR_OTHER_SCAN_INT_CLR Write 1 to clear JPEG_MARKER_ERR_OTHER_SCAN_INT. (WT)

JPEG_UNDET_INT_CLR Write 1 to clear JPEG_UNDET_INT. (WT)

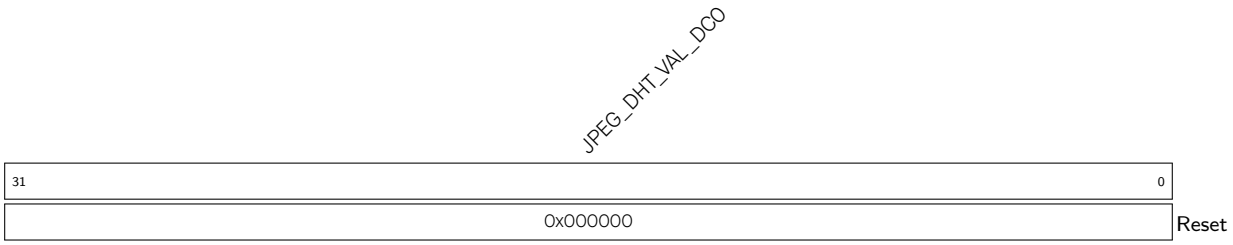
JPEG_DECODE_TIMEOUT_INT_CLR Write 1 to clear JPEG_DECODE_TIMEOUT_INT. (WT)

Register 28.18. JPEG_DHT_TOTLEN_DCO_REG (0x0058)



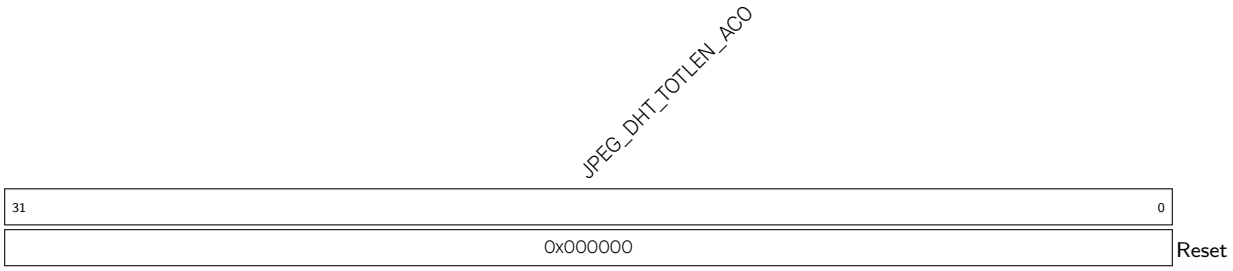
JPEG_DHT_TOTLEN_DCO Configures the number of codeword with a codeword length of 1 to 16 bits for DCO Huffman table in FIFO mode. (HRO)

Register 28.19. JPEG_DHT_VAl_DC0_REG (0x005C)



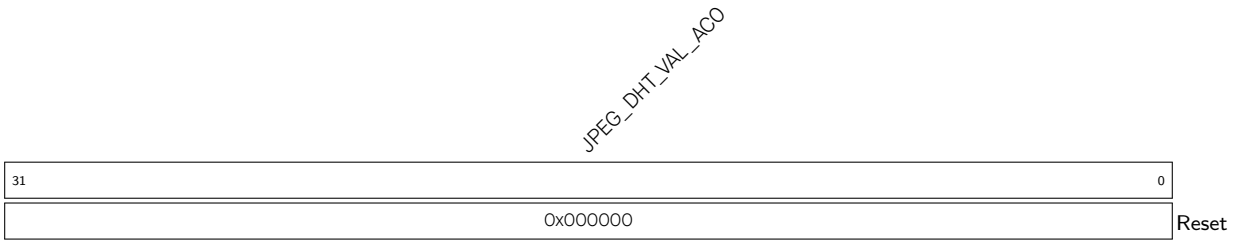
JPEG_DHT_VAL_DC0 Configures the symbols corresponding to the decoded codeword for DC0 Huffman table in FIFO mode. (HRO)

Register 28.20. JPEG_DHT_TOTLEN_AC0_REG (0x0060)



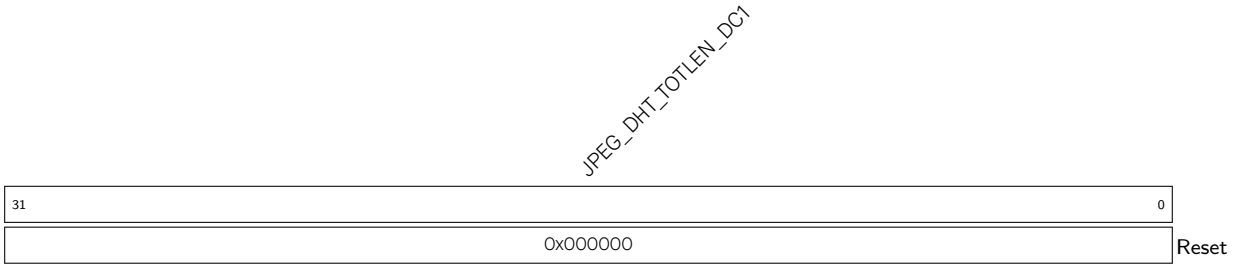
JPEG_DHT_TOTLEN_AC0 Configures the number of codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. (HRO)

Register 28.21. JPEG_DHT_VAl_AC0_REG (0x0064)



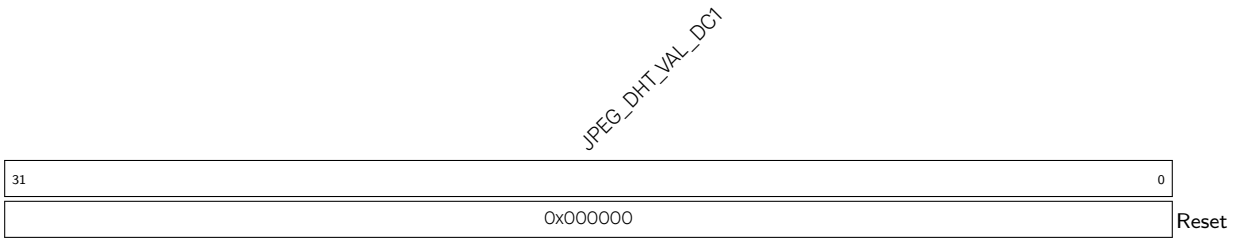
JPEG_DHT_VAL_AC0 Configures the symbols corresponding to the decoded codeword for AC0 Huffman table in FIFO mode. (HRO)

Register 28.22. JPEG_DHT_TOTLEN_DC1_REG (0x0068)



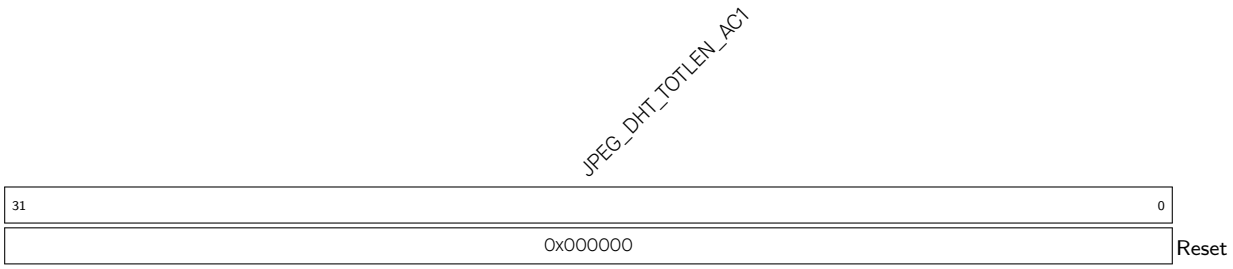
JPEG_DHT_TOTLEN_DC1 Configures the number of codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. (HRO)

Register 28.23. JPEG_DHT_VAl_DC1_REG (0x006C)



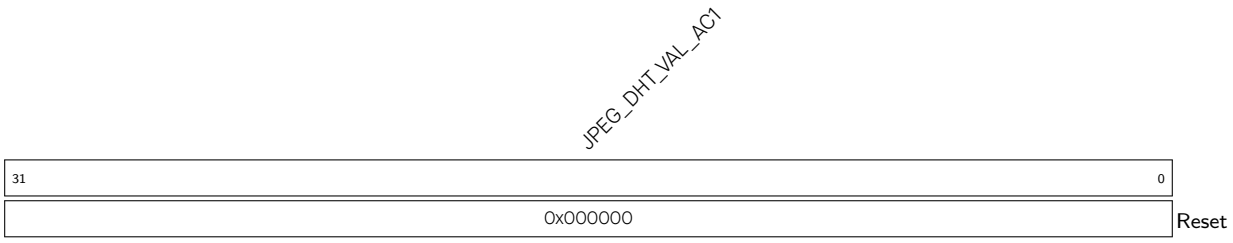
JPEG_DHT_VAL_DC1 Configures the symbols corresponding to the decoded codeword for DC1 Huffman table in FIFO mode. (HRO)

Register 28.24. JPEG_DHT_TOTLEN_AC1_REG (0x0070)



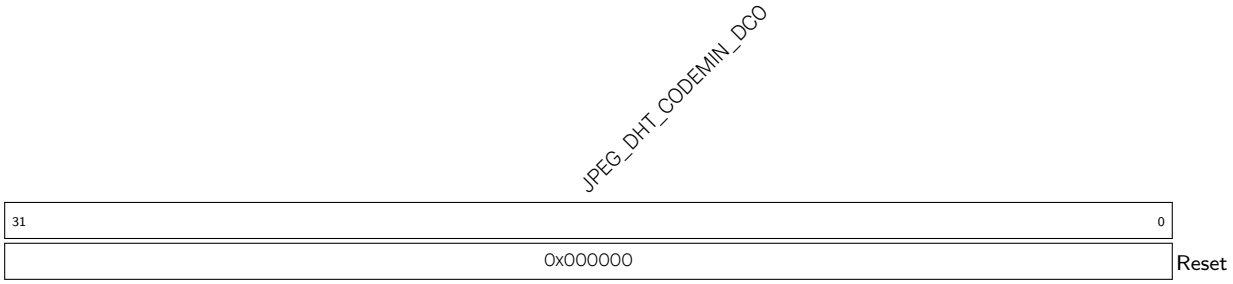
JPEG_DHT_TOTLEN_AC1 Configures the number of codeword with a codeword length of 1 to 16 bits for AC1 Huffman table in FIFO mode. (HRO)

Register 28.25. JPEG_DHT_VAl_AC1_REG (0x0074)



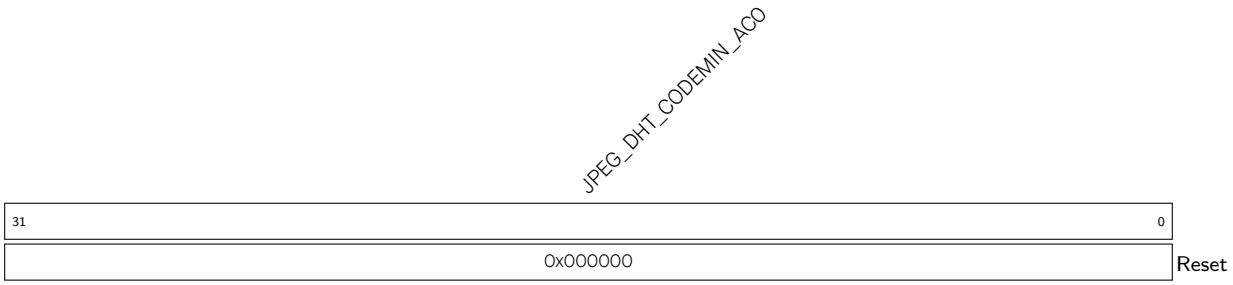
JPEG_DHT_VAL_AC1 Configures the symbols corresponding to the decoded codeword for AC1 Huffman table in FIFO mode. (HRO)

Register 28.26. JPEG_DHT_CODEMIN_DC0_REG (0x0078)



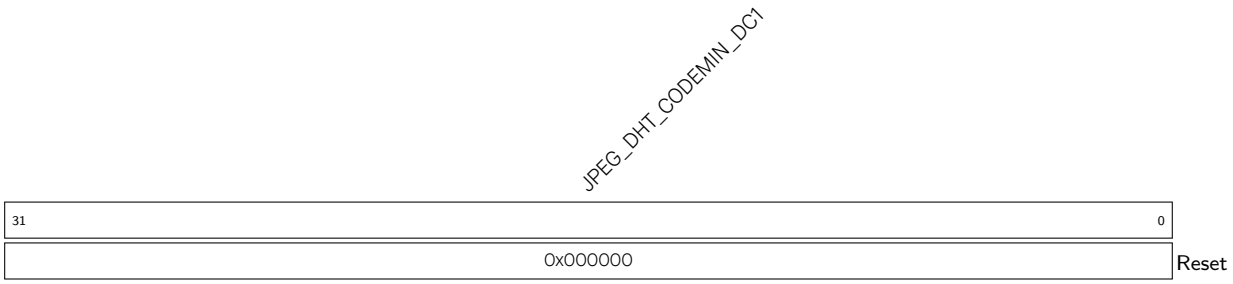
JPEG_DHT_CODEMIN_DC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 28.27. JPEG_DHT_CODEMIN_AC0_REG (0x007C)



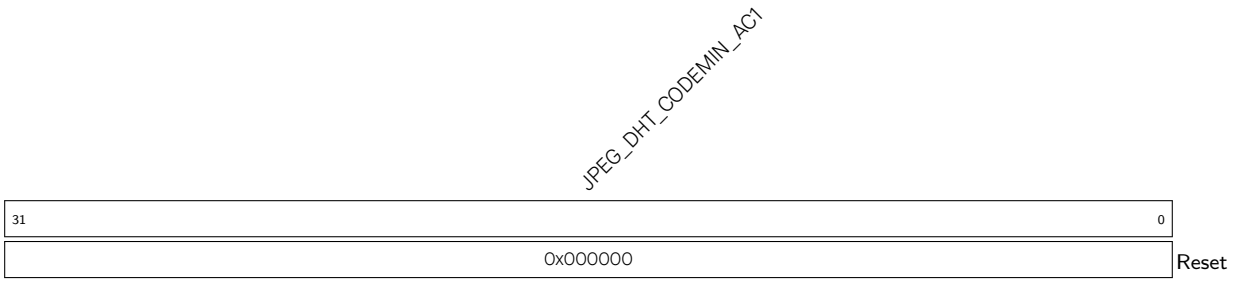
JPEG_DHT_CODEMIN_AC0 Configures the minimum codeword with a codeword length of 1 to 16 bits for AC0 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 28.28. JPEG_DHT_CODEMIN_DC1_REG (0x0080)



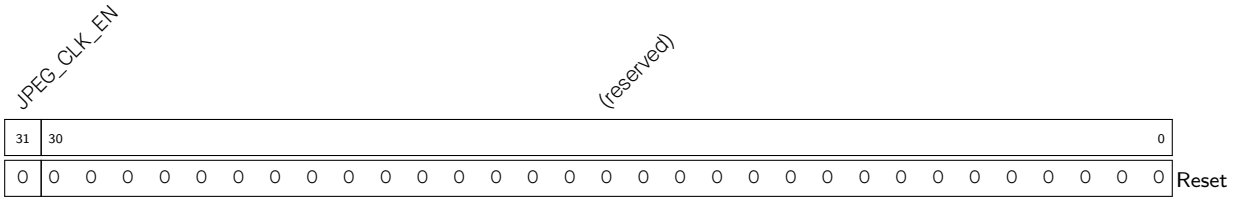
JPEG_DHT_CODEMIN_DC1 Configures the minimum codeword with a codeword length of 1 to 16 bits for DC1 Huffman table in FIFO mode. The codeword should be left shifted to the MSB position of a 16-bit word. (HRO)

Register 28.29. 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 28.30. JPEG_SYS_REG (0x00F8)



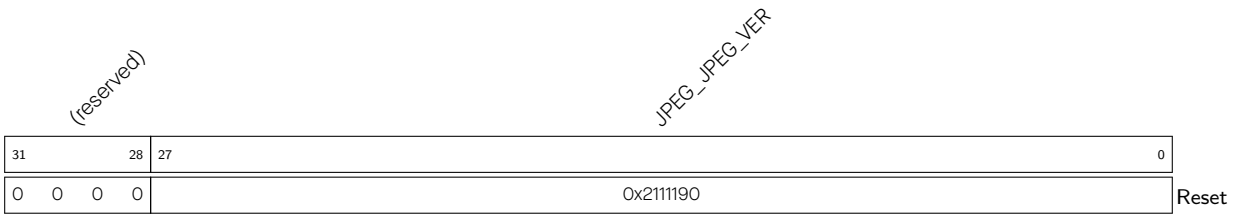
JPEG_CLK_EN Configures whether or not to force open register clock gate.

0: Open the clock gate only when the application writes registers

1: Force open the clock gate for register

(R/W)

Register 28.31. JPEG_VERSION_REG (0x00FC)



JPEG_JPEG_VER Version control. (R/W)

Chapter 29

Pixel-Processing Accelerator (PPA)

29.1 Overview

ESP32-P4 includes a pixel-processing accelerator (PPA) designed for 2D image display. This module realizes hardware-level acceleration of image algorithms and implements functions such as image rotation, scaling, mirroring, and blending.

PPA consists of two functional modules: scaling-rotation-mirroring (SRM) and image blending (BLEND). The SRM module implements image rotation, scaling, and XY-axis mirroring functions, while the BLEND module achieves the blending of two layers of the same size based on the Alpha channel, i.e., transparency.

PPA processes images in units of pixel block. These pixel blocks are acquired from memory through 2D-DMA. The connection between PPA and 2D-DMA is shown in Figure 29.1-1.

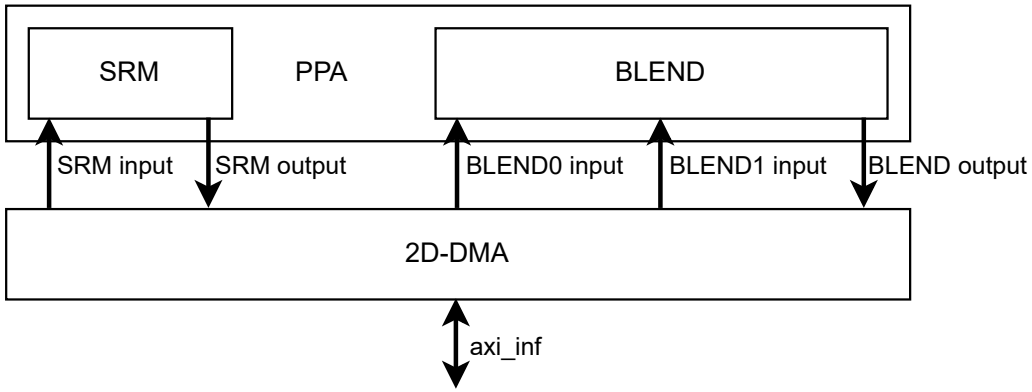


Figure 29.1-1. PPA 2D-DMA Connection

29.2 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.

29.3 Features

- Image rotation, scaling, and mirroring by SRM:

  - Input formats: ARGB8888, RGB888, RGB565, YUV420
  - Output formats: ARGB8888, RGB888, RGB565, YUV420
  - Counterclockwise rotation angles: 0°, 90°, 180°, 270°
  - Horizontal and vertical scaling with scaling factors of 4-bit integer part and 8-bit fractional part
  - Horizontal and vertical mirroring
- Blending two layers of the same size and filling images with specific pixels by BLEND:
  - Input formats: ARGB8888, RGB888, RGB565, L4, L8, A4, A8
  - Output formats: ARGB8888, RGB888, RGB565
  - 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

29.4 Architectural Overview

As shown in figure 29.4-1, there are two independent functional modules in PPA: SRM and BLEND.

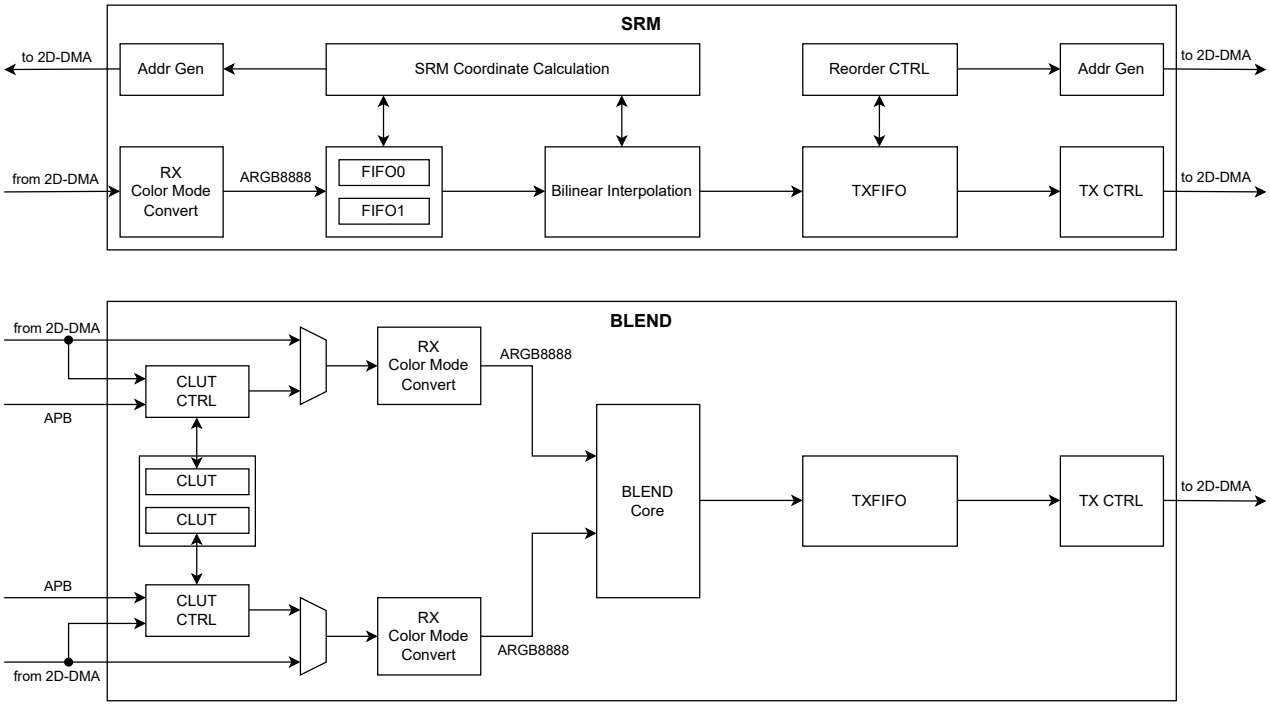


Figure 29.4-1. PPA Architecture

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.

29.5 Functional Description

29.5.1 PPA Color Spaces

PPA supports both true color and pseudo color. True color refers to a color representation in which each pixel in a color image consists of three primary color components: R (red), G (green), and B (blue). Each of these components directly determines the intensity of the corresponding primary color on the display device. The true color formats supported by PPA include ARGB8888, RGB888, and RGB565.

Pseudo color refers to a color representation in which the color of each pixel in a color image is not directly determined by the numerical values of the primary color components. Instead, the pixel value serves as an entry into a Color Look-Up Table (CLUT) to retrieve the corresponding A, R, G, and B values. PPA supports CLUT entry addresses with 8-bit or 4-bit width.

In addition, SRM supports the YUV420 data format. The storage order of supported color formats in memory is shown in Table 29.5-1.

Table 29.5-1. 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]
YUV420	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
Odd rows	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]
such as 1, 3, 5...	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
YUV420	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
Even rows	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]
such as 2, 4, 6...	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]

29.5.1.1 SRM Color Space

SRM supports ARGB8888, RGB888, RGB565, and YUV420 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 29.5-1.

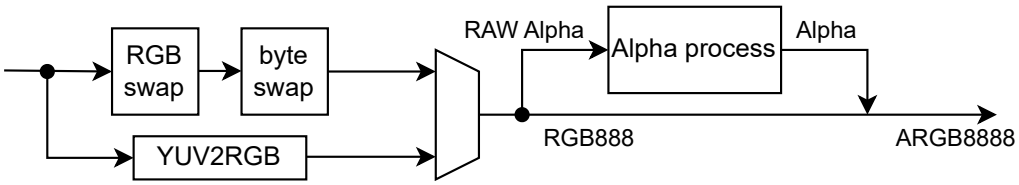


Figure 29.5-1. PPA SRM Input Pixel Processing

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 29.5-2. For specific configuration effects, please refer to Table 29.5-2.

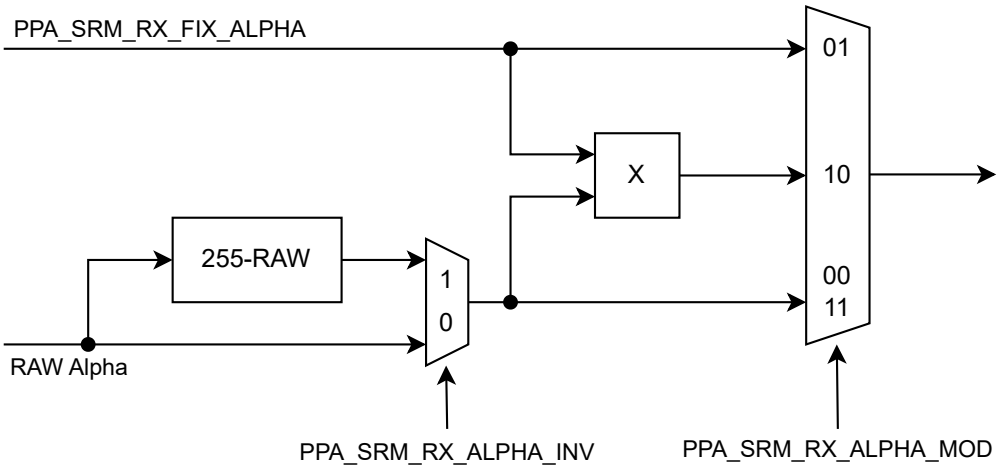


Figure 29.5-2. PPA SRM Alpha Configuration Logic

Table 29.5-2. 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

Continued on the next page...

Table 29.5-2 – ...Continued from the previous page

PPA_SRM_RX_ALPHA_INV	PPA_SRM_RX_ALPHA_MOD	Output
	10	The high 8 bits of the value obtained by multiplying the original Alpha value with PPA_SRM_RX_FIX_ALPHA
1	00/11	The value obtained by subtracting the original Alpha value from 255
	01	The configuration value of PPA_SRM_RX_FIX_ALPHA
	10	The high 8 bits of the value obtained by multiplying the result of subtracting the original Alpha value from 255 by PPA_SRM_RX_FIX_ALPHA

Note that when the input or output format is YUV, further configuration of YUV to RGB color conversion is required.

When the input format is YUV420, the input YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_RX_RANGE field, and the protocol used for YUV to RGB conversion needs to be configured as BT601 or BT709 via the PPA_YUV2RGB_PROTOCAL field. The conversion formulas are as follows:

- YUV_{full} to YUV_{limit}
  - $Y_{limit} = \frac{220}{256}Y_{full} + 16$
  - $U_{limit} = \frac{225}{256}U_{full} + 16$
  - $V_{limit} = \frac{225}{256}V_{full} + 16$
- YUV_{limit} to RGB in BT601
  - $R = \frac{298}{256}Y_{limit} + \frac{409}{256}V_{limit} - \frac{56906}{256}$
  - $G = \frac{298}{256}Y_{limit} - \frac{100}{256}U_{limit} - \frac{208}{256}V_{limit} + \frac{34707}{256}$
  - $B = \frac{298}{256}Y_{limit} + \frac{516}{256}U_{limit} - \frac{70836}{256}$
- YUV_{limit} to RGB in BT709
  - $R = \frac{298}{256}Y_{limit} + \frac{459}{256}V_{limit} - \frac{63367}{256}$
  - $G = \frac{298}{256}Y_{limit} - \frac{55}{256}U_{limit} - \frac{136}{256}V_{limit} + \frac{19681}{256}$
  - $B = \frac{298}{256}Y_{limit} + \frac{541}{256}U_{limit} - \frac{73918}{256}$

When the output format is YUV420, the output YUV data range needs to be configured as full-range or limit-range via the PPA_YUV_TX_RANGE field, and the protocol used for YUV to RGB conversion needs to be configured as BT601 or BT709 via the PPA_RGB2YUV_PROTOCAL field. The conversion formulas are as follows:

- YUV_{limit} to YUV_{full}
  - $Y_{full} = \frac{298}{256}Y_{limit} + \frac{4768}{256}$
  - $U_{full} = \frac{291}{256}U_{limit} + \frac{4550}{256}$
  - $V_{full} = \frac{291}{256}V_{limit} + \frac{4550}{256}$
- RGB to YUV_{limit} in BT601

  - $Y_{limit} = \frac{4096}{256} + \frac{66}{256}R + \frac{129}{256}G + \frac{25}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{38}{256}R - \frac{74}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{94}{256}G - \frac{18}{256}B$
- RGB to YUV_{limit} in BT709
  - $Y_{limit} = \frac{4096}{256} + \frac{47}{256}R + \frac{157}{256}G + \frac{16}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{26}{256}R - \frac{86}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{102}{256}G - \frac{10}{256}B$

29.5.1.2 BLEND Color Space

For the input format, BLEND supports ARGB8888, RGB888, RGB565, L4, L8, A4, and A8. The input color format of the foreground and background layers can be configured respectively via the PPA_BLEND0/1_RX_CM field. Among the above input formats, A4 and A8 only support the foreground layer, and when the input format is L4 or A4, the size of the image block, hb, and the offset in the image, x, must be an even number. For the output format, BLEND supports ARGB8888, RGB888, and RGB565, which can be configured via the PPA_BLEND_TX_CM field.

BLEND's RGB conversion, byte conversion, and alpha channel configuration are consistent with SRM input. Additionally, when the foreground layer data format is A4 or A8, the input data will be Alpha channel values. In this case, if PPA_BLEND_BYPASS is set as 1, the foreground layer's input Alpha value replaces the background layer's Alpha channel and output. If set as 0, it will be a normal BLEND operation, and the foreground layer's RGB values will be specified by PPA_BLEND_RGB_REG.

When the input data format is L4 or L8, it requires to first initialize the CLUT for BLEND. BLEND has two CLUTs with a depth of 256 and a width of 32. There are two initialization modes: FIFO and MEM.

- Write 0 to PPA_APB_FIFO_MASK to enter the FIFO mode.
  In the FIFO mode, access to CLUT is achieved by reading/writing PPA_RDWR_WORD_BLENDx_CLUT, where 0 corresponds to the background channel CLUT and 1 corresponds to the foreground channel CLUT.
  Each time this register is read/written, the corresponding read/write address is automatically incremented by 1. The read/write address can be reset by writing 1 and then 0 to PPA_BLENDx_CLUT_MEM_RST. The read address can be reset by writing 1 then 0 to PPA_BLENDx_CLUT_MEM_RDADDR_RST.
- Write 1 to PPA_APB_FIFO_MASK to enter the MEM mode.
  In the MEM mode, CLUT can be directly accessed through addresses. The CLUT addresses in the PPA's bus address range are as shown in Table 29.5-3.

Table 29.5-3. 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

29.5.2 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 29.5-3. For detailed information, please refer to *4 2D-DMA Controller (2D-DMA)*.

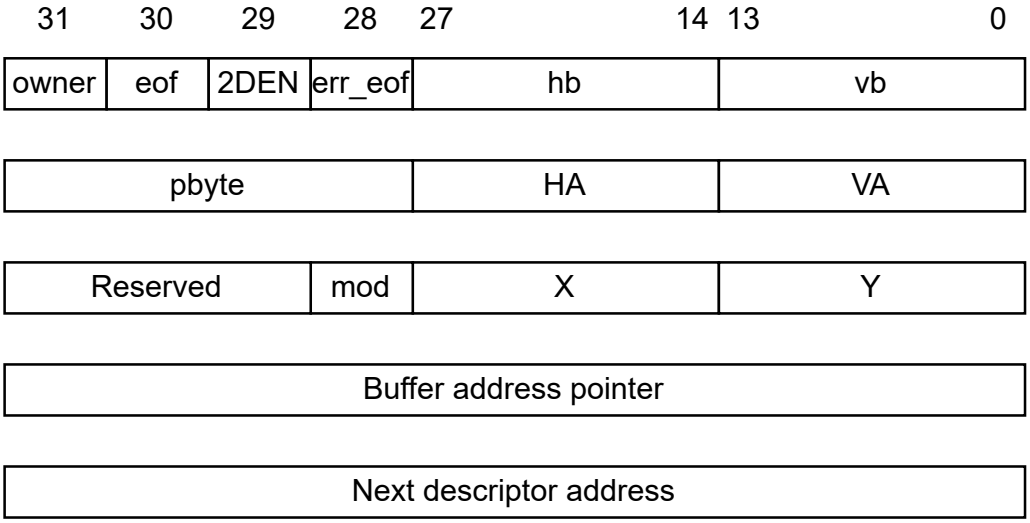


Figure 29.5-3. 2D-DMA Linked List

The corresponding linked list parameters of PPA are shown in Table 29.5-4.

Table 29.5-4. PPA Linked List Configuration

Field	Configuration
owner	1
eof	1
2DEN	1
err_eof	0
hb	Horizontal size of the image block. Must be even if the input or output data format is YUV420
vb	Vertical size of the image block. Must be even if the input or output data format is YUV420
pbyte	ARGB8888: 5 (4 byte/pixel) RGB888: 4 (3 byte/pixel) RGB565: 3 (2 byte/pixel) YUV420: 2 (1.5 byte/pixel) A8/L8: 1 (1 byte/pixel) A4/L4: 0 (0.5 byte/pixel)

Continued on the next page...

Table 29.5-4 – ...Continued from the previous page

Field	Configuration
HA	Horizontal size of the entire image. Must be even if the input or output data format is YUV420
VA	Vertical size of the entire image. Must be even if the input or output data format is YUV420
mod	0
X	Horizontal offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV420
Y	Vertical offset of the image block within the image (offset starts from 0). Must be even if the input or output data is YUV420
Buffer address pointer	Starting address of the image
Next descriptor address	N/A

29.5.2.1 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 pixel block size is 18x18 when the input data is ARGB8888, RGB888, or RGB565; and 20x20 when the input data is YUV420.

29.5.3 Scaling - Rotation - Mirroring (SRM)

29.5.3.1 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 29.5-4. 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

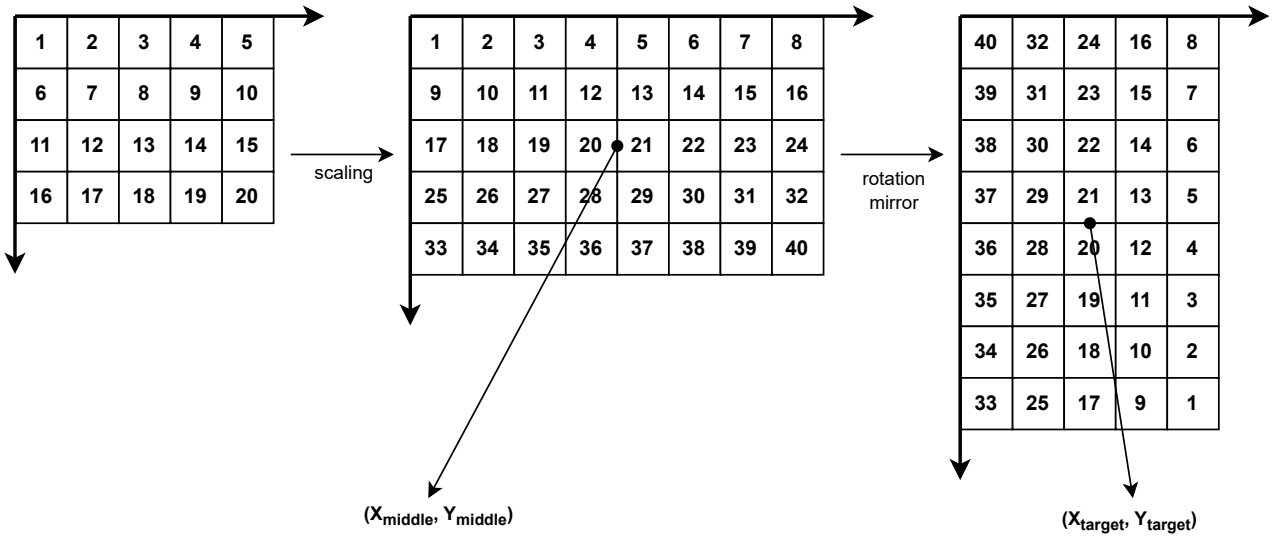


Figure 29.5-4. SRM Procedure

29.5.3.2 Pixel Block Rearrangement

When the output pixel block is small, initiating a 2D-DMA transfer for each pixel block would result in low bus bandwidth utilization. To improve bus bandwidth utilization, SRM can cache and concatenate output pixel blocks into a larger pixel block before outputting them all at once. This feature is enabled by default and can be disabled by setting PPA_SRM_MACRO_BK_RO_BYPASS to 1.

Once pixel block rearrangement is enabled, under certain conditions, the output pixel blocks will be automatically rearranged and concatenated, as specified in Table 29.5-5.

- For the output pixel block in ARGB8888/RGB888 format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 16, pixel block rearrangement will be automatically enabled.
- For the output pixel block in RGB565 format, if its vertical height is less than or equal to 32, and the horizontal width is less than or equal to 32, pixel block rearrangement will be automatically enabled.
- For the output pixel block in YUV420 format, if its vertical height is less than or equal to 32, and horizontal width is less than or equal to 32, pixel block rearrangement will be automatically enabled.

Table 29.5-5. Number of Pixel Block Rearrangements in SRM Output

ARGB8888/RGB888		RGB565		YUV420	
Horizontal Size	Rearrangemer Number	Horizontal Size	Rearrangemer Number	Horizontal Size (Even)	Rearrangement Number
1	32	1	64	2	32
2	16	2	32	4	16
3	10	3	20	6	10
4	8	4	16	8	8
5	6	5	12	10	6
6	5	6	10	12	5
7	4	7	9	14	4
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		

29.5.4 Layer Blending (BLEND)

29.5.4.1 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.

29.5.4.2 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.

29.6 Interrupts

ESP32-P4's PPA can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- PPA_INTR

The following interrupt sources can generate PPA_INTR interrupt signal:

- PPA_SRM_PARAM_CFG_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 29.6-1.
- PPA_SRM_EOF_INT: Triggered when SRM workflow reaches EOF.
- PPA_BLEND_EOF_INT:: Triggered when BLEND workflow reaches EOF.

Table 29.6-1. SRM Parameter Error Type

Error Type	Detail
PPA_TX_DSCR_VB_ERR_ST	The sum of the vertical size of the output image block from SRM and the vertical offset Y specified in the output descriptor exceeds the total vertical size of the image configured in the output descriptor (VA)
PPA_TX_DSCR_HB_ERR_ST	The sum of the horizontal size of the output image block from SRM and the vertical offset X specified in the output descriptor exceeds the total horizontal size of the image configured in the output descriptor (HA)
PPA_Y_RX_SCAL_EQUAL_0_ERR_ST	The vertical scaling factor for the input image block in SRM is set to 0
PPA_RX_DSCR_VB_ERR_ST	The sum of the vertical size VB and the vertical offset Y in the SRM input descriptor exceeds the overall vertical size VA in the output descriptor

Continued on the next page...

Table 29.6-1 – ...Continued from the previous page

Error Type	Detail
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 of the input image block configured in the SRM input descriptor and the horizontal offset X exceeds the entire image's horizontal size HA configured in the output descriptor
PPA_XDST_LEN_TOO_SAMLL_ERR_ST	The horizontal size of the image block after SRM scaling is 0
PPA_XDST_LEN_TOO_LARGE_ERR_ST	The horizontal size of the image block after SRM scaling exceeds 8191
PPA_X_YUV420_RX_SCALE_ERR_ST	The input descriptor parameters HA/hb/X happen to be odd numbers when the SRM input format is YUV420
PPA_Y_YUV420_RX_SCALE_ERR_ST	The input descriptor parameters VA/vb/Y happen to be odd numbers when the SRM input format is YUV420
PPA_X_YUV420_TX_SCALE_ERR_ST	The output descriptor parameters HA/hb/X happen to be odd numbers when the SRM output format is YUV420
PPA_Y_YUV420_TX_SCALE_ERR_ST	The output descriptor parameters VA/vb/Y happen to be odd numbers when the SRM output format is YUV420

29.7 Programming Procedures

29.7.1 PPA Clock Reset Configuration

1. Enable the clock of the PPA and 2D-DMA modules:
   - Write 1 to HP_SYS_CLKRST_PPA_SYS_CLK_EN to enable the PPA clock
   - Write 1 to HP_SYS_CLKRST_DMA2D_SYS_CLK_EN to enable the 2D-DMA clock
2. Reset the PPA and 2D-DMA modules:
   - Write 1 and then 0 to HP_SYS_CLKRST_RST_EN_PPA reset PPA
   - Write 1 and then 0 to HP_SYS_CLKRST_RST_EN_DMA2D reset 2D-DMA

29.7.2 SRM Configuration

1. Refer to Section 29.7.1 to enable and reset PPA and 2D-DMA.
2. Configure the inlink and outlink of 2D-DMA. For detailed parameters, please refer to Table 29.5-4.
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 to 20 for YUV420, or to 18 for other color modes
   - Configure the vertical completion parameters via DMA2D_OUT_DSCR_PORT_BLK_V_CHx to 20 for YUV420, or to 18 for other color modes
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 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.

29.7.3 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 29.5-3

29.7.4 BLEND Configuration

1. Refer to Section 29.7.1 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 29.5-4.

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 29.7.3
   - Configure the data type for the background layer input of BLEND via PPA_BLEND1_RX_CM. If the format is L4/L8, initialize the CLUT first as described in 29.7.3
   - 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

29.7.5 BLEND Image Filling Configuration

1. Refer to Section 29.7.1 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 29.5-4.

3. Write 0x2 to DMA2D_IN_PERI_SEL_CHx to allocate one 2D-DMA input channel to PPA BLEND output.

4. Write 1 to DMA2D_INLINK_START_CHx to enable the output channel of 2D-DMA.

5. Configure BLEND parameters:

   - Configure the output data format of BLEND via PPA_BLEND_TX_CM

- Write 1 to PPA_BLEND_FIX_PIXEL_FILL_EN to enable image filling
- Configure the pixel value for image filling via PPA_BLEND_TX_FIX_PIXEL
- Configure the filled image size via PPA_BLEND_HB and PPA_BLEND_VB

6. Write 1 to PPA_BLEND_TRANS_MODE_UPDATE to start BLEND workflow.

29.7.6 Error Handling

- Refer to Section 29.7.1 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.

29.8 Register Summary

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

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

Name	Description	Address	Access
Configuration Registers			
PPA_BLEND0_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND background layer	0x0000	R/W
PPA_BLEND1_CLUT_DATA_REG	CLUT SRAM data read/write register of BLEND foreground layer	0x0004	R/W
PPA_CLUT_CONF_REG	Configures BLEND CLUT	0x000C	R/W
PPA_SRM_COLOR_MODE_REG	Configures SRM color space	0x0020	R/W
PPA_BLEND_COLOR_MODE_REG	Configures BLEND color space	0x0024	R/W
PPA_SRM_BYTE_ORDER_REG	Configures SRM byte order	0x0028	R/W
PPA_BLEND_BYTE_ORDER_REG	Configures BLEND byte order	0x002C	R/W
PPA_BLEND_TRANS_MODE_REG	Configures the BLEND mode	0x0034	varies
PPA_SRM_FIX_ALPHA_REG	Configures SRM Alpha channel	0x0038	R/W
PPA_BLEND_TX_SIZE_REG	Configures BLEND image filling size	0x003C	R/W
PPA_BLEND_FIX_ALPHA_REG	Configures BLEND Alpha override	0x0040	R/W
PPA_BLEND_RGB_REG	Configures RGB color	0x0048	R/W
PPA_BLEND_FIX_PIXEL_REG	Configures BLEND image filling pixel	0x004C	R/W
PPA_CK_FG_LOW_REG	Configures the foreground color-key lower threshold of BLEND	0x0050	R/W
PPA_CK_FG_HIGH_REG	Configures the foreground color-key higher threshold of BLEND	0x0054	R/W
PPA_CK_BG_LOW_REG	Configures the background color-key lower threshold of BLEND	0x0058	R/W
PPA_CK_BG_HIGH_REG	Configures the background color-key higher threshold of BLEND	0x005C	R/W
PPA_CK_DEFAULT_REG	Configures the default color-key value of BLEND	0x0060	R/W
PPA_SRM_SCAL_ROTATE_REG	Configures the SRM mode	0x0064	varies
PPA_SRM_MEM_PD_REG	Configures SRM memory	0x0068	R/W
PPA_REG_CONF_REG	Enables register clock	0x006C	R/W
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
Status Registers			

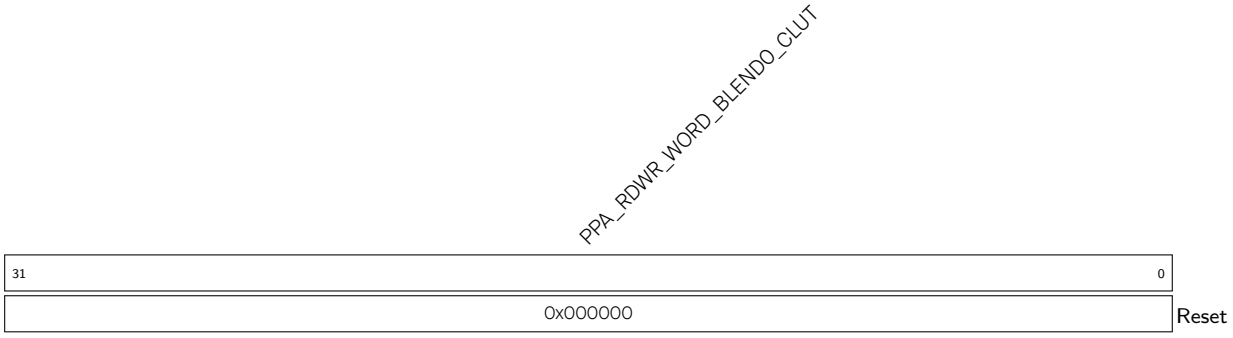
Name	Description	Address	Access
PPA_CLUT_CNT_REG	BLEND CLUT write counter register	0x0070	RO
PPA_BLEND_ST_REG	BLEND status register	0x0074	RO
PPA_SRM_PARAM_ERR_ST_REG	SRM configuration error status register	0x0078	RO
PPA_SRM_STATUS_REG	SRM FSM register	0x007C	RO
Version Register			
PPA_DATE_REG	PPA Version register	0x0100	R/W

29.9 Registers

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

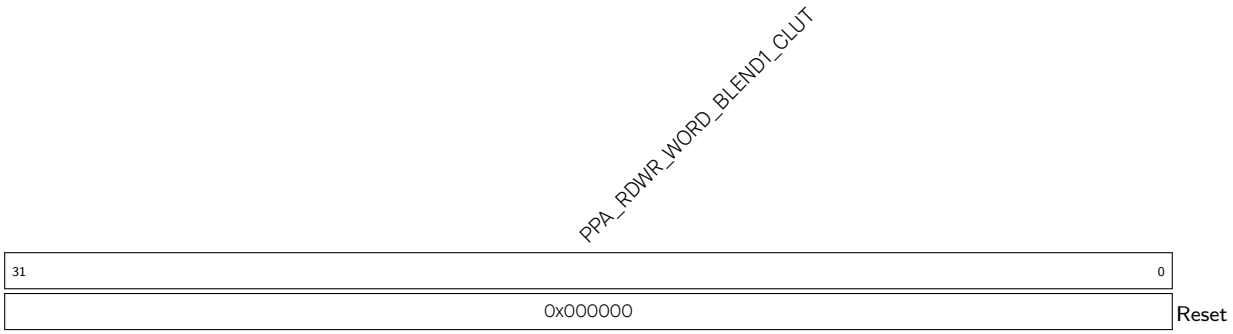
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 29.1. PPA_BLEND0_CLUT_DATA_REG (0x0000)



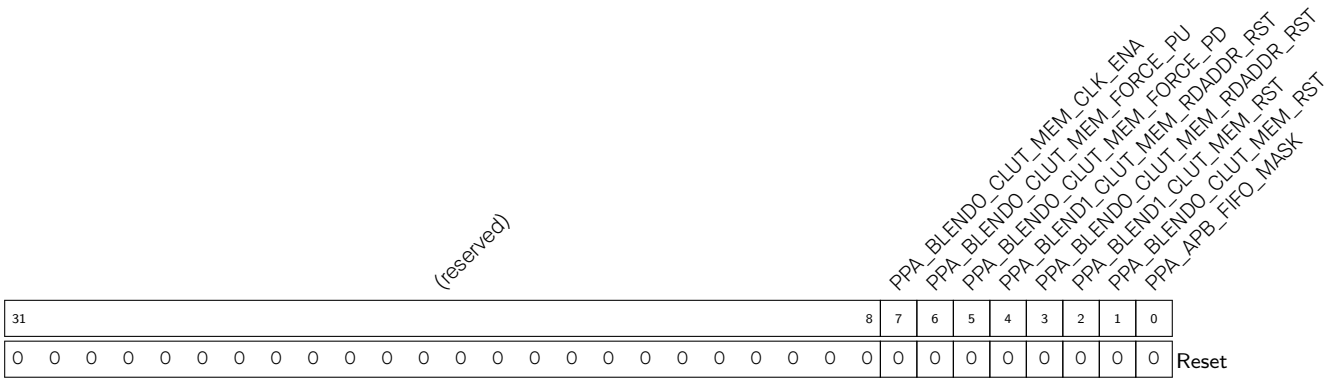
PPA_RDWR_WORD_BLEND0_CLUT Configures BLEND background layer CLUT access in FIFO mode. (R/W)

Register 29.2. PPA_BLEND1_CLUT_DATA_REG (0x0004)



PPA_RDWR_WORD_BLEND1_CLUT Configures BLEND foreground layer CLUT access in FIFO mode. (R/W)

Register 29.3. 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 29.5.1.2.

1'b1: Access CLUT in MEM mode. For details, please refer to 29.5.1.2.

(R/W)

PPA_BLEND0_CLUT_MEM_RST Configures whether to reset BLEND0 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND1_CLUT_MEM_RST Configures whether to reset BLEND1 CLUT.

0: Release reset

1: Reset

(R/W)

PPA_BLEND0_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND0 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

PPA_BLEND1_CLUT_MEM_RDADDR_RST Configures whether to reset BLEND1 CLUT read address in FIFO mode.

0: Release reset

1: Reset

(R/W)

PPA_BLEND_CLUT_MEM_FORCE_PD Configures whether to enable the force power down of BLEND CLUT.

0: Disable

1: Enable

(R/W)

Continued on the next page...

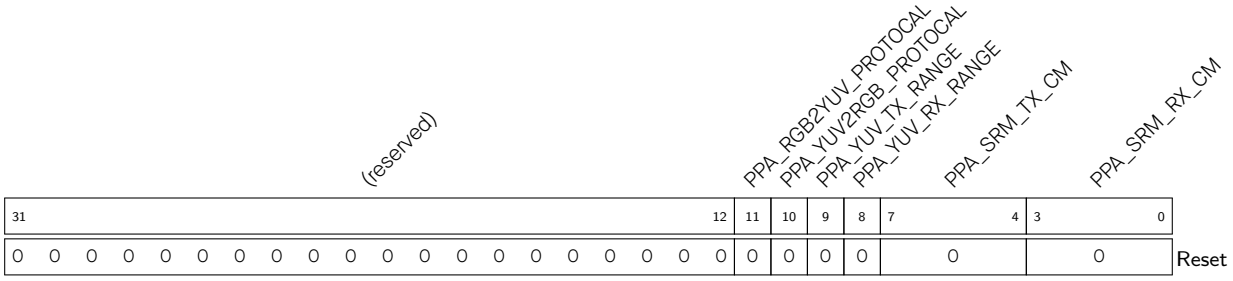
Register 29.3. PPA_CLUT_CONF_REG (0x000C)

Continued from the previous page...

PPA_BLEND_CLUT_MEM_FORCE_PU Configures whether to enable the force power up of BLEND CLUT.
0: Disable
1: Enable
(R/W)

PPA_BLEND_CLUT_MEM_CLK_ENA Configures whether to enable the force clock on of BLEND CLUT.
0: Disable
1: Enable
(R/W)

Register 29.4. PPA_SRM_COLOR_MODE_REG (0x0020)



PPA_SRM_RX_CM Configures the input image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

Others: Reserved

(R/W)

PPA_SRM_TX_CM Configures the output image color format for SRM.

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

Others: Reserved

(R/W)

PPA_YUV_RX_RANGE Configures the YUV range when the SRM input format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV_TX_RANGE Configures the YUV range when the SRM output format is YUV.

0: limit range

1: full range

(R/W)

PPA_YUV2RGB_PROTOCAL Configures the protocol used for the SRM input direction YUV to RGB.

0: BT601

1: BT709

(R/W)

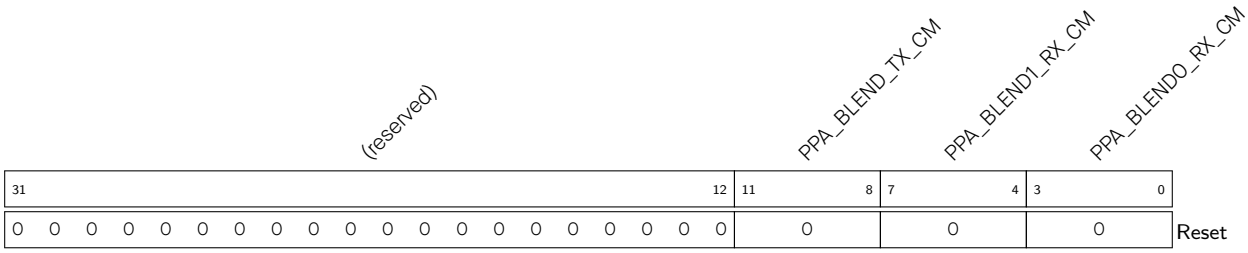
PPA_RGB2YUV_PROTOCAL Configures the protocol used for the SRM output direction RGB to YUV.

0: BT601

1: BT709

(R/W)

Register 29.5. 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

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

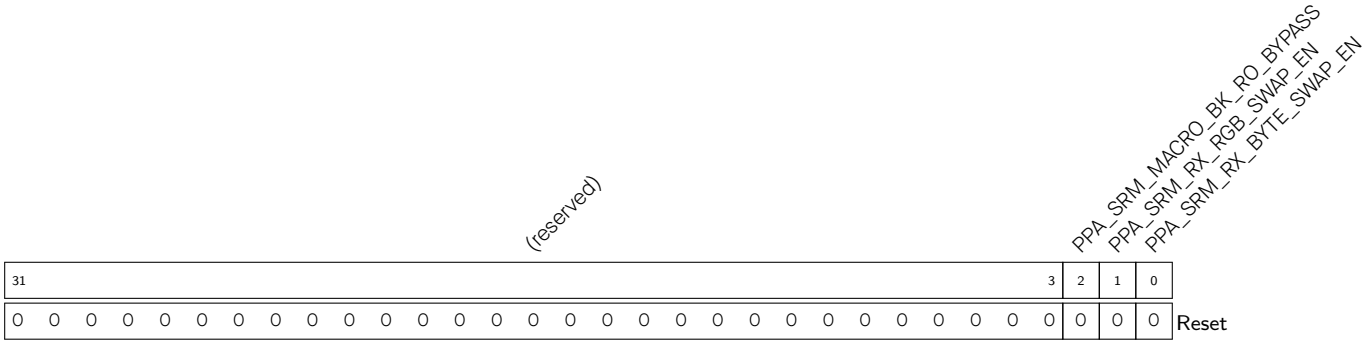
1: RGB888

2: RGB565

Others: Reserved

(R/W)

Register 29.6. 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)

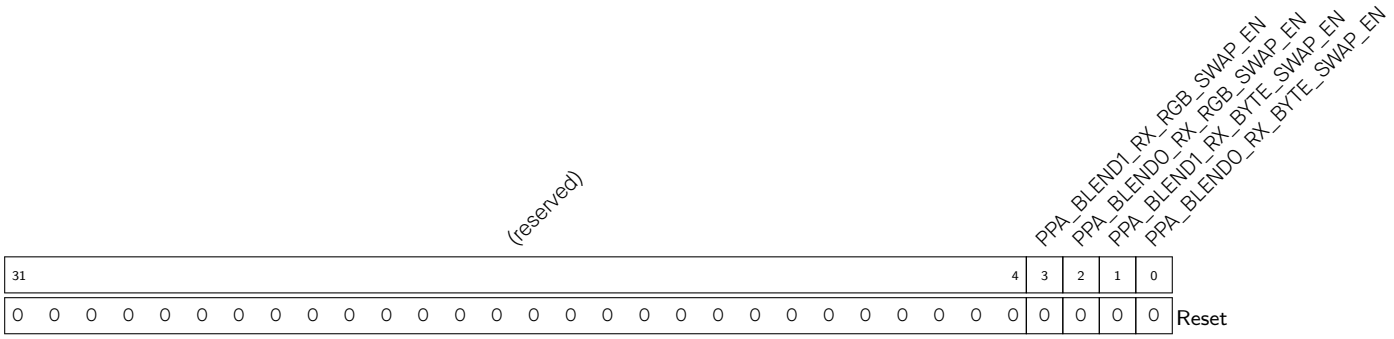
PPA_SRM_MACRO_BK_RO_BYPASS Configures whether to enable the pixel block order function

0: Enable and automatically calculate if the current block needs to be reordered

1: Disable

(R/W)

Register 29.7. 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)

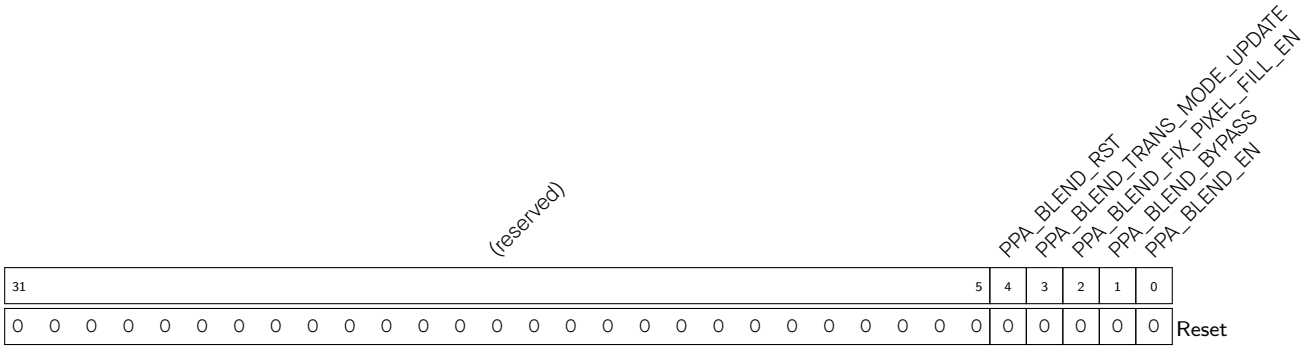
PPA_BLEND1_RX_RGB_SWAP_EN Configures whether to swap the order of each channel when the BLEND foreground layer input is RGB or ARGB.

0: Not swap

1: Swap RGB to BGR, and ARGB to BGRA

(R/W)

Register 29.8. PPA_BLEND_TRANS_MODE_REG (0x0034)



PPA_BLEND_EN Configures whether to enable BLEND.

0: Disable

1: Enable

(R/W)

PPA_BLEND_BYPASS Configures whether to bypass BLEND and directly output the background layer.

0: Bypass

1: Not bypass

(R/W)

PPA_BLEND_FIX_PIXEL_FILL_EN Configures whether to enable filled image output.

0: Disable

1: Enable, where only BLEND TX works and the output pixel and size are configured by PPA_BLEND_TX_SIZE_REG and PPA_BLEND_TX_FIX_PIXEL.

(R/W)

PPA_BLEND_TRANS_MODE_UPDATE Write 1 to update the transfer mode, active PPA_BLEND_FIX_PIXEL_FILL_EN, and start BLEND workflow. (WT)

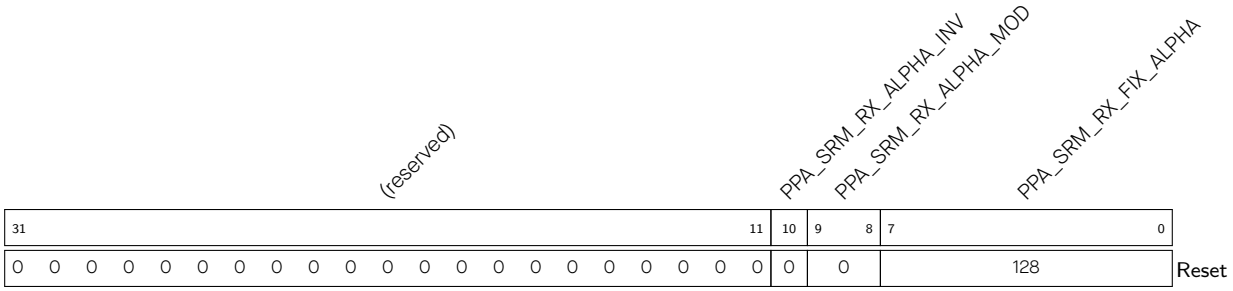
PPA_BLEND_RST Configures whether to reset BLEND.

0: Release reset

1: Reset

(R/W)

Register 29.9. PPA_SRM_FIX_ALPHA_REG (0x0038)



PPA_SRM_RX_FIX_ALPHA Configures the Alpha channal value of SRM. (R/W)

PPA_SRM_RX_ALPHA_MOD Configures the Alpha mode of SRM input data.

1: Use the value of PPA_SRM_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_SRM_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_SRM_RX_ALPHA_INV Configures whether to invert the original Alpha value of SRM.

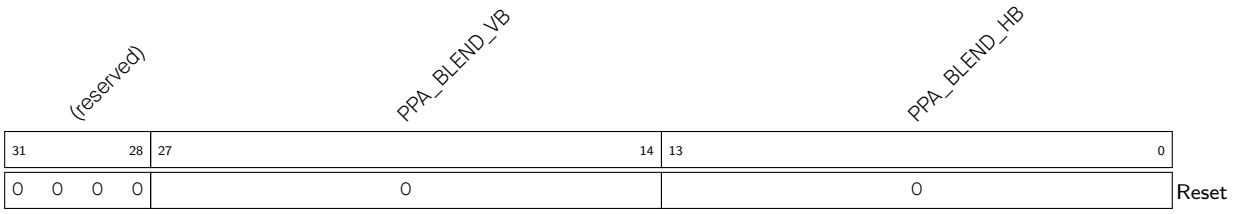
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

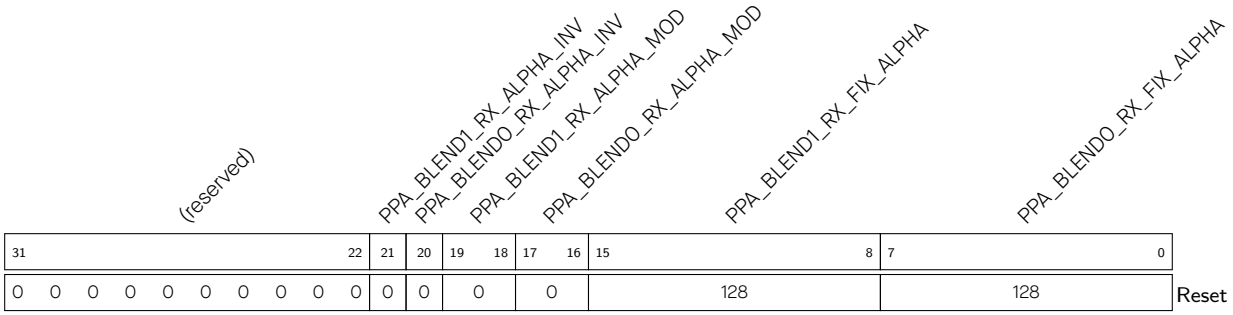
Register 29.10. PPA_BLEND_TX_SIZE_REG (0x003C)



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 29.11. PPA_BLEND_FIX_ALPHA_REG (0x0040)



PPA_BLEND0_RX_FIX_ALPHA Configures the Alpha channal value of the BLEND background layer. (R/W)

PPA_BLEND1_RX_FIX_ALPHA Configures the Alpha channal value of the BLEND foreground layer. (R/W)

PPA_BLEND0_RX_ALPHA_MOD Configures the Alpha mode of BLEND background layer data.

1: Use the value of PPA_BLEND0_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND0_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND1_RX_ALPHA_MOD Configures the Alpha mode of BLEND foreground layer data.

1: Use the value of PPA_BLEND1_RX_FIX_ALPHA

2: Use the value of the higher eight bits of the original Alpha value multiplied by PPA_BLEND1_RX_FIX_ALPHA

Others: Use the value of the original Alpha value

(R/W)

PPA_BLEND0_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND background layer data.

0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

PPA_BLEND1_RX_ALPHA_INV Configures whether to invert the original Alpha value of the BLEND foreground layer data.

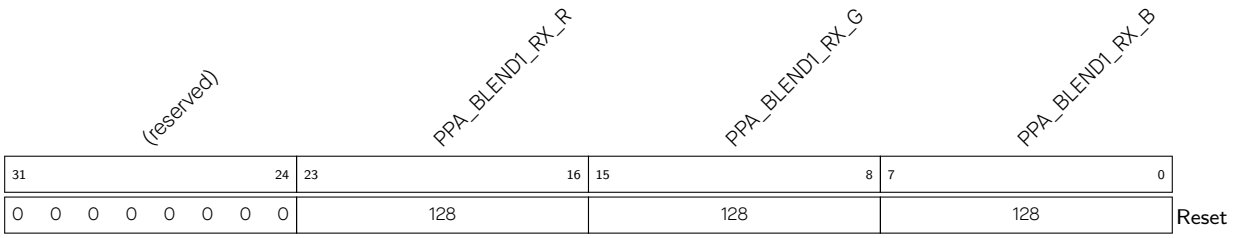
0: Not invert

1: Invert to 255 minus the original Alpha value

For data formats without Alpha channels, the original Alpha value is 255.

(R/W)

Register 29.12. PPA_BLEND_RGB_REG (0x0048)

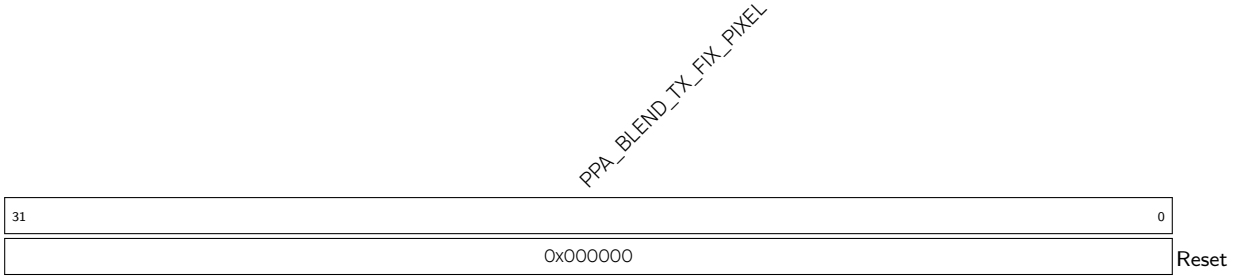


PPA_BLEND1_RX_B Configures the B channel value when the BLEND foreground layer is in A4/A8. (R/W)

PPA_BLEND1_RX_G Configures the G channel value when the BLEND foreground layer is in A4/A8. (R/W)

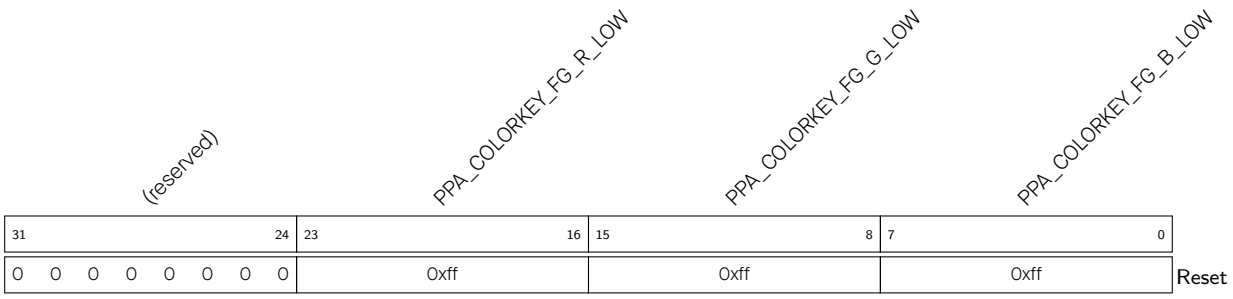
PPA_BLEND1_RX_R Configures the R channel value when the BLEND foreground layer is in A4/A8. (R/W)

Register 29.13. 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 29.14. PPA_CK_FG_LOW_REG (0x0050)

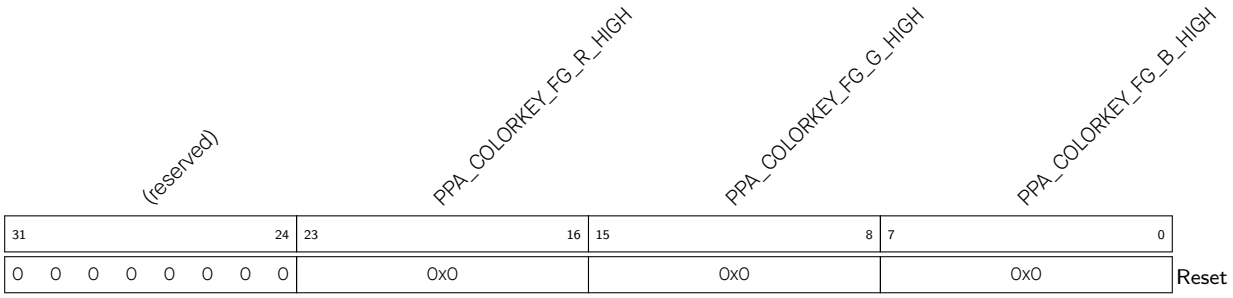


PPA_COLORKEY_FG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND foreground layer. (R/W)

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 29.15. PPA_CK_FG_HIGH_REG (0x0054)

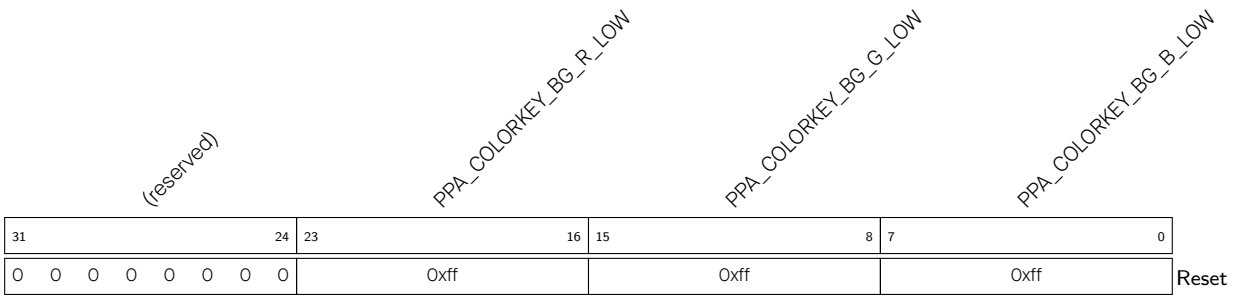


PPA_COLORKEY_FG_B_HIGH Configures the color-key higher threshold of the B channel value of the BLEND foreground layer. (R/W)

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 29.16. PPA_CK_BG_LOW_REG (0x0058)

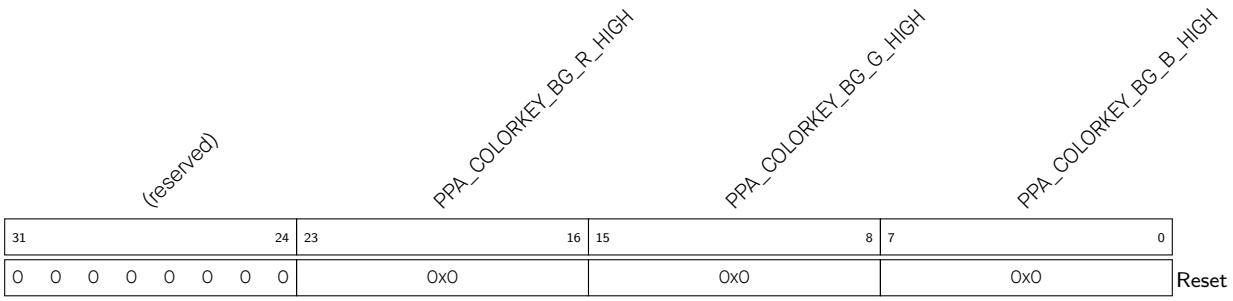


PPA_COLORKEY_BG_B_LOW Configures the color-key lower threshold of the B channel value of the BLEND background layer. (R/W)

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 29.17. 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 29.18. 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)

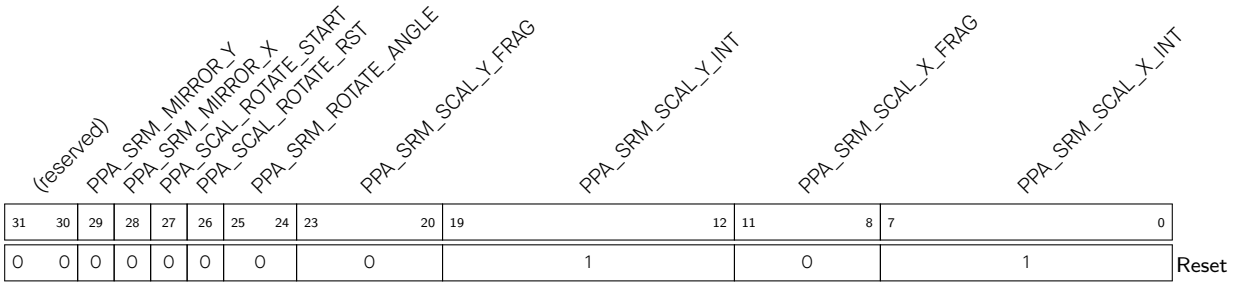
PPA_COLORKEY_FG_BG_REVERSE Configures the workflow when when a pixel is within the background color-key range but not within the foreground color-key range.

0: Output background pixel

1: Output foreground pixel

(R/W)

Register 29.19. 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)

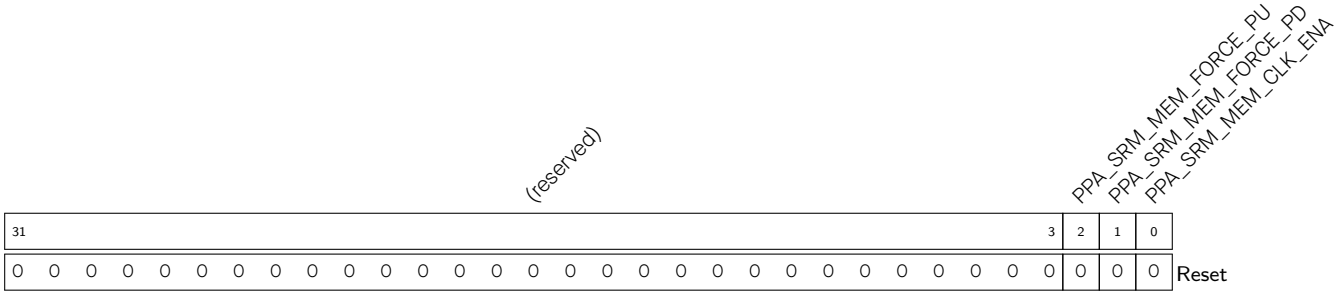
PPA_SRM_MIRROR_X Configures whether to enable the horizontal mirroring for SRM.

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 29.20. PPA_SRM_MEM_PD_REG (0x0068)



PPA_SRM_MEM_CLK_ENA Configures whether to enable the force enable of the SRM MEM clock.

0: Disable

1: Enable

(R/W)

PPA_SRM_MEM_FORCE_PD Configures whether to enable the force power down of SRM MEM.

0: Disable

1: Enable

(R/W)

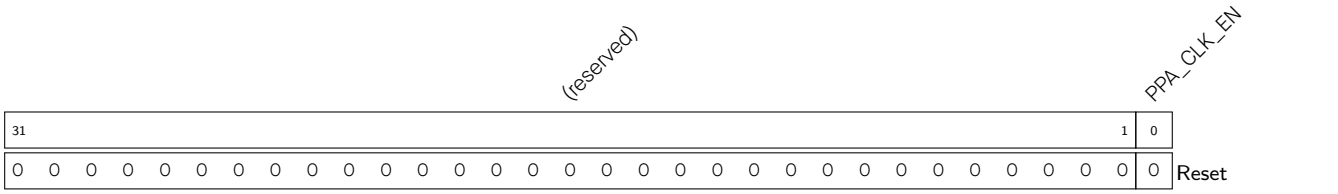
PPA_SRM_MEM_FORCE_PU Configures whether to enable the force power up of SRM MEM.

0: Disable

1: Enable

(R/W)

Register 29.21. PPA_REG_CONF_REG (0x006C)



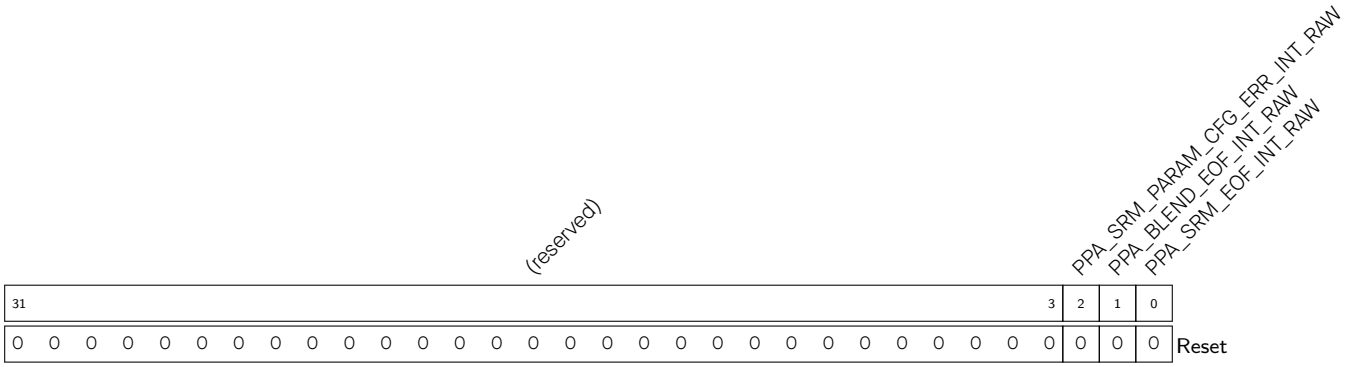
PPA_CLK_EN Configures whether to keep the PPA register clock always on.

0: Clock only turns on when there's a register access

1: Clock always on

(R/W)

Register 29.22. PPA_INT_RAW_REG (0x0010)



PPA_SRM_EOF_INT_RAW The raw interrupt status of PPA SRM EOF.

0: No PPA SRM EOF occured

1: PPA SRM EOF occured

(R/WTC/SS)

PPA_BLEND_EOF_INT_RAW The raw interrupt status of PPA BLEND EOF.

0: No PPA BLEND EOF occured

1: PPA BLEND EOF occured

(R/WTC/SS)

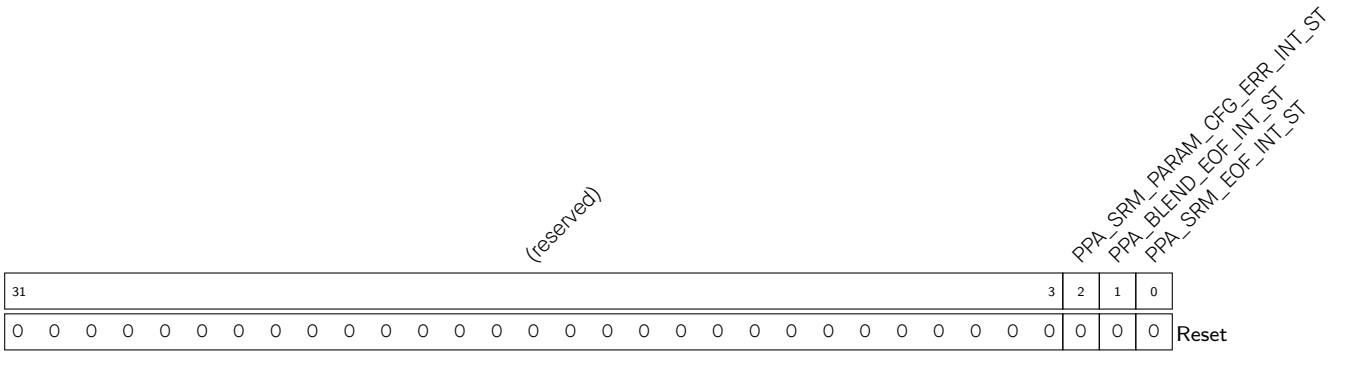
PPA_SRM_PARAM_CFG_ERR_INT_RAW The raw interrupt status of PPA SRM parameter error. Check PPA_SRM_PARAM_ERR_ST_REG to get the specific error.

0: No PPA SRM parameter error occured

1: PPA SRM parameter error occured

(R/WTC/SS)

Register 29.23. PPA_INT_ST_REG (0x0014)

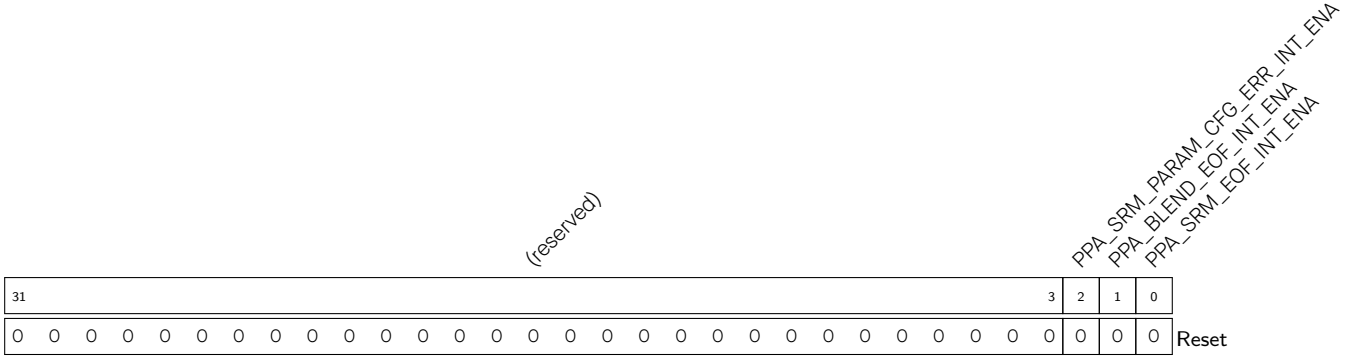


PPA_SRM_EOF_INT_ST The masked interrupt status of PPA_SRM_EOF_INT. (RO)

PPA_BLEND_EOF_INT_ST The masked interrupt status of PPA_BLEND_EOF_INT. (RO)

PPA_SRM_PARAM_CFG_ERR_INT_ST The masked interrupt status of PPA_SRM_RX_YSCAL_ERR_INT. (RO)

Register 29.24. PPA_INT_ENA_REG (0x0018)



PPA_SRM_EOF_INT_ENA Configures whether to enable the maskable interrupt PPA_SRM_EOF_INT.

0: Not trigger the PPA interrupt

1: Trigger the PPA interrupt

(R/W)

PPA_BLEND_EOF_INT_ENA Configures whether to enable the maskable interrupt PPA_BLEND_EOF_INT.

0: Not trigger the PPA interrupt

1: Trigger the PPA interrupt

(R/W)

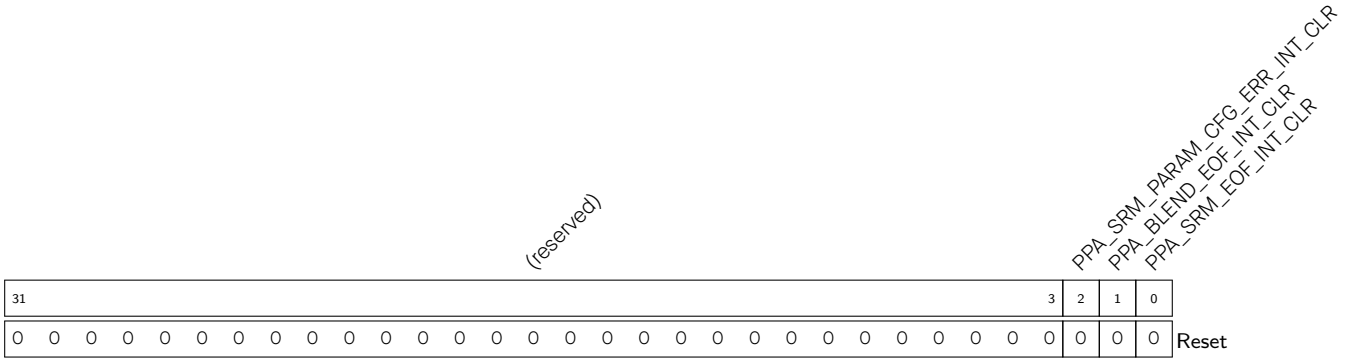
PPA_SRM_PARAM_CFG_ERR_INT_ENA Configures whether to enable the maskable interrupt PPA_SRM_RX_YSCAL_ERR_INT.

0: Not trigger the PPA interrupt

1: Trigger the PPA interrupt

(R/W)

Register 29.25. PPA_INT_CLR_REG (0x001C)

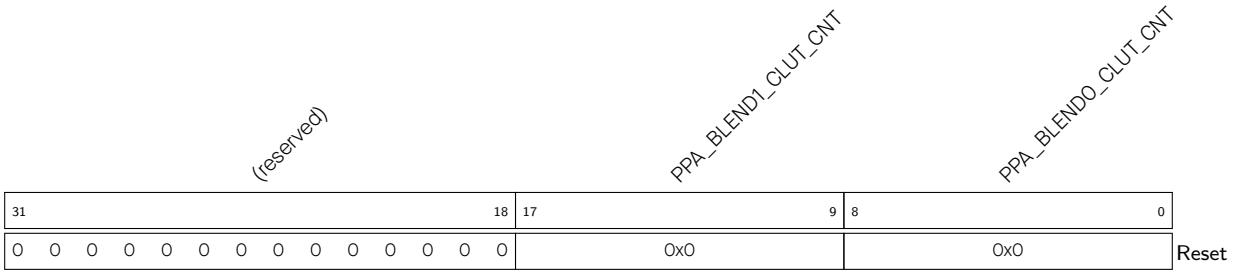


PPA_SRM_EOF_INT_CLR Write 1 to clear PPA_SRM_EOF_INT_RAW. (WT)

PPA_BLEND_EOF_INT_CLR Write 1 to clear PPA_BLEND_EOF_INT_RAW. (WT)

PPA_SRM_PARAM_CFG_ERR_INT_CLR Write 1 to clear PPA_SRM_PARAM_CFG_ERR_INT_RAW. (WT)

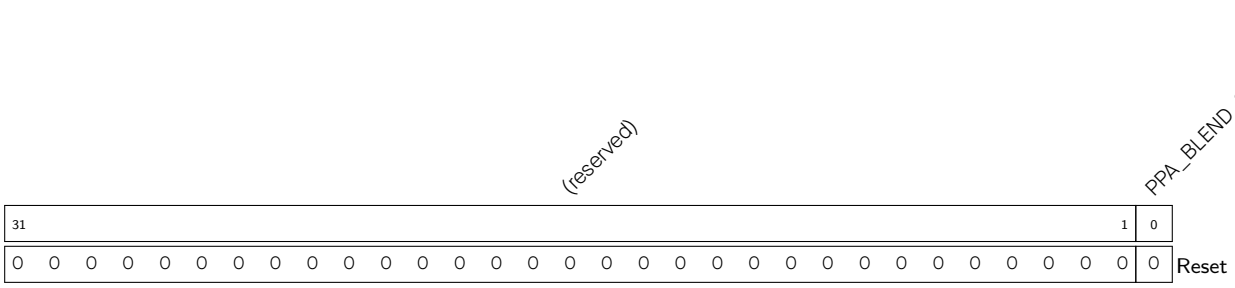
Register 29.26. PPA_CLUT_CNT_REG (0x0070)



PPA_BLEND0_CLUT_CNT Represents the current write address of the BLEND background layer CLUT in FIFO mode. (RO)

PPA_BLEND1_CLUT_CNT Represents the current write address of the BLEND foreground layer CLUT in FIFO mode. (RO)

Register 29.27. PPA_BLEND_ST_REG (0x0074)



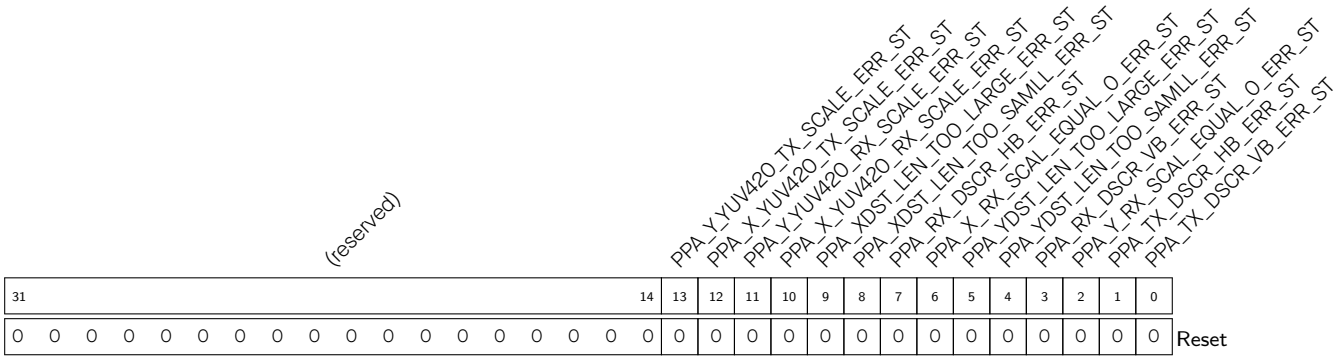
PPA_BLEND_SIZE_DIFF_ST Represents whether the size of the BLEND background and foreground images are consistent.

0: Consistent

1: Different

(RO)

Register 29.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)



PPA_TX_DSCR_VB_ERR_ST Represents whether the vertical size of the SRM output image block plus the vertical offset Y configured in the output list exceeds the vertical size of the entire image configured in the output list, VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_TX_DSCR_HB_ERR_ST Represents whether the horizontal size of the SRM output image block plus the vertical offset X configured in the output list exceeds the horizontal size of the entire image configured in the output list, HA.

0: Not exceeded

1: Exceeded

(RO)

PPA_Y_RX_SCAL_EQUAL_0_ERR_ST Represents whether the vertical direction scaling factor of the SRM input image block is 0.

0: Not 0

1: 0

(RO)

PPA_RX_DSCR_VB_ERR_ST Represents whether the sum of the vertical direction image block size VB configured in the SRM input list and the vertical offset Y exceeds the entire image's vertical size VA.

0: Not exceeded

1: Exceeded

(RO)

PPA_YDST_LEN_TOO_SAMLL_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is 0.

0: Not 0

1: 0

(RO)

Continued on the next page...

Register 29.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_YDST_LEN_TOO_LARGE_ERR_ST Represents whether the vertical size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_RX_SCAL_EQUAL_0_ERR_ST Represents whether the horizontal direction scaling factor of the SRM input image block is 0.
0: Not 0
1: 0
(RO)

PPA_RX_DSCR_HB_ERR_ST Represents whether the sum of the horizontal direction image block size HB configured in the SRM input list and the horizontal offset X exceeds the entire image's horizontal size HA.
0: Not exceeded
1: Exceeded
(RO)

PPA_XDST_LEN_TOO_SAMLL_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is 0.
0: Not 0
1: 0
(RO)

PPA_XDST_LEN_TOO_LARGE_ERR_ST Represents whether the horizontal size of the image block after scaling in the SRM is larger than 8191.
0: Within 8191
1: Larger than 8191
(RO)

PPA_X_YUV420_RX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM input list are odd when the input format is YUV420.
0: Even
1: Odd
(RO)

PPA_Y_YUV420_RX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM input list are odd when the input format is YUV420.
0: Even
1: Odd
(RO)

Continued on the next page...

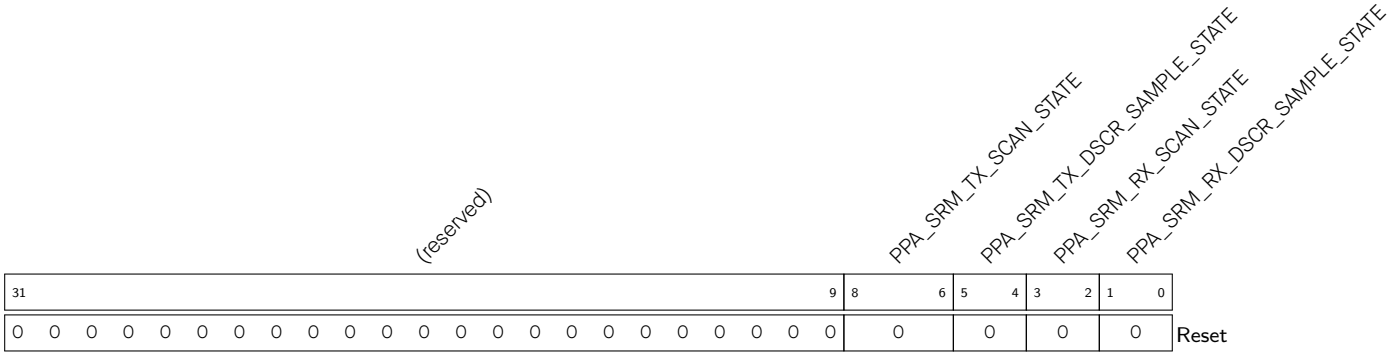
Register 29.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

Continued from the previous page...

PPA_X_YUV420_TX_SCALE_ERR_ST Represents whether the parameters HA/HB/X of the SRM output list are odd when the output format is YUV420.
0: Even
1: Odd
(RO)

PPA_Y_YUV420_TX_SCALE_ERR_ST Represents whether the parameters VA/VB/Y of the SRM output list are odd when the output format is YUV420.
0: Even
1: Odd
(RO)

Register 29.29. PPA_SRM_STATUS_REG (0x007C)



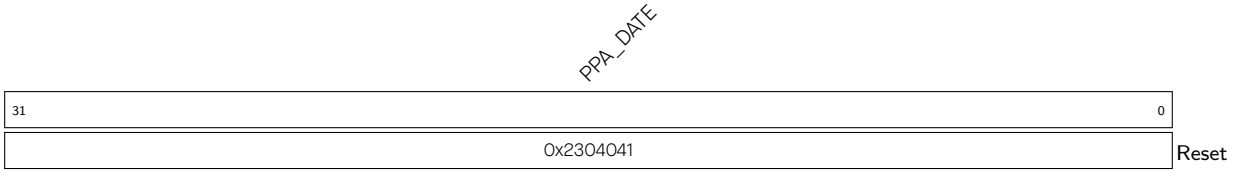
PPA_SRM_RX_DSCR_SAMPLE_STATE Reserved. (RO)

PPA_SRM_RX_SCAN_STATE Reserved. (RO)

PPA_SRM_TX_DSCR_SAMPLE_STATE Reserved. (RO)

PPA_SRM_TX_SCAN_STATE Reserved. (RO)

Register 29.30. PPA_DATE_REG (0x0100)



PPA_DATE Version control register. (R/W)

Chapter 30

LCD and Camera Controller (LCD_CAM)

30.1 Overview

The LCD and Camera controller (LCD_CAM) on the ESP32-P4, consisting of an independent LCD control module and a camera control module, is a versatile component designed to facilitate interfacing with both LCDs and cameras.

The LCD module provides an LCD interface that can connect to RGB, Motorola 6800 (MOTO6800), and Intel 8080 (I8080) compatible LCD devices. The LCD module is responsible for sending video data to the display.

The Camera module provides one parallel camera interface that can connect to external digital video port (DVP) cameras in 8/16-bit modes. It manages communication with the camera to capture images or video frames.

The LCD_CAM controller offers flexibility for projects that involve both displaying information on screens and working with cameras for image or video processing.

30.2 Features

LCD_CAM has the following features:

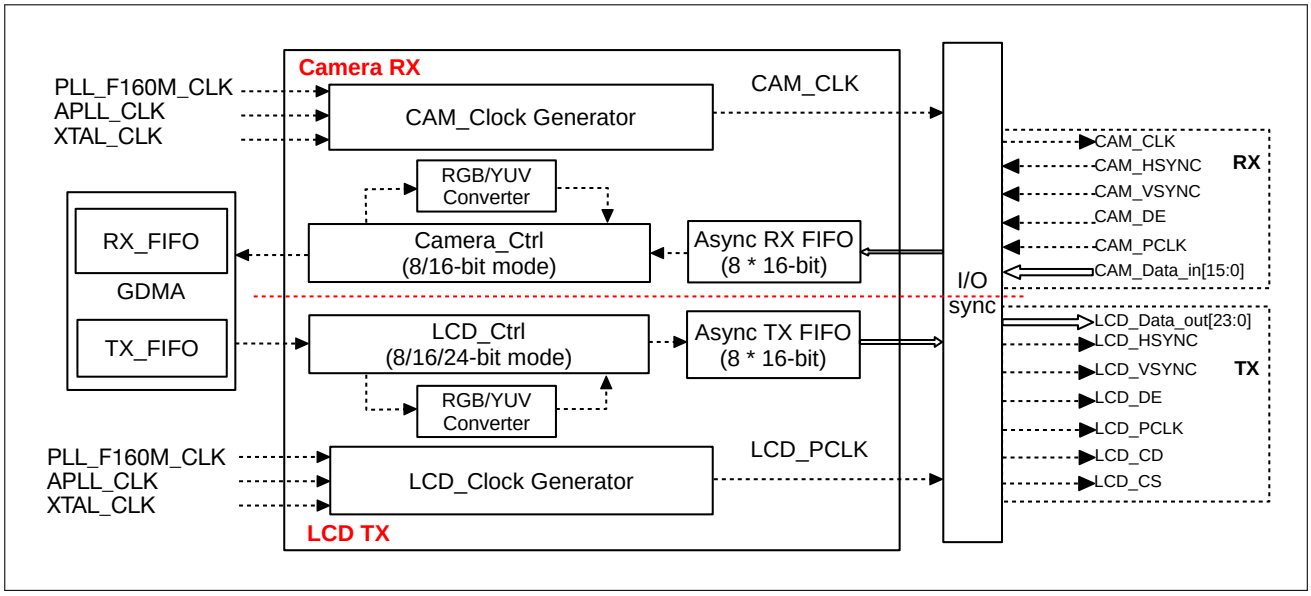
- Supports the following operation modes:
  - LCD master TX mode
  - Camera slave RX mode
  - Camera master RX mode
- Supports simultaneous connection to an external LCD and a camera
- When interfacing with an external LCD, the following is supported:
  - 8/16/24-bit parallel output modes
  - RGB, MOTO6800, and I8080 LCD formats
  - LCD data retrieved from internal memory or external memory via GDMA
- When interfacing with an external camera (i.e., DVP image sensor), the following is supported:
  - 8/16-bit parallel input modes
  - Camera data stored in internal or external memory via GDMA
- Supports interrupts

30.3 Functional Description

30.3.1 Block Diagram

Figure 30.3-1 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 30.3-1. LCD_CAM Block Diagram

30.3.2 Signal Description

Table 30.3-1 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 30.3-1. LCD_CAM Input and Output Signal Descriptions

Operation Mode	Signal[1]	Direction	Function
Camera Slave RX Mode	CAM_PCLK	Input	Camera pixel clock signal
	CAM_VSYNC	Input	Vertical synchronization signal (VSYNC)[3]
	CAM_HSYNC	Input	Horizontal synchronization signal (HSYNC)[3]
	CAM_DE	Input	Horizontal enable signal (DE)[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
Camera Master RX Mode	CAM_PCLK	Input	Camera pixel clock input signal
	CAM_CLK	Output	Camera master clock output signal
	CAM_VSYNC	Input	Vertical synchronization signal[3]
	CAM_HSYNC	Input	Horizontal synchronization signal[3]
	CAM_DE	Input	Horizontal enable signal[3]
	CAM_Data_in[*N*:0][2]	Input	Camera parallel input data bus, 8/16-bit modes supported
LCD Master TX Mode	LCD_PCLK	Output	LCD pixel clock signal
	LCD_HSYNC	Output	Horizontal synchronization signal in RGB format
	LCD_VSYNC	Output	Vertical synchronization signal in RGB format
	LCD_DE	Output	Horizontal enable signal in RGB format
	LCD_CD	Output	Command and data (CD) signal in I8080 format
	LCD_CS	Output	Chip select (CS) signal in I8080/MOTO6800 format
	LCD_Data_out[*M*:0][2]	Output	LCD parallel output data bus, 8/16/24-bit modes supported

[1] All the signals must be mapped to the chip's pins via GPIO matrix. For more information, see Chapter 7 *GPIO Matrix and IO MUX*.

[2] For the input signals with 8 or 16-bit width, *N* = 7 or 15 respectively. For the output signals with 8, 16, or 24-bit width, *M* = 7, 15, or 23 respectively.

[3] If LCD_CAM_CAM_VH_DE_MODE_EN is set, i.e., VSYNC + HSYNC mode is selected, then VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines. If LCD_CAM_CAM_VH_DE_MODE_EN is cleared, i.e., DE mode is selected, then VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the Camera module is not available.

30.3.3 LCD_CAM Module Clocks

30.3.3.1 LCD Clock

The clocks used in the LCD module are generated from clock sources by the LCD_Clock Generator, see Figure 30.3-2. 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

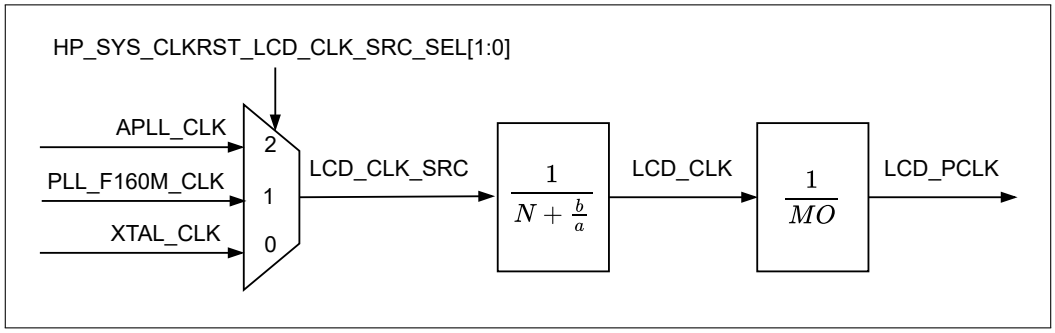


Figure 30.3-2. LCD Clock

The following formula shows the relation between the frequency of LCD_CLK (f_{LCD_CLK}) and LCD_CLK_SRC ($f_{LCD\_CLK\_SRC}$):

$$f_{LCD\_CLK} = \frac{f_{LCD\_CLK\_SRC}}{N + \frac{b}{a}}$$

"N" is an integer value between 2 and 256. The value of "N" corresponds to the value of HP_SYS_CLKRST_LCD_CLK_DIV_NUM as follows:

- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM = 0, "N" = 256.
- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM = 1, "N" = 2.
- When HP_SYS_CLKRST_LCD_CLK_DIV_NUM = other value, "N" = HP_SYS_CLKRST_LCD_CLK_DIV_NUM.

"b" corresponds to the value of HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR, and "a" to the value of HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR.

- For integer divider, HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR are cleared.
- For fractional divider, 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_{LCD_PCLK}) and LCD_CLK (f_{LCD_CLK}):

$$f_{LCD\_PCLK} = \frac{f_{LCD\_CLK}}{MO}$$

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 8 *Reset and Clock*.

30.3.3.2 Camera Clock

The clocks used in the Camera module are generated from clock sources by the CAM_Clock Generator, see Figure 30.3-3. 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

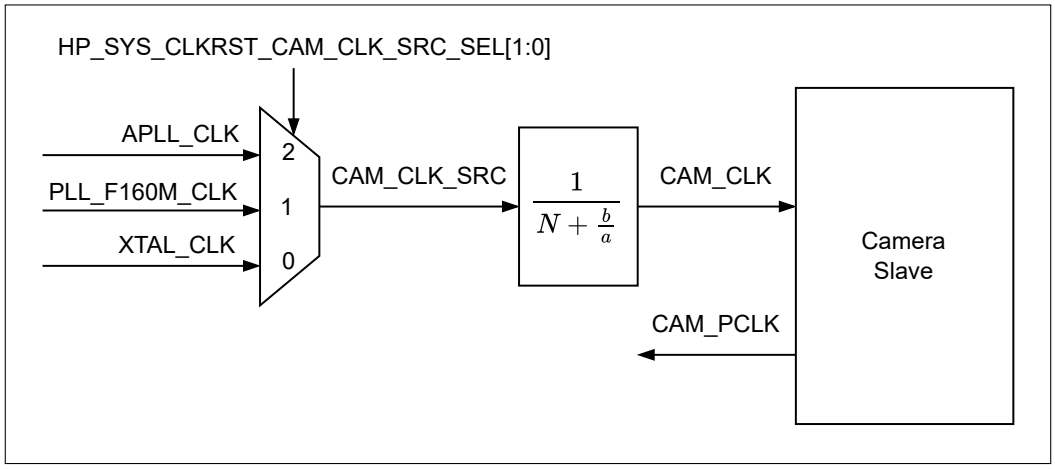


Figure 30.3-3. Camera Clock

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, HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR and HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR are cleared.
- 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.

30.3.4 LCD_CAM Reset

LCD_CAM module has the following reset registers that can reset different parts of the module:

- HP_SYS_CLKRST_RST_EN_LCDCAM: Setting this register will reset the whole LCD_CAM module.
- LCD_CAM_LCD_RESET: Setting this register will reset the whole LCD module, including the LCD control unit (LCD_Ctrl), the RGB/YUV Converter, and Async TX FIFO. Configuring this register will not affect the Camera module.
- LCD_CAM_CAM_RESET: Setting this register will reset the whole Camera module, including the Camera control unit (Camera_Ctrl), the RGB/YUV Converter, and Async RX FIFO. Configuring this register will not affect the LCD module.
- LCD_CAM_LCD_AFIFO_RESET: Setting this register will reset the Async TX FIFO without affecting other parts in the module. For example, setting this reset register every time the LCD finishes sending a frame can prevent transmission errors of the previous frame from affecting the transmission of the next frame.
- LCD_CAM_CAM_AFIFO_RESET: Setting this register will reset the Async RX FIFO without affecting other parts in the module. For example, setting this reset register every time the Camera finishes receiving a frame can prevent transmission errors of the previous frame from affecting the receiving of the next frame.

Notes:

- The reset register HP_SYS_CLKRST_RST_EN_LCDCAM is not self-clearing, meaning writing 1 enables the reset, and writing 0 releases the reset.
- All the above reset registers, except for HP_SYS_CLKRST_RST_EN_LCDCAM, are self-clearing, meaning that after writing 1, the hardware will generate a reset pulse and automatically clear the register.
- The module clocks LCD_CLK and CAM_CLK must be configured first before the module and FIFO are reset.

30.3.5 LCD_CAM Data Format Control

30.3.5.1 LCD Data Format Control

When the data is transmitted by the LCD module, it undergoes three stages of processing:

1. Stage One: Data preprocessing

The LCD module processes the input data from GDMA and sends it to the YUV-RGB converter.

2. Stage Two: Data color space conversion
   The YUV-RGB converter processes the data. If the YUV-RGB converter is disabled, the data remains unchanged at this stage.
3. Stage Three: Data post-processing
   The LCD module gets data from the YUV-RGB converter for post-processing and then transmits the data to GPIO.

Note:
The output data from the previous stage is the input data for the subsequent stage.

The detailed configuration of each stage is described below.

Stage One: Data Preprocessing

At this stage, configure the following registers to adjust the bit width, bit order, and byte order of the data from GDMA.

- LCD_CAM_LCD_BYTE_MODE
  - 0: The data width of the LCD input from GDMA is 8 bits.
  - 1: The data width of the LCD input from GDMA is 16 bits.
  - 2: The data width of the LCD input from GDMA is 24 bits.
- LCD_CAM_LCD_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the input data bit order.
- LCD_CAM_LCD_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16/24-bit modes.

For the detailed configuration, see Table 30.3-2.

Table 30.3-2. LCD Data Format Control – Stage One

GDMA Data Order[1]	LCD_CAM_LCD_BYTE_MODE	LCD_CAM_LCD_BIT_ORDER	LCD_CAM_LCD_BYTE_ORDER[2]	Output Data Order[3,4]
B0,B1,B2,B3,B4,B5	0	0	0	{B0}{B1}{B2}{B3}{B4}{B5}
		1	0	{B0'}{B1'}{B2'}{B3'}{B4'}{B5'}
	1	0	0	{B1,B0}{B3,B2}{B5,B4}
			1	{B0,B1}{B2,B3}{B4,B5}
		1	0	{B1',B0'}{B3',B2'}{B5',B4'}
			1	{B0',B1'}{B2',B3'}{B4',B5'}
	2	0	0	{B2,B1,B0}{B5,B4,B3}
			1	{B0,B1,B2}{B3,B4,B5}
		1	0	{B2',B1',B0'}{B5',B4',B3'}
			1	{B0',B1',B2'}{B3',B4',B5'}

[1] B0 ~ B5 represent the bytes of the data from GDMA, from low address to high address.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] In output data, the bits in {} are in big-endian, and are in parallel with each other, while the data of {} is in serial with the data of other {}. Data is sent out from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is sent out first.

[4] In output data, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3,4,5).

Stage Two: Data Color Space Conversion

At this stage, the data undergoes color space conversion. For detailed configurations, please refer to Section 30.3.6.

> **Note:**
> Users can skip Stage Two by configuring LCD_CAM_LCD_CONV_ENABLE as 0, in which case the output of Stage One will directly become the input of Stage Three.

Stage Three: Data Post-processing

At this stage, configure the following registers to adjust the bit and byte order of the output data.

- LCD_CAM_LCD_WIRE_MODE
  - 0: The bit width of the data transmitted to GPIO is 8 bits.
  - 1: The bit width of the data transmitted to GPIO 16 bits.
  - 2: The bit width of the data transmitted to GPIO is 24 bits.
- LCD_CAM_LCD_DOUT_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert the output data bit order.
- LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE
  - 0: Disable LCD output data byte reordering.
  - 1: Enable LCD output data byte reordering by configuring LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE. For the reordering results see Table 30.3-3.

Table 30.3-3. 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.

30.3.5.2 Camera Data Format Control

When the Camera module receives data, configure the following registers to adjust the bit/byte order of the data sent to GDMA.

- LCD_CAM_CAM_2BYTE_EN
  - 0: The data width of the Camera input is 8 bits.
  - 1: The data width of the Camera input is 16 bits.
- LCD_CAM_CAM_BIT_ORDER
  - 0: Do not invert.
  - 1: Invert data bit order.
    - Invert CAM_DATA_in[7:0] to CAM_DATA_in[0:7] in 8-bit mode.
    - Invert CAM_DATA_in[15:0] to CAM_DATA_in[0:15] in 16-bit mode.
- LCD_CAM_CAM_BYTE_ORDER
  - 0: Do not invert.
  - 1: Invert data byte order, only valid in 16-bit mode.

For the detailed configuration, see Table 30.3-4.

Table 30.3-4. Camera Data Format Control

RX Data Order[1]	LCD_CAM_CAM_2BYTE_EN	LCD_CAM_CAM_BIT_ORDER[2]	LCD_CAM_CAM_BYTE_ORDER[2]	GDMA Data Order[3,4]
{B0}{B1}{B2}{B3}	0	0	0	B0,B1,B2,B3
		1	0	B0',B1',B2',B3'
{B1,B0}{B3,B2}	1	0	0	B0,B1,B2,B3
			1	B1,B0,B3,B2
		1	0	B0',B1',B2',B3'
			1	B1',B0',B3',B2'

[1] In RX data, the bits in {} are in big-endian, and are in parallel with each other, while the data in {} is in serial with the data in other {}. Data is received from left to right. Take {B0}{B1}{B2}{B3} as an example. The bits in {B0} are in parallel with each other, but are in serial with the bits in {B1}{B2}{B3}. {B0} is received first.

[2] Only the configuration listed in the table is valid. Other configurations may cause unexpected data errors.

[3] B0 ~ B3 represent the bytes of the data to GDMA, from low address to high address.

[4] In the data to GDMA, Bn'[7:0] = Bn[0:7] (n = 0,1,2,3).

Note:
If only one byte is received each time, CAM_Data_in[7:0] is valid data. For such case, users must connect CAM_Data_in[7:0] with the master.

30.3.6 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
- between RGB565 (full/limited range) and YUV422/420/411 (full/limited range) formats
- between YUV422/420/411 (full/limited range) formats
- The converter in the LCD module also supports conversion:
  - from YUV422/420/411 (full/limited range) to RGB888 (full/limited range)
  - from RGB565 (full/limited range) to YUV444 (full/limited range)
  - from YUV422/420/411 (full/limited range) to YUV444 (full/limited range)

30.3.6.1 YUV Formats

In LCD_CAM module, assume that there are 8 pixels to be transmitted, corresponding to YUV data [Y_i, U_i, V_i] (i = 1 ~ 8). Then:

- In YUV422 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	V_2	Y_3	U_3	Y_4	V_4	Y_5	U_5	Y_6	V_6	Y_7	U_7	Y_8	V_8

- In YUV420 mode, the LCD sends (or the camera receives) the data as follows:

Y_1	U_1	Y_2	Y_3	U_3	Y_4	Y_5	V_5	Y_6	Y_7	V_7	Y_8

- 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

30.3.6.2 Format Conversion Configuration

The configuration process for the format conversion in the Camera module is almost identical to that in the LCD module. Therefore, we will illustrate the process using the example of format conversion in the LCD module.

1. Enable YUV-RGB format converter by setting LCD_CAM_LCD_CONV_ENABLE.
2. Configure the valid bit width of the input data by configuring LCD_CAM_LCD_CONV_MODE_8BITS_ON:
   - 0: The valid bit width is 16 bits.
   - 1: The valid bit width is 8 bits.
3. Configure the valid bit width of the output data:
   - For the format conversion in the Camera module, the valid bit width of the output data is equal to that of the input data.
   - For the format conversion in the LCD module, when LCD_CAM_LCD_WIRE_MODE is:
     - 2: The valid bit width of the format converter's output data is 24 bits.
     - Other value: The valid bit width of the format converter's output data is equal to that of the input data.
4. Select the standard by configuring LCD_CAM_LCD_CONV_PROTOCOL_MODE:
   - 0: BT601 standard
   - 1: BT709 standard
5. Configure the conversion mode:
   - In 24-bit output mode:

Table 30.3-5. Conversion Mode Control in 24-bit Mode

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]
RGB565 → YUV444	1	-	3
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

- In other output modes:

Table 30.3-6. Conversion Mode Control in Other Modes

Conversion Mode	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]
RGB565 → YUV422	1	0	3
RGB565 → YUV420	1	1	3
RGB565 → YUV411	1	2	3
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 → YUV411	1	1	2
YUV411 → YUV422	1	2	0
YUV411 → YUV420	1	2	1

[1] The value of LCD_CAM_LCD_CONV_TRANS_MODE

[2] The value of LCD_CAM_LCD_CONV_YUV_MODE

[3] The value of LCD_CAM_LCD_CONV_YUV2YUV_MODE

6. Configure the color range for the input data by configuring LCD_CAM_LCD_CONV_DATA_IN_MODE:
   - 0: limited color range[1]
   - 1: full color range[2]
7. Configure the color range for the output data by configuring LCD_CAM_LCD_CONV_DATA_OUT_MODE:
   - 0: limited color range[1]
   - 1: full color range[2]

Note:

1. If the limited color range is selected,
   - the color range of RGB is: 16 ~ 240.
   - the color range of YUV is:
     - Y: 16 ~ 240.
     - U-V: 16 ~ 235.
2. If the full color range is selected, the color range of RGB or YUV is 0 ~ 255.

30.3.7 LCD_CAM Timing

30.3.7.1 LCD Timing (RGB Format)

Figure 30.3-4 shows the LCD frame structure.

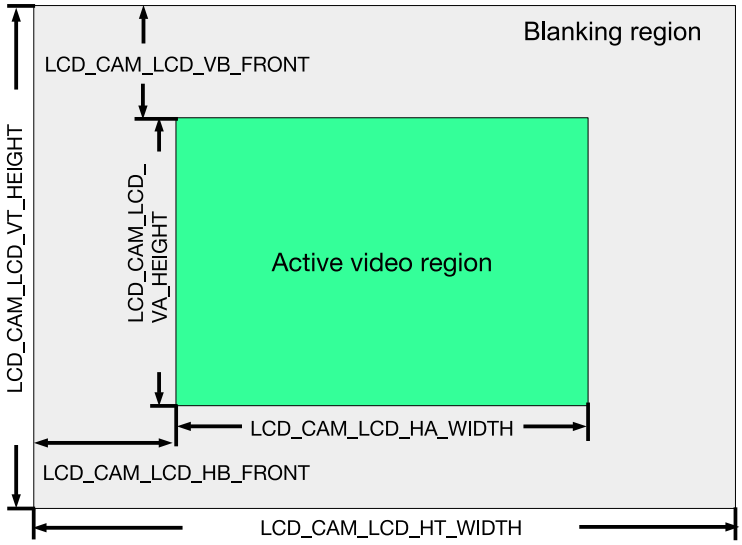


Figure 30.3-4. LCD Frame Structure

As depicted in the figure, the frame structure can be set up with the following registers:

- LCD_CAM_LCD_VT_HEIGHT
- LCD_CAM_LCD_VA_HEIGHT
- LCD_CAM_LCD_HB_FRONT
- LCD_CAM_LCD_HT_WIDTH
- LCD_CAM_LCD_HA_WIDTH
- LCD_CAM_VB_FRONT

Figure 30.3-5 shows the timing of an LCD full frame.

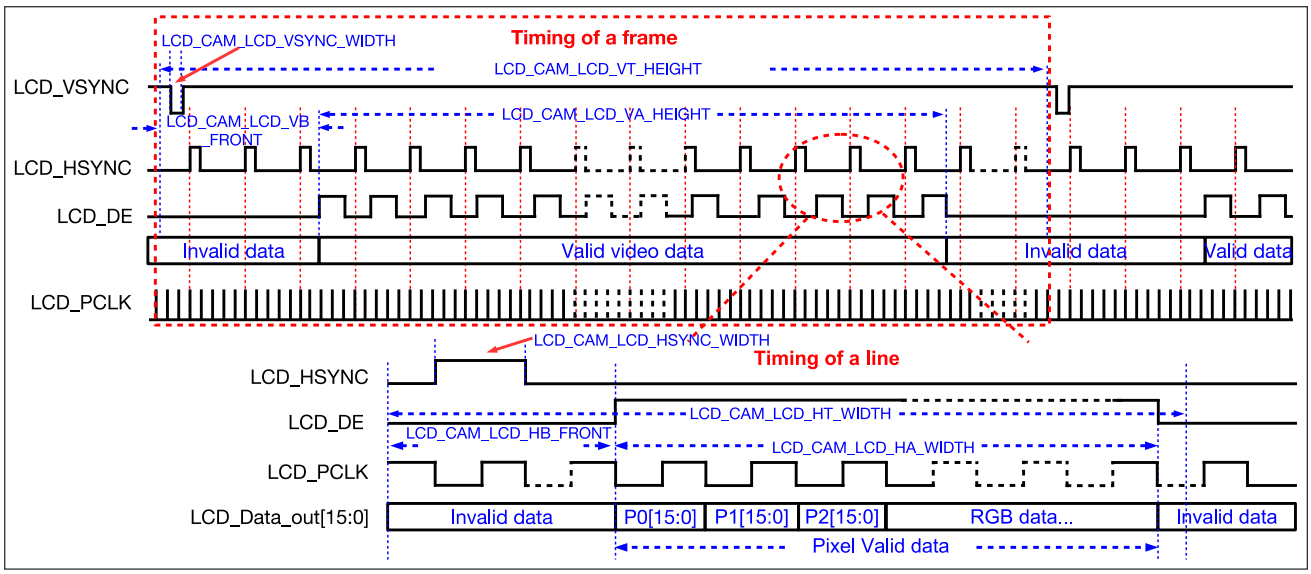


Figure 30.3-5. LCD Timing (RGB Format)

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.

30.3.7.2 LCD Timing (I8080/MOTO6800 Format)

Figure 30.3-6 shows the LCD timing sequence in I8080/MOTO6800 format.

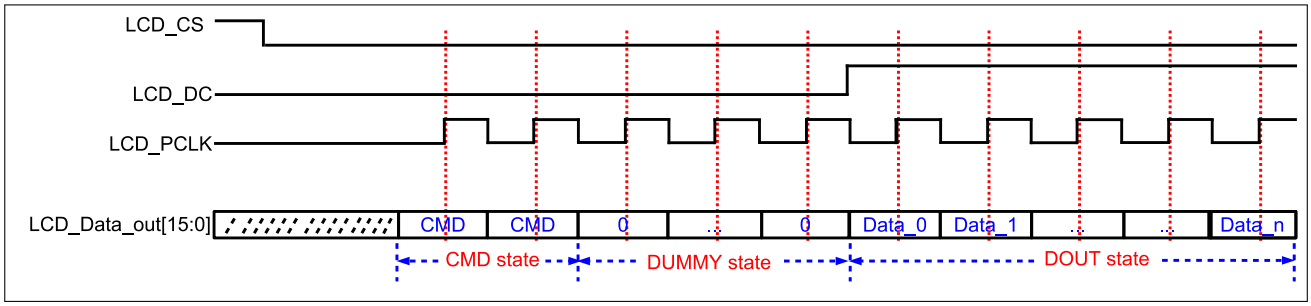


Figure 30.3-6. LCD Timing (I8080/MOTO6800 Format)

30.4 Interrupts

ESP32-P4's LCD_CAM can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- LCD_CAM_INTR

The following internal interrupt sources from LCD_CAM can generate LCD_CAM_INTR:

- LCD_CAM_CAM_HS_INT: Triggered when the Camera module receives lines more than or equal to the value of LCD_CAM_CAM_LINE_INT_NUM + 1.
- LCD_CAM_CAM_VSYNC_INT: Triggered when the Camera module receives a whole frame.

- LCD_CAM_LCD_TRANS_DONE_INT: Triggered when the LCD module finishes the transmission.
- LCD_CAM_LCD_VSYNC_INT: Triggered when the LCD module transmits a whole frame.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

30.5 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.

30.5.1 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 30.3.3.
2. Configure signal pins according to Table 30.3-1 and Chapter 7 *GPIO Matrix and IO MUX*.
3. Enable corresponding interrupts, see Section 30.4.
4. Enable RGB format by setting LCD_CAM_LCD_RGB_MODE_EN.
5. Configure frame structure according to Section 30.3.7.1.
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 30.3.4.
8. Configure GDMA outlink, see Chapter 3 *DW-GDMA Controller [to be added later]*.
9. Start transmitting data:
   - Wait till LCD slave gets ready.
   - Set LCD_CAM_LCD_START to start transmitting data.
10. Wait for the interrupt signals set in Step 3.
11. In frame intervals during data transmission, check if LCD_CAM_LCD_NEXT_FRAME_EN is set:
    - If yes, set LCD_CAM_LCD_UPDATE (repeating the steps above) to continue transmitting next frame.
    - If not, LCD stops transmitting data.
12. Clear LCD_CAM_LCD_START if data transmission is done.

30.5.2 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 30.3.3.

2. Configure signal pins according to Table 30.3-1 and Chapter 7 *GPIO Matrix and IO MUX*.
3. Enable corresponding interrupts, see Section 30.4.
4. Clear LCD_CAM_LCD_RGB_MODE_EN to disable RGB format.
5. Configure CMD phase with LCD_CAM_LCD_CMD, LCD_CAM_LCD_CMD_2_CYCLE_EN, and LCD_CAM_LCD_LATTER_CMD_VALUE.
6. Configure DUMMY phase with LCD_CAM_LCD_DUMMY and LCD_CAM_LCD_DUMMY_CYCLELEN.
7. Configure DOUT phase with LCD_CAM_LCD_DOUT, depending on the output mode:
   - In a fixed-length output[1], configure data length in LCD_CAM_LCD_DOUT_CYCLELEN.
   - In a continuous output[2], set LCD_CAM_LCD_ALWAYS_OUT_EN. Users do not need to configure LCD_CAM_LCD_DOUT_CYCLELEN.

   Note:

   (a) In a fixed-length output, the LCD module stops sending data once the data length reaches the value set in LCD_CAM_LCD_DOUT_CYCLELEN.

   (b) In a continuous output, LCD module keeps sending data till:

   - LCD_CAM_LCD_START is cleared;
   - or LCD_CAM_LCD_RESET is set;
   - or all the data in GDMA is sent out.

8. Configure the CD signal mode, including the default value of the CD signal and the values at each phase, see the description of LCD_CAM_LCD_MISC_REG.
9. Set LCD_CAM_LCD_UPDATE so that the configurations take effect.
10. Reset TX control unit (LCD_Ctrl) and Async TX FIFO as described in Section 30.3.4.
11. Configure GDMA outlink, see Chapter 3 *DW-GDMA Controller [to be added later]*.
12. Start transmitting data:
    - Wait till LCD slave gets ready.
    - Set LCD_CAM_LCD_START to start transmitting data.
13. Wait for the interrupt signals set in Step 2.
14. Clear LCD_CAM_LCD_START if data transmission is done.

Notes:

No matter in which format, RGB or I8080/MOTO6800, the rules below must be followed when accessing internal and external memory via GDMA:

- If LCD data bus is configured to 8-bit parallel output mode, then
  - The pixel clock frequency must be less than 80 MHz.
  - If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 60 MHz.
- If LCD data bus is configured to 16-bit parallel output mode, then

- The pixel clock frequency must be less than 40 MHz.
- If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

30.5.3 Configure Camera as RX Mode

Follow the steps below to configure the Camera as RX mode via software:

1. Configure clock according to Section 30.3.3. Note that in slave mode, the module clock frequency should be twice as fast as the PCLK frequency of the image sensor.
2. Configure signal pins according to Table 30.3-1 and Chapter 7 *GPIO Matrix and IO MUX*.
3. Set or clear LCD_CAM_CAM_VH_DE_MODE_EN according to the control signal HSYNC.
4. Set needed RX data format according to Section 30.3.5.2.
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 30.3.4.
7. Enable corresponding interrupts, see Section 30.4.
8. Configure GDMA inlink, and set the length of RX data in LCD_CAM_CAM_REC_DATA_BYTELEN.
9. Start receiving data:
   - In master mode, when the slave is ready, set LCD_CAM_CAM_START to start receiving data.
   - In slave mode, set LCD_CAM_CAM_START. Receiving data starts after the master provides clock signal and control signal.
10. Receive data and store the data to GDMA. Then corresponding interrupts set in Step 7 will be generated.

Notes:

- No matter in which operation mode, camera master RX mode or camera slave RX mode, the rules below must be followed when accessing internal memory via GDMA:
  - If 8-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 80 MHz.
    * If YUV-RGB format is being used at the same time, the pixel clock frequency must be less than 60 MHz.
  - If 16-bit parallel data input mode is selected, then
    * The pixel clock frequency must be less than 40 MHz.
    * If YUV-RGB format conversion is being used at the same time, the pixel clock frequency must be less than 30 MHz.

Note:
If both LCD and camera are connected externally at the same time, it is necessary to ensure that the maximum data throughput on the interface is less than the total data bandwidth of GDMA when accessing internal/external storage. The default frequency of APB_CLK is 80 MHz in this scenario. For more information about APB_CLK, see Chapter 8 *Reset and Clock*.

30.6 Register Summary

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

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

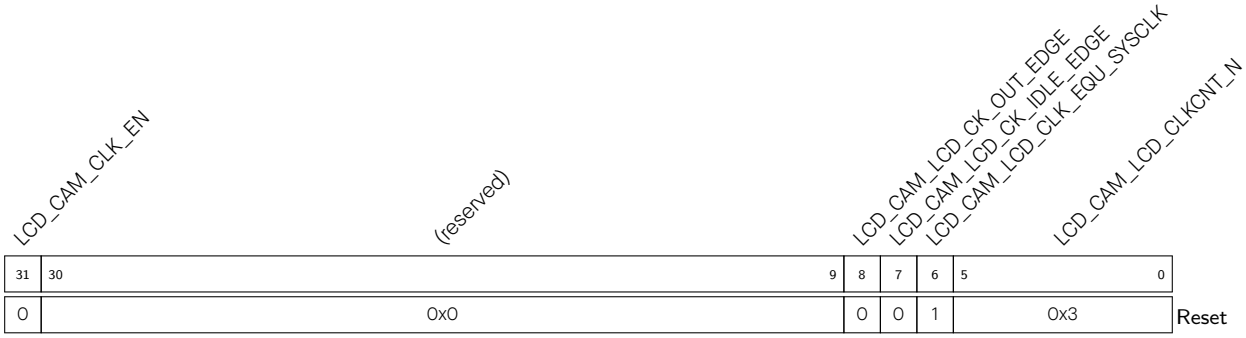
Name	Description	Address	Access
LCD Module Configuration Registers			
LCD_CAM_LCD_CLOCK_REG	LCD clock configuration register	0x0000	R/W
LCD_CAM_LCD_RGB_YUV_REG	LCD data format conversion register	0x0010	R/W
LCD_CAM_LCD_USER_REG	LCD user configuration register	0x0014	varies
LCD_CAM_LCD_MISC_REG	LCD MISC configuration register	0x0018	varies
LCD_CAM_LCD_CTRL_REG	LCD signal configuration register	0x001C	R/W
LCD_CAM_LCD_CTRL1_REG	LCD signal configuration register 1	0x0020	R/W
LCD_CAM_LCD_CTRL2_REG	LCD signal configuration register 2	0x0024	R/W
LCD_CAM_LCD_FIRST_CMD_VAL_REG	LCD command value configuration register	0x0028	R/W
LCD_CAM_LCD_LATTER_CMD_VAL_REG	LCD command value configuration register	0x002C	R/W
LCD_CAM_LCD_DLY_MODE_CFG1_REG	LCD data/signal delay configuration register	0x0030	R/W
LCD_CAM_LCD_DLY_MODE_CFG2_REG	LCD signal delay mode configuration register	0x0038	R/W
Camera Module Configuration Registers			
LCD_CAM_CAM_CTRL_REG	Camera clock configuration	0x0004	R/W
LCD_CAM_CAM_CTRL1_REG	Camera control register	0x0008	varies
LCD_CAM_CAM_RGB_YUV_REG	Camera data format conversion register	0x000C	R/W
Interrupt Registers			
LCD_CAM_LC_DMA_INT_ENA_REG	LCD_CAM GDMA interrupt enable register	0x0064	R/W
LCD_CAM_LC_DMA_INT_RAW_REG	LCD_CAM GDMA raw interrupt status register	0x0068	RO
LCD_CAM_LC_DMA_INT_ST_REG	LCD_CAM GDMA masked interrupt status register	0x006C	RO
LCD_CAM_LC_DMA_INT_CLR_REG	LCD_CAM GDMA interrupt clear register	0x0070	WO
Version Control Register			
LCD_CAM_LC_DATE_REG	Version control register	0x00FC	R/W

30.7 Registers

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

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

Register 30.1. LCD_CAM_LCD_CLOCK_REG (0x0000)



LCD_CAM_LCD_CLKCNT_N Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLK_EQU_SYSCLK.

When LCD_CAM_LCD_CLK_EQU_SYSCLK = 0,

$f_{LCD\_PCLK} = f_{LCD\_CLK}/$(LCD_CAM_LCD_CLKCNT_N + 1)

Note: this field must not be configured to 0.

(R/W)

LCD_CAM_LCD_CLK_EQU_SYSCLK Determines the frequency of the LCD pixel clock LCD_PCLK together with LCD_CAM_LCD_CLK_EQU_SYSCLK.

0: $f_{LCD\_PCLK} = f_{LCD\_CLK}/$(LCD_CAM_LCD_CLKCNT_N + 1)

1: $f_{LCD\_PCLK} = f_{LCD\_CLK}$

(R/W)

LCD_CAM_LCD_CK_IDLE_EDGE Indicates the level of LCD_PCLK in idle state.

0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_LCD_CK_OUT_EDGE Indicates the level of LCD_PCLK in the first half clock cycle.

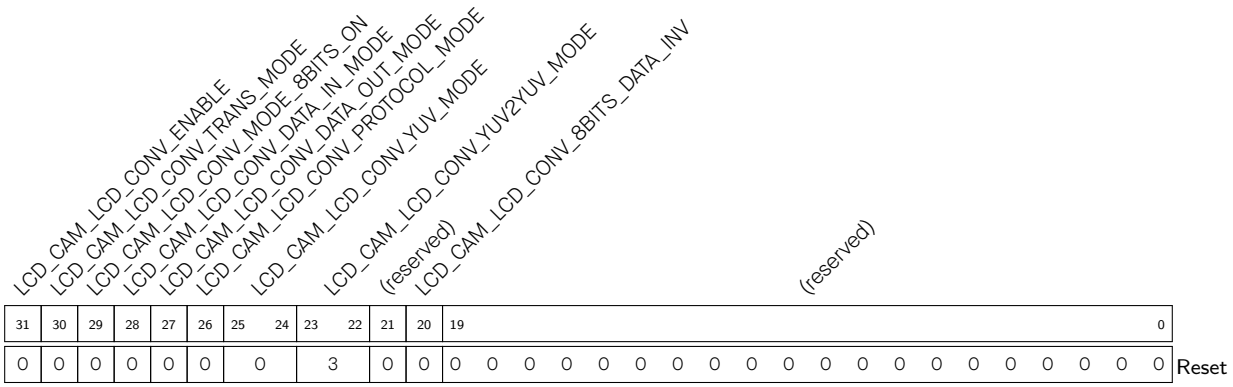
0: LCD_PCLK is low.

1: LCD_PCLK is high.

(R/W)

LCD_CAM_CLK_EN Set this bit to force enable the clock for all configuration registers. Clock gate is not used. (R/W)

Register 30.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)



LCD_CAM_LCD_CONV_8BITS_DATA_INV Configures whether to swap every two 8-bit GDMA input data.

0: Do not swap.

1: Swap.

(R/W)

LCD_CAM_LCD_CONV_YUV2YUV_MODE Configures data format conversion in YUV-to-YUV mode.

0: Data is converted to YUV422 format.

1: Data is converted to YUV420 format.

2: Data is converted to YUV411 format.

3: Data format conversion is disabled.

Only valid when LCD_CAM_LCD_CONV_TRANS_MODE = 1.

(R/W)

LCD_CAM_LCD_CONV_YUV_MODE Configures the YUV format of input data in YUV-to-YUV mode and YUV-to-RGB mode.

0: Input data is in YUV422 format.

1: Input data is in YUV420 format.

2: Input data is in YUV411 format.

(R/W)

LCD_CAM_LCD_CONV_PROTOCOL_MODE Configures the data format conversion standard.

0: BT601.

1: BT709.

(R/W)

LCD_CAM_LCD_CONV_DATA_OUT_MODE Configures the color range for output data.

0: Limited color range.

1: Full color range.

(R/W)

LCD_CAM_LCD_CONV_DATA_IN_MODE Configures the color range for input data.

0: Limited color range.

1: Full color range.

(R/W)

Continued on the next page...

Register 30.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)

Continued from the previous page...

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)

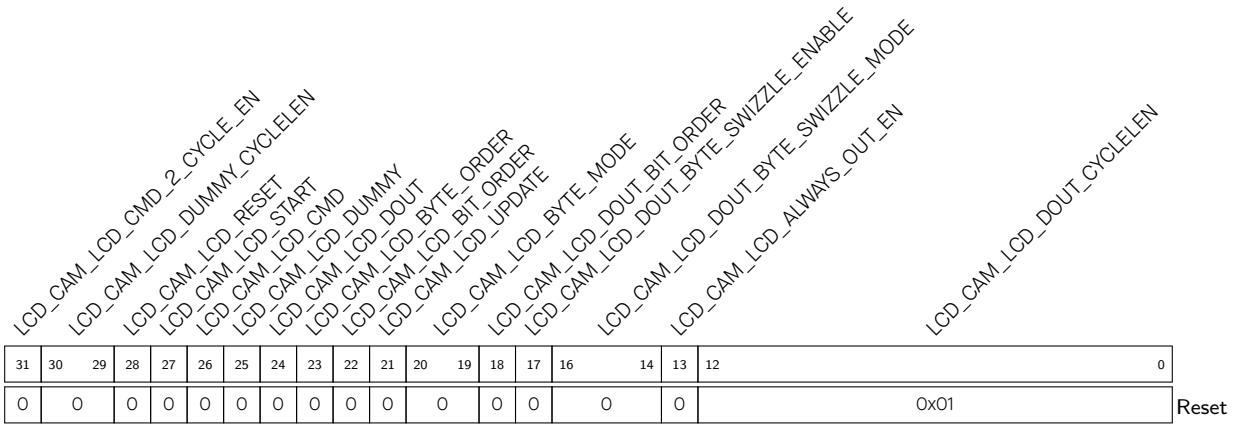
LCD_CAM_LCD_CONV_ENABLE Configures whether to enable YUV-RGB converter.

0: Bypass the converter.

1: Enable the converter.

(R/W)

Register 30.3. 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 30.3-3. (R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE Configures whether to enable LCD output data byte reordering.

0: Disable

1: Enable

(R/W)

LCD_CAM_LCD_DOUT_BIT_ORDER Configures whether to invert the LCD output data bit order.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_in[n-1:0] is inverted to LCD_DATA_in[0:n-1].

(R/W)

LCD_CAM_LCD_BYTE_MODE Configures the data bit width from GDMA.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_UPDATE Configures whether to update the LCD register configurations.

0: Do not update.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_LCD_BIT_ORDER Configures whether to invert the bit order of the LCD input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, LCD_DATA_in[n-1:0] is invertd to LCD_DATA_in[0:n-1].

(R/W)

Continued on the next page...

Register 30.3. 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)

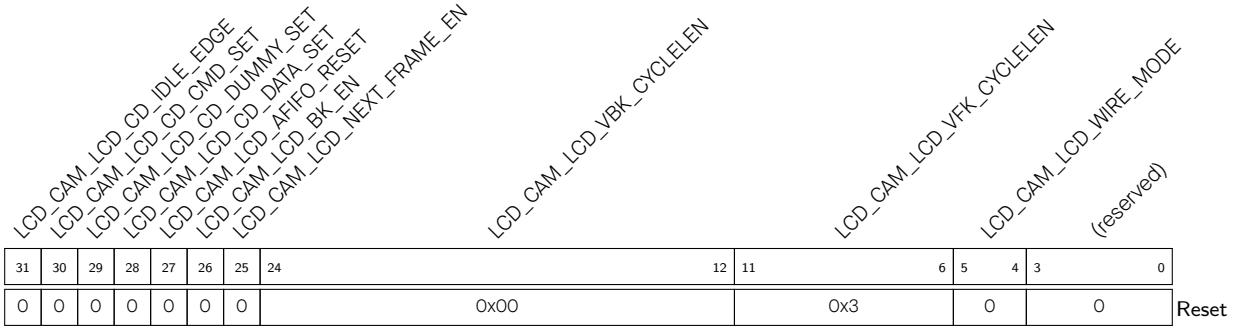
LCD_CAM_LCD_START When set to 1, the LCD module starts transmitting data. (R/W)

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 30.4. LCD_CAM_LCD_MISC_REG (0x0018)



LCD_CAM_LCD_WIRE_MODE Configures the bit width of LCD output data to GPIO.

0: 8-bit

1: 16-bit

2: 24-bit

(R/W)

LCD_CAM_LCD_VFK_CYCLELEN Configures the clock cycles of the setup time in LCD non-RGB mode. Setup clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_VBK_CYCLELEN Configures the clock cycles of the hold time in LCD non-RGB mode. Hold clock cycles = this value + 1. (R/W)

LCD_CAM_LCD_NEXT_FRAME_EN Configures whether to send the next frame.

0: The LCD module stops when the current frame is sent out.

1: The LCD module continues sending the next frame when the current frame is sent out.

(R/W)

LCD_CAM_LCD_BK_EN Configures whether to enable blanking region when LCD sends data.

0: No blanking region.

1: Enable blanking region.

(R/W)

LCD_CAM_LCD_AFIFO_RESET When set to 1, Async TX FIFO is reset. (WO)

LCD_CAM_LCD_CD_DATA_SET Configures LCD_CD value in DOUT phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_DUMMY_SET Configures LCD_CD value in DUMMY phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_CMD_SET Configures LCD_CD value in CMD phase for LCD I8080 mode.

0: LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1: LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_IDLE_EDGE The default value of LCD_CD. (R/W)

Register 30.5. LCD_CAM_LCD_CTRL_REG (0x001C)



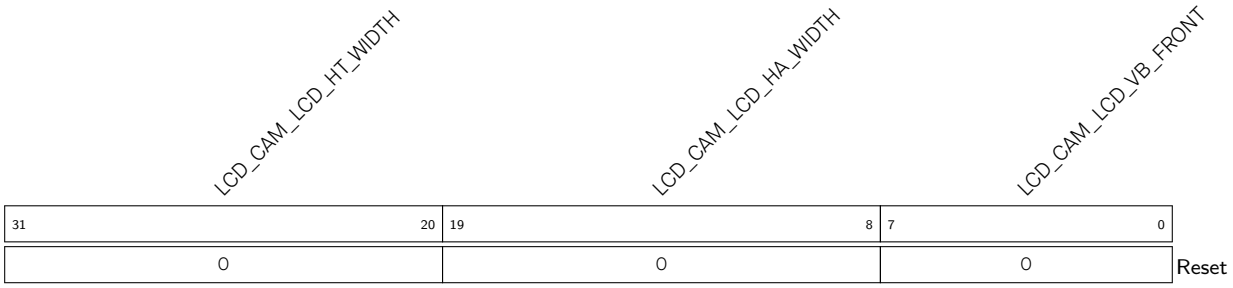
LCD_CAM_LCD_HB_FRONT Configures the value of (HSYNC_POSITION + HSYNC_WIDTH + horizontal back porch)
(R/W)

LCD_CAM_LCD_VA_HEIGHT Configures the vertical active height of a frame. (R/W)

LCD_CAM_LCD_VT_HEIGHT Configures the vertical total height of a frame. (R/W)

LCD_CAM_LCD_RGB_MODE_EN Configures whether to enable RGB mode.
0: Disable RGB mode.
1: Enable RGB mode and input VSYNC, HSYNC, and DE signals.
(R/W)

Register 30.6. LCD_CAM_LCD_CTRL1_REG (0x0020)

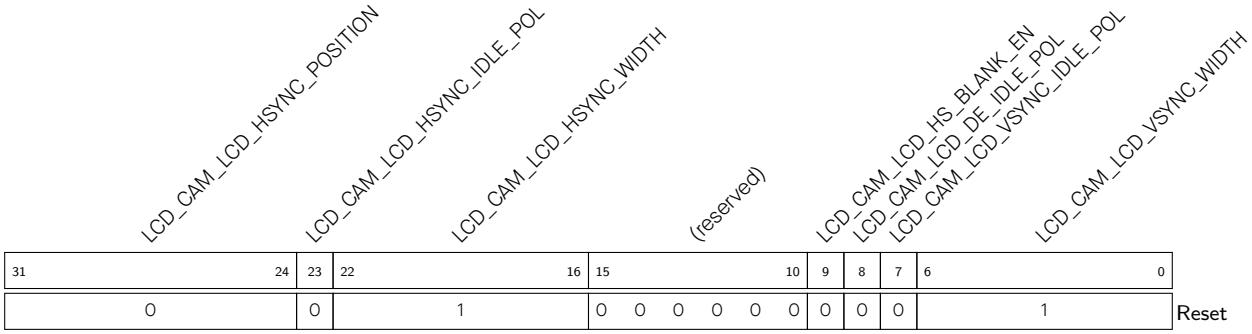


LCD_CAM_LCD_VB_FRONT Configures the value of (VSYNC_WIDTH + vertical back porch)
(R/W)

LCD_CAM_LCD_HA_WIDTH Configures the horizontal active width of a frame. (R/W)

LCD_CAM_LCD_HT_WIDTH Configures the horizontal total width of a frame. (R/W)

Register 30.7. LCD_CAM_LCD_CTRL2_REG (0x0024)



LCD_CAM_LCD_VSYNC_WIDTH Configures the width of LCD_VSYNC active pulse in a line. Expected width = this value + 1. (R/W)

LCD_CAM_LCD_VSYNC_IDLE_POL Configures the idle value of LCD_VSYNC.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_DE_IDLE_POL Configures the idle value of LCD_DE.

0: The idle value is low.

1: The idle value is high.

(R/W)

LCD_CAM_LCD_HS_BLANK_EN Configures the LCD_HSYNC output in RGB mode.

0: LCD_HSYNC is output only in active video lines.

1: LCD_HSYNC can be output in vertical blanking intervals.

(R/W)

LCD_CAM_LCD_HSYNC_WIDTH Configures the width of LCD_HSYNC active pulse. Expected width = this value +1. (R/W)

LCD_CAM_LCD_HSYNC_IDLE_POL Configures the idle value of LCD_HSYNC.

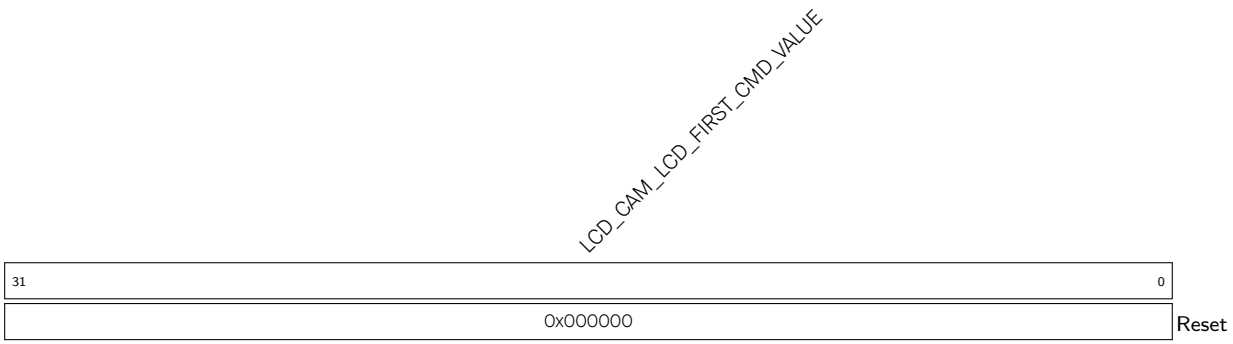
0: The idle value is low.

1: The idle value is high.

(R/W)

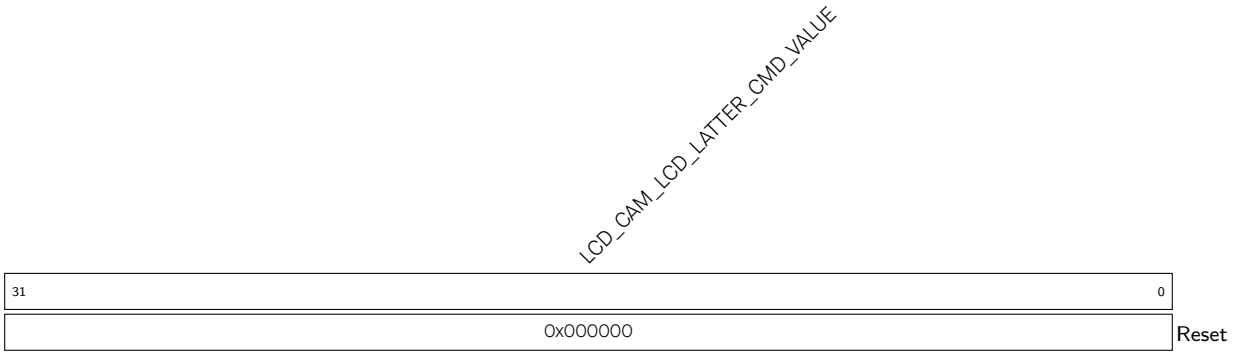
LCD_CAM_LCD_HSYNC_POSITION Configures the position of LCD_HSYNC active pulse. Expected position = this value + 1. Unit is a pixel. (R/W)

Register 30.8. LCD_CAM_LCD_FIRST_CMD_VAL_REG (0x0028)



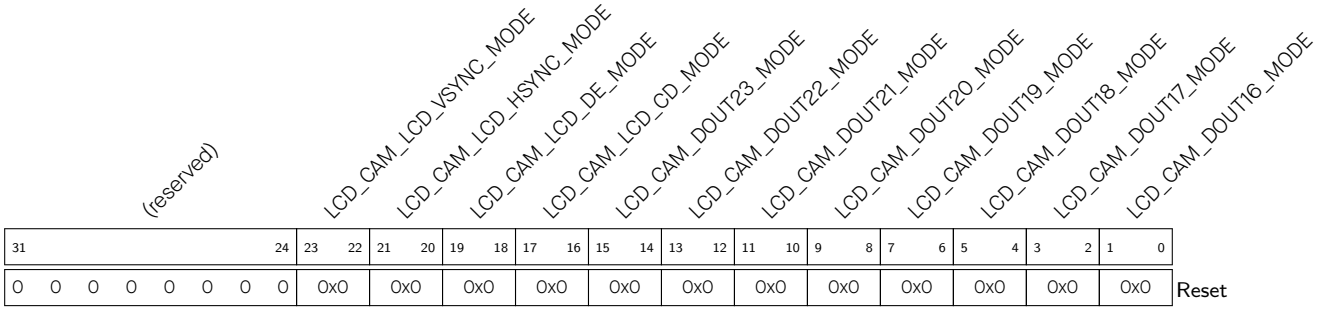
LCD_CAM_LCD_FIRST_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the first cycle of the CMD phase. (R/W)

Register 30.9. LCD_CAM_LCD_LATTER_CMD_VAL_REG (0x002C)



LCD_CAM_LCD_LATTER_CMD_VALUE Configures the value of the transmitted command when the LCD module is in the second cycle of the CMD phase. (R/W)

Register 30.10. LCD_CAM_LCD_DLY_MODE_CFG1_REG (0x0030)



LCD_CAM_DOUT*n*_MODE (*n*: 16 - 23) Configures the delay of the output data bit (*n*).

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_CD_MODE Configures the delay of LCD_CD output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_DE_MODE Configures the delay of LCD_DE output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_HSYNC_MODE Configures the delay of LCD_HSYNC output signal.

0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

LCD_CAM_LCD_VSYNC_MODE Configures the delay of LCD_VSYNC output signal.

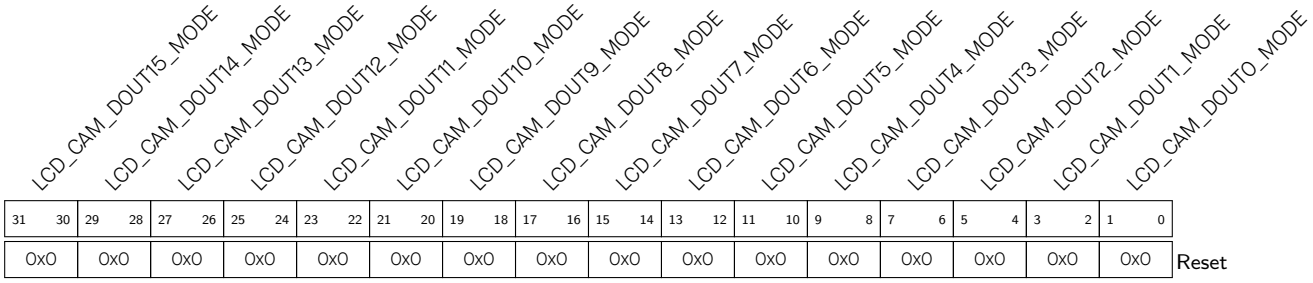
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 30.11. LCD_CAM_LCD_DLY_MODE_CFG2_REG (0x0038)



LCD_CAM_DOUT*n*_MODE (*n*: 0 - 15) Configures the delay of the output data bit *n*.

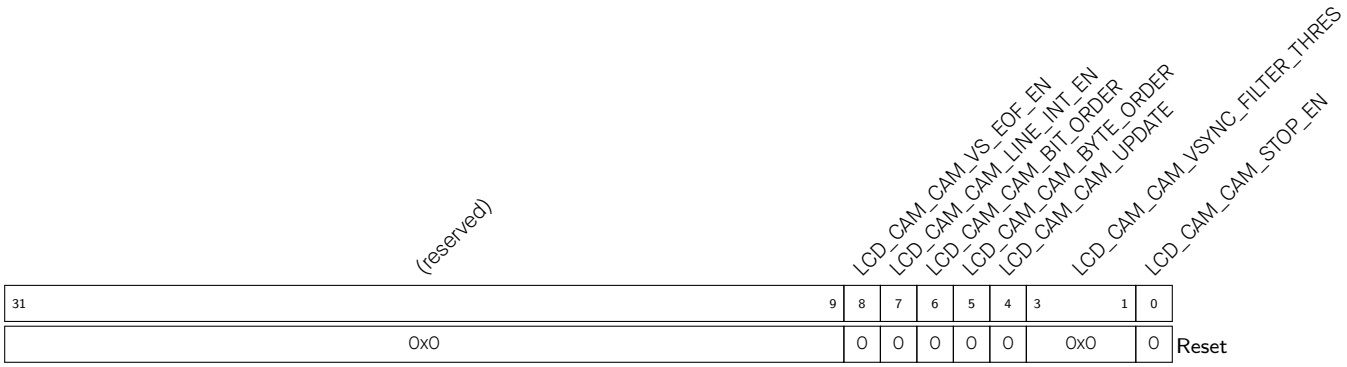
0: No delay.

1: Delayed at the rising edge of LCD_CLK.

2: Delayed at the falling edge of LCD_CLK.

(R/W)

Register 30.12. LCD_CAM_CAM_CTRL_REG (0x0004)



LCD_CAM_CAM_STOP_EN Configures whether to stop the Camera module.

0: Do not stop.

1: The Camera module stops when GDMA RX FIFO is full.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_THRES Configures the filter threshold value for CAM_VSYNC signal. (R/W)

LCD_CAM_CAM_UPDATE Configures whether to update the Camera register configurations.

0: No effect.

1: Update configurations. This bit is cleared by hardware.

(R/W)

LCD_CAM_CAM_BYTE_ORDER Configures whether to invert the byte order of the Camera's input data.

0: Do not invert.

1: Invert data byte order, only valid in 16-bit mode.

(R/W)

LCD_CAM_CAM_BIT_ORDER Configures whether to invert the bit order of the Camera's input data.

0: Do not invert.

1: Invert the bit order. In n-bit mode, CAM_DATA_in[n-1:0] is invertd to CAM_DATA_in[0:n-1]. n = 8 or 15.

(R/W)

LCD_CAM_CAM_LINE_INT_EN Configures whether the RAW bit of LCD_CAM_CAM_HS_INT can generate.

0: Disable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW cannot generate.

1: Enable the interrupt, i.e., LCD_CAM_CAM_HS_INT_RAW can generate.

(R/W)

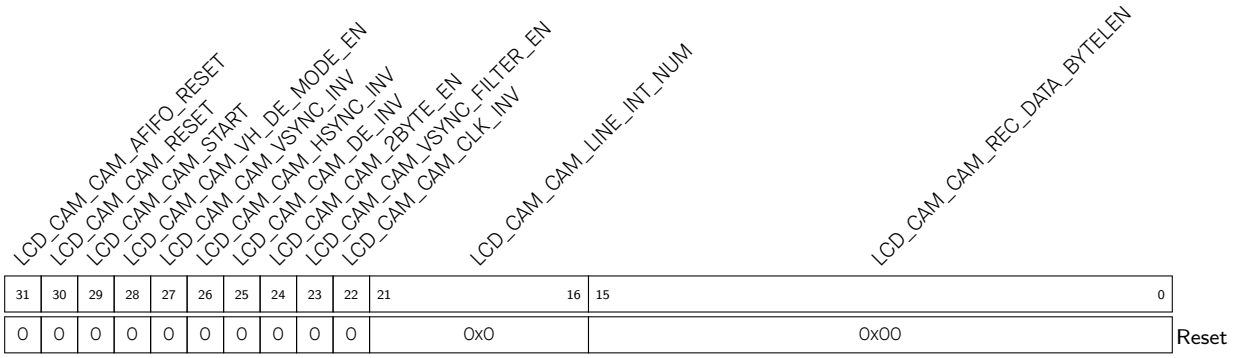
LCD_CAM_CAM_VS_EOF_EN Configures whether to enable CAM_VSYNC to generate in_suc_eof.

0: in_suc_eof is controlled by LCD_CAM_CAM_REC_DATA_BYTELEN.

1: in_suc_eof is controlled by CAM_VSYNC.

(R/W)

Register 30.13. LCD_CAM_CAM_CTRL1_REG (0x0008)



LCD_CAM_CAM_REC_DATA_BYTELEN Configures the data byte length received by the Camera module. When the length of received data reaches this value + 1, GDMA in_suc_eof_int is triggered. (R/W)

LCD_CAM_CAM_LINE_INT_NUM Configures the number of video lines. When the number of video lines reaches this value + 1, LCD_CAM_CAM_HS_INT is triggered. (R/W)

LCD_CAM_CAM_CLK_INV Configures whether to invert the input signal CAM_PCLK.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_EN Configures whether to enable CAM_VSYNC filter function.

0: Bypass.

1: Enable.

(R/W)

LCD_CAM_CAM_2BYTE_EN Configures the width of input data.

0: 8 bits.

1: 16 bits.

(R/W)

LCD_CAM_CAM_DE_INV Configures whether to invert the input signal CAM_DE.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_HSYNC_INV Configures whether to invert the input signal CAM_HSYNC.

0: Do not invert.

1: Invert.

(R/W)

LCD_CAM_CAM_VSYNC_INV Configures whether to invert the input signal CAM_VSYNC.

0: Do not invert.

1: Invert.

(R/W)

Continued on the next page...

Register 30.13. LCD_CAM_CAM_CTRL1_REG (0x0008)

Continued from the previous page...

LCD_CAM_CAM_VH_DE_MODE_EN Configures the input control signals.

0: VSYNC and DE signals control the data. In this case, wiring HSYNC signal line is not a must. But in this case, the YUV-RGB conversion function of the camera module is not available.

1: VSYNC, HSYNC, and DE signals control the data. In this case, users need to wire the three signal lines.

(R/W)

LCD_CAM_CAM_START Camera module start signal. (R/W)

LCD_CAM_CAM_RESET When set to 1, the Camera module is reset. (WO)

LCD_CAM_CAM_AFIFO_RESET When set to 1, the Camera Async RX FIFO is reset. (WO)

Register 30.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

Bits	Field	Reset
31	LCD_CAM_CAM_CONV_ENABLE	0
30	LCD_CAM_CAM_CONV_TRANS_MODE	0
29	LCD_CAM_CAM_CONV_MODE_8BITS_ON	0
28	LCD_CAM_CAM_CONV_DATA_IN_MODE	0
27	LCD_CAM_CAM_CONV_DATA_OUT_MODE	0
26	LCD_CAM_CAM_CONV_PROTOCOL_MODE	0
25:24	LCD_CAM_CAM_CONV_YUV_MODE	0
23:22	LCD_CAM_CAM_CONV_YUV2YUV_MODE	3
21	LCD_CAM_CAM_CONV_8BITS_DATA_INV	0
20:0	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

LCD_CAM_CAM_CONV_8BITS_DATA_INV Configures whether to swap every two bytes of input data.
0: Do not invert.
1: Invert.
(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 the Camera input data in YUV-to-YUV mode or YUV-to-RGB mode.
0: Data is converted to YUV422 format.
1: Data is converted to YUV420 format.
2: Data is converted to 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 30.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

Continued from the previous page...

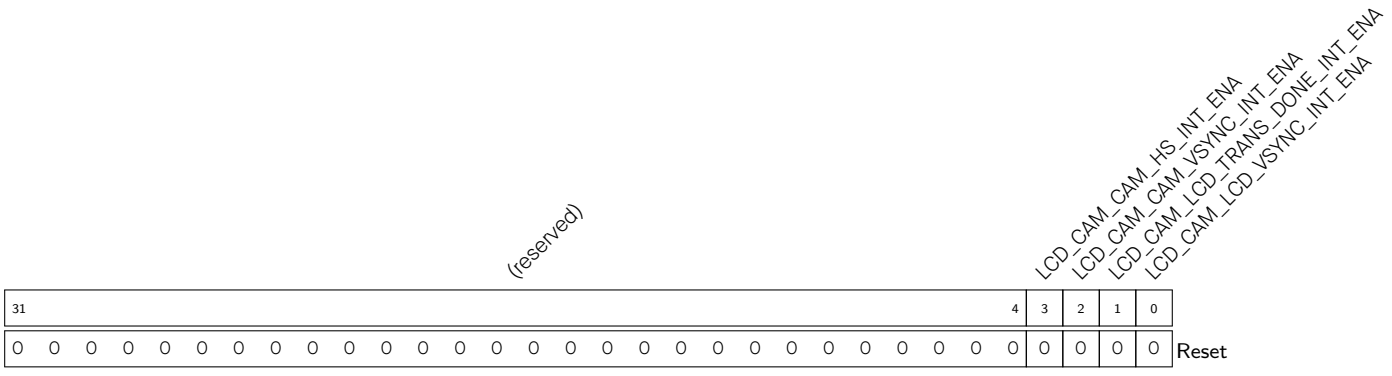
LCD_CAM_CAM_CONV_DATA_IN_MODE Configures the color range for the YUV-RGB converter's input data.
0: Limited color range.
1: Full color range.
(R/W)

LCD_CAM_CAM_CONV_MODE_8BITS_ON Configures the bit width of the YUV-RGB converter's input data.
0: 16-bit mode.
1: 8-bit mode.
(R/W)

LCD_CAM_CAM_CONV_TRANS_MODE Configures the data conversion format.
0: Data is converted to RGB format.
1: Data is converted to YUV format.
(R/W)

LCD_CAM_CAM_CONV_ENABLE Configures whether to enable the YUV-RGB converter.
0: Bypass.
1: Enable.
(R/W)

Register 30.15. LCD_CAM_LC_DMA_INT_ENA_REG (0x0064)



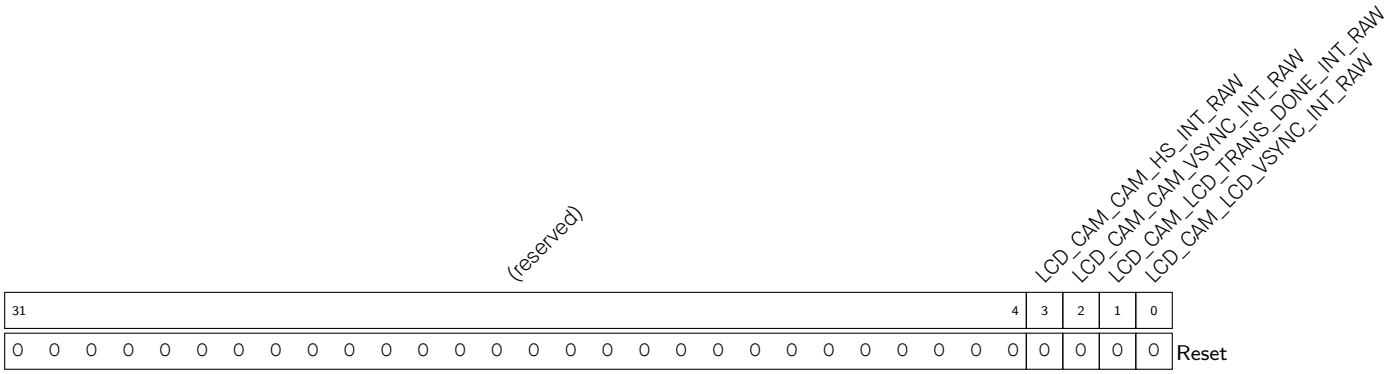
LCD_CAM_LCD_VSYNC_INT_ENA Write 1 to enable LCD_CAM_LCD_VSYNC_INT. (R/W)

LCD_CAM_LCD_TRANS_DONE_INT_ENA Write 1 to enable LCD_CAM_LCD_TRANS_DONE_INT. (R/W)

LCD_CAM_CAM_VSYNC_INT_ENA Write 1 to enable LCD_CAM_CAM_VSYNC_INT. (R/W)

LCD_CAM_CAM_HS_INT_ENA Write 1 to enable LCD_CAM_CAM_HS_INT. (R/W)

Register 30.16. LCD_CAM_LC_DMA_INT_RAW_REG (0x0068)



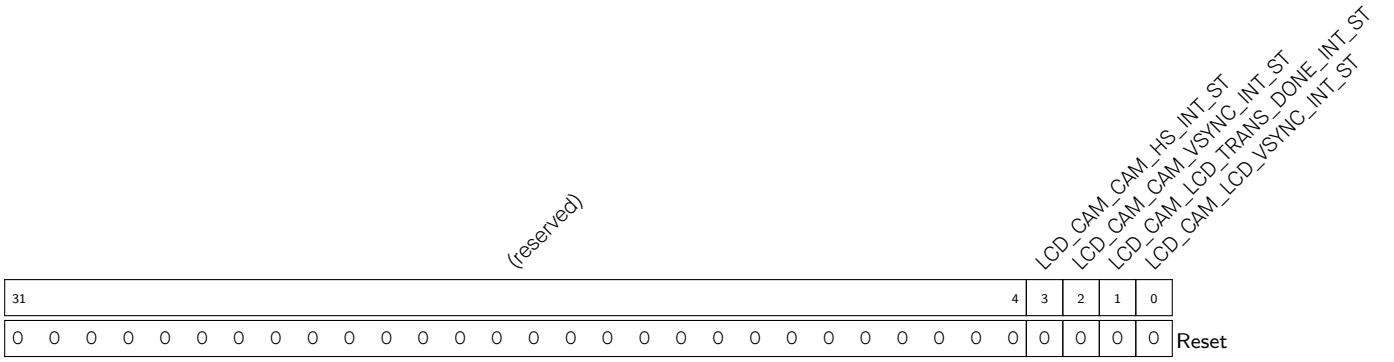
LCD_CAM_LCD_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_RAW The raw interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_RAW The raw interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_RAW The raw interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 30.17. LCD_CAM_LC_DMA_INT_ST_REG (0x006C)



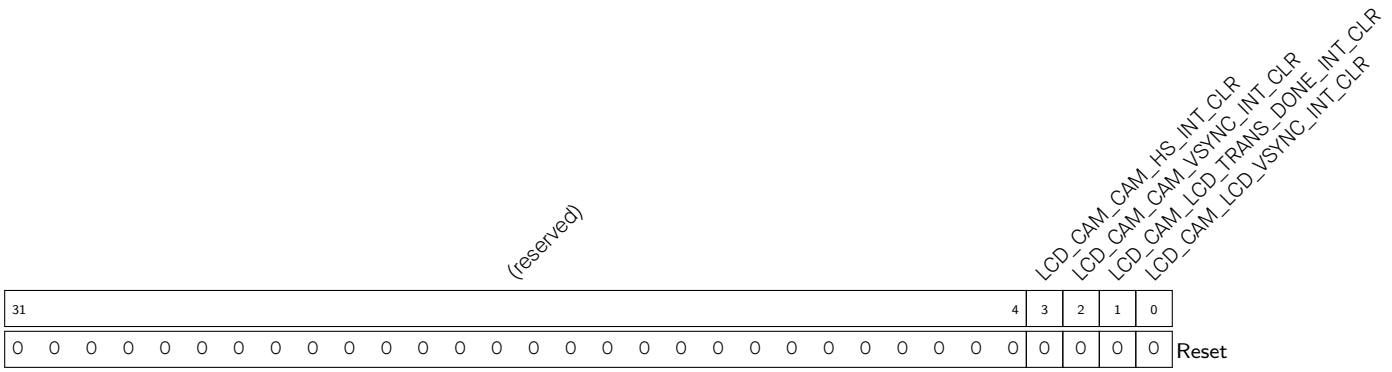
LCD_CAM_LCD_VSYNC_INT_ST The masked interrupt status of LCD_CAM_LCD_VSYNC_INT. (RO)

LCD_CAM_LCD_TRANS_DONE_INT_ST The masked interrupt status of LCD_CAM_LCD_TRANS_DONE_INT. (RO)

LCD_CAM_CAM_VSYNC_INT_ST The masked interrupt status of LCD_CAM_CAM_VSYNC_INT. (RO)

LCD_CAM_CAM_HS_INT_ST The masked interrupt status of LCD_CAM_CAM_HS_INT. (RO)

Register 30.18. LCD_CAM_LC_DMA_INT_CLR_REG (0x0070)



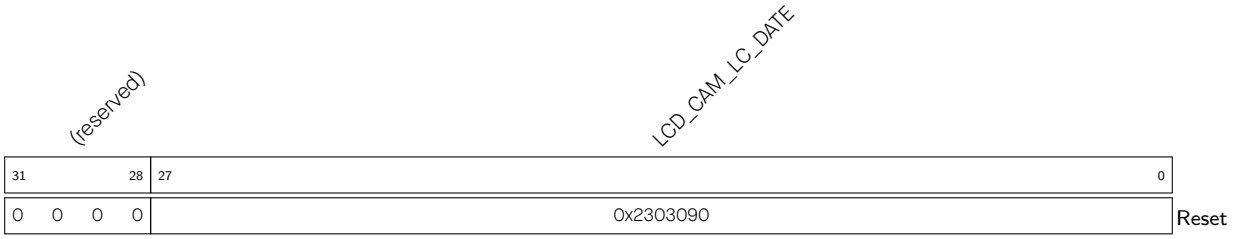
LCD_CAM_LCD_VSYNC_INT_CLR Write 1 to clear LCD_CAM_LCD_VSYNC_INT. (WO)

LCD_CAM_LCD_TRANS_DONE_INT_CLR Write 1 to clear LCD_CAM_LCD_TRANS_DONE_INT. (WO)

LCD_CAM_CAM_VSYNC_INT_CLR Write 1 to clear LCD_CAM_CAM_VSYNC_INT. (WO)

LCD_CAM_CAM_HS_INT_CLR Write 1 to clear LCD_CAM_CAM_HS_INT. (WO)

Register 30.19. LCD_CAM_LC_REG_DATE_REG (0x00FC)



LCD_CAM_LC_DATE Version control register. (R/W)

Part VII

Connectivity Interface

This part addresses the connectivity aspects of the system, describing components related to various communication interfaces like I2C, I2S, SPI, UART, USB, and more. The part also covers interfaces to generate signals used in remote control, motor control, LED control, etc.

Chapter 31

UART Controller (UART)

31.1 Overview

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to the data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit(s) and parity bit(s).

A UART data frame usually begins with one start bit, followed by data bits, one parity bit (optional), and one or more stop bits. UART controllers on ESP32-P4 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as GDMA for high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

ESP32-P4 has six UART controllers, including five regular UARTs and one low-power (LP) UART. These UARTs are compatible with various UART devices, and support Infrared Data Association (IrDA) and RS485 communication. In this chapter, the five regular UART controllers are referred to as UART*n*, in which *n* denotes 0, 1, 2, 3, or 4. LP UART is the cut-down version of the regular UART, with a separate group of registers. For differences between UART and LP UART, please refer to Table 31.2-1.

31.2 Features

Table 31.2-1 lists the feature comparison between UART and LP UART:

Table 31.2-1. UART and LP UART Feautre Comparison

UART Feature	LP UART Feature
Programmable baud rate up to 5 MBaud	
260 x 8-bit RAM, shared by TX FIFOs and RX FIFOs of the UART controllers	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	—
High-speed data communication using GDMA	—

Cont'd on next page

Table 31.2-1 – cont'd from previous page

UART Feature	LP_UART Feature
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.

31.3 UART Structure

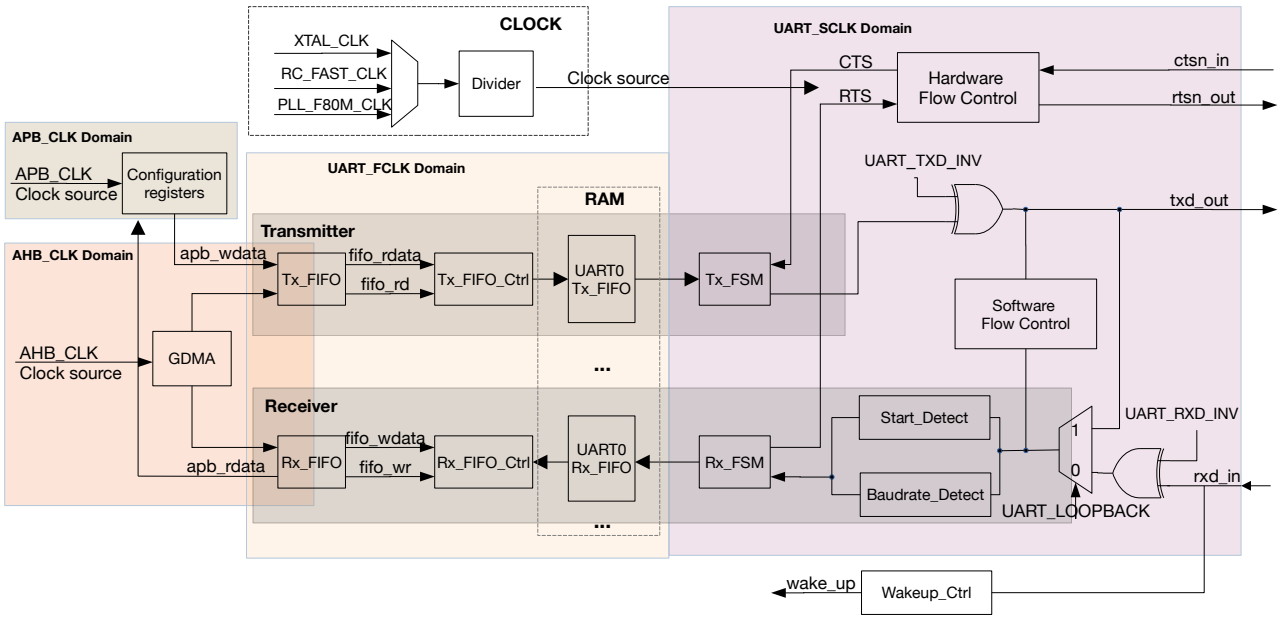


Figure 31.3-1. UART Structure

Figure 31.3-1 shows the basic structure of a UART controller. A UART controller works in four clock domains, namely APB_CLK, AHB_CLK, UART_SCLK, and UART_FCLK. APB_CLK and AHB_CLK are synchronized but with different frequencies (APB_CLK is derived from AHB_CLK by division), and likewise UART_SCLK and UART_FCLK are synchronized but with different frequencies (UART_SCLK is derived from UART_FCLK by division). UART_FCLK has three clock sources: PLL_F80M_CLK, RC_FAST_CLK, and crystal clock XTAL_CLK (for details, please refer to Chapter 8 *Reset and Clock*), which are selected by configuring HP_SYS_UART*n*_CLK_SRC_SEL. The selected clock source is divided by a divider to generate UART_SCLK clock signals. The divisor is configured by HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM for the integral part, HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR for the denominator of the fractional part, and HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR for the numerator of the fractional part. The divisor ranges from 1 ~ 256.

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 31.3-1), 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 31.3-1), 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 5 *Low-Power Management [to be added later]* for more details), a wake_up signal can be generated in four ways and sent to RTC, which then wakes up the ESP32-P4 chip. For more information about wakeup, please refer to Section 31.4.8.

31.4 Functional Description

31.4.1 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 Section31.5.

To reset the whole UART, please:

- Enable the UART Core's clock by setting HP_SYS_CLKRST_UART*n*_CLK_EN to 1.
- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_CORE.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_CORE to 0.

31.4.2 UART FIFO

Each UART controller on ESP32-P4 instantiates a 256 × 8-bit RAM respectively. They access the RAM via a 4 × 8-bit asynchronous FIFO interface with a fixed address.

UART Tx_FIFO is reset by setting UART_TXFIFO_RST. UART Rx_FIFO is reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using GDMA, read automatically, and converted from a frame into a bitstream by hardware Tx_FSM. Data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using GDMA. The five UART controllers share one GDMA channel.

The empty signal threshold for Tx_FIFO is configured by setting UART_TXFIFO_EMPTY_THRHD. When data stored in Tx_FIFO is less than UART_TXFIFO_EMPTY_THRHD, a UART_TXFIFO_EMPTY_INT interrupt is generated. The full signal threshold for Rx_FIFO is configured by setting UART_RXFIFO_FULL_THRHD. When data stored in Rx_FIFO is greater than 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.

31.4.3 Baud Rate Generation and Detection

31.4.3.1 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 31.4-1, 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 31.4-1 below, to make the output timing more uniform:

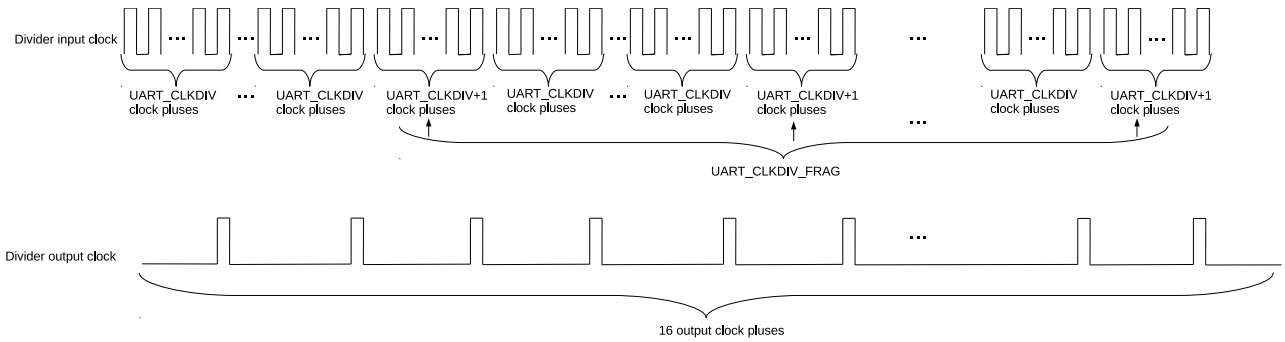


Figure 31.4-1. UART Controllers Division

To support IrDA (see Section 31.4.7 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.

31.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 31.4-2 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.

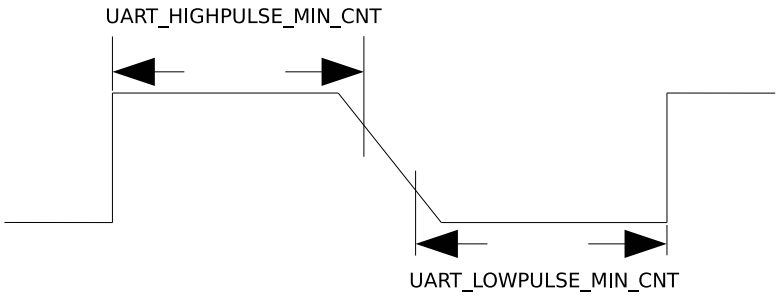


Figure 31.4-2. The Timing Diagram of Weak UART Signals Along Negative Edges

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 31.4-2, 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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

31.4.4 UART Data Frame

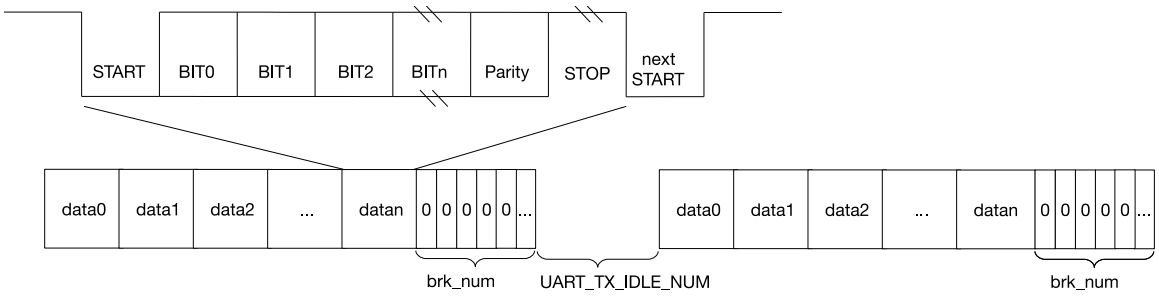


Figure 31.4-3. Structure of UART Data Frame

Figure 31.4-3 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 31.4.6.2). The start bit is logical low, whereas stop bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in the data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received will still be stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set, then the transmitter will enter the break condition and send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK_IDLE_DONE_INT interrupt is generated.

The receiver can also detect the Break conditions when the RX data line detects any low logical level for one NULL character transmission, and a UART_BRK_DET_INT interrupt will be triggered to detect when a break condition has been completed.

The receiver can detect the current bus state through the timeout interrupt UART_RXFIFO_TOUT_INT. The UART_RXFIFO_TOUT_INT interrupt will be triggered when the bus is in the idle state for more than UART_RX_TOUT_THRHD bit time on current baud rate after the receiver has received at least one byte. You can use this interrupt to detect whether all the data from the transmitter has been sent.

31.4.5 AT_CMD Character Structure

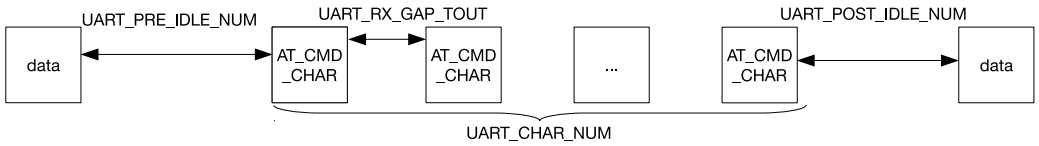


Figure 31.4-4. AT_CMD Character Structure

Figure 31.4-4 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.

31.4.6 RS485

The five regular UART controllers support RS485 communication mode. In this mode differential signals are used to transmit data, so it can communicate over longer distances at higher bit rates than RS232. RS485 has two-wire half-duplex and four-wire full-duplex options. UART controllers support two-wire half-duplex transmission and bus snooping.

31.4.6.1 Driver Control

As shown in Figure 31.4-5, 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 31.4-5, DE is connected to dtrn_out of UART (please refer to Section 31.4.9.1 for more details).

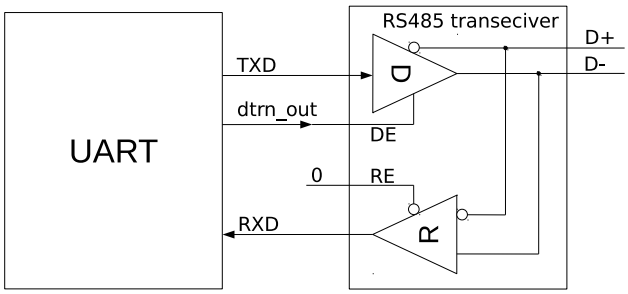


Figure 31.4-5. Driver Control Diagram in RS485 Mode

31.4.6.2 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.

31.4.6.3 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 31.4-5, a UART controller may receive data in transmitter mode and snoop the bus. If UART_RS485RXBY_TX_EN is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop the data sent by themselves. In transmitter mode, when a UART controller monitors a collision between the data sent and the data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitors a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a parity bit error, a UART_RS485_PARITY_ERR_INT is generated.

31.4.7 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 31.4-6, 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.

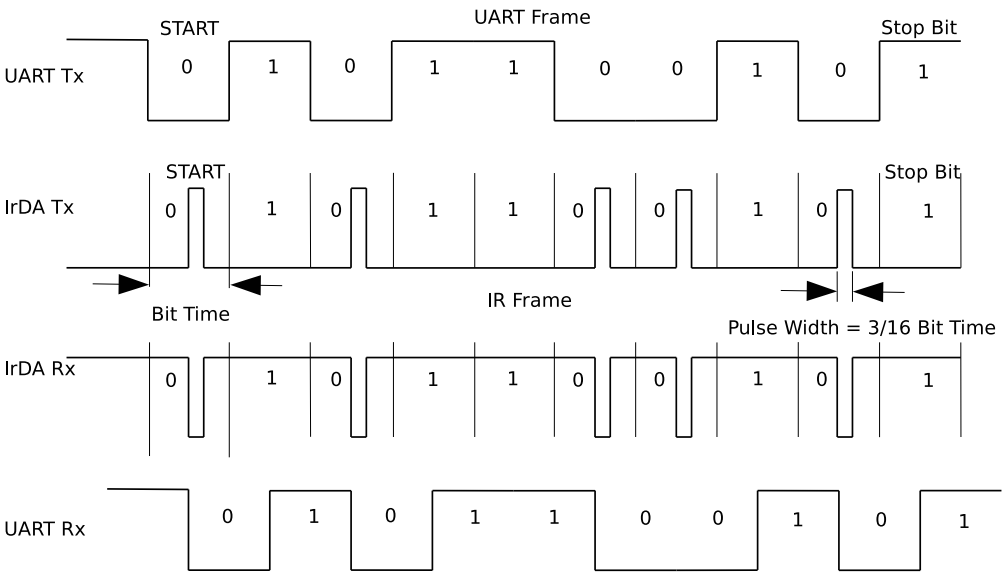


Figure 31.4-6. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 31.4-7, 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.

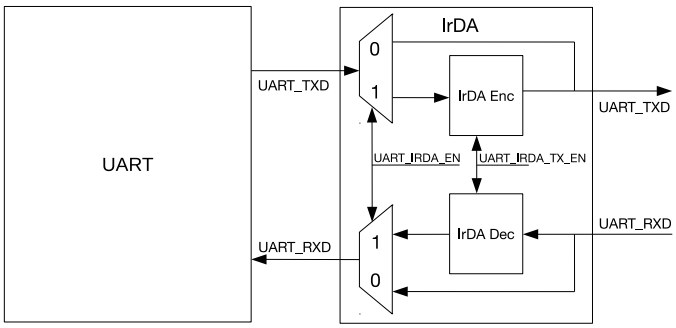


Figure 31.4-7. IrDA Encoding and Decoding Diagram

31.4.8 Wakeup

UART can be set as wakeup source. When a UART controller is in Light-sleep mode, a wake_up signal can be generated in four ways and be sent to the RTC module, which then wakes up ESP32-P4.

- UART_WK_MODE_SEL = 0: When all the clocks are disabled, the chip can be woken up by reverting RXD for multiple cycles until the number of positive edges is greater than or equal to UART_ACTIVE_THRESHOLD + 3.
- UART_WK_MODE_SEL = 1: UART Core keeps working, so the UART receiver can still receive data and store the received data in RX FIFO. When the number of data bytes in RX FIFO is greater than UART_RX_WAKE_UP_THRHD, the chip can be woken up from the Light-sleep mode.
- UART_WK_MODE_SEL = 2: When the UART receiver detects a start bit, the chip will be woken up.

- UART_WK_MODE_SEL = 3: When the UART receiver receives a specific character sequence, the chip will be woken up. The wakeup characters can be defined by configuring UART_WK_CHAR0, UART_WK_CHAR1, UART_WK_CHAR2, UART_WK_CHAR3, and UART_WK_CHAR4. These characters can be formed into different character sequences by configuring UART_CHAR_NUM and UART_WK_CHAR_MASK, as shown in Table 31.4-1. Once the sequence is detected, the chip will be woken up. For the last configuration in Table 31.4-1, UART will detect for CHAR0 ~ CHAR4 in order.

Table 31.4-1. 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.

31.4.9 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.

31.4.9.1 Hardware Flow Control

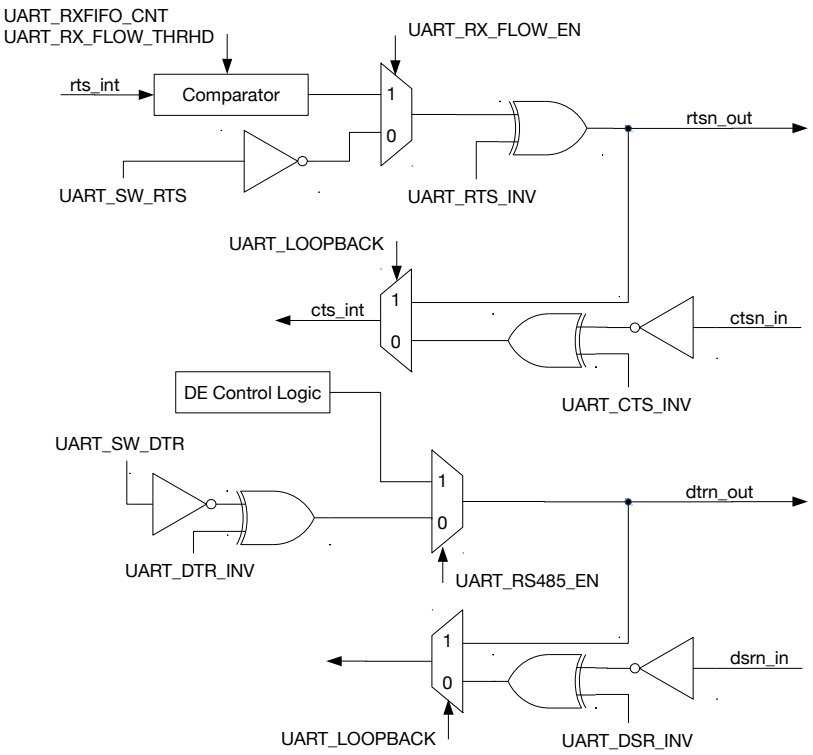


Figure 31.4-8. Hardware Flow Control Diagram

Figure 31.4-8 shows the hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 31.4-9 illustrates how these signals are connected between UART on ESP32-P4 (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

When rtsn_out of IU0 is low, EU0 is allowed to send data. When rtsn_out of IU0 is high, EU0 is notified to stop sending data until rtsn_out of IU0 returns to low. The output signal rtsn_out can be controlled in two ways.

- Software control: Enter this mode by clearing UART_RX_FLOW_EN to 0. In this mode, the level of rtsn_out is changed by configuring UART_SW_RTS.
- Hardware control: Enter this mode by setting UART_RX_FLOW_EN to 1. In this mode, rtsn_out is pulled high when data in Rx_FIFO exceeds UART_RX_FLOW_THRHD.

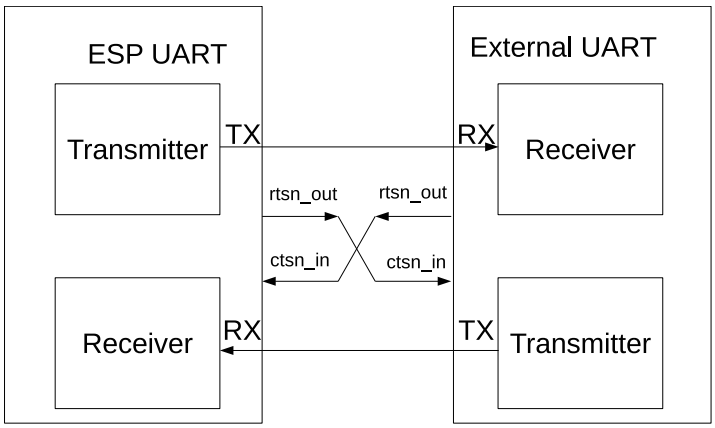


Figure 31.4-9. Connection between Hardware Flow Control Signals

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 IU0 is high, it indicates that IU0 is ready to transmit data. dtrn_out is generated by configuring the UART_SW_DTR field. When the IU0 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 EU0 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.

31.4.9.2 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.

31.4.10 GDMA Mode

The five UART controllers on ESP32-P4 share one TX/RX GDMA (General Direct Memory Access) channel via UHCI (Universal Host Controller Interface). In GDMA mode, UART controllers support the decoding and encoding of HCI data packets. The UHCI_UART_SEL field determines which UART controller occupies the GDMA TX/RX channel.

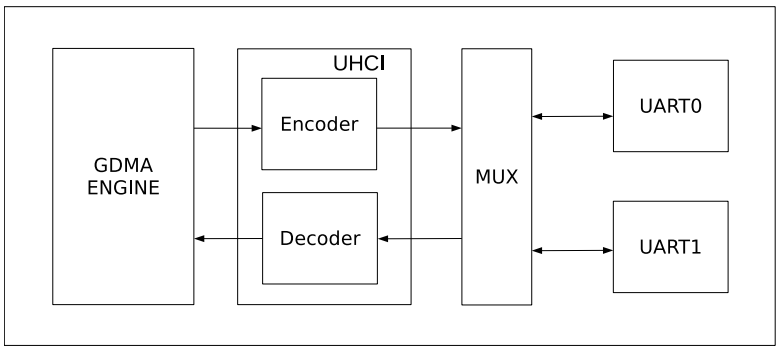


Figure 31.4-10. Data Transfer in GDMA Mode

Figure 31.4-10 shows how data is transferred using GDMA. Before GDMA receives data, software prepares an inlink. GDMA_INLINK_ADDR_CH*n* points to the first receive descriptor in the inlink. After GDMA_INLINK_START_CH*n* is set, UHCI sends data that UART has received to the decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of GDMA.

Before GDMA sends data, software prepares an outlink and data to be sent. GDMA_OUTLINK_ADDR_CH*n* points to the first transmit descriptor in the outlink. After GDMA_OUTLINK_START_CH*n* is set, GDMA reads data from the RAM pointed by outlink. The data is then encoded by the encoder, and sent sequentially by the UART transmitter.

HCI data packets have separators at the beginning and the end, with data bits in the middle (separators + data bits + separators). The encoder inserts separators in front of and after data bits, and replaces data bits identical to separators with special characters. The decoder removes separators in front of and after data bits, and replaces special characters with separators. There can be more than one continuous separator at the beginning and the end of a data packet. The separator is configured by UHCI_SEPER_CHAR, 0xC0 by default. The special character is configured by UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). When all data has been sent, a GDMA_OUT_TOTAL_EOF_CH*n*_INT interrupt is generated. When all data has been received, a GDMA_IN_SUC_EOF_CH*n*_INT is generated.

31.4.11 Interrupts

ESP32-P4's UART*n* and UHCI can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- UART*n*_INTR
- UHCI_INTR

There are several internal interrupt sources from UART*n* and UHCI that can generate the above interrupt signals.

UART*n* interrupt sources are listed as follows:

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.
- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than UART_RX_TOUT_THRHD to receive one byte.
- 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 receiver has received at least one byte, and the bus remains idle for UART_RX_TOUT_THRHD bit time.
- 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 31.6 *Register Summary*.

31.5 Programming Procedures

31.5.1 Register Type

All UART registers are in the APB_CLK domain.

UART configuration registers can be classified into two groups. One group of registers are read in the APB_CLK or AHB_CLK domains, so once such registers are configured no extra operations are required. The other group of registers are read in the UART_SCLK domain, and therefore need to implement the clock domain crossing design. Once these registers are configured, the configured values need to be synchronized to the UART Core's clock domain by writing to UART_REG_UPDATE. Once all values have been synchronized, UART_REG_UPDATE will be automatically cleared by hardware. After configuring registers that need synchronization, it is recommended to check whether UART_REG_UPDATE is 0. This is to ensure that register values configured before have already been synchronized.

To distinguish between these two groups of registers easily, all registers that implement the clock domain crossing design have the _SYNC suffix, and are put together in Section 31.6. Those without the _SYNC suffix in Section 31.6 are configuration registers that require no clock domain crossing.

31.5.2 Detailed Steps

Figure 31.5-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

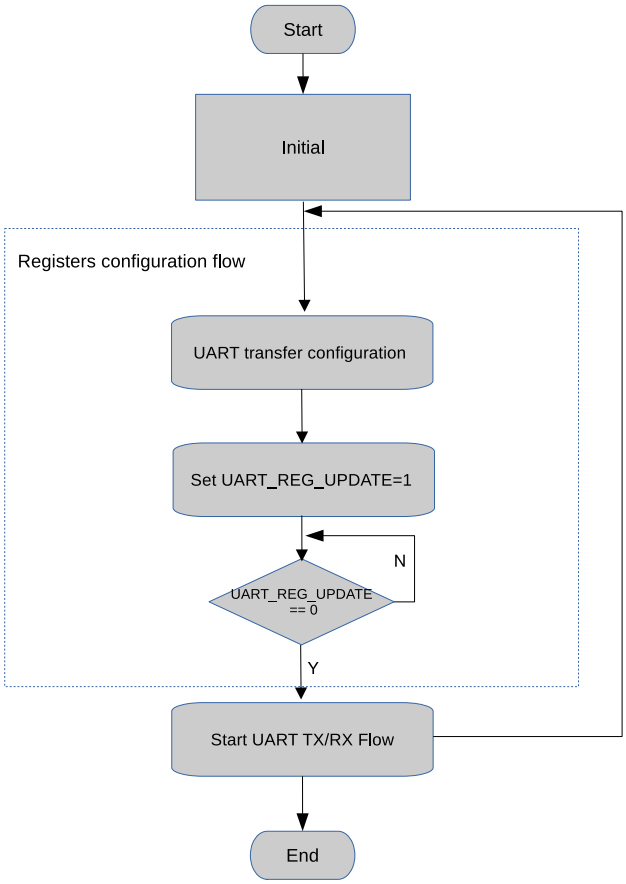


Figure 31.5-1. UART Programming Procedures

31.5.2.1 Initializing UART*n*

To initialize UART*n*:

- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_APB.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_APB to 0.
- Write 1 to HP_SYS_CLKRST_RST_EN_UART*n*_CORE.
- Clear HP_SYS_CLKRST_RST_EN_UART*n*_CORE to 0.

31.5.2.2 Configuring UART*n* Communication

To configure UART*n* communication:

- Wait for UART_REG_UPDATE to become 0, which indicates the completion of the last synchronization.
- Select the clock source via HP_SYS_CLKRST_UART*n*_CLK_SRC_SEL.
- Configure divisor of the divider via HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM, HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR, and

HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR.

- Configure the baud rate for transmission via UART_CLKDIV and UART_CLKDIV_FRAG.
- Configure data length via UART_BIT_NUM.
- Configure odd or even parity check via UART_PARITY_EN and UART_PARITY.
- Optional steps depending on application ...
- Synchronize the configured values to the Core Clock domain by writing 1 to UART_REG_UPDATE.

31.5.2.3 Enabling UART*n*

To enable UART*n* transmitter:

- Configure TX FIFO's empty threshold via UART_TXFIFO_EMPTY_THRHD.
- Disable UART_TXFIFO_EMPTY_INT interrupt by clearing UART_TXFIFO_EMPTY_INT_ENA.
- Write data to be sent to UART_RXFIFO_RD_BYTE.
- Clear UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_CLR.
- Enable UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_ENA.
- Check UART_TXFIFO_EMPTY_INT_ST and wait for the completion of data transmission.

To enable UART*n* receiver:

- Configure RX FIFO's full threshold via UART_RXFIFO_FULL_THRHD.
- Enable UART_RXFIFO_FULL_INT interrupt by setting UART_RXFIFO_FULL_INT_ENA.
- Check UART_RXFIFO_FULL_INT_ST and wait until the RX FIFO is full.
- Read data from RX FIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RX FIFO via UART_RXFIFO_CNT.

31.6 Register Summary

31.6.1 UART Register Summary

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

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

Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART threshold and allocation configuration	0x0064	R/W
UART Interrupt Register			
UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
UART_SLEEP_CONF0_REG	UART sleep configuration register 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART sleep configuration register 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART sleep configuration register 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 mode configuration	0x004C	R/W
UART_CLK_CONF_REG	UART core clock configuration	0x0088	R/W
UART_REG_UPDATE_REG	UART register configuration update	0x0098	R/W/SC
UART_ID_REG	UART ID register	0x009C	R/W
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
UART_MEM_RX_STATUS_REG	Rx FIFO write and read offset address	0x006C	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0070	RO
UART_AFIFO_STATUS_REG	UART asynchronous FIFO status	0x0090	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W

Name	Description	Address	Access
UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Autobaud Register			
UART_POSPULSE_REG	Autobaud high pulse register	0x0074	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0078	RO
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x007C	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x0080	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0084	RO
Version Register			
UART_DATE_REG	UART version control register	0x008C	R/W

31.6.2 LP UART Register Summary

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

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

Name	Description	Address	Access
FIFO Configuration			
LP_UART_FIFO_REG	FIFO data register	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART threshold and allocation configuration	0x0064	R/W
LP UART Interrupt Register			
LP_UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	Masked interrupt status	0x0008	RO
LP_UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
LP_UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
LP_UART_CLKDIV_SYNC_REG	Clock divider configuration	0x0014	R/W
LP_UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
LP_UART_CONF0_SYNC_REG	Configuration register 0	0x0020	R/W
LP_UART_CONF1_REG	Configuration register 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	Hardware flow control configuration	0x002C	R/W
LP_UART_SLEEP_CONF0_REG	LP UART sleep configuration register 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART sleep configuration register 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART sleep configuration register 2	0x0038	R/W
LP_UART_SWFC_CONF0_SYNC_REG	Software flow control character configuration	0x003C	varies
LP_UART_SWFC_CONF1_REG	Software flow control character configuration	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX break character configuration	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	Frame end idle time configuration	0x0048	R/W
LP_UART_CLK_CONF_REG	LP UART core clock configuration	0x0088	R/W

Name	Description	Address	Access
LP_UART_REG_UPDATE_REG	LP UART register configuration update register	0x0098	R/W/SC
LP_UART_ID_REG	LP UART ID register	0x009C	R/W
Status Register			
LP_UART_STATUS_REG	LP UART status register	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART transmit and receive status	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART asynchronous FIFO Status	0x0090	RO
AT Escape Sequence Selection Configuration			
LP_UART_AT_CMD_PRECNT_SYNC_REG	Pre-sequence timing configuration	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	Post-sequence timing configuration	0x0054	R/W
LP_UART_AT_CMD_GAPTOUT_SYNC_REG	Timeout configuration	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT escape sequence detection configuration	0x005C	R/W
Version Register			
LP_UART_DATE_REG	LP UART version register	0x008C	R/W

31.6.3 UHCI Register Summary

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

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

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x0014	varies
UHCI_ESCAPE_CONF_REG	Escape character configuration	0x0020	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0024	R/W
UHCI_ACK_NUM_REG	UHCI ACK number configuration	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 quick send register	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 quick send register	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 quick send register	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 quick send register	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 quick send register	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 quick send register	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 quick send register	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 quick send register	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 quick send register	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 quick send register	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 quick send register	0x005C	R/W

Name	Description	Address	Access
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 quick send register	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 quick send register	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 quick register	0x0068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x006C	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x0070	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x0074	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x0078	R/W
UHCI_PKT_THRES_REG	Configuration register for packet length	0x007C	R/W
UHCI Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	varies
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WT
UHCI Status Register			
UHCI_STATE0_REG	UHCI receive status	0x0018	RO
UHCI_STATE1_REG	UHCI transmit status	0x001C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x002C	RO
Version Register			
UHCI_DATE_REG	UHCI version control register	0x0080	R/W

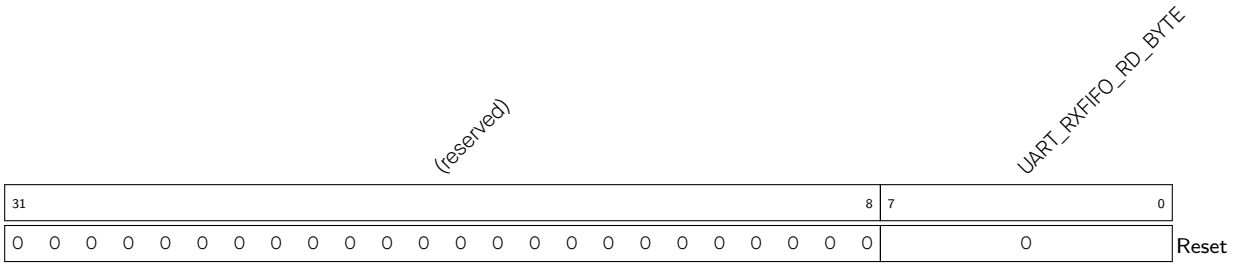
31.7 Registers

31.7.1 UART Registers

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

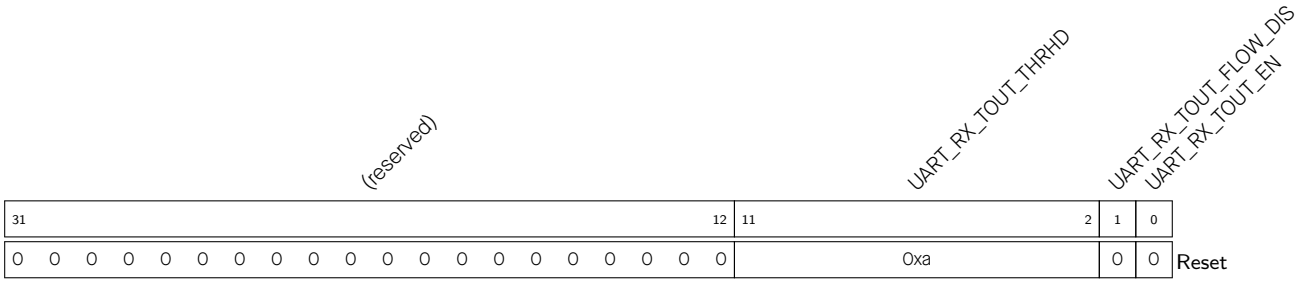
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 31.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE Represents the data UART *n* read from FIFO.
Measurement unit: byte. (RO)

Register 31.2. UART_TOUT_CONF_SYNC_REG (0x0064)

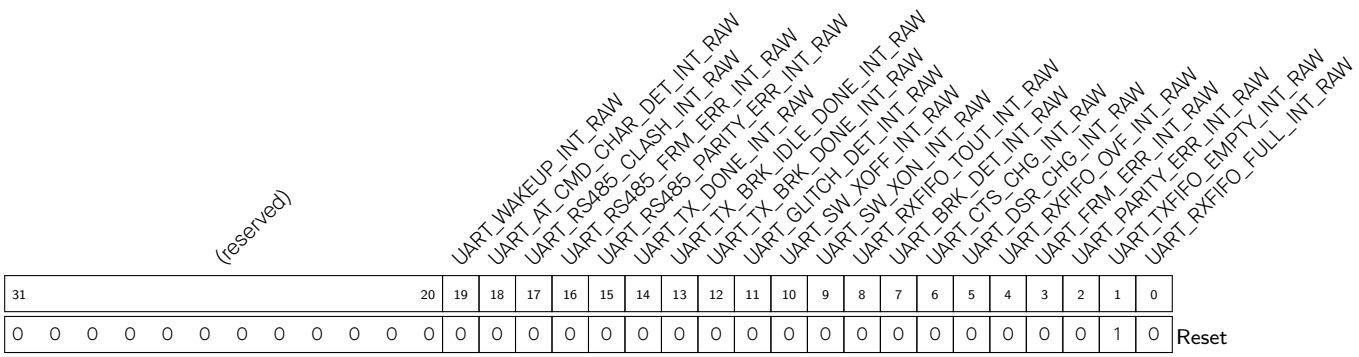


UART_RX_TOUT_EN Configures whether or not to enable UART receiver's timeout function.
0: Disable
1: Enable
(R/W)

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 31.3. 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 31.3. UART_INT_RAW_REG (0x0004)

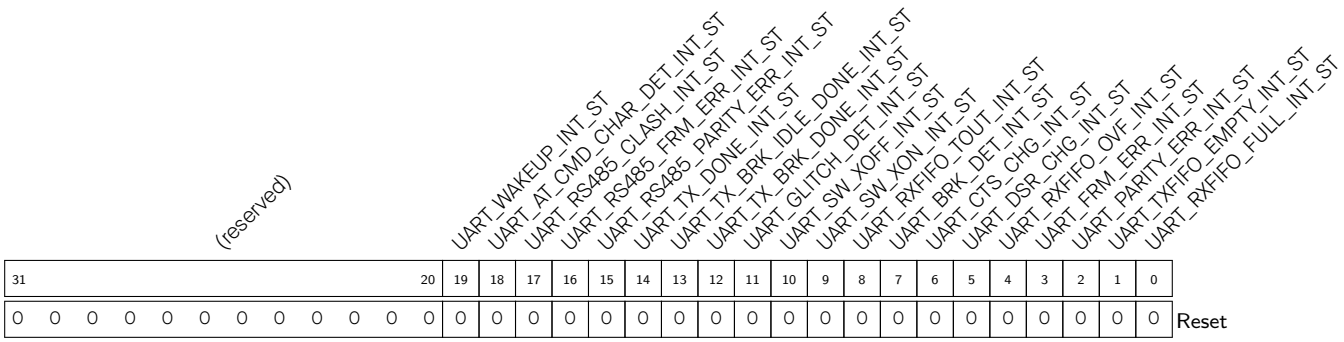
Continued from the previous page...

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 31.4. 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 31.5. 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)

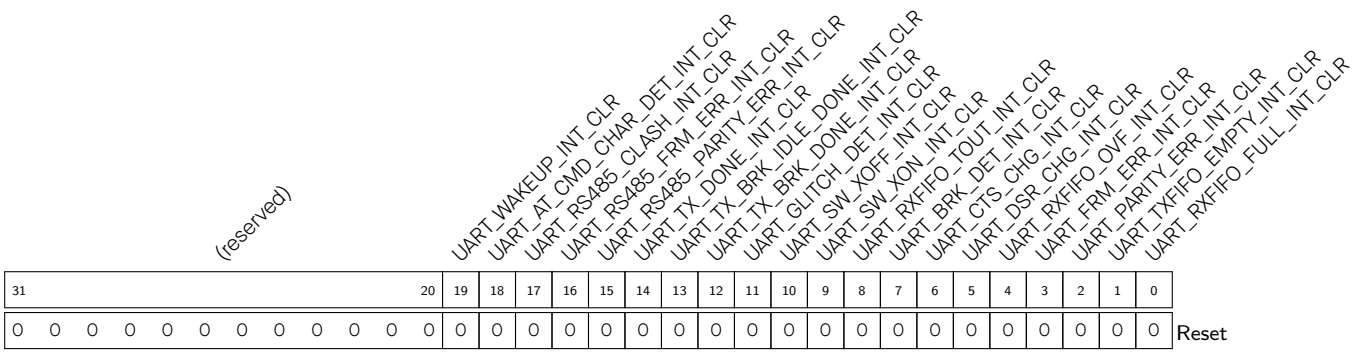
UART_RS485_FRM_ERR_INT_ENA Write 1 to enable UART_RS485_FRM_ERR_INT. (R/W)

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 31.6. 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)

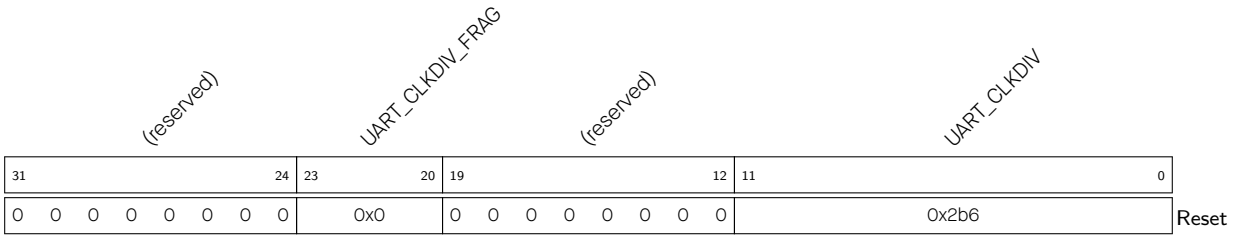
UART_RS485_FRM_ERR_INT_CLR Write 1 to clear UART_RS485_FRM_ERR_INT. (WT)

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)

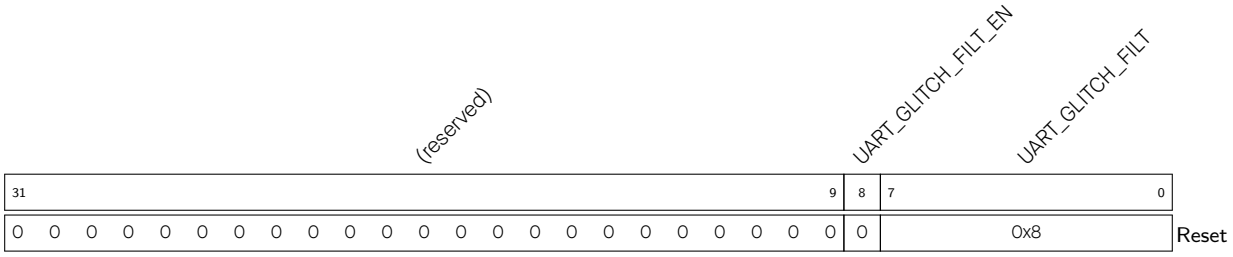
Register 31.7. UART_CLKDIV_SYNC_REG (0x0014)



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 31.8. 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)

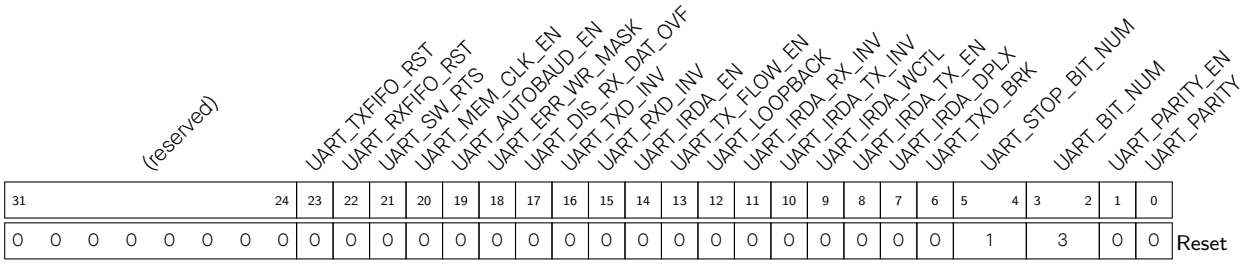
UART_GLITCH_FILT_EN Configures whether or not to enable RX signal filter.

0: Disable

1: Enable

(R/W)

Register 31.9. 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 31.9. 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 31.9. 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 allowed to send data.

1: The UART transmitted is not 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)

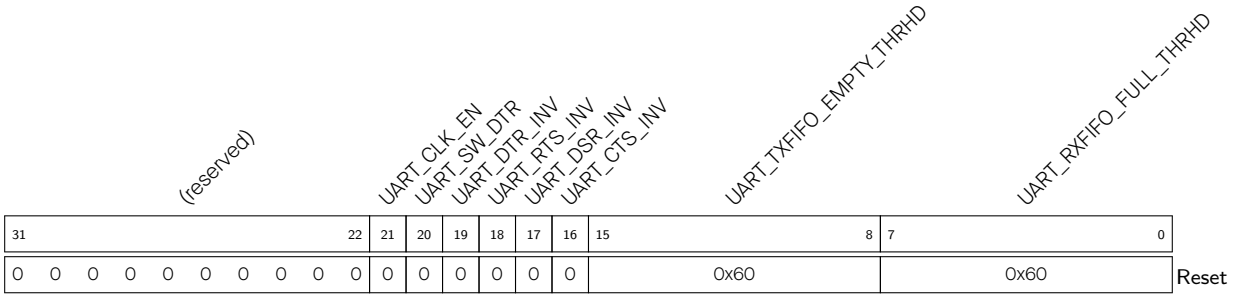
UART_TXFIFO_RST Configures whether or not to reset the UART TX FIFO.

0: Not reset

1: Reset

(R/W)

Register 31.10. 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)

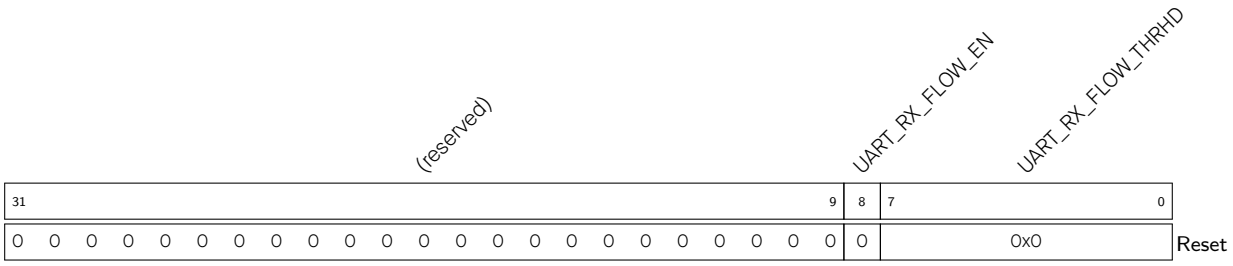
UART_RTS_INV Configures whether or not to invert the level of UART RTS signal.
0: Not invert
1: Invert
(R/W)

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 31.11. 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)

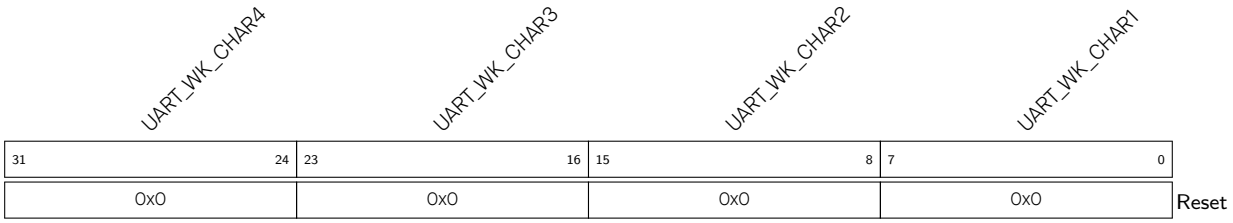
UART_RX_FLOW_EN Configures whether or not to enable the UART receiver.

0: Disable

1: Enable

(R/W)

Register 31.12. UART_SLEEP_CONF0_REG (0x0030)



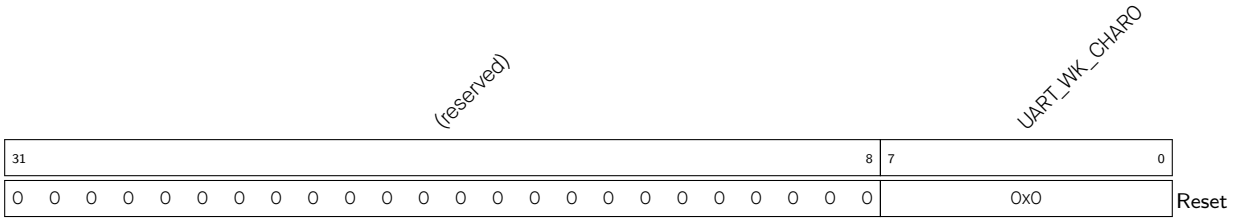
UART_WK_CHAR1 Configures wakeup character 1. (R/W)

UART_WK_CHAR2 Configures wakeup character 2. (R/W)

UART_WK_CHAR3 Configures wakeup character 3. (R/W)

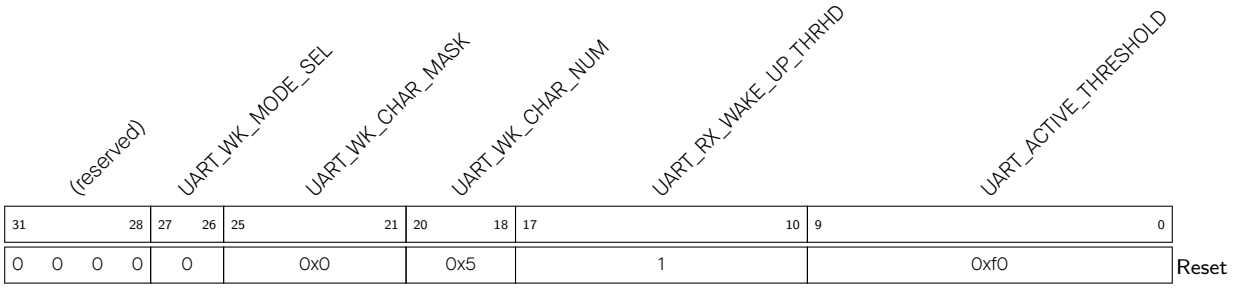
UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 31.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 31.14. 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 31.4.8 *Wakeup* for the explanation of each mode.

0: Mode 0

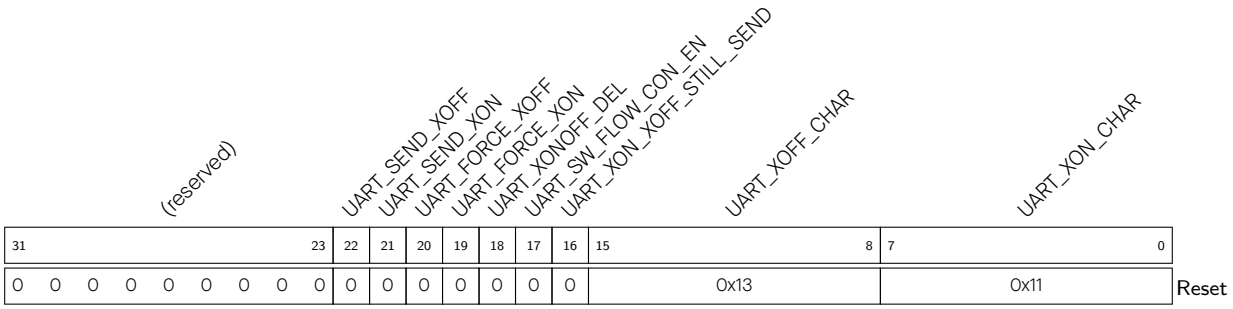
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 31.15. 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)

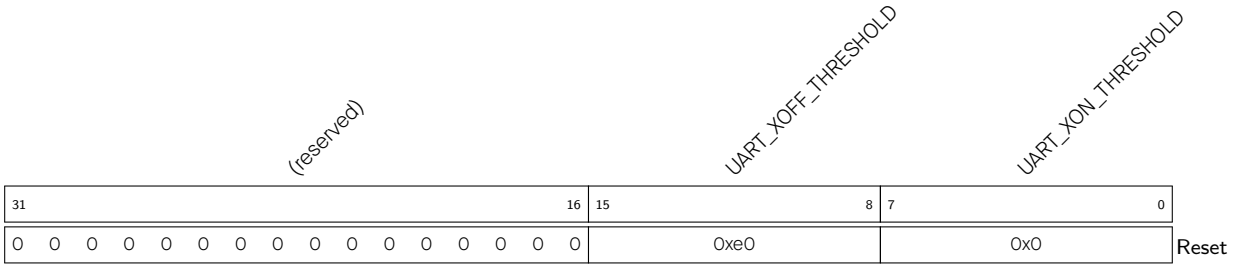
UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 31.16. UART_SWFC_CONF1_REG (0x0040)



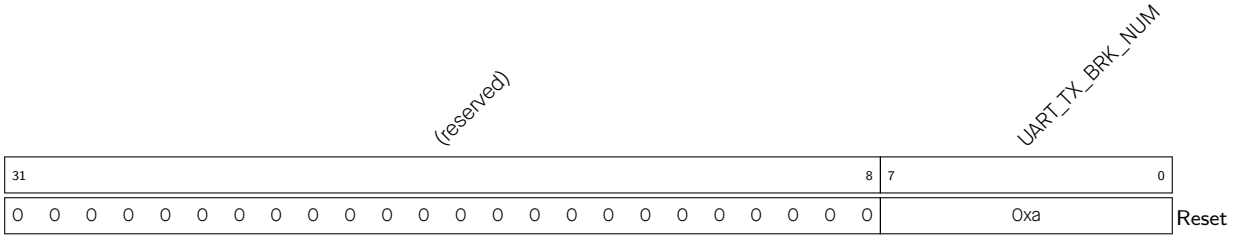
UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

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)

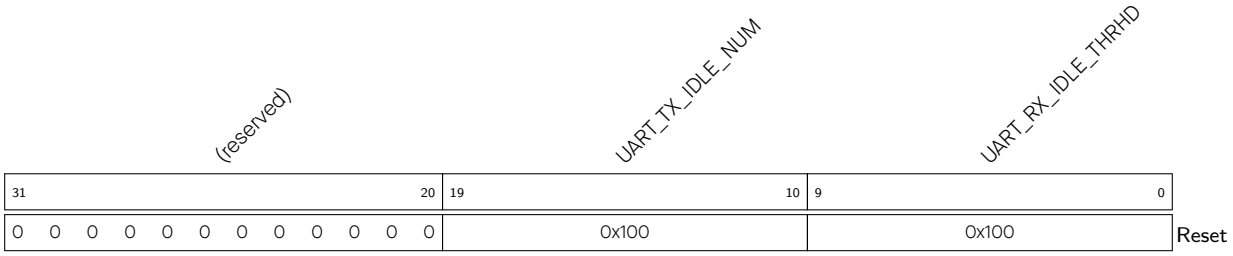
Register 31.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



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 31.18. UART_IDLE_CONF_SYNC_REG (0x0048)



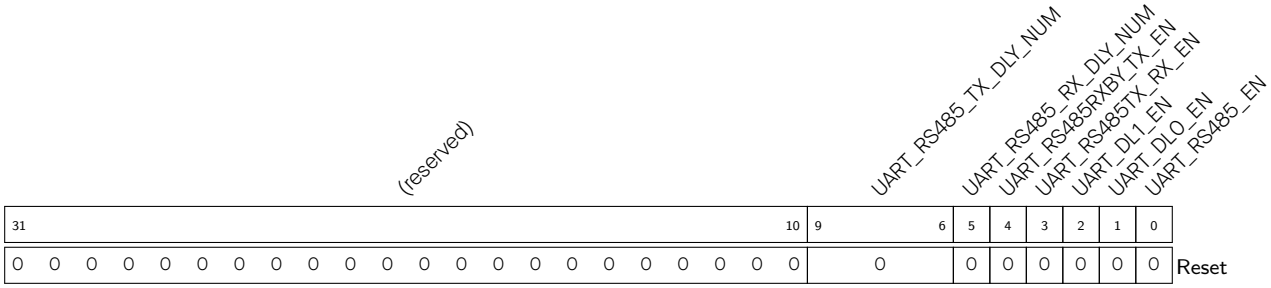
UART_RX_IDLE_THRHD Configures the threshold to generate a frame end signal when the receiver takes more time to receive one data byte data.

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 31.19. 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)

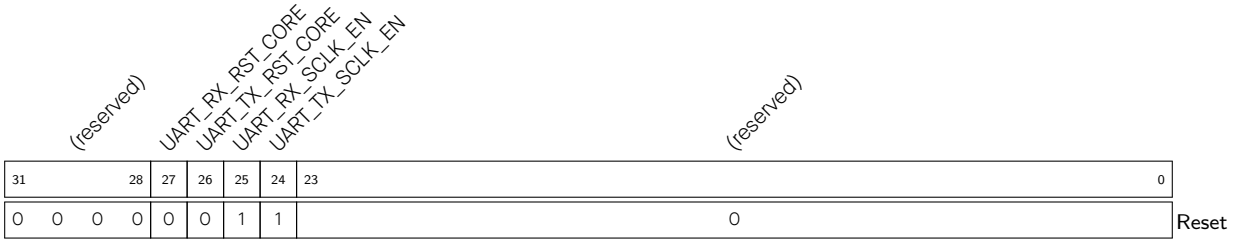
UART_RS485_RX_DLY_NUM Configures the delay of internal data signals in the receiver.

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 31.20. 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

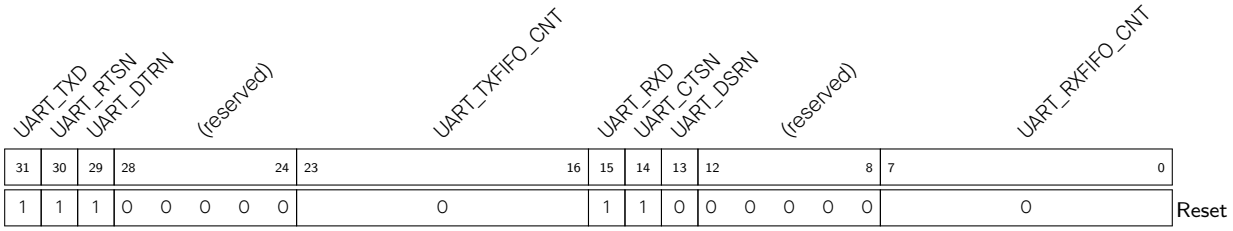
1: Enable

(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 31.21. 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)

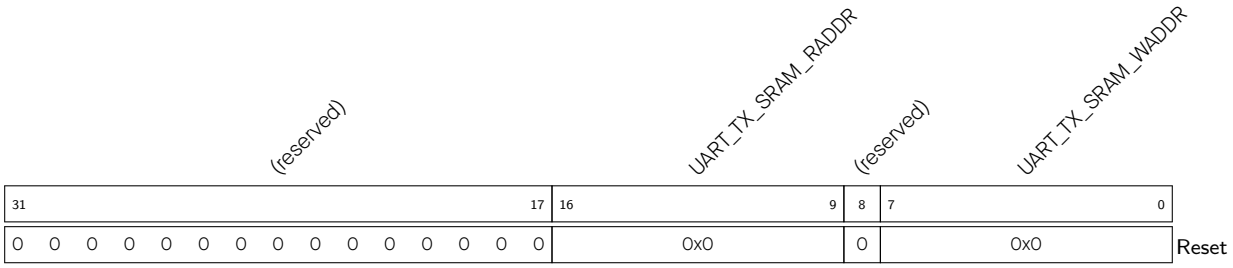
UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

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)

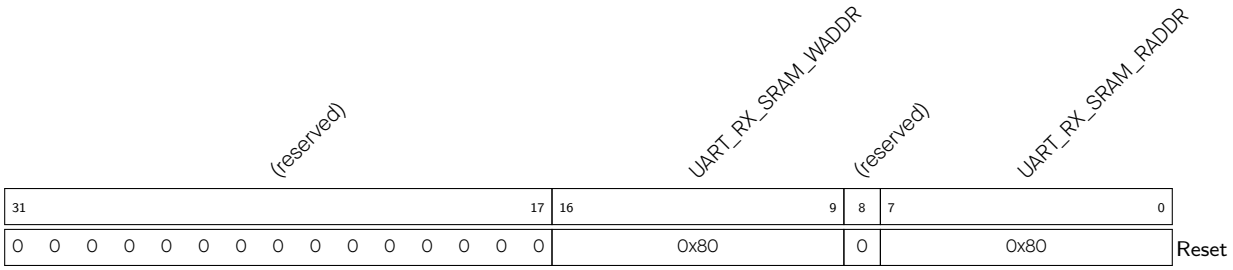
Register 31.22. UART_MEM_TX_STATUS_REG (0x0068)



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)

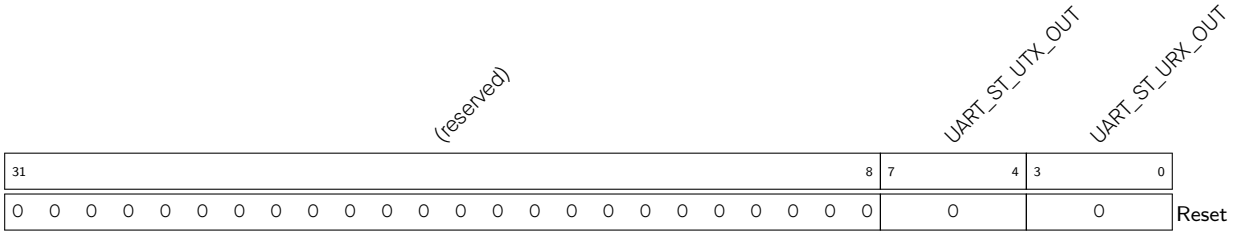
Register 31.23. UART_MEM_RX_STATUS_REG (0x006C)



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)

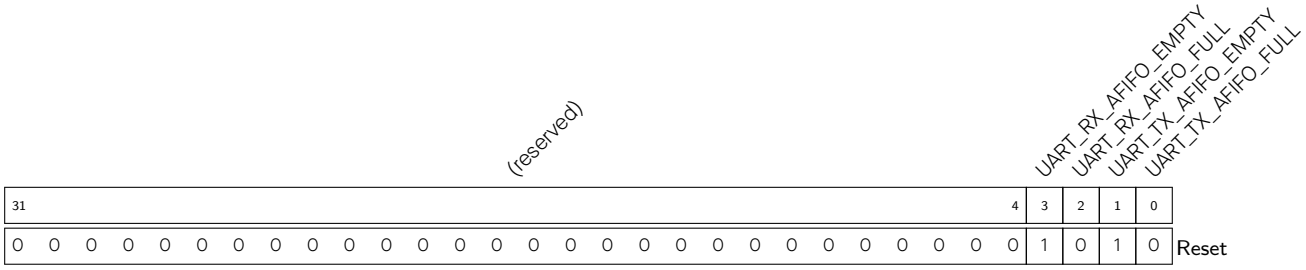
Register 31.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT Represents the status of the receiver. (RO)

UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 31.25. 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)

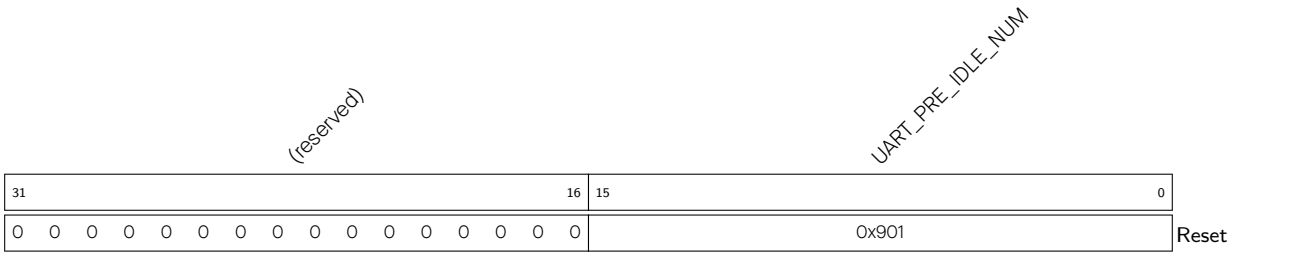
UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

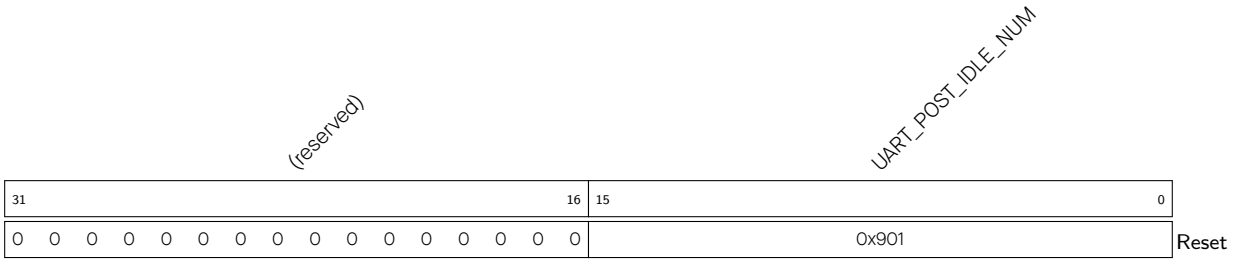
Register 31.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD.

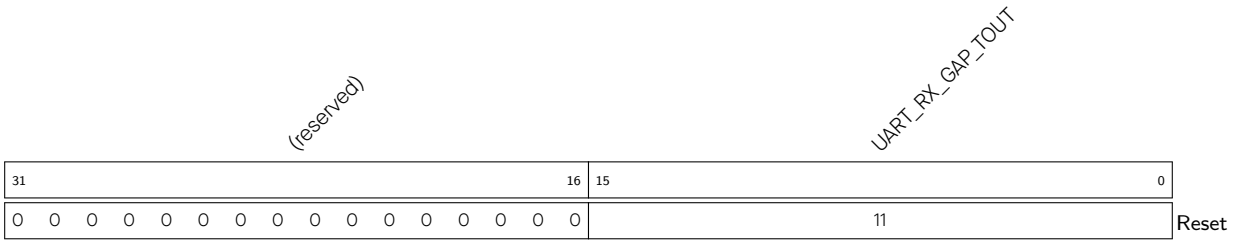
Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 31.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



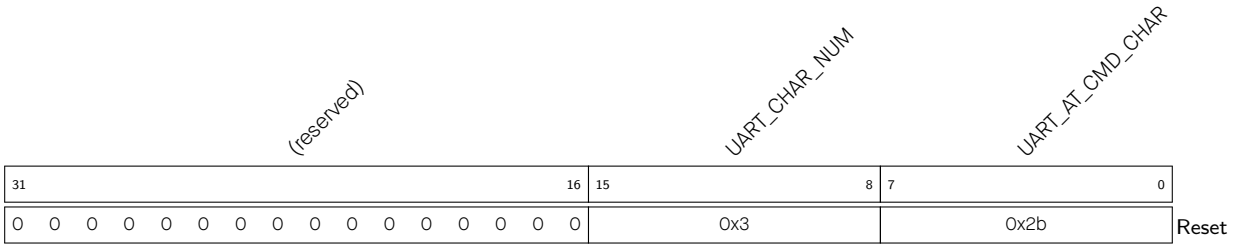
UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 31.28. 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)

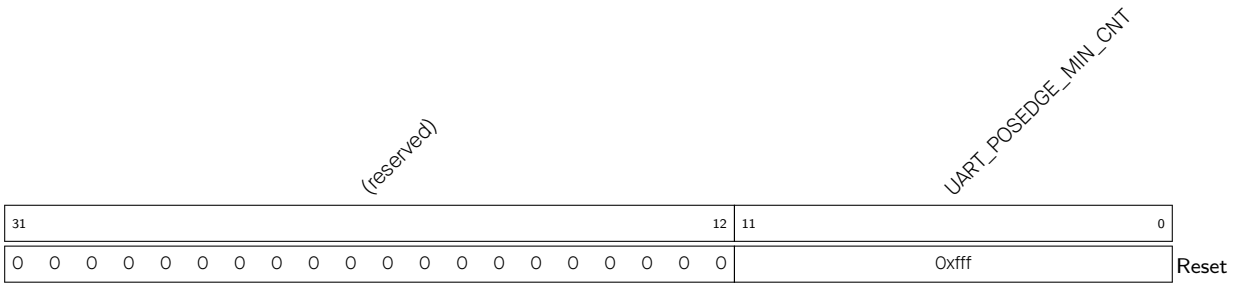
Register 31.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

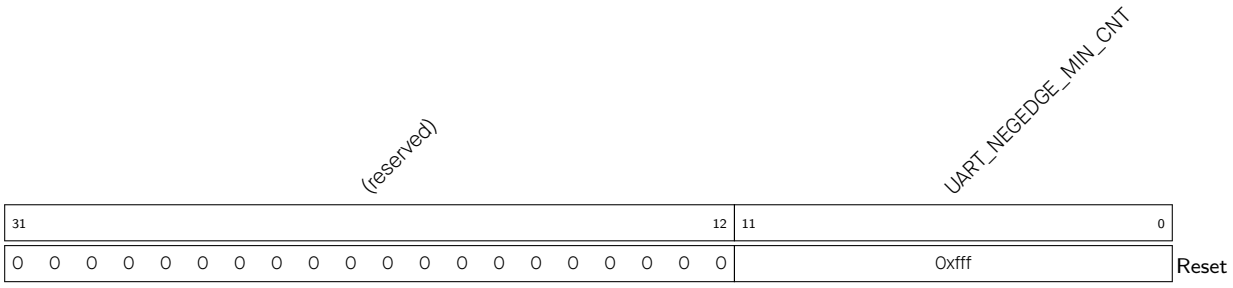
UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 31.30. UART_POSPULSE_REG (0x0074)



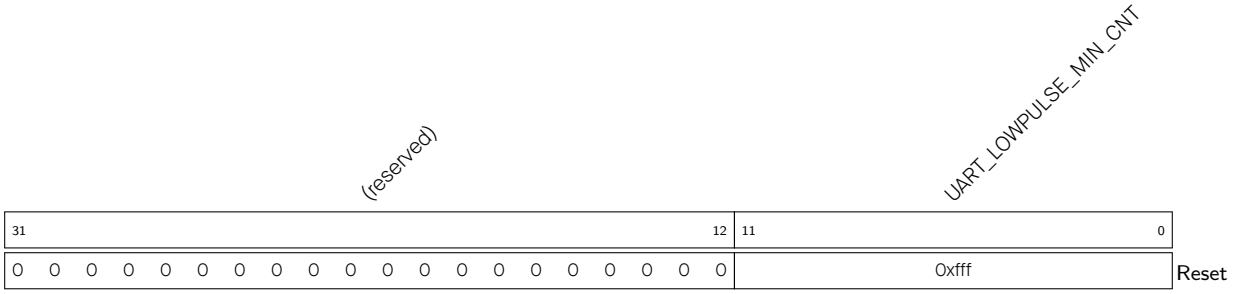
UART_POSEDGE_MIN_CNT Represents the minimal input clock counter value between two positive edges. It is used for baud rate detection. (RO)

Register 31.31. 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)

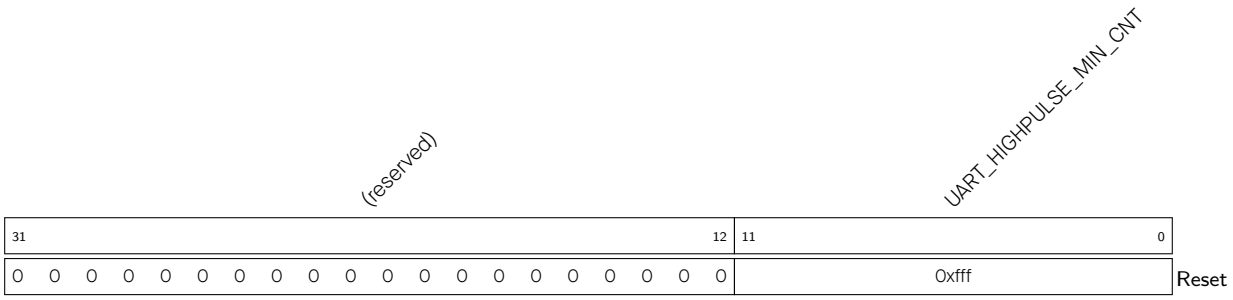
Register 31.32. UART_LOWPULSE_REG (0x007C)



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)

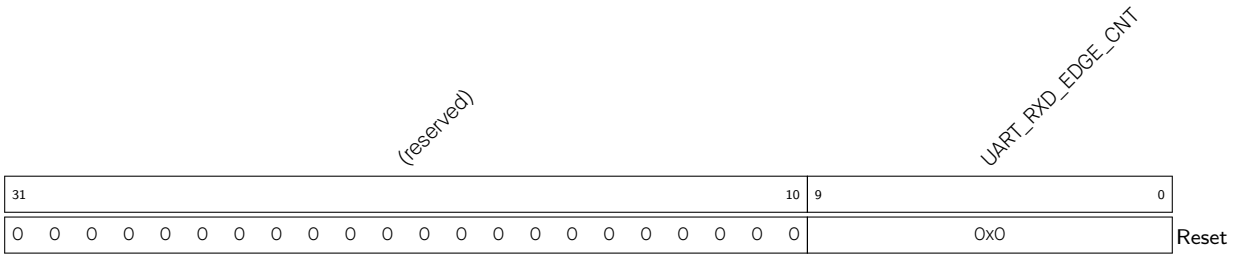
Register 31.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT Represents the maximum duration time for a high-level pulse. It is used for baud rate detection.

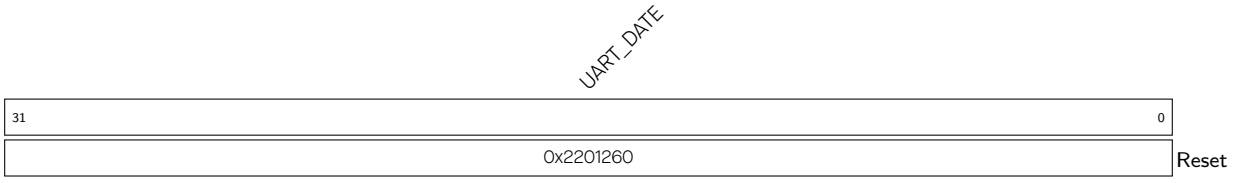
Measurement unit: APB_CLK clock cycle. (RO)

Register 31.34. UART_RXD_CNT_REG (0x0084)



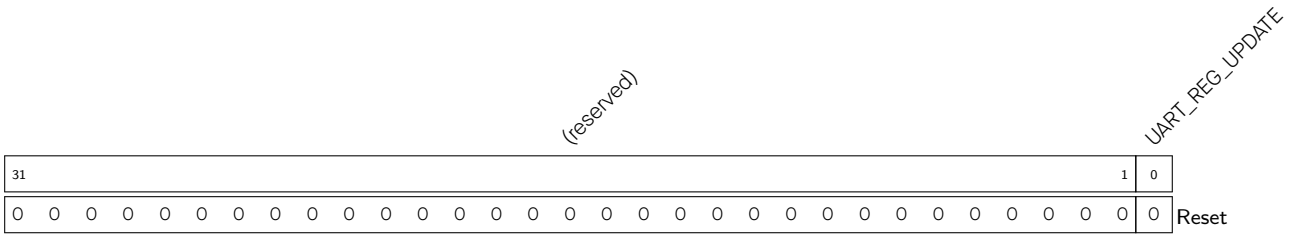
UART_RXD_EDGE_CNT Represents the number of RXD edge changes. It is used for baud rate detection. (RO)

Register 31.35. UART_DATE_REG (0x008C)



UART_DATE Version control register. (R/W)

Register 31.36. UART_REG_UPDATE_REG (0x0098)



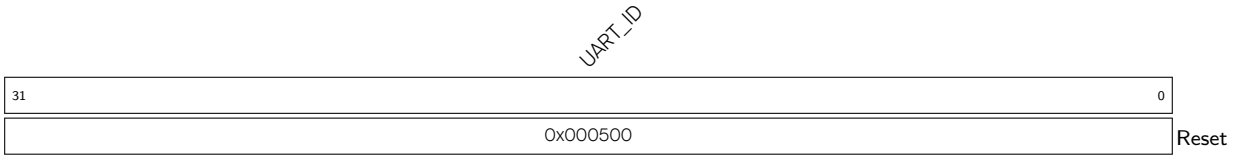
UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 31.37. UART_ID_REG (0x009C)



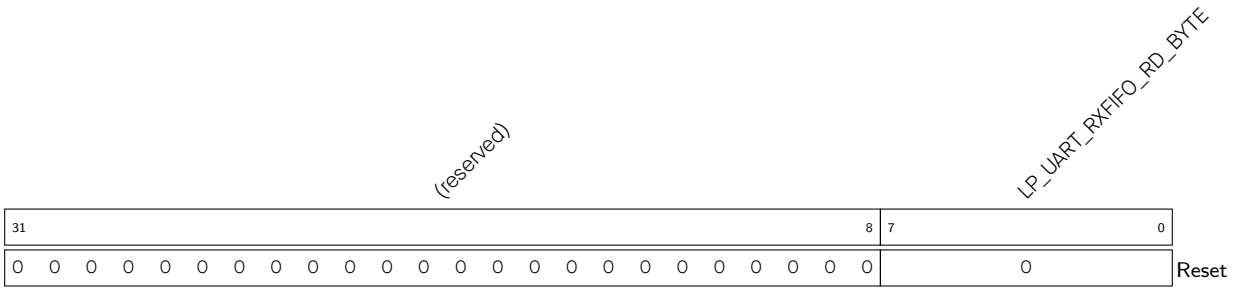
UART_ID Configures the UART ID. (R/W)

31.7.2 LP UART Registers

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

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

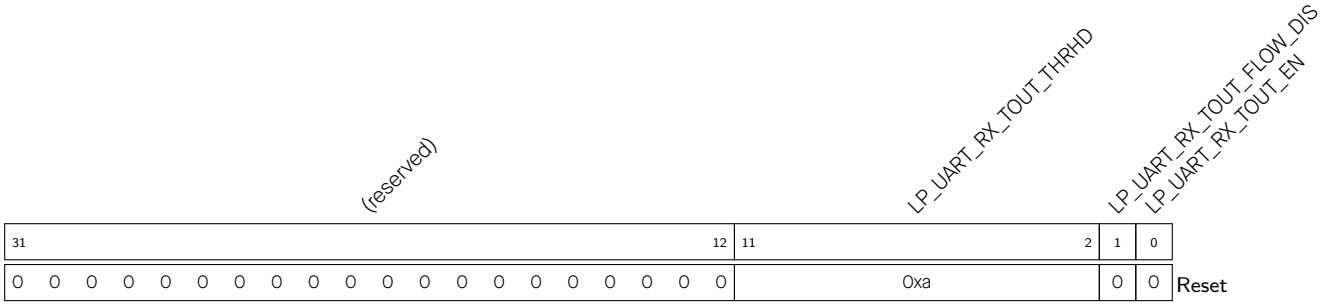
Register 31.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE Represents the data LP UART *n* read from FIFO.

Measurement unit: byte. (RO)

Register 31.39. 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

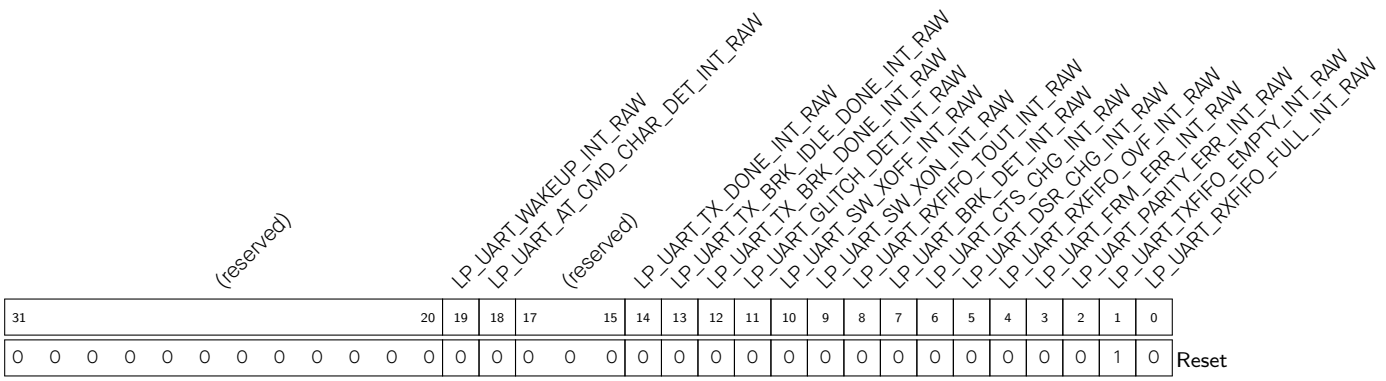
1: Disable

(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 31.40. 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 31.40. LP_UART_INT_RAW_REG (0x0004)

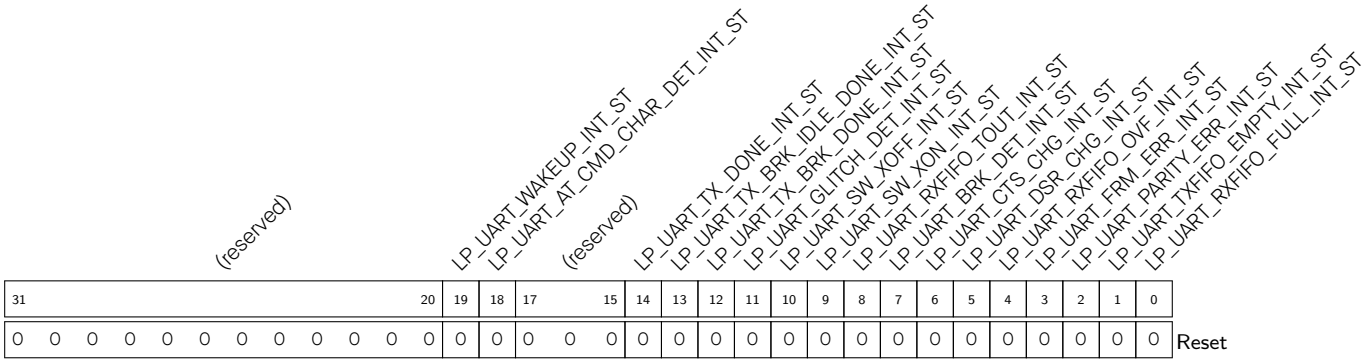
Continued from the previous page...

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 31.41. 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 31.42. LP_UART_INT_ENA_REG (0x000C)

Bits	Field	Reset
31–20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	LP_UART_WAKEUP_INT_ENA	0
18	LP_UART_AT_CMD_CHAR_DET_INT_ENA	0
17–15	(reserved)	0 0 0
14	LP_UART_TX_DONE_INT_ENA	0
13	LP_UART_TX_BRK_IDLE_DONE_INT_ENA	0
12	LP_UART_TX_BRK_DONE_INT_ENA	0
11	LP_UART_GLITCH_DET_INT_ENA	0
10	LP_UART_SW_XOFF_INT_ENA	0
9	LP_UART_SW_XON_INT_ENA	0
8	LP_UART_RXFIFO_TOUT_INT_ENA	0
7	LP_UART_BRK_DET_INT_ENA	0
6	LP_UART_CTS_CHG_INT_ENA	0
5	LP_UART_DSR_CHG_INT_ENA	0
4	LP_UART_RXFIFO_OVF_INT_ENA	0
3	LP_UART_FRM_ERR_INT_ENA	0
2	LP_UART_PARITY_ERR_INT_ENA	0
1	LP_UART_TXFIFO_EMPTY_INT_ENA	0
0	LP_UART_RXFIFO_FULL_INT_ENA	0

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)

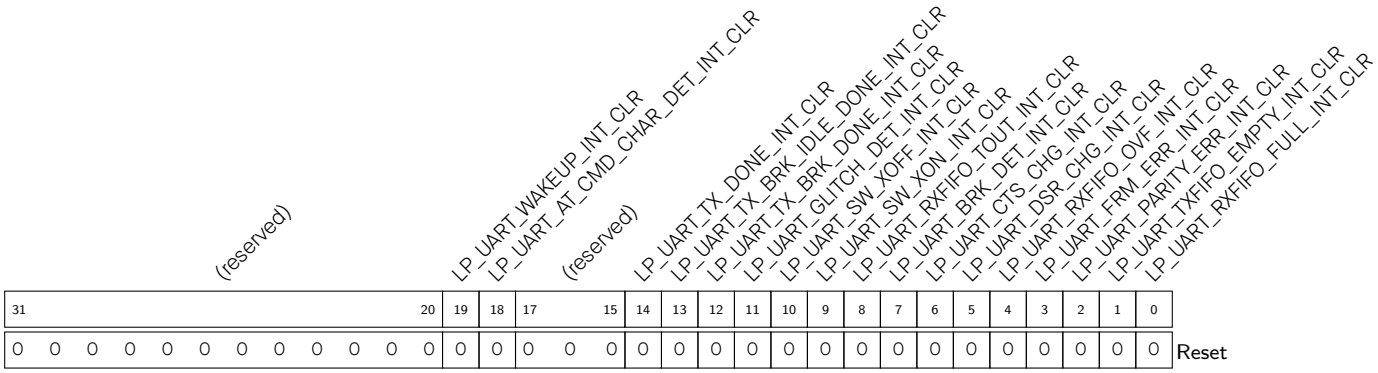
LP_UART_TX_BRK_IDLE_DONE_INT_ENA Write 1 to enable LP_UART_TX_BRK_IDLE_DONE_INT. (R/W)

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 31.43. 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)

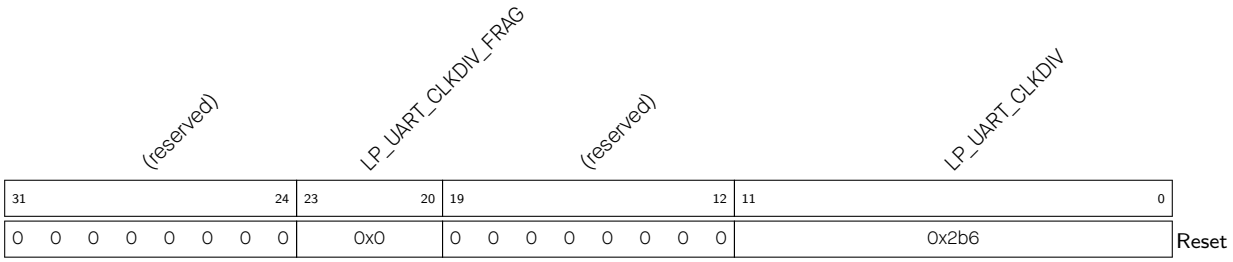
LP_UART_TX_BRK_IDLE_DONE_INT_CLR Write 1 to clear LP_UART_TX_BRK_IDLE_DONE_INT. (WT)

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)

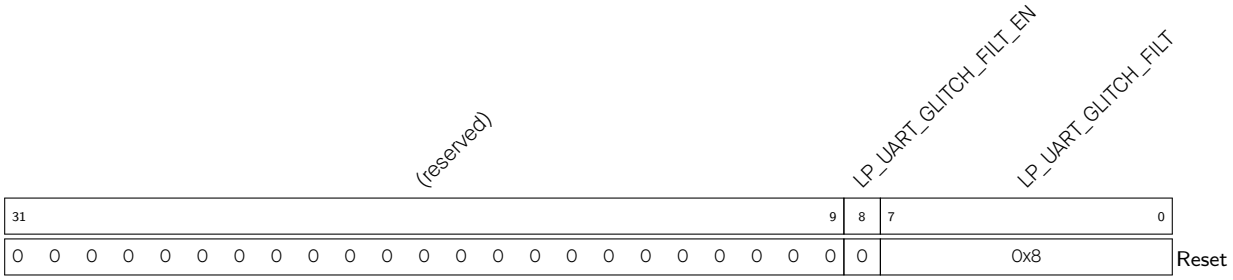
Register 31.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



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 31.45. 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 31.46. 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 31.46. 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)

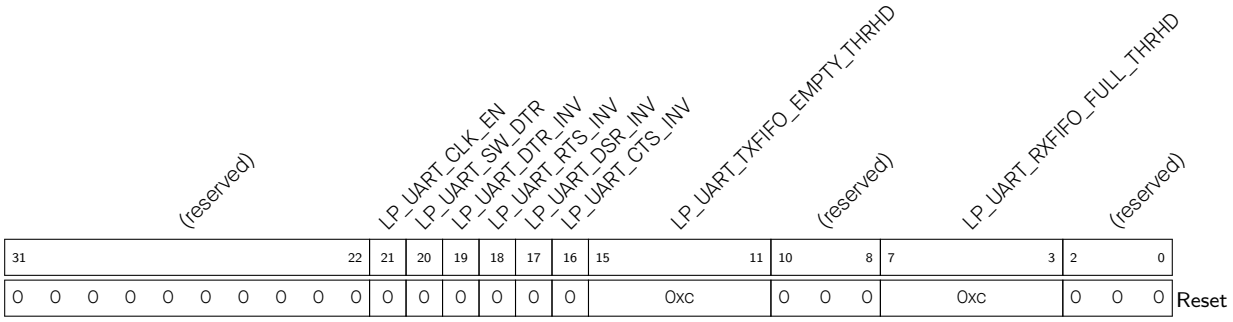
LP_UART_MEM_CLK_EN Configures whether or not to enable clock gating for LP UART memory.
0: Disable
1: Enable
(R/W)

LP_UART_SW_RTS Configures the RTS signal used in software flow control.
0: The LP UART transmitter is allowed to send data.
1: The LP UART transmitted is not 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 31.47. 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)

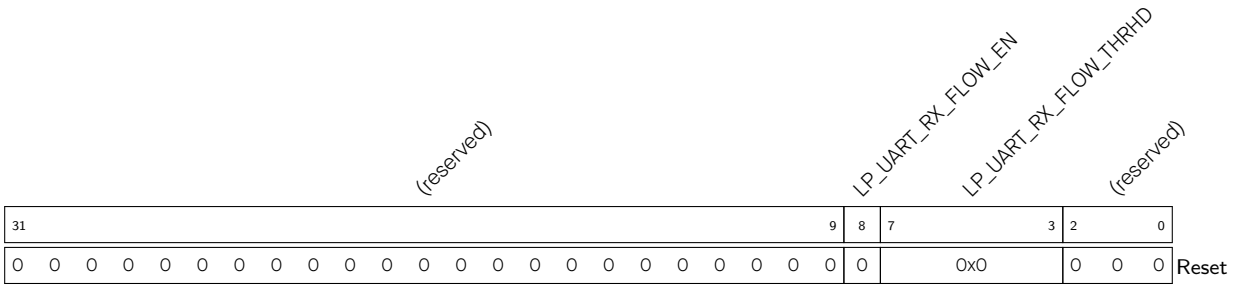
LP_UART_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

Register 31.48. 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)

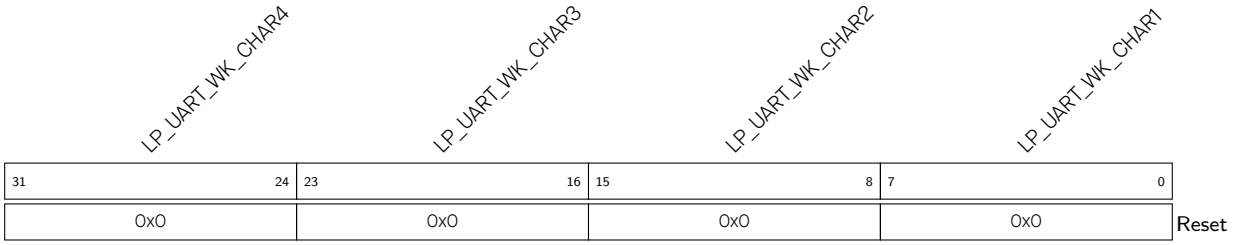
LP_UART_RX_FLOW_EN Configures whether or not to enable the LP UART receiver.

0: Disable

1: Enable

(R/W)

Register 31.49. LP_UART_SLEEP_CONF0_REG (0x0030)



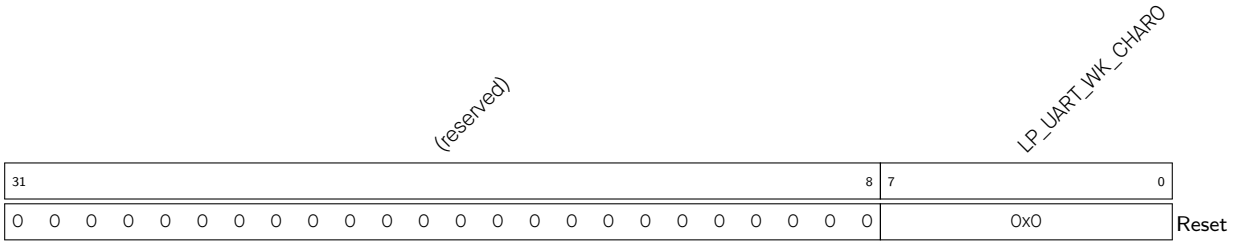
LP_UART_WK_CHAR1 Configures wakeup character 1. (R/W)

LP_UART_WK_CHAR2 Configures wakeup character 2. (R/W)

LP_UART_WK_CHAR3 Configures wakeup character 3. (R/W)

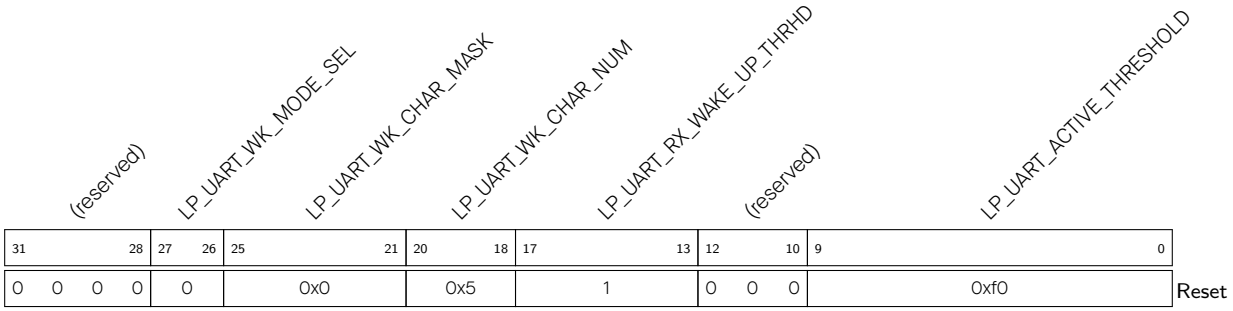
LP_UART_WK_CHAR4 Configures wakeup character 4. (R/W)

Register 31.50. LP_UART_SLEEP_CONF1_REG (0x0034)



LP_UART_WK_CHAR0 Configures wakeup character 0. (R/W)

Register 31.51. 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

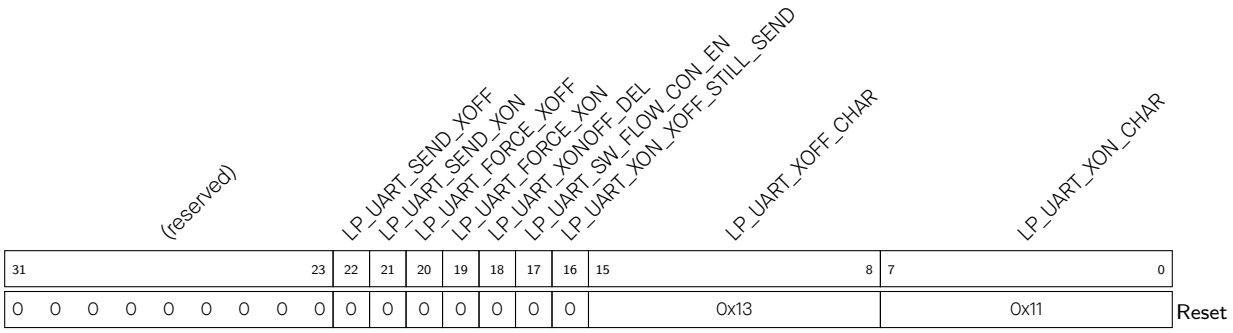
1: Mode 1

2: Mode 2

3: Mode 3

(R/W)

Register 31.52. 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)

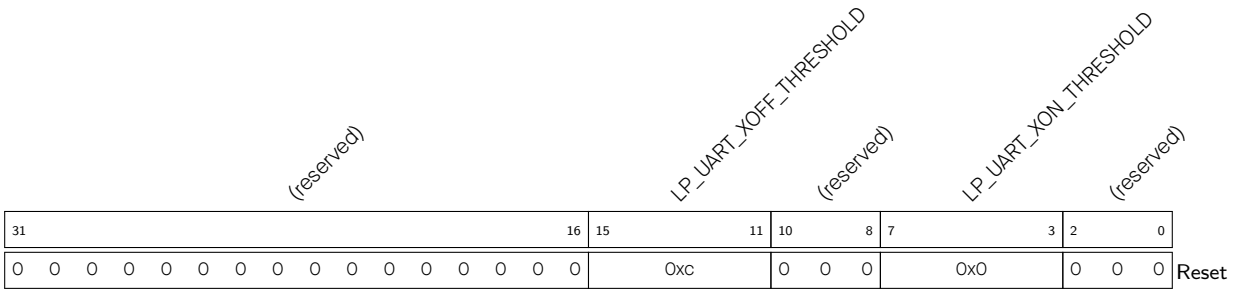
LP_UART_SEND_XOFF Configures whether or not to send XOFF characters.

0: Not send

1: Send

(R/W/SS/SC)

Register 31.53. LP_UART_SWFC_CONF1_REG (0x0040)



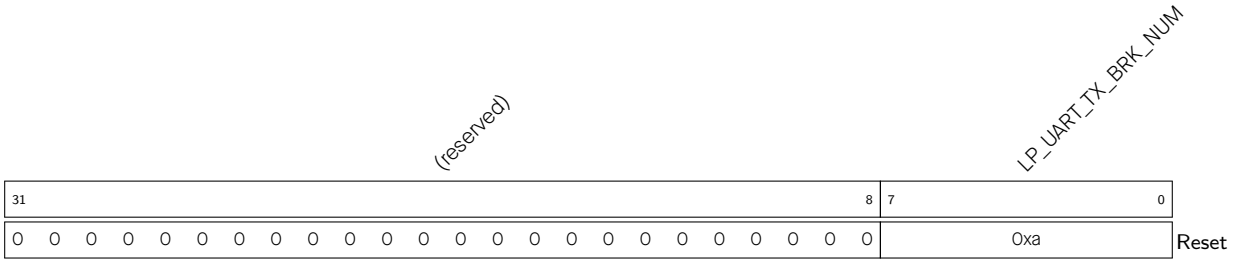
LP_UART_XON_THRESHOLD Configures the threshold for data in RX FIFO to send XON characters in software flow control.

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)

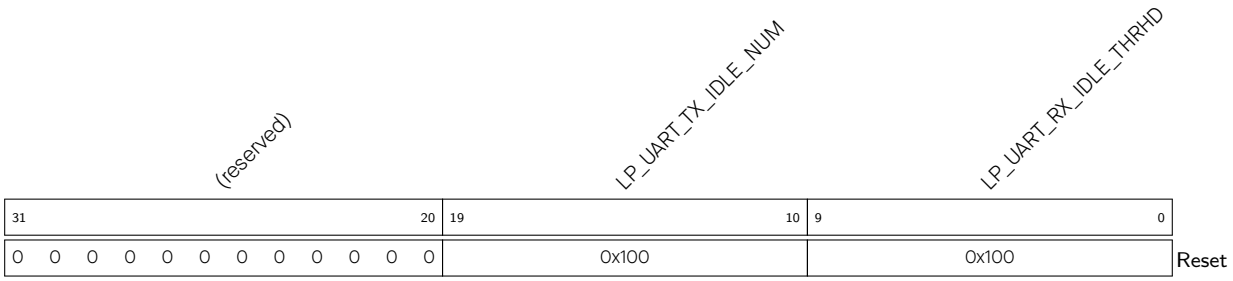
Register 31.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)



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)

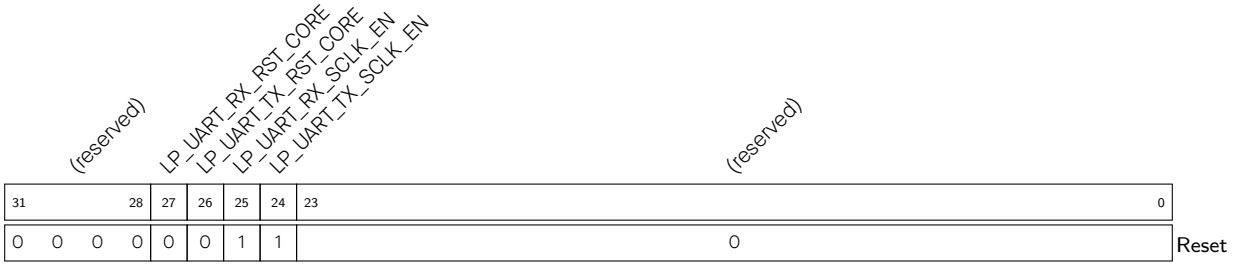
Register 31.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)



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 31.56. LP_UART_CLK_CONF_REG (0x0088)



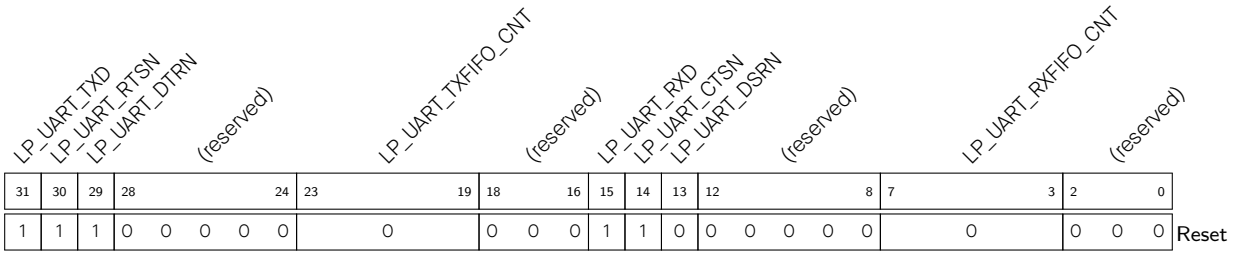
LP_UART_TX_SCLK_EN Configures whether or not to enable LP UART TX clock.
0: Disable
1: Enable
(R/W)

LP_UART_RX_SCLK_EN Configures whether or not to enable LP UART RX clock.
0: Disable
1: Enable
(R/W)

LP_UART_TX_RST_CORE Write 1 and then write 0 to reset LP UART TX. (R/W)

LP_UART_RX_RST_CORE Write 1 and then write 0 to reset LP UART RX. (R/W)

Register 31.57. LP_UART_STATUS_REG (0x001C)



LP_UART_RXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DSRN Represents the level of the internal LP UART DSR signal. (RO)

LP_UART_CTSN Represents the level of the internal LP UART CTS signal. (RO)

LP_UART_RXD Represents the level of the internal LP UART RXD signal. (RO)

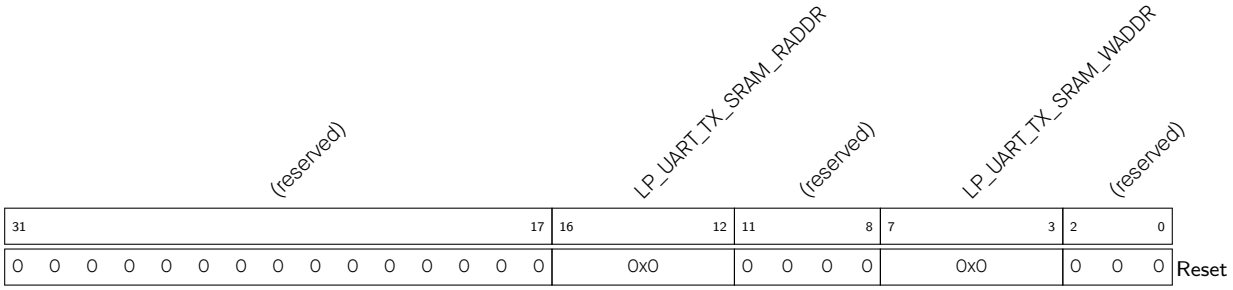
LP_UART_TXFIFO_CNT Represents the number of valid data bytes in RX FIFO. (RO)

LP_UART_DTRN Represents the level of the internal LP UART DTR signal. (RO)

LP_UART_RTSN Represents the level of the internal LP UART RTS signal. (RO)

LP_UART_TXD Represents the level of the internal LP UART TXD signal. (RO)

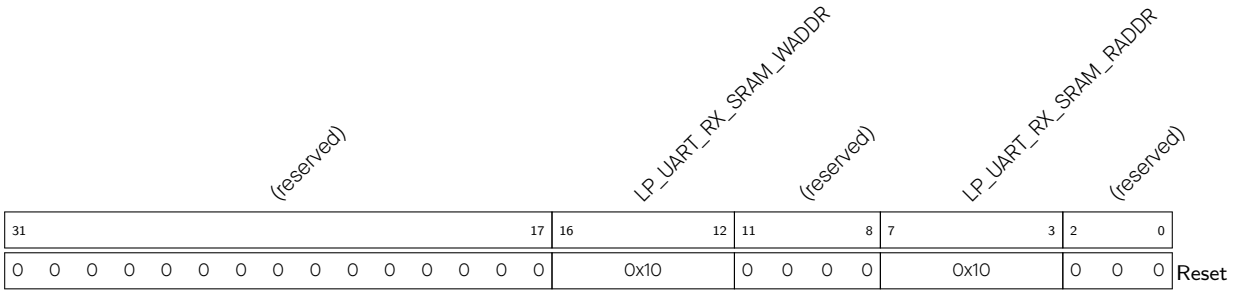
Register 31.58. LP_UART_MEM_TX_STATUS_REG (0x0068)



LP_UART_TX_SRAM_WADDR Represents the offset address to write TX FIFO. (RO)

LP_UART_TX_SRAM_RADDR Represents the offset address to read TX FIFO. (RO)

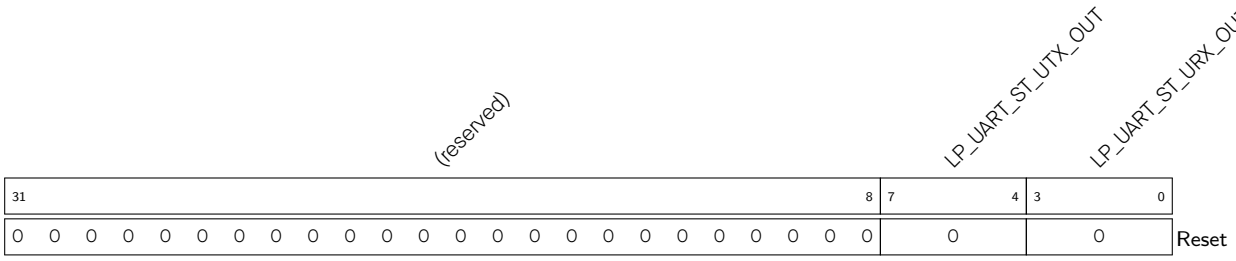
Register 31.59. LP_UART_MEM_RX_STATUS_REG (0x006C)



LP_UART_RX_SRAM_RADDR Represents the offset address to read RX FIFO. (RO)

LP_UART_RX_SRAM_WADDR Represents the offset address to write RX FIFO. (RO)

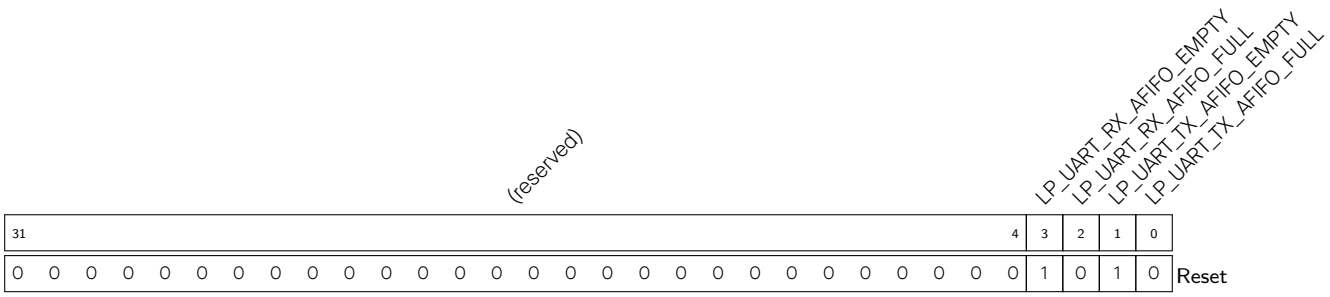
Register 31.60. LP_UART_FSM_STATUS_REG (0x0070)



LP_UART_ST_URX_OUT Represents the status of the receiver. (RO)

LP_UART_ST_UTX_OUT Represents the status of the transmitter. (RO)

Register 31.61. LP_UART_AFIFO_STATUS_REG (0x0090)



LP_UART_TX_AFIFO_FULL Represents whether or not the APB TX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

LP_UART_TX_AFIFO_EMPTY Represents whether or not the APB TX asynchronous FIFO is empty.

0: Not empty

1: Empty

(RO)

LP_UART_RX_AFIFO_FULL Represents whether or not the APB RX asynchronous FIFO is full.

0: Not full

1: Full

(RO)

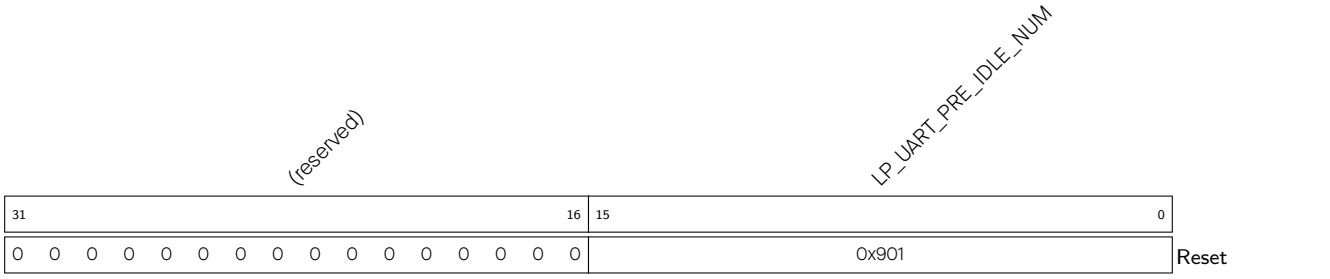
LP_UART_RX_AFIFO_EMPTY Represents whether or not the APB RX asynchronous FIFO is empty.

0: Not empty

1: Empty

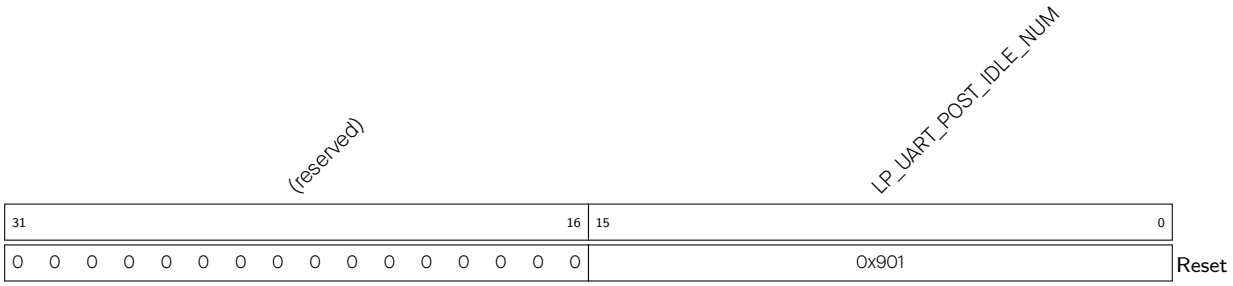
(RO)

Register 31.62. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



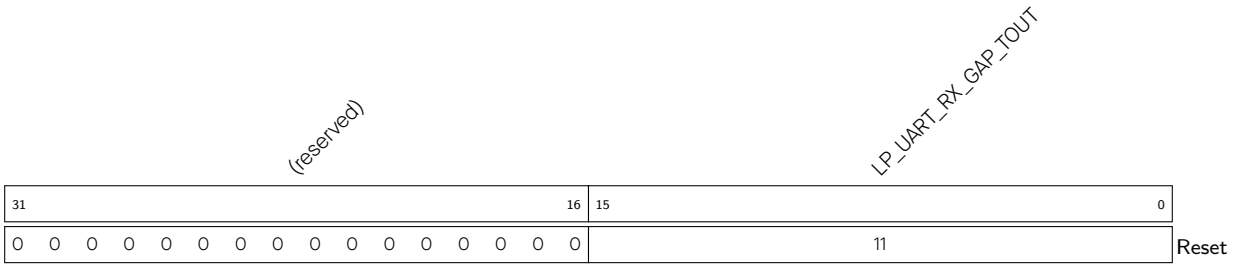
LP_UART_PRE_IDLE_NUM Configures the idle time before the receiver receives the first AT_CMD. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 31.63. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



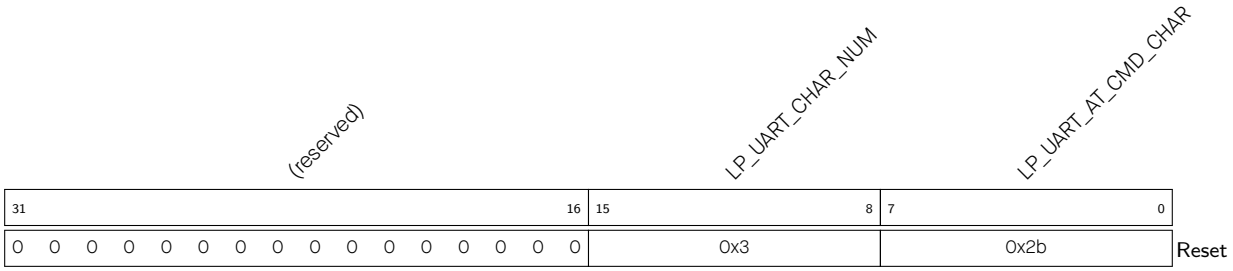
LP_UART_POST_IDLE_NUM Configures the interval between the last AT_CMD and subsequent data. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

Register 31.64. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



LP_UART_RX_GAP_TOUT Configures the interval between two AT_CMD characters. Measurement unit: bit time (the time to transmit 1 bit). (R/W)

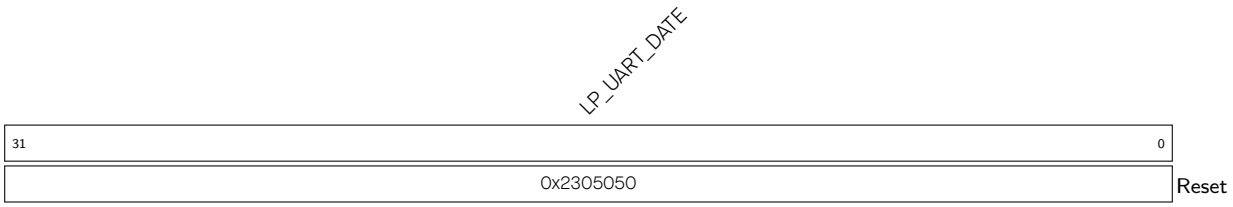
Register 31.65. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)



LP_UART_AT_CMD_CHAR Configures the AT_CMD character. (R/W)

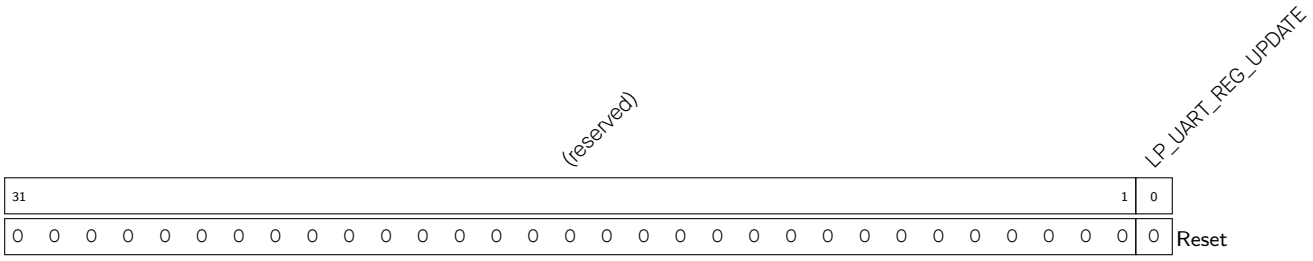
LP_UART_CHAR_NUM Configures the number of continuous AT_CMD characters a receiver can receive. (R/W)

Register 31.66. LP_UART_DATE_REG (0x008C)



LP_UART_DATE Version control register. (R/W)

Register 31.67. LP_UART_REG_UPDATE_REG (0x0098)



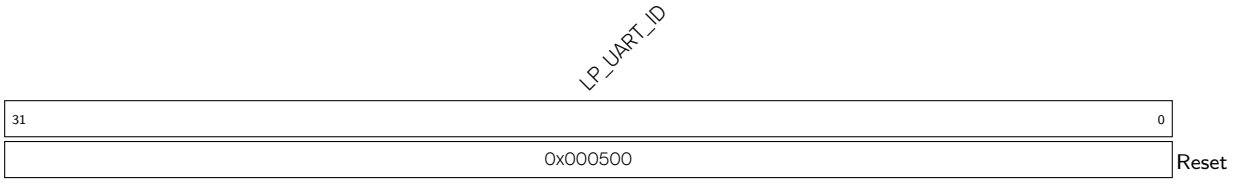
LP_UART_REG_UPDATE Configures whether or not to synchronize registers.

0: Not synchronize

1: Synchronize

(R/W/SC)

Register 31.68. LP_UART_ID_REG (0x009C)



LP_UART_ID Configures the LP UART ID. (R/W)

31.7.3 UHCI Registers

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

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

Register 31.69. UHCI_CONF0_REG (0x0000)



UHCI_TX_RST Write 1 and then write 0 to reset the decoder state machine. (R/W)

UHCI_RX_RST Write 1 and then write 0 to reset the encoder state machine. (R/W)

UHCI_UART0_CE Select one UART from UART 0 ~ 4 to connect with UHCI.

0: Select UART0
1: Select UART1
2: Select UART2
3: Select UART3
4: Select UART4
5 ~ 7: No effect, as no UART will connect with UHCI
(R/W)

UHCI_UART1_CE Configures whether or not to connect UHCI with UART1.

0: Not connect
1: Connect
(R/W)

UHCI_SEPER_EN Configures whether or not to separate the data frame with a special character.

0: Not separate
1: Separate
(R/W)

UHCI_HEAD_EN Configures whether or not to encode the data packet with a formatting header.

0: Not use formatting header
1: Use formatting header
(R/W)

Continued on the next page...

Register 31.69. UHCI_CONF0_REG (0x0000)

Continued from the previous page...

UHCI_CRC_REC_EN Configures whether or not to enable the reception of the 16-bit CRC.

0: Disable

1: Enable

(R/W)

UHCI_UART_IDLE_EOF_EN Configures whether or not to stop receiving data when UART is idle.

0: Not stop

1: Stop

(R/W)

UHCI_LEN_EOF_EN Configures when the UHCI decoder stops receiving data.

0: Stops after receiving 0xC0

1: Stops when the number of received data bytes reach the specified value. When UHCI_HEAD_EN is 1, the specified value is the data length indicated by the UHCI packet header; when UHCI_HEAD_EN is 0, the specified value is the configured value.

(R/W)

UHCI_ENCODE_CRC_EN Configures whether or not to enable data integrity check by appending a 16 bit CCITT-CRC to the end of the data.

0: Disable

1: Enable

(R/W)

UHCI_CLK_EN Configures clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

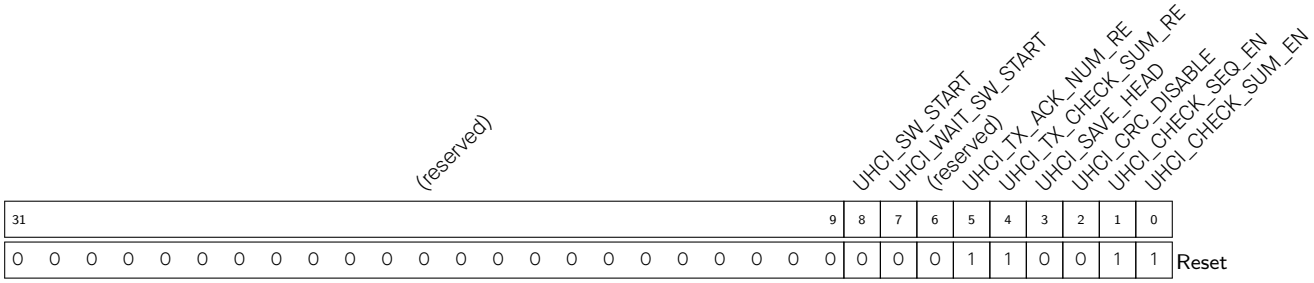
UHCI_UART_RX_BRK_EOF_EN Configures whether or not to stop UHCI from receiving data after UART has received a NULL frame.

0: Not stop

1: Stop

(R/W)

Register 31.70. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN Configures whether or not to enable header checksum validation when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CHECK_SEQ_EN Configures whether or not to enable the sequence number check when UHCI receives a data packet.

0: Disable

1: Enable

(R/W)

UHCI_CRC_DISABLE Configures whether or not to enable CRC calculation.

0: Disable

1: Enable

Valid only when the Data Integrity Check Present bit in UHCI packet is 1.

(R/W)

UHCI_SAVE_HEAD Configures whether or not to save the packet header when UHCI receives a data packet.

0: Not save

1: Save

(R/W)

UHCI_TX_CHECK_SUM_RE Configures whether or not to encode the data packet with a checksum.

0: Not use checksum

1: Use checksum

(R/W)

UHCI_TX_ACK_NUM_RE Configures whether or not to encode the data packet with an acknowledgment when a reliable packet is to be transmitted.

0: Not use acknowledgement

1: Use acknowledgement

(R/W)

Continued on the next page...

Register 31.70. UHCI_CONF1_REG (0x0014)

Continued from the previous page...

UHCI_WAIT_SW_START Configures whether or not to put the UHCI encoder state machine to ST_SW_WAIT state.

0: No

1: Yes

(R/W)

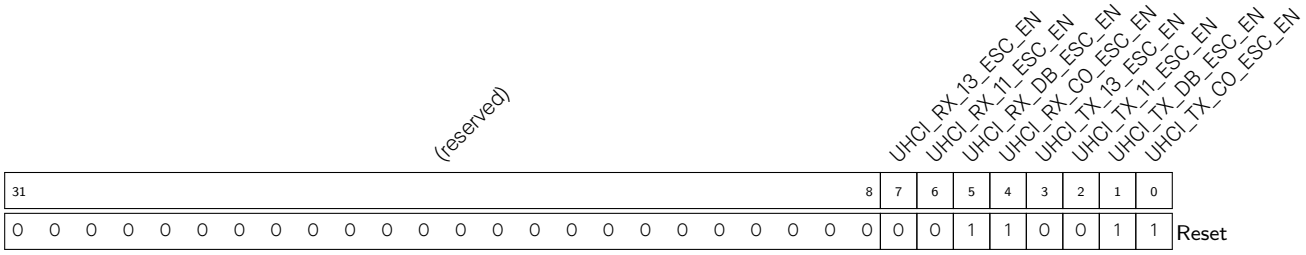
UHCI_SW_START Configures whether or not to send data packets when the encoder state machine is in ST_SW_WAIT state.

0: Not send

1: Send

(R/W/SC)

Register 31.71. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN Configures whether or not to decode character 0xC0 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_DB_ESC_EN Configures whether or not to decode character 0xDB when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_11_ESC_EN Configures whether or not to decode flow control character 0x11 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_TX_13_ESC_EN Configures whether or not to decode flow control character 0x13 when DMA receives data.

0: Not decode

1: Decode

(R/W)

UHCI_RX_C0_ESC_EN Configures whether or not to replace 0xC0 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

UHCI_RX_DB_ESC_EN Configures whether or not to replace 0xDB by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

Continued on the next page...

Register 31.71. UHCI_ESCAPE_CONF_REG (0x0020)

Continued from the previous page...

UHCI_RX_11_ESC_EN Configures whether or not to replace flow control character 0x11 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

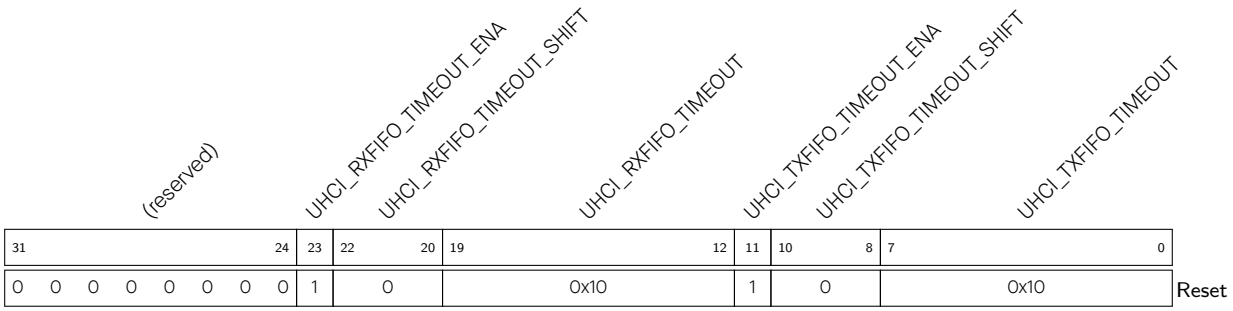
UHCI_RX_13_ESC_EN Configures whether or not to replace flow control character 0x13 by special characters when DMA sends data.

0: Not replace

1: Replace

(R/W)

Register 31.72. UHCI_HUNG_CONF_REG (0x0024)



UHCI_TXFIFO_TIMEOUT Configures the timeout value for DMA data reception.
Measurement unit: ms. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for TX FIFO. (R/W)

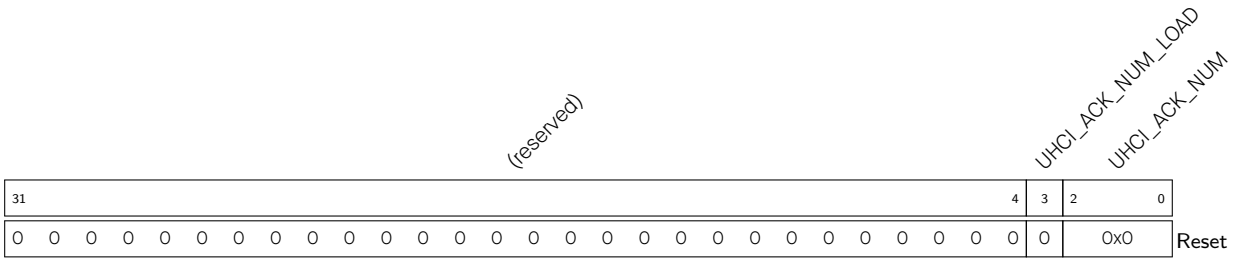
UHCI_TXFIFO_TIMEOUT_ENA Configures whether or not to enable the data reception timeout for TX FIFO.
0: Disable
1: Enable
(R/W)

UHCI_RXFIFO_TIMEOUT Configures the timeout value for DMA to read data from RAM.
Measurement unit: ms. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT Configures the upper limit of the timeout counter for RX FIFO. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA Configures whether or not to enable the DMA data transmission timeout.
0: Disable
1: Enable
(R/W)

Register 31.73. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM Configures the number of acknowledgements used in software flow control. (R/W)

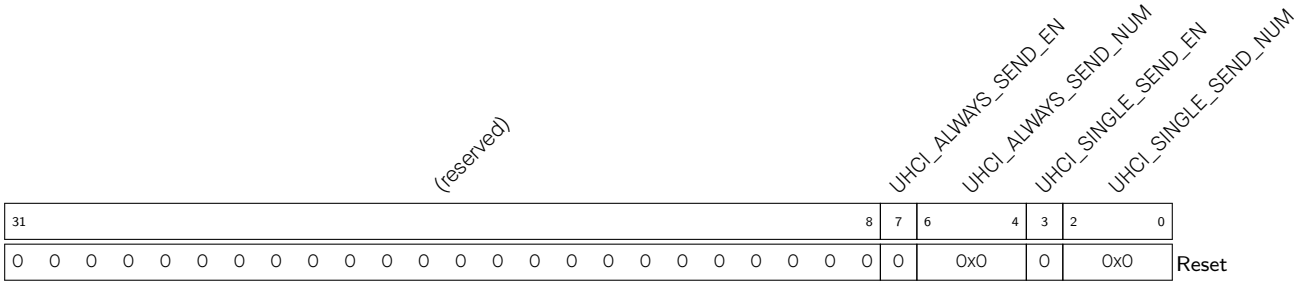
UHCI_ACK_NUM_LOAD Configures whether or not load acknowledgements.

0: Not load

1: Load

(WT)

Register 31.74. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM Configures the source of data to be transmitted in single_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

UHCI_SINGLE_SEND_EN Configures whether or not to enable single_send mode.

0: Disable

1: Enable

(R/W/SC)

UHCI_ALWAYS_SEND_NUM Configures the source of data to be transmitted in always_send mode.

0: Q0 register

1: Q1 register

2: Q2 register

3: Q3 register

4: Q4 register

5: Q5 register

6: Q6 register

7: Invalid. No effect

(R/W)

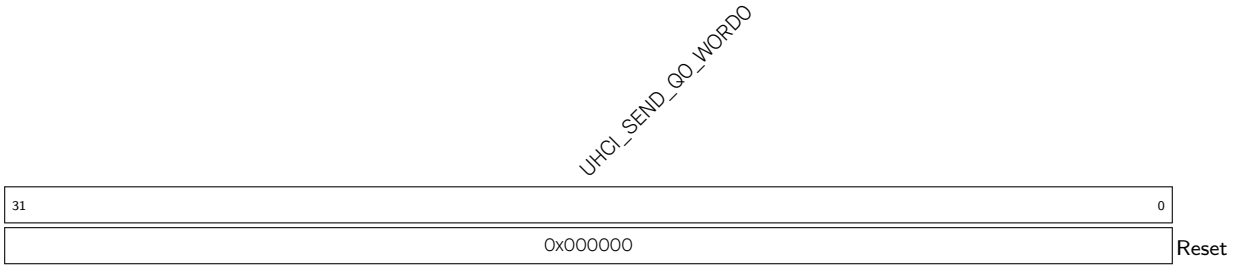
UHCI_ALWAYS_SEND_EN Configures whether or not to enable always_send mode.

0: Disable

1: Enable

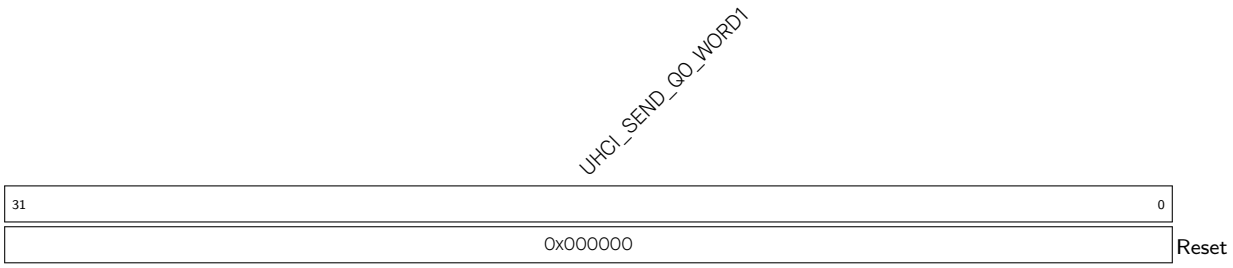
(R/W)

Register 31.75. UHCI_REG_Q0_WORD0_REG (0x0034)



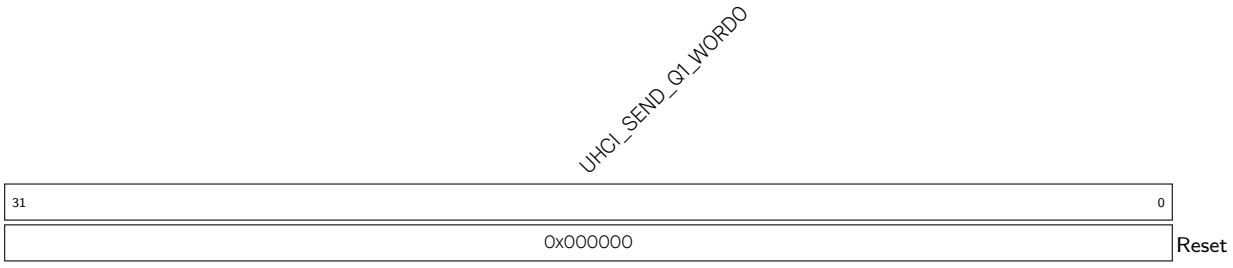
UHCI_SEND_Q0_WORD0 Data to be transmitted in Q0 register. (R/W)

Register 31.76. UHCI_REG_Q0_WORD1_REG (0x0038)



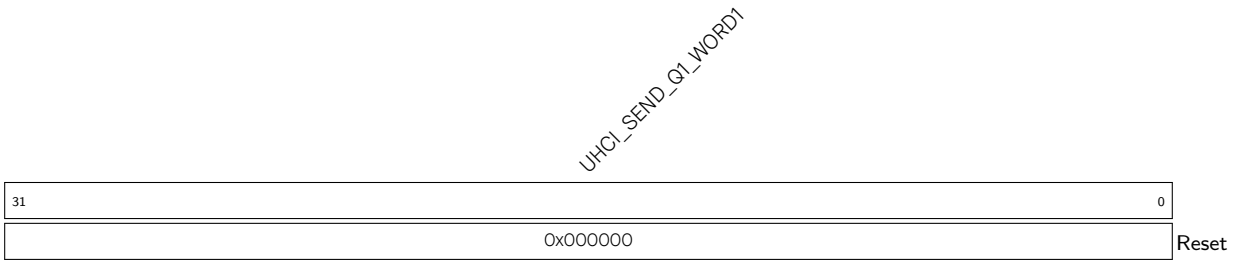
UHCI_SEND_Q0_WORD1 Data to be transmitted in Q0 register. (R/W)

Register 31.77. UHCI_REG_Q1_WORD0_REG (0x003C)



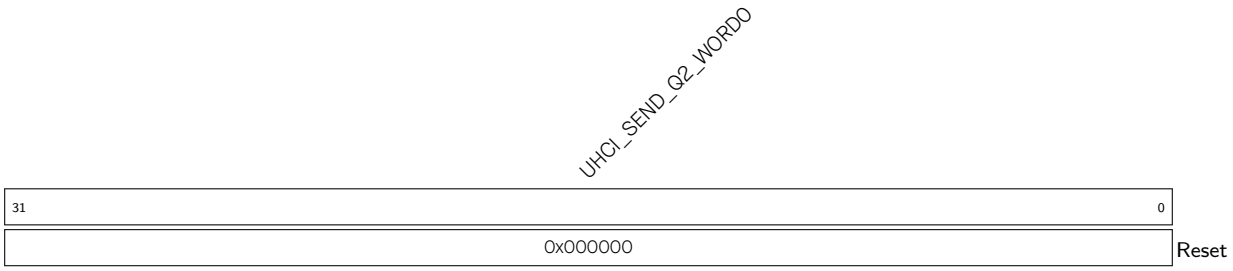
UHCI_SEND_Q1_WORD0 Data to be transmitted in Q1 register. (R/W)

Register 31.78. UHCI_REG_Q1_WORD1_REG (0x0040)



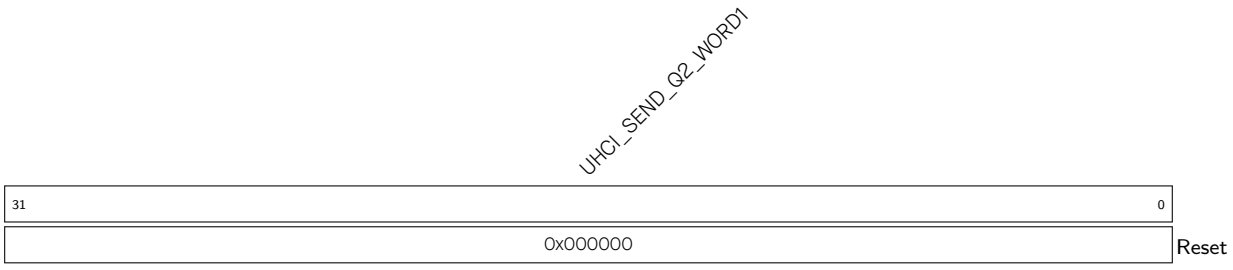
UHCI_SEND_Q1_WORD1 Data to be transmitted in Q1 register. (R/W)

Register 31.79. UHCI_REG_Q2_WORD0_REG (0x0044)



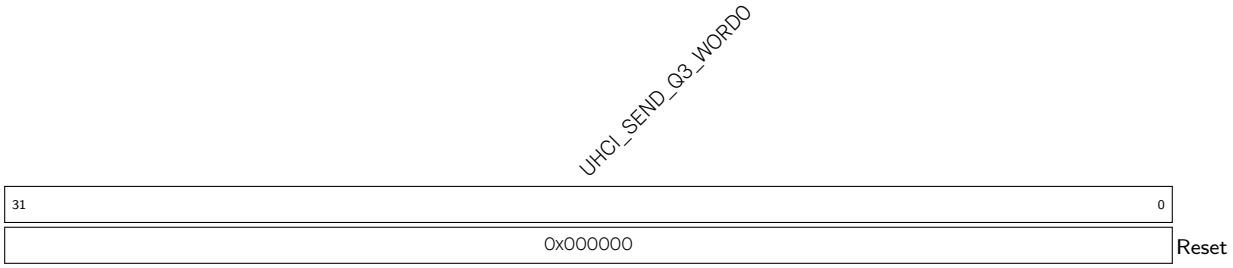
UHCI_SEND_Q2_WORD0 Data to be transmitted in Q2 register. (R/W)

Register 31.80. UHCI_REG_Q2_WORD1_REG (0x0048)



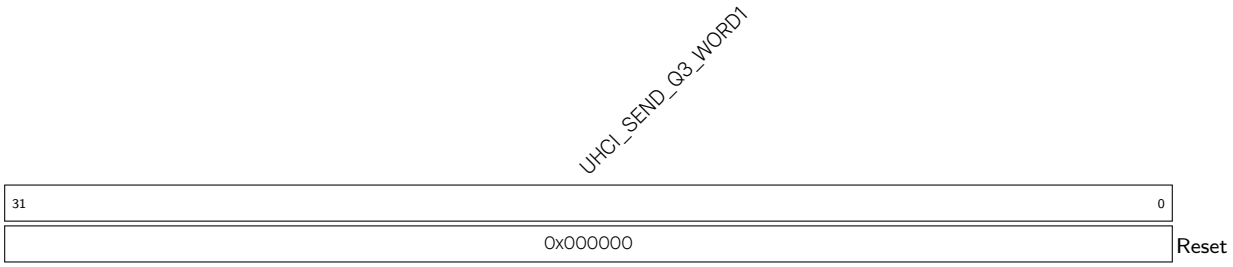
UHCI_SEND_Q2_WORD1 Data to be transmitted in Q2 register. (R/W)

Register 31.81. UHCI_REG_Q3_WORD0_REG (0x004C)



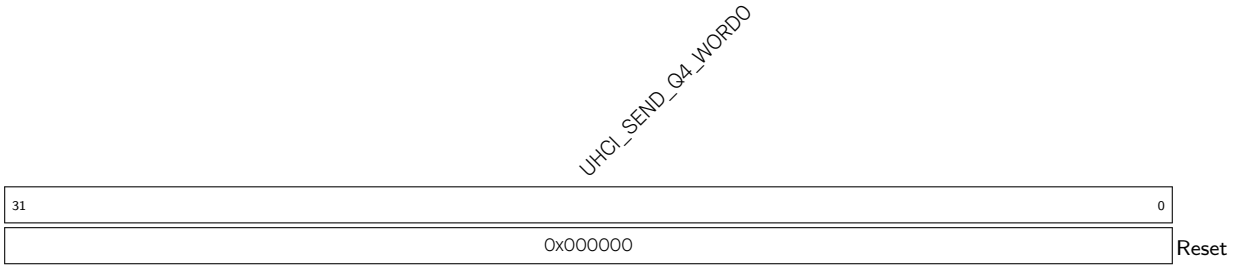
UHCI_SEND_Q3_WORD0 Data to be transmitted in Q3 register. (R/W)

Register 31.82. UHCI_REG_Q3_WORD1_REG (0x0050)



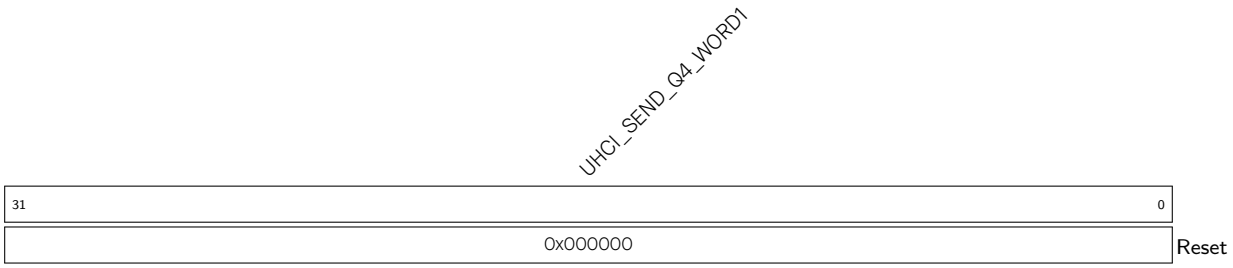
UHCI_SEND_Q3_WORD1 Data to be transmitted in Q3 register. (R/W)

Register 31.83. UHCI_REG_Q4_WORD0_REG (0x0054)



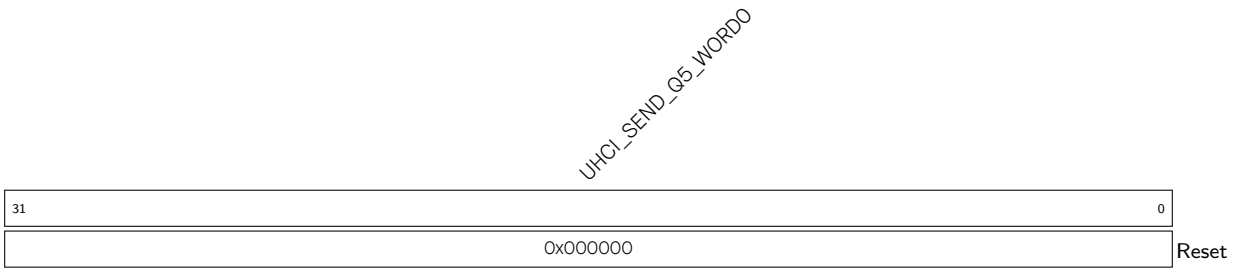
UHCI_SEND_Q4_WORD0 Data to be transmitted in Q4 register. (R/W)

Register 31.84. UHCI_REG_Q4_WORD1_REG (0x0058)



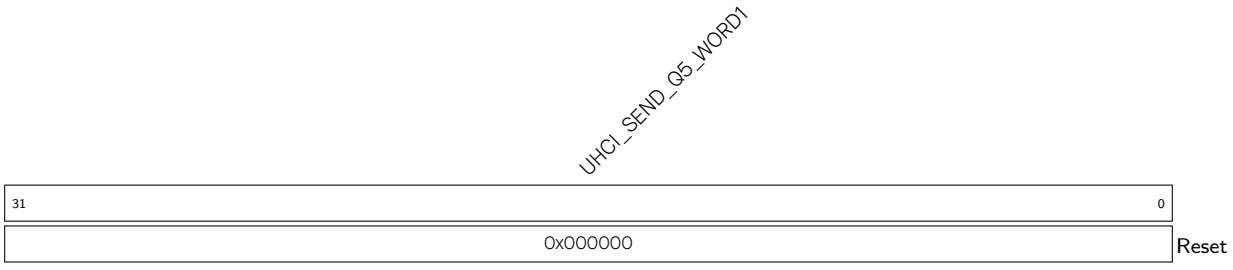
UHCI_SEND_Q4_WORD1 Data to be transmitted in Q4 register. (R/W)

Register 31.85. UHCI_REG_Q5_WORD0_REG (0x005C)



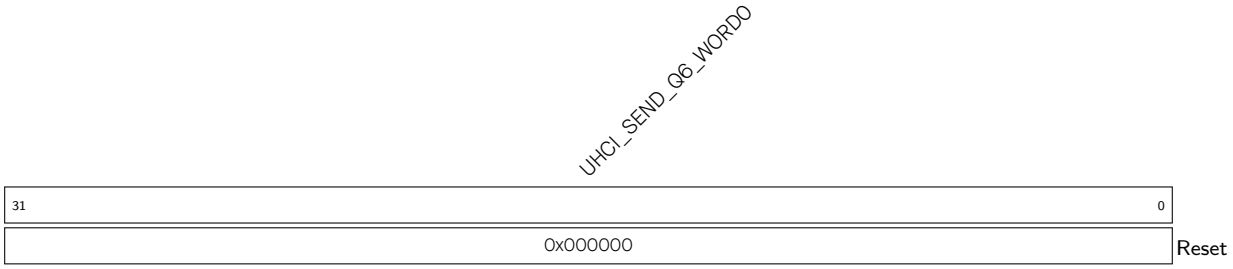
UHCI_SEND_Q5_WORD0 Data to be transmitted in Q5 register. (R/W)

Register 31.86. UHCI_REG_Q5_WORD1_REG (0x0060)



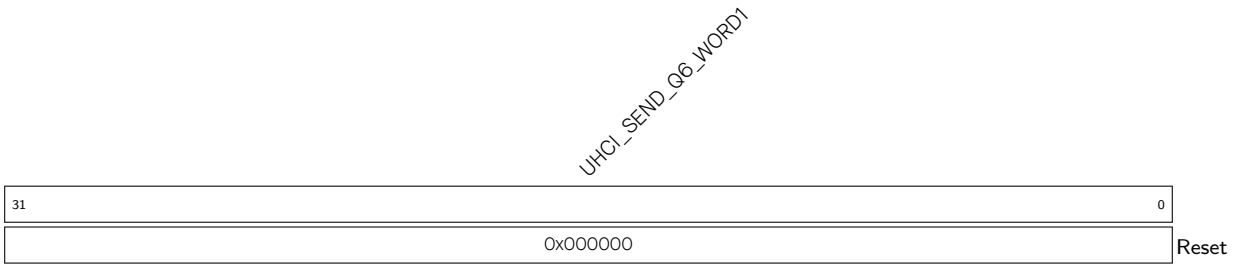
UHCI_SEND_Q5_WORD1 Data to be transmitted in Q5 register. (R/W)

Register 31.87. UHCI_REG_Q6_WORD0_REG (0x0064)



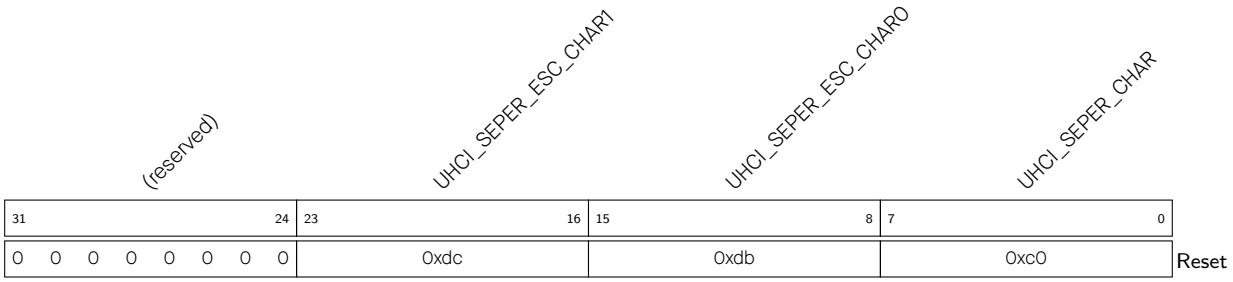
UHCI_SEND_Q6_WORD0 Data to be transmitted in Q6 register. (R/W)

Register 31.88. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Data to be transmitted in Q6 register. (R/W)

Register 31.89. UHCI_ESC_CONF0_REG (0x006C)

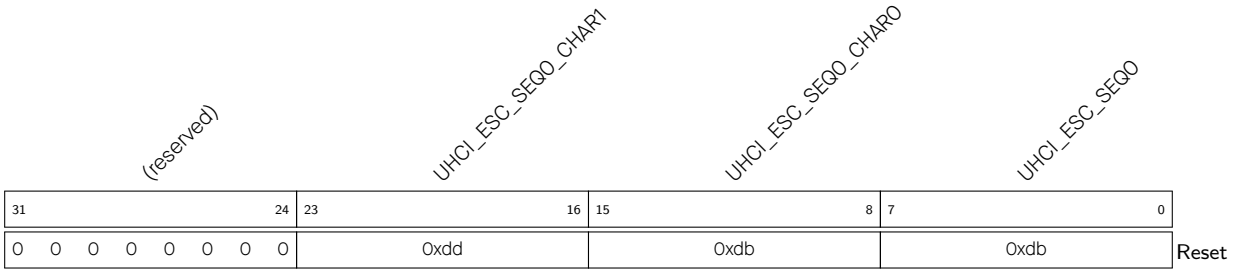


UHCI_SEPER_CHAR Configures separators to encode data packets. The default value is 0xC0. (R/W)

UHCI_SEPER_ESC_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_SEPER_ESC_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDC. (R/W)

Register 31.90. UHCI_ESC_CONF1_REG (0x0070)

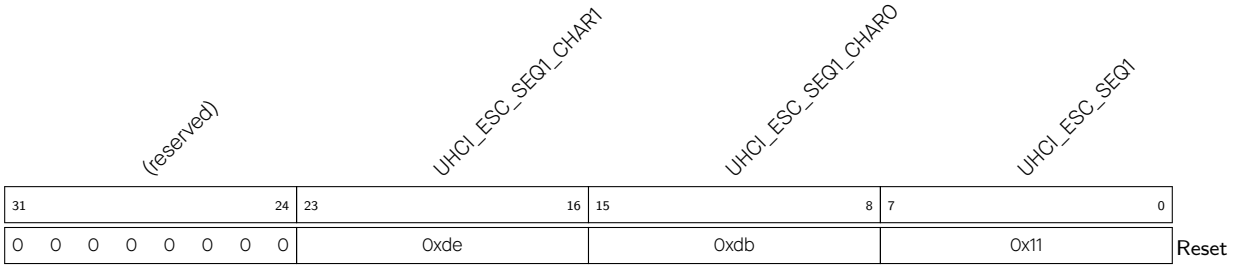


UHCI_ESC_SEQ0 Configures the character that needs to be encoded. The default value is 0xDB used as the first character of SLIP escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ0_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDD. (R/W)

Register 31.91. UHCI_ESC_CONF2_REG (0x0074)

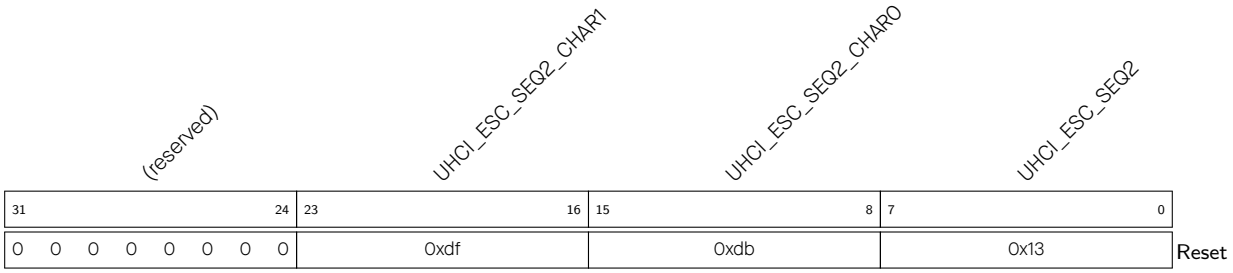


UHCI_ESC_SEQ1 Configures a character that need to be encoded. The default value is 0x11 used as a flow control character. (R/W)

UHCI_ESC_SEQ1_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

UHCI_ESC_SEQ1_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDE. (R/W)

Register 31.92. UHCI_ESC_CONF3_REG (0x0078)

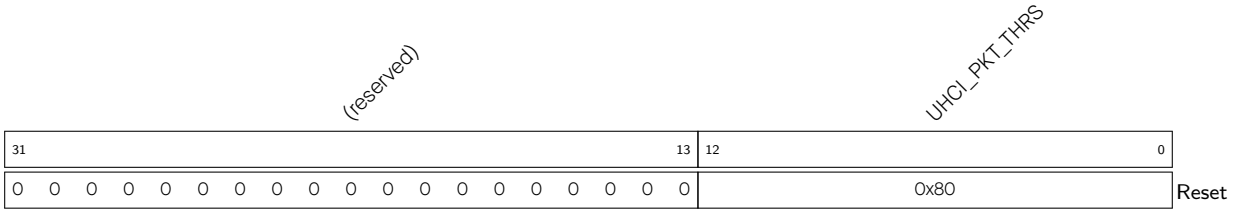


UHCI_ESC_SEQ2 Configures the character that needs to be decoded. The default value is 0x13 used as a flow control character. (R/W)

UHCI_ESC_SEQ2_CHAR0 Configures the first character of SLIP escape sequence. The default value is 0xDB. (R/W)

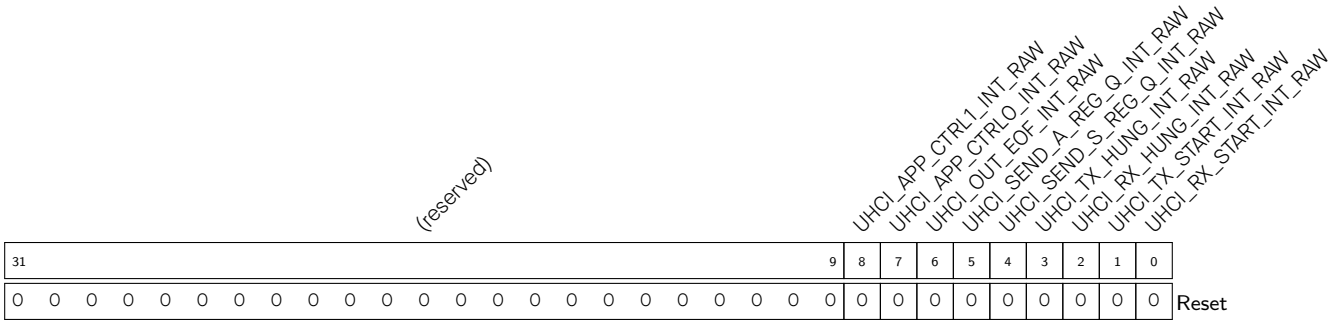
UHCI_ESC_SEQ2_CHAR1 Configures the second character of SLIP escape sequence. The default value is 0xDF. (R/W)

Register 31.93. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS Configures the maximum value of the packet length.
Measurement unit: byte.
Valid only when UHCI_HEAD_EN is 0. (R/W)

Register 31.94. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW The raw interrupt status of UHCI_RX_START_INT. (R/WTC/SS)

UHCI_TX_START_INT_RAW The raw interrupt status of UHCI_TX_START_INT. (R/WTC/SS)

UHCI_RX_HUNG_INT_RAW The raw interrupt status of UHCI_RX_HUNG_INT. (R/WTC/SS)

UHCI_TX_HUNG_INT_RAW The raw interrupt status of UHCI_TX_HUNG_INT. (R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_S_REG_Q_INT. (R/WTC/SS)

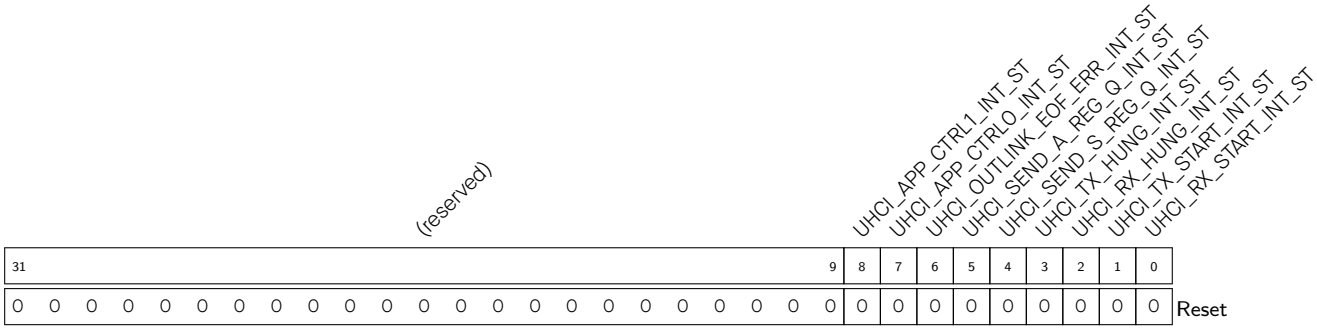
UHCI_SEND_A_REG_Q_INT_RAW The raw interrupt status of UHCI_SEND_A_REG_Q_INT. (R/WTC/SS)

UHCI_OUT_EOF_INT_RAW The raw interrupt status of UHCI_OUT_EOF_INT. (R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW The raw interrupt status of UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_RAW The raw interrupt status of UHCI_APP_CTRL1_INT. (R/W)

Register 31.95. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST The masked interrupt status of UHCI_RX_START_INT. (RO)

UHCI_TX_START_INT_ST The masked interrupt status of UHCI_TX_START_INT. (RO)

UHCI_RX_HUNG_INT_ST The masked interrupt status of UHCI_RX_HUNG_INT. (RO)

UHCI_TX_HUNG_INT_ST The masked interrupt status of UHCI_TX_HUNG_INT. (RO)

UHCI_SEND_S_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_S_REG_Q_INT. (RO)

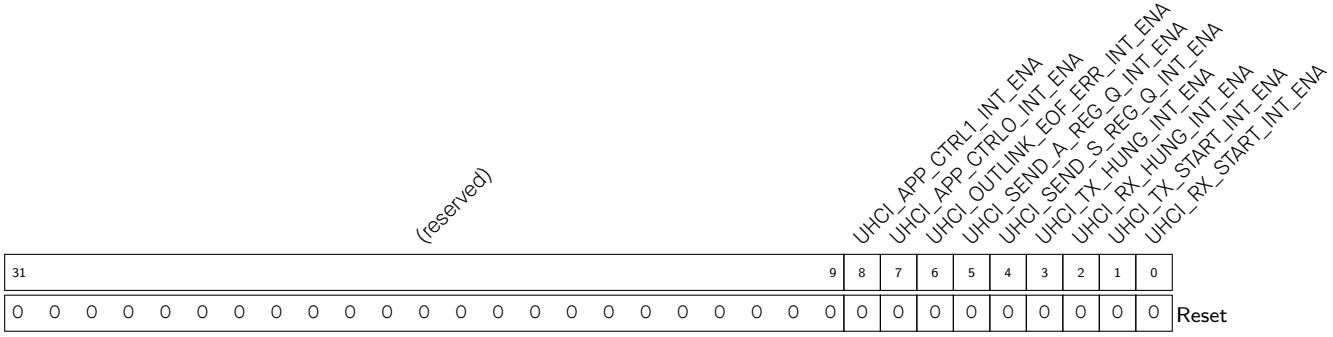
UHCI_SEND_A_REG_Q_INT_ST The masked interrupt status of UHCI_SEND_A_REG_Q_INT. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST The masked interrupt status of UHCI_OUTLINK_EOF_ERR_INT. (RO)

UHCI_APP_CTRL0_INT_ST The masked interrupt status of UHCI_APP_CTRL0_INT. (RO)

UHCI_APP_CTRL1_INT_ST The masked interrupt status of UHCI_APP_CTRL1_INT. (RO)

Register 31.96. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA Write 1 to enable UHCI_RX_START_INT. (R/W)

UHCI_TX_START_INT_ENA Write 1 to enable UHCI_TX_START_INT. (R/W)

UHCI_RX_HUNG_INT_ENA Write 1 to enable UHCI_RX_HUNG_INT. (R/W)

UHCI_TX_HUNG_INT_ENA Write 1 to enable UHCI_TX_HUNG_INT. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_S_REG_Q_INT. (R/W)

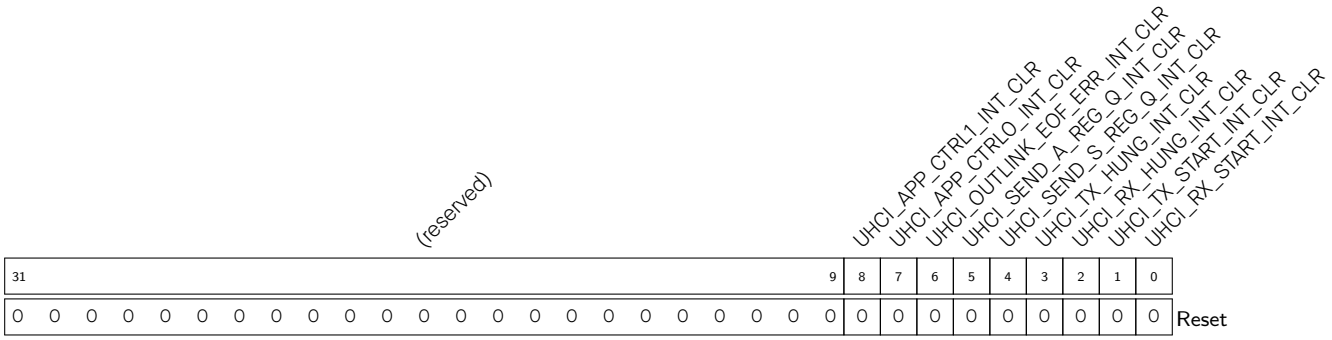
UHCI_SEND_A_REG_Q_INT_ENA Write 1 to enable UHCI_SEND_A_REG_Q_INT. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA Write 1 to enable UHCI_OUTLINK_EOF_ERR_INT. (R/W)

UHCI_APP_CTRL0_INT_ENA Write 1 to enable UHCI_APP_CTRL0_INT. (R/W)

UHCI_APP_CTRL1_INT_ENA Write 1 to enable UHCI_APP_CTRL1_INT. (R/W)

Register 31.97. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR Write 1 to clear UHCI_RX_START_INT. (WT)

UHCI_TX_START_INT_CLR Write 1 to clear UHCI_TX_START_INT. (WT)

UHCI_RX_HUNG_INT_CLR Write 1 to clear UHCI_RX_HUNG_INT. (WT)

UHCI_TX_HUNG_INT_CLR Write 1 to clear UHCI_TX_HUNG_INT. (WT)

UHCI_SEND_S_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_S_REG_Q_INT. (WT)

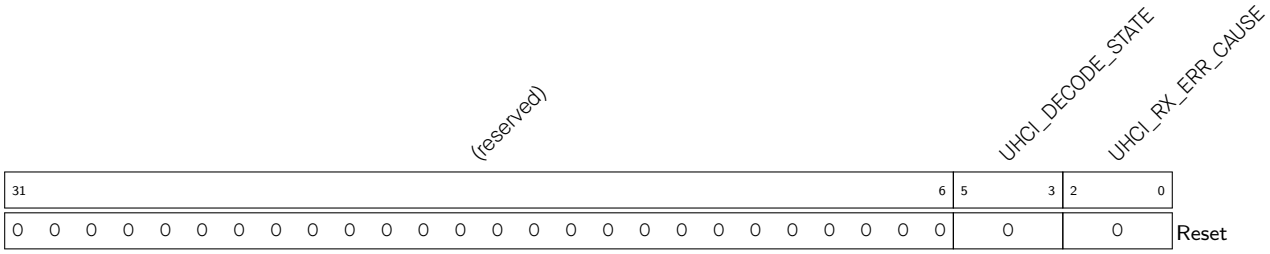
UHCI_SEND_A_REG_Q_INT_CLR Write 1 to clear UHCI_SEND_A_REG_Q_INT. (WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR Write 1 to clear UHCI_OUTLINK_EOF_ERR_INT. (WT)

UHCI_APP_CTRL0_INT_CLR Write 1 to clear UHCI_APP_CTRL0_INT. (WT)

UHCI_APP_CTRL1_INT_CLR Write 1 to clear UHCI_APP_CTRL1_INT. (WT)

Register 31.98. UHCI_STATE0_REG (0x0018)



UHCI_RX_ERR_CAUSE Represents the error type when DMA has received a packet with error.

0: Invalid. No effect

1: Checksum error in the HCI packet

2: Sequence number error in the HCI packet

3: CRC bit error in the HCI packet

4: 0xC0 is found but the received HCI packet is not complete 5: 0xC0 is not found when the HCI packet has been received

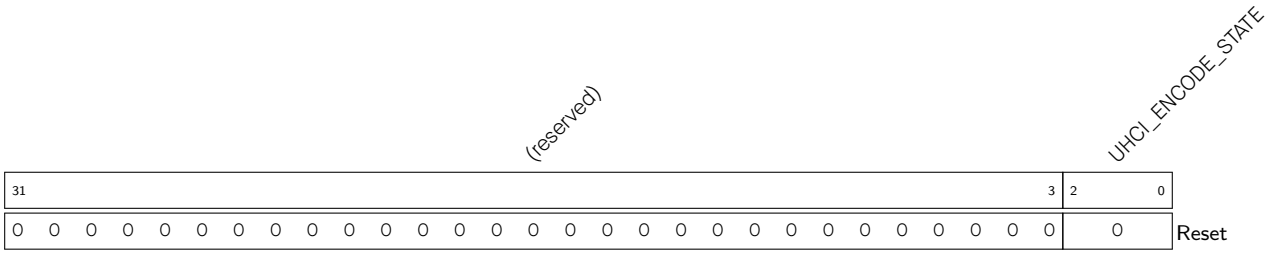
6: CRC check error

7: Invalid. No effect

(RO)

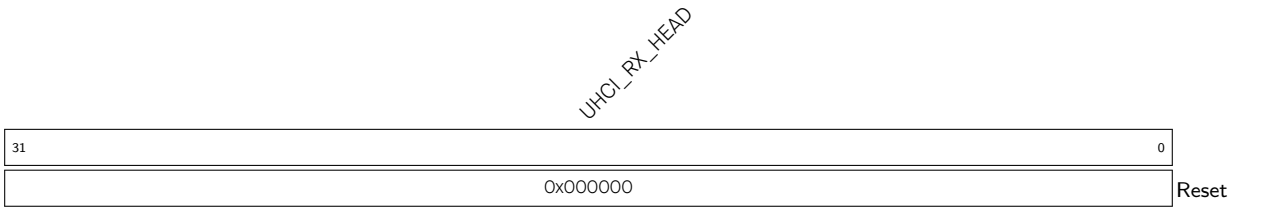
UHCI_DECODE_STATE Represents the UHCI decoder status. (RO)

Register 31.99. UHCI_STATE1_REG (0x001C)



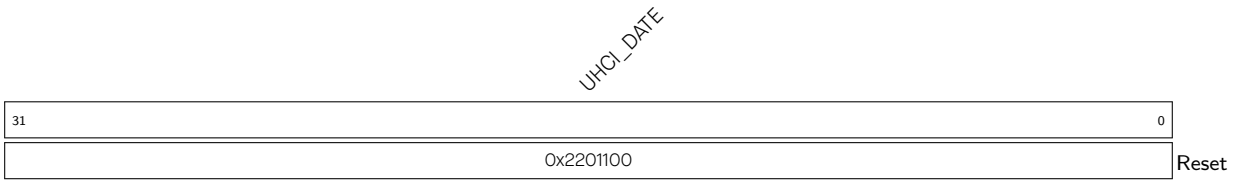
UHCI_ENCODE_STATE Represents the UHCI encoder status. (RO)

Register 31.100. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD Represents the header of the current received packet. (RO)

Register 31.101. UHCI_DATE_REG (0x0080)



UHCI_DATE Version control register. (R/W)

Chapter 32

I2C Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP32-P4 to communicate with multiple external devices. These external devices can share one I2C bus. ESP32-P4 has three I2C controllers: two in the main system and one in the low-power system. The two I2C controllers in the main system can act as a master or a slave (referred to as I2C below), while the one in the low-power system can only act as a master (referred to as LP_I2C below), which can still work when the main system sleeps.

32.1 Overview

The I2C bus has two lines, namely a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I2C bus can be connected to a single or multiple master devices and a single or multiple slave devices. However, only one master device can access a slave at a time via the bus.

The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high. Then it issues nine clock pulses via SCL. The first eight pulses are used to transmit a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of an I2C slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master and the slave can send or receive data according to the $R/\overline{W}$ bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once the communication has finished, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send a RSTART condition, a slave address, and a $R/\overline{W}$ bit before changing its operation. The RSTART condition is used to change the transfer direction and the mode of the devices (master mode or slave mode).

32.2 Features

The I2C controller of ESP32-P4 has the following features:

- Master mode and slave mode
- Communication between multiple masters and slaves
- Standard mode (100 Kbit/s)
- Fast mode (400 Kbit/s)
- 7-bit addressing and 10-bit addressing
- Continuous data transfer achieved by pulling SCL low in slave mode
- Programmable digital noise filtering
- Dual address mode, which uses slave address and slave memory or register address

32.3 I2C Architecture

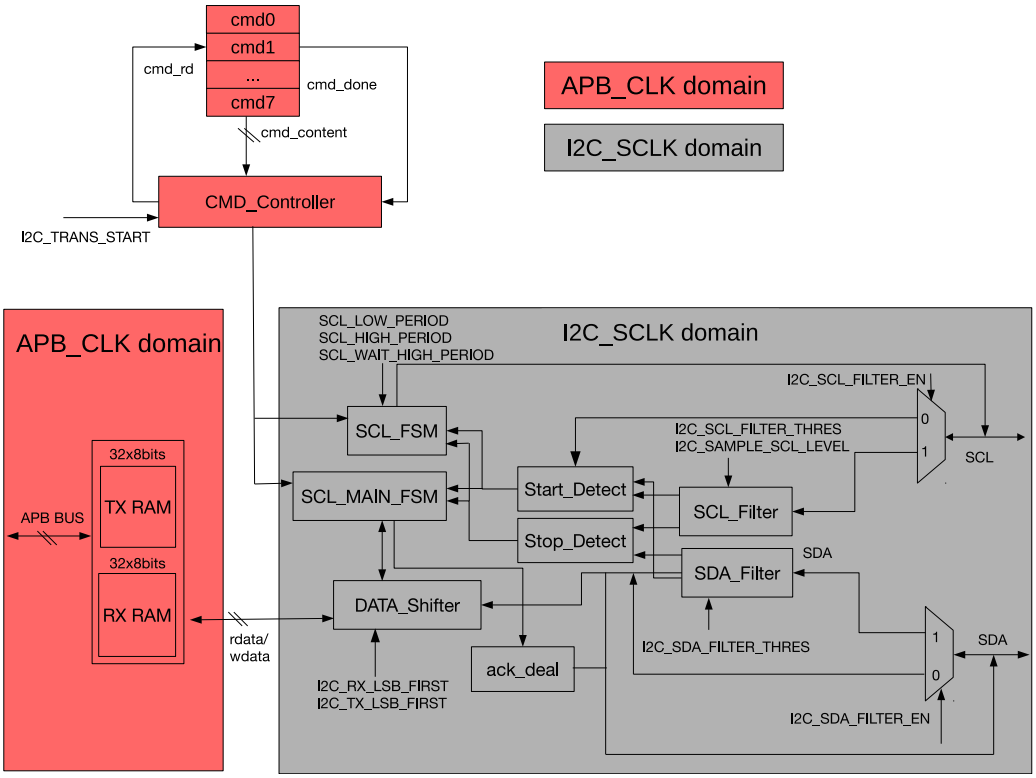


Figure 32.3-1. I2C Master Architecture

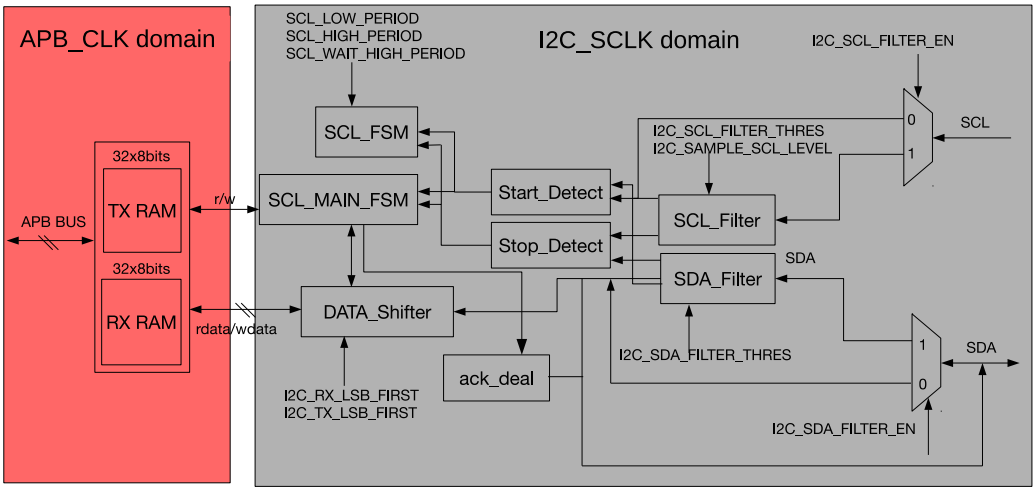


Figure 32.3-2. I2C Slave Architecture

The I2C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE. Figure 32.3-1 shows the architecture of a master, while Figure 32.3-2 shows that of a slave. The I2C controller has the following main parts:

- Transmit and receive memory (TX/RX RAM): store data to be transmitted and data received respectively.
- Command controller (CMD_Controller): generate (R)START, STOP, WRITE, READ, and END commands

- SCL clock controller (SCL_FSM): generate the timing sequence conforming to the I2C protocol. Figure 32.3-3 and Table 32.3-1 are the timing diagram and corresponding parameters of the I2C protocol.
- SDA data controller (SCL_MAIN_FSM): control the execution of I2C commands and the data sequence of the SDA line. It also controls the ACK_deal module to generate the ACK bit and detect the level of the ACK bit on the SDA line.
- Serial/parallel data converter (DATA_Shifter): shift data between serial and parallel form
- Filter for SCL (SCL_Filter): remove noises on SCL input signals
- Filter for SDA (SDA_Filter): remove noises on SDA input signals
- ACK bit controller (ack_deal): generate the ACK bit and detect the level of the ACK bit on the SDA line under the control of SCL_MAIN_FSM.

Besides, the I2C controller also has a clock module that generates I2C clocks, and a synchronization module that synchronizes the APB bus and the I2C controller.

The clock module is used to select clock sources, turn on and off clocks, and divide clocks. The synchronization module synchronizes signal transfer between different clock domains.

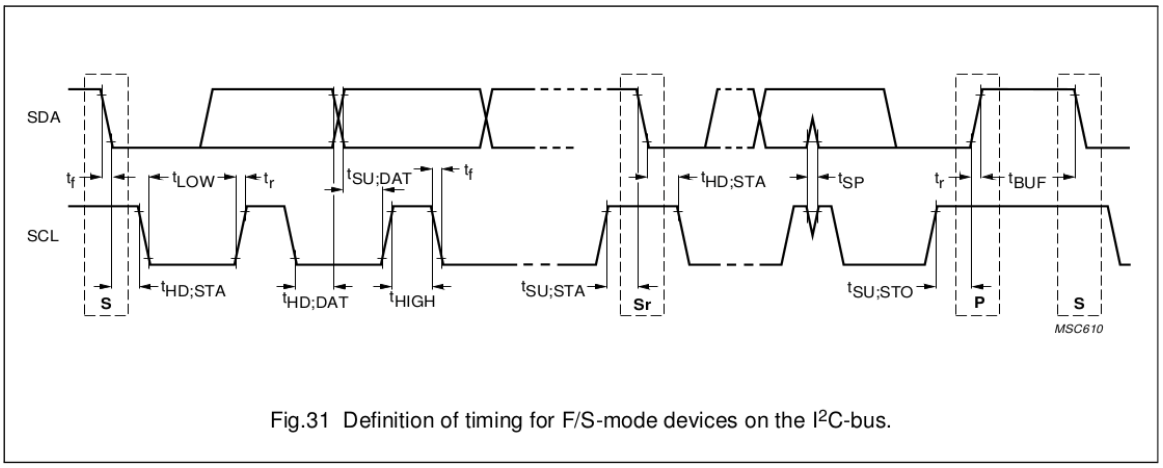


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

Figure 32.3-3. I2C Protocol Timing (Cited from Fig.31 in The I2C-bus specification Version 2.1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	$20 + 0.1C_b$(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	$20 + 0.1C_b$(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

Table 32.3-1. I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1)

32.4 Functional Description

As mentioned above, one or more masters and one or more slaves can be connected on the I2C bus. The following sections describe the operations of the ESP32-P4 I2C controllers. Note that operations may differ between the I2C controllers in ESP32-P4 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

32.4.1 Clock Configuration

Registers, TX RAM, and RX RAM are configured and accessed in the APB_CLK clock domain. The main logic of the I2C controller, including SCL_FSM, SCL_MAIN_FSM, SCL_FILTER, SDA_FILTER, and DATA_SHIFTER, are in the I2C_SCLK clock domain.

You can configure the clock source for I2C_SCLK of I2C0 in the main system (HP) to FOSC_20M_CLK or XTAL_40M_CLK via HP_SYS_CLKRST_I2C0_CLK_SRC_SEL and configure that of I2C1 in the main system via HP_SYS_CLKRST_I2C1_CLK_SRC_SEL. For the clock source for I2C_SCLK of LP_I2C in the low-power system (LP), you can configure it to XTAL_D2_CLK, LP_DYN_FAST_CLK, or PLL_LP_CLK 8 MHz via LPPERI_LP_I2C_CLK_SEL.

The steps to configure the clock source for HP I2C0 are as follows:

- Enable the clock source for I2C_SCLK of I2C0 by configuring HP_SYS_CLKRST_I2C0_CLK_EN to 1.
- When HP_SYS_CLKRST_I2C0_SRC_SEL is 0, the clock source is XTAL_40M_CLK.
- When HP_SYS_CLKRST_I2C0_SRC_SEL is 1, the clock source is FOSC_20M_CLK.

The steps to configure the clock source for HP I2C1 are as follows:

- Enable the clock source for I2C_SCLK of I2C1 by configuring HP_SYS_CLKRST_I2C1_CLK_EN to 1.

- When HP_SYS_CLKRST_I2C1_SRC_SEL is 0, the clock source is XTAL_40M_CLK.
- When HP_SYS_CLKRST_I2C1_SRC_SEL is 1, the clock source is FOSC_20M_CLK.

The steps to configure the clock source for LP_I2C are as follows:

- Enable the clock source for I2C_SCLK of LP_I2C by configuring LPPERI_CK_EN_LP_I2C to 1.
- When LPPERI_LP_I2C_CLK_SEL is 0, the clock source is LP_DYN_FAST_CLK.
- When LPPERI_LP_I2C_CLK_SEL is 1, the clock source is XTAL_D2_CLK.
- When LPPERI_LP_I2C_CLK_SEL is 2, the clock source is PLL_LP_CLK 8 MHz.

The clock source then passes through a fractional divider to generate I2C_SCLK according to the following equation:

$$I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

In the equation, I2C_SCLK_DIV_NUM represents the integer part of the divisor, I2C_SCLK_DIV_A represents the numerator of the fractional part of the divisor, and I2C_SCLK_DIV_B represents the denominator of the fractional part of the divisor. Limited by timing parameters, the derived clock I2C_SCLK should operate at a frequency 20 times larger than SCL's frequency.
For I2C0:

- Configure I2C_SCLK_DIV_NUM via HP_SYS_CLKRST_I2C0_CLK_DIV_NUM.
- Configure I2C_SCLK_DIV_A via HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR.
- Configure I2C_SCLK_DIV_B via HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR.

For I2C1:

- Configure I2C_SCLK_DIV_NUM via HP_SYS_CLKRST_I2C1_CLK_DIV_NUM.
- Configure I2C_SCLK_DIV_A via HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR.
- Configure I2C_SCLK_DIV_B via HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR.

32.4.2 SCL and SDA Noise Filtering

SCL_Filter and SDA_Filter modules are identical and are used to filter signal noise on SCL and SDA, respectively. These filters can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN.

Take SCL_Filter as an example. When enabled, SCL_Filter samples input signals on the SCL line continuously. These input signals are valid only if they remain unchanged for consecutive I2C_SCL_FILTER_THRES I2C_SCLK clock cycles. Given that only valid input signals can pass through the filter, SCL_Filter can remove glitches whose pulse width is shorter than I2C_SCL_FILTER_THRES I2C_SCLK clock cycles, while SDA_Filter can remove glitches whose pulse width is shorter than I2C_SDA_FILTER_THRES I2C_SCLK clock cycles.

32.4.3 SCL Clock Stretching

The I2C controller in slave mode (i.e., slave) can realize the function called clock stretching by holding the SCL line low to suspend data transmission in exchange for more time to process data. This function is enabled

by setting the I2C_SLAVE_SCL_STRETCH_EN bit. The time period to release the SCL line from stretching is configured by setting the I2C_STRETCH_PROTECT_NUM field, in order to avoid timing sequence errors. The slave can choose to achieve clock stretching by holding the SCL line low when one of the following four events occurs:

1. Address match: The address of the slave matches the address sent by the master via the SDA line, and the $R/\overline{W}$ bit is 1.
2. RAM being full: RX RAM of the slave is full. Note that when the slave receives less than the FIFO depth, which is 32 bytes in ESP32-P4 I2C, it is not necessary to enable clock stretching; when the slave receives FIFO depth bytes or more, you may interrupt data transmission to wrapped around RAM via the FIFO threshold, or enable clock stretching for more time to process data. When clock stretching is enabled, I2C_RX_FULL_ACK_LEVEL must be cleared, otherwise there will be unpredictable consequences.
3. RAM being empty: The slave is sending data, but its TX RAM is empty.
4. Sending an ACK: If I2C_SLAVE_BYTE_ACK_CTL_EN is set, the slave pulls SCL low when sending an ACK bit. At this stage, software validates data and configures I2C_SLAVE_BYTE_ACK_LVL to control the level of the ACK bit. Note that when RX RAM of the slave is full, the level of the ACK bit to be sent is determined by I2C_RX_FULL_ACK_LEVEL, instead of I2C_SLAVE_BYTE_ACK_LVL. In this case, I2C_RX_FULL_ACK_LEVEL should also be cleared to ensure proper functioning of clock stretching.

When clock stretching occurs, the cause of stretching can be read from the I2C_STRETCH_CAUSE bit. Clock stretching can be disabled by setting the I2C_SLAVE_SCL_STRETCH_CLR bit.

32.4.4 Generating SCL Pulses in Idle State

Usually, when the I2C bus is idle, the SCL line is held high. The I2C controller in ESP32-P4 can be programmed to generate SCL pulses in an idle state. This function only works when the I2C controller is configured as master. If the I2C_SCL_RST_SLV_EN bit is set, hardware will send I2C_SCL_RST_SLV_NUM SCL pulses, and then automatically clear the I2C_SCL_RST_SLV_EN bit.

32.4.5 Synchronization

I2C registers are configured in the APB_CLK domain, whereas the I2C controller is configured in the asynchronous I2C_SCLK domain. Therefore, before being used by the I2C controller, register values should be synchronized by first writing configuration registers and then writing 1 to I2C_CONF_UPGATE. Registers that need synchronization are listed in Table 32.4-1.

Table 32.4-1. I2C Synchronous Registers

Register	Field	Address
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

32.4.6 Open-Drain Output

SCL and SDA output drivers must be configured as open-drain. There are two ways to achieve this:

1. Set I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT, and configure GPIO_PIN*n*_PAD_DRIVER for corresponding SCL and SDA pads as open-drain.
2. Clear I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT.

Because these lines are configured as open-drain, the low-to-high transition time of each line is longer, determined together by the pull-up resistor and line capacitance. The output duty cycle of I2C is limited by the SDA and SCL line's pull-up speed, mainly SCL's speed.

In addition, when I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN are set to 1, SCL can be forced low; when I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN are set to 1, SDA can be forced low.

32.4.7 Timing Parameter Configuration

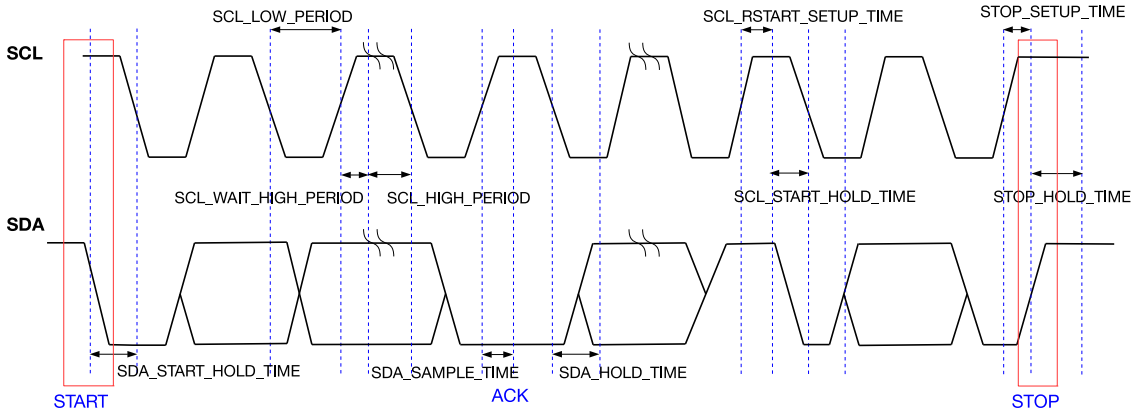


Figure 32.4-1. I2C Timing Diagram

Figure 32.4-1 shows the timing diagram of an I2C master. This figure also specifies registers used to configure the START bit, STOP bit, data hold time, data sample time, waiting time on the rising SCL edge, etc. Timing parameters are calculated as follows in I2C_SCLK clock cycles:

1. $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2. $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3. $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4. $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5. $t_{r} = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6. $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7. $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8. $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

Timing registers below are divided into two groups, depending on the mode in which these registers are active:

- Master mode only:

1. I2C_SCL_START_HOLD_TIME: Specifies the interval between the moment SDA is pulled low and the moment SCL is pulled low when the master generates a START condition. This interval is (I2C_SCL_START_HOLD_TIME +1) in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode.
2. I2C_SCL_LOW_PERIOD: Specifies the low period of SCL. This period lasts (I2C_SCL_LOW_PERIOD +1) in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode.

   However, this period could be extended in the following scenarios:

   - SCL is pulled low by peripheral devices when I2C acts as a master.
   - SCL is pulled low by an END command executed by the I2C controller.
   - SCL is pulled low by clock stretching when I2C acts as a slave.
3. I2C_SCL_WAIT_HIGH_PERIOD: Specifies the time for SCL to switch from low to high in I2C_SCLK cycles. Please make sure that SCL can be pulled high within this time period. Otherwise, the high period of SCL may be incorrect. This register is active only when the I2C controller works in master mode.
4. I2C_SCL_HIGH_PERIOD: Specifies the high period of SCL in I2C_SCLK cycles. This register is active only when the I2C controller works in master mode. When SCL goes high within (I2C_SCL_WAIT_HIGH_PERIOD + 1) in I2C_SCLK cycles, its frequency is:

   $$f_{scl} = \frac{f_{\text{I2C\_SCLK}}}{\text{I2C\_SCL\_LOW\_PERIOD} + \text{I2C\_SCL\_HIGH\_PERIOD} + \text{I2C\_SCL\_WAIT\_HIGH\_PERIOD} + 3 + \text{I2C\_SCL\_FILTER\_THRES}}$$

   where 3 represents the amount of clock cycles required to synchronize the SCL. If the SCL filtering function is turned on, the delay caused by I2C_SCL_FILTER_THRES needs to be added. As the SCL low-to-high transition time represented by I2C_SCL_WAIT_HIGH_PERIOD + 1 module clock can be affected by the pull-up resistor, IO drive capability, SCL line capacitance, etc., deviation may occur between the actual frequency of the test and the theoretical frequency. At this point, deviations can be reduced by adjusting the value of I2C_SCL_WAIT_HIGH_PERIOD.

- Master mode and slave mode:
  1. I2C_SDA_SAMPLE_TIME: Specifies the interval between the rising edge of SCL and the level sampling time of SDA. It is advised to set a value in the middle of SCL's high period, so as to correctly sample the level of SCL. This register is active both in master mode and slave mode.
  2. I2C_SDA_HOLD_TIME: Specifies the interval between changing the SDA output level and the falling edge of SCL. This register is active both in master mode and slave mode.

Timing parameters limits corresponding register configuration.

1. $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME + I2C_SCL_RSTART_SETUP_TIME
6. I2C_STRETCH_PROTECT_NUM + I2C_SDA_HOLD_TIME > I2C_SCL_LOW_PERIOD

32.4.8 Timeout Control

The I2C controller has three types of timeout control, namely timeout control for SCL_FSM, for SCL_MAIN_FSM, and for the SCL line. The first two are always enabled, while the third is configurable.

When SCL_FSM remains unchanged for more than $2^{I2C\_SCL\_ST\_TO\_I2C}$ clock cycles, an I2C_SCL_ST_TO_INT interrupt is triggered, and then SCL_FSM goes to idle state. The value of I2C_SCL_ST_TO_I2C should be less than or equal to 22, which means SCL_FSM could remain unchanged for 2^{22} I2C_SCLK clock cycles at most before the interrupt is generated.

When SCL_MAIN_FSM remains unchanged for more than $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C}$ I2C_SCLK clock cycles, an I2C_SCL_MAIN_ST_TO_INT interrupt is triggered, and then SCL_MAIN_FSM goes to idle state. The value of I2C_SCL_MAIN_ST_TO_I2C should be less than or equal to 22, which means SCL_MAIN_FSM could remain unchanged for 2^{22} clock cycles at most before the interrupt is generated.

Timeout control for SCL is enabled by setting I2C_TIME_OUT_EN. When the level of SCL remains unchanged for more than $2^{I2C\_TIME\_OUT\_VALUE}$ clock cycles, an I2C_TIME_OUT_INT interrupt is triggered, and then the I2C bus goes to idle state.

32.4.9 Command Configuration

When the I2C controller works in master mode, CMD_Controller reads commands from 8 sequential command registers and controls SCL_FSM and SCL_MAIN_FSM accordingly.

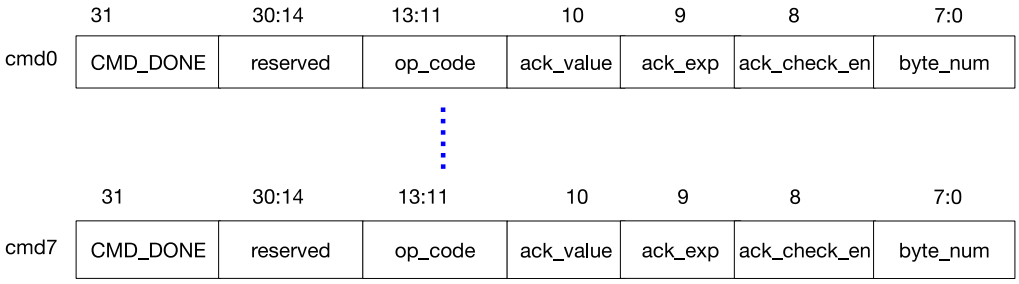


Figure 32.4-2. Structure of I2C Command Registers

Command registers, whose structure is illustrated in Figure 32.4-2, are active only when the I2C controller works in master mode. Fields of command registers are:

1. CMD_DONE: Indicates that a command has been executed. After each command has been executed, the CMD_DONE bit in the corresponding command register is set to 1 by hardware. By reading this bit, the software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. op_code: Indicates the command. The I2C controller supports five commands:
   - WRITE: op_code = 1. The I2C controller sends a slave address, a register address (only in dual address mode), and data to the slave.
   - STOP: op_code = 2. The I2C controller sends a STOP bit defined by the I2C protocol. This code also indicates that the command sequence has been executed, and the CMD_Controller stops reading commands. After restarted by software, the CMD_Controller resumes reading commands from command register 0.

- READ: op_code = 3. The I2C controller reads data from the slave.
- END: op_code = 4. The I2C controller pulls the SCL line down and suspends I2C communication. This code also indicates that the command sequence has been completed, and the CMD_Controller stops executing commands. Once the software refreshes data in command registers and the RAM, the CMD_Controller can be restarted to execute commands from command register 0 again.
- RSTART: op_code = 6. The I2C controller sends a START bit or a RSTART bit defined by the I2C protocol.

3. ack_value: Used to configure the level of the ACK bit sent by the I2C controller during a read operation. This bit is ignored in RSTART, STOP, END, and WRITE conditions.
4. ack_exp: Used to configure the level of the ACK bit expected by the I2C controller during a write operation. This bit is ignored during RSTART, STOP, END, and READ conditions.
5. ack_check_en: Used to enable the I2C controller during a write operation to check whether the ACK level sent by the slave matches ack_exp in the command. If this bit is set and the level received does not match ack_exp in the WRITE command, the master will generate an I2C_NACK_INT interrupt and a STOP condition for data transfer. If this bit is cleared, the controller will not check the ACK level sent by the slave. This bit is ignored during RSTART, STOP, END, and READ conditions.
6. byte_num: Specifies the length of data (in bytes) to be read or written. It can range from 1 to 255 bytes. This bit is ignored during RSTART, STOP, and END conditions.

Each command sequence is executed starting from command register 0 and terminated by a STOP or an END. Therefore, there must be a STOP or an END command in the eight command registers.

A complete data transfer on the I2C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, separated by END commands. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I2C communication.

32.4.10 TX/RX RAM Data Storage

Both TX RAM and RX RAM are 32 × 8 bits and can be accessed in FIFO or non-FIFO mode. If I2C_NONFIFO_EN bit is cleared, both RAMs are accessed in FIFO mode; if I2C_NONFIFO_EN bit is set, both RAMs are accessed in non-FIFO mode.

TX RAM stores data that the I2C controller needs to send. During communication, when the I2C controller needs to send data (except acknowledgment bits), it reads data from TX RAM and sends them sequentially via SDA. When the I2C controller works in master mode, all data must be stored in TX RAM in the order they need to be sent to slaves. The data stored in TX RAM include slave addresses, read/write bits, register addresses (only in dual address mode) and data to be sent. When the I2C controller works in slave mode, TX RAM only stores data to be sent.

TX RAM can be read and written by the CPU. The CPU writes to TX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU writes to TX RAM via the fixed address I2C_DATA_REG, with addresses for writing in TX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x100) ~(I2C Base Address + 0x17C). Each byte in TX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base

Address + 0x100, the second byte is I2C Base Address + 0x104, the third byte is I2C Base Address + 0x108, and so on. The CPU can only read TX RAM via direct addresses. Bytes written to the TX RAM can be read back by the CPU, via the direct addresses. Addresses for reading TX RAM are the same as addresses for writing TX RAM.

RX RAM stores data the I2C controller receives during communication. When the I2C controller works in slave mode, neither slave addresses sent by the master nor register addresses (only in dual address mode) will be stored in RX RAM. Values of RX RAM can be read by software after I2C communication is completed.

RX RAM can only be read by the CPU. The CPU reads RX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU reads RX RAM via the fixed address I2C_DATA_REG, with addresses for reading RX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x180) ~(I2C Base Address + 0x1FC). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base Address + 0x180, the second byte is I2C Base Address + 0x184, the third byte is I2C Base Address + 0x188 and so on.

In FIFO mode, the TX RAM of a master may wrap around to send data larger than the FIFO depth. Set I2C_FIFO_PRT_EN. If the size of data to be sent is smaller than I2C_TXFIFO_WM_THRHD (master), an I2C_TXFIFO_WM_INT (master) interrupt is generated. After receiving the interrupt, the software continues writing to I2C_DATA_REG (master). Please ensure that software writes to or refreshes TX RAM before the master sends data, otherwise it may result in unpredictable consequences.

In FIFO mode, the RX RAM of a slave may also wrap around to receive data larger than the FIFO depth. Set I2C_FIFO_PRT_EN and clear I2C_RX_FULL_ACK_LEVEL. If data already received (to be overwritten) is larger than I2C_RXFIFO_WM_THRHD (slave), an I2C_RXFIFO_WM_INT (slave) interrupt is generated. After receiving the interrupt, the software continues reading from I2C_DATA_REG (slave).

32.4.11 Data Conversion

DATA_Shifter is used for serial/parallel conversion, converting byte data in TX RAM to an outgoing serial bitstream or an incoming serial bitstream to byte data in RX RAM. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used to select LSB- or MSB-first storage and transmission of data.

32.4.12 Addressing Mode

The ESP32-P4 I2C controller supports 7-bit and 10-bit addressing. 10-bit addressing can be mixed with 7-bit addressing. Besides, the ESP32-P4 I2C controller also supports dual address mode.

Define the slave address as SLV_ADDR. In 7-bit addressing mode, the slave address is SLV_ADDR[6:0]; in 10-bit addressing mode, the slave address is SLV_ADDR[9:0].

In 7-bit addressing mode, the master only needs to send one byte of the address, which comprises SLV_ADDR[6:0] and a $R/\overline{W}$ bit. In the 7-bit addressing mode, there is a special case called general call addressing (broadcast). It is enabled by setting I2C_ADDR_BROADCASTING_EN in a slave. When the slave receives the general call address (0x00) from the master and the $R/\overline{W}$ bit followed is 0, it responds to the master regardless of its own address.

In 10-bit addressing mode, the master needs to send two bytes of address. The first byte is slave_addr_first_7bits followed by a $R/\overline{W}$ bit, and slave_addr_first_7bits should be configured as (0x78 | SLV_ADDR[9:8]). The second byte is slave_addr_second_byte, which should be configured as SLV_ADDR[7:0].

The slave can enable 10-bit addressing by configuring I2C_ADDR_10BIT_EN. I2C_SLAVE_ADDR is used to configure I2C slave address. Specifically, I2C_SLAVE_ADDR[14:7] should be configured as SLV_ADDR[7:0], and I2C_SLAVE_ADDR[6:0] should be configured as (0x78 | SLV_ADDR[9:8]). Since a 10-bit slave address has one more byte than a 7-bit address, byte_num of the WRITE command and the number of bytes in the RAM increase by one. Please refer to Programming Example for detailed descriptions.

When working in slave mode, the I2C controller supports dual address mode, where the first address is the address of an I2C slave, and the second one is the slave's memory address. When using dual address mode, RAM must be accessed in non-FIFO mode. Dual address mode is enabled by setting I2C_FIFO_ADDR_CFG_EN. When the slave address received by the slave is inconsistent with the internally configured slave address, the I2C_SLAVE_ADDR_UNMATCH interrupt will be generated.

32.4.13 $R/\overline{W}$ Bit Check in 10-bit Addressing Mode

In 10-bit addressing mode, when I2C_ADDR_10BIT_RW_CHECK_EN is set to 1, the I2C controller performs a check on the first byte, which consists of slave_addr_first_7bits and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit does not indicate a WRITE operation, i.e., not in line with the I2C protocol, the data transfer ends. If the check feature is not enabled, when the $R/\overline{W}$ bit does not indicate a WRITE, the data transfer still continues, but transfer failure may occur.

32.4.14 To Start the I2C Controller

To start the I2C controller in master mode, after configuring the controller to master mode and command registers, write 1 to I2C_TRANS_START in order to let the master start to parse and execute command sequences. The master always executes a command sequence starting from the command register 0 to a STOP or an END. To execute another command sequence starting from command register 0, refresh commands by writing 1 again to I2C_TRANS_START.

There are two ways to start the I2C controller in slave mode:

- Set I2C_SLV_TX_AUTO_START_EN, and the slave starts automatic transfer upon an address match;
- Clear I2C_SLV_TX_AUTO_START_EN, and always set I2C_TRANS_START before accepting any transfer.

32.5 Functional Differences Between LP_I2C and I2C

LP_I2C can be used as a master to communicate with external devices when the main system sleeps. LP_I2C includes all the functions of the ESP32-P4 $I2C_{master}$, but doesn't include any functions of ESP32-P4 $I2C_{slave}$. It does not contain any registers related to the $I2C_{slave}$. For detailed register list, see 32.8.2 LP_I2C Register Summary.

The design differences between LP_I2C and I2C master are as follows:

- The size of TX/RX RAM in LP_I2C is 16*8 bit, which means the TX/RX FIFO depth is 16 bytes.
- The clock source of APB_CLK in LP_I2C is CLK_AON_FAST. The clock source for I2C_SCLK is configured via LPPERI_LP_I2C_CLK_SEL. The corresponding bits are:
  - 0: CLK_ROOT_FAST
  - 1: CLK_XTALD2

- 2: PLL_F8M_CLK

Configuring LPPERI_CK_EN_LP_I2C to 1 enables the clock source of I2C_SCLK. Adjust the timing registers accordingly when the clock frequency changes.

See the programming examples of ESP32-P4 $I2C_{slave}$ in 32.6 for that of LP_I2C.

32.6 Programming Example

This section provides programming examples for typical communication scenarios. ESP32-P4 has two I2C controllers. For the convenience of description, I2C masters and slaves in all subsequent figures are ESP32-P4 I2C controllers. I2C master is referred to as $I2C_{master}$, and I2C slave is referred to as $I2C_{slave}$.

32.6.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence

32.6.1.1 Introduction

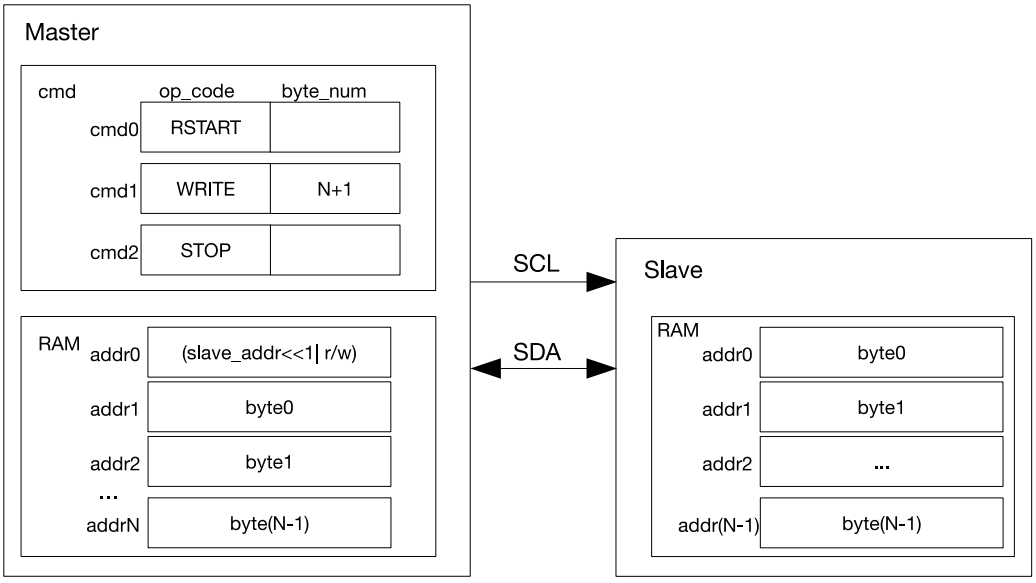


Figure 32.6-1. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address

Figure 32.6-1 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit addressing. As shown in figure 32.6-1, the first byte in the RAM of $I2C_{master}$ is a 7-bit $I2C_{slave}$ address followed by a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 0, it indicates a WRITE operation. The remaining bytes are used to store data ready for transfer. The cmd box contains related command sequences.

After the command sequence is configured and data in RAM is ready, $I2C_{master}$ enables the controller and initiates data transfer by setting the I2C_TRANS_START bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL being used by other masters or slaves.
2. Execute a RSTART command by sending a START bit.
3. Execute a WRITE command by taking N+1 bytes from the RAM in order and sending them to $I2C_{slave}$ in the same order. The first byte is the address of $I2C_{slave}$.

4. Execute a STOP command. Once the $I2C_{master}$ transfers a STOP bit, an I2C_TRANS_COMPLETE_INT interrupt is generated.

32.6.1.2 Configuration Example

1. Configure the timing parameter registers of $I2C_{master}$ and $I2C_{slave}$ according to Section 32.4.7.
2. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command register	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
10. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
11. If data to be sent (N) is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 32.4.10.
12. If data to be received (N) is larger than RX FIFO depth, RX RAM of $I2C_{slave}$ may wrap around in FIFO mode. For details, please refer to Section 32.4.10.

    If data to be received (N) is larger than RX FIFO depth, the other way is to enable clock stretching by setting the I2C_SLAVE_SCL_STRETCH_EN (slave), and clearing I2C_RX_FULL_ACK_LEVEL. When RX RAM is full, an I2C_SLAVE_STRETCH_INT (slave) interrupt is generated. In this way, $I2C_{slave}$ can hold SCL low, in exchange for more time to read data. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.

13. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.2 $I2C_{master}$ Writes to $I2C_{slave}$ with a 10-bit Address in One Command Sequence

32.6.2.1 Introduction

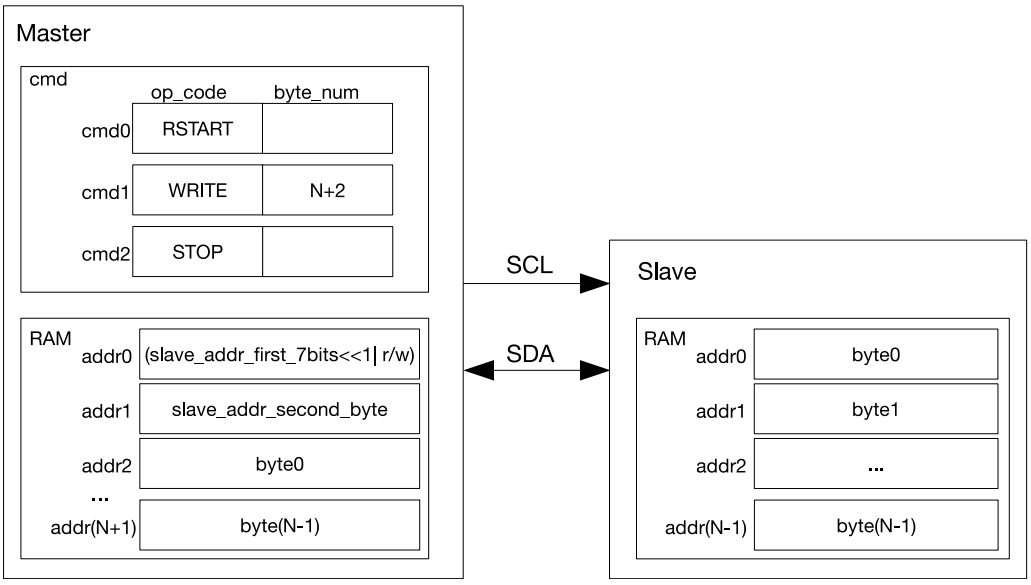


Figure 32.6-2. $I2C_{master}$ Writing to a Slave with a 10-bit Address

Figure 32.6-2 shows how $I2C_{master}$ writes N bytes of data using 10-bit addressing to an I2C slave. The configuration and transfer process is similar to what is described in 32.6.1, except that a 10-bit $I2C_{slave}$ address is formed from two bytes. Since a 10-bit $I2C_{slave}$ address has one more byte than a 7-bit $I2C_{slave}$ address, byte_num and length of data in TX RAM increase by 1 accordingly.

32.6.2.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2 (master)	STOP	—	—	—	—

4. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) as $I2C_{slave}$'s 10-bit address, and set I2C_ADDR_10BIT_EN (slave) to 1 to enable 10-bit addressing.

5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$. The first byte of the address of $I2C_{slave}$ comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit. The second byte of the address of $I2C_{slave}$ is I2C_SLAVE_ADDR[7:0]. These two bytes are followed by data to be sent in FIFO or non-FIFO mode.
6. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
7. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
10. If data to be sent is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 32.4.10.
11. If data to be received is larger than RX FIFO depth, RX RAM of $I2C_{slave}$ may wrap around in FIFO mode. For details, please refer to Section 32.4.10.

    If data to be received is larger than RX FIFO depth, the other way is to enable clock stretching by setting I2C_SLAVE_SCL_STRETCH_EN (slave), and clearing I2C_RX_FULL_ACK_LEVEL to 0. When RX RAM is full, an I2C_SLAVE_STRETCH_INT (slave) interrupt is generated. In this way, $I2C_{slave}$ can hold SCL low, in exchange for more time to read data. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
12. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

32.6.3.1 Introduction

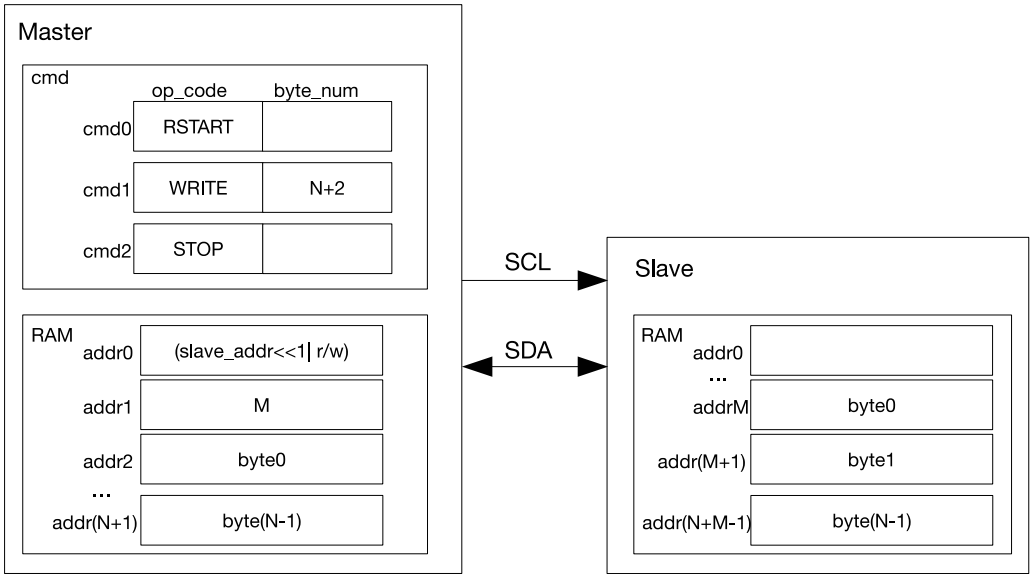


Figure 32.6-3. $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses

Figure 32.6-3 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$ registers or RAM using 7-bit double addressing. The configuration and transfer process is similar to what is described in Section 32.6.1, except that in 7-bit dual address mode $I2C_{master}$ sends two 7-bit addresses. The first address is the address of an I2C slave, and the second one is $I2C_{slave}$'s memory address (i.e., addrM in Figure 32.6-3). When using double addressing, RAM must be accessed in non-FIFO mode. The I2C slave puts received byte0 ~ byte (N-1) into its RAM in an order starting from addrM. The RAM is overwritten every 32 bytes.

32.6.3.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.

9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
10. $I2C_{slave}$ receives the RX RAM address sent by $I2C_{master}$ and adds the offset.
11. $I2C_{master}$ sends data, and determines whether to check ACK value according to ack_check_en (master).
12. If data to be sent is larger than TX FIFO depth, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 32.4.10.
13. If data to be received is larger than RX FIFO depth, you may enable clock stretching by setting I2C_SLAVE_SCL_STRETCH_EN (slave), and clearing I2C_RX_FULL_ACK_LEVEL to 0. When RX RAM is full, an I2C_SLAVE_STRETCH_INT (slave) interrupt is generated. In this way, $I2C_{slave}$ can hold SCL low, in exchange for more time to read data. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
14. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.4 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

32.6.4.1 Introduction

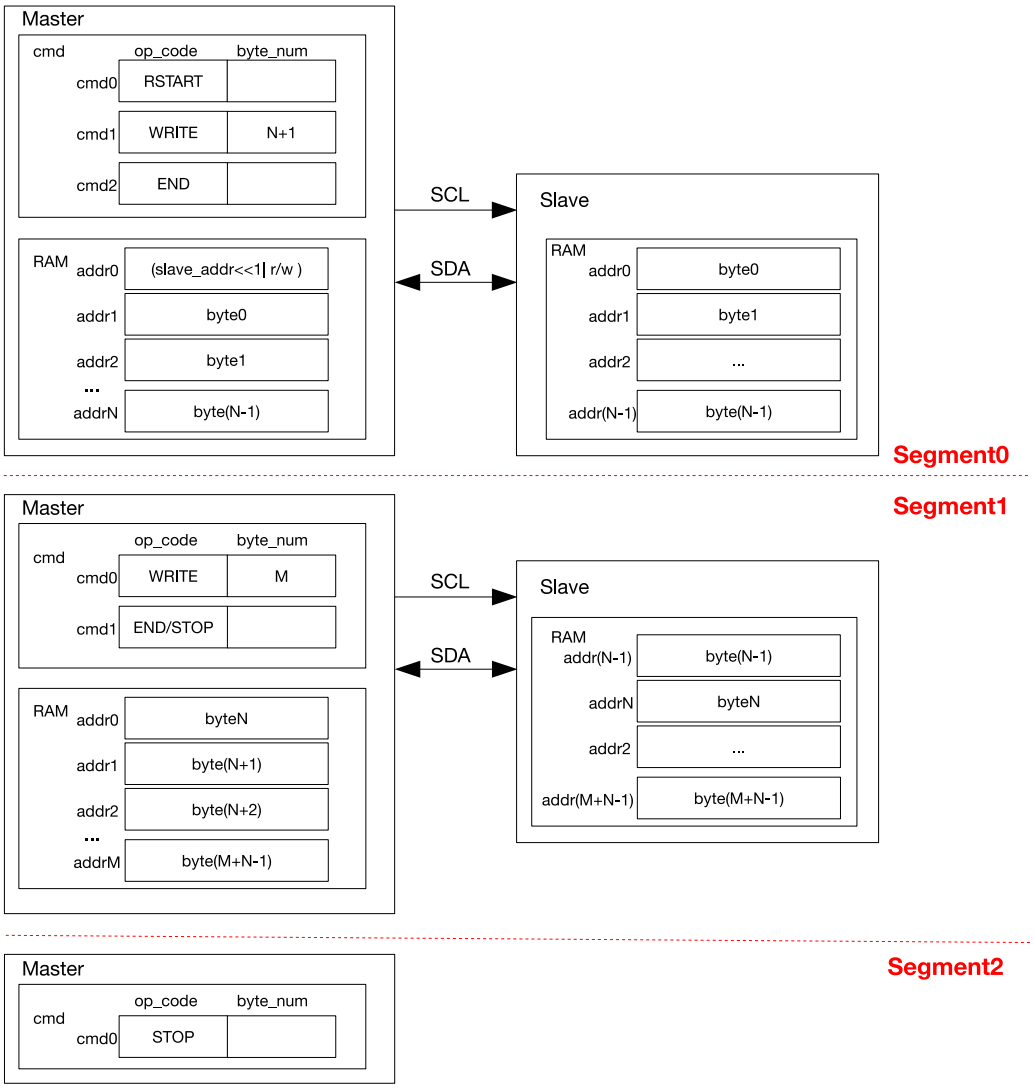


Figure 32.6-4. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address in Multiple Sequences

Given that the I2C Controller RAM holds only the size of TX/RX FIFO depth, when data are too large to be processed, it is advised to transmit them in multiple command sequences. At the end of every command sequence is an END command. When the controller executes this END command, SCL will be pulled low, and the software can refresh command sequence registers and the RAM for the next transfer.

Figure 32.6-4 shows how $I2C_{master}$ writes to an I2C slave in two or three segments as an example. For the first segment, the CMD_Controller registers are configured as shown in Segment0. Once data in $I2C_{master}$'s RAM is ready and I2C_TRANS_START is set, $I2C_{master}$ initiates data transfer. After executing the END command, $I2C_{master}$ turns off the SCL clock and pulls SCL low to reserve the bus. Meanwhile, the controller generates an I2C_END_DETECT_INT interrupt.

For the second segment, after detecting the I2C_END_DETECT_INT interrupt, the software refreshes the CMD_Controller registers, reloads the RAM, and clears this interrupt, as shown in Segment1. If cmd1 in the second segment is a STOP, then data is transmitted to $I2C_{slave}$ in two segments. $I2C_{master}$ resumes data transfer after I2C_TRANS_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C_END_DETECT_INT is detected, the

CMD_Controller registers of $I2C_{master}$ are configured as shown in Segment2. Once I2C_TRANS_START is set, $I2C_{master}$ generates a STOP bit and terminates the transfer.

Note that other $I2C_{master}$s will not transact on the bus between two segments. The bus is only released after a STOP command is sent. The I2C controller can be reset by setting I2C_FSM_RST field at any time. This field will later be cleared automatically by hardware.

32.6.4.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2 (master)	END	—	—	—	—

4. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
5. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register
6. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
7. Write 1 to I2C_TRANS_START (master) and I2C_TRANS_START (slave) to start transfer.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en (master).
10. After the I2C_END_DETECT_INT (master) interrupt is generated, set I2C_END_DETECT_INT_CLR (master) to 1 to clear this interrupt.
11. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1 (master)	END/STOP	—	—	—	—

12. Write M bytes of data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
13. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 9.
14. If the command is a STOP, I2C stops the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
15. If the command is an END, repeat step 10.
16. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

17. Write 1 to I2C_TRANS_START (master) bit to start transfer.
18. $I2C_{master}$ executes the STOP command and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence

32.6.5.1 Introduction

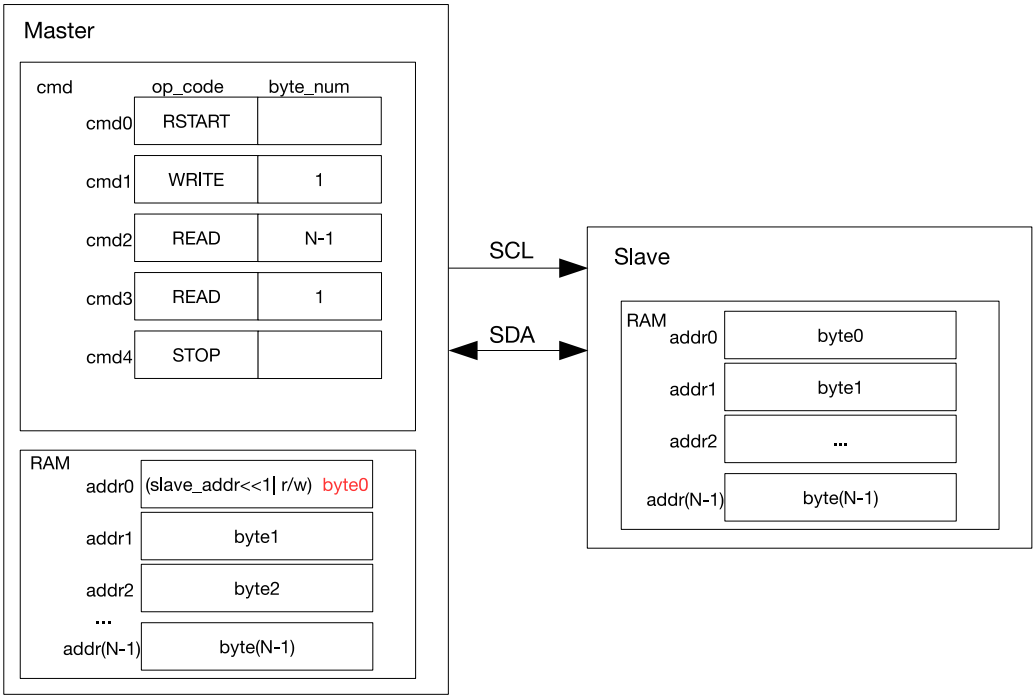


Figure 32.6-5. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address

Figure 32.6-5 shows how $I2C_{master}$ reads N bytes of data from an I2C slave using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed $I2C_{master}$ sends the address of $I2C_{slave}$. The byte sent comprises a 7-bit $I2C_{slave}$ address and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 1, it indicates a READ operation. If the

address of an I2C slave matches the sent address, this matching slave starts sending data to $I2C_{master}$. $I2C_{master}$ generates acknowledgments according to ack_value defined in the READ command upon receiving a byte.

As illustrated in Figure 32.6-5, $I2C_{master}$ executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd 3. This configuration may be changed as required. $I2C_{master}$ writes received data into the controller RAM from addr0, whose original content (the address of $I2C_{slave}$ and a $R/\overline{W}$ bit) is overwritten by byte0 marked red in Figure 32.6-5.

32.6.5.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to the scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N-1
I2C_COMMAND3 (master)	READ	1	0	1	1
I2C_COMMAND4 (master)	STOP	—	—	—	—

5. Write the address of $I2C_{slave}$ to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) bit to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 32.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.

- Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.

11. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
12. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
13. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
14. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
15. If data to be read by $I2C_{master}$ is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
16. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop the transfer once receiving the I2C_NACK_INT interrupt.
17. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence

32.6.6.1 Introduction

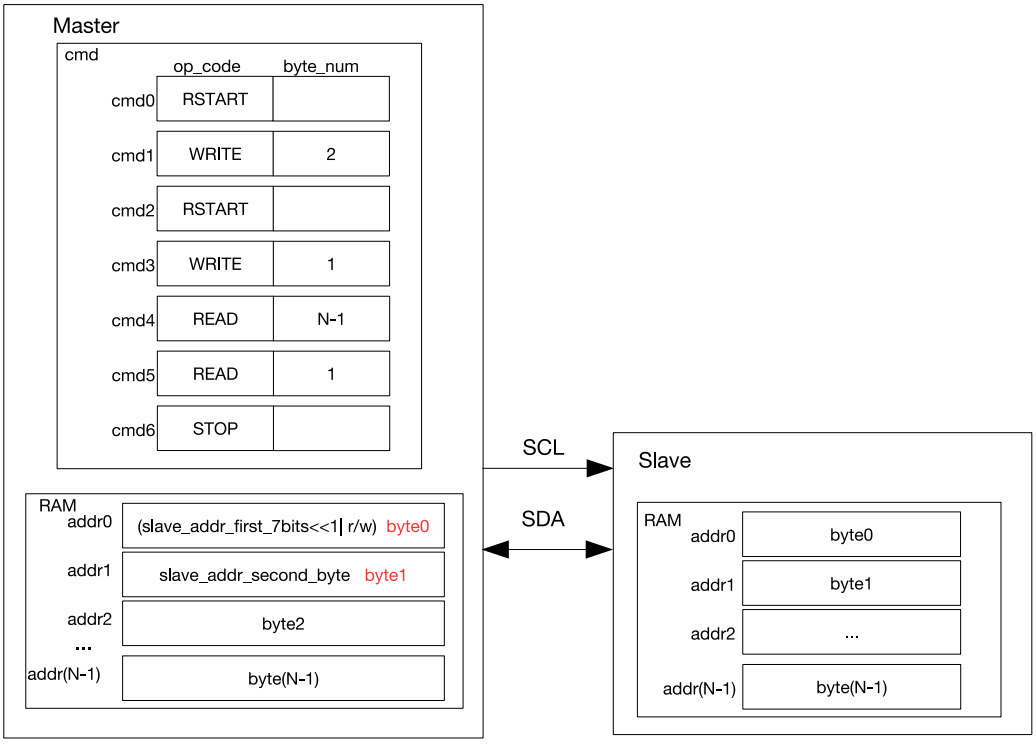


Figure 32.6-6. $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address

Figure 32.6-6 shows how $I2C_{master}$ reads data from an I2C slave using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the $I2C_{master}$ is formed from two bytes, and correspondingly TX RAM of this master stores a 10-bit address of two bytes. The $R/\overline{W}$ bit in the first byte is 0, which indicates a WRITE operation. After a RSTART condition, $I2C_{master}$ sends the first byte of address again to read data from $I2C_{slave}$, but the $R/\overline{W}$ bit is 1, which indicates a READ operation. The two address bytes can be configured as described in Section 32.6.2.

32.6.6.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2

I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

5. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) as $I2C_{slave}$'s 10-bit address, and set I2C_ADDR_10BIT_EN (slave) to 1 to enable 10-bit addressing.
6. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode. The first byte of address comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a WRITE operation. The second byte of address is I2C_SLAVE_ADDR[7:0]. The third byte is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ operation.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 32.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit that indicates READ.
12. $I2C_{slave}$ repeats step 10. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
13. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
14. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
15. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
16. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.

17. If data to be read by $I2C_{master}$ is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
18. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
19. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.7 $I2C_{master}$ Reads $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

32.6.7.1 Introduction

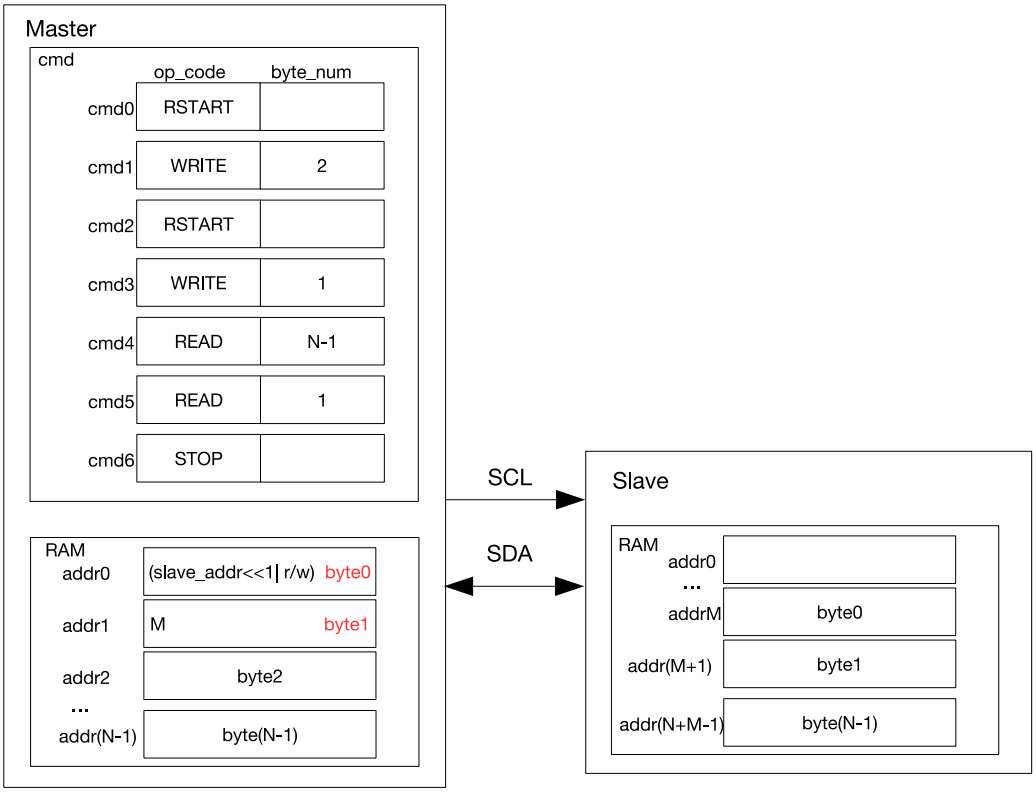


Figure 32.6-7. $I2C_{master}$ Reading N Bytes of Data from addrM of $I2C_{slave}$ with a 7-bit Address

Figure 32.6-7 shows how $I2C_{master}$ reads data from specified addresses in an I2C slave. $I2C_{master}$ sends two bytes of addresses: the first byte is a 7-bit $I2C_{slave}$ address followed by a $R/\overline{W}$ bit, which is 0 and indicates a WRITE; the second byte is $I2C_{slave}$'s memory address. After a RSTART condition, $I2C_{master}$ sends the first byte of address again, but the $R/\overline{W}$ bit is 1 which indicates a READ. Then, $I2C_{master}$ reads data starting from addrM.

32.6.7.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to the scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Set I2C_FIFO_ADDR_CFG_EN (slave) to 1 to enable dual address mode.
4. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
5. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	2
I2C_COMMAND2 (master)	RSTART	—	—	—	—
I2C_COMMAND3 (master)	WRITE	0	0	1	1
I2C_COMMAND4 (master)	READ	0	0	1	N-1
I2C_COMMAND5 (master)	READ	1	0	1	1
I2C_COMMAND6 (master)	STOP	—	—	—	—

6. Configure I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register as $I2C_{slave}$'s 7-bit address, and set I2C_ADDR_10BIT_EN (slave) to 0 to enable 7-bit addressing.
7. Write the address of $I2C_{slave}$ and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode according to Section 32.4.10. The first byte of address comprises (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 0 and indicates a WRITE. The second byte of the address is memory address M of $I2C_{slave}$. The third byte is (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ.
8. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
9. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
10. Start $I2C_{slave}$'s transfer according to Section 32.4.14.
11. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.

- Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.

12. $I2C_{slave}$ receives memory address sent by $I2C_{master}$ and adds the offset.
13. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a R bit.
14. $I2C_{slave}$ repeats step 11. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
15. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
16. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
17. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.
18. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
19. If data to be read by $I2C_{master}$ is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low, so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
20. After $I2C_{master}$ has received the last byte of data, set ack_value (master) to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
21. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.6.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

32.6.8.1 Introduction

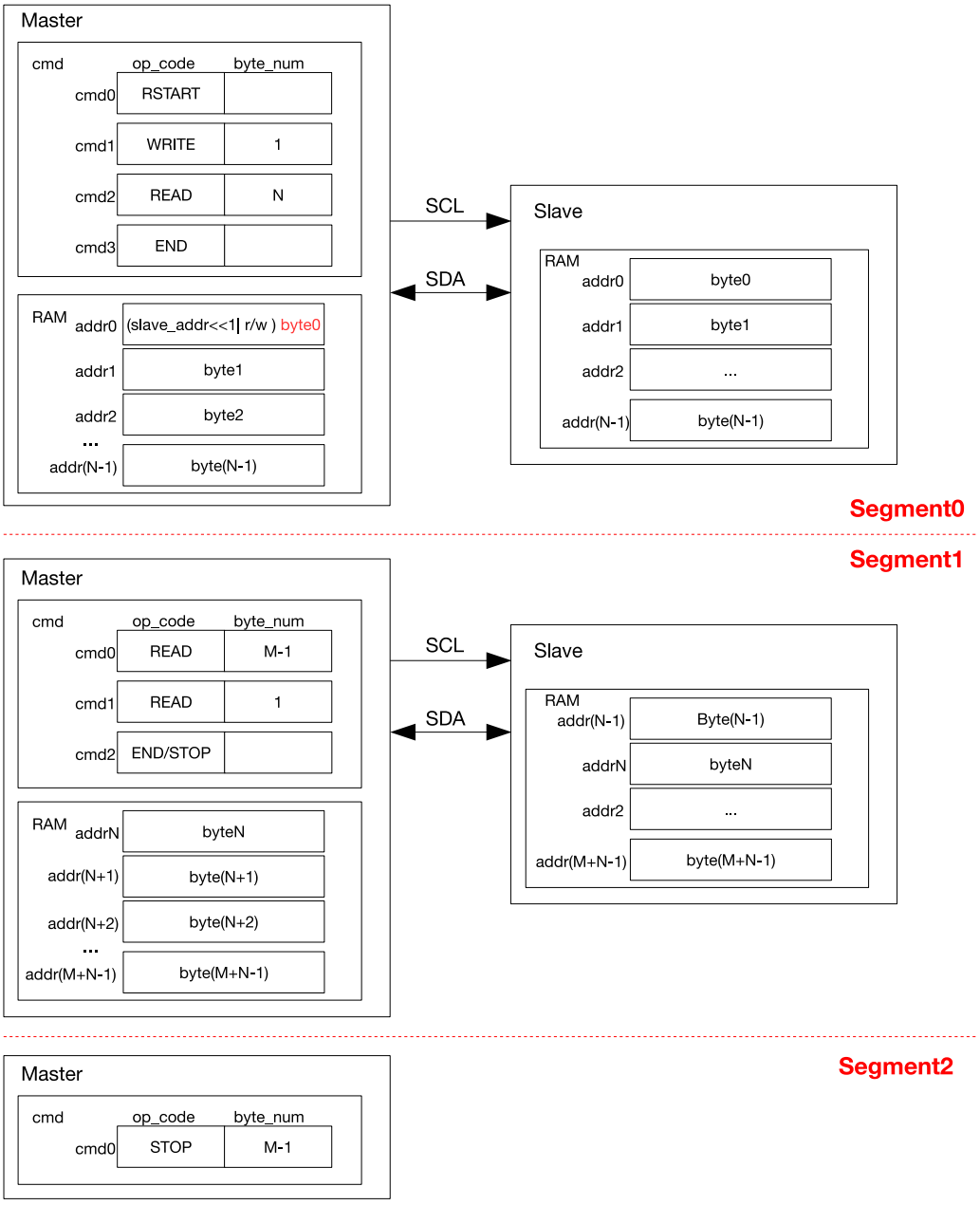


Figure 32.6-8. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments

Figure 32.6-8 shows how $I2C_{master}$ reads (N+M) bytes of data from an I2C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. The procedures for Segment0 is similar to 32.6-5, except that the last command is an END.
2. Prepare data in the TX RAM of $I2C_{slave}$, and set I2C_TRANS_START to start data transfer. After executing the END command, $I2C_{master}$ refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C_END_DETECT_INT interrupt. If cmd2 in Segment1 is a STOP, then data is read from $I2C_{slave}$ in two segments. $I2C_{master}$ resumes data transfer by setting I2C_TRANS_START and terminates the transfer by sending a STOP bit.
3. If cmd2 in Segment1 is an END, then data is read from $I2C_{slave}$ in three segments. After the second data

transfer finishes and an I2C_END_DETECT_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once I2C_TRANS_START is set, $I2C_{master}$ terminates the transfer by sending a STOP bit.

32.6.8.2 Configuration Example

1. Set I2C_MS_MODE (master) to 1, and I2C_MS_MODE (slave) to 0.
2. We recommend setting I2C_SLAVE_SCL_STRETCH_EN (slave) to 1, so that SCL can be held low for more processing time when $I2C_{slave}$ needs to send data. If this bit is not set, the software should write data to be sent to $I2C_{slave}$'s TX RAM before $I2C_{master}$ initiates the transfer. The configuration below is applicable to a scenario where I2C_SLAVE_SCL_STRETCH_EN (slave) is 1.
3. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	RSTART	—	—	—	—
I2C_COMMAND1 (master)	WRITE	0	0	1	1
I2C_COMMAND2 (master)	READ	0	0	1	N
I2C_COMMAND3 (master)	END	—	—	—	—

5. Write the address of $I2C_{slave}$ to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Write the address of $I2C_{slave}$ to I2C_SLAVE_ADDR (slave) in I2C_SLAVE_ADDR_REG (slave) register.
7. Write 1 to I2C_CONF_UPGATE (master) and I2C_CONF_UPGATE (slave) to synchronize registers.
8. Write 1 to I2C_TRANS_START (master) to start $I2C_{master}$'s transfer.
9. Start $I2C_{slave}$'s transfer according to Section 32.4.14.
10. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address in I2C_SLAVE_ADDR (slave). When ack_check_en (master) in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en (master) is 0, $I2C_{master}$ does not check the ACK value and takes $I2C_{slave}$ as a matching slave by default.
    - Match: If the received ACK value matches ack_exp (master) (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT (master) interrupt and stops data transfer.
11. After I2C_SLAVE_STRETCH_INT (slave) is generated, the I2C_STRETCH_CAUSE bit is 0. The address of $I2C_{slave}$ matches the address sent over SDA, and $I2C_{slave}$ needs to send data.
12. Write data to be sent to TX RAM of $I2C_{slave}$ in either FIFO mode or non-FIFO mode according to Section 32.4.10.
13. Set I2C_SLAVE_SCL_STRETCH_CLR (slave) to 1 to release SCL.

14. $I2C_{slave}$ sends data, and $I2C_{master}$ checks ACK value or not according to ack_check_en (master) in the READ command.
15. If data to be read by $I2C_{master}$ in one READ command (N or M) is larger than the TX FIFO depth of $I2C_{slave}$, an I2C_SLAVE_STRETCH_INT (slave) interrupt will be generated when TX RAM of $I2C_{slave}$ becomes empty. In this way, $I2C_{slave}$ can hold SCL low so that software has more time to pad data in TX RAM of $I2C_{slave}$ and read data in RX RAM of $I2C_{master}$. After the software has finished reading, you can set I2C_SLAVE_STRETCH_INT_CLR (slave) to 1 to clear interrupt, and set I2C_SLAVE_SCL_STRETCH_CLR (slave) to release the SCL line.
16. Once finishing reading data in the first READ command, $I2C_{master}$ executes the END command and triggers an I2C_END_DETECT_INT (master) interrupt, which is cleared by setting I2C_END_DETECT_INT_CLR (master) to 1.
17. Update $I2C_{master}$'s command registers using one of the following two methods:

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	ack_value	ack_exp	1	M
I2C_COMMAND1 (master)	END	—	—	—	—

Or

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0 (master)	READ	0	0	1	M-1
I2C_COMMAND0 (master)	READ	1	0	1	1
I2C_COMMAND1 (master)	STOP	—	—	—	—

18. Write M bytes of data to be sent to TX RAM of $I2C_{slave}$. If M is larger than the TX FIFO depth, then repeat step 12 in FIFO or non-FIFO mode.
19. Write 1 to I2C_TRANS_START (master) bit to start the transfer and repeat step 14.
20. If the last command is a STOP, then set ack_value (master) to 1 after $I2C_{master}$ has received the last byte of data. $I2C_{slave}$ stops transfer upon the I2C_NACK_INT interrupt. $I2C_{master}$ executes the STOP command to stop the transfer and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.
21. If the last command is an END, then repeat step 16 and proceed on to the next steps.
22. Update $I2C_{master}$'s command registers.

Command registers of $I2C_{master}$	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1 (master)	STOP	—	—	—	—

23. Write 1 to I2C_TRANS_START (master) bit to start transfer.

24. $I2C_{master}$ executes the STOP command to stop transfer, and generates an I2C_TRANS_COMPLETE_INT (master) interrupt.

32.7 Interrupts

ESP32-P4's I2C*n* can generate the I2C*n*_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from I2C*n* that can generate the above interrupt signal(s) as follows:

- I2C_SLAVE_STRETCH_INT: Triggered when one of the four stretching events occurs in slave mode.
- I2C_DET_START_INT: Triggered when the master or the slave detects a START signal.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when the main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO_I2C[23:0] clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when the state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO_I2C[23:0] clock cycles.
- I2C_RXFIFO_UDF_INT: Triggered when the I2C controller reads RX FIFO via the APB bus, but RX FIFO is empty.
- I2C_TXFIFO_OVF_INT: Triggered when the I2C controller writes TX FIFO via the APB bus, but TX FIFO is full.
- I2C_NACK_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C_TRANS_START_INT: Triggered when the I2C controller sends a START bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than $2^{I2C\_TIME\_OUT\_VALUE}$ clock cycles during data transfer.
- I2C_TRANS_COMPLETE_INT: Triggered when the I2C controller detects a STOP bit.
- I2C_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_ARBITRATION_LOST_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.
- I2C_BYTE_TRANS_DONE_INT: Triggered when the I2C controller sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_RXFIFO_OVF_INT: Triggered when RX FIFO of the I2C controller overflows.
- I2C_TXFIFO_WM_INT: I2C TX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of TX FIFO are less than I2C_TXFIFO_WM_THRHD[4:0].
- I2C_RXFIFO_WM_INT: I2C RX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of RX FIFO are greater than I2C_RXFIFO_WM_THRHD[4:0].
- I2C_SLAVE_ADDR_UNMATCH_INT: Triggered when the received slave address is inconsistent with the internally configured slave address in slave mode.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 32.8.1 *I2C Register Summary*.

32.8 Register Summary

32.8.1 I2C Register Summary

The addresses in this section are relative to the I2C Controller base address provided in Table 5.3-2 in Chapter *5 System and Memory*.

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

Name	Description	Address	Access
Timing registers			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL Clock	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL rising edge for a stop condition Measurement unit: i2c_sclk	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x007C	R/W
Configuration registers			
I2C_CTR_REG	Transmission setting	0x0004	varies
I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
I2C_SLAVE_ADDR_REG	Local slave address setting	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
I2C_FILTER_CFG_REG	SCL and SDA filter configuration register	0x0050	R/W
I2C_SCL_SP_CONF_REG	Power configuration register	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	Set SCL stretch of I2C slave	0x0084	varies
Status registers			
I2C_SR_REG	Describe I2C work status	0x0008	RO
I2C_FIFO_ST_REG	FIFO status register	0x0014	RO
I2C_DATA_REG	Rx FIFO read data	0x001C	HRO
Interrupt registers			
I2C_INT_RAW_REG	Raw interrupt status	0x0020	R/SS/WTC
I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WT
I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W

Name	Description	Address	Access
I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Command registers			
I2C_COMD0_REG	I2C command register 0	0x0058	varies
I2C_COMD1_REG	I2C command register 1	0x005C	varies
I2C_COMD2_REG	I2C command register 2	0x0060	varies
I2C_COMD3_REG	I2C command register 3	0x0064	varies
I2C_COMD4_REG	I2C command register 4	0x0068	varies
I2C_COMD5_REG	I2C command register 5	0x006C	varies
I2C_COMD6_REG	I2C command register 6	0x0070	varies
I2C_COMD7_REG	I2C command register 7	0x0074	varies
Version register			
I2C_DATE_REG	Version register	0x00F8	R/W
Address register			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO base address register	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO base address register	0x0180	HRO

32.8.2 LP_I2C Register Summary

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

Name	Description	Address	Access
Timing Registers			
LP_I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL Clock	0x0000	R/W
LP_I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
LP_I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
LP_I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
LP_I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x0040	R/W
LP_I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
LP_I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
LP_I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x004C	R/W
LP_I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0078	R/W
LP_I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x007C	R/W
Configuration Registers			
LP_I2C_CTR_REG	Transmission setting	0x0004	varies
LP_I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
LP_I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
LP_I2C_FILTER_CFG_REG	SCL and SDA filter configuration register	0x0050	R/W
LP_I2C_SCL_SP_CONF_REG	Power configuration register	0x0080	varies

Name	Description	Address	Access
Status Registers			
LP_I2C_SR_REG	Describe I2C work status	0x0008	RO
LP_I2C_FIFO_ST_REG	FIFO status register	0x0014	RO
LP_I2C_DATA_REG	Rx FIFO read data	0x001C	RO
Interrupt Registers			
LP_I2C_INT_RAW_REG	Raw interrupt status	0x0020	R/SS/WTC
LP_I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WT
LP_I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W
LP_I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Command Registers			
LP_I2C_COMD0_REG	I2C command register 0	0x0058	varies
LP_I2C_COMD1_REG	I2C command register 1	0x005C	varies
LP_I2C_COMD2_REG	I2C command register 2	0x0060	varies
LP_I2C_COMD3_REG	I2C command register 3	0x0064	varies
LP_I2C_COMD4_REG	I2C command register 4	0x0068	varies
LP_I2C_COMD5_REG	I2C command register 5	0x006C	varies
LP_I2C_COMD6_REG	I2C command register 6	0x0070	varies
LP_I2C_COMD7_REG	I2C command register 7	0x0074	varies
Version Register			
LP_I2C_DATE_REG	Version register	0x00F8	R/W
Address Registers			
LP_I2C_TXFIFO_START_ADDR_REG	I2C TXFIFO base address register	0x0100	HRO
LP_I2C_RXFIFO_START_ADDR_REG	I2C RXFIFO base address register	0x0180	HRO

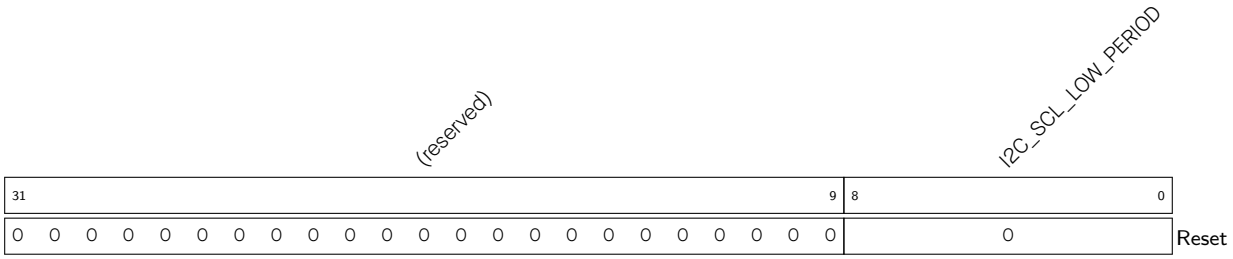
32.9 Registers

32.9.1 I2C Registers

The addresses in this section are relative to the I2C Controller base address provided in Table 5.3-2 in Chapter *5 System and Memory*.

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

Register 32.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

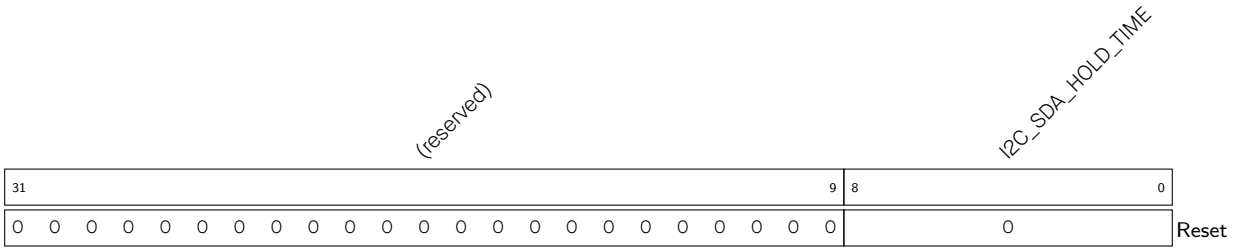


I2C_SCL_LOW_PERIOD Configures the low level width of the SCL Clock in master mode.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.2. I2C_SDA_HOLD_REG (0x0030)

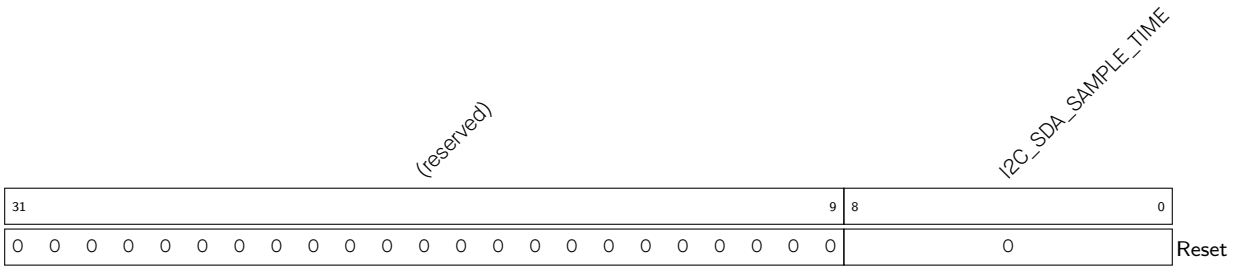


I2C_SDA_HOLD_TIME Configures the time to hold the data after the falling edge of SCL.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.3. I2C_SDA_SAMPLE_REG (0x0034)



I2C_SDA_SAMPLE_TIME Configures the time for sampling SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD Configures for how long SCL remains high in master mode.

Measurement unit: I2C_SCLK clock cycles

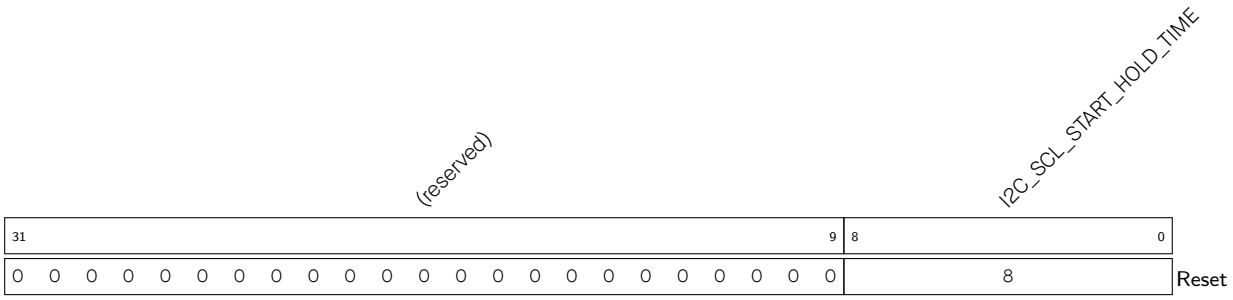
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD Configures the SCL_FSM's waiting period for SCL high level in master mode.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.5. I2C_SCL_START_HOLD_REG (0x0040)

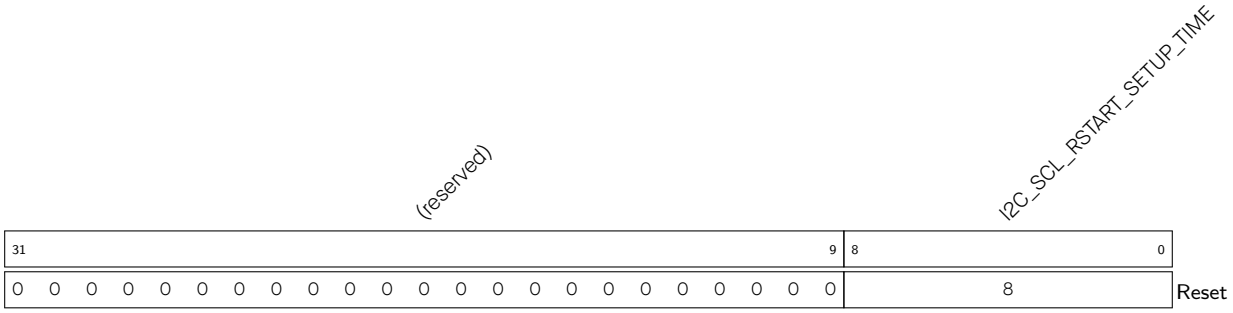


I2C_SCL_START_HOLD_TIME Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

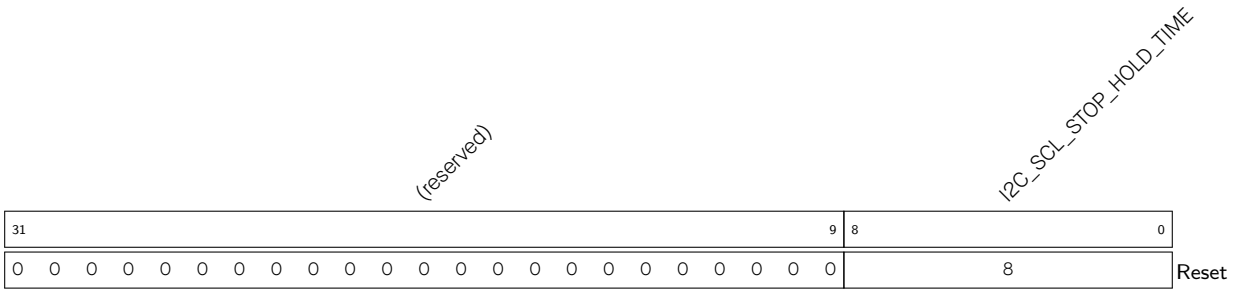


I2C_SCL_RSTART_SETUP_TIME Configures the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.7. I2C_SCL_STOP_HOLD_REG (0x0048)

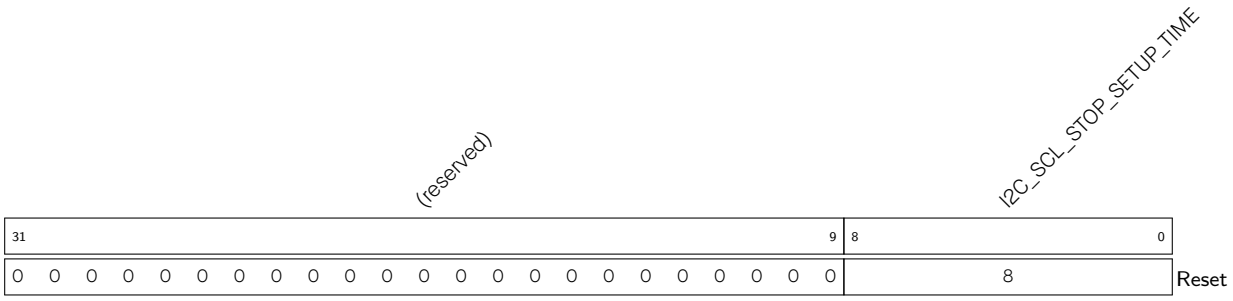


I2C_SCL_STOP_HOLD_TIME Configures the delay after the STOP condition.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.8. I2C_SCL_STOP_SETUP_REG (0x004C)

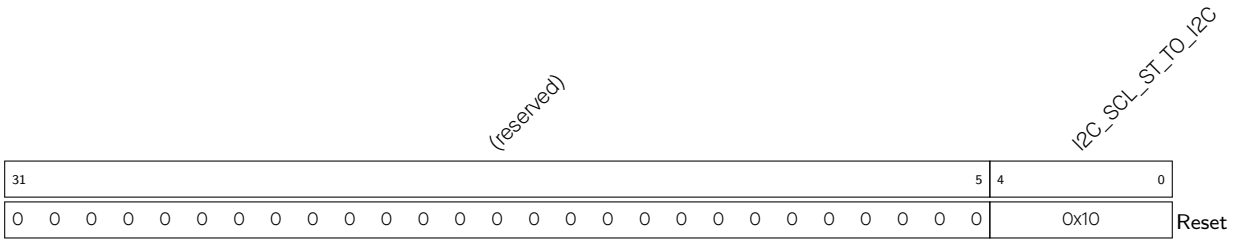


I2C_SCL_STOP_SETUP_TIME Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)

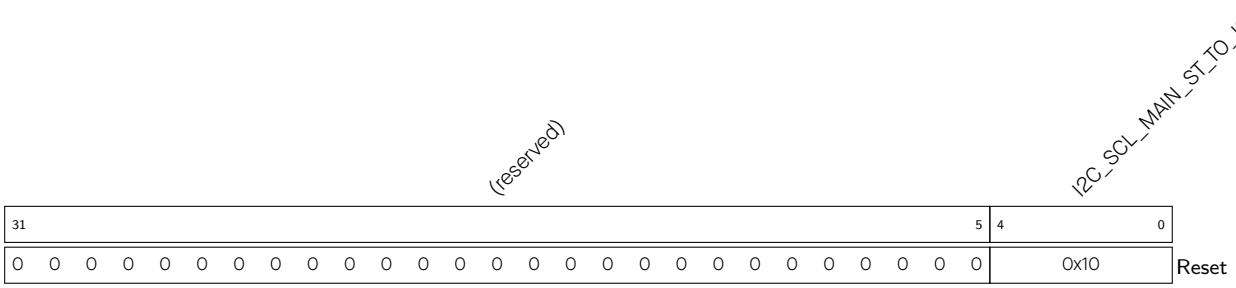


I2C_SCL_ST_TO_I2C Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

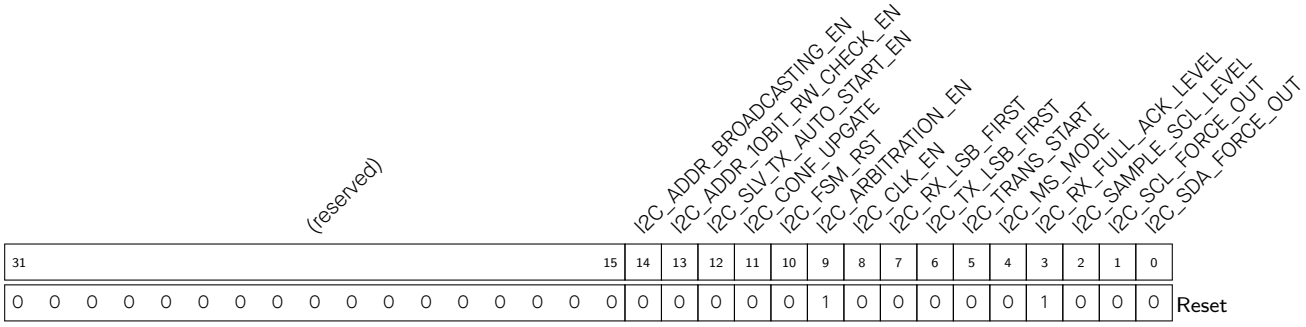


I2C_SCL_MAIN_ST_TO_I2C Configures the threshold value of SCL_MAIN_FSM state unchanged period. It should be no more than 23.

Measurement unit: I2C_SCLK clock cycles

(R/W)

Register 32.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT Configures the SDA output mode.

0: Open drain output

1: Direct output

(R/W)

I2C_SCL_FORCE_OUT Configures the SCL output mode.

0: Open drain output

1: Direct output

(R/W)

I2C_SAMPLE_SCL_LEVEL Configures the sample mode for SDA.

0: Sample SDA data on the SCL high level

1: Sample SDA data on the SCL low level

(R/W)

I2C_RX_FULL_ACK_LEVEL Configures the ACK value that needs to be sent by the master when the received RX RAM has reached the threshold.

(R/W)

I2C_MS_MODE Configures the module as an I2C Master or Slave.

0: Slave

1: Master

(R/W)

I2C_TRANS_START Configures whether the slave starts sending the data in TX FIFO.

0: No effect

1: Start

(WT)

I2C_TX_LSB_FIRST Configures to control the sending order for data needing to be sent.

0: send data from the most significant bit

1: send data from the least significant bit

(R/W)

I2C_RX_LSB_FIRST Configures to control the storage order for received data.

0: receive data from the most significant bit

1: receive data from the least significant bit

(R/W)

Continued on the next page...

Register 32.11. I2C_CTR_REG (0x0004)

Continued from the previous page...

I2C_CLK_EN Configures whether to gate clock signal for registers.

0: Support clock only when registers are read or written to by software

1: Force clock on for registers

(R/W)

I2C_ARBITRATION_EN Configures to enable I2C bus arbitration detection.

0: No effect

1: Enable

(R/W)

I2C_FSM_RST Configures to reset the SCL_FSM.

0: No effect

1: Reset (WT)

I2C_CONF_UPGATE Configures this bit for synchronization.

0: No effect

1: Synchronize (WT)

I2C_SLV_TX_AUTO_START_EN Configures to enable slave to send data automatically

0: Disable

1: Enable

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN Configures to check if the R/W bit of 10-bit addressing consists with I2C protocol.

0: Not check

1: Check (R/W)

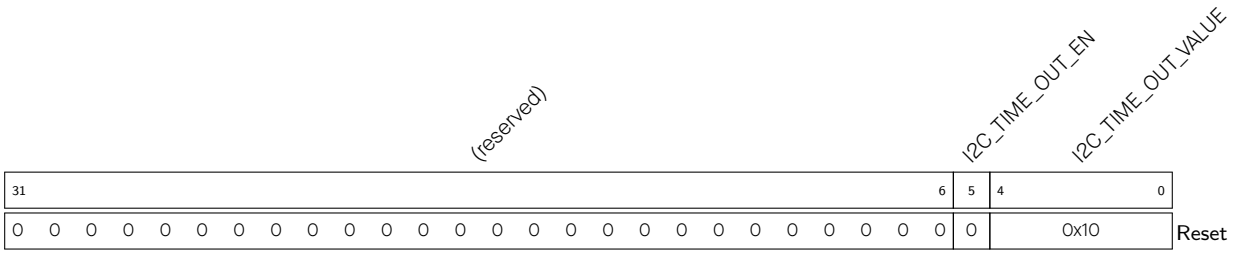
I2C_ADDR_BROADCASTING_EN Configures to support the 7-bit general call function.

0: Not support

1: Support

(R/W)

Register 32.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE Configures the timeout threshold period for SCL stuck at high or low level. The actual period is 2^(I2C_TIME_OUT_VALUE).

Measurement unit: I2C_SCLK clock cycles

(R/W)

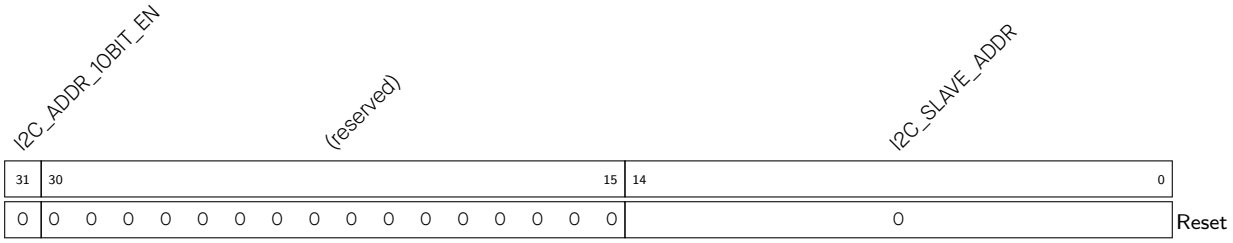
I2C_TIME_OUT_EN Configures to enable time out control.

0: No effect

1: Enable

(R/W)

Register 32.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR Configure the slave address of I2C Slave.

(R/W)

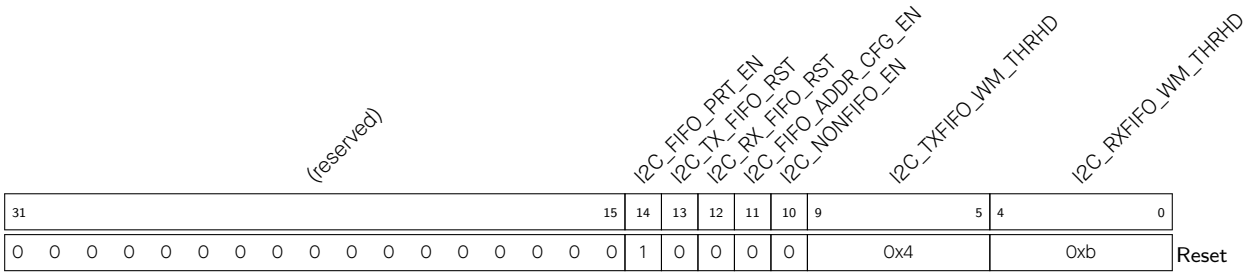
I2C_ADDR_10BIT_EN Configures to enable the slave 10-bit addressing mode in master mode.

0: No effect

1: Enable

(R/W)

Register 32.14. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD Configures the watermark threshold of RX FIFO in non-FIFO access mode. When I2C_FIFO_PRT_EN is 1 and RX FIFO counter is bigger than I2C_RXFIFO_WM_THRHD[4:0], I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_TXFIFO_WM_THRHD Configures the watermark threshold of TX FIFO in non-FIFO access mode. When I2C_FIFO_PRT_EN is 1 and TC FIFO counter is bigger than I2C_TXFIFO_WM_THRHD[4:0], I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_NONFIFO_EN Configures to enable APB non-FIFO access. (R/W)

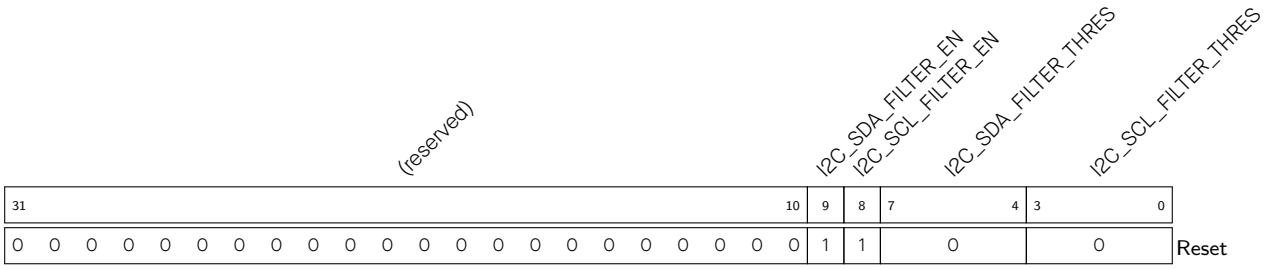
I2C_FIFO_ADDR_CFG_EN Configures the slave to enable dual address mode. When this mode is enabled, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM.
0: Disable
1: Enable
(R/W)

I2C_RX_FIFO_RST Configures to reset RX FIFO.
0: No effect
1: Reset (R/W)

I2C_TX_FIFO_RST Configures to reset TX FIFO.
0: No effect
1: Reset (R/W)

I2C_FIFO_PRT_EN Configures to enable FIFO pointer in non-FIFO access mode. This bit controls the valid bits and the TX/RX FIFO overflow, underflow, full and empty interrupts.
0: No effect
1: Enable
(R/W)

Register 32.15. I2C_FILTER_CFG_REG (0x0050)



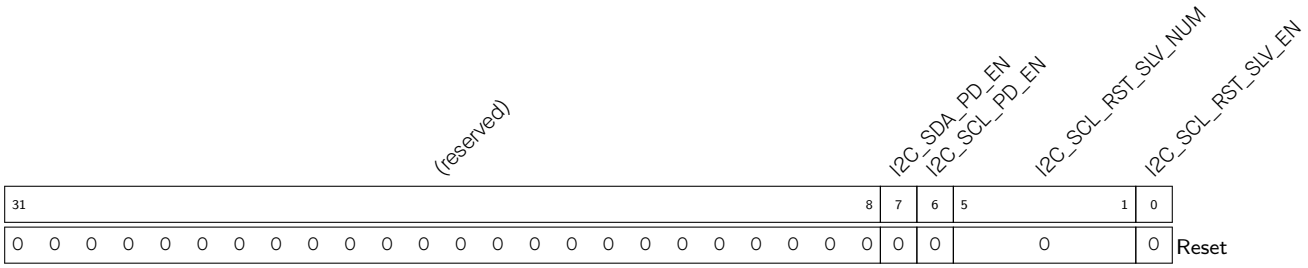
I2C_SCL_FILTER_THRES Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK clock cycles
(R/W)

I2C_SDA_FILTER_THRES Configures the threshold pulse width to be filtered on SDA. When a pulse on the SDA input has smaller width than this register value, the I2C controller will ignore that pulse. Measurement unit: I2C_SCLK clock cycles
(R/W)

I2C_SCL_FILTER_EN Configures to enable the filter function for SCL.
0: No effect
1: Enable
(R/W)

I2C_SDA_FILTER_EN Configures to enable the filter function for SDA.
0: No effect
1: Enable
(R/W)

Register 32.16. I2C_SCL_SP_CONF_REG (0x0080)



I2C_SCL_RST_SLV_EN Configures to send out SCL pulses when I2C master is IDLE. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0].

0: Invalid

1: Send out SCL pulses

(R/W/SC)

I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode.

Valid when I2C_SCL_RST_SLV_EN is 1.

Measurement unit: I2C_SCLK clock cycles

(R/W)

I2C_SCL_PD_EN Configures to power down the I2C output SCL line.

0: Not power down.

1: Not work and power down.

Valid only when I2C_SCL_FORCE_OUT is 1. (R/W)

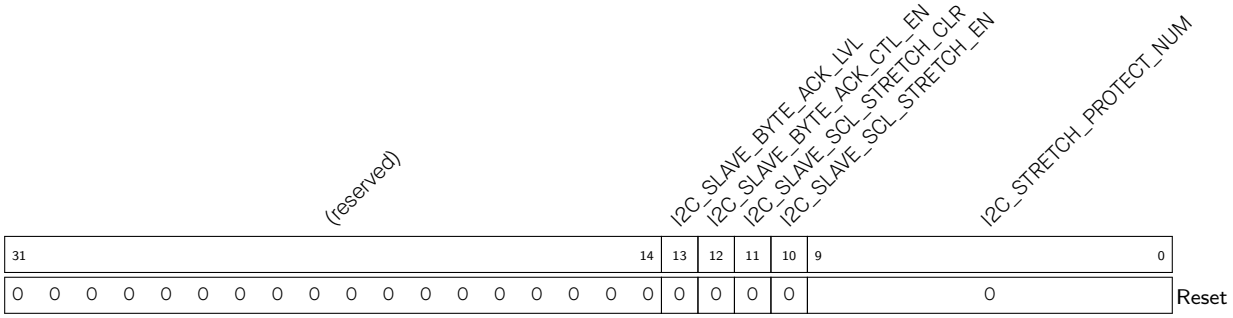
I2C_SDA_PD_EN Configures to power down the I2C output SDA line.

0: Not power down.

1: Not work and power down.

Valid only when I2C_SDA_FORCE_OUT is 1. (R/W)

Register 32.17. I2C_SCL_STRETCH_CONF_REG (0x0084)



I2C_STRETCH_PROTECT_NUM Configures the time period to release the SCL line from stretching to avoid timing violation. Usually it should be larger than the SDA setup time.
Measurement unit: I2C_SCLK clock cycles
(R/W)

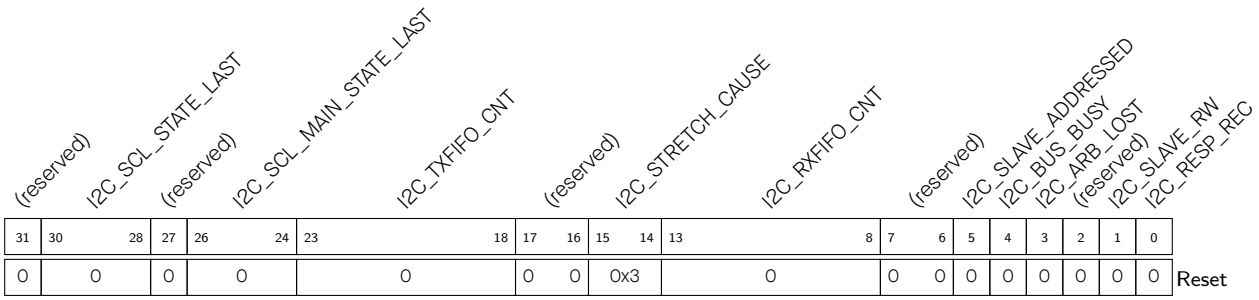
I2C_SLAVE_SCL_STRETCH_EN Configures to enable slave SCL stretch function. The SCL output line will be stretched low when I2C_SLAVE_SCL_STRETCH_EN is 1 and a stretch event happens. The stretch cause can be seen in I2C_STRETCH_CAUSE.
0: Disable
1: Enable
(R/W)

I2C_SLAVE_SCL_STRETCH_CLR Configures to clear the I2C slave SCL stretch function.
0: No effect
1: Clear
(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN Configures to enable the function for the slave to control ACK level.
0: Disable
1: Enable
(R/W)

I2C_SLAVE_BYTE_ACK_LVL Configures the ACK level when slave controlling ACK level function is enabled.
0: Low level
1: High level
(R/W)

Register 32.18. I2C_SR_REG (0x0008)



I2C_RESP_REC Represents the received ACK value in master mode or slave mode.

0: ACK

1: NACK. (RO)

I2C_SLAVE_RW Represents the transfer direction in slave mode.

0: Master writes to slave. 1: Master reads from slave

(RO)

I2C_ARB_LOST Represents whether the I2C controller loses control of SCL line.

0: No arbitration lost

1: Arbitration lost

(RO)

I2C_BUS_BUSY Represents the I2C bus state.

0: The I2C bus is in an idle state.

1: The I2C bus is busy transferring data

(RO)

I2C_SLAVE_ADDRESSED Represents whether the address sent by the master is equal to the address of the slave.

Valid only when the module is configured as an I2C Slave.

0: Not equal

1: Equal

(RO)

I2C_RXFIFO_CNT Represents the number of data bytes received in RAM. (RO)

I2C_STRETCH_CAUSE Represents the cause of SCL clocking stretching in slave mode.

0: Stretching SCL low when the master starts to read data.

1: Stretching SCL low when I2C TX FIFO is empty in slave mode.

2: Stretching SCL low when I2C RX FIFO is full in slave mode.

3: Invalid

(RO)

I2C_TXFIFO_CNT Represents the number of data bytes to be sent. (RO)

Continued on the next page...

Register 32.18. I2C_SR_REG (0x0008)

Continued from the previous page...

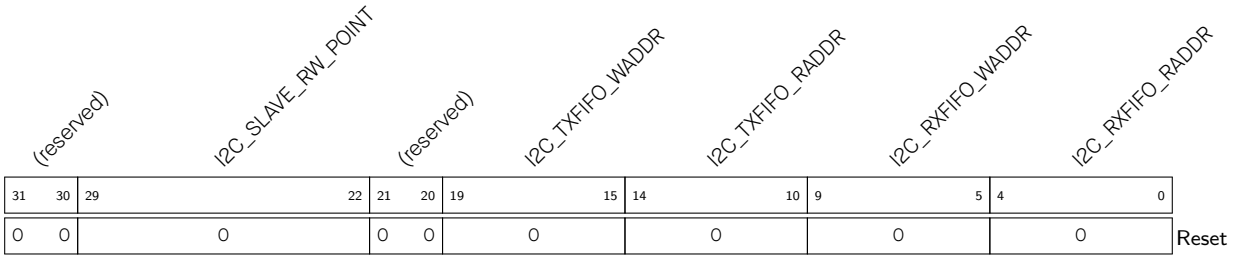
I2C_SCL_MAIN_STATE_LAST Represents the states of the I2C module state machine.

0: Idle
1: Address shift
2: ACK address
3: Rx data
4: Tx data
5: Send ACK
6: Wait ACK
7: Invalid
(RO)

I2C_SCL_STATE_LAST Represents the states of the state machine used to produce SCL.

0: Idle
1: Start
2: Negative edge
3: Low
4: Positive edge
5: High
6: Stop
7: Invalid
(RO)

Register 32.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR Represents the offset address of the APB reading from RX FIFO. (RO)

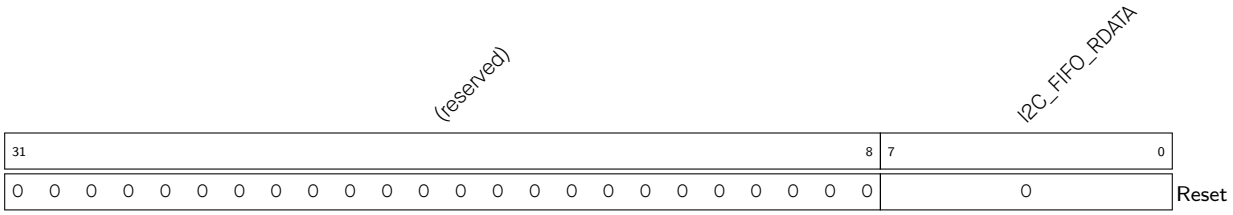
I2C_RXFIFO_WADDR Represents the offset address of the I2C module receiving data and writing to RX FIFO. (RO)

I2C_TXFIFO_RADDR Represents the offset address of the I2C module reading from TX FIFO. (RO)

I2C_TXFIFO_WADDR Represents the offset address of the APB bus writing to TX FIFO. (RO)

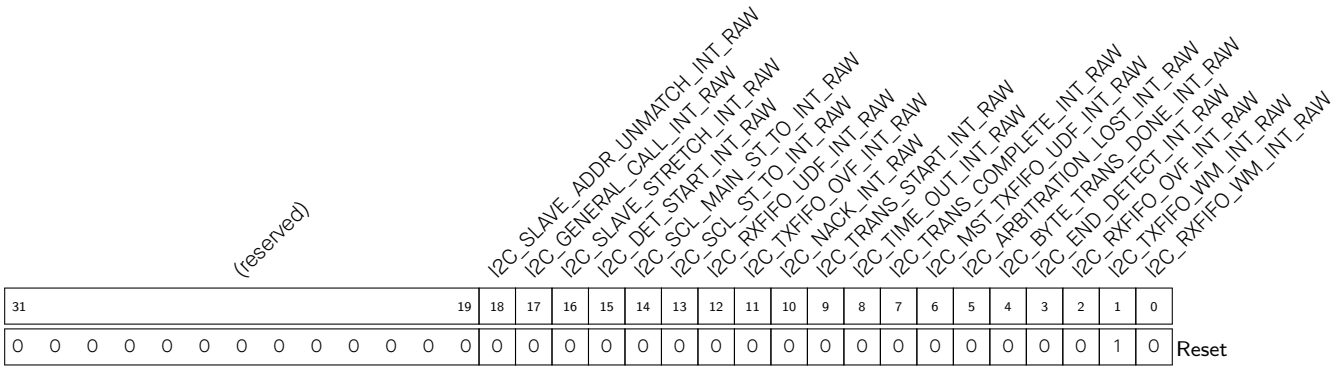
I2C_SLAVE_RW_POINT Represents the offset address in the I2C Slave RAM addressed by I2C Master when in I2C slave mode. (RO)

Register 32.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA Represents the value of RX FIFO read data. (RO)

Register 32.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW The raw interrupt status of I2C_RXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW The raw interrupt status of I2C_TXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt status of I2C_RXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_END_DETECT_INT_RAW The raw interrupt status of the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt status of the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status of the I2C_ARBITRATION_LOST_INT interrupt. (R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt status of I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status of the I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

Continued on the next page...

Register 32.21. I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

I2C_TIME_OUT_INT_RAW The raw interrupt status of the I2C_TIME_OUT_INT interrupt. (R/SS/WTC)

I2C_TRANS_START_INT_RAW The raw interrupt status of the I2C_TRANS_START_INT interrupt. (R/SS/WTC)

I2C_NACK_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt status of I2C_TXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt status of I2C_RXFIFO_UDF_INT interrupt. (R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_ST_TO_INT interrupt. (R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt status of I2C_SCL_MAIN_ST_TO_INT interrupt. (R/SS/WTC)

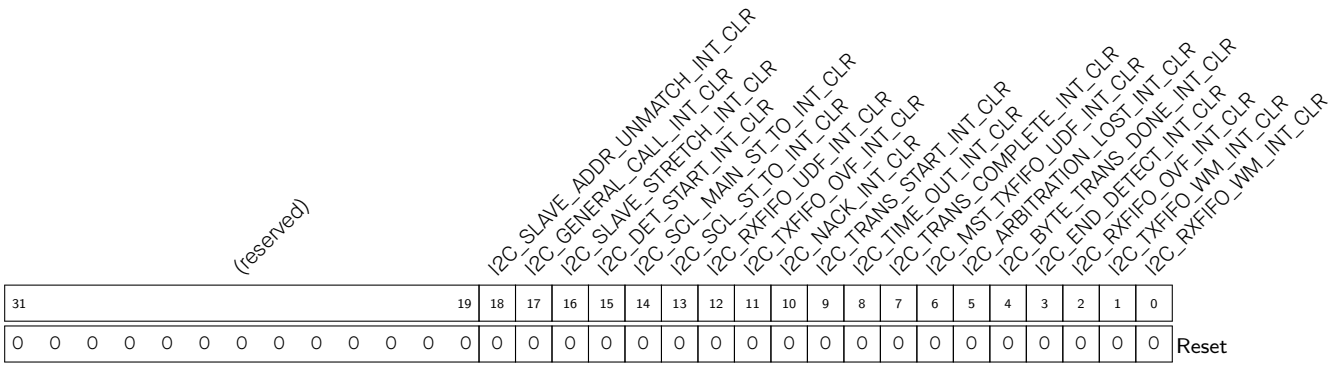
I2C_DET_START_INT_RAW The raw interrupt status of I2C_DET_START_INT interrupt. (R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW The raw interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW The raw interrupt status of I2C_GENARAL_CALL_INT interrupt. (R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW The raw interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT_RAW interrupt. (R/SS/WTC)

Register 32.22. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR Write 1 to clear I2C_RXFIFO_WM_INT interrupt. (WT)

I2C_TXFIFO_WM_INT_CLR Write 1 to clear I2C_TXFIFO_WM_INT interrupt. (WT)

I2C_RXFIFO_OVF_INT_CLR Write 1 to clear I2C_RXFIFO_OVF_INT interrupt. (WT)

I2C_END_DETECT_INT_CLR Write 1 to clear the I2C_END_DETECT_INT interrupt. (WT)

I2C_BYTE_TRANS_DONE_INT_CLR Write 1 to clear the I2C_BYTE_TRANS_DONE_INT interrupt. (WT)

I2C_ARBITRATION_LOST_INT_CLR Write 1 to clear the I2C_ARBITRATION_LOST_INT interrupt. (WT)

I2C_MST_TXFIFO_UDF_INT_CLR Write 1 to clear I2C_MST_TXFIFO_UDF_INT interrupt. (WT)

I2C_TRANS_COMPLETE_INT_CLR Write 1 to clear the I2C_TRANS_COMPLETE_INT interrupt. (WT)

I2C_TIME_OUT_INT_CLR Write 1 to clear the I2C_TIME_OUT_INT interrupt. (WT)

I2C_TRANS_START_INT_CLR Write 1 to clear the I2C_TRANS_START_INT interrupt. (WT)

I2C_NACK_INT_CLR Write 1 to clear I2C_NACK_INT interrupt. (WT)

I2C_TXFIFO_OVF_INT_CLR Write 1 to clear I2C_TXFIFO_OVF_INT interrupt. (WT)

I2C_RXFIFO_UDF_INT_CLR Write 1 to clear I2C_RXFIFO_UDF_INT interrupt. (WT)

I2C_SCL_ST_TO_INT_CLR Write 1 to clear I2C_SCL_ST_TO_INT interrupt. (WT)

I2C_SCL_MAIN_ST_TO_INT_CLR Write 1 to clear I2C_SCL_MAIN_ST_TO_INT interrupt. (WT)

I2C_DET_START_INT_CLR Write 1 to clear I2C_DET_START_INT interrupt. (WT)

I2C_SLAVE_STRETCH_INT_CLR Write 1 to clear I2C_SLAVE_STRETCH_INT interrupt. (WT)

I2C_GENERAL_CALL_INT_CLR Write 1 to clear I2C_GENARAL_CALL_INT interrupt. (WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR Write 1 to clear I2C_SLAVE_ADDR_UNMATCH_INT_RAW interrupt. (WT)

Register 32.23. I2C_INT_ENA_REG (0x0028)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ENA	0
17	I2C_GENERAL_CALL_INT_ENA	0
16	I2C_SLAVE_STRETCH_INT_ENA	0
15	I2C_DET_START_INT_ENA	0
14	I2C_SCL_MAIN_ST_TO_INT_ENA	0
13	I2C_SCL_ST_TO_INT_ENA	0
12	I2C_RXFIFO_UDF_INT_ENA	0
11	I2C_TXFIFO_OVF_INT_ENA	0
10	I2C_NACK_INT_ENA	0
9	I2C_TRANS_START_INT_ENA	0
8	I2C_TIME_OUT_INT_ENA	0
7	I2C_TRANS_COMPLETE_INT_ENA	0
6	I2C_MST_TXFIFO_UDF_INT_ENA	0
5	I2C_ARBITRATION_LOST_INT_ENA	0
4	I2C_BYTE_TRANS_DONE_INT_ENA	0
3	I2C_END_DETECT_INT_ENA	0
2	I2C_RXFIFO_OVF_INT_ENA	0
1	I2C_TXFIFO_WM_INT_ENA	0
0	I2C_RXFIFO_WM_INT_ENA	0

I2C_RXFIFO_WM_INT_ENA Write 1 to enable I2C_RXFIFO_WM_INT interrupt. (R/W)

I2C_TXFIFO_WM_INT_ENA Write 1 to enable I2C_TXFIFO_WM_INT interrupt. (R/W)

I2C_RXFIFO_OVF_INT_ENA Write 1 to enable I2C_RXFIFO_OVF_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA Write 1 to enable the I2C_END_DETECT_INT interrupt. (R/W)

I2C_BYTE_TRANS_DONE_INT_ENA Write 1 to enable the I2C_END_DETECT_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA Write 1 to enable the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_MST_TXFIFO_UDF_INT_ENA Write 1 to enable I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TRANS_COMPLETE_INT_ENA Write 1 to enable the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA Write 1 to enable the I2C_TIME_OUT_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA Write 1 to enable the I2C_TRANS_START_INT interrupt. (R/W)

I2C_NACK_INT_ENA Write 1 to enable I2C_SLAVE_STRETCH_INT interrupt. (R/W)

I2C_TXFIFO_OVF_INT_ENA Write 1 to enable I2C_TXFIFO_OVF_INT interrupt. (R/W)

I2C_RXFIFO_UDF_INT_ENA Write 1 to enable I2C_RXFIFO_UDF_INT interrupt. (R/W)

I2C_SCL_ST_TO_INT_ENA Write 1 to enable I2C_SCL_ST_TO_INT interrupt. (R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA Write 1 to enable I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

I2C_DET_START_INT_ENA Write 1 to enable I2C_DET_START_INT interrupt. (R/W)

I2C_SLAVE_STRETCH_INT_ENA Write 1 to enable I2C_SLAVE_STRETCH_INT interrupt. (R/W)

I2C_GENERAL_CALL_INT_ENA Write 1 to enable I2C_GENARAL_CALL_INT interrupt. (R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA Write 1 to enable I2C_SLAVE_ADDR_UNMATCH_INT interrupt. (R/W)

Register 32.24. I2C_INT_STATUS_REG (0x002C)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ST	0
17	I2C_GENERAL_CALL_INT_ST	0
16	I2C_SLAVE_STRETCH_INT_ST	0
15	I2C_DET_START_INT_ST	0
14	I2C_SCL_MAIN_ST_TO_INT_ST	0
13	I2C_SCL_ST_TO_INT_ST	0
12	I2C_RXFIFO_UDF_INT_ST	0
11	I2C_TXFIFO_OVF_INT_ST	0
10	I2C_NACK_INT_ST	0
9	I2C_TRANS_START_INT_ST	0
8	I2C_TIME_OUT_INT_ST	0
7	I2C_TRANS_COMPLETE_INT_ST	0
6	I2C_MST_TXFIFO_UDF_INT_ST	0
5	I2C_ARBITRATION_LOST_INT_ST	0
4	I2C_BYTE_TRANS_DONE_INT_ST	0
3	I2C_END_DETECT_INT_ST	0
2	I2C_RXFIFO_OVF_INT_ST	0
1	I2C_TXFIFO_WM_INT_ST	0
0	I2C_RXFIFO_WM_INT_ST	0

I2C_RXFIFO_WM_INT_ST The masked interrupt status of I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_ST The masked interrupt status of I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_ST The masked interrupt status of I2C_RXFIFO_OVF_INT interrupt. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status of the I2C_END_DETECT_INT interrupt. (RO)

I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status of the I2C_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status of the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status of I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_ST The masked interrupt status of the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status of the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status of the I2C_TRANS_START_INT interrupt. (RO)

I2C_NACK_INT_ST The masked interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_ST The masked interrupt status of I2C_TXFIFO_OVF_INT interrupt. (RO)

Continued on the next page...

Register 32.24. I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

I2C_RXFIFO_UDF_INT_ST The masked interrupt status of I2C_RXFIFO_UDF_INT interrupt. (RO)

I2C_SCL_ST_TO_INT_ST The masked interrupt status of I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status of I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

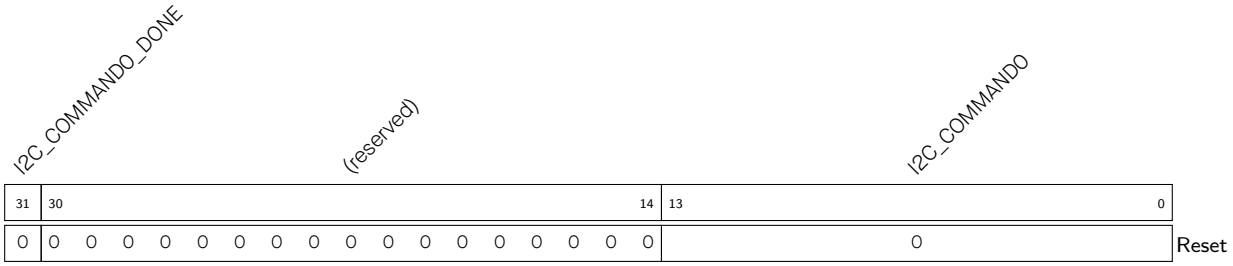
I2C_DET_START_INT_ST The masked interrupt status of I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_ST The masked interrupt status of I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_GENERAL_CALL_INT_ST The masked interrupt status of I2C_GENARAL_CALL_INT interrupt. (RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST The masked interrupt status of I2C_SLAVE_ADDR_UNMATCH_INT interrupt. (RO)

Register 32.25. I2C_COMD0_REG (0x0058)



I2C_COMMAND0 Configures command 0.

It consists of three parts:

op_code is the command

1: WRITE

2: STOP

3: READ

4: END

6: RSTART

Byte_num represents the number of bytes that need to be sent or received.

ack_check_en, ack_exp, and ack are used to control the ACK bit. See I2C cmd structure 32.4-2 for more information.

(R/W)

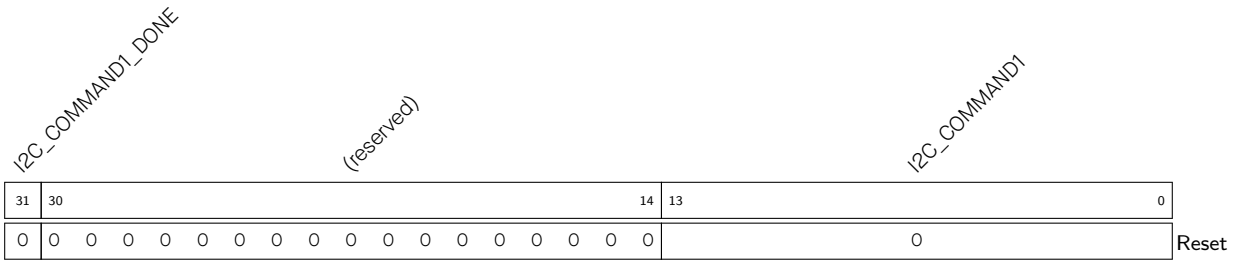
I2C_COMMAND0_DONE Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 Configures command 1.

See details in I2C_CMD0_REG[13:0]. (R/W)

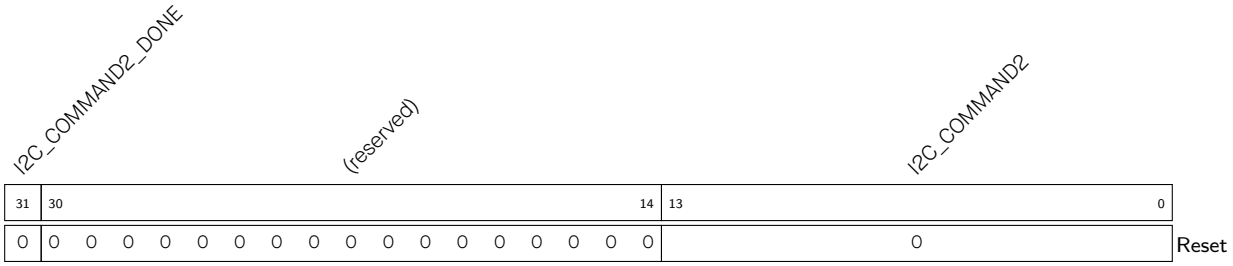
I2C_COMMAND1_DONE Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 Configures command 2. See details in I2C_CMD0_REG[13:0]. (R/W)

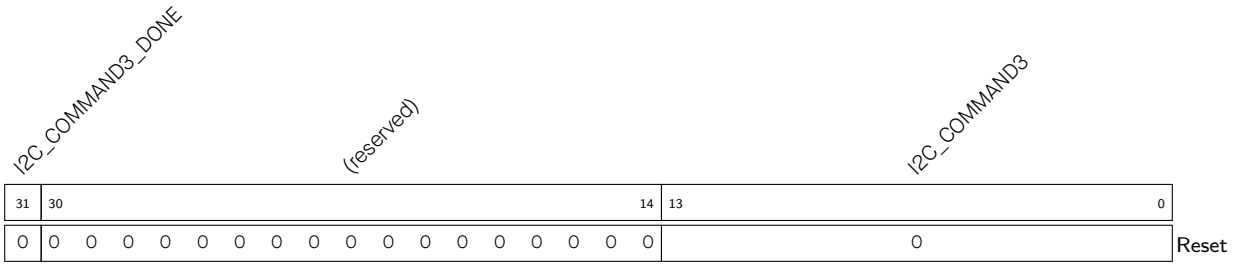
I2C_COMMAND2_DONE Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 Configures command 3. See details in I2C_CMD0_REG[13:0]. (R/W)

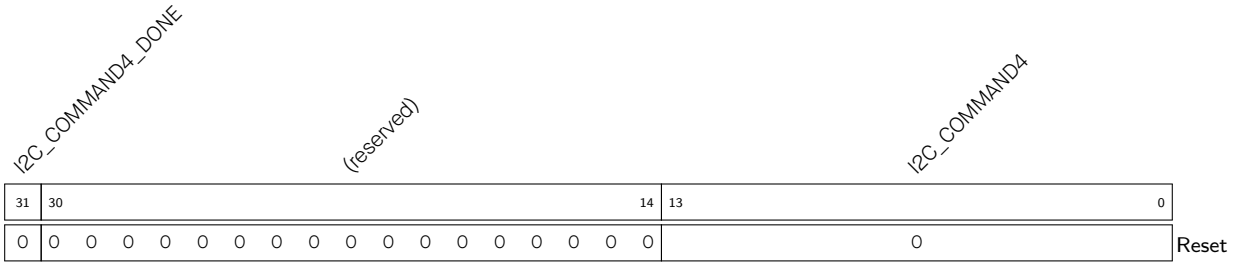
I2C_COMMAND3_DONE Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 Configures command 4. See details in I2C_CMD0_REG[13:0]. (R/W)

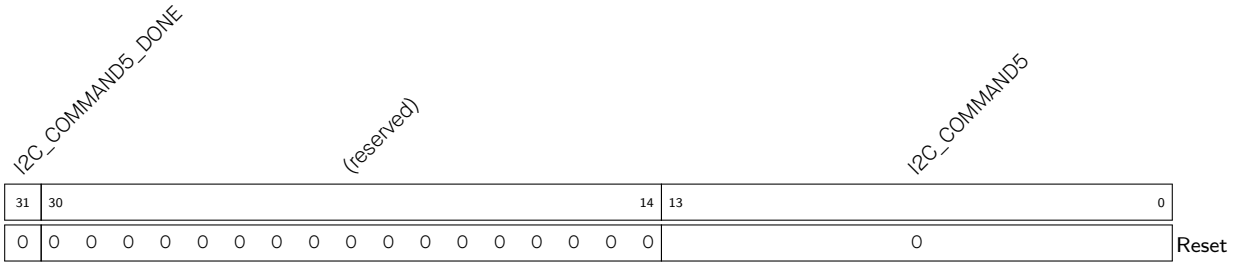
I2C_COMMAND4_DONE Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 Configures command 5. See details in I2C_CMD0_REG[13:0]. (R/W)

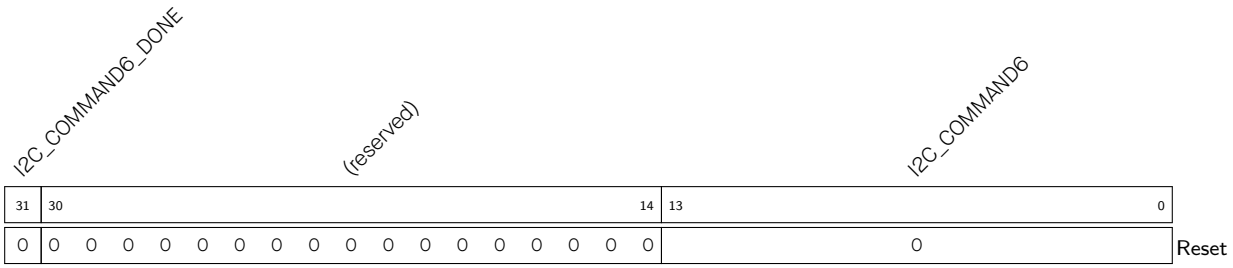
I2C_COMMAND5_DONE Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 Configures command 6. See details in I2C_CMD0_REG[13:0]. (R/W)

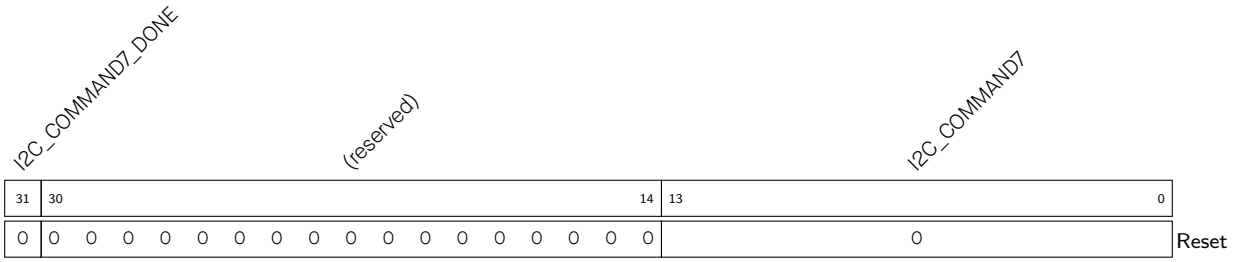
I2C_COMMAND6_DONE Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 Configures command 7. See details in I2C_CMD0_REG[13:0]. (R/W)

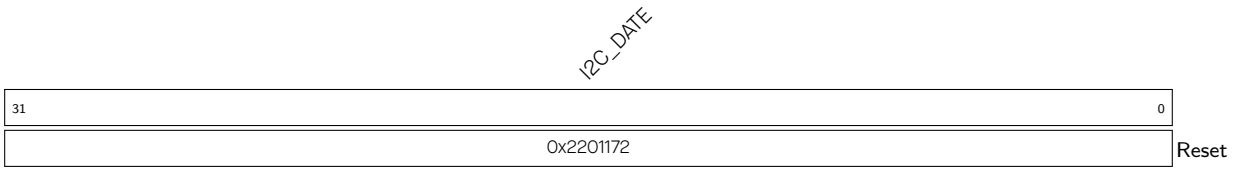
I2C_COMMAND7_DONE Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

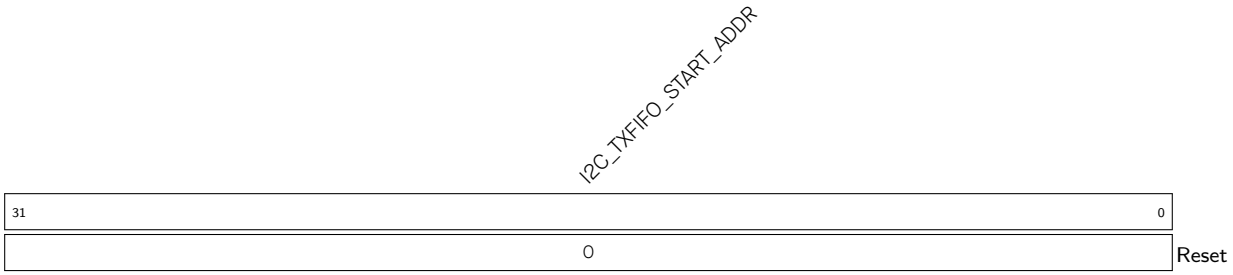
(R/W/SS)

Register 32.33. I2C_DATE_REG (0x00F8)



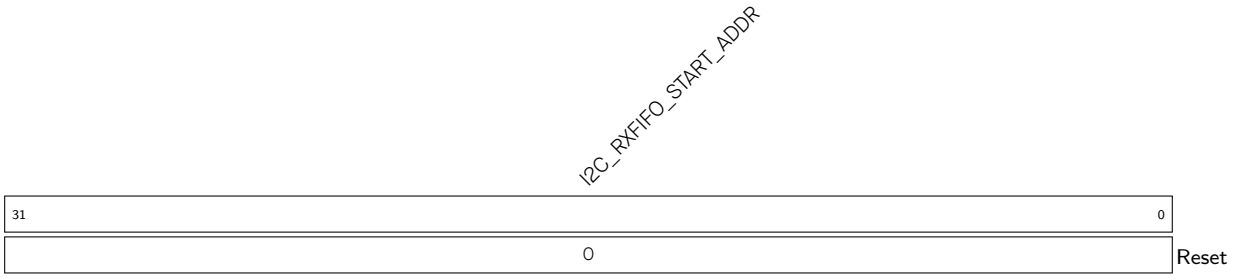
I2C_DATE Version control register. (R/W)

Register 32.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR Represents the I2C TX FIFO first address. (HRO)

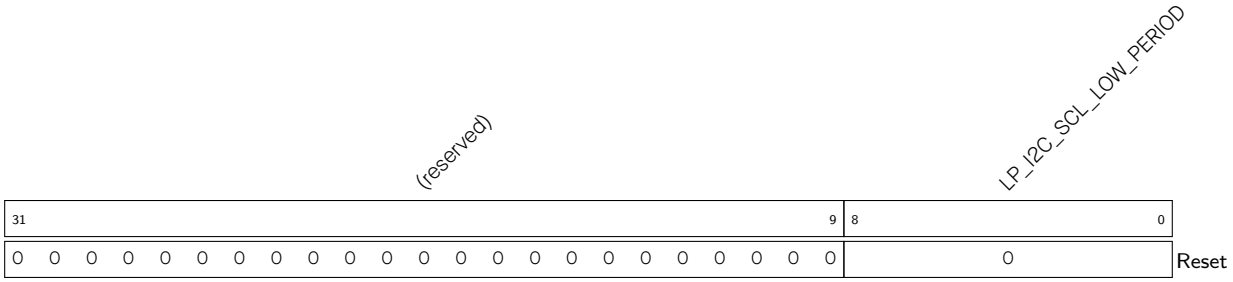
Register 32.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR Represents the I2C RX FIFO first address. (HRO)

32.9.2 LP_I2C Registers

Register 32.36. LP_I2C_SCL_LOW_PERIOD_REG (0x0000)

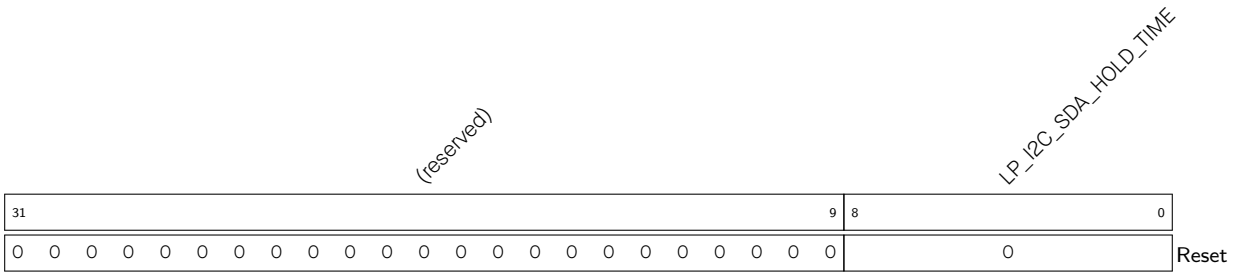


LP_I2C_SCL_LOW_PERIOD Configures the low level width of the SCL Clock in master mode.

Measurement unit: i2c_sclk

(R/W)

Register 32.37. LP_I2C_SDA_HOLD_REG (0x0030)

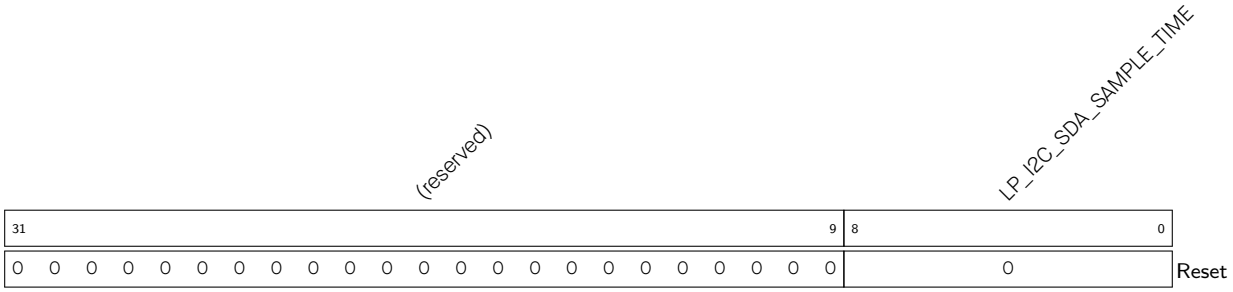


LP_I2C_SDA_HOLD_TIME Configures the time to hold the data after the falling edge of SCL.

Measurement unit: i2c_sclk

(R/W)

Register 32.38. LP_I2C_SDA_SAMPLE_REG (0x0034)

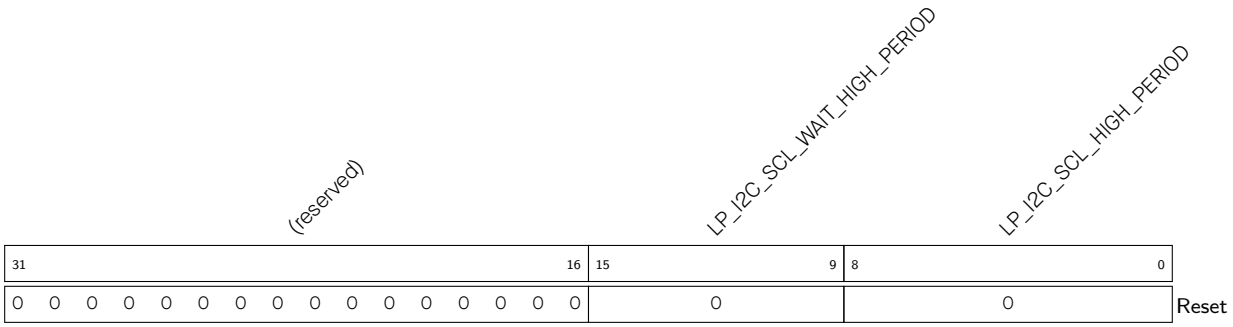


LP_I2C_SDA_SAMPLE_TIME Configures the time for sampling SDA.

Measurement unit: i2c_sclk

(R/W)

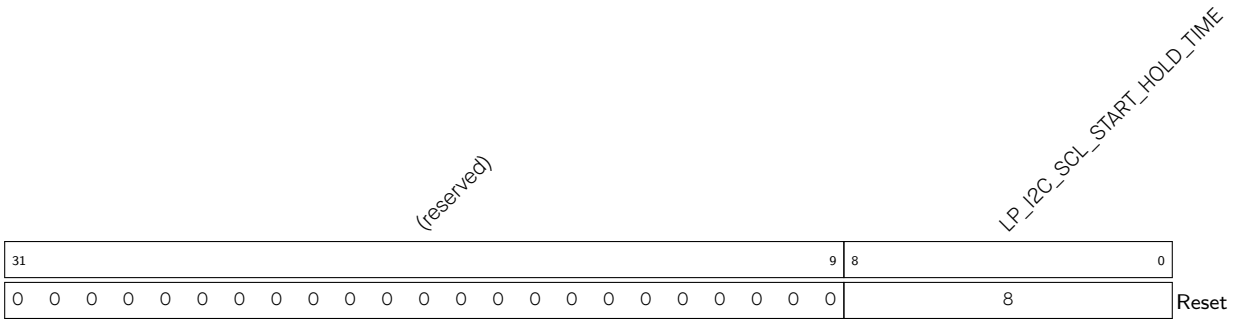
Register 32.39. LP_I2C_SCL_HIGH_PERIOD_REG (0x0038)



LP_I2C_SCL_HIGH_PERIOD Configures for how long SCL remains high in master mode.
Measurement unit: i2c_sclk
(R/W)

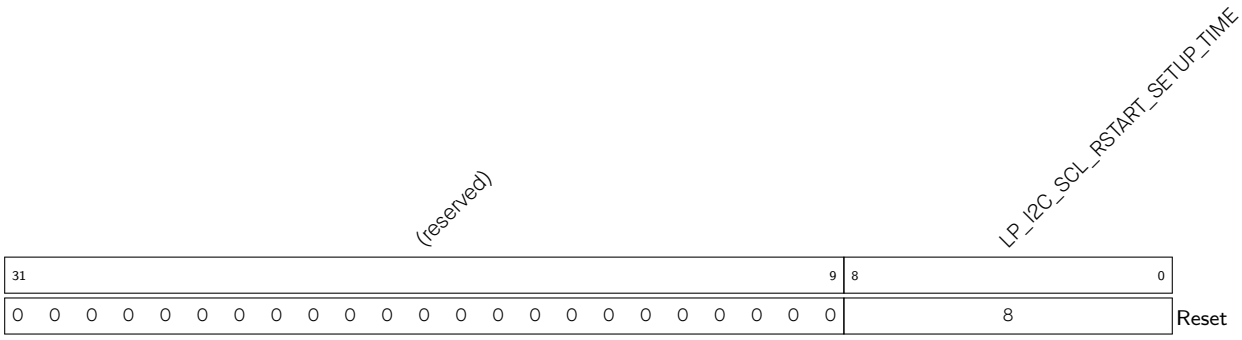
LP_I2C_SCL_WAIT_HIGH_PERIOD Configures the SCL_FSM's waiting period for SCL high level in master mode.
Measurement unit: i2c_sclk
(R/W)

Register 32.40. LP_I2C_SCL_START_HOLD_REG (0x0040)



LP_I2C_SCL_START_HOLD_TIME Configures the time between the falling edge of SDA and the falling edge of SCL for a START condition.
Measurement unit: i2c_sclk
(R/W)

Register 32.41. LP_I2C_SCL_RSTART_SETUP_REG (0x0044)

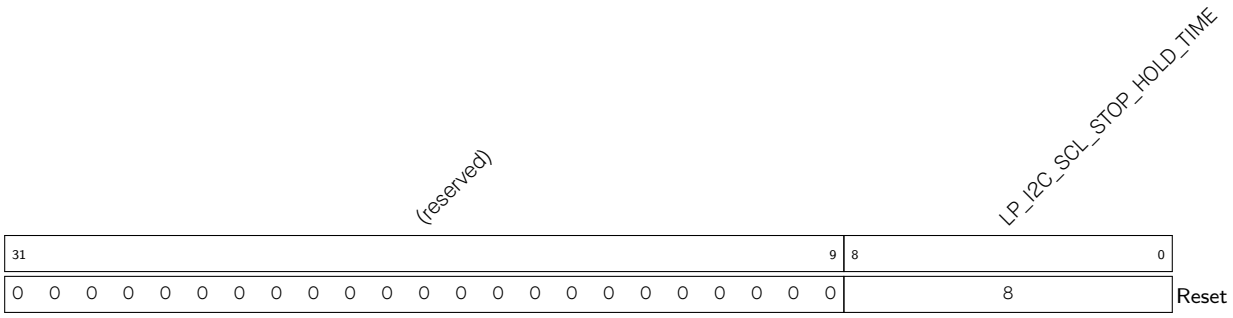


LP_I2C_SCL_RSTART_SETUP_TIME Configures the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition.

Measurement unit: i2c_sclk

(R/W)

Register 32.42. LP_I2C_SCL_STOP_HOLD_REG (0x0048)

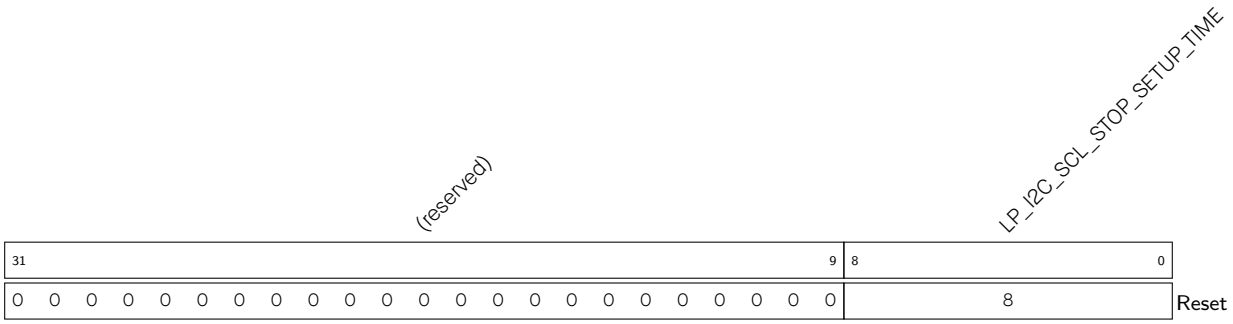


LP_I2C_SCL_STOP_HOLD_TIME Configures the delay after the STOP condition.

Measurement unit: i2c_sclk

(R/W)

Register 32.43. LP_I2C_SCL_STOP_SETUP_REG (0x004C)

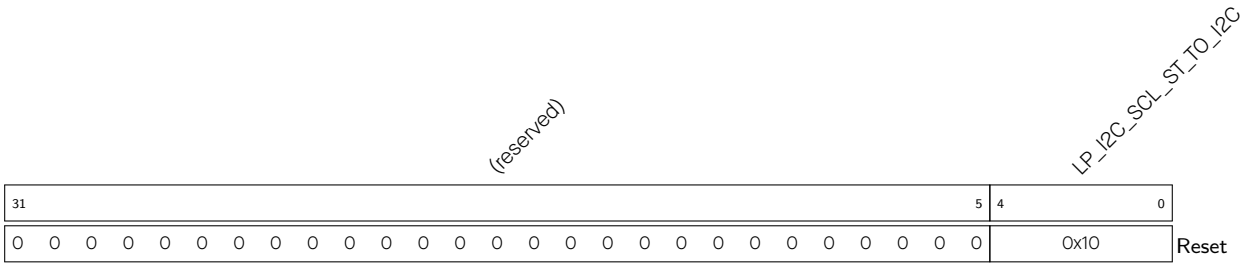


LP_I2C_SCL_STOP_SETUP_TIME Configures the time between the rising edge of SCL and the rising edge of SDA.

Measurement unit: i2c_sclk

(R/W)

Register 32.44. LP_I2C_SCL_ST_TIME_OUT_REG (0x0078)

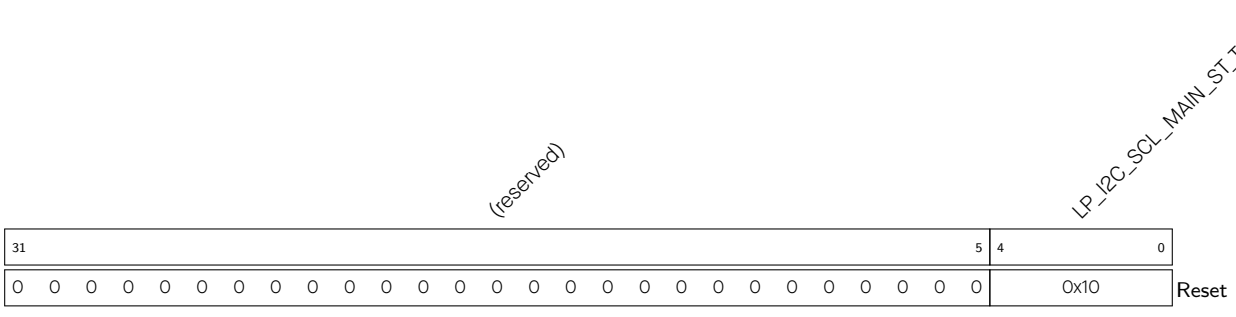


LP_I2C_SCL_ST_TO_I2C Configures the threshold value of SCL_FSM state unchanged period. It should be no more than 23.

Measurement unit: i2c_sclk

(R/W)

Register 32.45. LP_I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

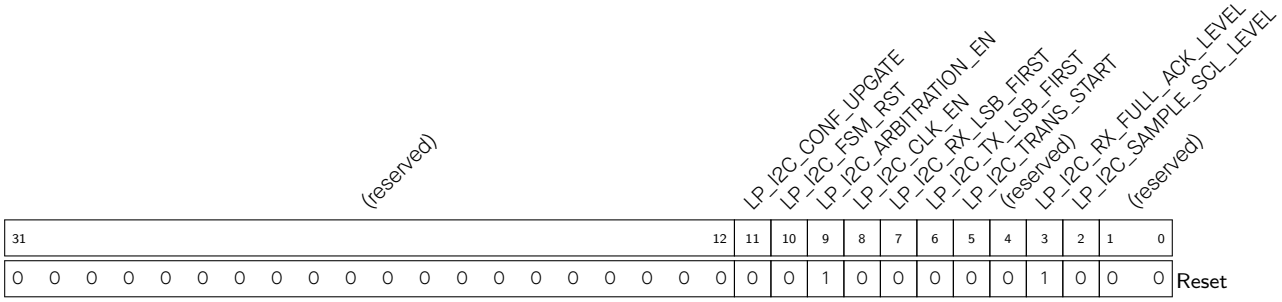


LP_I2C_SCL_MAIN_ST_TO_I2C Configures the threshold value of SCL_MAIN_FSM state unchanged period. It should be no more than 23.

Measurement unit: i2c_sclk

(R/W)

Register 32.46. LP_I2C_CTR_REG (0x0004)



LP_I2C_SAMPLE_SCL_LEVEL Configures the sample mode for SDA.

1: Sample SDA data on the SCL low level.

0: Sample SDA data on the SCL high level.

(R/W)

LP_I2C_RX_FULL_ACK_LEVEL Configures the ACK value that needs to be sent by master when the rx_fifo_cnt has reached the threshold. (R/W)

LP_I2C_TRANS_START Configures to start sending the data in txfifo for slave.

0: No effect

1: Start

(WT)

LP_I2C_TX_LSB_FIRST Configures to control the sending order for data to be sent.

1: send data from the least significant bit

0: send data from the most significant bit

(R/W)

LP_I2C_RX_LSB_FIRST Configures to control the storage order for received data.

1: receive data from the least significant bit

0: receive data from the most significant bit

(R/W)

LP_I2C_CLK_EN Configures whether to gate clock signal for registers.

0: Support clock only when registers are read or written to by software

1: Force clock on for registers.

(R/W)

LP_I2C_ARBITRATION_EN Configures to enable I2C bus arbitration detection.

0: No effect

1: Enable

(R/W)

LP_I2C_FSM_RST Configures to reset the SCL_FSM.

0: No effect

1: Reset

(WT)

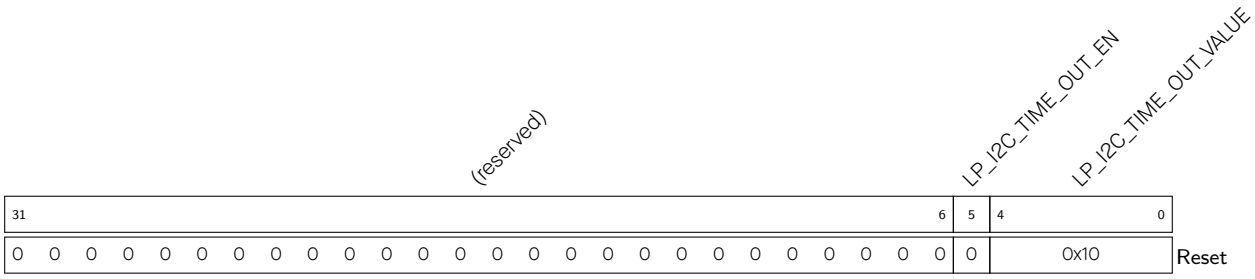
LP_I2C_CONF_UPGATE Configures this bit for synchronization.

0: No effect

1: Synchronize

(WT)

Register 32.47. LP_I2C_TO_REG (0x000C)



LP_I2C_TIME_OUT_VALUE Configures the timeout threshold period for SCL stucking at high or low level. The actual period is 2^(reg_time_out_value).

Measurement unit: i2c_sclk.

(R/W)

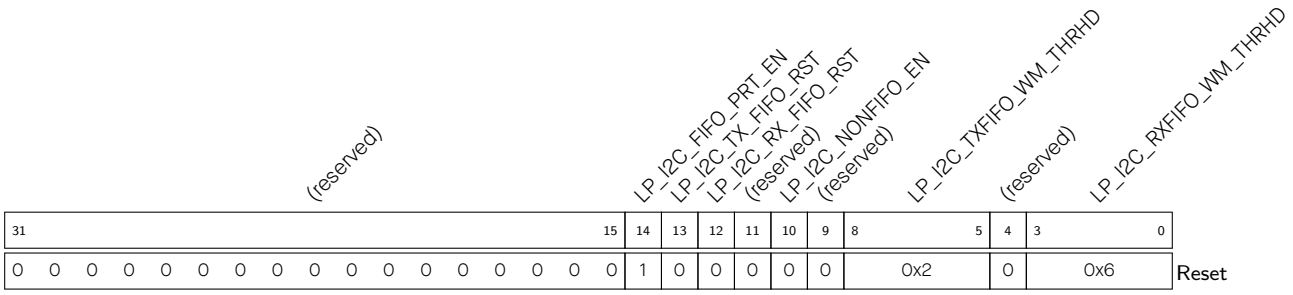
LP_I2C_TIME_OUT_EN Configures to enable time out control.

0: No effect

1: Enable

(R/W)

Register 32.48. LP_I2C_FIFO_CONF_REG (0x0018)



LP_I2C_RXFIFO_WM_THRHD Configures the water mark threshold of RXFIFO in nonfifo access mode. When LP_I2C_FIFO_PRT_EN is 1 and rx FIFO counter is bigger than LP_I2C_RXFIFO_WM_THRHD[4:0], LP_I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

LP_I2C_TXFIFO_WM_THRHD Configures the water mark threshold of TXFIFO in nonfifo access mode. When LP_I2C_FIFO_PRT_EN is 1 and rx FIFO counter is bigger than LP_I2C_TXFIFO_WM_THRHD[4:0], LP_I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

LP_I2C_NONFIFO_EN Configures to enable APB nonfifo access. (R/W)

LP_I2C_RX_FIFO_RST Configures to reset RXFIFO.

0: No effect

1: Reset

(R/W)

LP_I2C_TX_FIFO_RST Configures to reset TXFIFO. 0: No effect

1: Reset

(R/W)

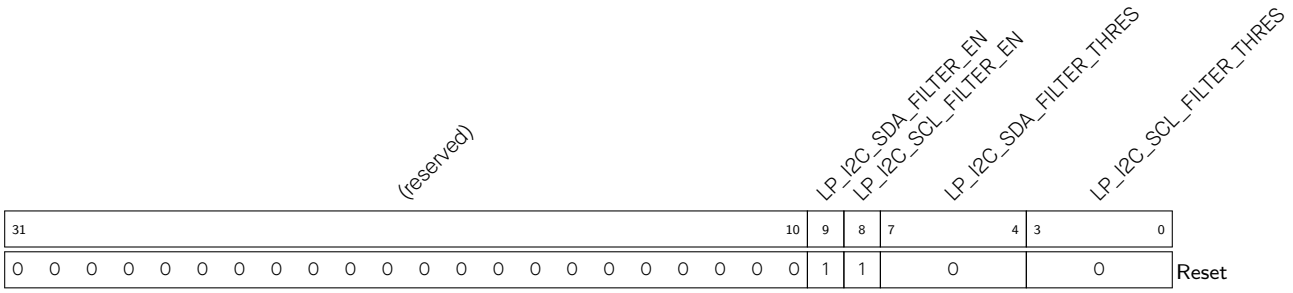
LP_I2C_FIFO_PRT_EN Configures to enable FIFO pointer in non-fifo access mode. This bit controls the valid bits and the TX/RX FIFO overflow, underflow, full and empty interrupts.

0: No effect

1: Enable

(R/W)

Register 32.49. LP_I2C_FILTER_CFG_REG (0x0050)



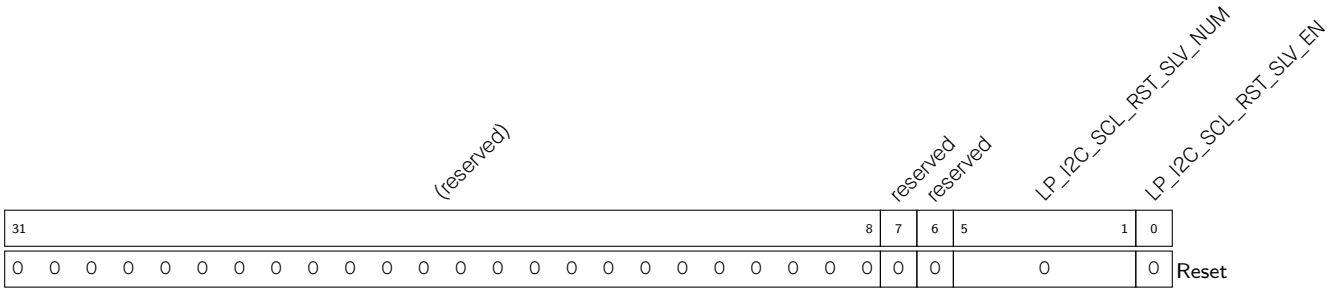
LP_I2C_SCL_FILTER_THRES Configures the threshold pulse width to be filtered on SCL. When a pulse on the SCL input has smaller width than this value, the I2C controller will ignore that pulse. Measurement unit: i2c_sclk
(R/W)

LP_I2C_SDA_FILTER_THRES Configures the threshold pulse width to be filtered on SDA. When a pulse on the SDA input has smaller width than this value, the I2C controller will ignore that pulse. Measurement unit: i2c_sclk
(R/W)

LP_I2C_SCL_FILTER_EN Configures to enable the filter function for SCL.
0: No effect
1: Enable
(R/W)

LP_I2C_SDA_FILTER_EN Configures to enable the filter function for SDA.
0: No effect
1: Enable
(R/W)

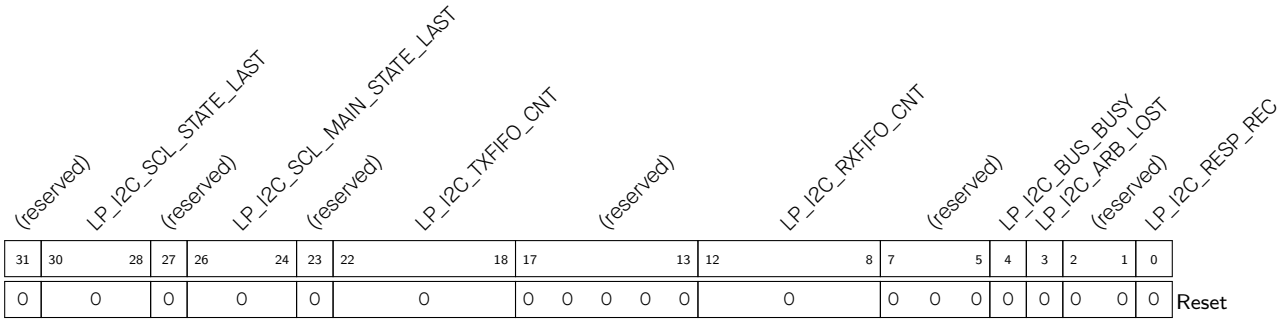
Register 32.50. LP_I2C_SCL_SP_CONF_REG (0x0080)



LP_I2C_SCL_RST_SLV_EN Configures to send out SCL pulses when I2C master is IDLE. The number of pulses equals to LP_I2C_SCL_RST_SLV_NUM[4:0]. (R/W/SC)

LP_I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode. Valid when LP_I2C_SCL_RST_SLV_EN is 1.
Measurement unit: i2c_sclk
(R/W)

Register 32.51. LP_I2C_SR_REG (0x0008)



LP_I2C_RESP_REC Represents the received ACK value in master mode or slave mode.

0: ACK

1: NACK

(RO)

LP_I2C_ARB_LOST Represents whether the I2C controller loses control of SCL line.

0: No arbitration lost

1: Arbitration lost

(RO)

LP_I2C_BUS_BUSY Represents the I2C bus state.

1: The I2C bus is busy transferring data

0: The I2C bus is in idle state

(RO)

LP_I2C_RXFIFO_CNT Represents the number of data bytes received in RAM. (RO)

LP_I2C_TXFIFO_CNT Represents the number of data bytes to be sent. (RO)

LP_I2C_SCL_MAIN_STATE_LAST Represents the states of the I2C module state machine.

0: Idle

1: Address shift

2: ACK address

3: Rx data

4: Tx data

5: Send ACK

6: Wait ACK

(RO)

LP_I2C_SCL_STATE_LAST Represents the states of the state machine used to produce SCL.

0: Idle

1: Start

2: Negative edge

3: Low

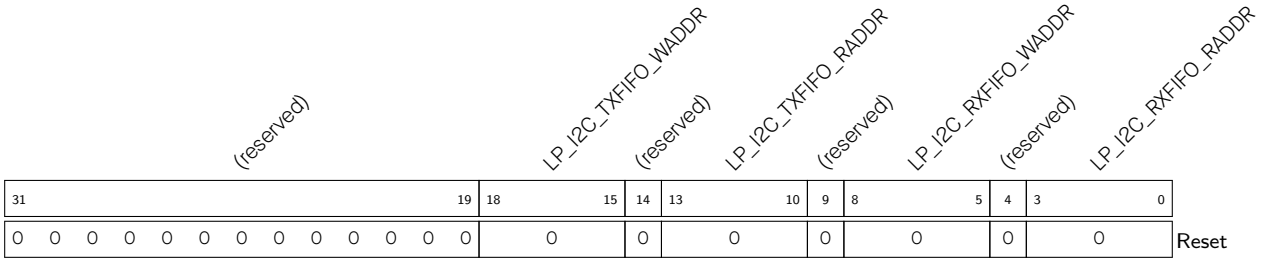
4: Positive edge

5: High

6: Stop

(RO)

Register 32.52. LP_I2C_FIFO_ST_REG (0x0014)



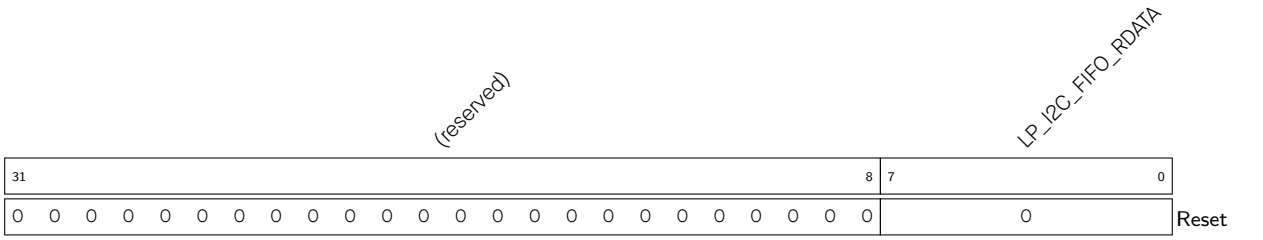
LP_I2C_RXFIFO_RADDR Represents the offset address of the APB reading from RXFIFO (RO)

LP_I2C_RXFIFO_WADDR Represents the offset address of i2c module receiving data and writing to RXFIFO. (RO)

LP_I2C_TXFIFO_RADDR Represents the offset address of i2c module reading from TXFIFO. (RO)

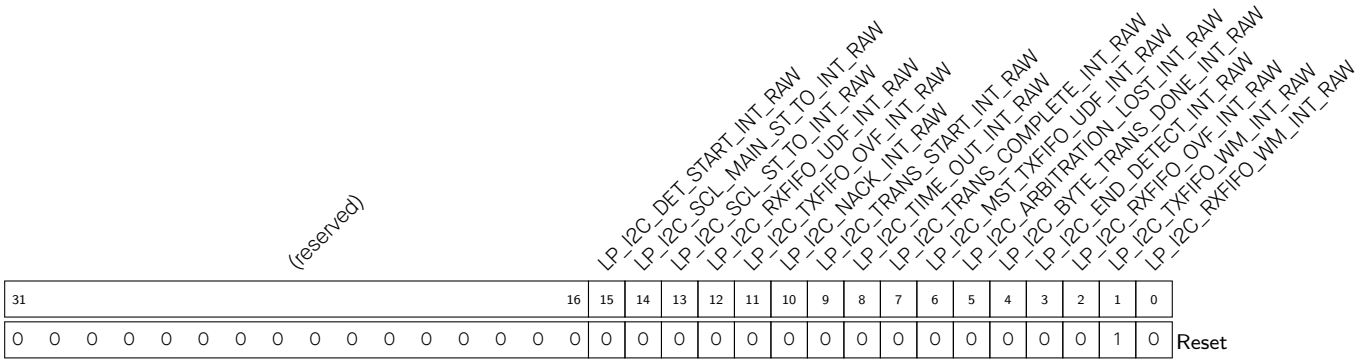
LP_I2C_TXFIFO_WADDR Represents the offset address of APB bus writing to TXFIFO. (RO)

Register 32.53. LP_I2C_DATA_REG (0x001C)



LP_I2C_FIFO_RDATA Represents the value of RXFIFO read data. (RO)

Register 32.54. LP_I2C_INT_RAW_REG (0x0020)



LP_I2C_RXFIFO_WM_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_WM_INT interrupt. (R/SS/WTC)

LP_I2C_TXFIFO_WM_INT_RAW The raw interrupt status of LP_I2C_TXFIFO_WM_INT interrupt. (R/SS/WTC)

LP_I2C_RXFIFO_OVF_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_OVF_INT interrupt. (R/SS/WTC)

LP_I2C_END_DETECT_INT_RAW The raw interrupt status of the LP_I2C_END_DETECT_INT interrupt. (R/SS/WTC)

LP_I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt status of the LP_I2C_END_DETECT_INT interrupt. (R/SS/WTC)

LP_I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status of the LP_I2C_ARBITRATION_LOST_INT interrupt. (R/SS/WTC)

LP_I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt status of LP_I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

LP_I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status of the LP_I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

LP_I2C_TIME_OUT_INT_RAW The raw interrupt status of the LP_I2C_TIME_OUT_INT interrupt. (R/SS/WTC)

Continued on the next page...

Register 32.54. LP_I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

LP_I2C_TRANS_START_INT_RAW The raw interrupt status of the LP_I2C_TRANS_START_INT interrupt. (R/SS/WTC)

LP_I2C_NACK_INT_RAW The raw interrupt status of LP_I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

LP_I2C_TXFIFO_OVF_INT_RAW The raw interrupt status of LP_I2C_TXFIFO_OVF_INT interrupt. (R/SS/WTC)

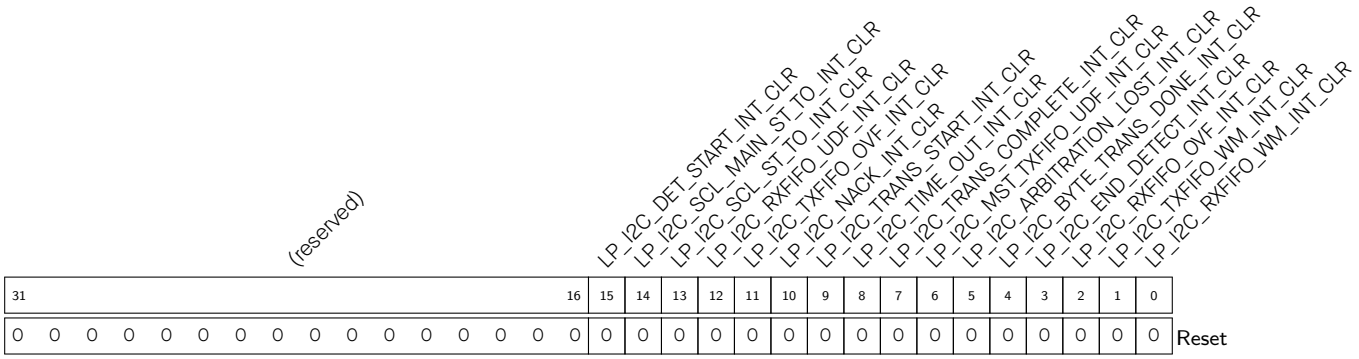
LP_I2C_RXFIFO_UDF_INT_RAW The raw interrupt status of LP_I2C_RXFIFO_UDF_INT interrupt. (R/SS/WTC)

LP_I2C_SCL_ST_TO_INT_RAW The raw interrupt status of LP_I2C_SCL_ST_TO_INT interrupt. (R/SS/WTC)

LP_I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt status of LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (R/SS/WTC)

LP_I2C_DET_START_INT_RAW The raw interrupt status of LP_I2C_DET_START_INT interrupt. (R/SS/WTC)

Register 32.55. LP_I2C_INT_CLR_REG (0x0024)



LP_I2C_RXFIFO_WM_INT_CLR Write 1 to clear LP_I2C_RXFIFO_WM_INT interrupt. (WT)

LP_I2C_TXFIFO_WM_INT_CLR Write 1 to clear LP_I2C_TXFIFO_WM_INT interrupt. (WT)

LP_I2C_RXFIFO_OVF_INT_CLR Write 1 to clear LP_I2C_RXFIFO_OVF_INT interrupt. (WT)

LP_I2C_END_DETECT_INT_CLR Write 1 to clear the LP_I2C_END_DETECT_INT interrupt. (WT)

LP_I2C_BYTE_TRANS_DONE_INT_CLR Write 1 to clear the LP_I2C_END_DETECT_INT interrupt. (WT)

LP_I2C_ARBITRATION_LOST_INT_CLR Write 1 to clear the LP_I2C_ARBITRATION_LOST_INT interrupt. (WT)

LP_I2C_MST_TXFIFO_UDF_INT_CLR Write 1 to clear LP_I2C_TRANS_COMPLETE_INT interrupt. (WT)

LP_I2C_TRANS_COMPLETE_INT_CLR Write 1 to clear the LP_I2C_TRANS_COMPLETE_INT interrupt. (WT)

LP_I2C_TIME_OUT_INT_CLR Write 1 to clear the LP_I2C_TIME_OUT_INT interrupt. (WT)

LP_I2C_TRANS_START_INT_CLR Write 1 to clear the LP_I2C_TRANS_START_INT interrupt. (WT)

LP_I2C_NACK_INT_CLR Write 1 to clear LP_I2C_SLAVE_STRETCH_INT interrupt. (WT)

LP_I2C_TXFIFO_OVF_INT_CLR Write 1 to clear LP_I2C_TXFIFO_OVF_INT interrupt. (WT)

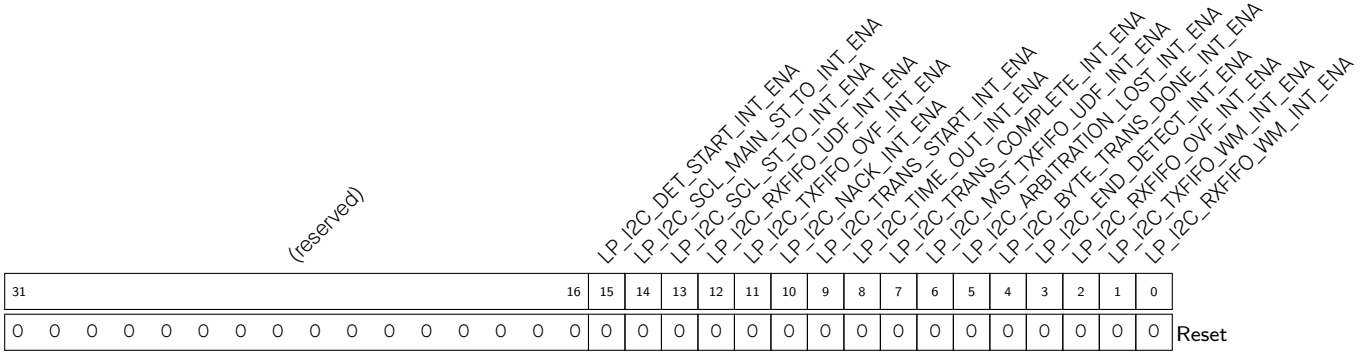
LP_I2C_RXFIFO_UDF_INT_CLR Write 1 to clear LP_I2C_RXFIFO_UDF_INT interrupt. (WT)

LP_I2C_SCL_ST_TO_INT_CLR Write 1 to clear LP_I2C_SCL_ST_TO_INT interrupt. (WT)

LP_I2C_SCL_MAIN_ST_TO_INT_CLR Write 1 to clear LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (WT)

LP_I2C_DET_START_INT_CLR Write 1 to clear LP_I2C_DET_START_INT interrupt. (WT)

Register 32.56. LP_I2C_INT_ENA_REG (0x0028)



LP_I2C_RXFIFO_WM_INT_ENA Write 1 to enable LP_I2C_RXFIFO_WM_INT interrupt. (R/W)

LP_I2C_TXFIFO_WM_INT_ENA Write 1 to enable LP_I2C_TXFIFO_WM_INT interrupt. (R/W)

LP_I2C_RXFIFO_OVF_INT_ENA Write 1 to enable LP_I2C_RXFIFO_OVF_INT interrupt. (R/W)

LP_I2C_END_DETECT_INT_ENA Write 1 to enable the LP_I2C_END_DETECT_INT interrupt. (R/W)

LP_I2C_BYTE_TRANS_DONE_INT_ENA Write 1 to enable the LP_I2C_END_DETECT_INT interrupt. (R/W)

LP_I2C_ARBITRATION_LOST_INT_ENA Write 1 to enable the LP_I2C_ARBITRATION_LOST_INT interrupt. (R/W)

LP_I2C_MST_TXFIFO_UDF_INT_ENA Write 1 to enable LP_I2C_TRANS_COMPLETE_INT interrupt. (R/W)

LP_I2C_TRANS_COMPLETE_INT_ENA Write 1 to enable the LP_I2C_TRANS_COMPLETE_INT interrupt. (R/W)

LP_I2C_TIME_OUT_INT_ENA Write 1 to enable the LP_I2C_TIME_OUT_INT interrupt. (R/W)

LP_I2C_TRANS_START_INT_ENA Write 1 to enable the LP_I2C_TRANS_START_INT interrupt. (R/W)

LP_I2C_NACK_INT_ENA Write 1 to enable LP_I2C_SLAVE_STRETCH_INT interrupt. (R/W)

LP_I2C_TXFIFO_OVF_INT_ENA Write 1 to enable LP_I2C_TXFIFO_OVF_INT interrupt. (R/W)

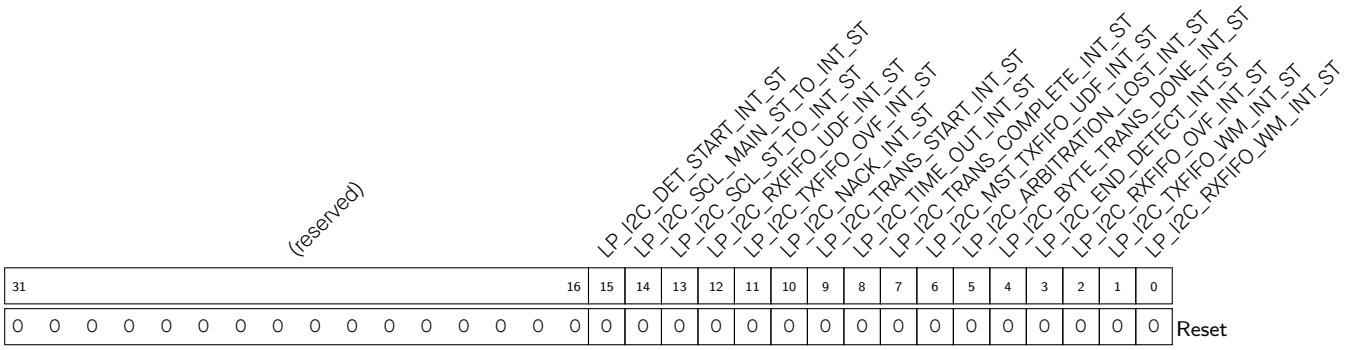
LP_I2C_RXFIFO_UDF_INT_ENA Write 1 to enable LP_I2C_RXFIFO_UDF_INT interrupt. (R/W)

LP_I2C_SCL_ST_TO_INT_ENA Write 1 to enable LP_I2C_SCL_ST_TO_INT interrupt. (R/W)

LP_I2C_SCL_MAIN_ST_TO_INT_ENA Write 1 to enable LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

LP_I2C_DET_START_INT_ENA Write 1 to enable LP_I2C_DET_START_INT interrupt. (R/W)

Register 32.57. LP_I2C_INT_STATUS_REG (0x002C)



LP_I2C_RXFIFO_WM_INT_ST The masked interrupt status of LP_I2C_RXFIFO_WM_INT interrupt. (RO)

LP_I2C_TXFIFO_WM_INT_ST The masked interrupt status of LP_I2C_TXFIFO_WM_INT interrupt. (RO)

LP_I2C_RXFIFO_OVF_INT_ST The masked interrupt status of LP_I2C_RXFIFO_OVF_INT interrupt. (RO)

LP_I2C_END_DETECT_INT_ST The masked interrupt status of the LP_I2C_END_DETECT_INT interrupt. (RO)

LP_I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status of the LP_I2C_END_DETECT_INT interrupt. (RO)

LP_I2C_ARBITRATION_LOST_INT_ST The masked interrupt status of the LP_I2C_ARBITRATION_LOST_INT interrupt. (RO)

LP_I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status of LP_I2C_TRANS_COMPLETE_INT interrupt. (RO)

LP_I2C_TRANS_COMPLETE_INT_ST The masked interrupt status of the LP_I2C_TRANS_COMPLETE_INT interrupt. (RO)

LP_I2C_TIME_OUT_INT_ST The masked interrupt status of the LP_I2C_TIME_OUT_INT interrupt. (RO)

LP_I2C_TRANS_START_INT_ST The masked interrupt status of the LP_I2C_TRANS_START_INT interrupt. (RO)

Continued on the next page...

Register 32.57. LP_I2C_INT_STATUS_REG (0x002C)

Continued from the previous page...

LP_I2C_NACK_INT_ST The masked interrupt status of LP_I2C_SLAVE_STRETCH_INT interrupt. (RO)

LP_I2C_TXFIFO_OVF_INT_ST The masked interrupt status of LP_I2C_TXFIFO_OVF_INT interrupt. (RO)

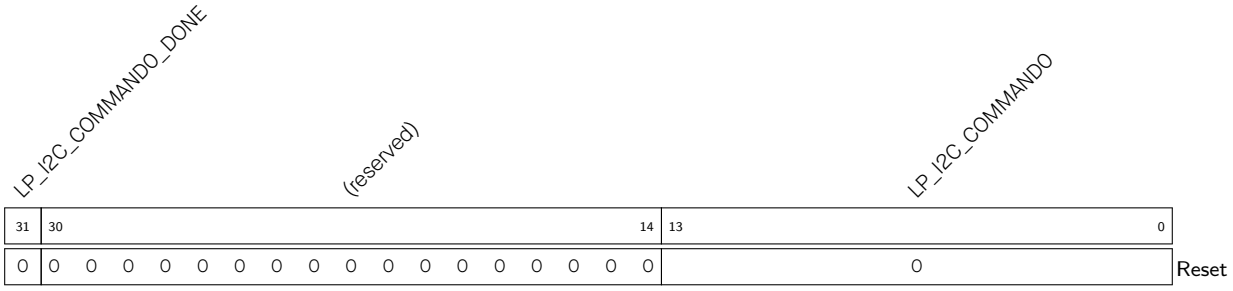
LP_I2C_RXFIFO_UDF_INT_ST The masked interrupt status of LP_I2C_RXFIFO_UDF_INT interrupt. (RO)

LP_I2C_SCL_ST_TO_INT_ST The masked interrupt status of LP_I2C_SCL_ST_TO_INT interrupt. (RO)

LP_I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status of LP_I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

LP_I2C_DET_START_INT_ST The masked interrupt status of LP_I2C_DET_START_INT interrupt. (RO)

Register 32.58. LP_I2C_COMD0_REG (0x0058)



LP_I2C_COMMAND0 Configures command 0.

It consists of three parts:

op_code is the command

1: WRITE

2: STOP

3: READ

4: END

6: RSTART

Byte_num represents the number of bytes that need to be sent or received.

ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure 32.4-2 for more information. (R/W)

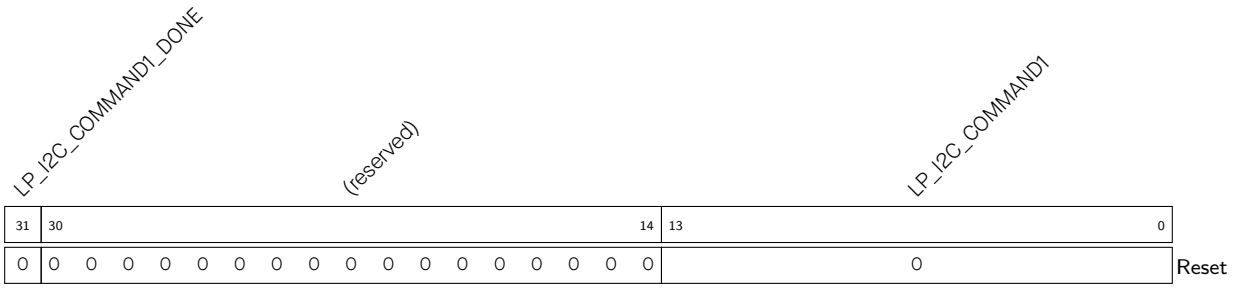
LP_I2C_COMMAND0_DONE Represents whether command 0 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.59. LP_I2C_COMD1_REG (0x005C)



LP_I2C_COMMAND1 Configures command 1.

See details in I2C_CMD0_REG[13:0]. (R/W)

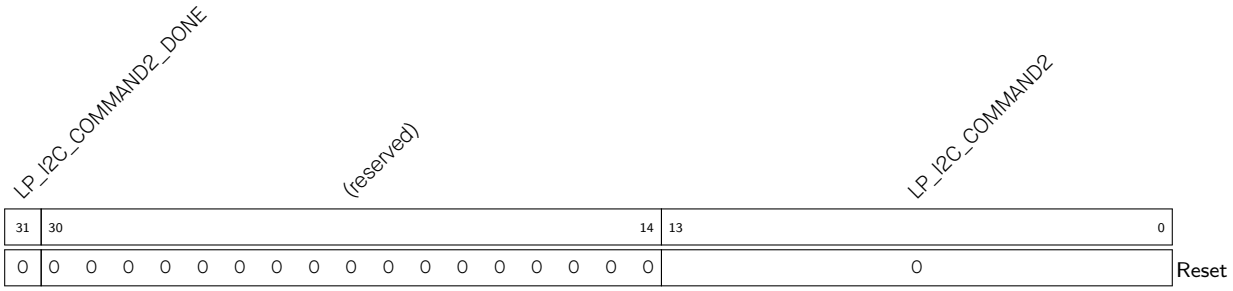
LP_I2C_COMMAND1_DONE Represents whether command 1 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.60. LP_I2C_COMD2_REG (0x0060)



LP_I2C_COMMAND2 Configures command 2. See details in I2C_CMD0_REG[13:0]. (R/W)

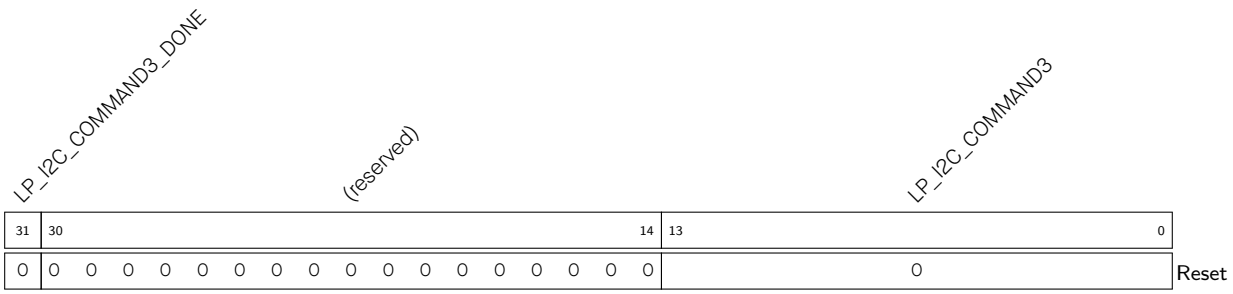
LP_I2C_COMMAND2_DONE Represents whether command 2 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.61. LP_I2C_COMD3_REG (0x0064)



LP_I2C_COMMAND3 Configures command 3. See details in I2C_CMD0_REG[13:0]. (R/W)

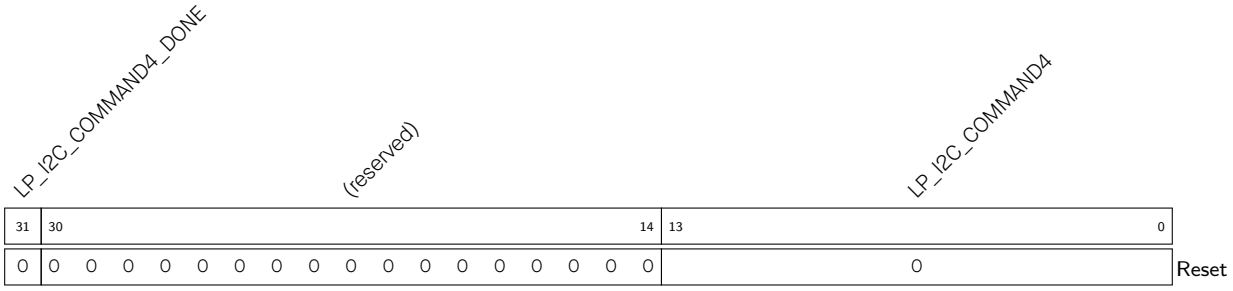
LP_I2C_COMMAND3_DONE Represents whether command 3 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.62. LP_I2C_COMD4_REG (0x0068)



LP_I2C_COMMAND4 Configures command 4. See details in I2C_CMD0_REG[13:0]. (R/W)

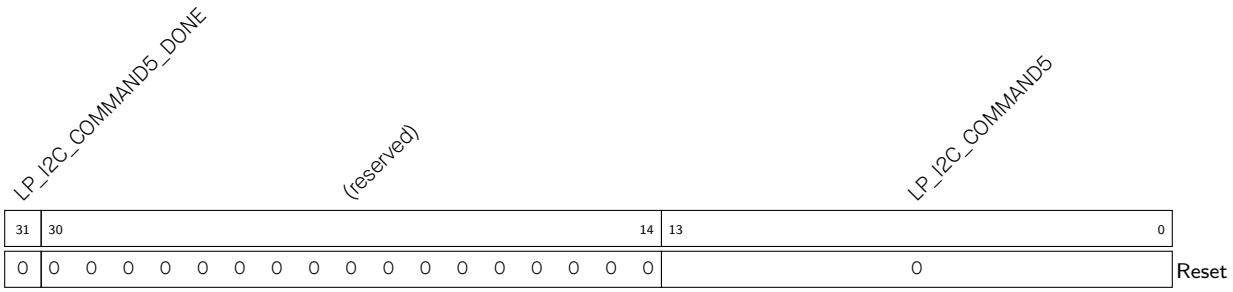
LP_I2C_COMMAND4_DONE Represents whether command 4 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.63. LP_I2C_COMD5_REG (0x006C)



LP_I2C_COMMAND5 Configures command 5. See details in I2C_CMD0_REG[13:0]. (R/W)

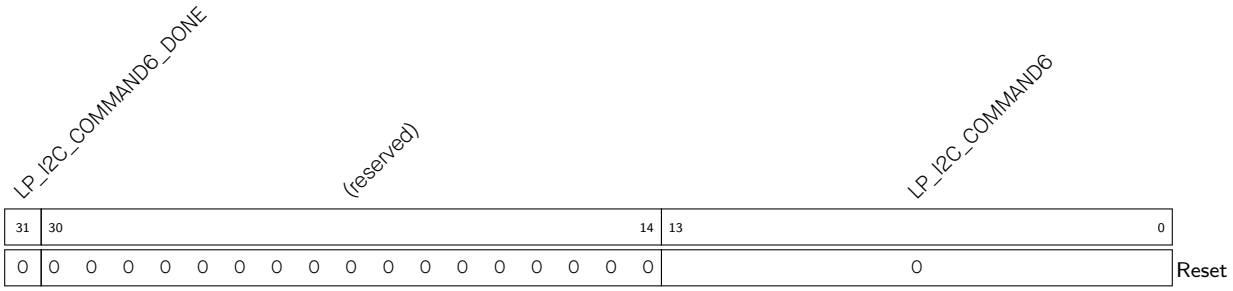
LP_I2C_COMMAND5_DONE Represents whether command 5 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.64. LP_I2C_COMD6_REG (0x0070)



LP_I2C_COMMAND6 Configures command 6. See details in I2C_CMD0_REG[13:0]. (R/W)

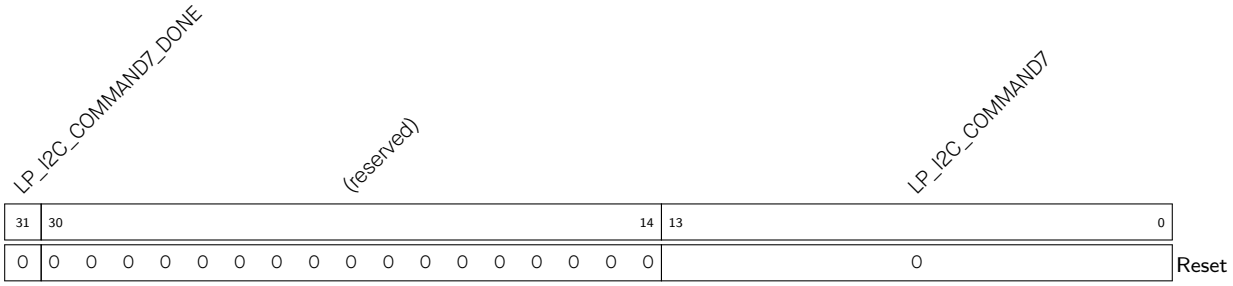
LP_I2C_COMMAND6_DONE Represents whether command 6 is done in I2C Master mode.

0: Not done

1: Done

(R/W/SS)

Register 32.65. LP_I2C_COMD7_REG (0x0074)



LP_I2C_COMMAND7 Configures command 7. See details in I2C_CMD0_REG[13:0]. (R/W)

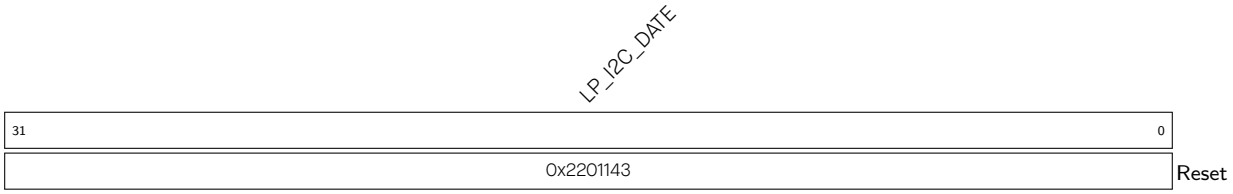
LP_I2C_COMMAND7_DONE Represents whether command 7 is done in I2C Master mode.

0: Not done

1: Done

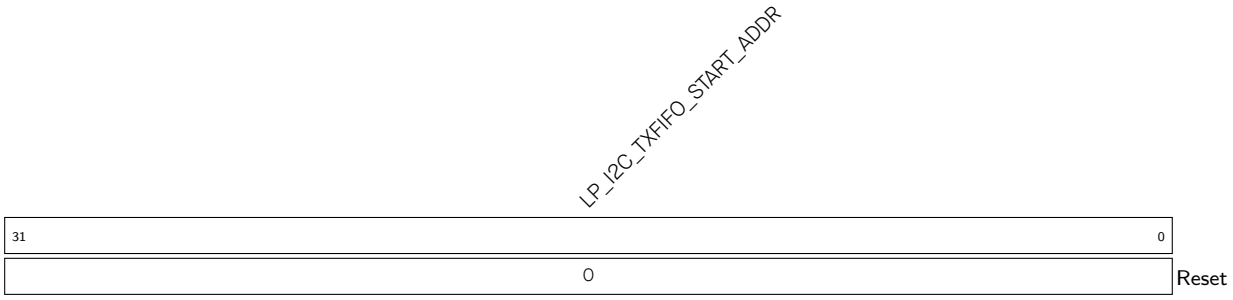
(R/W/SS)

Register 32.66. LP_I2C_DATE_REG (0x00F8)



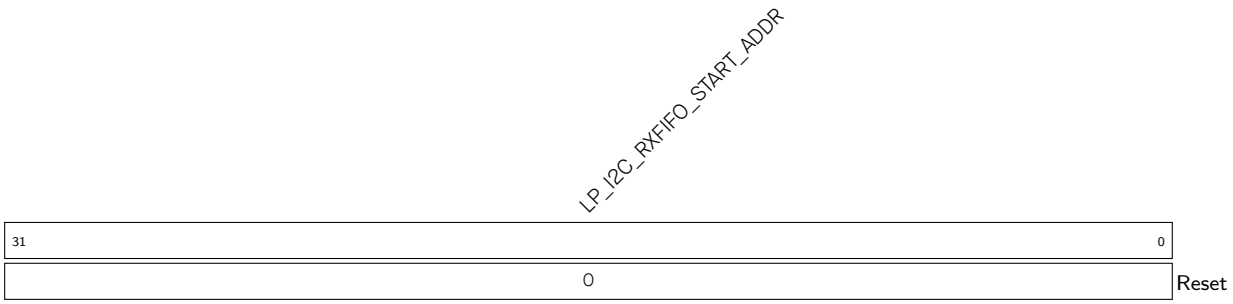
LP_I2C_DATE Version control register. (R/W)

Register 32.67. LP_I2C_TXFIFO_START_ADDR_REG (0x0100)



LP_I2C_TXFIFO_START_ADDR Represents the I2C txfifo first address. (HRO)

Register 32.68. LP_I2C_RXFIFO_START_ADDR_REG (0x0180)



LP_I2C_RXFIFO_START_ADDR Represents the I2C rxfifo first address. (HRO)

Chapter 33

Analog I2C Controller

33.1 Introduction

The Analog I2C Controller contains two I2C masters dedicated to configuring and communicating with certain analog circuit modules. Those modules can adjust their operating states by configuring their internal registers to better suit the current application scenario of the chip. For example, the Analog I2C Controller can configure an analog sensor to add a uniform offset to the ADC sampling values, thereby extending the sampling voltage range. Each configurable analog module has an I2C slave internally, which is assigned with an independent address. The Analog I2C Controller supports the configuration of the following analog modules:

- CPU_PLL: This mainly includes the configuration register for CPLL_CLK, with a slave address of 0x67. For more information on CPLL_CLK, refer to Chapter 8 *Reset and Clock*.
- SYS_PLL: This mainly includes the configuration register for SPLL_CLK, with a slave address of 0x66. For more information on SPLL_CLK, refer to Chapter 8 *Reset and Clock*.
- MSPI: This mainly includes the configuration register for MPLL_CLK , with a slave address of 0x63. For more information on MPLL_CLK, refer to Chapter 8 *Reset and Clock*.
- PLLA: This mainly includes the configuration register for APLL_CLK , with a slave address of 0x6f. For more information on APLL_CLK, refer to Chapter 8 *Reset and Clock*.
- ANA_SENSOR: This mainly includes the configuration register for analog sensors (such as *Temperature Sensor (TSENS)* and *ADC Controller (ADC)*), with a slave address of 0x69.
- BIAS: This mainly includes the voltage detection configuration register, with a slave address of 0x6a. For more information, refer to Chapter 19 *Brown-out Detector*.

33.2 Feature List

The Analog I2C Controller has the following features:

- Master mode only
- 7-bit addressing
- Adjustable transmission rate
- Communication in the sleep modes supported by the Low-Power CPU
- Dual master operation mode

33.3 Architectural Overview

The figure below shows the architecture of the Analog I2C Controller.

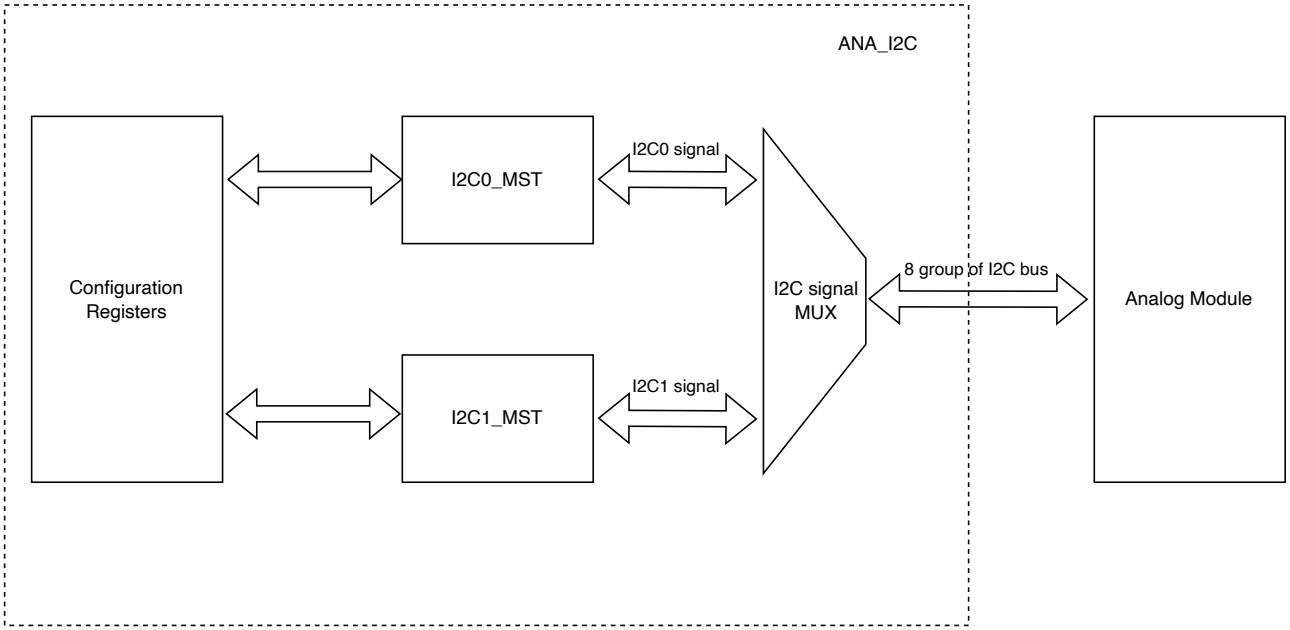


Figure 33.3-1. Analog I2C Controller Architecture

- Configuration Registers: A register module interfacing with the software, providing relevant configurations for the operation of the I2C master. This module controls the transmission address, data, etc., and monitors this transmission process by reading the status registers of I2C0/1_MST.
- I2C0/1_MST: Two I2C masters that can independently configure analog modules in parallel. The I2C signals they generate will be selected by a subsequent MUX and output to the analog modules.
- I2C signal MUX: A module that determines which analog master, I2C0_MST or I2C1_MST, configures a particular analog module.

33.4 Functional Description

Transmission Rate Adjustment

The transmission rate and signal phase of the I2C signal can be configured by ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD and ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR. The relationship between the register values and the waveform is shown in the following figure.

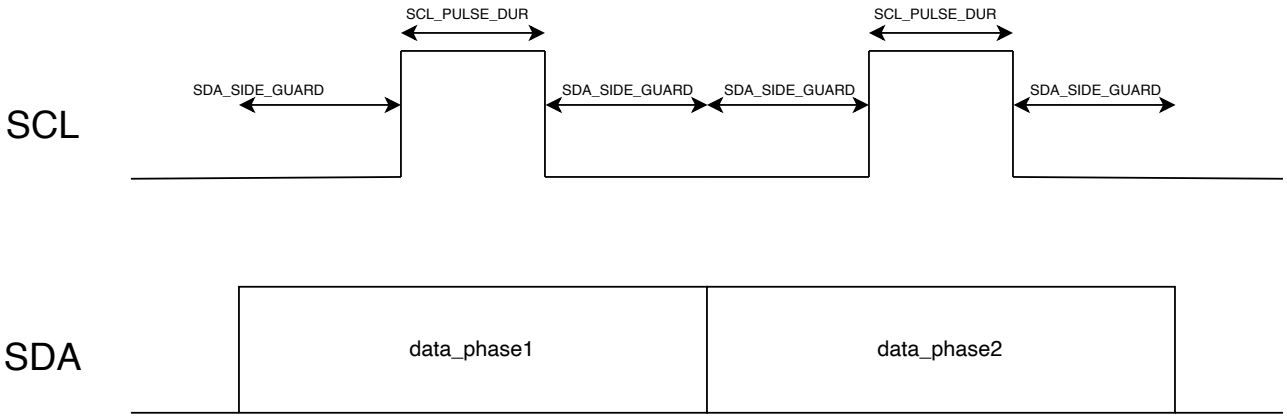


Figure 33.4-1. Signal Phase

Operation in Sleep Modes

In chip sleep modes, the operation of the Analog I2C Controller can be driven by the Low-Power CPU (see Chapter *2 Low-Power CPU*).

Dual Master Operation Mode

The module integrates two independent master modules, each with its own set of configurations. The two masters can operate independently, configuring different analog modules, thereby improving operational efficiency.

33.5 Programming Procedures

The following is the process for configuring analog modules using the Analog I2C Controller.

- Write 1 to LPPERI_CK_EN_LP_I2CMST to enable the clock of the Controller.
- Configure ANA_I2C_MST_ANA_CONF2 to select the module for communication.
- Configure ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD and ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR to determine the transmission rate and signal phase of the I2C signal.
- Configure ANA_I2C_MST_I2C*x*_CTRL to select relevant information for the current communication.
- Read ANA_I2C_MST_I2C*x*_BUSY. A low state indicates the end of the current communication for I2C*x*.

33.6 Register Summary

The addresses in this section are relative to Analog I2C Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

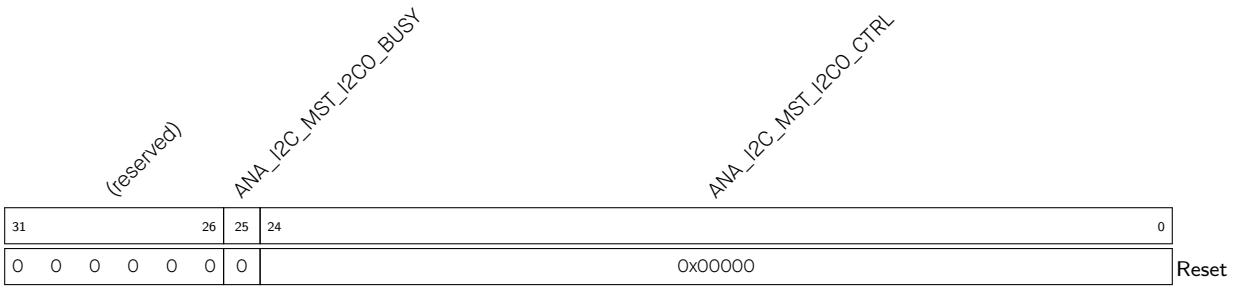
Name	Description	Address	Access
Configuration Registers			
ANA_I2C_MST_I2C0_CTRL_REG	I2C0 transmission configuration register	0x0000	varies
ANA_I2C_MST_I2C1_CTRL_REG	I2C1 transmission configuration register	0x0004	varies
ANA_I2C_MST_ANA_CONF2_REG	I2C master selection register	0x0020	varies
ANA_I2C_MST_I2C0_CTRL1_REG	I2C0 transmission rate and signal phase configuration register	0x0024	R/W
ANA_I2C_MST_I2C1_CTRL1_REG	I2C1 transmission rate and signal phase configuration register	0x0028	R/W
ANA_I2C_MST_DATE_REG	Version control register	0x0038	R/W

33.7 Registers

The addresses in this section are relative to Analog I2C Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 33.1. ANA_I2C_MST_I2C0_CTRL_REG (0x0000)



ANA_I2C_MST_I2C0_CTRL Configures the transmission information for I2C0.

Bit[0:7]: Configures the slave address

Bit[8:15]: Configures the slave register address

Bit[16:23]: Configures the transmitted data

Bit[24]: Configures the read or write operation

0: Write

1: Read

Once this register is configured, the I2C0 master will generate I2C read or write signals. (R/W)

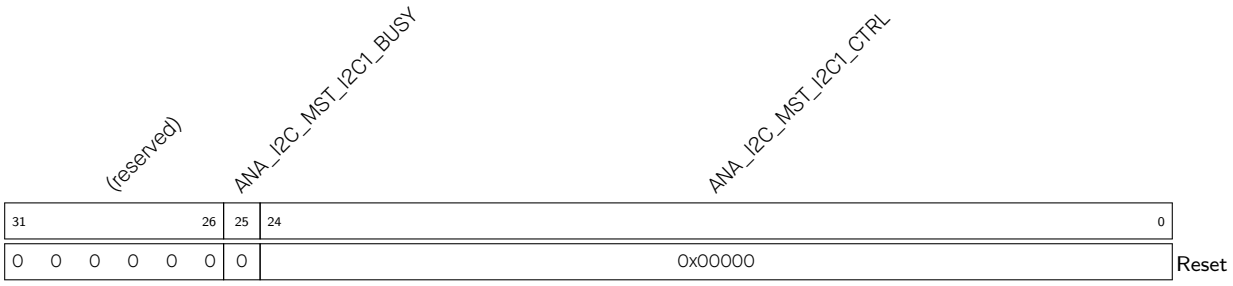
ANA_I2C_MST_I2C0_BUSY Represents whether I2C0 is currently transferring data.

0: I2C0 is not transferring data

1: I2C0 is transferring data

(RO)

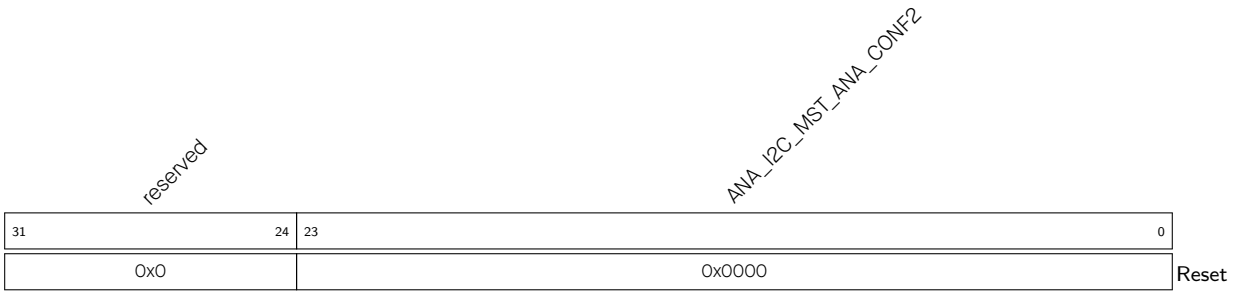
Register 33.2. ANA_I2C_MST_I2C1_CTRL_REG (0x0004)



ANA_I2C_MST_I2C1_CTRL Configures the transmission information for I2C1. Its subfields have the same meaning with ANA_I2C_MST_I2C0_CTRL. (R/W)

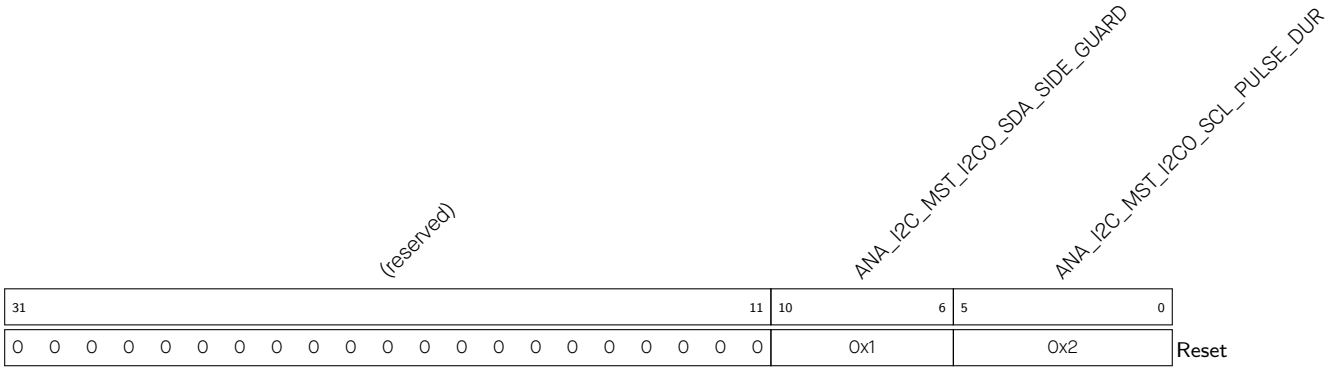
ANA_I2C_MST_I2C1_BUSY Represents whether I2C1 is currently transferring data
0: I2C1 is not transferring data
1: I2C1 is transferring data
(RO)

Register 33.3. ANA_I2C_MST_ANA_CONF2_REG (0x0020)



ANA_I2C_MST_ANA_CONF2 Configures which I2C master the following analog modules uses. If the corresponding bit is set to 1, I2C0 master is used for communication; if set to 0, I2C1 master is used. Since the slave address has already been defined, erroneous communication will not be caused.
Bit5: SYS_PLL
Bit6: SDIO_PLL
Bit7: ANA_SENSOR
Bit8: PLLA
Bit9: MSPI
Bit10: DIG_REG
Bit11: CPU_PLL
Bit12: BIAS
(R/W)

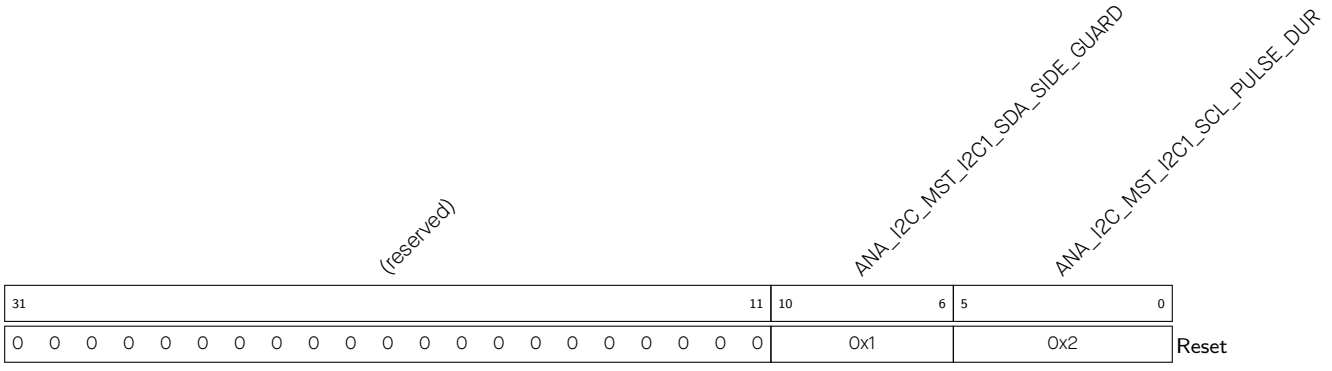
Register 33.4. ANA_I2C_MST_I2C0_CTRL1_REG (0x0024)



ANA_I2C_MST_I2C0_SCL_PULSE_DUR Configures the duration of the high-level period of the SCL driven by I2C0. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

ANA_I2C_MST_I2C0_SDA_SIDE_GUARD Configures the duration of the low-level period of the SCL driven by I2C0. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

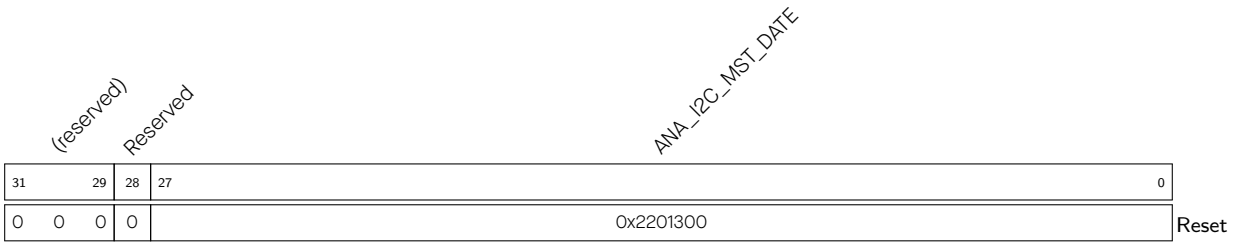
Register 33.5. ANA_I2C_MST_I2C1_CTRL1_REG (0x0028)



ANA_I2C_MST_I2C1_SCL_PULSE_DUR Configures the duration of the high-level period of the SCL driven by I2C1. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

ANA_I2C_MST_I2C1_SDA_SIDE_GUARD Configures the duration of the low-level period of the SCL driven by I2C1. The duration is measured by the counter operating on the LP_FAST_CLK clock. (R/W)

Register 33.6. ANA_I2C_MST_DATE_REG (0x0038)



ANA_I2C_MST_DATE Version control register. (R/W)

Chapter 34

I2S Controller (I2S)

34.1 Overview

ESP32-P4 has three built-in I2S interfaces (i.e., I2S*0*, I2S*1*, and I2S*2*), which provide flexible communication interfaces for streaming digital data in multimedia applications, especially digital audio applications.

The I2S standard bus defines three signals, namely, a bit clock signal (BCK), a channel/word select signal (WS), and a serial data signal (SD). A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication. The I2S module on ESP32-P4 provides separate transmit (TX) and receive (RX) units for high performance.

Note:
The information provided in this chapter applies to I2S*0*, I2S*1*, and I2S*2*. Unless otherwise indicated, I2S or I2S*n* in this chapter refer to I2S*0*, I2S*1*, and I2S*2*.

34.2 Terminology

To better illustrate the functionality of I2S, the following terms are used in this chapter.

Master mode — As a master, I2S*n* drives BCK/WS signals and transmits data to or receives data from a slave.

Slave mode — As a slave, I2S*n* is driven by BCK/WS signals and receives data from or transmits data to a master.

Full-duplex — There are two separate data lines. The transmitted and received data is carried simultaneously.

Half-duplex — Only one side, the master or the slave, transmits data first, and the other side receives data. Data transmission and reception cannot occur simultaneously.

A-law and μ-law — A-law and μ-law are compression/decompression algorithms in digital pulse code modulated (PCM) non-uniform quantization, which can effectively improve the signal-to-quantization noise ratio.

TDM RX mode — In this mode, pulse code modulated (PCM) data is received utilizing time division multiplexing (TDM). The signal lines include BCK, WS, and SD. Data from 16 channels at most can be received. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM RX mode In this mode, pulse density modulation (PDM) data is received. Used signals include WS and DATA. PDM standard is supported in this mode by user configuration.

TDM TX mode In this mode, pulse code modulated (PCM) data is sent in a way of time division multiplexing (TDM). The signal lines include BCK, WS, and DATA. Data up to 16 channels can be sent. TDM Philips standard, TDM MSB alignment standard, and TDM PCM standard are supported in this mode, depending on user configuration.

PDM TX mode In this mode, pulse density modulation (PDM) data is sent. The signal lines include WS and DATA. PDM standard is supported in this mode by user configuration.

PCM-to-PDM Data Format Converter (for I2S*0* only) A data format converter that converts PCM data into PDM data (hereinafter referred to as PCM-to-PDM converter), which can be enabled in PDM TX master mode. When the converter is enabled, I2S fetches pulse code modulated (PCM) data from storage via GDMA, converts it to pulse density modulation (PDM) data, and then transmits it.

PDM-to-PCM Data Format Converter (for I2S*0* only) A data format converter that converts PDM data into PCM data (hereinafter referred to as PDM-to-PCM converter), which can be enabled in PDM RX master or slave mode. When the converter is enabled, I2S receives pulse density modulation (PDM) data, converts it into pulse code modulated (PCM) data, and stores it into memory via GDMA.

34.3 Features

The I2S module has the following features:

- Master mode and slave mode
- Full-duplex and half-duplex communications
- Separate TX and RX units that can work independently or simultaneously
- A variety of audio standards supported:
  - TDM Philips standard
  - TDM MSB alignment standard
  - TDM PCM standard
  - PDM standard
- Various TX/RX modes supported:
  - TDM TX mode, up to 16 channels supported
  - TDM RX mode, up to 16 channels supported

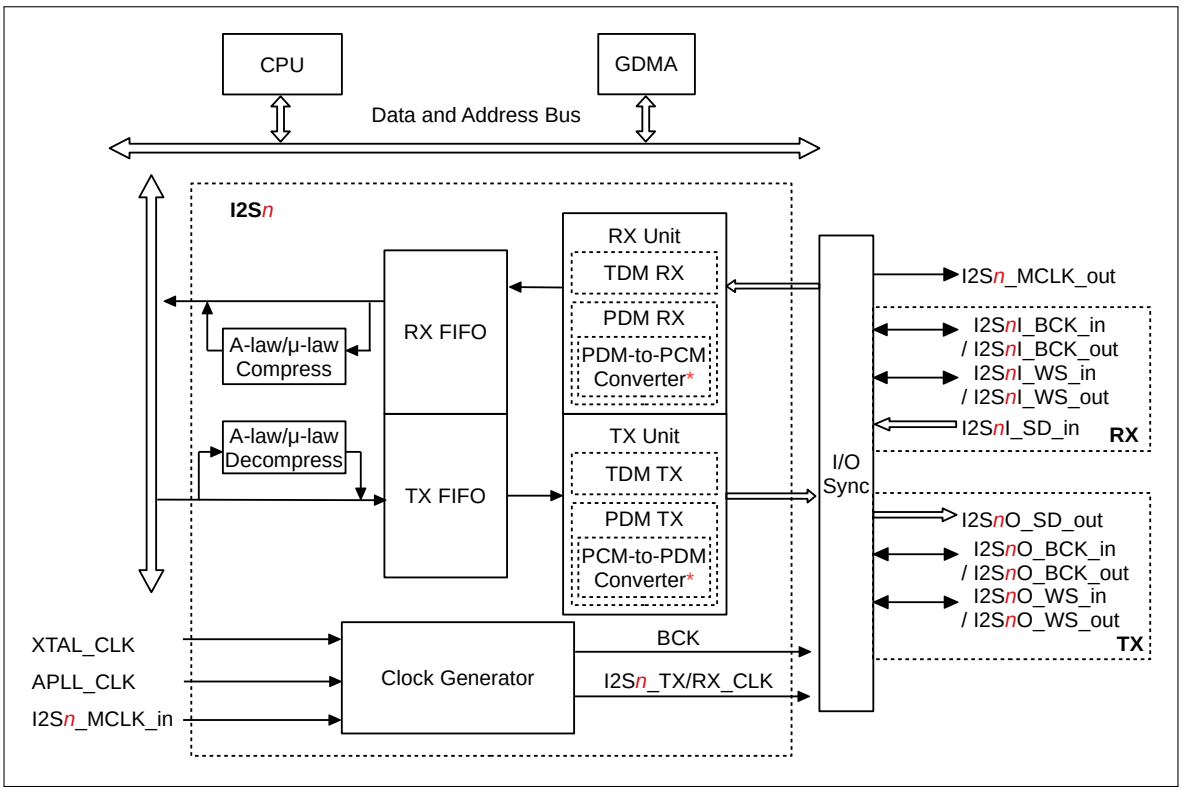
  - PDM TX mode
    * Raw PDM data transmission
    * PCM-to-PDM data format conversion (for I2S*0* only), up to 2 channels supported
  - PDM RX mode
    * Raw PDM data reception
    * PDM-to-PCM data format conversion (for I2S*0* only), up to 8 channels supported
- Configurable APLL clock with frequencies up to 240 MHz
- Configurable high-precision sample clock with a variety of sampling frequencies supported (refer to the note below for more details)
- 8/16/24/32-bit data width
- Synchronous counter in TX mode
- ETM feature
- Direct Memory Access (GDMA-AHB only)
- Standard I2S interface interrupts

Note:
In slave mode, to ensure the sampling accuracy, the module clock frequency must be greater than or equal to 8 times the BCK clock frequency. Therefore, the maximum sampling frequency of I2S*n* is limited by the data bit width and the number of channels. For example, since the clock source frequency is up to 240 MHz, the module clock can be configured up to 120 MHz, and the BCK clock can be configured up to 15 MHz. Therefore, when transmitting dual-channel 32-bit width data, I2S*n* supports sample frequencies up to 234.375 kHz, e.g., 8 kHz, 16 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 128 kHz, and 192 kHz.Please refer to Section 34.6 *I2S TX/RX Clock* for detailed information.

34.4 System Architecture

Figure 34.4-1 shows the structure of the ESP32-P4 I2S*n* module, consisting of:

- Transmit control unit (TX Unit)
- Receive control unit (RX Unit)
- Input and output timing unit (I/O Sync)
- Clock divider (Clock Generator)
- 64 x 32-bit TX FIFO
- 64 x 32-bit RX FIFO
- Compress/Decompress units



Note: The PDM-to-PCM converter and PCM-to-PDM converter are only supported by I2S*O*.

Figure 34.4-1. ESP32-P4 I2S System Diagram

The I2S*n* module supports direct memory access (DMA) to internal memory. For more information, see Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

Both the TX unit and the RX unit have a three-line interface that uses a bit clock line (BCK), a word select line (WS), and a serial data line (SD). The SD line of the TX unit is used for data output and the SD line of the RX unit for data input. The BCK and WS signal lines of the TX and RX units are used for data output in master mode, and the BCK and WS signal lines of the TX and RX units are used for data input in slave mode.

The signal bus of the I2S*n* module is shown at the right part of Figure 34.4-1. The naming of these signals in RX and TX units follows the pattern of I2S*n*A_B_C such as I2S*n*I_BCK_in.

- "A" indicates that the signal belongs to the I2S*n* TX or RX unit, which includes:
  - "I": Signal input to/output from the RX unit
  - "O": Signal input to/output from the TX unit
- "B" represents the signal function, which includes:
  - BCK
  - WS
  - SD
- "C" represents the signal direction, which includes:
  - "in": Input signal into the I2S*n* module
  - "out": Output signal from the I2S*n* module

Table 34.4-1 provides a detailed description of I2S signals.

Table 34.4-1. I2S*n* Signal Description

Signal	Direction	Function
I2S*n*I_BCK_in	Input	In I2S*n* slave mode, inputs BCK signal for RX unit.
I2S*n*I_BCK_out	Output	In I2S*n* master mode, outputs BCK signal for RX unit.
I2S*n*I_WS_in	Input	In I2S*n* slave mode, inputs WS signal for RX unit.
I2S*n*I_WS_out	Output	In I2S*n* master mode, outputs WS signal for RX unit.
I2S*n*I_Data_in	Input	Works as the serial input data bus for I2S*n* RX unit.
I2S*n*O_Data_out	Output	Works as the serial output data bus for I2S*n* TX unit.
I2S*n*O_BCK_in	Input	In I2S*n* slave mode, inputs BCK signal for TX unit.
I2S*n*O_BCK_out	Output	In I2S*n* master mode, outputs BCK signal for TX unit.
I2S*n*O_WS_in	Input	In I2S*n* slave mode, inputs WS signal for TX unit.
I2S*n*O_WS_out	Output	In I2S*n* master mode, outputs WS signal for TX unit.
I2S*n*_MCLK_in	Input	In I2S*n* slave mode, works as a clock source from the external master.
I2S*n*_MCLK_out	Output	In I2S*n* master mode, works as a clock source for the external slave.
I2S*O*I_Data1_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*I_Data2_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*I_Data3_in	Input	When the PDM-to-PCM converter is enabled, works as the serial input data line for RX unit.
I2S*O*O_Data1_out	Output	When the PCM-to-PDM converter is enabled, works as the serial output data line for TX unit.

* Any required signals of I2S*n* must be mapped to the chip's pins via HP GPIO matrix, see Chapter 7 *GPIO Matrix and IO MUX*.

34.5 Supported Audio Standards

ESP32-P4 I2S*n* supports multiple audio standards, including TDM Philips standard, TDM MSB alignment standard, TDM PCM standard, and PDM standard.

Select the needed standard by configuring the following bits:

- I2S_TX/RX_TDM_EN
  - 0: Disable TDM mode
  - 1: Enable TDM mode
- I2S_TX/RX_PDM_EN
  - 0: Disable PDM mode
  - 1: Enable PDM mode
- I2S_TX/RX_MSB_SHIFT
  - 0: WS and SD signals change simultaneously, i.e., enable MSB alignment standard

– 1: WS signal changes one BCK clock cycle earlier than SD signal, i.e., enable Philips standard or select PCM standard

Note:
I2S_TX/RX_TDM_EN and I2S_TX/RX_PDM_EN must not be configured to 1 or 0 at the same time, otherwise ESP32-P4 I2S*n* will transmit data incorrectly in a mode that is neither TDM nor PDM.

34.5.1 TDM Philips Standard

Philips specifications require that WS signal changes one BCK clock cycle earlier than SD signal on BCK falling edge, which means that WS signal is valid from one clock cycle before transmitting the first bit of channel data and changes one clock before the end of channel data transfer. SD signal line transmits the most significant bit of audio data first.

Compared with the Philips standard, TDM Philips standard supports multiple channels. See Figure 34.5-1.

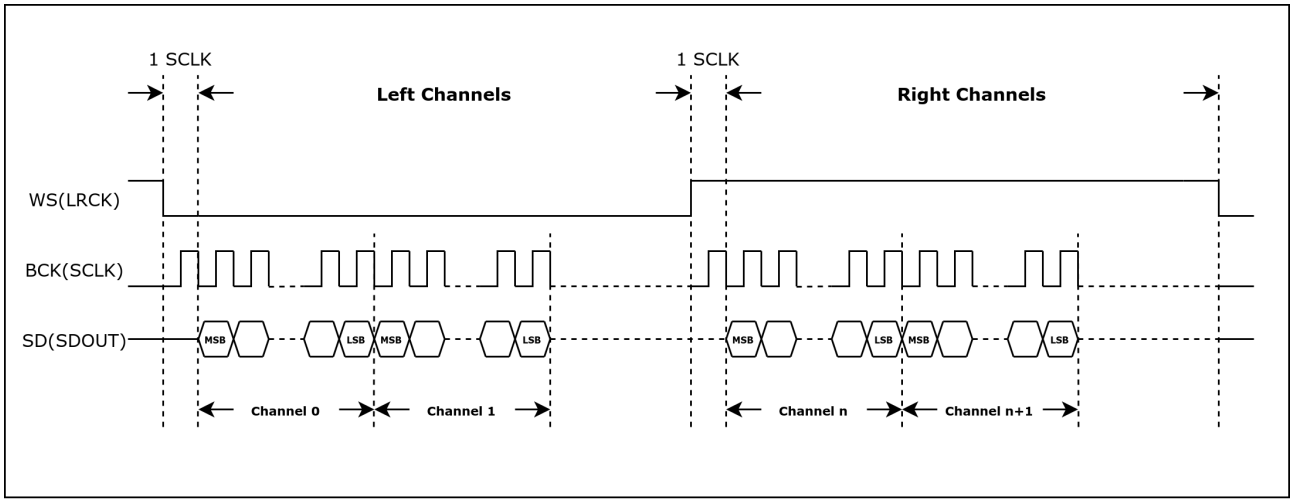


Figure 34.5-1. TDM Philips Standard Timing Diagram

34.5.2 TDM MSB Alignment Standard

MSB alignment specifications require that WS and SD signals change simultaneously on the falling edge of BCK. The WS signal is valid until the end of the channel data transfer. The SD signal line transmits the most significant bit of audio data first.

Compared with MSB alignment standard, TDM MSB alignment standard supports multiple channels. See Figure 34.5-2.

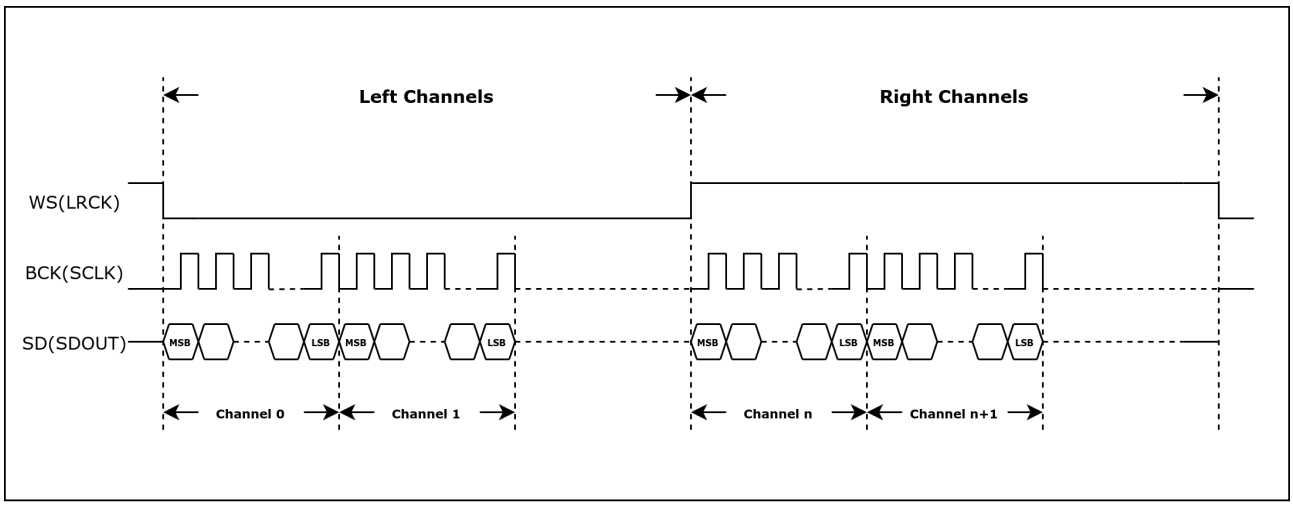


Figure 34.5-2. TDM MSB Alignment Standard Timing Diagram

34.5.3 TDM PCM Standard

Short frame synchronization under the PCM standard requires that the WS signal changes one BCK clock cycle earlier than the SD signal on the falling edge of BCK, which means that the WS signal becomes valid one clock cycle before transferring the first bit of channel data and remains unchanged in this BCK clock cycle. SD signal line first transmits the most significant bit of audio data.

Compared with PCM standard, TDM PCM standard supports multiple channels. See Figure 34.5-3.

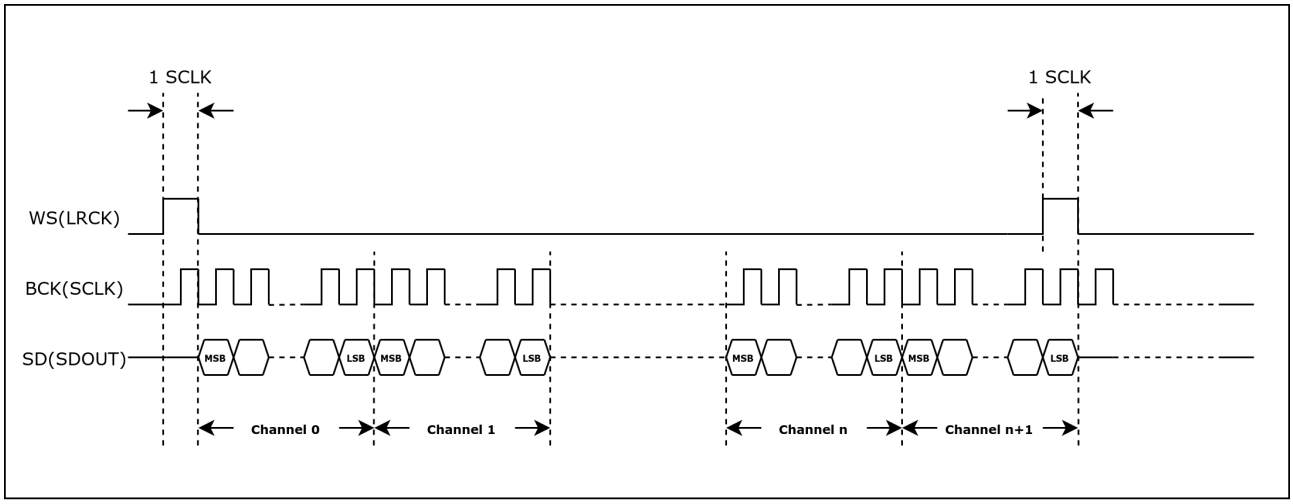


Figure 34.5-3. TDM PCM Standard Timing Diagram

34.5.4 PDM Standard

Under PDM standard, WS signal changes continuously during data transmission. The low-level and high-level of this signal indicates the left channel and right channel respectively. WS and SD signals change simultaneously. See Figure 34.5-4.

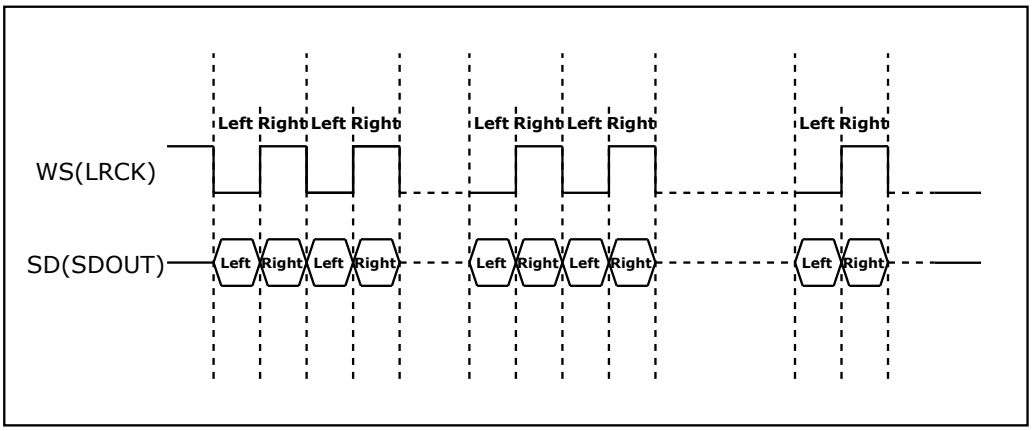


Figure 34.5-4. PDM Standard Timing Diagram

34.6 I2S TX/RX Clock

I2S*n*_TX/RX_CLK is the master clock of I2S*n* TX/RX unit, divided from:

- 40 MHz XTAL_CLK
- APLL_CLK with adjustable frequency
- I2S*n*_MCLK_in (external input clock)

The serial clock (BCK) of the I2S*n* TX/RX unit is divided from I2S*n*_TX/RX_CLK, as shown in Figure 34.6-1. HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_SRC_SEL is used to select clock source for TX/RX unit, and HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_EN to enable or disable the clock source.

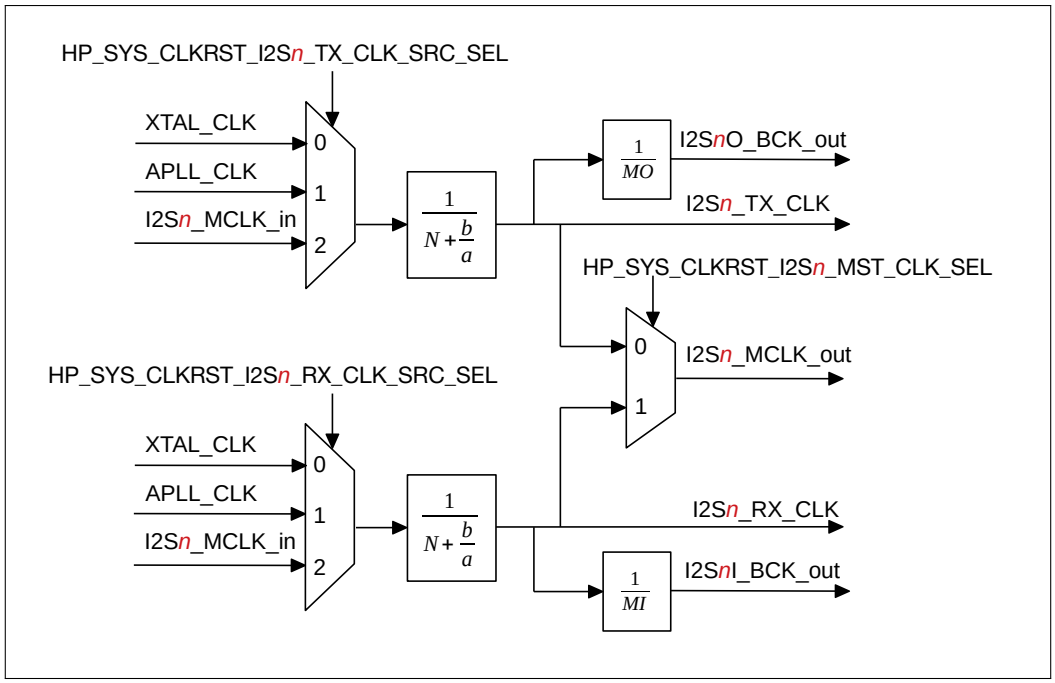


Figure 34.6-1. I2S*n* Clock Generator

The following formula shows the relation between I2S*n*_TX/RX_CLK frequency f_{I2Sn_TX/RX_CLK} and the divider

clock source frequency $f_{I2Sn\_CLK\_S}$:

$$f_{I2Sn\_TX/RX\_CLK} = \frac{f_{I2Sn\_CLK\_S}}{N + \frac{b}{a}}$$

N is an integer value between 2 and 256. The value of N is mapped to that of HP_SYS_CLKRST_I2Sn_TX/RX_DIV_N as follows:

- When HP_SYS_CLKRST_I2Sn_TX/RX_DIV_N = 0, N = 256;
- When HP_SYS_CLKRST_I2Sn_TX/RX_DIV_N = 1, N = 2;
- When HP_SYS_CLKRST_I2Sn_TX/RX_DIV_N has any other value, N = HP_SYS_CLKRST_I2Sn_TX/RX_DIV_N.

The values of "a" and "b" in fractional divider depend only on x, y, z, and yn1. The corresponding formulas are as follows:

- When b $<= \frac{a}{2}$, yn1 $= 0$, x $= floor([\frac{a}{b}]) - 1$, y $=$ a%b, z $=$ b;
- When b $> \frac{a}{2}$, yn1 $= 1$, x $= floor([\frac{a}{a-b}]) - 1$, y $=$ a%(a - b), z $=$ a - b.

The values of x, y, z, and yn1 are configured in HP_SYS_CLKRST_I2Sn_TX/RX_DIV_X, HP_SYS_CLKRST_I2Sn_TX/RX_DIV_Y, HP_SYS_CLKRST_I2Sn_TX/RX_DIV_Z, and HP_SYS_CLKRST_I2Sn_TX/RX_DIV_YN1.

To configure the integer divider, clear HP_SYS_CLKRST_I2Sn_TX/RX_DIV_X and HP_SYS_CLKRST_I2Sn_TX/RX_DIV_Z, then set HP_SYS_CLKRST_I2Sn_TX/RX_DIV_Y to 1.

Note:
Using fractional divider may introduce some clock jitter.

In master TX mode, the serial clock BCK for I2Sn TX unit is I2SnO_BCK_out divided from I2Sn_TX_CLK which is:

$$f_{I2SnO\_BCK\_out} = \frac{f_{I2Sn\_TX\_CLK}}{MO}$$

"MO" is an integer value:

$$MO = I2Sn\_TX\_BCK\_DIV\_NUM + 1$$

Note:
Note that I2S_TX_BCK_DIV_NUM must not be configured as 1.

In master RX mode, the serial clock BCK for I2Sn RX unit is I2SnI_BCK_out divided from I2Sn_RX_CLK, which is:

$$f_{I2SnI\_BCK\_out} = \frac{f_{I2Sn\_RX\_CLK}}{MI}$$

"MI" is an integer value:

$$MI = I2S\_RX\_BCK\_DIV\_NUM + 1$$

Note:

- I2S_RX_BCK_DIV_NUM must not be configured as 1.
- In I2S slave mode, make sure f_{I2S_TX/RX_CLK} >= 8 * f_{BCK}. The I2S*n* module can output I2S*n*_MCLK_out as the master clock for peripherals.

34.7 I2S Reset

The units and FIFOs in the I2S*n* module are reset by the following bits.

- I2S*n* TX/RX units: Reset by the bits I2S_TX_RESET and I2S_RX_RESET;
- I2S*n* TX/RX FIFO: Reset by the bits I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET.

Note:
The I2S*n* module clock must be configured first before the module and FIFO are reset.

34.8 I2S Master/Slave Mode

The ESP32-P4 I2S*n* module can operate as a master or a slave in half-duplex and full-duplex communications, depending on the configuration of I2S_RX_SLAVE_MOD and I2S_TX_SLAVE_MOD.

- I2S_TX_SLAVE_MOD
  - 0: Master TX mode
  - 1: Slave TX mode
- I2S_RX_SLAVE_MOD
  - 0: Master RX mode
  - 1: Slave RX mode

34.8.1 Master/Slave TX Mode

- I2S*n* works as a master transmitter:
  - Set I2S_TX_START to start transmitting data. When this bit is set, the TX unit keeps driving the clock signal and serial data.
  - If I2S_TX_STOP_EN is set and all the data in FIFO is transmitted, the master stops transmitting data and clock signals.
  - If I2S_TX_STOP_EN is cleared and all the data in FIFO is transmitted, meanwhile no new data is filled into FIFO, the TX unit keeps transmitting the last data frame and clock signal.
  - Master stops transmitting data when the bit I2S_TX_START is cleared.
- I2S*n* works as a slave transmitter:
  - Set I2S_TX_START. Wait for the master BCK clock to enable a transmit operation.

- If I2S_TX_STOP_EN is set and all the data in FIFO is transmitted, then the slave keeps transmitting zeros, till the master stops providing BCK signal.
- If I2S_TX_STOP_EN is cleared and all the data in FIFO is transmitted, meanwhile no new data is filled into FIFO, the TX unit keeps transmitting the last data frame.
- If I2S_TX_START is cleared, slave keeps transmitting zeros till the master stops providing BCK clock signal.

34.8.2 Master/Slave RX Mode

- I2S*n* works as a master receiver:
  - Set I2S_RX_START to start receiving data. When this bit is set, the RX unit keeps outputting clock signal and sampling input data.
  - Configure I2S_RX_STOP_MODE to control the suspension of data reception:
    * 0: The RX unit only suspends data reception when I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when I2S_RX_START is cleared or the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG. When I2S_RX_START is not cleared, data reception can be restarted by setting I2S_RX_RESET to 1.
    * 2: The RX unit suspends data reception when I2S_RX_START is cleared or GDMA RX FIFO is full. When I2S_RX_START is not cleared and the GDMA RX FIFO is no longer full, data reception can be restarted by setting I2S_RX_RESET to 1.
  - RX unit stops receiving data when the bit I2S_RX_START is cleared.
- I2S*n* works as a slave receiver:
  - Set I2S_RX_START. Wait for the master BCK signal to start receiving data.
  - Configure I2S_RX_STOP_MODE to control data reception suspension:
    * 0: The RX unit only suspends data reception when I2S_RX_START is cleared.
    * 1: The RX unit suspends data reception when the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG. When I2S_RX_START is not cleared, data reception can be restarted by setting I2S_RX_RESET to 1.
    * 2: The RX unit suspends data reception when I2S_RX_START is cleared or GDMA RX FIFO is full. When I2S_RX_START is not cleared and the GDMA RX FIFO is no longer full, data reception can be restarted by setting I2S_RX_RESET to 1.
  - The RX unit stops receiving data when the bit I2S_RX_START is cleared.

34.9 Transmitting Data

Note:
Updating the configuration described in this and subsequent sections requires to set I2S_TX_UPDATE accordingly to synchronize registers from APB clock domain to TX clock domain. For more detailed configurations, see Section 34.13.1.

In TX mode, I2S*n* first reads data through GDMA and transmits these data out via output signals according to the configured data mode and channel mode.

34.9.1 Data Format Control

Data format is controlled in the following phases:

- Phase I: Read data from memory and write it to TX FIFO;
- Phase II: Read the TX data from TX FIFO and convert the data according to the output data mode;
- Phase III: Clock out the TX data serially.

34.9.1.1 Bit Width Control of Channel Valid Data

The bit width of the valid data in each channel is determined by I2S_TX_BITS_MOD and I2S_TX_24_FILL_EN. For details, see the table below.

Table 34.9-1. Bit Width of Channel Valid Data

Channel Valid Data Width	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x*
	23	1
24	23	0
16	15	x
8	7	x

* "x" represents that this value is ignored.

34.9.1.2 Endian Control of Channel Valid Data

When I2S*n* reads data through GDMA, the data endian under various data width is controlled by I2S_TX_BIG_ENDIAN. Table 34.9-2 shows how I2S_TX_BIG_ENDIAN controls the reading of the data with different valid data widths of the channel.

Table 34.9-2. Endian of Channel Valid Data

Channel Valid Data Width	Original Data	Endian of Processed Data	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

Note:
B0, B1, B2, B3 each represents an 8-bit data, and the symbol {} indicates that the bytes are combined together. For

example, {B3, B2, B1, B0} represents a 32-bit data, wherein B0 represents bit 0-7, B1 represents bit 8-15, B2 represents bit 16-23, and B3 represents bit 24-31.

34.9.1.3 A-law/μ-law Compression and Decompression

ESP32-P4 I2S*n* compresses/decompresses the valid data into 32-bit by A-law or by μ-law. If the bit width of valid data is smaller than 32, zeros are filled to the extra high bits of the data to be compressed/decompressed by default.

Note:
Extra high bits here mean the bits[31: channel valid data width] of the data to be compressed/decompressed.

Configure I2S_TX_PCM_BYPASS:

- 0: Do not compress or decompress the data
- 1: Compress or decompress the data

Configure I2S_TX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the first phase of data format control is completed.

34.9.1.4 Bit Width Control of Channel TX Data

The TX data width in each channel is determined by I2S_TX_TDM_CHAN_BITS.

- If TX data width in each channel is larger than the valid data width, zeros will be filled to these extra bits. Configure I2S_TX_LEFT_ALIGN:
  - 0: The valid data is at the lower bits of TX data. Zeros are filled into higher bits of TX data;
  - 1: The valid data is at the higher bits of TX data. Zeros are filled into lower bits of TX data.
- If the TX data width in each channel is smaller than the valid data width, only the lower bits of valid data are sent out, and the higher bits are discarded.

At this point, the second phase of data format control is completed.

34.9.1.5 Bit Order Control of Channel Data

The data bit order in each channel is controlled by I2S_TX_BIT_ORDER:

- 0: Data is sent out from high bits to low bits;
- 1: Data is sent out from low bits to high bits.

At this point, the data format control is completed. Figure 34.9-1 shows the complete process of TX data format control.

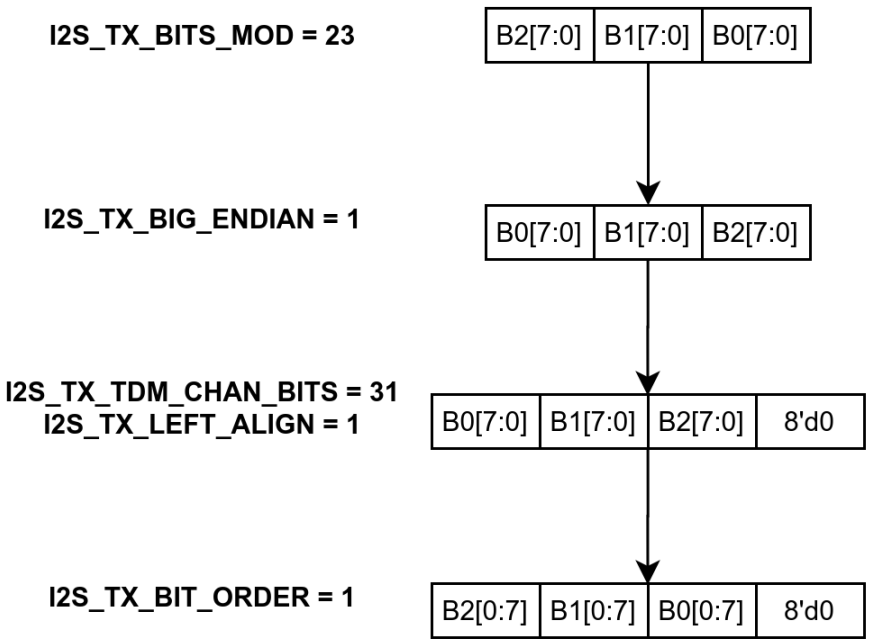


Figure 34.9-1. TX Data Format Control

34.9.2 Channel Mode Control

ESP32-P4 I2S*n* supports both TDM TX mode and PDM TX mode. Set I2S_TX_TDM_EN to enable TDM TX mode, or set I2S_TX_PDM_EN to enable PDM TX mode.

Note:
I2S_TX_TDM_EN and I2S_TX_PDM_EN must not be cleared or set simultaneously.

34.9.2.1 TDM TX Mode

In TDM TX mode, I2S*n* supports up to 16 channels to transmit data. The total number of TX channels in use is controlled by I2S_TX_TDM_TOT_CHAN_NUM. For example, If I2S_TX_TDM_TOT_CHAN_NUM is set to 5, six channels in total (channel 0 ~ 5) will be used to transmit data. See Figure 34.9-2.

In these TX channels, if I2S_TX_TDM_CHANn_EN is set to:

- 1: This channel transmits the channel data out;
- 0: The TX data to be sent by this channel is controlled by I2S_TX_CHAN_EQUAL:
  - 1: The data of the previous channel is sent out;
  - 0: The data stored in I2S_SINGLE_DATA is sent out.

Note:
Most stereo I2S*n* codecs can be controlled by setting the I2S*n* module into 2-channel mode under TDM standard.

In TDM TX master mode, WS signal is controlled by I2S_TX_WS_IDLE_POL and I2S_TX_TDM_WS_WIDTH:

- I2S_TX_WS_IDLE_POL: The default level of WS signal;
- I2S_TX_TDM_WS_WIDTH: The cycles the WS default level lasts for when transmitting all channel data.

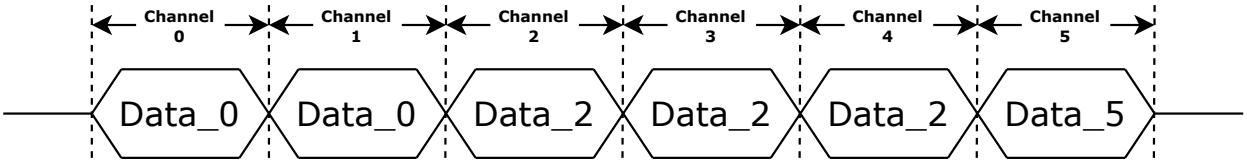
I2S_TX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

TDM Channel Configuration Example

In this example, the register configuration is as follows:

- I2S_TX_TDM_CHAN_NUM = 5, i.e., channel 0 ~ 5 are used to transmit data.
- I2S_TX_CHAN_EQUAL = 1, i.e., data of the previous channel will be transmitted if the bit I2S_TX_TDM_CHAN*n*_EN (*n* = 0 ~ 5) is cleared.
- I2S_TX_TDM_CHAN0/2/5_EN = 1, i.e., these channels transmit their channel data out.
- I2S_TX_TDM_CHAN1/3/4_EN = 0, i.e., these channels transmit the previous channel's data out.

Once the configuration is done, data is transmitted as follows.



I2S_TX_TDM_CHAN_NUM = 5; I2S_TX_CHAN_EQUAL = 1;
I2S_TX_TDM_CHAN0_EN = 1; I2S_TX_TDM_CHAN1_EN = 0; I2S_TX_TDM_CHAN2_EN = 1;
I2S_TX_TDM_CHAN3_EN = 0; I2S_TX_TDM_CHAN4_EN = 0; I2S_TX_TDM_CHAN5_EN = 1;

Figure 34.9-2. TDM Channel Control

34.9.2.2 PDM TX Mode

In PDM TX mode, I2S*n* supports both PDM raw data transmission and PCM-to-PDM data format conversion (the latter only supported by I2S*0*).

In PDM TX mode, fetching data through GDMA is controlled by I2S_TX_MONO and I2S_TX_MONO_FST_VLD. See Table 34.9-3. Configure the two bits according to the data stored in memory, be it the single-channel or dual-channel data.

Table 34.9-3. Data-Fetching Control in PDM Mode

Data-Fetching Control Option	Mode	I2S_TX_MONO	I2S_TX_MONO_FST_VLD
Post data-fetching request to GDMA at any edge of WS signal	Stereo mode	0	x
Post data-fetching request to GDMA only at the second half period of WS signal	Mono mode	1	0
Post data-fetching request to GDMA only at the first half period of WS signal	Mono mode	1	1

When I2S*n* is in PDM TX master mode, the default level of WS signal is controlled by I2S_TX_WS_IDLE_POL, and the WS signal frequency is half of the BCK signal frequency. The configuration of WS signal is similar to that of BCK signal. Please refer to Section 34.6 and Figure 34.6.

When **transmitting raw PDM data**, the I2S*n* channel mode is controlled by I2S_TX_CHAN_MOD and I2S_TX_WS_IDLE_POL. See the table below.

Table 34.9-4. I2S*n* Channel Control for Raw PDM Data

Channel Control Option	Left Channel	Right Channel	Mode Control Field[1]	Channel Select Bit[2]
Stereo mode	Transmit the left channel data	Transmit the right channel data	0	x
Mono mode	Transmit the left channel data	Transmit the left channel data	1	0
	Transmit the right channel data	Transmit the right channel data	1	1
	Transmit the right channel data	Transmit the right channel data	2	0
	Transmit the left channel data	Transmit the left channel data	2	1
	Transmit the value of "single"[3]	Transmit the right channel data	3	0
	Transmit the left channel data	Transmit the value of "single"	3	1
	Transmit the left channel data	Transmit the value of "single"	4	0
	Transmit the value of "single"	Transmit the right channel data	4	1

[1] I2S_TX_CHAN_MOD

[2] I2S_TX_WS_IDLE_POL

[3] The "single" value is equal to the value of I2S_SINGLE_DATA.

If the **PCM-to-PDM converter is enabled** for I2S*O* , the PCM data through GDMA is converted to PDM data and then output in PDM signal format. Configure I2S_PCM2PDM_CONV_EN to enable the converter. The register configuration for the PCM-to-PDM converter is as follows:

- Configure 1-line PDM output format or 1-/2-line DAC output mode as the table below:

Table 34.9-5. I2S*O* PCM-to-PDM Data Output

Channel Output Format	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
1-line PDM output format[1]	0	x
1-line DAC output format[2]	1	0
2-line DAC output format	1	1

Note:

1. In PDM output format, SD data of two channels is sent out in one WS period.
2. In DAC output format, SD data of one channel is sent out in one WS period.
3. 1-line PDM/DAC output format uses I2S*n*O_Data_out as the data output line.
4. 2-line DAC output format uses I2S*n*O_Data_out and I2S*n*O_Data1_out as data output lines.
5. In terms of application, the PDM output format is targeted at applications that require clock signals and can decode PDM format codecs, while the DAC output format is targeted at applications that do not rely on clock signals, do not have PDM format codecs, and can recover analog waveforms directly through low-pass filtering.

- Configure sampling frequency and upsampling rate as below:

When the I2S0 PCM-to-PDM converter is enabled, PDM clock frequency is equal to BCK frequency. The relation of sampling frequency ($f_{Sampling}$, the sampling frequency of PCM data) and BCK frequency (the sampling frequency of PDM data) is as follows:

$$f_{Sampling} = \frac{f_{BCK}}{OSR}$$

Upsampling rate (OSR) is related to I2S_TX_PDM_SINC_OSR2 as follows:

$$OSR = I2S\_TX\_PDM\_SINC\_OSR2 \times 64$$

Sampling frequency $f_{Sampling}$ is related to I2S_TX_PDM_FS as follows:

$$f_{Sampling} = I2S\_TX\_PDM\_FS \times 100$$

Configure the registers according to the needed sampling frequency, upsampling rate, and PDM clock frequency.

PDM Channel Configuration Example

In this example of I2S0, the register configuration is as follows.

- I2S_PCM2PDM_CONV_EN = 0, i.e., transmit the raw PDM data.
- I2S_TX_MONO = 0, i.e., data is fetched from memory via GDMA in both the high and low levels of WS.
- I2S_TX_CHAN_MOD = 2, i.e., mono mode is selected.
- I2S_TX_WS_IDLE_POL = 1, i.e., both the left and right channels transmit the left channel data, and the right channel data will be discarded.

Once the configuration is done, assume that the data in memory **after data format control** is:

Left	Right	Left	Right	...	Left	Right

Note:

1. The data above refers to the processed data after data format control instead of the original data.
2. "Left" and "Right" represent channel data, and their bit widths are channel valid data width. Please refer to Section 34.9.1.

Then the channel data is transmitted after channel mode control as follows.

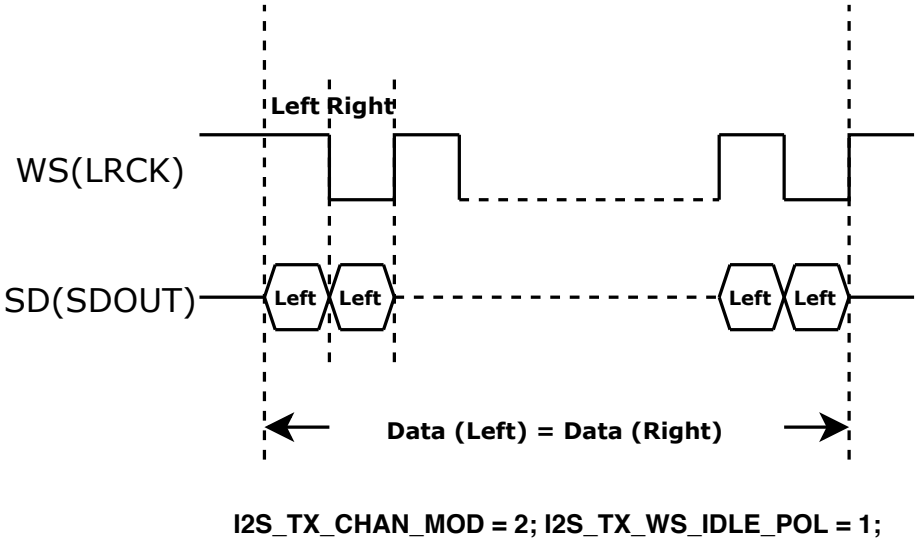


Figure 34.9-3. PDM Channel Control Example

34.9.3 Synchronous Counter

ESP32-P4 I2S*n* TX unit contains synchronous counters to reflect the current I2S*n* data transmission status. The counters can be used to detect I2S*n* transmit data errors or to synchronize data when multiple devices are using I2S*n* for data transmission.

ESP32-P4 I2S*n* contains two types of synchronous counters:

- I2S_TX_FIFO_CNT
  - Each time the I2S*n* TX FIFO reads data, the counter is incremented by 1.
  - The counter can be reset by setting I2S_TX_FIFO_CNT_RST.
  - The counter is 31-bit wide and is automatically reset to zero on overflow.
- I2S_TX_BCK_CNT
  - Each time I2S*n* TX transmits a BCK cycle, the counter is incremented by 1.
  - The counter can be reset by setting I2S_TX_BCK_CNT_RST.
  - The counter is 31-bit wide and is automatically reset to zero on overflow.

Note:

- Each time the I2S*n* TX FIFO transmits data, a channel data is sent. Therefore, when I2S*n* transmits the data correctly, I2S_TX_BCK_CNT = I2S_TX_FIFO_CNT * the bit width of channel TX data (see Section 34.9.1.4).
- When the I2S*n* TX unit transmits the data, there is some delay between fetching data from the FIFO and transmitting it to the line, so the value of I2S_TX_BCK_CNT might be a little different from that of I2S_TX_FIFO_CNT.

Application scenarios

1. I2S TX synchronization for multiple devices
   When multiple devices transmit data through the I2S TX at the same time, desynchronization may occur after a long period of time due to small frequency differences in the clock sources. In this case, read the

I2S_TX_FIFO_CNT value of each device to determine the location of the audio data being sent to perform synchronization.

2. I2S TX stability check
   When the device is in an application scenario with high GDMA usage, there may be numbers missing in I2S TX occasionally due to multiplexing arbitration. In this case, check the ratio of I2S_TX_BCK_CNT to I2S_TX_FIFO_CNT to confirm the stability of the device when running I2S TX.

34.10 Receiving Data

In RX mode, I2S*n* first reads data from the peripheral interface and then stores the data in memory via GDMA according to the configured channel mode and data mode.

34.10.1 Channel Mode Control

ESP32-P4 I2S*n* supports both TDM RX mode and PDM RX mode. Set I2S_RX_TDM_EN to enable TDM RX mode, or set I2S_RX_PDM_EN to enable PDM RX mode.

Note:
I2S_RX_TDM_EN and I2S_RX_PDM_EN must not be cleared or set simultaneously.

34.10.1.1 TDM RX Mode

In TDM RX mode, I2S*n* supports up to 16 channels to input data. The total number of RX channels in use is controlled by I2S_RX_TDM_TOT_CHAN_NUM. For example, if I2S_RX_TDM_TOT_CHAN_NUM is set to 5, channel 0 ~ 5 will be used to receive data.

In these RX channels, if I2S_RX_TDM_CHAN*n*_EN is set to:

- 1: The channel data is valid and will be stored into RX FIFO;
- 0: The channel data is invalid and will not be stored into RX FIFO.

In TDM master mode, WS signal is controlled by I2S_RX_WS_IDLE_POL and I2S_RX_TDM_WS_WIDTH.

- I2S_RX_WS_IDLE_POL: The default level of WS signal;
- I2S_RX_TDM_WS_WIDTH: The cycles the WS default level lasts for when receiving all channel data.

I2S_RX_HALF_SAMPLE_BITS x 2 is equal to the BCK cycles in one WS period.

34.10.1.2 PDM RX Mode

In PDM RX mode, I2S*n* supports both PDM raw data reception and PDM-to-PCM data format conversion (the latter only supported by I2S*0*).

In PDM RX mode, I2S*n* converts the serial data from channels to the data to be entered into memory.

In PDM RX master mode, the default level of the WS signal is controlled by I2S_RX_WS_IDLE_POL. WS frequency is half of the BCK frequency. The configuration of the BCK signal is similar to that of WS signal as described in Section 34.6. Note, in PDM RX mode, the value of I2S_RX_HALF_SAMPLE_BITS must be same as that of I2S_RX_BITS_MOD.

When the **PDM-to-PCM converter is enabled** for I2S*0*, the received PDM data is converted to PCM data and controlled according to the data mode. Configure I2S_RX_PDM2PCM_EN to enable this converter. The register configuration for PDM-to-PCM converter is as follows:

- Configure sampling frequency and downsampling rate.
  When I2S*0* PDM-to-PCM converter is enabled, PDM clock frequency is:
  - in master mode: PDM clock frequency is equal to BCK frequency.
  - in slave mode: PDM clock is provided by external device.

  The sampling frequency ($f_{Sampling}$) is related to PDM clock frequency as follows:

$$f_{Sampling} = \frac{f_{PDM}}{DSR}$$

  Downsampling rate (DSR) is related to I2S_RX_PDM_SINC_DSR_16_EN as follows:

$$DSR = I2S\_RX\_PDM\_SINC\_DSR\_16\_EN \times 64$$

  Configure the registers according to needed master/slave mode, sampling frequency, and downsampling rate.
- Configure valid channels.
  When the PDM-to-PCM converter is enabled, input signals from eight channels are supported at most. See Table 34.10-1 for the register configuration and related channels.

Table 34.10-1. PDM-to-PCM Data Input

Input Data Signal	Channel	Enable Register
I2S*0*I_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN0_EN
	Right channel	I2S_RX_TDM_PDM_CHAN1_EN
I2S*0*I1_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN2_EN
	Right channel	I2S_RX_TDM_PDM_CHAN3_EN
I2S*0*I2_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN4_EN
	Right channel	I2S_RX_TDM_PDM_CHAN5_EN
I2S*0*I3_Data_in	Left channel	I2S_RX_TDM_PDM_CHAN6_EN
	Right channel	I2S_RX_TDM_PDM_CHAN7_EN

34.10.2 Data Format Control

The data format of I2S*n* is controlled in the following phases:

- Phase I: Serial input data is converted into the data to be saved to RX FIFO;
- Phase II: The data is read from RX FIFO and converted according to the input data mode.

34.10.2.1 Bit Order Control of Channel Data

The data bit order in each channel is controlled by I2S_RX_BIT_ORDER:

- 0: Serial data is entered from high bits to low bits;
- 1: Serial data is entered from low bits to high bits.

At this point, the first phase of data format control is completed.

34.10.2.2 Bit Width Control of Channel Storage (Valid) Data

The storage data width in each channel is controlled by I2S_RX_BITS_MOD and I2S_RX_24_FILL_EN. See the table below.

Table 34.10-2. Channel Storage Data Width

Channel Storage Data Width	I2S_RX_BITS_MOD	I2S_RX_24_FILL_EN
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

34.10.2.3 Bit Width Control of Channel RX Data

The RX data width in each channel is determined by I2S_RX_TDM_CHAN_BITS.

- If the storage data width in each channel is smaller than the received (RX) data width, then only the bits within the storage data width is saved into memory. Configure I2S_RX_LEFT_ALIGN to:
  - 0: Only the lower bits of the received data within the storage data width is stored to memory;
  - 1: Only the higher bits of the received data within the storage data width is stored to memory.
- If the received data width is smaller than the storage data width in each channel, the higher bits of the received data will be filled with zeros and then the data is saved to memory.

34.10.2.4 Endian Control of Channel Storage Data

The received data is then converted into storage data (to be stored to memory) after some processing, such as discarding extra bits or filling zeros in missing bits. The endian of the storage data is controlled by I2S_RX_BIG_ENDIAN under various data width. See the table below.

Table 34.10-3. Channel Storage Data Endian

Channel Storage Data Width	Original Data	Endian of Processed Data	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

34.10.2.5 A-law/μ-law Compression and Decompression

ESP32-P4 I2S*n* compresses/decompresses the storage data in 32-bit by A-law or by μ-law. By default, zeros are filled into high bits.

Configure I2S_RX_PCM_BYPASS:

- 0: Do not compress or decompress the data
- 1: Compress or decompress the data

Configure I2S_RX_PCM_CONF:

- 0: Decompress the data using A-law
- 1: Compress the data using A-law
- 2: Decompress the data using μ-law
- 3: Compress the data using μ-law

At this point, the data format control is completed. Data then is stored into memory via GDMA.

34.11 Event Task Matrix Feature

ESP32-P4 I2S*n* supports the Event Task Matrix (ETM) function, which allows I2S*n*'s ETM tasks to be triggered by any peripherals' ETM events, or I2S*n*'s ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to I2S*n*. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

I2S*n* can receive the following ETM tasks:

- I2S*n*_TASK_START_TX: Enables I2S*n* TX for data transfer.
- I2S*n*_TASK_START_RX: Enables I2S*n* RX for data transfer.
- I2S*n*_TASK_STOP_TX: Stops I2S*n* TX data transfer.
- I2S*n*_TASK_STOP_RX: Stops I2S*n* RX data transfer.

I2S*n* can generate the following ETM events:

- I2S*n*_EVT_TX_DONE: Indicates that I2S*n* TX has completed data transmission.
- I2S*n*_EVT_RX_DONE: Indicates that I2S*n* RX has completed data reception.
- I2S*n*_EVT_X_WORDS_SENT: Indicates that the word number sent by I2S*n* TX is equal to or larger than the value set by I2S_ETM_TX_SEND_WORD_NUM.
- I2S*n*_EVT_X_WORDS_RECEIVED: Indicates that the word number received by I2S*n* RX is equal to or larger than the value set by I2S_ETM_RX_RECEIVE_WORD_NUM.

In practical applications, I2S*n*'s ETM events can trigger its own ETM tasks. For example, the I2S*n*_EVT_X_WORDS_SENT event can trigger the I2S*n*_TASK_STOP_TX task, and in this way stop the I2S*n* operation through ETM.

34.12 I2S Interrupts

ESP32-P4's I2S*n* can generate the I2S*n*_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from I2S*n* that can generate the above interrupt signal(s) as follows:

- I2S*n*_TX_HUNG_INT: Triggered when data transmission is timed out. For example, if the I2S*n* module is configured as TX slave mode, but the master does not provide BCK or WS signals for a long time (specified in I2S_LC_HUNG_CONF_REG), this interrupt will be triggered.

- I2S*n*_RX_HUNG_INT: Triggered when the data reception is timed out. For example, if the I2S*n* module is configured as RX slave mode, but the master does not transmit data for a long time (specified in I2S_LC_HUNG_CONF_REG), this interrupt will be triggered.
- I2S*n*_TX_DONE_INT: Triggered when data transmission is completed.
- I2S*n*_RX_DONE_INT: Triggered when the data reception is completed.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 34.14 *Register Summary*.

34.13 Software Configuration Process

34.13.1 Configure I2S as TX Mode

Follow the steps below to configure I2S*n* as TX mode via software:

1. Configure the clock as described in Section 34.6.
2. Configure signal pins according to Table 34.4-1.
3. Select the mode needed by configuring I2S_TX_SLAVE_MOD.
   - 0: Master TX mode
   - 1: Slave TX mode
4. Set needed TX data mode and TX channel mode as described in Section 34.9, and then set I2S_TX_UPDATE.
5. Reset TX unit and TX FIFO as described in Section 34.7.
6. Enable corresponding interrupts. See Section 34.12.
7. Configure GDMA outlink.
8. Set I2S_TX_STOP_EN if needed. For more information, please refer to Section 34.8.1.
9. Start transmitting data:
   - In master mode, wait till I2S*n* slave gets ready, then set I2S_TX_START to start transmitting data;
   - In slave mode, set I2S_TX_START. When the I2S*n* master supplies BCK and WS signals, I2S*n* slave starts transmitting data.
10. Wait for the interrupt signals set in Step 6, or check whether the transfer is completed by querying I2S_TX_IDLE:
    - 0: Transmitter is working;
    - 1: Transmitter is in idle state.
11. Clear I2S_TX_START to stop data transfer.

34.13.2 Configure I2S as RX Mode

Follow the steps below to configure I2S*n* as RX mode via software:

1. Configure the clock as described in Section 34.6.
2. Configure signal pins according to Table 34.4-1.
3. Select the mode needed by configuring I2S_RX_SLAVE_MOD.
   - 0: Master RX mode
   - 1: Slave RX mode
4. Set needed RX data mode and RX channel mode as described in Section 34.10, and then set I2S_RX_UPDATE.
5. Reset the RX unit and its FIFO according to Section 34.7.
6. Enable the corresponding interrupts. See Section 34.12.
7. Configure GDMA inlink and set the length of RX data by configuring I2S_RX_EOF_NUM_REG.
8. Start receiving data:
   - In master mode, when the slave is ready, set I2S_RX_START to start receiving data.
   - In slave mode, set I2S_RX_START to start receiving data when BCK and WS signals are received from the master.
9. The received data is then stored to the specified memory address according to the configuration of GDMA. Then the corresponding interrupt set in step 6 is generated.

34.14 Register Summary

The addresses in this section are relative to I2S*n* base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Interrupt registers			
I2S_INT_RAW_REG	I2S interrupt raw register	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S interrupt status register	0x0010	RO
I2S_INT_ENA_REG	I2S interrupt enable register	0x0014	R/W
I2S_INT_CLR_REG	I2S interrupt clear register	0x0018	WT
RX control and configuration registers			
I2S_RX_CONF_REG	I2S RX configuration register	0x0020	varies
I2S_RX_CONF1_REG	I2S RX configuration register 1	0x0028	R/W
I2S_RX_PDM2PCM_CONF_REG	I2S RX PDM-to-PCM configuration register	0x0048	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM mode control register	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX data number control register	0x0064	R/W
TX control and configuration registers			
I2S_TX_CONF_REG	I2S TX configuration register	0x0024	varies
I2S_TX_CONF1_REG	I2S TX configuration register 1	0x002C	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM configuration register (for I2S*0* only)	0x0040	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM configuration register (for I2S*0* only)	0x0044	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM mode control register	0x0054	R/W
RX clock and timing register			
I2S_RX_TIMING_REG	I2S RX timing control register (some fields for I2S*0* only)	0x0058	R/W
TX clock and timing register			
I2S_TX_TIMING_REG	I2S TX timing control register (some fields for I2S*0* only)	0x005C	R/W
Control and configuration registers			
I2S_LC_HUNG_CONF_REG	I2S timeout configuration register	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single data register	0x0068	R/W
TX status register			
I2S_STATE_REG	I2S TX status register	0x006C	RO
ETM register			
I2S_ETM_CONF_REG	I2S ETM configuration register	0x0070	R/W
Sync counter registers			
I2S_FIFO_CNT_REG	I2S sync counter register	0x0074	varies
I2S_BCK_CNT_REG	I2S sync counter register	0x0078	varies
Clock registers			
I2S_CLK_GATE_REG	Clock gate register	0x007C	R/W

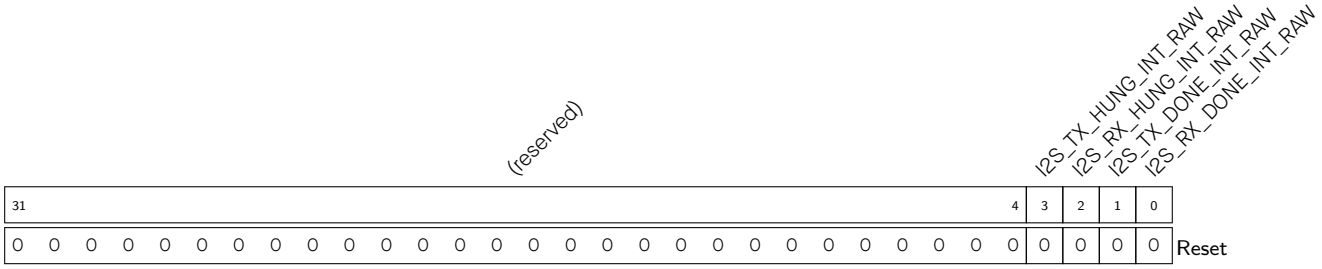
Name	Description	Address	Access
Version register			
I2S_DATE_REG	Version control register	0x0080	R/W

34.15 Registers

The addresses in this section are relative to I2S*n* base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 34.1. I2S_INT_RAW_REG (0x000C)



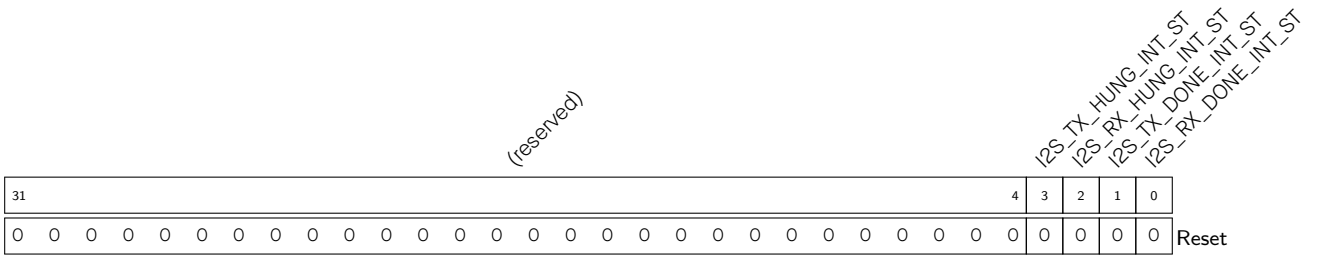
I2S_RX_DONE_INT_RAW The raw interrupt status of the I2S_RX_DONE_INT interrupt. (R/SS/WTC)

I2S_TX_DONE_INT_RAW The raw interrupt status of the I2S_TX_DONE_INT interrupt. (R/SS/WTC)

I2S_RX_HUNG_INT_RAW The raw interrupt status of the I2S_RX_HUNG_INT interrupt. (R/SS/WTC)

I2S_TX_HUNG_INT_RAW The raw interrupt status of the I2S_TX_HUNG_INT interrupt. (R/SS/WTC)

Register 34.2. I2S_INT_ST_REG (0x0010)



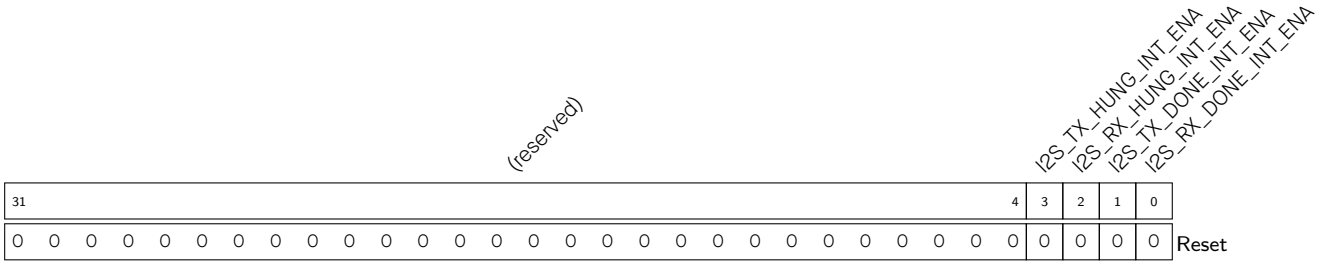
I2S_RX_DONE_INT_ST The masked interrupt status of the I2S_RX_DONE_INT interrupt. (RO)

I2S_TX_DONE_INT_ST The masked interrupt status of the I2S_TX_DONE_INT interrupt. (RO)

I2S_RX_HUNG_INT_ST The masked interrupt status of the I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_HUNG_INT_ST The masked interrupt status of the I2S_TX_HUNG_INT interrupt. (RO)

Register 34.3. I2S_INT_ENA_REG (0x0014)



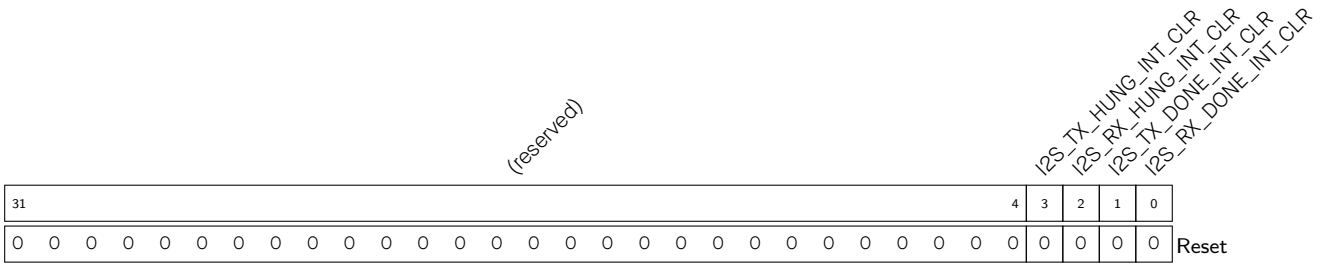
I2S_RX_DONE_INT_ENA Write 1 to enable the I2S_RX_DONE_INT interrupt. (R/W)

I2S_TX_DONE_INT_ENA Write 1 to enable the I2S_TX_DONE_INT interrupt. (R/W)

I2S_RX_HUNG_INT_ENA Write 1 to enable the I2S_RX_HUNG_INT interrupt. (R/W)

I2S_TX_HUNG_INT_ENA Write 1 to enable the I2S_TX_HUNG_INT interrupt. (R/W)

Register 34.4. I2S_INT_CLR_REG (0x0018)



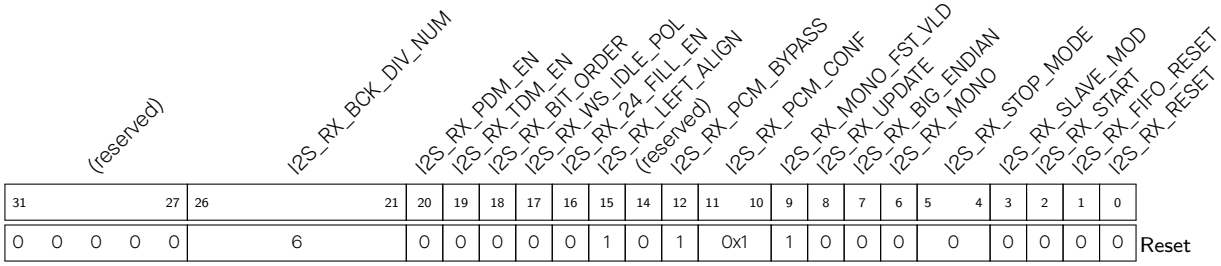
I2S_RX_DONE_INT_CLR Write 1 to clear the I2S_RX_DONE_INT interrupt. (WT)

I2S_TX_DONE_INT_CLR Write 1 to clear the I2S_TX_DONE_INT interrupt. (WT)

I2S_RX_HUNG_INT_CLR Write 1 to clear the I2S_RX_HUNG_INT interrupt. (WT)

I2S_TX_HUNG_INT_CLR Write 1 to clear the I2S_TX_HUNG_INT interrupt. (WT)

Register 34.5. I2S_RX_CONF_REG (0x0020)



I2S_RX_RESET Configures whether to reset RX unit.

0: No effect

1: Reset

(WT)

I2S_RX_FIFO_RESET Configures whether to reset RX FIFO.

0: No effect

1: Reset

(WT)

I2S_RX_START Configures whether to start receiving data.

0: No effect

1: Start

(R/W/SC)

I2S_RX_SLAVE_MOD Configures whether to enable slave RX mode.

0: Enable master mode

1: Enable slave mode

(R/W)

I2S_RX_STOP_MODE Configures when I2S RX stop data reception.

0: Only stops when I2S_RX_START is cleared

1: Stops when I2S_RX_START is cleared or the number of received bytes is greater than the value configured in I2S_RX_EOF_NUM_REG

2: Stops when I2S_RX_START is cleared or GDMA RX FIFO is full

(R/W)

I2S_RX_MONO Configures whether to enable RX unit in mono mode.

0: Disable

1: Enable

(R/W)

I2S_RX_BIG_ENDIAN Configures I2S RX byte endian.

0: Low address data is saved to low address

1: Low address data is saved to high address

(R/W)

Continued on the next page...

Register 34.5. I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

I2S_RX_UPDATE Configures whether to update I2S RX registers from APB clock domain to I2S RX clock domain. This bit will be cleared by hardware after the register update is done.
0: No effect
1: Update
(R/W/SC)

I2S_RX_MONO_FST_VLD Configures the valid data channel in I2S RX mono mode.
0: The second channel data valid
1: The first channel data valid
(R/W)

I2S_RX_PCM_CONF Configures I2S RX compress/decompress mode.
0 (atol): A-law decompress
1 (ltoa): A-law compress
2 (utol): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_RX_PCM_BYPASS Configures whether to bypass the Compress/Decompress units for received data.
0: No effect
1: Bypass
(R/W)

I2S_RX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: The WS signal changes one BCK clock earlier
(R/W)

I2S_RX_LEFT_ALIGN Configures I2S RX alignment mode.
0: Right alignment mode
1: Left alignment mode
(R/W)

Continued on the next page...

Register 34.5. I2S_RX_CONF_REG (0x0020)

Continued from the previous page...

I2S_RX_24_FILL_EN Configures the bit number that the 24-bit channel data is stored to.
0: Store 24-bit channel data to 24 bits
1: Store 24-bit channel data to 32 bits (Extra bits are filled with zeros)
(R/W)

I2S_RX_WS_IDLE_POL Configures the relationship between WS level and which channel data to receive.
0: WS remains low when receiving left channel data and high when receiving right channel data
1: WS remains high when receiving left channel data and low when receiving right channel data
(R/W)

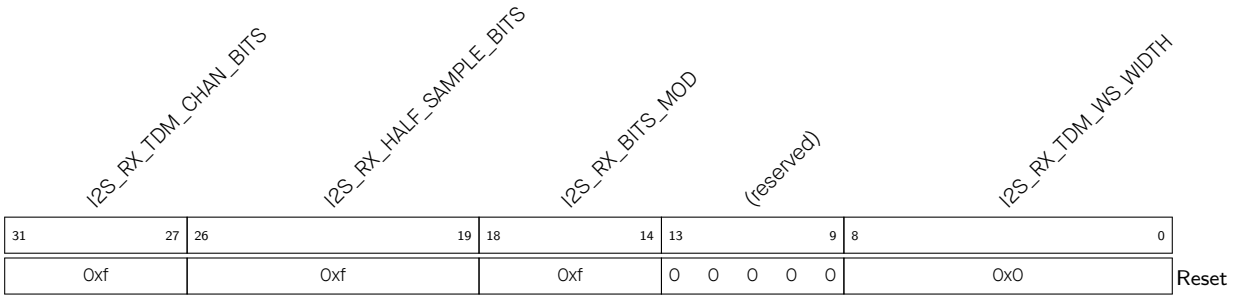
I2S_RX_BIT_ORDER Configures I2S RX bit order.
0: The highest bit is received first
1: The lowest bit is received first
(R/W)

I2S_RX_TDM_EN Configures whether to enable I2S TDM RX mode.
0: Disable
1: Enable
(R/W)

I2S_RX_PDM_EN Configures whether to enable I2S PDM RX mode.
0: Disable
1: Enable
(R/W)

I2S_RX_BCK_DIV_NUM Configures the divider of BCK in RX mode. Note this divider must not be configured to 1. (R/W)

Register 34.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH Configures the width of I2S*n*I_WS_out (WS default level) at idle level in TDM mode. Width of I2S*n*I_WS_out at idle level in TDM mode = (I2S_RX_TDM_WS_WIDTH[8:0] +1) x T_BCK. (R/W)

I2S_RX_BITS_MOD Configures the valid data bit length of I2S RX channel.

7: All the valid channel data is in 8-bit mode

15: All the valid channel data is in 16-bit mode

23: All the valid channel data is in 24-bit mode

31: All the valid channel data is in 32-bit mode

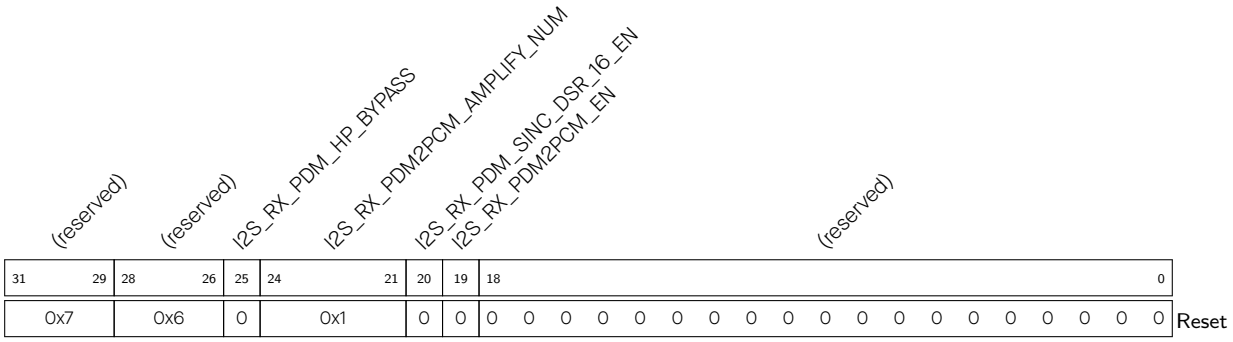
Other values are invalid.

(R/W)

I2S_RX_HALF_SAMPLE_BITS Configures I2S RX sample bits. BCK cycles in one WS period = I2S_RX_HALF_SAMPLE_BITS x 2. (R/W)

I2S_RX_TDM_CHAN_BITS Configures RX bit number for each channel in TDM mode. Bit number expected = I2S_RX_TDM_CHAN_BITS + 1. (R/W)

Register 34.7. I2S_RX_PDM2PCM_CONF_REG (for I2S0 only) (0x0048)



I2S_RX_PDM2PCM_EN Configures whether to enable the PDM-to-PCM converter in RX mode.

0: Disable

1: Enable

(R/W)

I2S_RX_PDM_SINC_DSR_16_EN Configures the downsampling rate of PDM RX filter group 1 module.

0: 64

1: 128

(R/W)

I2S_RX_PDM2PCM_AMPLIFY_NUM Configures the PDM-to-PCM RX amplify number. PCM data will be multiplied by this value before outputting.

(R/W)

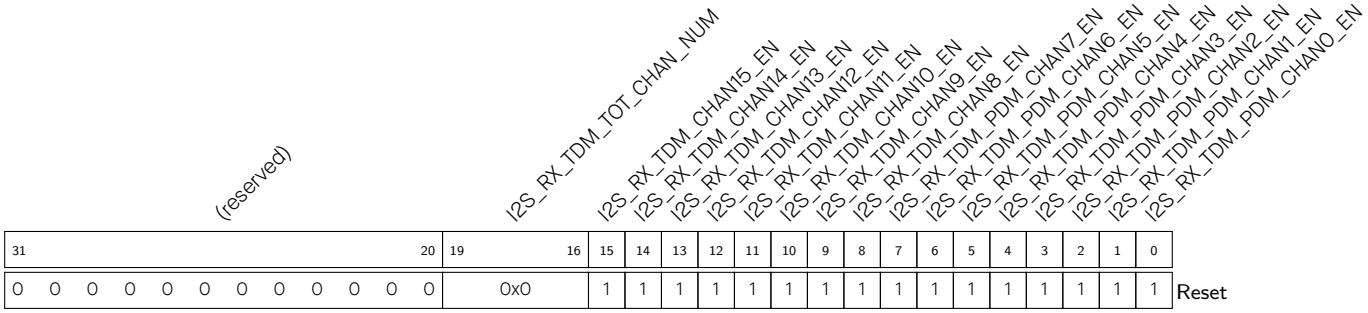
I2S_RX_PDM_HP_BYPASS Configures whether PDM-to-PCM RX bypasses the HP filter.

0: Not bypass

1: Bypass

(R/W)

Register 34.8. I2S_RX_TDM_CTRL_REG (0x0050)



I2S_RX_TDM_PDM_CHAN*n*_EN (*n*: 0-7) Configures whether to enable the valid data input of I2S RX TDM or PDM channel *n*.

0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

I2S_RX_TDM_CHAN*n*_EN (*n*: 8-15) Configures whether to enable the valid data input of I2S RX TDM channel *n*.

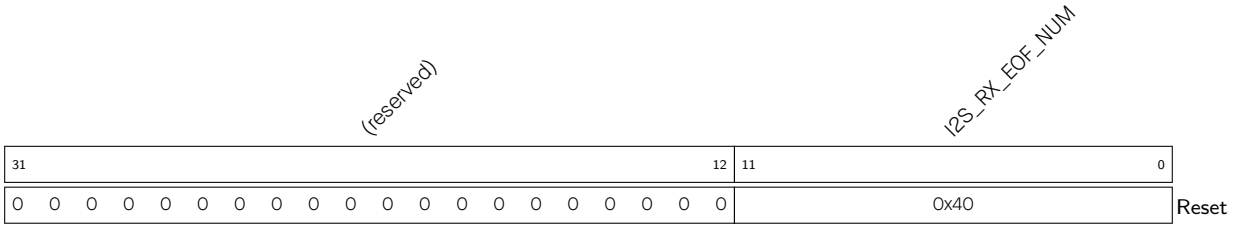
0: Disable. Channel *n* only inputs 0

1: Enable

(R/W)

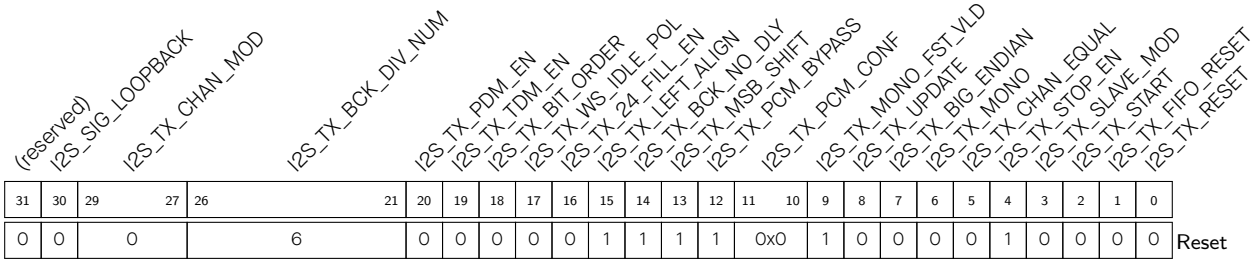
I2S_RX_TDM_TOT_CHAN_NUM Configures the total number of channels in use in I2S RX TDM mode. Total channel number in use = I2S_RX_TDM_TOT_CHAN_NUM + 1. (R/W)

Register 34.9. I2S_RXEOF_NUM_REG (0x0064)



I2S_RX_EOF_NUM Configures the bit length of RX data. Bit length of RX data = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1). Once the received data reaches such bit length, an AHB_DMA_IN_SUC_EOF_CH*n*_INT interrupt is triggered in the configured GDMA RX channel. (R/W)

Register 34.10. I2S_TX_CONF_REG (0x0024)



I2S_TX_RESET Configures whether to reset TX unit.

0: No effect

1: Reset

(WT)

I2S_TX_FIFO_RESET Configures whether to reset TX FIFO.

0: No effect

1: Reset

(WT)

I2S_TX_START Configures whether to start transmitting data.

0: No effect

1: Start

(R/W/SC)

I2S_TX_SLAVE_MOD Configures whether to enable slave TX mode.

0: Enable master mode

1: Enable slave mode

(R/W)

I2S_TX_STOP_EN Configures whether to stop outputting the BCK signal and the WS signal when TX FIFO is empty.

0: No effect

1: Stop

(R/W)

I2S_TX_CHAN_EQUAL Configures whether to equalize left channel data and right channel data in I2S TX mono mode or TDM mode.

0: The I2S_SINGLE_DATA is invalid channel data in I2S TX mono mode or TDM mode

1: The left channel data is equal to right channel data in I2S TX mono mode or TDM mode

(R/W)

I2S_TX_MONO Configures whether to enable TX unit in mono mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 34.10. I2S_TX_CONF_REG (0x0024)

Continued from the previous page...

I2S_TX_BIG_ENDIAN Configures I2S TX byte endian.
0: Low address with low address value
1: Low address value to high address
(R/W)

I2S_TX_UPDATE Configures whether to update I2S TX registers from APB clock domain to I2S TX clock domain.
0: No effect
1: Update
This bit will be cleared by hardware after update register done.
(R/W/SC)

I2S_TX_MONO_FST_VLD Configures the valid data channel in I2S TX mono mode.
0: The second channel data valid
1: The first channel data valid
(R/W)

I2S_TX_PCM_CONF Configures the I2S TX compress/decompress mode.
0 (atol): A-law decompress
1 (ltoa): A-law compress
2 (utol): μ-law decompress
3 (ltou): μ-law compress
(R/W)

I2S_TX_PCM_BYPASS Configures whether to bypass Compress/Decompress units for transmitted data.
0: No effect
1: Bypass
(R/W)

I2S_TX_MSB_SHIFT Configures the timing between the WS signal and the MSB of data.
0: Align at the rising edge
1: WS signal changes one BCK clock earlier
(R/W)

I2S_TX_BCK_NO_DLY Configures the source of the BCK rising and falling edges in master mode.
0: Rising and falling edges are constructed based on the BCK input from I2S*n*O_BCK_in
1: Rising and falling edges are constructed by dividing the clock of I2S*n* TX
(R/W)

I2S_TX_LEFT_ALIGN Configures I2S TX alignment mode.
0: Right alignment mode
1: Left alignment mode
(R/W)

Continued on the next page...

Register 34.10. I2S_TX_CONF_REG (0x0024)

Continued from the previous page...

I2S_TX_24_FILL_EN Configures the bit number that the 24 channel bits are stored to.
0: Store 24-bit channel data to 24 bits
1: Store 24-bit channel data to 32 bits (Extra bits are filled with zeros)
(R/W)

I2S_TX_WS_IDLE_POL Configures the relationship between WS and which channel data to transmit.
0: WS remains low when transmitting left channel data and high when transmitting right channel data
1: WS remains high when transmitting left channel data and low when transmitting right channel data
(R/W)

I2S_TX_BIT_ORDER Configures I2S TX bit endian.
0: The highest bit is sent first
1: The lowest bit is sent first
(R/W)

I2S_TX_TDM_EN Configures whether to enable I2S TDM TX mode.
0: Disable
1: Enable
(R/W)

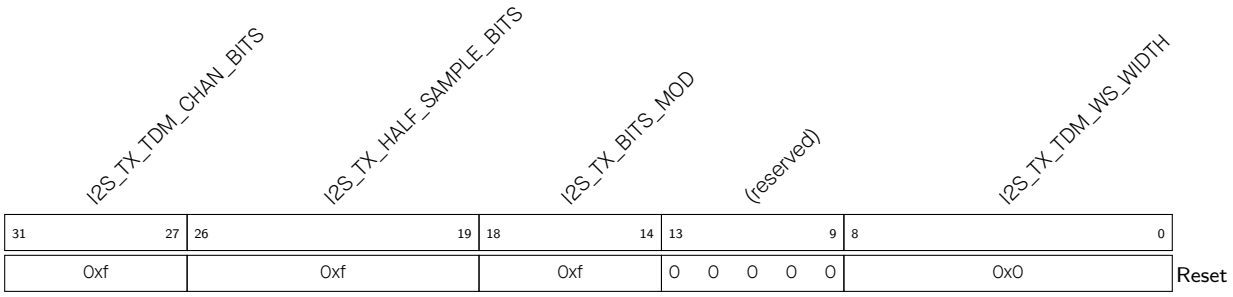
I2S_TX_PDM_EN Configures whether to enable I2S PDM TX mode.
0: Disable
1: Enable
(R/W)

I2S_TX_BCK_DIV_NUM Configures the divider of BCK in TX mode. Note this divider must not be configured to 1. (R/W)

I2S_TX_CHAN_MOD Configures I2S TX channel mode. For more information, see Table 34.9-4. (R/W)

I2S_SIG_LOOPBACK Configures whether to enable TX unit and RX unit sharing the same WS and BCK signals.
0: Disable
1: Enable
(R/W)

Register 34.11. I2S_TX_CONF1_REG (0x002C)



I2S_TX_TDM_WS_WIDTH Configures the width of I2S*n*O_WS_out (WS default level) at idle level in TDM mode. The width of I2S*n*O_WS_out at idle level in TDM mode = (I2S_TX_TDM_WS_WIDTH[8:0] +1) x T_BCK. (R/W)

I2S_TX_BITS_MOD Configures the valid data bit length of I2S TX channel.

7: All the valid channel data is in 8-bit mode

15: All the valid channel data is in 16-bit mode

23: All the valid channel data is in 24-bit mode

31: All the valid channel data is in 32-bit mode

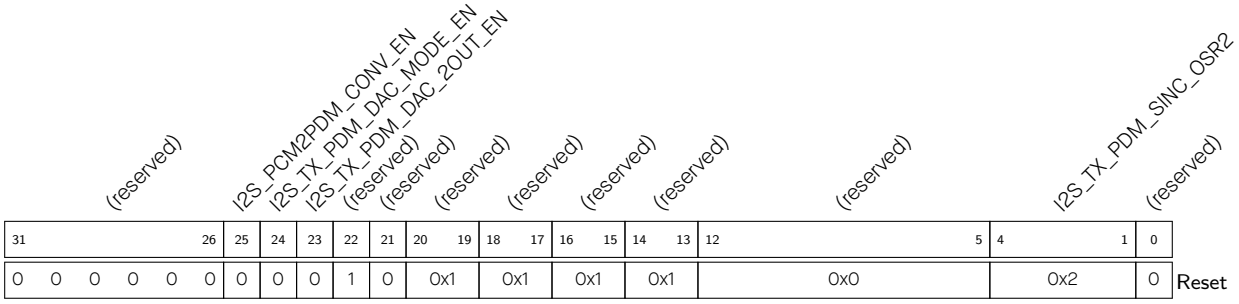
Other values are invalid.

(R/W)

I2S_TX_HALF_SAMPLE_BITS Configures I2S TX sample bits. BCK cycles in one WS period = I2S_TX_HALF_SAMPLE_BITS x 2. (R/W)

I2S_TX_TDM_CHAN_BITS Configures TX bit number for each channel in TDM mode. Bit number expected = I2S_TX_TDM_CHAN_BITS + 1. (R/W)

Register 34.12. I2S_TX_PCM2PDM_CONF_REG (for I2S*0* only) (0x0040)



I2S_TX_PDM_SINC_OSR2 Configures I2S TX PDM OSR value. (R/W)

I2S_TX_PDM_DAC_2OUT_EN Configures whether to enable I2S TX PDM DAC mode.

0: Enable 1-line DAC output mode

1: Enable 2-line DAC output mode

Only valid when I2S_TX_PDM_DAC_MODE_EN is set.

(R/W)

I2S_TX_PDM_DAC_MODE_EN Configures whether to enable 1-line PDM output mode or DAC output mode.

0: Enable 1-line PDM output mode

1: Enable DAC output mode

(R/W)

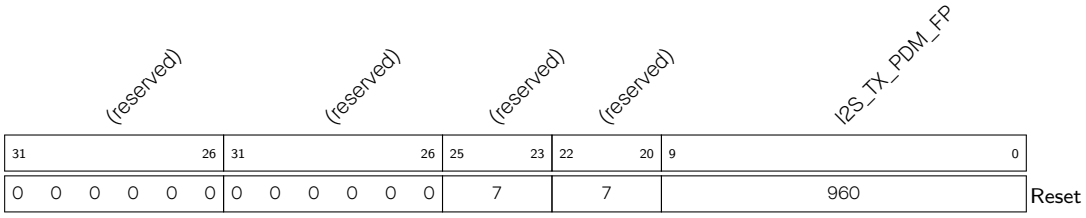
I2S_PCM2PDM_CONV_EN Configures whether to enable the I2S TX PCM-to-PDM converter.

0: Disable

1: Enable

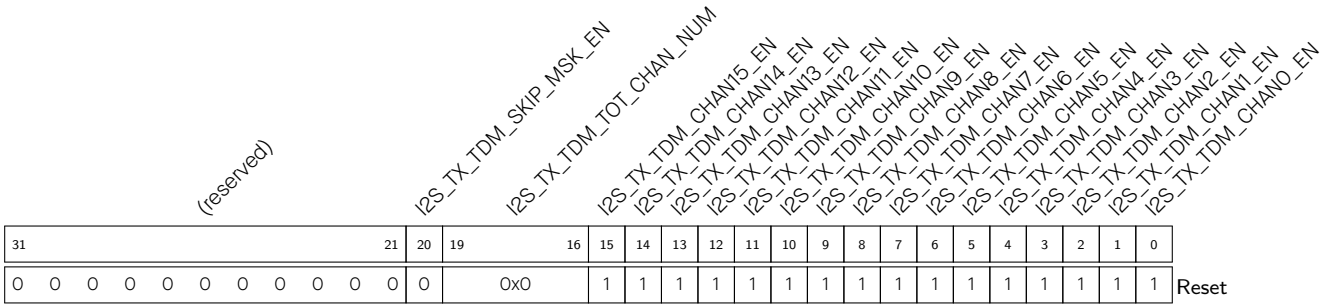
(R/W)

Register 34.13. I2S_TX_PCM2PDM_CONF1_REG (0x0044)



I2S_TX_PDM_FS Configures I2S PDM TX upsampling parameter. (R/W)

Register 34.14. I2S_TX_TDM_CTRL_REG (0x0054)



I2S_TX_TDM_CHAN*n*_EN (*n*: 0-15) Configures whether to enable the valid data output of I2S TX TDM channel *n*.

0: Channel TX data is controlled by I2S_TX_CHAN_EQUAL and I2S_SINGLE_DATA. See Section 34.9.2.1

1: Enable

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM Configures the total number of channels in use in I2S TX TDM mode. Total channel number in use = I2S_TX_TDM_TOT_CHAN_NUM + 1. (R/W)

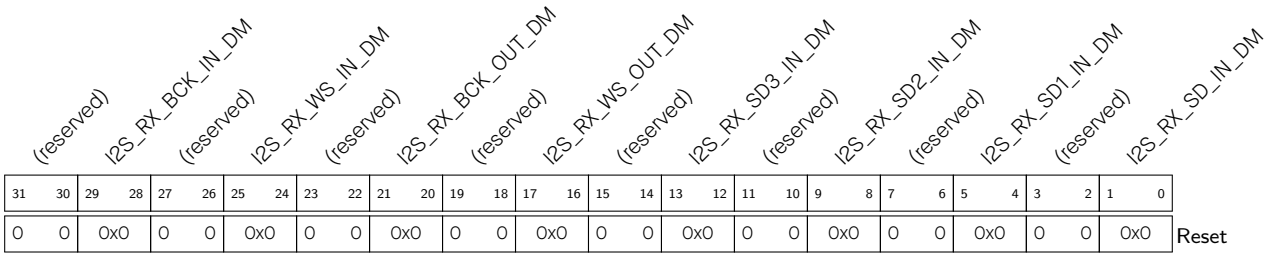
I2S_TX_TDM_SKIP_MSK_EN Configures the data to be sent in GDMA TX buffer.

0: Data stored in GDMA TX buffer is used by enabled channels and will not be read by channels that are not enabled.

1: Data stored in GDMA TX buffer is read by all channels and will be skipped by channels that are not enabled.

(R/W)

Register 34.15. I2S_RX_TIMING_REG (0x0058)



I2S_RX_SD_IN_DM Configures the delay mode of I2S RX SD input signal.

0: Bypass

1: Delay by rising edge

2: Delay by falling edge

3: Reserved

(R/W)

I2S_RX_SD1_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD1 input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_SD2_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD2 input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_SD3_IN_DM (for I2S*0* only) Configures the delay mode of I2S RX SD3 input signal.For detailed configuration values, please refer to I2S_RX_SD_IN_DM.(R/W)

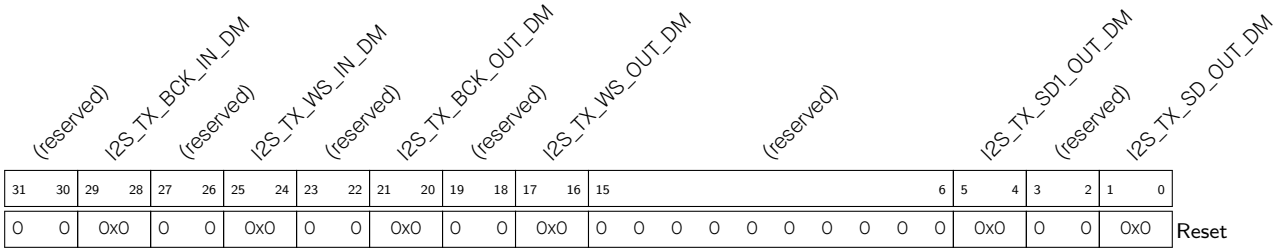
I2S_RX_WS_OUT_DM Configures the delay mode of I2S RX WS output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_BCK_OUT_DM Configures the delay mode of I2S RX BCK output signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_WS_IN_DM Configures the delay mode of I2S RX WS input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

I2S_RX_BCK_IN_DM Configures the delay mode of I2S RX BCK input signal. For detailed configuration values, please refer to I2S_RX_SD_IN_DM. (R/W)

Register 34.16. I2S_TX_TIMING_REG (0x005C)



I2S_TX_SD_OUT_DM Configures the delay mode of I2S TX SD output signal.

0: Bypass

1: Delay by rising edge

2: Delay by falling edge

3: Reserved

(R/W)

I2S_TX_SD1_OUT_DM (for I2S*0* only) Configures the delay mode of I2S TX SD1 output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

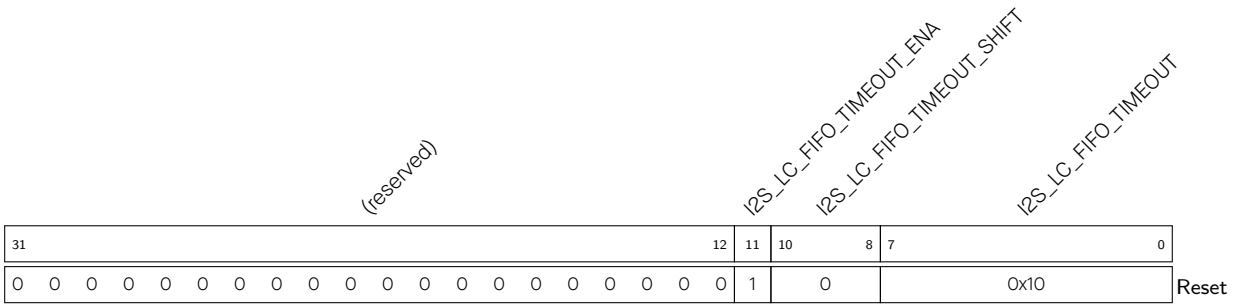
I2S_TX_WS_OUT_DM Configures the delay mode of I2S TX WS output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_BCK_OUT_DM Configures the delay mode of I2S TX BCK output signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_WS_IN_DM Configures the delay mode of I2S TX WS input signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

I2S_TX_BCK_IN_DM Configures the delay mode of I2S TX BCK input signal. For detailed configuration values, please refer to I2S_TX_SD_OUT_DM. (R/W)

Register 34.17. I2S_LC_HUNG_CONF_REG (0x0060)



I2S_LC_FIFO_TIMEOUT Configures FIFO timeout threshold. The FIFO hung counter is incremented by one each time I2S is in an error state and the tick counter reaches its threshold. I2S_TX_HUNG_INT or I2S_RX_HUNG_INT interrupt will be triggered when FIFO hung counter is equal to the FIFO timeout threshold. (R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT Configures tick counter threshold. The tick counter is incremented by one each time I2S is in an error state and it is on the rising edge in each system clock (APB). The tick counter is reset when counter value >= $88000/2^{I2S\_LC\_FIFO\_TIMEOUT\_SHIFT}$. (R/W)

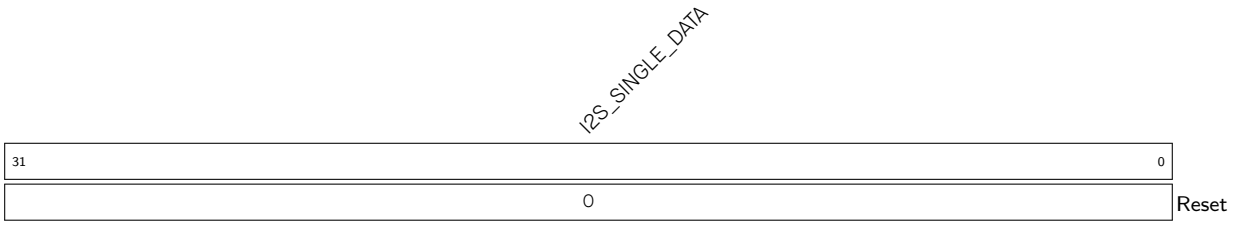
I2S_LC_FIFO_TIMEOUT_ENA Configures whether to enable FIFO timeout.

0: Disable

1: Enable

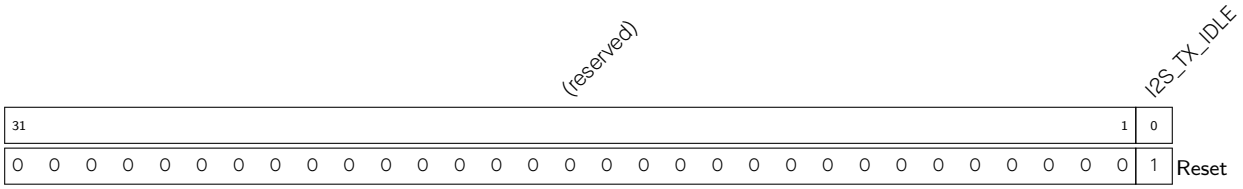
(R/W)

Register 34.18. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA Configures constant channel data to be sent out. (R/W)

Register 34.19. I2S_STATE_REG (0x006C)



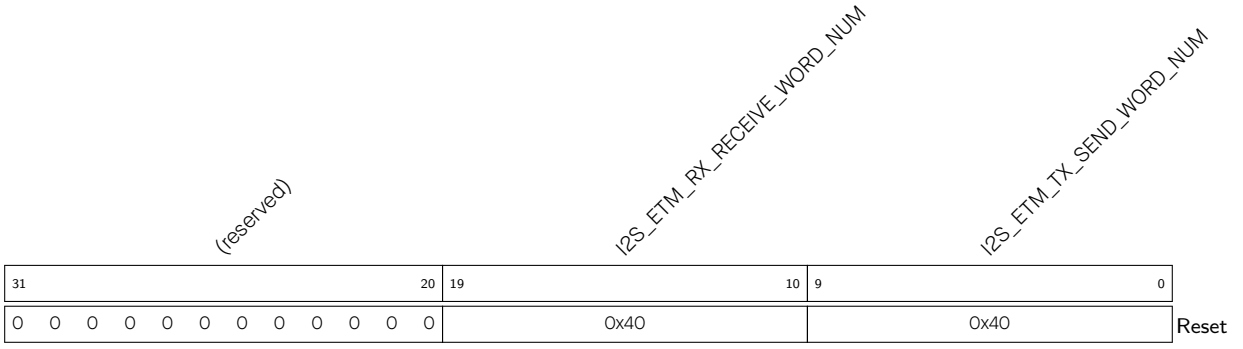
I2S_TX_IDLE Represents the TX unit state.

0: I2S TX unit is working

1: I2S TX unit is in idle state

(RO)

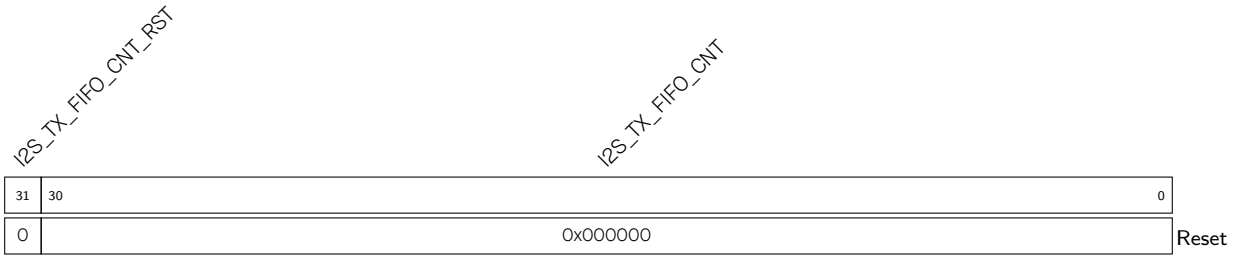
Register 34.20. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM Configures the threshold of triggering ETM I2S_TX_X_WORDS_SENT event. When transmitting word number of I2S_ETM_TX_SEND_WORD_NUM [9:0], I2S will trigger the corresponding ETM event. (R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM Configures the threshold of triggering ETM I2S_RX_X_WORDS_RECEIVED event. When receiving word number of I2S_ETM_RX_RECEIVE_WORD_NUM [9:0], I2S will trigger the corresponding ETM event. (R/W)

Register 34.21. I2S_FIFO_CNT_REG (0x0074)



I2S_TX_FIFO_CNT Configures the TX FIFO counter value. (RO)

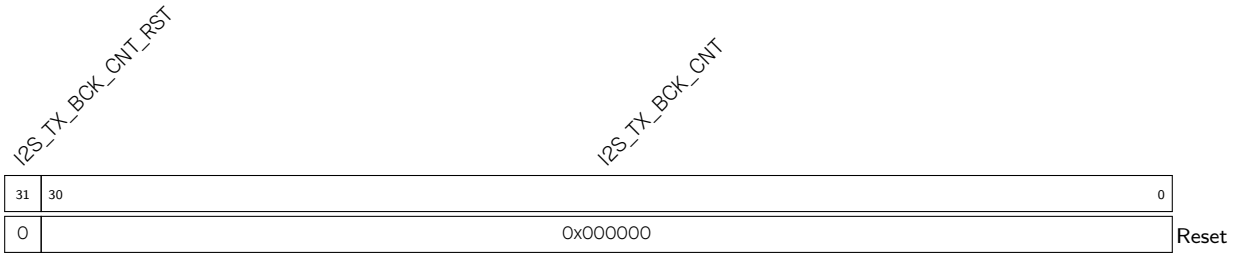
I2S_TX_FIFO_CNT_RST Configure whether to reset the TX FIFO counter.

0: No effect

1: Reset

(WT)

Register 34.22. I2S_BCK_CNT_REG (0x0078)



I2S_TX_BCK_CNT Configures the TX BCK counter value. (RO)

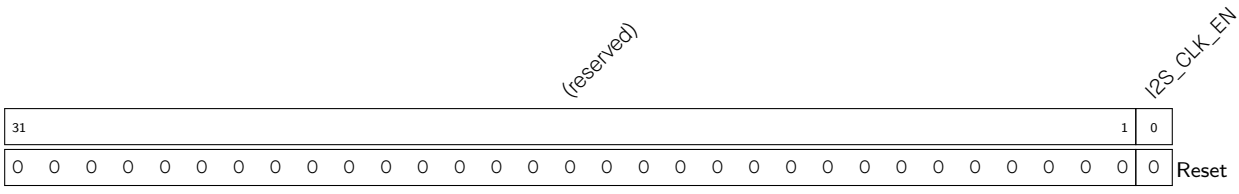
I2S_TX_BCK_CNT_RST Configures whether to reset TC BCK counter.

0: No effect

1: Reset

(WT)

Register 34.23. I2S_CLK_GATE_REG (0x007C)



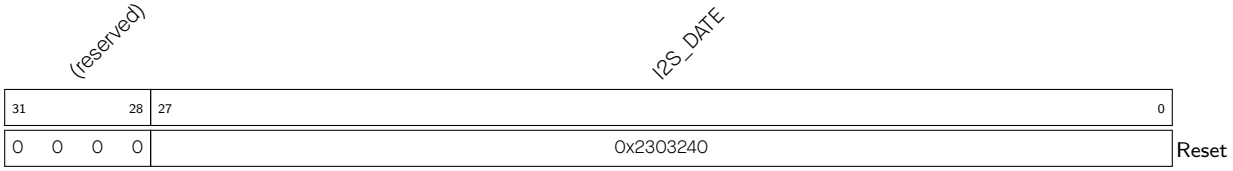
I2S_CLK_EN Configures whether to enable clock gate.

0: Disable

1: Enable

(R/W)

Register 34.24. I2S_DATE_REG (0x0080)



I2S_DATE Version control register. (R/W)

Chapter 35

Pulse Count Controller (PCNT)

The pulse count controller (PCNT) is designed to count input pulses.

35.1 Introduction

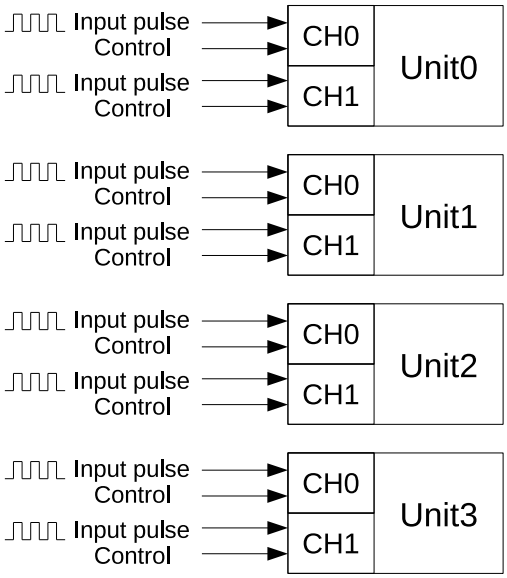


Figure 35.1-1. PCNT Overview

As shown in Figure 35.1-1, PCNT has four independent pulse counters called units, which have their groups of registers. Each unit includes two channels (ch0 and ch1) and a 16-bit signed counter. Each channel has an input pulse signal and an input signal filtering module. The 16-bit signed counter can be configured for incremental or decremental counting. The clock for PCNT module is APB_CLK.

35.2 Feature List

A PCNT has the following features:

- Four independent pulse counters (units) that count from 1 to 65535
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g., sig_ch0_u*n*) with their corresponding control signals (e.g., ctrl_ch0_u*n*)
- Independently filter glitches of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit

- Each channel has the following parameters:
  1. Selection between counting on rising or falling edges of the input pulse signal
  2. Configuration to Increment, Decrement, or Disable counter mode for control signal's high and low states
- Maximum frequency of pulses: $\frac{f_{APB\_CLK}}{2}$

35.3 Architectural Overview

Each unit of PCNT includes two channels (ch0 and ch1) and the functionality of the two channels is identical. The remainder of the chapter will take channel 0 (ch0) as example and *n* denotes the number of a unit from 0 ~ 3. Each channel of PCNT has two input signals:

1. One input pulse signal (e.g., sig_ch0_u*n*, the input pulse signal for ch0 of unit *n* ch0)
2. One control signal (e.g., ctrl_ch0_u*n*, the control signal for ch0 of unit *n* ch0)

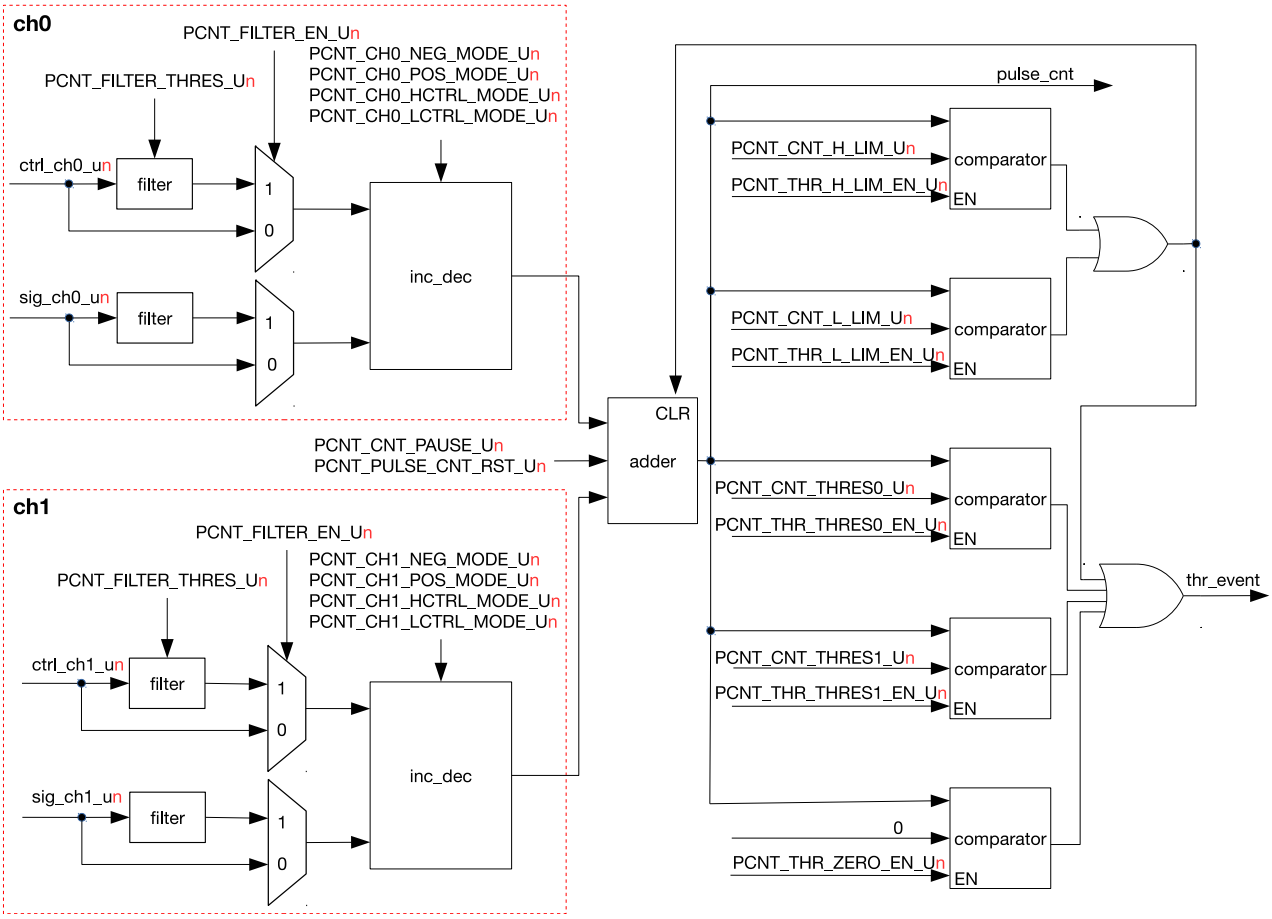


Figure 35.3-1. PCNT Architectural Overview

Figure 35.3-1 shows PCNT's architecture. As stated above, ctrl_ch0_u*n* is the control signal for ch0 of unit *n*. Its high and low states can be assigned in different counter modes and used for pulse counting of the channel's input pulse signal sig_ch0_u*n* on falling or rising edges.

35.4 Functional Description

The available counter modes of PCNT are as follows:

- Increment mode: When a channel detects an active edge of sig_ch0_u*n* (can be configured by software), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If PCNT_CNT_PAUSE_U*n* is set to 1, then pulse_cnt freezes.
- Decrement mode: When a channel detects an active edge of sig_ch0_u*n* (can be configured by software), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If PCNT_CNT_PAUSE_U*n* is set to 1, then pulse_cnt freezes.
- Disable mode: Counting is disabled, and the counter value pulse_cnt freezes.

Table 35.4-1 to Table 35.4-4 provide information on how to configure the counter mode for channel 0.

Table 35.4-1. Counter Mode. Rising edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 35.4-2. Counter Mode. Rising edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 35.4-3. Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 35.4-4. Counter Mode. Falling Edge of Input Pulse Signal. Control Signal in High State

PCNT_CHO_NEG_MODE_U*n*	PCNT_CHO_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Each unit has one filter for all its control and input pulse signals. A filter can be enabled by setting the bit PCNT_FILTER_EN_U*n*. The filter monitors the signals and ignores all the noise, i.e., the glitches with pulse widths shorter than PCNT_FILTER_THRES_U*n* APB clock cycles in length.

As shown on Figure 35.3-1, each unit has two channels which process different input pulse signals and increase or decrease values via their respective inc_dec modules, then the two channels send these values to the counter module which has a 16-bit wide signed register. This counter can be suspended by setting PCNT_CNT_PAUSE_U*n*, and cleared by setting PCNT_PULSE_CNT_RST_U*n*.

The PCNT has five watchpoints that share one interrupt. The interrupt can be enabled or disabled by interrupt enable signals of each individual watchpoint.

- Maximum count value: When pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, a high limit interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_U*n* is high.
- Minimum count value: When pulse_cnt reaches PCNT_CNT_L_LIM_U*n*, a low limit interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_U*n* is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_THRESO_U*n* or PCNT_CNT_THRES1_U*n*, an interrupt is triggered and either PCNT_CNT_THR_THRESO_LAT_U*n* or PCNT_CNT_THR_THRES1_LAT_U*n* is high respectively.
- Zero: When pulse_cnt is 0, an interrupt is triggered and PCNT_CNT_THR_ZERO_LAT_U*n* is valid.

If PCNT_CNT_H_LIM_U*n* and/or PCNT_CNT_L_LIM_U*n* are reconfigured by software when PCNT is working, the new configuration will take effect after pluse_cnt counts to any of the above five watchpoints; If PCNT_CNT_THRESO_U*n* and/or PCNT_CNT_THRES1_U*n* are reconfigured by software, the new configuration will take effect immediately.

35.5 Interrupts

ESP32-P4's PCNT can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- PCNT_INT

There are several internal interrupt sources from PCNT that can generate the above interrupt signal. The interrupt sources from PCNT are listed with their trigger conditions and the resulted interrupt signal in Table 35.5-1.

Table 35.5-1. PCNT's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
PCNT_CNT_THR_EVENT_U0_INT	reaching the watchpoint in PCNT unit 0	PCNT_INT
PCNT_CNT_THR_EVENT_U1_INT	reaching the watchpoint in PCNT unit 1	PCNT_INT
PCNT_CNT_THR_EVENT_U2_INT	reaching the watchpoint in PCNT unit 2	PCNT_INT
PCNT_CNT_THR_EVENT_U3_INT	reaching the watchpoint in PCNT unit 3	PCNT_INT

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

35.6 Programming Procedures

In each unit, channel 0 and channel 1 can be configured to work independently or together. The three subsections below provide details of channel 0 incrementing independently, channel 0 decrementing independently, and channel 0 and channel 1 incrementing together. For other working modes not elaborated in this section (e.g., channel 1 incrementing/decremeting independently, or one channel incrementing while the other decrementing), reference can be made to these three subsections.

35.6.1 Channel 0 Incrementing Independently

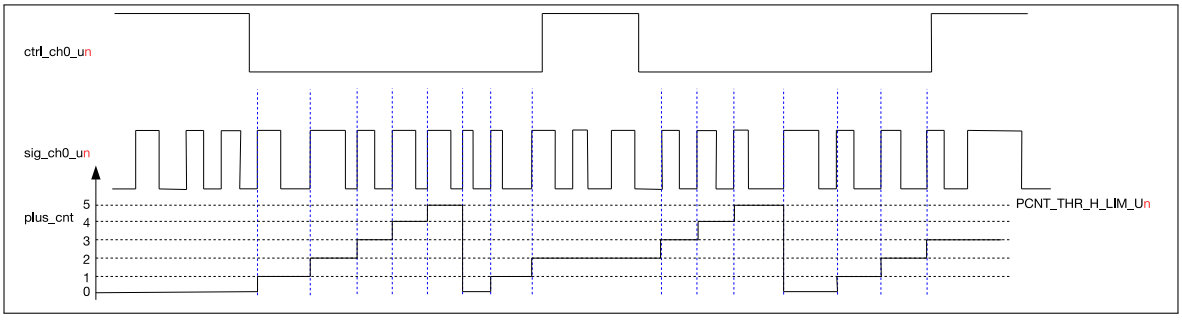


Figure 35.6-1. Channel 0 Up Counting Diagram

Figure 35.6-1 illustrates how channel 0 is configured to increment independently on the rising edge of sig_ch0_u*n* while channel 1 is disabled (see Subsection 35.4 for how to disable channel 1). The configuration of channel 0 is shown below.

- PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
- PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
- PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch0_u*n*.

- PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch0_u*n*.
- PCNT_CNT_H_LIM_U*n*=5: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

35.6.2 Channel 0 Decrementing Independently

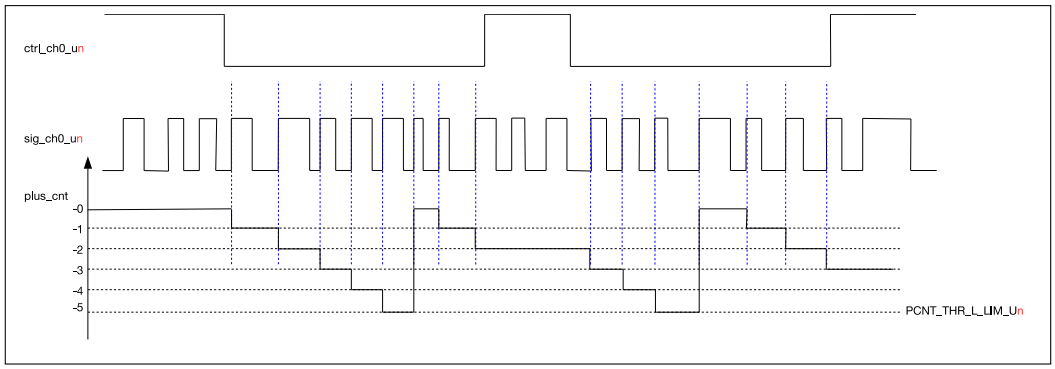


Figure 35.6-2. Channel 0 Down Counting Diagram

Figure 35.6-2 illustrates how channel 0 is configured to decrement independently on the rising edge of sig_ch0_u*n* while channel 1 is disabled. The configuration of channel 0 in this case differs from that in Figure 35.6-1 in the following aspects:

- PCNT_CH0_POS_MODE_U*n*=2: the counter decrements on the rising edge of sig_ch0_u*n*.
- PCNT_CNT_L_LIM_U*n*=-5: when pulse_cnt counts down to PCNT_CNT_L_LIM_U*n*, it is cleared.

35.6.3 Channel 0 and Channel 1 Incrementing Together

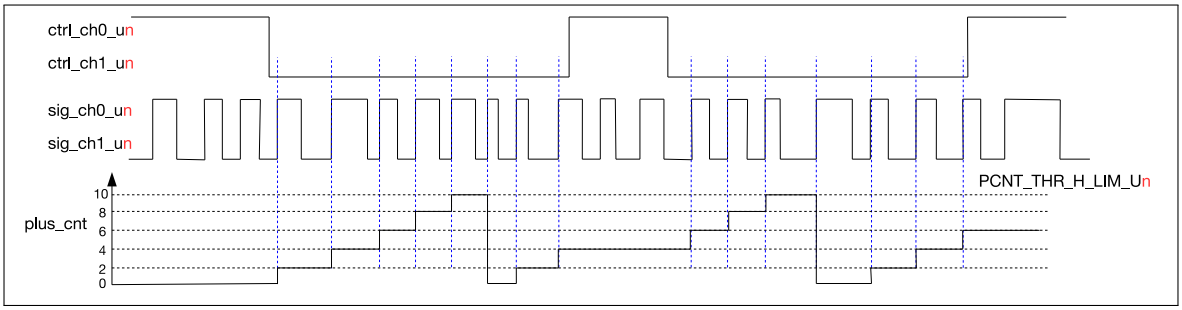


Figure 35.6-3. Two Channels Up Counting Diagram

Figure 35.6-3 illustrates how channel 0 and channel 1 are configured to increment on the rising edge of sig_ch0_u*n* and sig_ch1_u*n* respectively at the same time. It can be seen in Figure 35.6-3 that control signal ctrl_ch0_u*n* and ctrl_ch1_u*n* have the same waveform, so as input pulse signal sig_ch0_u*n* and sig_ch1_u*n*. The configuration procedure is shown below.

- For channel 0:
  - PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.

  - PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch0_u*n*.
  - PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch0_u*n*.
- For channel 1:
  - PCNT_CH1_LCTRL_MODE_U*n*=0: When ctrl_ch1_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH1_HCTRL_MODE_U*n*=2: When ctrl_ch1_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH1_POS_MODE_U*n*=1: The counter increments on the rising edge of sig_ch1_u*n*.
  - PCNT_CH1_NEG_MODE_U*n*=0: The counter idles on the falling edge of sig_ch1_u*n*.
- PCNT_CNT_H_LIM_U*n*=10: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

35.7 Register Summary

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

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

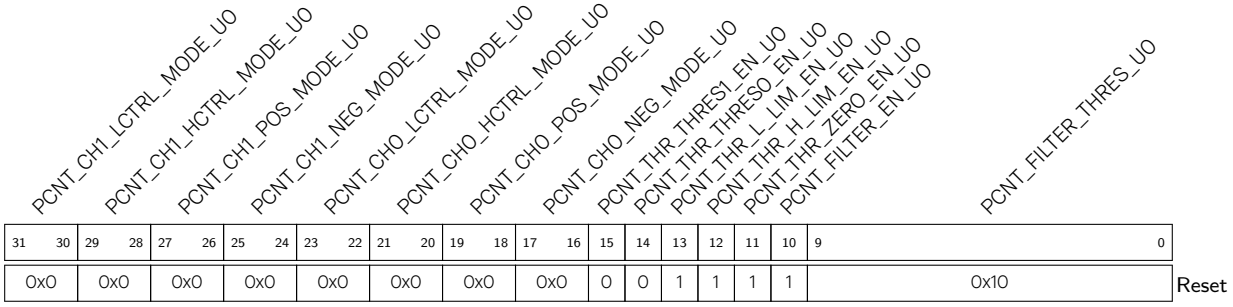
Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT UNIT0 status register	0x0050	RO
PCNT_U1_STATUS_REG	PNCT UNIT1 status register	0x0054	RO
PCNT_U2_STATUS_REG	PNCT UNIT2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT UNIT3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

35.8 Registers

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

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

Register 35.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)



PCNT_FILTER_THRES_U*n* Configures the maximum threshold for the filter. Any pulses with width less than this will be ignored when the filter is enabled.

Measurement unit: APB_CLK cycles.

(R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. Configures it to enable the high limit interrupt. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. Configures it to enable the low limit interrupt. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_CH0_NEG_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a falling edge.

1: Increment the counter

2: Decrement the counter

0, 3: No effect

(R/W)

PCNT_CH0_POS_MODE_U*n* Configures the behavior when the signal input of channel 0 detects a rising edge.

1: Increment the counter

2: Decrement the counter

0, 3: No effect

(R/W)

PCNT_CH0_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.

0: No modification

1: Invert behavior (increase → decrease, decrease → increase)

2, 3: Inhibit counter modification

(R/W)

Continued on the next page...

Register 35.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)

Continued from the previous page...

PCNT_CH0_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

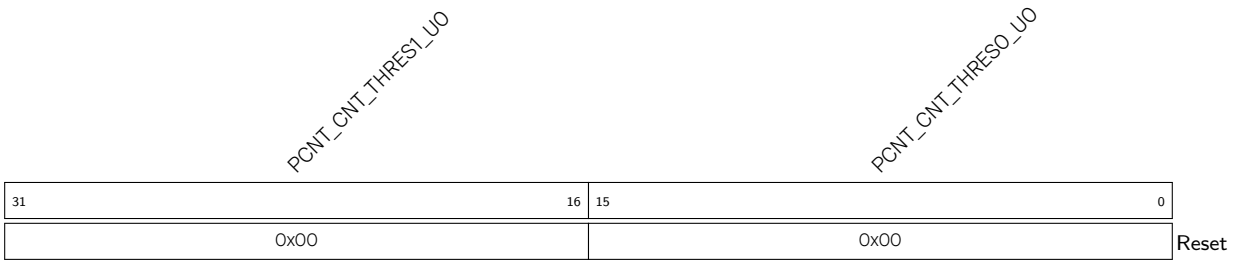
PCNT_CH1_NEG_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a falling edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_POS_MODE_U*n* Configures the behavior when the signal input of channel 1 detects a rising edge.
1: Increment the counter
2: Decrement the counter
0, 3: No effect
(R/W)

PCNT_CH1_HCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

PCNT_CH1_LCTRL_MODE_U*n* Configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low.
0: No modification
1: Invert behavior (increase → decrease, decrease → increase)
2, 3: Inhibit counter modification
(R/W)

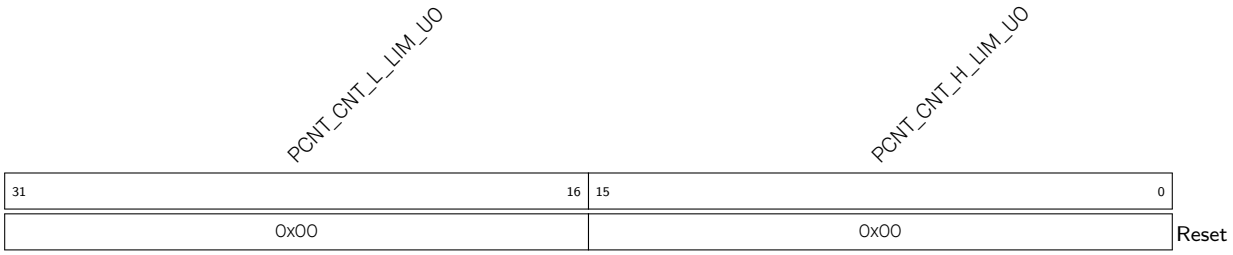
Register 35.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xC**n*)



PCNT_CNT_THRES0_U*n* Configures the thres0 value for unit *n*. (R/W)

PCNT_CNT_THRES1_U*n* Configures the thres1 value for unit *n*. (R/W)

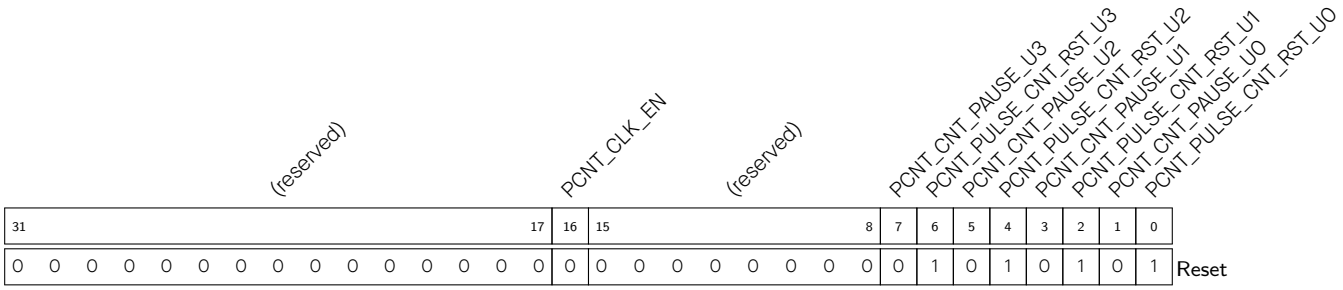
Register 35.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xC**n*)



PCNT_CNT_H_LIM_U*n* Configures the thr_h_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0. (R/W)

PCNT_CNT_L_LIM_U*n* Configures the thr_l_lim value for unit *n*. When pulse_cnt reaches this value, the counter will be cleared to 0.(R/W)

Register 35.4. PCNT_CTRL_REG (0x0060)



PCNT_PULSE_CNT_RST_U*n* Write 1 to clear unit *n*'s counter.

0: No effect

1: Clear

(R/W)

PCNT_CNT_PAUSE_U*n* Write 1 to freeze unit *n*'s counter.

0: No effect

1: Freeze

(R/W)

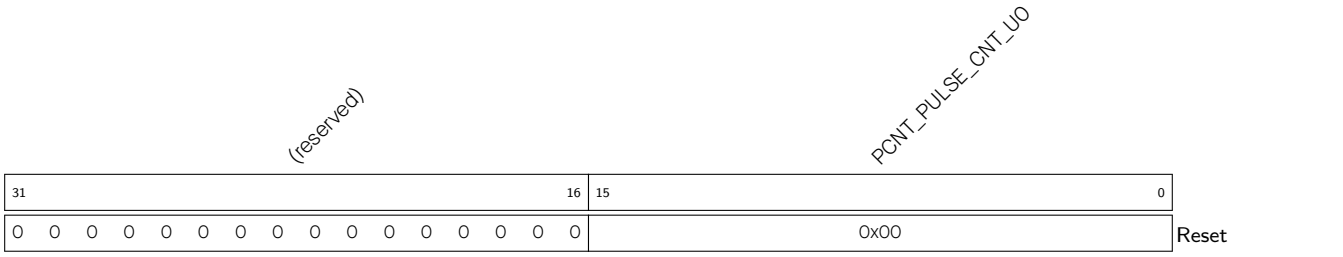
PCNT_CLK_EN Configures whether or not to enable the registers clock gate of PCNT module.

0: the clock for registers is enabled when registers are read and written

1: the clock for registers is always on

(R/W)

Register 35.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4**n*)



PCNT_PULSE_CNT_U*n* Represents the current pulse count value for unit *n*. (RO)

Register 35.6. PCNT_Un_STATUS_REG (n: 0-3) (0x0050+0x4*n)

Bits	31–7	6	5	4	3	2	1–0	
Field	(reserved)	PCNT_CNT_THR_ZERO_LAT_U0	PCNT_CNT_THR_H_LIM_LAT_U0	PCNT_CNT_THR_L_LIM_LAT_U0	PCNT_CNT_THR_THRES0_LAT_U0	PCNT_CNT_THR_THRES1_LAT_U0	PCNT_CNT_THR_ZERO_MODE_U0	
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0x0	Reset

PCNT_CNT_THR_ZERO_MODE_Un Represents the pulse counter status of PCNT_Un corresponding to 0.

0: pulse counter decreases from positive to 0

1: pulse counter increases from negative to 0

2: pulse counter is negative

3: pulse counter is positive

(RO)

PCNT_CNT_THR_THRES1_LAT_Un Represents whether the current value of PCNT_Un equals to thres1 when thres1 comparator is enabled.

0: others

1: the current pulse counter equals to thres1 and thres1 event is valid

(RO)

PCNT_CNT_THR_THRES0_LAT_Un Represents whether the current value of PCNT_Un equals to thres0 when thres0 comparator is enabled.

0: others

1: the current pulse counter equals to thres0 and thres0 event is valid

(RO)

PCNT_CNT_THR_L_LIM_LAT_Un Represents whether the current value of PCNT_Un is equal to the low limit threshold when the low limit comparator is enabled.

0: others

1: the current pulse counter equals to thr_l_lim and low limit event is valid.

(RO)

PCNT_CNT_THR_H_LIM_LAT_Un Represents whether the current value of PCNT_Un is equal to the high limit threshold when the high limit comparator is enabled.

0: others

1: the current pulse counter equals to thr_h_lim and high limit event is valid.

(RO)

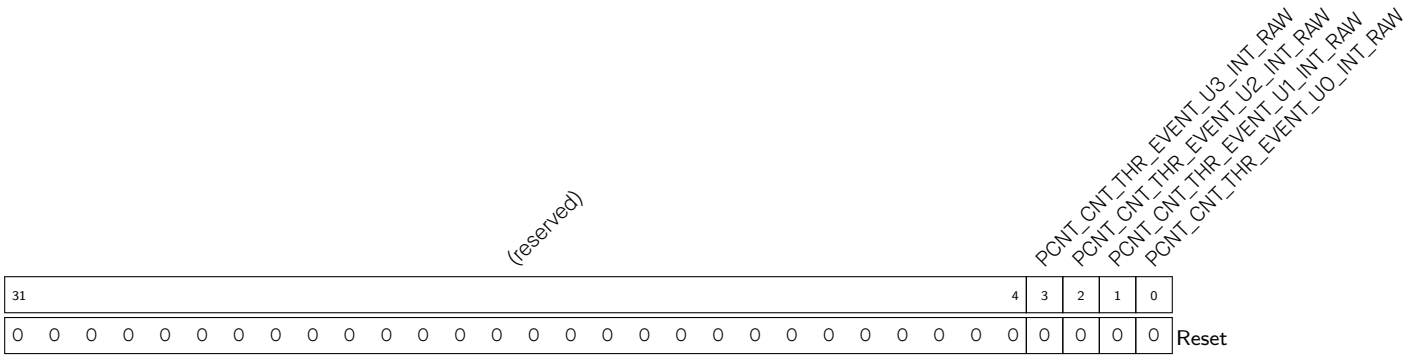
PCNT_CNT_THR_ZERO_LAT_Un Represents whether the current value of PCNT_Un is 0 when the over-zero comparator is enabled.

0: others

1: the current pulse counter equals to 0 and zero threshold event is valid.

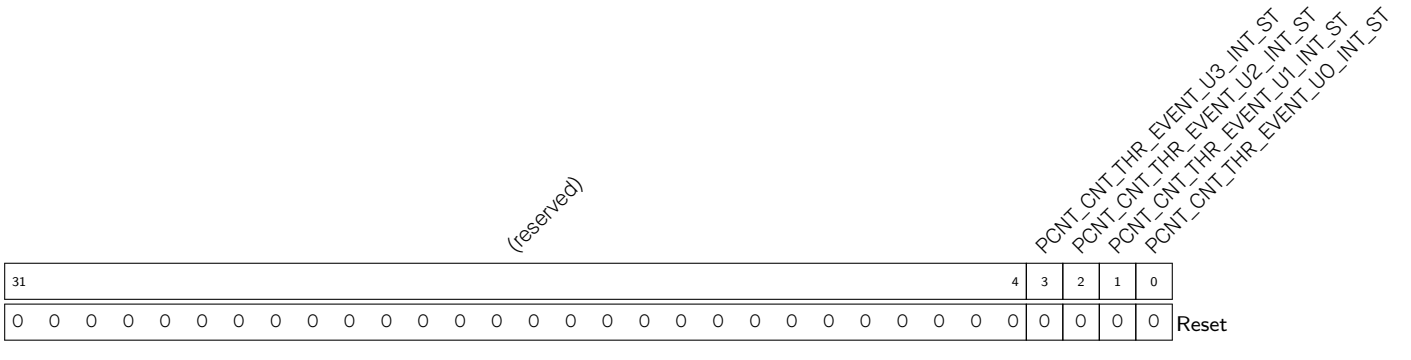
(RO)

Register 35.7. PCNT_INT_RAW_REG (0x0040)



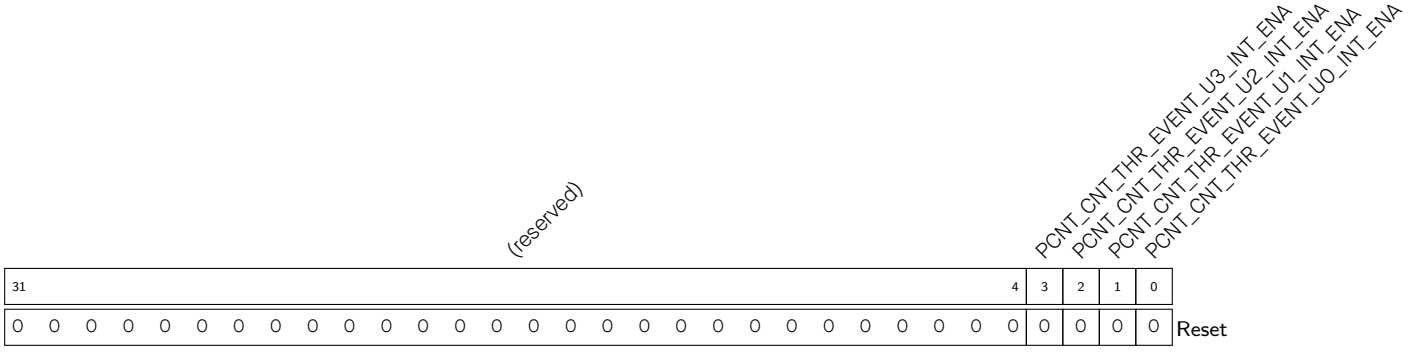
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 35.8. PCNT_INT_ST_REG (0x0044)



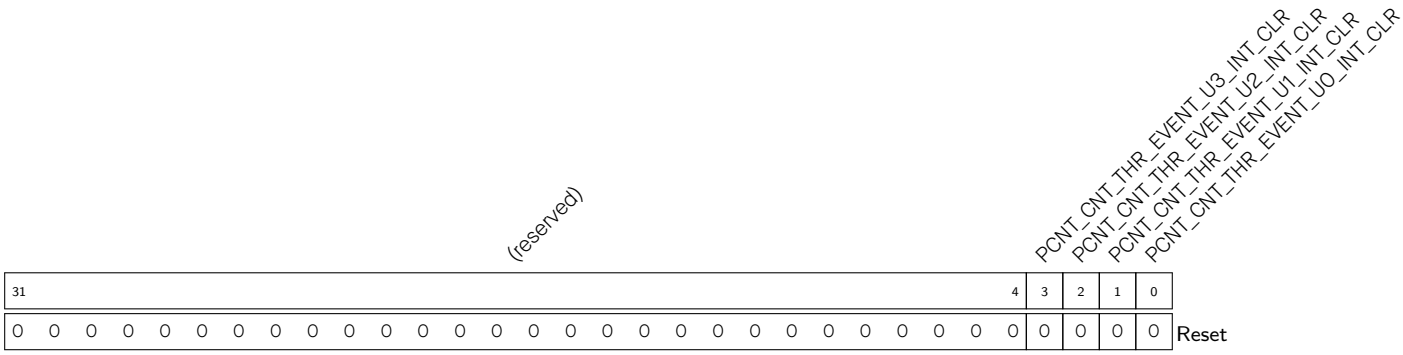
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status of the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 35.9. PCNT_INT_ENA_REG (0x0048)



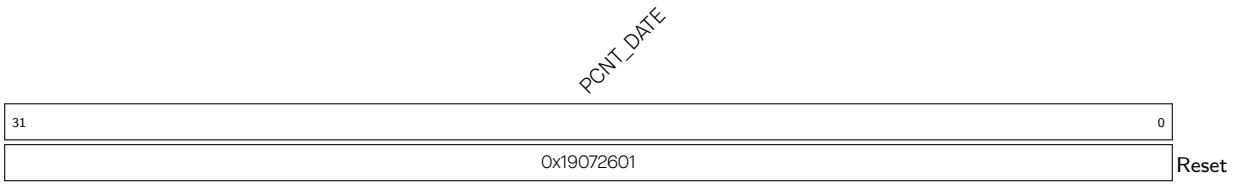
PCNT_CNT_THR_EVENT_U*n*_INT_ENA Write 1 to enable the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 35.10. PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Write 1 to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 35.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE Version control register. (R/W)

Chapter 36

USB Serial/JTAG Controller (USB_SERIAL_JTAG)

ESP32-P4 contains a USB Serial/JTAG Controller. This unit can be used to program the SoC's flash, read program output, as well as attach a debugger to the running program. All of these are possible for any computer with a USB host (hereafter referred to as 'host') without any active external components.

36.1 Overview

While programming and debugging an ESP32-P4 project using the UART and JTAG functionality is certainly possible, it has a few downsides. First of all, both UART and JTAG take up IO pins and as such, fewer pins are left usable for controlling external signals in software. Additionally, an external chip or adapter is needed for both UART and JTAG to interface with a host computer, which means it will be necessary to integrate these two functionalities in the form of external chips or debugging adapters.

In order to alleviate these issues, ESP32-P4 provides a USB Serial/JTAG Controller, which integrates the functionality of both a USB-to-serial converter as well as a USB-to-JTAG adapter. As this device directly interfaces with an external USB host using only the two data lines required by USB 2.0, only two pins are required to be dedicated to this functionality for debugging ESP32-P4.

36.2 Features

The USB Serial/JTAG controller has the following features:

- USB Full-speed device; Hardwired for CDC-ACM (Communication Device Class - Abstract Control Model) and JTAG adapter functionality
- CDC-ACM:
  - Integrates CDC-ACM adherent serial port emulation (plug-and-play on most modern OSes)
  - Supports host controllable chip reset and entry into download mode
- JTAG adapter functionality:
  - Allows fast communication with CPU debugging core using a compact representation of JTAG instructions
- A control endpoint, a dummy interrupt endpoint, two bulk input endpoints, and two bulk output endpoints; Up to 64-byte data payload size
- Internal PHY: very few or no external components needed to connect to a host computer

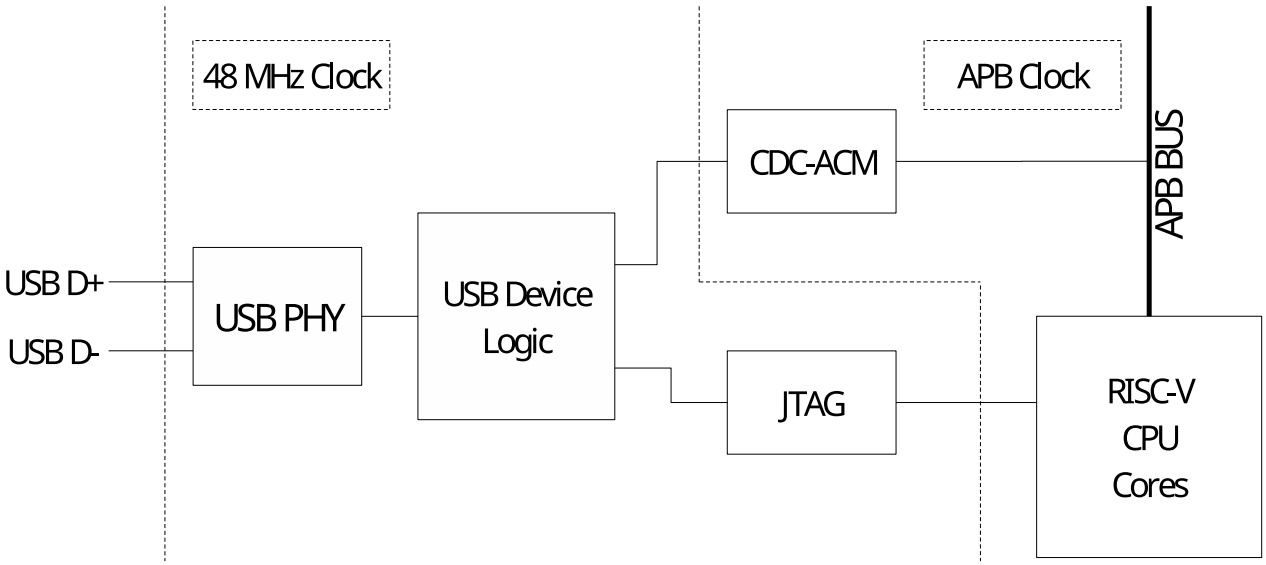


Figure 36.2-1. USB Serial/JTAG High Level Diagram

As shown in Figure 36.2-1, 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 36.2-2 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.

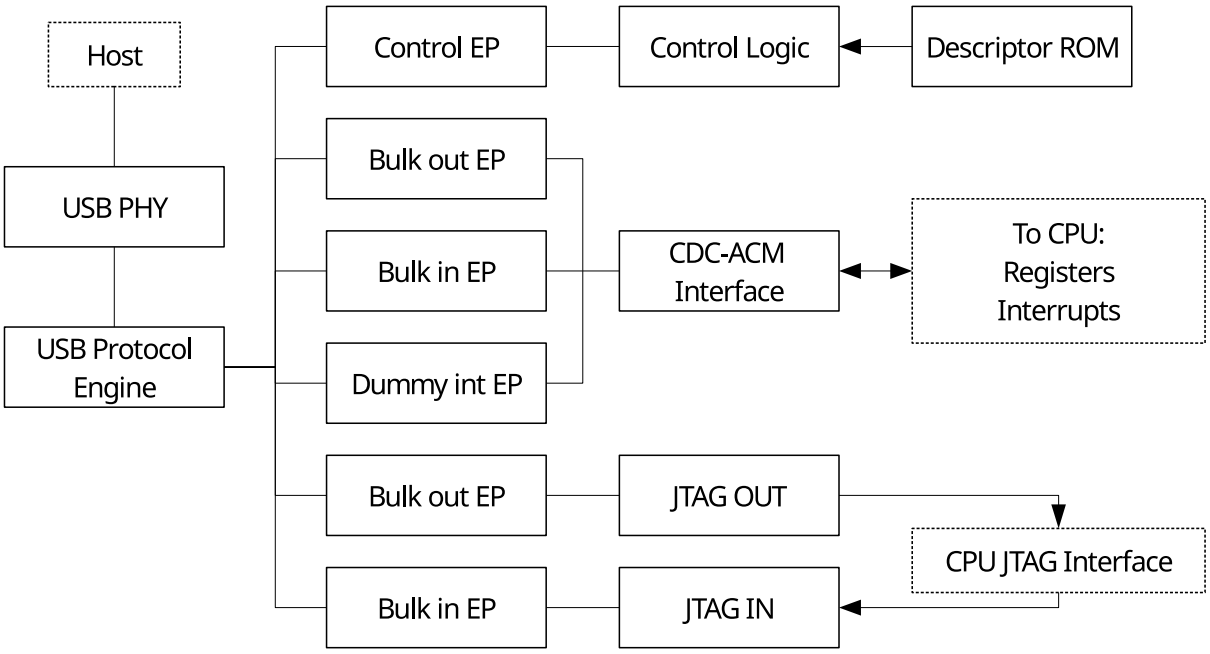


Figure 36.2-2. USB Serial/JTAG Block Diagram

36.3 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.

36.3.1 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 36.3-1. 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 36.3-2

Aside from general-purpose communication, the CDC-ACM interface can also be used to reset ESP32-P4 and optionally make it enter download mode to flash new firmware. This can be realized by setting the RTS and DTR lines on the virtual serial port.

Table 36.3-2. CDC-ACM Settings with RTS and DTR

RTS	DTR	Action
0	0	Clear download mode flag
0	1	Set download mode flag
1	0	Reset ESP32-P4
1	1	No action

Note that if the download mode flag is set when ESP32-P4 is reset, ESP32-P4 will reboot into download mode. When this flag is cleared and the chip is reset, ESP32-P4 will boot from flash. For specific sequences, please refer to Section 36.4. All these functions can also be disabled by programming various eFuses. Please refer to Chapter 6 *eFuse Controller* for more details.

36.3.2 CDC-ACM Firmware Interface Functional Description

The CPU can interact with the USB Serial/JTAG controller as the module is connected to the internal APB bus of ESP32-P4. This is mainly used to read and write data from and to the virtual serial port on the attached host.

USB CDC-ACM serial data is sent to and received from the host in packets of 0 to 64 bytes in size. When enough CDC-ACM data has accumulated in the host, the host sends a packet to the CDC-ACM receive endpoint, and the USB Serial/JTAG controller accepts this packet if it has a free buffer. Conversely, the host checks periodically if the USB Serial/JTAG controller has a packet ready to be sent to the host, and if so, receives this packet.

Firmware can get notified of new data from the host in one of the following two ways. First of all, the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit will remain set as long as there still is unread host data in the buffer. Secondly, the availability of data will trigger the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT interrupt. When data is available, it can be read by firmware through repeatedly reading bytes from USB_SERIAL_JTAG_EP1_REG. The amount of bytes to read can be determined by checking the USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL bit after reading each byte to see if there is more data to read. After all data is read, the USB device is automatically readied to receive a new data packet from the host.

When the firmware has data to send, it can put the data in the send buffer and trigger a flush to allow the host to receive the data in a USB packet. In order to do so, there needs to be space available in the send buffer. Firmware can check this by reading USB_REG_SERIAL_IN_EP_DATA_FREE. A 1 in this register field indicates there is still free room in the buffer, and firmware can fill the buffer by writing bytes to the USB_SERIAL_JTAG_EP1_REG register. Writing the buffer does not immediately trigger sending data to the host until the buffer is flushed. After the flush, the entire buffer will be ready to be received by the USB host at once. A flush can be triggered in two ways: 1) after the 64th byte is written to the buffer, the USB hardware will automatically flush the buffer to the host; or 2) firmware can trigger a flush by writing 1 to USB_REG_SERIAL_WR_DONE.

Regardless of how a flush is triggered, the send buffer will be unavailable for firmware to write into until it has

been fully read by the host. As soon as the send buffer has been fully read, the USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT interrupt will be triggered, indicating that the send buffer can receive another 64 bytes.

It is possible to handle some out-of-band serial requests in software, specifically, the host setting DTR and RTS and changing the line state. If the CDC-ACM interface receives a SET_LINE_CODING request, the peripheral can be configured to trigger a USB_SERIAL_JTAG_SET_LINE_CODE_INT interrupt, at which point the line coding can be read from the USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG register. Similarly, SET_CONTROL_LINE_STATE requests will trigger USB_SERIAL_JTAG_RTS_CHG_INT and USB_SERIAL_JTAG_DTR_CHG_INT interrupts if they change the state of these lines. Software can then read the specific state through the USB_SERIAL_JTAG_RTS and USB_SERIAL_JTAG_DTR bits. Note that as described earlier, certain RTS/DTR sequences lead to hardware reset of ESP32-P4. Software can disable hardware recognition of these DTR/RTS sequences by setting the USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS bit, allowing software to interpret these signals freely.

Finally, the host can read the current line state using GET_LINE_CODING. This event sends back the data in the USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG register and triggers a USB_SERIAL_JTAG_GET_LINE_CODE_INT interrupt.

36.3.3 USB-to-JTAG Interface: JTAG Command Processor

The USB-to-JTAG interface uses a vendor-specific class for its implementation. It consists of two endpoints, one to receive commands and another to send responses. Additionally, some less time-sensitive commands can be given as control requests.

Commands from the host to the JTAG interface are interpreted by the JTAG command processor. Internally, the JTAG command processor implements a full four-wire JTAG bus, consisting of the TCK, TMS and TDI output lines to the RISC-V CPU, as well as the TDO line signalling back from the CPU to the JTAG response capture unit. These signals adhere to the IEEE 1149.1 JTAG standards. Additionally, there is an SRST line to reset ESP32-P4.

Optionally, software can set USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN in order to redirect these signals to the GPIO matrix instead, where they can be routed to IO pads on ESP32-P4. This also allows external devices to be debugged via the USB Serial/JTAG peripheral.

The JTAG command processor parses each received nibble (4-bit value) as a command. As USB data is received in 8-bit bytes, this means each byte contains two commands. The USB command processor will execute high-nibble first and low-nibble second. The commands are used to control the TCK, TMS, TDI, and SRST lines of the internal JTAG bus, as well as to signal the JTAG response capture unit the state of the TDO line (which is driven by the CPU debugging logic) that needs to be captured.

In the internal JTAG bus, TCK, TMS, TDI, and TDO are connected directly to the JTAG debugging logic of the RISC-V CPU. SRST is connected to the reset logic of the digital circuitry in ESP32-P4 and a high level on this line will cause a digital system reset. Note that the USB Serial/JTAG controller itself is not affected by SRST.

A nibble can contain the following commands:

Table 36.3-3. Commands of a Nibble

bit	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK will set the TDI and TMS as the indicated values and emit one clock pulse on TCK. If the CAP bit is 1, it will instruct the JTAG response capture unit to capture the state of the TDO line. This instruction forms the basis of JTAG communication.
- CMD_RST will set the state of the SRST line as the indicated value. This can be used to reset ESP32-P4.
- CMD_FLUSH will instruct the JTAG response capture unit to flush the buffer of all bits it collected so the host is able to read them. Note that in some cases, a JTAG transaction will end in an odd number of commands and as such an odd number of nibbles. In this case, it is allowed to repeat the CMD_FLUSH command to get an even number of nibbles fitting an integer number of bytes.
- CMD_RSV is reserved in the current implementation. This command will be ignored when received by ESP32-P4.
- CMD_REP repeats the last (non-CMD_REP) command for a certain number of times. The purpose is to compress command streams which repeat the CMD_CLK instruction for multiple times. A command such as CMD_CLK can be followed by multiple CMD_REP commands. The number of repetitions done by one CMD_REP can be expressed as $repetition\_count = (R1 \times 2 + R0) \times (4^{cmd\_rep\_count})$, where cmd_rep_count indicates the number of the CMD_REP instruction that went directly before it. Note that the CMD_REP command is only intended to repeat a CMD_CLK command. Specifically, using it on a CMD_FLUSH command may lead to an unresponsive USB device, and a USB reset will be required to recover it.

36.3.4 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.

36.3.5 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.

36.3.6 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 36.3-4. 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 36.3-5). This structure allows host software to automatically support future revisions of the hardware without the need for an update.

The JTAG capability descriptors of ESP32-P4 are as follows. Note that all 16-bit values are little-endian.

Table 36.3-5. 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

36.4 Recommended Operation

Little setup is needed for using the USB Serial/JTAG device. The USB-to-JTAG hardware itself does not need any setup aside from the standard USB initialization that the host operating system already does. Apart from that, the CDC-ACM emulation on the host side is also plug-and-play.

On the firmware side, very little initialization is needed either. The USB hardware is self-initialized and after boot-up, if a host is connected and listening on the CDC-ACM interface, data can be exchanged as described above without any specific setup except for the situation when the firmware optionally sets up an interrupt service handler.

One thing to note is that there may be situations where either the host is not attached or the CDC-ACM virtual port is not opened. In such cases, the packets that are flushed to the host will never be picked up and the send buffer will never be empty. It is important to detect these situations and implement timeout, as this is the only way to reliably detect whether the port on the host side is closed or not.

Another thing to note is that the USB device is dependent on the BBPLL for the 48 MHz USB PHY clock. If this PLL is disabled, the USB communication will cease to function.

One scenario where this happens is Deep-sleep. The USB Serial/JTAG controller (as well as the attached RISC-V CPU) will be entirely powered down in Deep-sleep mode. If a device needs to be debugged in this mode, it may be preferable to use an external JTAG debugger and a serial interface instead.

The CDC-ACM interface can also be used to reset the SoC and take it into or out of download mode. Generating the correct sequence of handshake signals can be a bit complicated, since most operating systems only allow setting or resetting DTR and RTS separately, but not in tandem. Additionally, some drivers (e.g., the standard CDC-ACM driver on Windows) do not set DTR until RTS is set and the user needs to explicitly set RTS in order to 'propagate' the DTR value. The recommended procedures are introduced below.

To reset the SoC into download mode:

Table 36.4-1. Reset SoC into Download Mode

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	Initialize to known values
Clear RTS	RTS=0, DTR=0	-
Set DTR	RTS=0, DTR=1	Set download mode flag
Clear RTS	RTS=0, DTR=1	Propagate DTR
Set RTS	RTS=1, DTR=1	-
Clear DTR	RTS=1, DTR=0	Reset SoC
Set RTS	RTS=1, DTR=0	Propagate DTR
Clear RTS	RTS=0, DTR=0	Clear download flag

To reset the SoC into booting from flash:

Table 36.4-2. Reset SoC into Booting from flash

Action	Internal state	Note
Clear DTR	RTS=?, DTR=0	-
Clear RTS	RTS=0, DTR=0	Clear download flag
Set RTS	RTS=1, DTR=0	Reset SoC
Clear RTS	RTS=0, DTR=0	Exit reset

36.5 Interrupts

ESP32-P4's USB Serial/JTAG Controller can generate the USB_SERIAL_JTAG_INTR interrupt signal that will be sent to the Interrupt Matrix. There are several internal interrupt sources from the USB Serial/JTAG Controller that can generate this interrupt signal.

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT: triggered when flush cmd is received for the JTAG bulk out endpoint.
- USB_SERIAL_JTAG_SOF_INT: triggered when SOF frame is received.
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT: triggered when Serial Port OUT Endpoint receives one packet.

- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT: triggered when Serial Port IN Endpoint is empty.
- USB_SERIAL_JTAG_PID_ERR_INT: triggered when PID error is detected.
- USB_SERIAL_JTAG_CRC5_ERR_INT: triggered when CRC5 error is detected.
- USB_SERIAL_JTAG_CRC16_ERR_INT: triggered when CRC16 error is detected.
- USB_SERIAL_JTAG_STUFF_ERR_INT: triggered when a bit stuffing error is detected.
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT: triggered when IN token for IN endpoint 1 is received.
- USB_SERIAL_JTAG_USB_BUS_RESET_INT: triggered when USB bus reset is detected.
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT: triggered when OUT endpoint 1 receives packet with zero payload.
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT: triggered when OUT endpoint 2 receives packet with zero payload.
- USB_SERIAL_JTAG_RTS_CHG_INT: triggered when level of RTS from USB serial channel is changed.
- USB_SERIAL_JTAG_DTR_CHG_INT: triggered when level of DTR from USB serial channel is changed.
- USB_SERIAL_JTAG_GET_LINE_CODE_INT: triggered when level of GET LINE CODING request is received.
- USB_SERIAL_JTAG_SET_LINE_CODE_INT: triggered when level of SET LINE CODING request is received.

36.6 Register Summary

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

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

Name	Description	Address	Access
Configuration Registers			
USB_SERIAL_JTAG_EP1_REG	FIFO access for the CDC-ACM data IN and OUT endpoints	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	Configuration and control registers for the CDC-ACM FIFOs	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY hardware configuration	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	Registers used for debugging the PHY	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	Clock enable control	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	Memory power control	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM chip reset control	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	W0 of GET_LINE_CODING command	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	W1 of GET_LINE_CODING command	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	Configuration registers' value update	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	Serial AFIFO configure register	0x0064	varies
Interrupt Registers			
USB_SERIAL_JTAG_INT_RAW_REG	Interrupt raw status register	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	Interrupt status register	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	Interrupt enable status register	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	Interrupt clear status register	0x0014	WT
Status Registers			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO status and control registers	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	Last received SOF frame index register	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	Control IN endpoint status information	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM IN endpoint status information	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM interrupt IN endpoint status information	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG IN endpoint status information	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	Control OUT endpoint status information	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM OUT endpoint status information	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG OUT endpoint status information	0x0040	RO

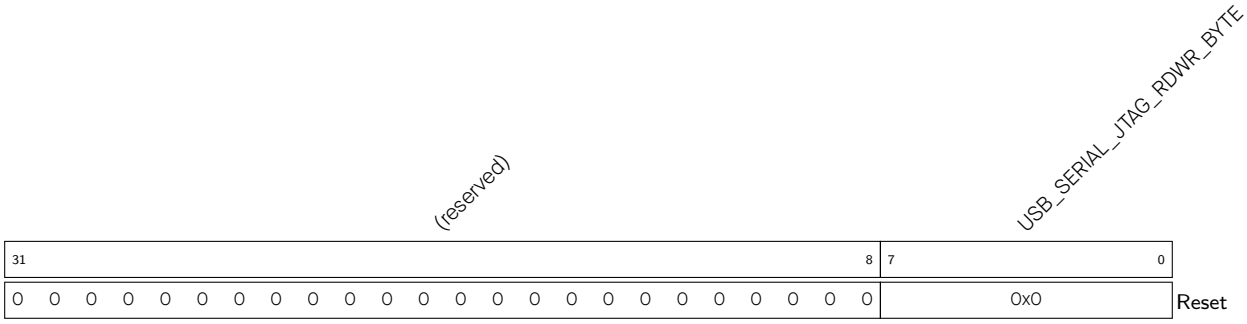
Name	Description	Address	Access
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	W0 of SET_LINE_CODING command	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	W1 of SET_LINE_CODING command	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB Bus reset status register	0x0068	RO
Version Registers			
USB_SERIAL_JTAG_DATE_REG	Date register	0x0088	R/W

36.7 Registers

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

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

Register 36.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



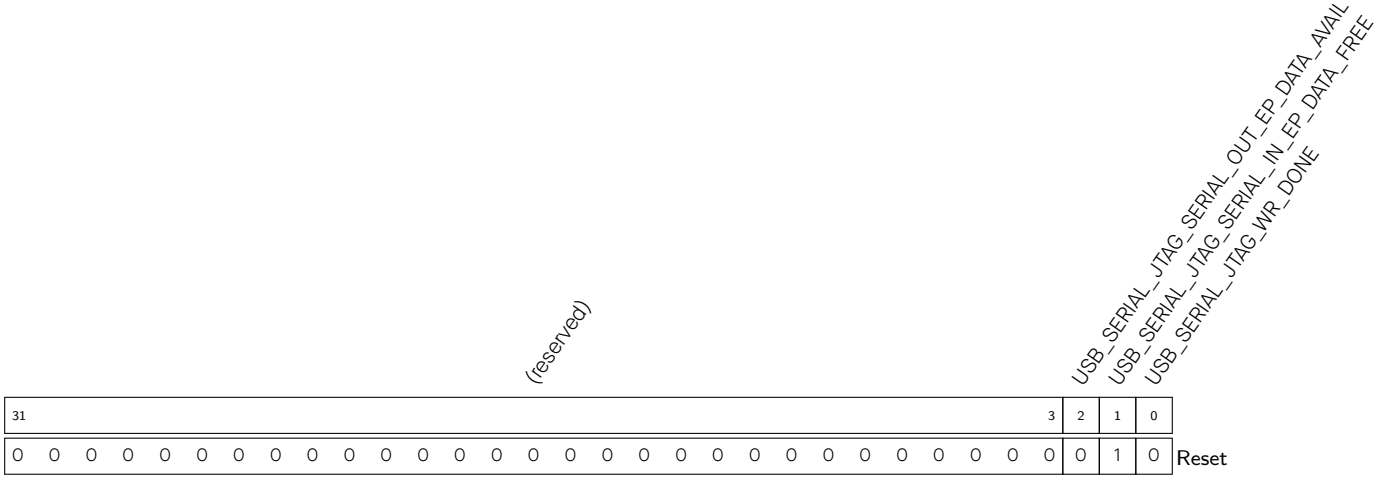
USB_SERIAL_JTAG_RDWR_BYTE A write to this register pushes the written data into the CDC TX FIFO; a read from this register pops a byte from the CDC RX FIFO and returns it.

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 36.2. 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)

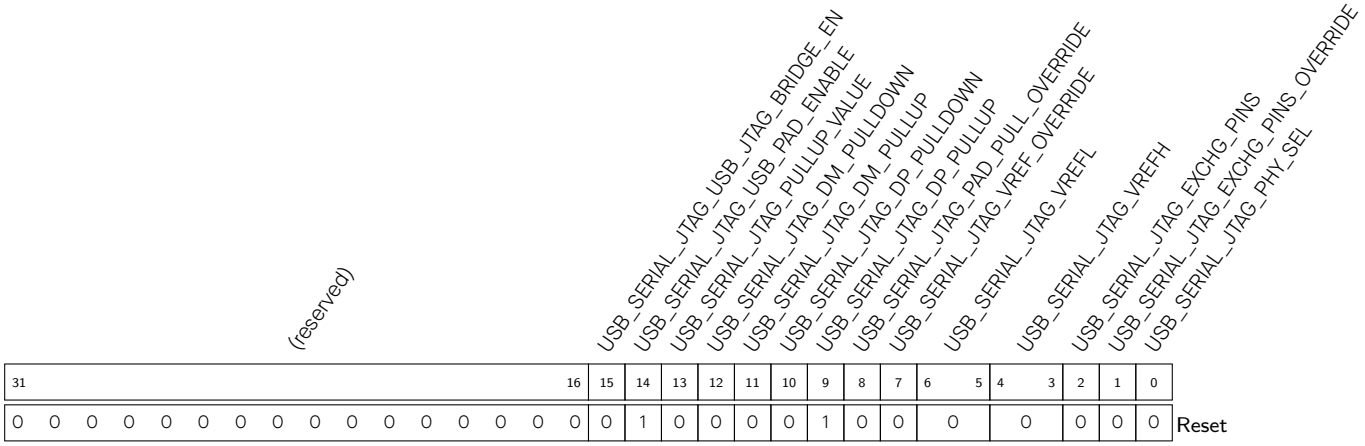
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL Represents whether there is data in CDC-ACM RX FIFO.

0: There is no data in CDC-ACM RX FIFO

1: There is data in CDC-ACM RX FIFO

(RO)

Register 36.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_PHY_SEL Configures whether to select internal or external PHY.

0: Internal PHY

1: External PHY

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE Configures whether to enable software control USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS Configures whether to enable USB D+ D- exchange.

0: Disable

1: Enable

(R/W)

USB_SERIAL_JTAG_VREFH Configures single-end input high threshold.

0: 1.76 V

1: 1.84 V

2: 1.92 V

3: 2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL Configures single-end input low threshold.

0: 0.80 V

1: 0.88 V

2: 0.96 V

3: 1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE Configures whether to enable software control input threshold.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 36.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

Continued from the previous page...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE Configures whether to enable software to control USB D+ D- pullup and pulldown.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLUP Configures whether to enable USB D+ pull up when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN Configures whether to enable USB D+ pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

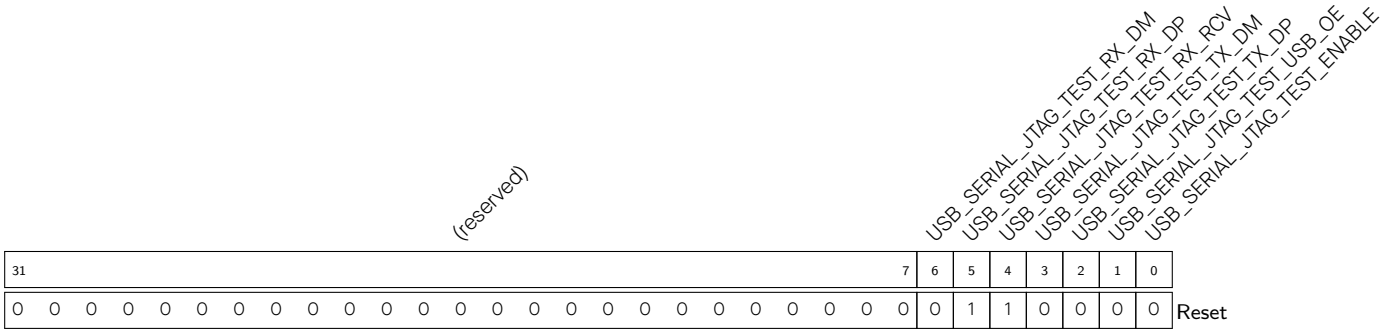
USB_SERIAL_JTAG_DM_PULLDOWN Configures whether to enable USB D- pull down when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE Configures the pull up value when USB_SERIAL_JTAG_PAD_PULL_OVERRIDE is 1.
0: 2.2 K
1: 1.1 K
(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE Configures whether to enable USB pad function.
0: Disable
1: Enable
(R/W)

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN Configures whether to disconnect USB_JTAG and internal JTAG.
0: USB_JTAG is connected to the internal JTAG port of CPU
1: USB_JTAG and the internal JTAG are disconnected, MTMS, MTDI, and MTCK are output through GPIO Matrix, and MTDO is input through GPIO Matrix
(R/W)

Register 36.4. 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+

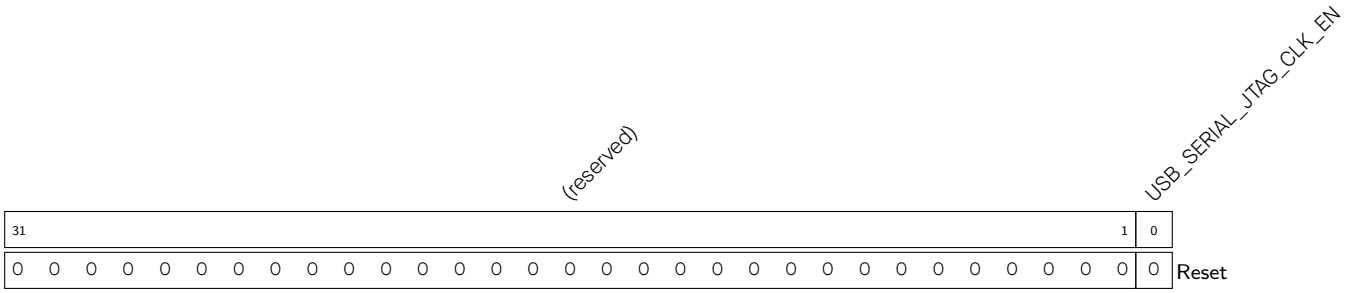
1: USB D+ voltage is higher than USB D-

(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 36.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



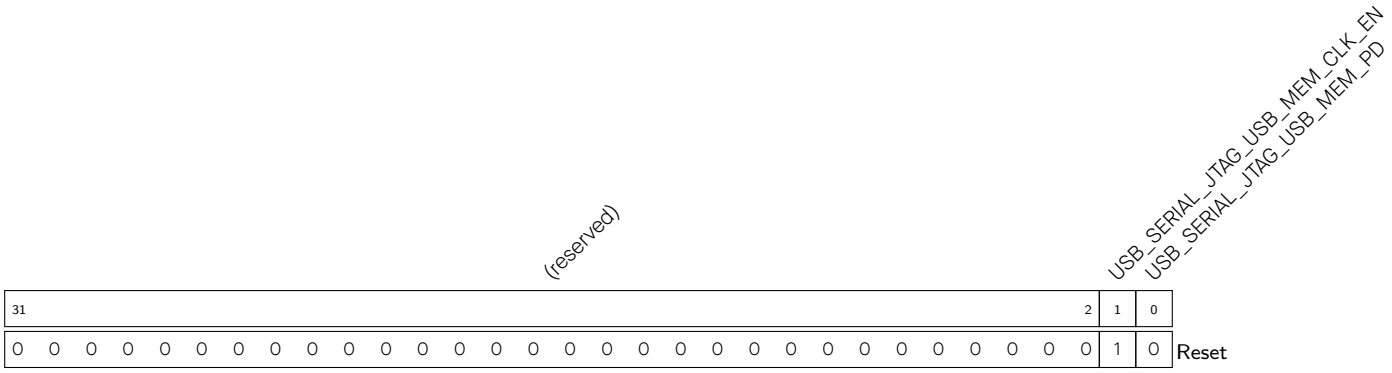
USB_SERIAL_JTAG_CLK_EN Configures whether to force clock on for register.

0: Support clock only when application writes registers

1: Force clock on for register

(R/W)

Register 36.6. 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)

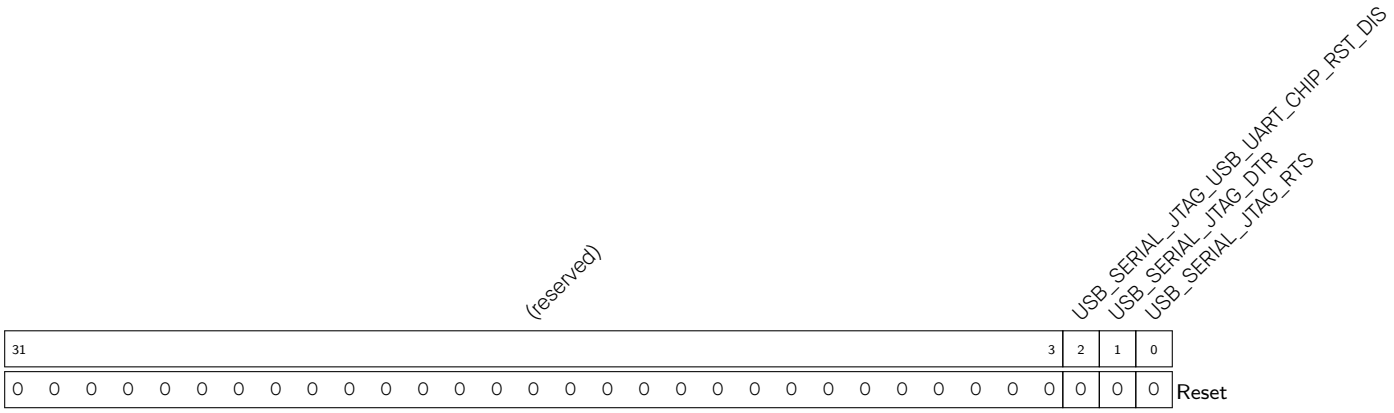
USB_SERIAL_JTAG_USB_MEM_CLK_EN Configures whether to force clock on for USB memory.

0: No effect

1: Force

(R/W)

Register 36.7. 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)

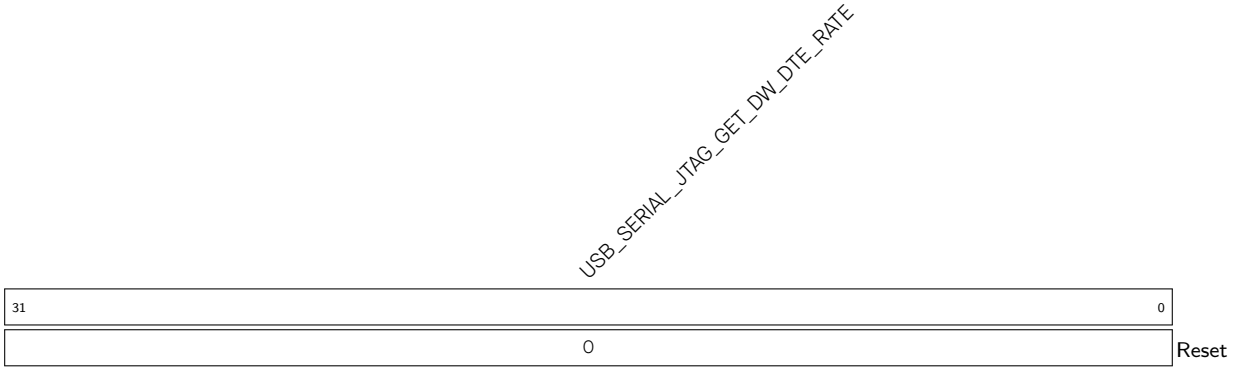
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS Configures whether to disable chip reset from USB serial channel.

0: No effect

1: Disable

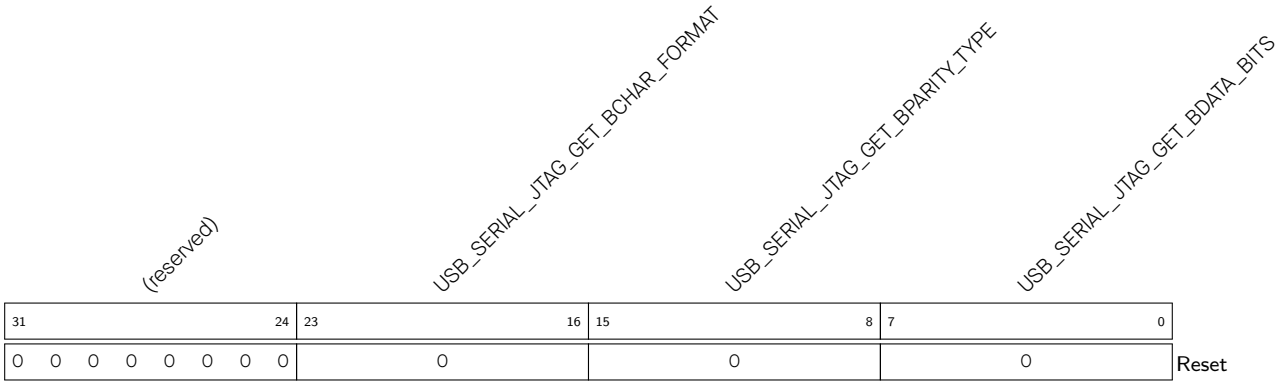
(R/W)

Register 36.8. 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 36.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

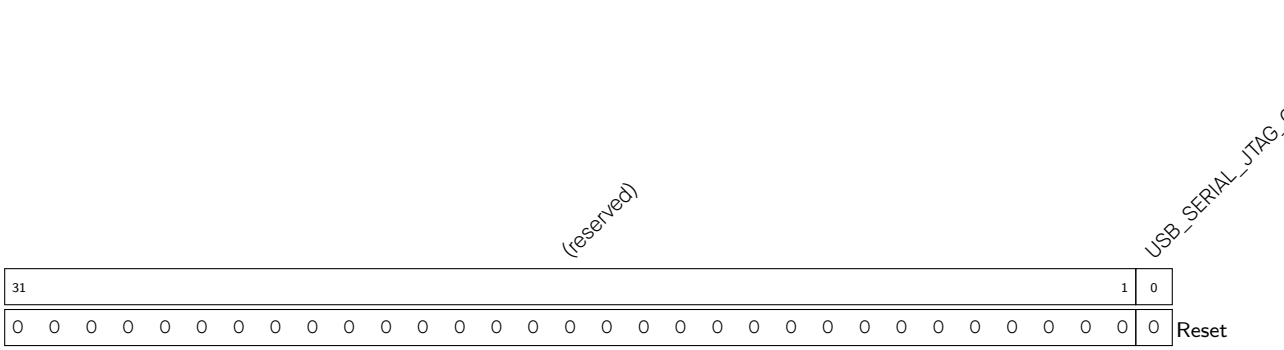


USB_SERIAL_JTAG_GET_BDATA_BITS Configures the value of bDataBits set by software, which is requested by GET_LINE_CODING command. (R/W)

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 36.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



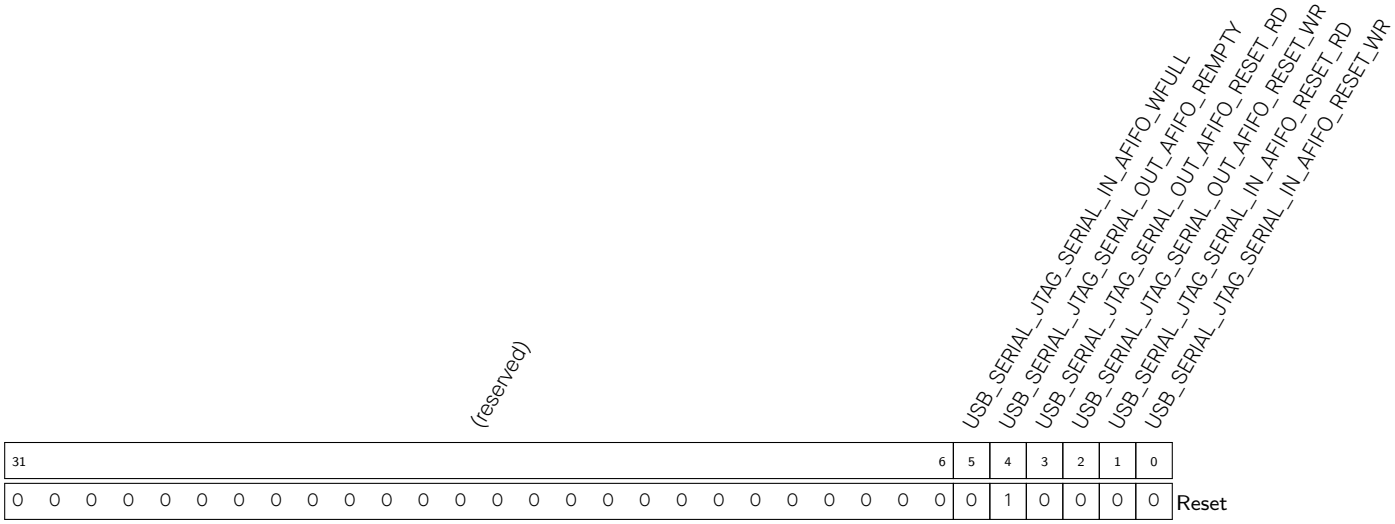
USB_SERIAL_JTAG_CONFIG_UPDATE Configures whether to update the value of configuration registers from APB clock domain to 48 MHz clock domain.

0: No effect

1: Update

(WT)

Register 36.11. 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

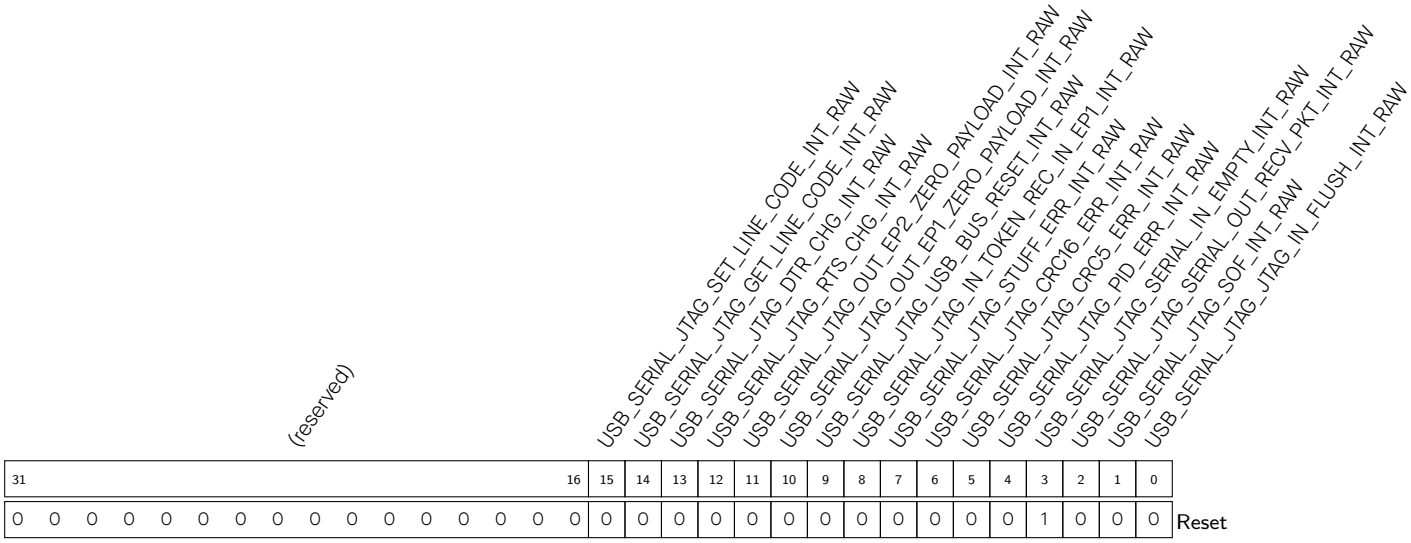
1: Reset

(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 36.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SOF_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/WTC/SS)

Continued on the next page...

Register 36.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

Continued from the previous page...

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/WTC/SS)

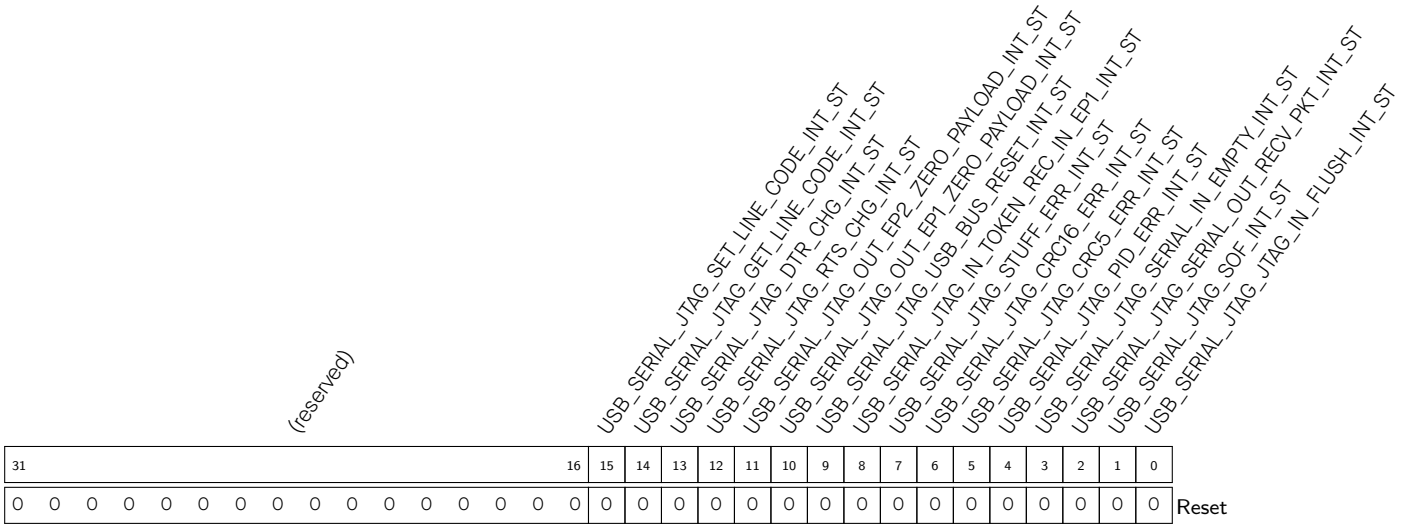
USB_SERIAL_JTAG_RTS_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW The raw interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/WTC/SS)

Register 36.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST The masked interrupt status of USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (RO)

USB_SERIAL_JTAG_SOF_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SOF_INT. (RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST The masked interrupt status of the USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_PID_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC5_ERR_INT. (RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_CRC16_ERR_INT. (RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST The masked interrupt status of USB_SERIAL_JTAG_STUFF_ERR_INT. (RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST The masked interrupt status of USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST The masked interrupt status of USB_SERIAL_JTAG_USB_BUS_RESET_INT. (RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT (RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST The masked interrupt status of USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (RO)

Continued on the next page...

Register 36.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

Continued from the previous page...

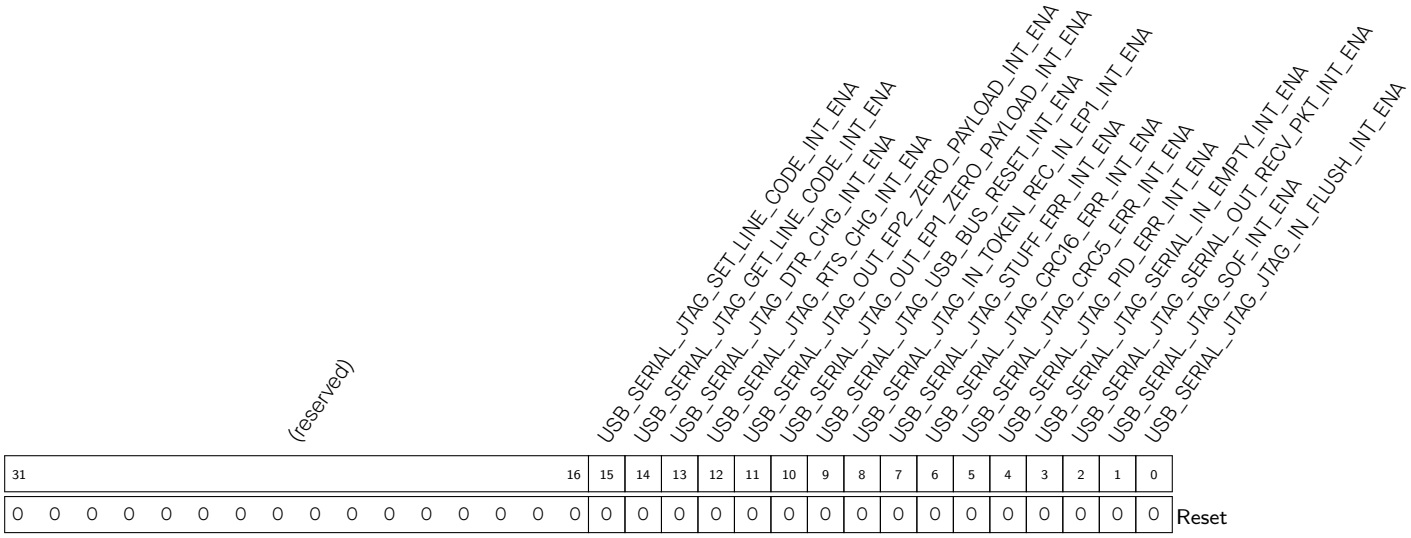
USB_SERIAL_JTAG_RTS_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_RTS_CHG_INT. (RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST The masked interrupt status of USB_SERIAL_JTAG_DTR_CHG_INT. (RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_GET_LINE_CODE_INT. (RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST The masked interrupt status of USB_SERIAL_JTAG_SET_LINE_CODE_INT. (RO)

Register 36.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA Write 1 to enable USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (R/W)

USB_SERIAL_JTAG_SOF_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SOF_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_PID_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC5_ERR_INT. (R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_CRC16_ERR_INT. (R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA Write 1 to enable USB_SERIAL_JTAG_STUFF_ERR_INT. (R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA Write 1 to enable USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA Write 1 to enable USB_SERIAL_JTAG_USB_BUS_RESET_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA Write 1 to enable USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (R/W)

Continued on the next page...

Register 36.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

Continued from the previous page...

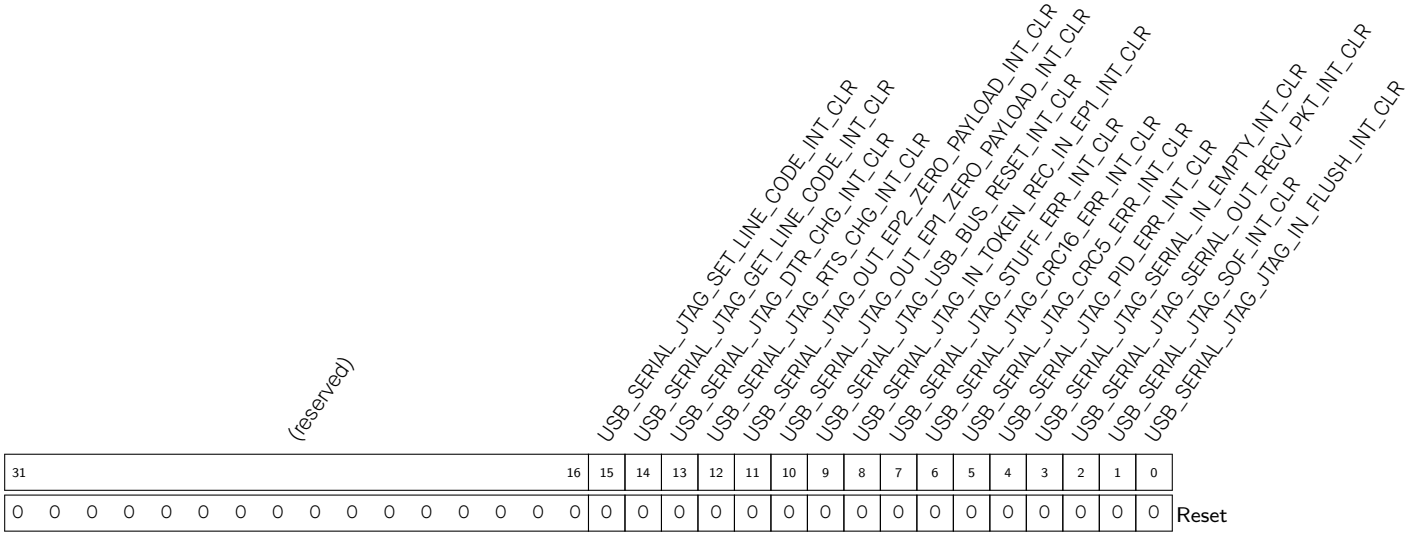
USB_SERIAL_JTAG_RTS_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_RTS_CHG_INT. (R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA Write 1 to enable USB_SERIAL_JTAG_DTR_CHG_INT. (R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_GET_LINE_CODE_INT. (R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA Write 1 to enable USB_SERIAL_JTAG_SET_LINE_CODE_INT. (R/W)

Register 36.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR Write 1 to clear USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT. (WT)

USB_SERIAL_JTAG_SOF_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SOF_INT. (WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT. (WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT. (WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_PID_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC5_ERR_INT. (WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_CRC16_ERR_INT. (WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR Write 1 to clear USB_SERIAL_JTAG_STUFF_ERR_INT. (WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR Write 1 to clear USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT. (WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR Write 1 to clear USB_SERIAL_JTAG_USB_BUS_RESET_INT. (WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT. (WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR Write 1 to clear USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT. (WT)

Continued on the next page...

Register 36.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

Continued from the previous page...

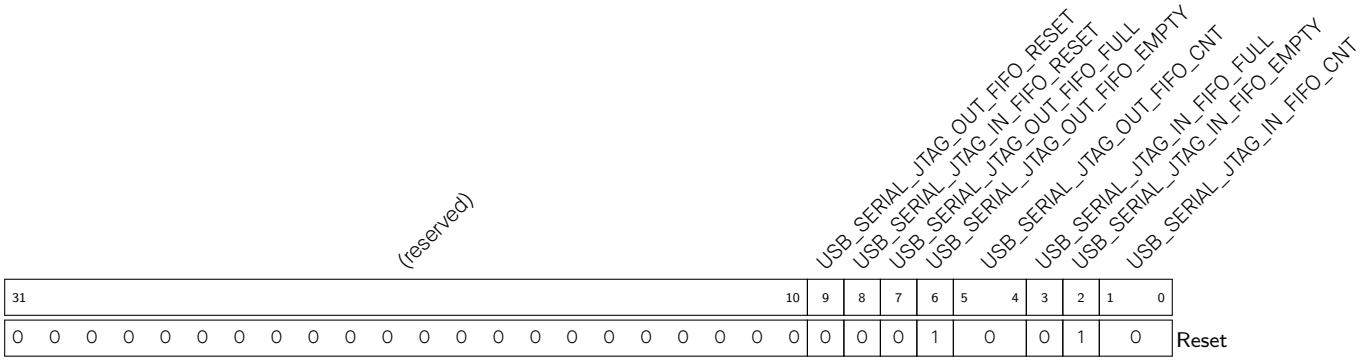
USB_SERIAL_JTAG_RTS_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_RTS_CHG_INT. (WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR Write 1 to clear USB_SERIAL_JTAG_DTR_CHG_INT. (WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_GET_LINE_CODE_INT. (WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR Write 1 to clear USB_SERIAL_JTAG_SET_LINE_CODE_INT. (WT)

Register 36.16. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT Represents JTAG IN FIFO counter. (RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY Represents whether JTAG IN FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL Represents whether JTAG IN FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT Represents JTAG OUT FIFO counter. (RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY Represents whether JTAG OUT FIFO is empty.

0: Not empty

1: Empty

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL Represents whether JTAG OUT FIFO is full.

0: Not full

1: Full

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET Configures whether to reset JTAG IN FIFO.

0: No effect

1: Reset

(R/W)

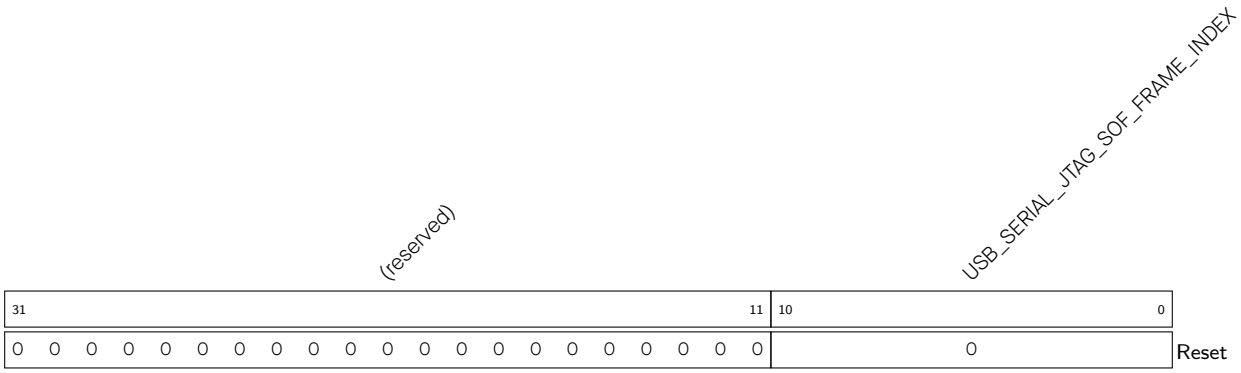
USB_SERIAL_JTAG_OUT_FIFO_RESET Configures whether to reset JTAG OUT FIFO.

0: No effect

1: Reset

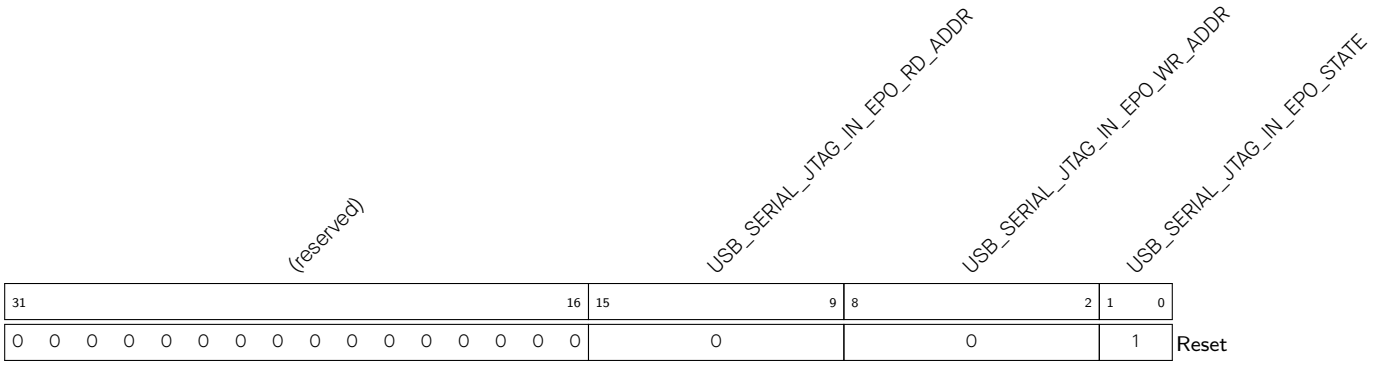
(R/W)

Register 36.17. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX Represents frame index of received SOF frame. (RO)

Register 36.18. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

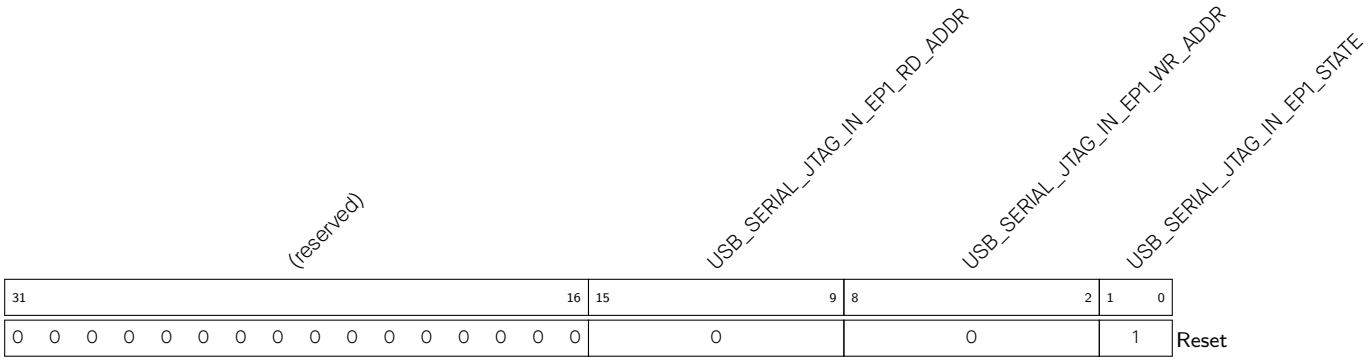


USB_SERIAL_JTAG_IN_EP0_STATE Represents state of IN Endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR Represents write data address of IN endpoint 0. (RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR Represents read data address of IN endpoint 0. (RO)

Register 36.19. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

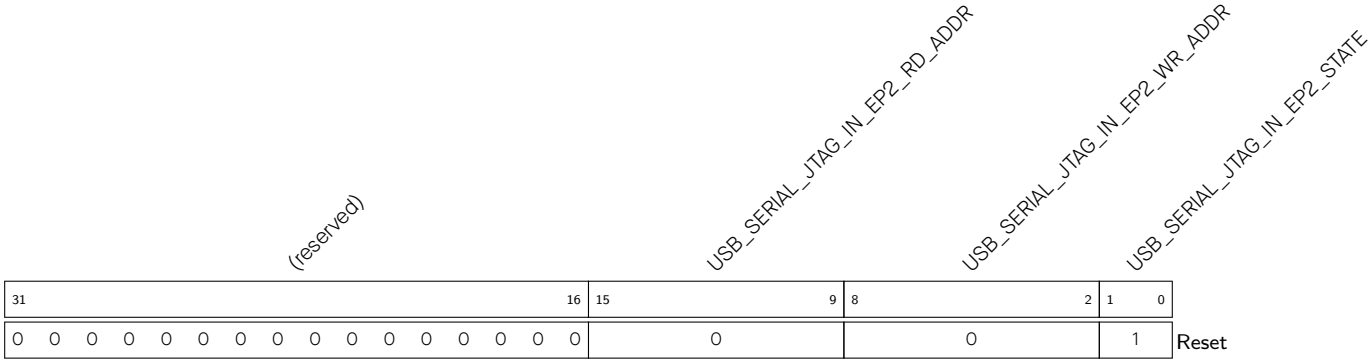


USB_SERIAL_JTAG_IN_EP1_STATE Represents state of IN Endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR Represents write data address of IN endpoint 1. (RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR Represents read data address of IN endpoint 1. (RO)

Register 36.20. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

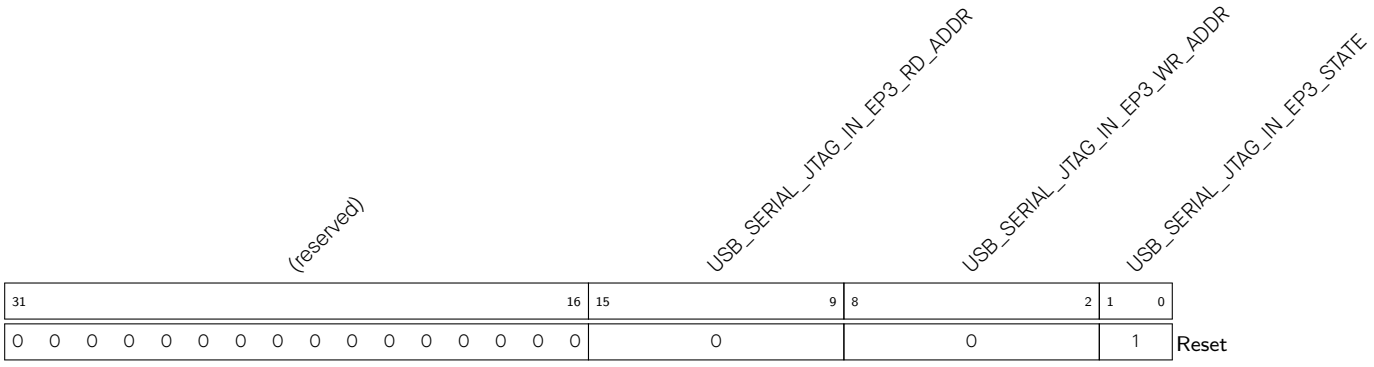


USB_SERIAL_JTAG_IN_EP2_STATE Represents state of IN Endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR Represents write data address of IN endpoint 2. (RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR Represents read data address of IN endpoint 2. (RO)

Register 36.21. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

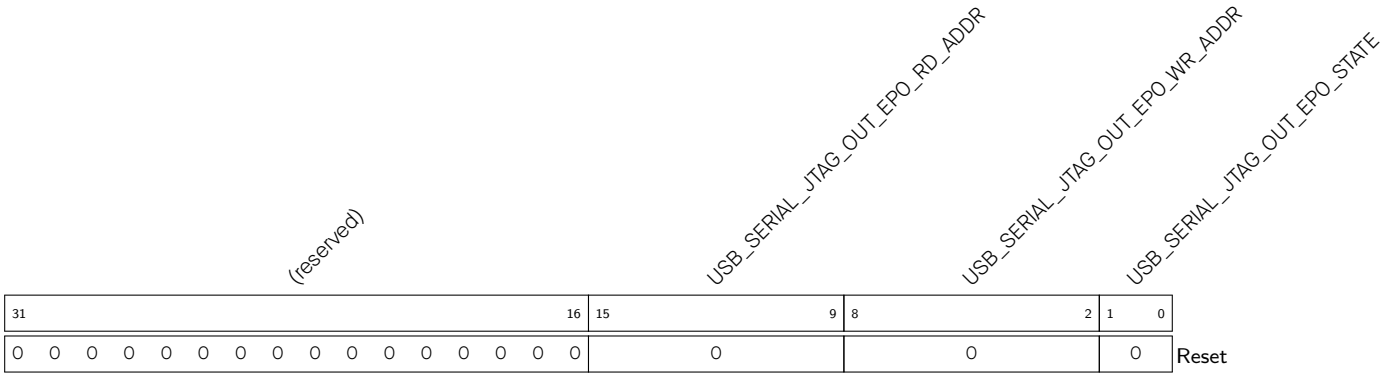


USB_SERIAL_JTAG_IN_EP3_STATE Represents state of IN Endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR Represents write data address of IN endpoint 3. (RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR Represents read data address of IN endpoint 3. (RO)

Register 36.22. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)



USB_SERIAL_JTAG_OUT_EP0_STATE Represents state of OUT Endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR Represents write data address of OUT endpoint 0. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) bytes data in OUT endpoint 0. (RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR Represents read data address of OUT endpoint 0. (RO)

Register 36.23. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



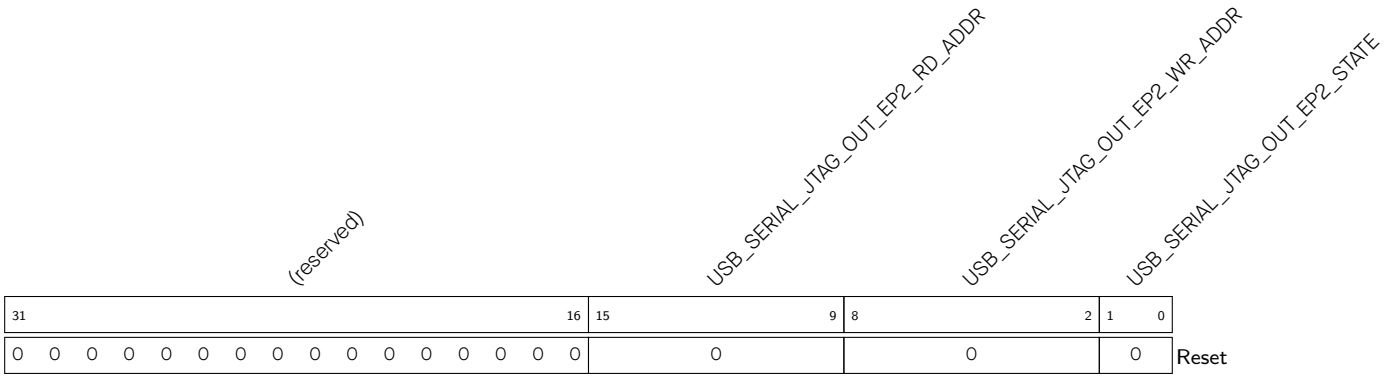
USB_SERIAL_JTAG_OUT_EP1_STATE Represents state of OUT Endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR Represents write data address of OUT endpoint 1. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected, there are (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR − 2) bytes data in OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR Represents read data address of OUT endpoint 1. (RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT Represents data count in OUT endpoint 1 when one packet is received. (RO)

Register 36.24. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)

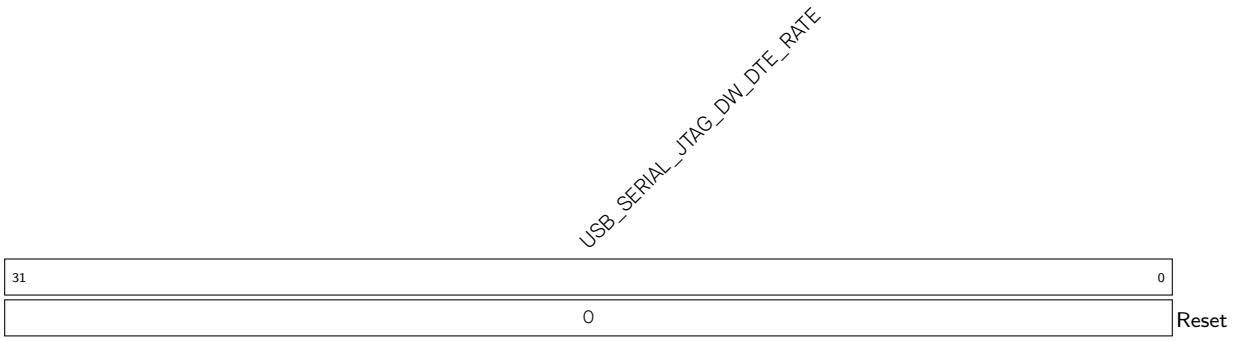


USB_SERIAL_JTAG_OUT_EP2_STATE Represents state of OUT Endpoint 2. (RO)

USB_SERIAL_JTAG_OUT_EP2_WR_ADDR Represents write data address of OUT endpoint 2. When USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT is detected there are (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR − 2) bytes data in OUT endpoint 2. (RO)

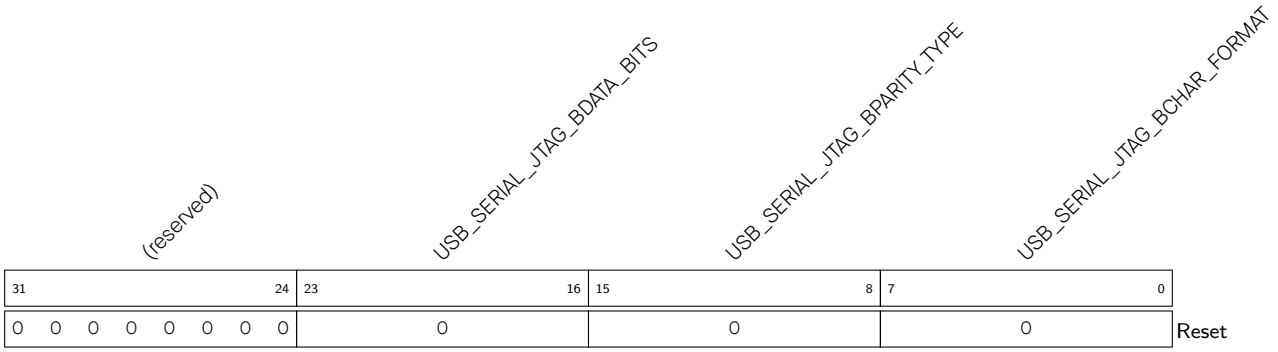
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR Represents read data address of OUT endpoint 2. (RO)

Register 36.25. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE Represents the value of dwDTERate set by host through SET_LINE_CODING command. (RO)

Register 36.26. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

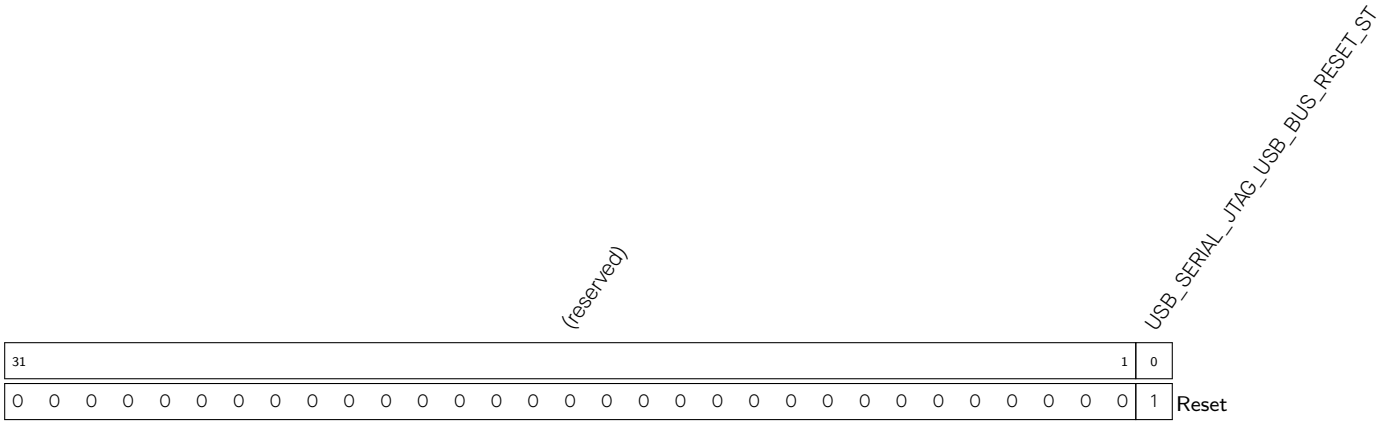


USB_SERIAL_JTAG_BCHAR_FORMAT Represents the value of bCharFormat set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BPARITY_TYPE Represents the value of bParityTpye set by host through SET_LINE_CODING command. (RO)

USB_SERIAL_JTAG_BDATA_BITS Represents the value of bDataBits set by host through SET_LINE_CODING command. (RO)

Register 36.27. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



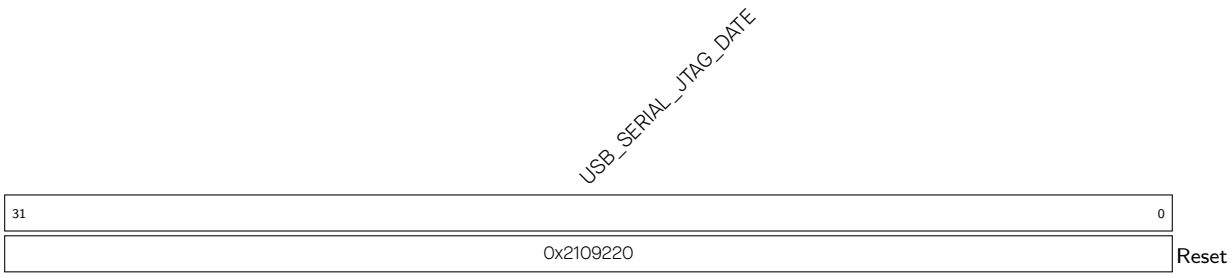
USB_SERIAL_JTAG_USB_BUS_RESET_ST Represents whether USB bus reset is released.

0: USB Serial/JTAG is in USB bus reset status

1: USB bus reset is released

(RO)

Register 36.28. USB_SERIAL_JTAG_DATE_REG (0x0088)



USB_SERIAL_JTAG_DATE Version control register. (R/W)

Chapter 37

Ethernet Media Access Controller (EMAC)

37.1 Overview

By using the external Ethernet PHY (physical layer), ESP32-P4 can send and receive data via Ethernet MAC (Media Access Controller) according to the IEEE 802.3 standard, as Figure 37.1-1 shows. Ethernet is currently the most commonly used network protocol that controls how data is transmitted over local- and wide-area networks, abbreviated as LAN and WAN, respectively.

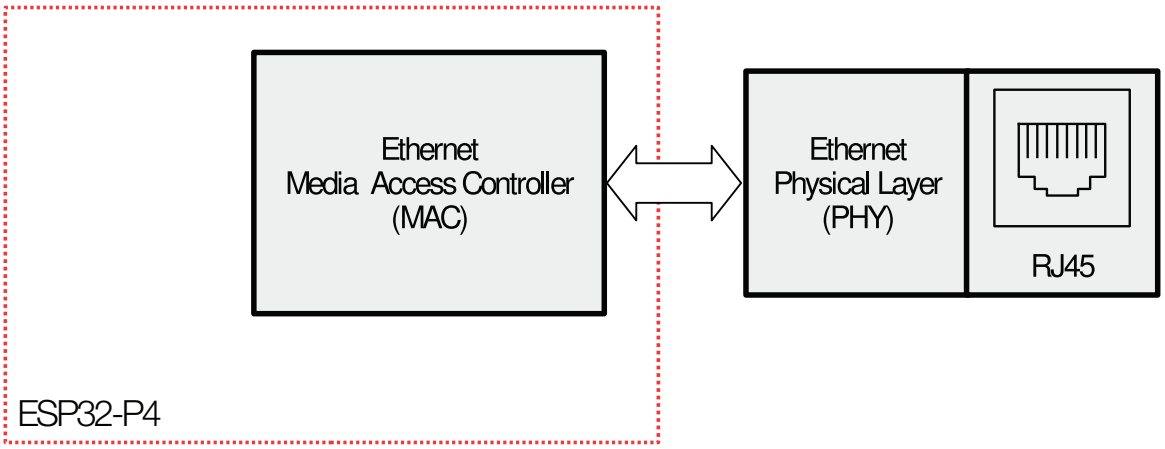


Figure 37.1-1. Ethernet MAC Functionality Overview

ESP32-P4 Ethernet MAC complies with the following standards:

- IEEE 802.3-2002 for Ethernet MAC
- IEEE 1588-2008 standard for precise networked clock synchronization
- IEEE 802.3 standard Media Independent Interface (MII) and Reduced Media Independent Interface (RMII)
- IEEE 802.3az-2010 for Energy Efficient Ethernet
- IEEE 802.1Q for VLAN frame format

37.2 Features

- Data rates of 10/100 Mbit/s through an external PHY interface
- Communication with an external Fast Ethernet PHY through IEEE 802.3-compliant MII and RMII interfaces
- Full-duplex and half-duplex modes

  - Carrier Sense Multiple Access or Collision Detection (CSMA/CD) protocol in half-duplex mode
  - IEEE 802.3x flow control in full-duplex mode
  - Optional forwarding of received pause control frame to the user application in full-duplex mode
  - Back-pressure flow control in half-duplex mode
  - Automatic transmission of zero-quanta pause frame on deassertion of flow control input in full-duplex mode
- Preamble and start-of-frame data (SFD) insertion in Transmit, and deletion in Receive paths
- Automatic CRC and padding (all 0) generation controllable on a per-frame basis
- Options for automatic padding generation for data below the minimum frame length
- Programmable frame length supporting jumbo frames of up to 16 KB
- Programmable inter-frame gap (IFG) from 40 to 96 bit times in steps of 8)
- Supports a variety of flexible address filtering modes:
  - Up to eight 48-bit perfect address filters with masks for each byte
  - Up to eight 48-bit source address (SA) address comparison check with masks for each byte
  - Option to pass all multicast addressed frames
  - Promiscuous mode support to pass all frames without any filtering for network monitoring
  - Passes all incoming packets (as per filter) with a status report
- Separate 32-bit status returned for transmission and reception packets
- Supports IEEE 802.1Q VLAN tag detection for reception frames
- Separate transmission, reception, and control interfaces for the application
- Management Data Input/Output (MDIO) interface for PHY device configuration and management
- Supports checksum off-load for received IPv4 and TCP packets encapsulated by the Ethernet frame
- Supports checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams
- Supports 64-bit timestamp on each transmit and receive frame (see IEEE 1588-2008)
- Supports Energy Efficient Ethernet (see IEEE 802.3az-2010)
- CRC replacement, SA insertion or replacement, and VLAN insertion, replacement or deletion in transmit frames
- Two FIFOs: 256-byte TX FIFO and 256-byte RX FIFO
- Receive status vectors inserted into RX FIFO after the EOF (end of frame) transfer, which enables multiple-frame storage in RX FIFO without requiring another FIFO to store those frames' receive status
- Option to forward good runt frames
- Supports statistics by generating pulses for frames dropped or corrupted due to overflow in RX FIFO
- Automatic re-transmission of collision frames (subject to certain conditions, see Section 37.4.1.1)

- Discarding frames in cases of late collisions, excessive collisions, excessive deferrals, and underflow conditions
- Software control to flush TX FIFO

37.3 Ethernet MAC Architecture

Figure 37.3-1 shows the block diagram of the Ethernet MAC.

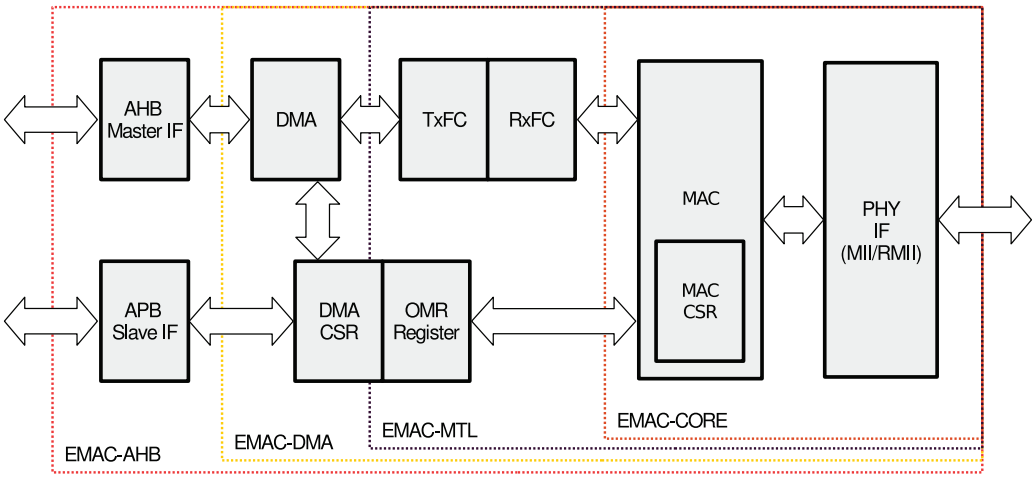


Figure 37.3-1. Ethernet MAC Block Diagram

Ethernet MAC consists of three layers: EMAC_CORE (MAC Core Layer), EMAC_MTL (MAC Transition Layer), and EMAC_DMA (Direct Memory Access). Each of these three layers has two directions: TX and RX. They are connected to the system through the Advanced High Performance Bus (AHB) and the Advanced Peripheral Bus (APB) on the chip. Off the chip, they communicate with the external PHY through the MII and RMII interfaces to establish an Ethernet connection.

37.4 Functional Description

37.4.1 EMAC_CORE

The MAC supports two interfaces (see Section 37.4.4) towards the PHY chip. The PHY interface can be selected only once after the chip reset. The MAC core communicates with the application side (DMA side) using the MAC Transmit Interface (MTI), MAC Receive Interface (MRI), and the MAC Control Interface (MCI).

37.4.1.1 Transmission

Transmission is initiated when the MTL application pushes in data with the SOF signal (start of frame) asserted. When the SOF signal is detected, the MAC accepts the data and begins transmitting to the RMII or MII. The time required to transmit the frame data to the RMII or MII after the application initiates transmission is variable, depending on delay factors like IFG delay, time to transmit preamble or SFD (Start Frame Delimiter), and any back-off delays for half-duplex mode. Until then, the MAC does not accept the data received from MTL by deasserting the ready signal.

After the EOF (end of frame) is transmitted to the MAC, the MAC completes normal transmission and then gives the Transmit Status to the MTL. If a normal collision (in half-duplex mode) occurs during transmission, the MAC core makes valid the Transmit Status to the MTL. It will then accept and drop all further data until the next SOF is received. The MTL block should retransmit the same frame from SOF on observing a retry request (in the Status) from the MAC.

The MAC issues an underflow status if the MTL is not able to provide the data continuously during the transmission. During the normal transmission of a frame from the MTL, if the MAC receives an SOF without getting an EOF for the previous frame, then the SOF is ignored and the new frame is considered as a continuation of the previous frame.

Transmit Flow Control

In full-duplex mode, when the Transmit Flow Control Enable bit (TFCE) is set to 1, the MAC generates pause frames and transmits them as necessary, with the calculated CRC appended.

Pause frame generation can be initiated in two ways. When the application sets the Flow Control Busy bit (FCBBA) to 1, or when the RX FIFO is full, a pause frame is transmitted.

- If the application has requested flow control by setting the FCBBA bit to 1, the MAC will generate and transmit a single pause frame. The value of the pause time in the generated frame is the pause time value programmed in the PAUSE_TIME field. If the application wants to extend or end the pause prior to the time specified in the previously transmitted pause frame, it must request another pause frame transmission after programming the pause time field PAUSE_TIME with an appropriate value.
- If the application has requested flow control when the RX FIFO is full, the MAC will generate and transmit a pause frame. The value of the pause time in the generated frame is the pause time value programmed in the PAUSE_TIME field. If the RX FIFO remains full at a configurable number of slot times (PLT) before the pause time runs out, a second pause frame will be transmitted. The process will be repeated as long as the RX FIFO remains full. If the FIFO is no longer full prior to the sampling time, the MAC will send a pause frame with zero pause time to indicate to the remote end that the RX buffer is ready to receive new data frames.

Retransmission During Collision

In half-duplex mode, a collision may occur on the MAC line interface when frames are transmitted to the MAC. The MAC may indicate a retry attempt by giving the status even before the EOF is transferred. Then the MAC will enable the retransmission by popping out the frame again from the FIFO. After more than 96 bytes are popped towards the MAC core, the FIFO controller frees the space in the FIFO, and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the MAC core indicates a late-collision event.

The MAC transmitter may abort the transmission of a frame because of collision, TX FIFO underflow, loss of carrier, jabber timeout, no carrier, excessive deferral, or late collision. When frame transmission is aborted because of collision, the MAC requests retransmission of the frame.

Transmit Status Word

At the end of the Ethernet frame transfer, the MAC outputs the transmit status to the application. The detailed description of the Transmit Status is the same as for TDES0.

37.4.1.2 Reception

A receive operation is initiated when the MAC detects an SFD on the RMII or MII. The MAC strips the preamble and SFD before proceeding to process the frame. The header fields are checked for filtering and the FCS (Frame Check Sequence) field used to verify the CRC for the frame. The received frame is stored in a shallow buffer until the address filtering is performed. The frame is dropped in the MAC if it fails the address filtering.

The frames received by the MAC are pushed into the RX FIFO. Once the RX FIFO crosses the configured Receive threshold (RX_THRESH_CTRL), its status is indicated to the DMA, so that the DMA can initiate a pre-configured burst transfers towards the AHB interface.

In the default cut-through mode, when the FIFO receives 64 bytes or a full packet of data, it pops out the data and indicates the availability to the DMA. Once the DMA initiates the transfer to the AHB interface, the data transmission continues from the FIFO until a complete packet has been transferred. Upon the completion of the EOF frame transfer, the status word will be popped out and transmitted to the DMA controller.

Receive Protocol

The receive module strips the preamble and SFD of the received frame. Once the SFD is detected, the MAC begins sending the Ethernet frame data to the RX FIFO, starting from the first byte following the SFD (destination address). If the IEEE 1588 timestamp feature is enabled, the snapshot of system time will be captured whenever an SFD of any frame is detected on MII, and sent to the application unless the MAC filters out and drops the frame.

If the Length/Type field of the receiving frame is less than 0x600 and if the MAC is programmed for the automatic CRC/paddingstripping option, the MAC sends the frame data up to the count specified in the Length/Type field to the RX FIFO, and then starts dropping bytes (including the FCS field). If the Length/Type field is greater than or equal to 0x600, the MAC will send all received Ethernet frame data to the RX FIFO, regardless of the value on the programmed automatic CRC stripping option. By default, the MAC is programmed for the watchdog timer to be enabled, that is frames above 2048 bytes (including DA, SA, LT, data, padding, and FCS) are cut off. This feature can be disabled by programming the Watchdog Disable bit EMACWATCHDOG. However, even if the watchdog timer is disabled, frames longer than 16 KB will still be cut off and the watchdog timeout status will be indicated.

Receive Frame Controller

If the RECEIVE_ALL bit in the MAC Frame Filter Register is reset, the MAC performs frame filtering based on the destination and source addresses. The application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, etc. On detecting a filter failure, the frame is dropped and not transmitted to the application. When the filter parameters change dynamically, if a DA and SA filter failure occurs, the reset of the frame is dropped and the Receive Status word (with zero frame length, CRC Error, and Runt Error bits set) is updated immediately indicating the filter failure.

Receive Flow Control

The MAC detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame (in full-duplex mode only). The Pause Frame Detection Function can be enabled or disabled with the RFCE bit. Once the receive flow control is enabled, the MAC starts monitoring the received frame destination address for any match with the multicast address of the control frame (0x0180 C200 0001).

If a match is detected (i.e., the destination address of the received frame matches the destination address of the reserved control frame), the MAC decides whether to transmit the received control frame to the application, according to the PCF field.

The MAC also decodes the type, the opcode, and the pause timer field of the receiving control Frame. If the byte count of the status indicates 64 bytes, and if there is no CRC error, the MAC transmitter will pause the transmission of any data frame for the duration of the decoded pause time value multiplied by the slot time (64 byte times for 10/100 Mbit/s mode). Meanwhile, if another pause frame is detected with a zero pause time, the MAC will reset the pause time and give another pause request.

If the received control frame matches neither the type field (0x8808), the opcode (0x00001), nor the byte length (64 bytes), or if there is a CRC error, the MAC will not generate a pause.

In the case of a pause frame with a multicast destination address, the MAC filters the frame according to the address match.

For a pause frame with a unicast destination address, the filtering depends on whether the DA matched the contents of the MAC Address Register 0 and the UPFD (detecting a pause frame even with a unicast destination address) is set. The PCF register controls the filtering for control frames in addition to the address filter.

Receive Operation Multiframe Handling

Since the status is available immediately following the data, the MAC is capable of storing any number of frames into the FIFO, as long as it is not full.

Error Handling

If the RX FIFO is full before it receives the EOF data from the MAC, an overflow will be declared and the whole frame will be dropped. The status bit RDESO[11] will reflect the fact that this frame is a partial frame due to overflow. The RX FIFO can perform the filtering of error frames and runt frames, if this function is enabled via FLUSH_TX_FIFO and FWD_UNDER_GF.

In cut-through mode, if a frame's status and length are available when reading an SOF from the RX FIFO, the whole error frame can be dropped. The DMA can flush the error frame being read from the FIFO by clearing FWD_ERR_FRAME to 0. The DMA then stops transferring data to the application, internally reads out the rest of the frame, and drops it. The MTL will then start the transfer of the next frame, if it is available. If FIFO is available, the transmission of the next frame will be initiated.

Receive Status Word

At the end of the Ethernet frame transfer, the MAC outputs the receive status to the application via EMAC_DMA. The detailed description of the receive status is the same for bit[31:0] in RDESO.

37.4.2 EMAC_MTL (MAC Transaction Layer)

The MAC Transaction Layer provides FIFO memory to buffer and regulate the frames between the application system memory and the MAC. It also enables the data to be transmitted between the application clock domain and the MAC clock domains. The MTL layer has two data paths, namely the Transmit path and the Receive path. The data path for both directions is 32-bit wide and operates with a simple FIFO protocol.

37.4.3 EMAC_DMA

EMAC has its own DMA module, with independent Transmit and Receive engines and a CSR (Control and Status Registers) space. The Transmit engine transfers data from the system memory to the device port (MTL), while the Receive engine transmits data from the device port to the system memory. The controller uses descriptors to efficiently move data from source to destination with minimal Host CPU intervention. The DMA is designed for packet-oriented data transfer such as frames in Ethernet. The controller can be programmed to interrupt the Host CPU for normal situations, such as the completion of frame transmission or reception, or error conditions.

37.4.3.1 Transmit Descriptors

Figure 37.4-1 shows the structure of the transmit descriptor linked list. Table 37.4-1 through Table 37.4-4 show the descriptions of each transmit descriptor.

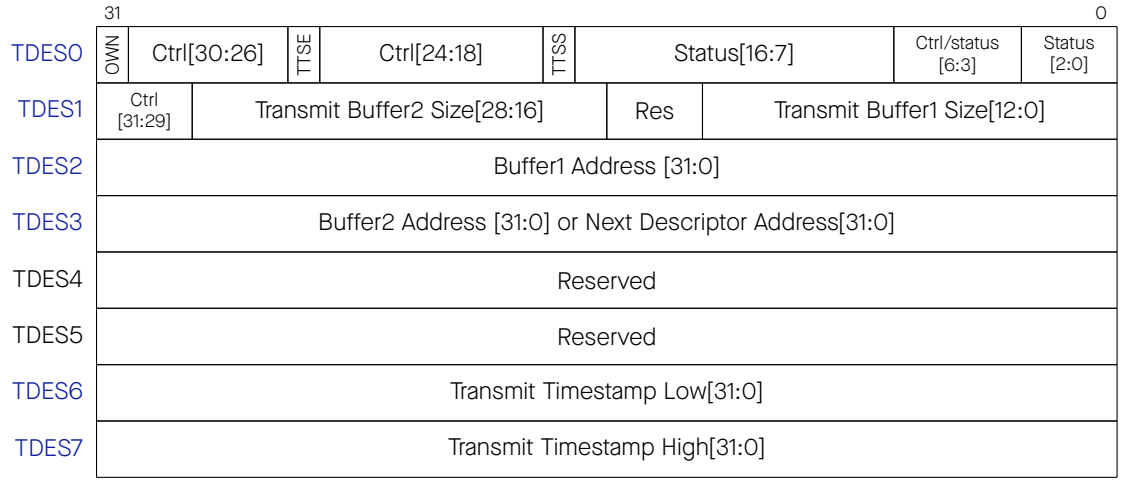


Figure 37.4-1. Transmit Descriptor Linked List

Table 37.4-1. Transmit Descriptor 0 (TDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA. When reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit, either when it completes the frame transmission or when the buffers allocated to the descriptor are empty. The ownership bit of the First Descriptor of the frame should be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
[30]	IC: Interrupt on Completion	When set, this bit sets the Transmit Interrupt TRANS_INT after the present frame has been transmitted. This bit is valid only when the last segment bit TDES0[29] is set.
[29]	LS: Last Segment	When set, this bit indicates that the buffer contains the last segment of the frame, and the TBS1 or TBS2 field in TDES1 should have a non-zero value.

Bits	Name	Description
[28]	FS: First Segment	When set, this bit indicates that the buffer contains the first segment of a frame.
[27]	DC: Disable CRC	When this bit is set, the MAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.
[26]	DP: Disable Padding	When this bit is set, the MAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.
[25]	TTSE: Transmit Time Stamp Enable	When this bit is set, the MAC enables IEEE1588 hardware timestamping for the transmit frame.
[24]	CRCR: CRC Replacement Control	When set, the MAC replaces the last four bytes of the transmitted packet with recalculated CRC bytes. The Host should ensure that the CRC bytes are present in the frame being transmitted from the Transmit Buffer. This bit is valid when the First Segment control bit (TDES0[28]) is set. In addition, CRC replacement is done only when Bit TDES0[27] is set to 1.
[23:22]	CIC: Checksum Insertion Control	These bits control the checksum calculation and insertion. Bit encodings are shown below: • 00: Checksum insertion is disabled. • 01: Only IP header checksum calculation and insertion are enabled. • 10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. • 11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware. This field is valid when the First Segment control bit (TDES0[28]) is set.
[21]	TER: Transmit End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[20]	TCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address, rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a "don't care" value. TDES0[21] takes precedence over TDES0[20]. This bit must be set to 1.

Bits	Name	Description
[19:18]	VLIC: VLAN Insertion Control	When set, these bits request the MAC to perform VLAN tagging or untagging before transmitting the frames. If the frame is modified for VLAN tags, the MAC automatically recalculates and replaces the CRC bytes. The following list describes the values of these bits: • 00: Do not add a VLAN tag. • 01: Remove the VLAN tag from the frames before transmission. This option should be used only with the VLAN frames. • 10: Insert a VLAN tag with the tag value programmed in VLAN Tag Inclusion or Replacement Register. • 11: Replace the VLAN tag in frames with the Tag value programmed in VLAN Tag Inclusion or Replacement Register. This option should be used only with the VLAN frames.
[17]	TTSS: Transmit Timestamp Status	This bit is used as a status bit to indicate that a timestamp was captured for the described transmit frame. When this bit is set, TDES6 and TDES7 have a timestamp value captured for the transmit frame. This bit is valid only when the descriptor's LS (TDES0[29]) is set.
[16]	IHE: IP Header Error	When set, this bit indicates that the MAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application, and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • TDES0[16]: IP Header Error • TDES0[14]: Jabber Timeout • TDES0[13]: Frame Flush • TDES0[12]: IP Payload Error • TDES0[11]: Loss of Carrier • TDES0[10]: No Carrier • TDES0[9]: Late Collision • TDES0[8]: Excessive Collision • TDES0[2]: Excessive Deferral • TDES0[1]: Underflow Error
[14]	JT: Jabber Timeout	When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit is only set when the EMACJABBER bit (disable jabber) of EMACCONFIG_REG is not set.
[13]	FF: Frame Flushed	When set, this bit indicates that the DMA or MTL flushed the frame because of a software Flush command given by the CPU.

Bits	Name	Description
[12]	IPE: IP Payload Error	When set, this bit indicates that the MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application, and issues an error status in case of a mismatch.
[11]	LOC: Loss of Carrier	When set, this bit indicates that a Loss of Carrier occurred during frame transmission (that is, the MII_CRS signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision and when the MAC operates in the half-duplex mode.
[10]	NC: No Carrier	When set, this bit indicates that the carrier sense signal from the PHY was not asserted during transmission.
[9]	LC: Late Collision	When set, this bit indicates that frame transmission was aborted because of a collision occurring after the collision window (in MII mode, 64 byte times including Preamble, and 512 byte times including Preamble and Carrier Extension). This bit is not valid if the Underflow Error bit is set.
[8]	EC: Excessive Collision	When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the EMACRETRY bit of EMACCONFIG_REG is set, this bit is set after the first collision, and the transmission of the frame is aborted.
[7]	VF: VLAN Frame	When set, this bit indicates that the transmitted frame was a VLAN-type frame.
[6:3]	Ctrl/status	These status bits indicate the number of collisions occurring before the frame was transmitted. This count is not valid when the Excessive Collisions bit (TDES0[8]) is set. The CPU updates this status field only in the half-duplex mode.
[2]	ED: Excessive Deferral	When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (if Jumbo Frame is enabled) if the EMACDEFERRAL bit (deferral check) of EMACCONFIG_REG is set high.
[1]	UF: Underflow Error	When set, this bit indicates that the MAC aborted the frame because the data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Transmit Underflow Register (Status Register TRANS_UNDFLOW) and Transmit Interrupt Register (Status Register TRANS_INT).
[0]	DB: Deferred Bit	When set, this bit indicates that the MAC defers before transmission because of the presence of a carrier. This bit is valid only in the half-duplex mode.

Table 37.4-2. Transmit Descriptor 1 (TDES1)

Bits	Name	Description
[31:29]	SAIC: SA Insertion Control	These bits request the MAC to add or replace the Source Address field in the Ethernet frame with the value given in EMACADDR0HIGH_REG, EMACADDR0LOW_REG, EMACADDR1HIGH_REG, and EMACADDR1LOW_REG. If the Source Address field is modified in a frame, the MAC automatically recalculates and replaces the CRC bytes. The Bit[31] specifies the MAC Address Register value (1 or 0) that is used for Source Address insertion or replacement. The following list describes the values of Bits[30:29]: • 00: Do not include the source address. • 01: Include or insert the source address. For reliable transmission, the application must provide frames without source addresses. • 10: Replace the source address. For reliable transmission, the application must provide frames with source addresses. • 11: Reserved. These bits are valid when the First Segment control bit (TDES0[28]) is set.
[28:16]	TBS2: Transmit Buffer2 Size	These bits specify the transmit Buffer 2 size in bytes. This field is not valid if TDES0[20] is set.
[15:13]	Reserved	Reserved
[12:0]	TBS1: Transmit Buffer 1 Size	These bits indicate the data buffer byte size in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

Table 37.4-3. Transmit Descriptor 2 (TDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 37.4-4. Transmit Descriptor 3 (TDES3)

Bits	Name	Description
[31:0]	Buffer2 Address Pointer (Next Descriptor Address)	These bits specify the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES0[20]) bit is set, this address specifies the physical memory where the Next Descriptor is present.

Table 37.4-5. Transmit Descriptor 6 (TDES6)

Bits	Name	Description
[31:0]	TTSL: Transmit Frame Time Stamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding transmit frame. This field has the timestamp only if the Last Segment (TDESO[29]) bit in the descriptor is set, and the Timestamp Status (TDESO[17]) bit is set.

Table 37.4-6. Transmit Descriptor 7 (TDES7)

Bits	Name	Description
[31:0]	TTSH: Transmit Frame Time Stamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field has the timestamp only if the Last Segment (TDESO[29]) bit in the descriptor is set, and the Timestamp Status (TDESO[17]) bit is set.

37.4.3.2 Receive Descriptors

Figure 37.4-2 shows the structure of the receive descriptor linked list. Table 37.4-7 through Table 37.4-11 show the descriptions of each receive descriptor.

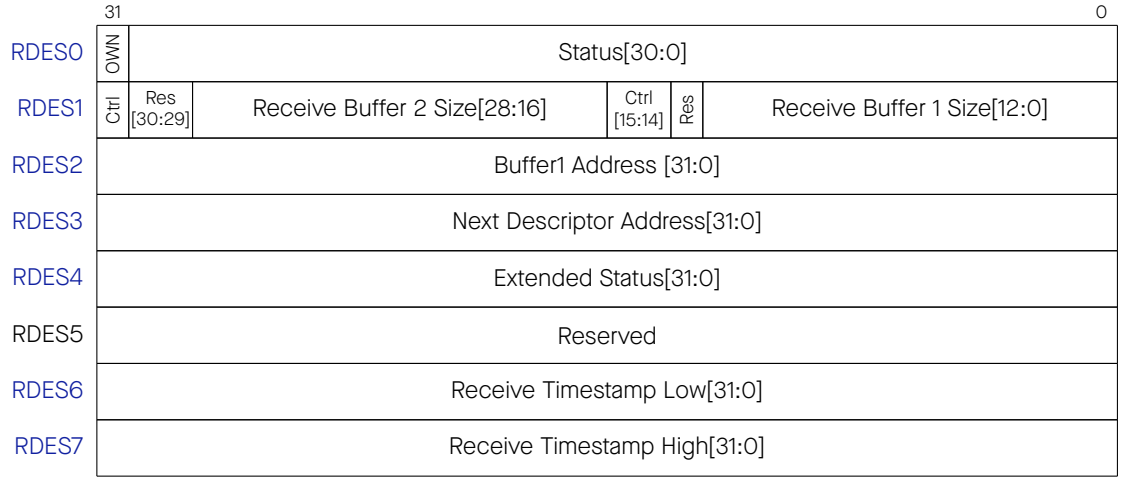


Figure 37.4-2. Receive Descriptor Linked List

Table 37.4-7. Receive Descriptor 0 (RDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA. When reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.

Bits	Name	Description
[30]	AFM: Destination Address Filter Fail	When set, this bit indicates a frame that failed in the DA Filter in the MAC.
[29:16]	FL: Frame Length	These bits indicate the byte length of the received frame that was transmitted to Host memory. This field is valid when Last Descriptor (RDESO[8]) is set and either the Descriptor Error (RDESO[14]) or Overflow Error bit RDESO[11] are reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • RDESO[1]: CRC Error • RDESO[3]: Receive Error • RDESO[4]: Watchdog Timeout • RDESO[6]: Late Collision • RDESO[7]: Giant Frame • RDES4[4:3]: IP Header or Payload Error • RDESO[11]: Overflow Error • RDESO[14]: Descriptor Error This field is valid only when RDESO[8] is set.
[14]	DE: Descriptor Error	When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when This field is valid only when RDESO[8] is set.
[13]	SAF: Source Address Filter Fail	When set, this bit indicates that the SA field of frame failed the SA Filter in the MAC.
[12]	LE: Length Error	When set, this bit indicates that the actual length of the frame received and that the Length/Type field does not match. This bit is valid only when the Frame Type (RDESO[5]) bit is reset.
[11]	OE: Overflow Error	When set, this bit indicates that the received frame was damaged because of buffer overflow in MTL.
[10]	VLAN: VLAN Tag	When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the MAC. The VLAN tagging depends on checking the VLAN fields of the received frame based on the VLAN Tag Register) configuration.
[9]	FS: First Descriptor	When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
[8]	LS: Last Descriptor	When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame.

Bits	Name	Description
[7]	IP Checksum Error (Type1), or Giant Frame	When the Advanced Timestamp feature is present, and when this bit set, it indicates that a snapshot of the Timestamp is written in descriptor word 6 (RDES6) and 7 (RDES7). This is valid only when the Last Descriptor bit (RDES0[8]) is set. When IP Checksum Engine (Type 1) is selected, this bit, if set, indicates one of the following: • The 16-bit IPv4 header checksum calculated by the core did not match the received checksum bytes. • The header checksum checking is bypassed for non-IPv4 frames. Otherwise, this bit, when set, indicates the Giant Frame Status. When normal frame processing is enabled, Giant frames are larger than 1,518 bytes (or 1,522 bytes for VLAN or 2,000 bytes when ASS2KP is set). When Jumbo Frame processing is enabled, Giant frames are larger than 9,018 bytes (or 9,022 bytes for VLAN).
[6]	LC: Late Collision	When set, this bit indicates that a late collision has occurred while receiving the frame in the half-duplex mode.
[5]	FT: Frame Type	When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than, or equal to, 1,536 bytes). When reset, it indicates that the received frame is an IEEE 802.3 frame. This bit is not valid for Runt frames which are less than 14 bytes.
[4]	RWT: Receive Watchdog Timeout	When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
[3]	RE: Receive Error	When set, this bit indicates that the MII_RXER signal is asserted while MII_RXDV is asserted during frame reception.
[2]	DE: Dribble Bit Error	When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in the MII Mode.
[1]	CE: CRC Error	When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.

Bits	Name	Description
[0]	Extended Status Available/ RX MAC Address	When the IP Checksum Offload (Type 2) is present, this bit, when set, indicates that the extended status is available in descriptor word 4 (RDES4). This is valid only when the Last Descriptor bit (RDES0[8]) is set. This bit is invalid when Bit 30 is set. When IP Checksum Offload (Type 2) is present, this bit is set even when the IP Checksum Offload engine bypasses the processing of the received frame. The bypassing may be because of a non-IP frame or an IP frame with a non-TCP/UDP/ICMP payload. When the IPC Full Offload is not selected, this bit indicates an RX MAC Address status. When set, this bit indicates that the RX MAC Address registers value (1 to 15) matched the frame's DA field. When reset, this bit indicates that the RX MAC Address Register 0 value matched the DA field.

Table 37.4-8. Receive Descriptor 1 (RDES1)

Bits	Name	Description
[31]	Ctrl	When set, this bit prevents setting the Status Register's RI bit (RECV_INT) for the received frame that ends in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to Host because of the RI for that frame.
[30:29]	Reserved	Reserved.
[28:16]	RBS2: Receive Buffer 2 Size	These bits indicate the size of Buffer 2 in bytes. The buffer size must be a multiple of 4, even if the value of RDES3 (Buffer 2 address pointer) is not aligned to bus width. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. This field is not valid if RDES1[14] is set.
[15]	RER: Receive End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[14]	RCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a "don't care" value. RDES1[15] takes precedence over RDES1[14].
[13]	Reserved	Reserved.
[12:0]	RBS1: Receive Buffer 1 Size	These bits indicate the size of Buffer 1 in bytes. The buffer size must be a multiple of 4, even if the value of RDES2 (buffer1 address pointer) is not aligned to bus width. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor depending on the value of RCH (Bit[14]).

Table 37.4-9. Receive Descriptor 2 (RDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 37.4-10. Receive Descriptor 3 (RDES3)

Bits	Name	Description
[31:0]	Buffer 2 Address Pointer (Next Descriptor Address)	These bits indicate the physical address of Buffer 2 when descriptor chaining is used. If the Second Address Chained (RDES1[14]) bit is set, then this address contains the pointer to the physical memory where the Next Descriptor is present.

Table 37.4-11. Receive Descriptor 4 (RDES4)

Bits	Name	Description
[31:28]	Reserved	Reserved.
[27:26]	Reserved	Reserved.
[25]	Reserved	Reserved.
[24]	Reserved	Reserved.
[23:21]	Reserved	Reserved.
[20:18]	Reserved	Reserved.
[17]	Reserved	Reserved.
[16]	Reserved	Reserved.
[15]	Reserved	Reserved.
[14]	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for the receive frame but got dropped in the MTL RX FIFO because of overflow.
[13]	PTP Version	When set, this bit indicates that the received PTP message is having the IEEE 1588 version 2 format. When reset, it has the version 1 format.
[12]	PTP Frame Type	When set, this bit indicates that the PTP message over Ethernet is sent directly over Ethernet. When this bit is not set and the MT (message type) is not 0, it indicates that the PTP message over UDP-IPv4 or UDP-IPv6 is received.

Bits	Name	Description
[11:8]	Message Type	These bits are encoded to give the type of the message received. • 0000: Reserved • 0001: SYNC (all clock types) • 0010: Follow_Up (all clock types) • 0011: Delay_Req (all clock types) • 0100: Delay_Resp (all clock types) • 0101: Pdelay_Req (in peer-to-peer transparent clock) • 0110: Pdelay_Resp (in peer-to-peer transparent clock) • 0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) • 1000: Announce • 1001: Management • 1010: Signaling • 1011-1110: Reserved • 1111: PTP packet with Reserved message type
[7]	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet. This bit is updated only when Bit[10] (IPC) of MAC Configuration Register (EMACRXIPCOFFLOAD) is set.
[6]	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet. This bit is updated only when Bit[10] (IPC) of MAC Configuration Register (EMACRXIPCOFFLOAD) is set.
[5]	IP Checksum Bypassed	When set, this bit indicates that the checksum offload engine is bypassed.
[4]	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the MAC core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field. This bit is valid when either Bit[7] or Bit[6] is set.
[3]	IP Header Error	When set, this bit indicates that either the 16-bit IPv4 header checksum calculated by the MAC core does not match the received checksum bytes, or the IP datagram version is not consistent with the Ethernet Type value. This bit is valid when either Bit[7] or Bit[6] is set.

Bits	Name	Description
[2:0]	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the Receive Checksum Offload Engine (COE). The COE also sets these bits to 00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP. • 000: Unknown or did not process IP payload • 001: UDP • 010: TCP • 011: ICMP • 1xx: Reserved This bit is valid when either Bit[7] or Bit[6] is set.

Table 37.4-12. Receive Descriptor 6 (RDES6)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Time Stamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

Table 37.4-13. Receive Descriptor 7 (RDES7)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Timestamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

37.4.4 PHY Interface

The MAC and the external PHY communicate through two interfaces:

- Media Independent Interface (MII)
- Reduced Media Independent Interface (RMII)

37.4.4.1 Media Independent Interface: MII

The Media Independent Interface (MII) defines the interconnection between MAC sublayers and PHYs at 10 Mbit/s and 100 Mbit/s.

MII Signals to PHY

The MII interface signals are shown in Figure 37.4-3.

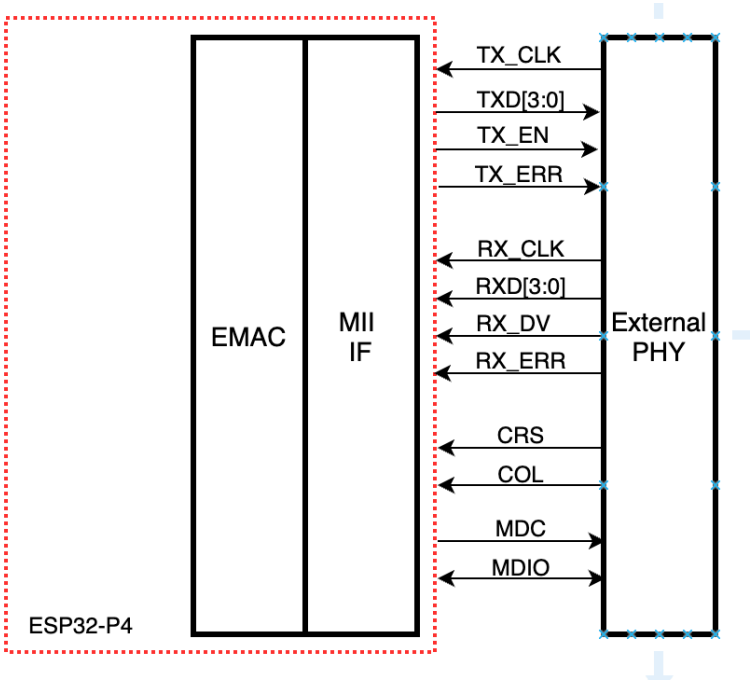


Figure 37.4-3. MII Interface

The MII interface signals are described below:

- MII_TX_CLK: TX clock signal. This signal provides the reference timing for TX data transmission. It has two frequencies: 2.5 MHz at 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_TXD[3:0]: A bundle of four transmit data signals driven by the MAC sublayer, and qualified (valid data) on the assertion of the MII_TX_EN signal. MII_TXD[0] is the lowest significant bit and MII_TXD[3] is the highest significant bit. When the MII_TX_EN signal is deasserted, the transmit data have no effect on the PHY.
- MII_TX_EN: Transmission enable signal. This signal indicates that the MAC is presenting nibbles (4 bits) on the MII for transmission. It must be asserted synchronously (MII_TX_CLK) with the first nibble of the preamble and must remain asserted while all nibbles to be transmitted are presented to the MII.
- MII_TX_ERR: Transmit error signal. The signal must be asserted for one or more clock periods (MII_TX_CLK) to indicate to the PHY layer that an error was detected somewhere in the frame.
- MII_RX_CLK: RX clock signal. This signal provides the reference timing for RX data transmission. It has two frequencies: 2.5 MHz at 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_RXD[3:0]: A bundle of four receive data signals driven by the MAC sublayer, and qualified (valid data) on the assertion of the MII_RX_DV signal. MII_RXD[0] is the lowest significant bit and MII_RXD[3] is the highest significant bit. When MII_RX_DV is deasserted and MII_RX_ERR is asserted, a specific MII_RXD[3:0] value is used to transfer specific information from the PHY.
- MII_RX_DV: Receive data valid signal. This signal indicates that the PHY is presenting recovered and decoded nibbles on the MII for reception. It must be asserted synchronously (MII_RX_CLK) with the first recovered nibble of the frame and must remain asserted through the final recovered nibble. It must be deasserted prior to the first clock cycle that follows the final nibble. In order to receive the frame correctly, the MII_RX_DV signal must encompass the frame, starting no later than the SFD field.

- MII_CRS: Carrier sense signal. When the transmit or receive medium is non-idle, the signal is asserted by the PHY. When the transmit or receive medium is idle, the signal is deasserted by the PHY. The PHY must ensure that the MII_CRS signal remains asserted throughout the duration of a collision condition. This signal does not need to be synchronized with the TX and RX clocks. In full duplex mode the state of this signal is don't care for the MAC sublayer.
- MII_COL: Collision detection signal. This signal must be asserted by the PHY upon detection of a collision on the medium and must remain asserted while the collision condition persists. This signal is not required to transition synchronously with respect to the TX and RX clocks. In full duplex mode, the state of this signal is don't care for the MAC sublayer.
- MII_RX_ERR: Receive error signal. The signal must be asserted for one or more clock periods (MII_RX_CLK) to indicate to the MAC sublayer that an error was detected somewhere in the frame.
- MDIO and MDC: Management data input/output and management data clock signal. The two signals constitute a serial bus defined for the Ethernet family of IEEE 802.3 standards, used to transfer control and data information to the PHY, see section Station Management Agent (SMA) Interface.

MII Clock

In MII mode, there are clock signals in two directions between MII and the PHY, namely clk_tx and clk_rx. clk_tx is used to synchronize the TX data, and clk_rx is used to synchronize the RX data. Both clocks are provided by the PHY.

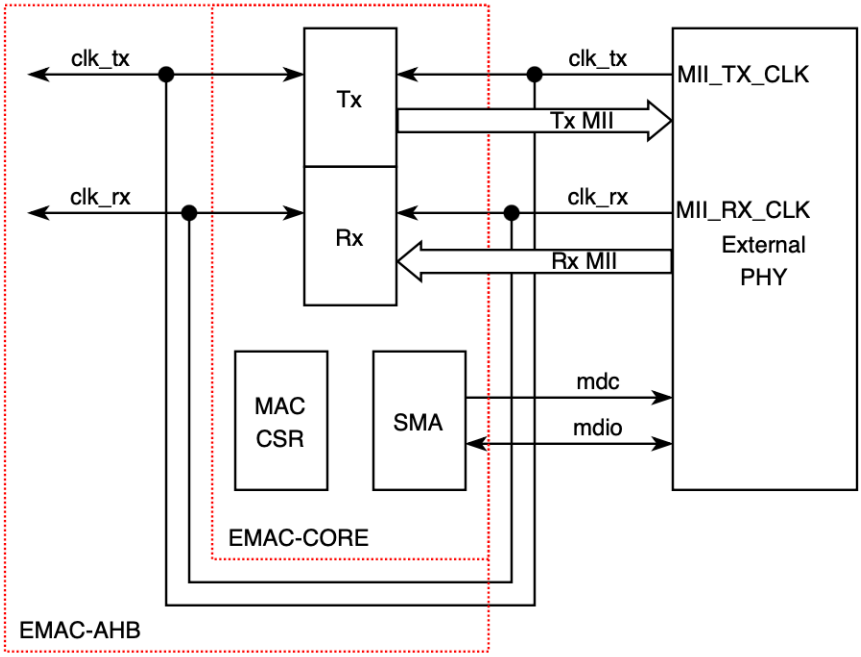


Figure 37.4-4. MII Clock

37.4.4.2 Reduced Media Independent Interface: RMII

RMII interface signals are shown in Figure 37.4-5.

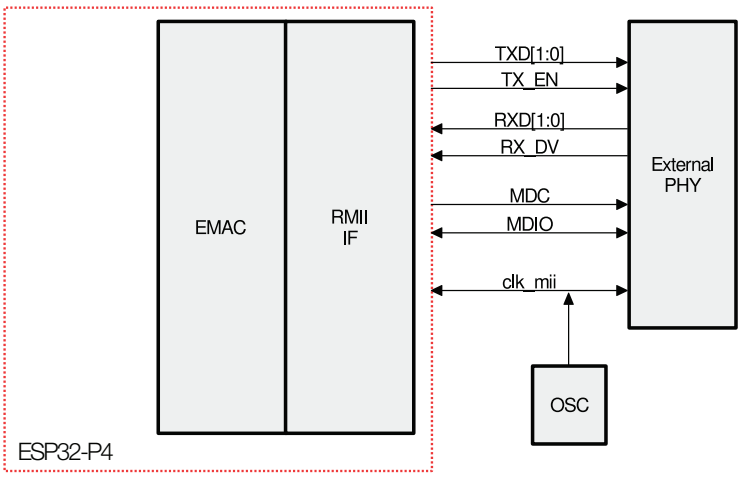


Figure 37.4-5. RMII Interface

RMII Signals

The Reduced Media Independent Interface (RMII) specification reduces the number of pins between the microcontroller's Ethernet MAC and the external PHY at 10 Mbit/s or 100 Mbit/s. According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. The RMII specification reduces the pin count to 7.

RMII has the following features:

- Supports an operating rate of 10 Mbit/s or 100 Mbit/s
- The clock reference must be 50 MHz.
- The same clock reference must be sourced externally both to the MAC and the external Ethernet PHY.
- Provides independent 2-bit wide TX and RX data paths.

RMII Clock

The 50 MHz RMII reference clock can be:

- Generated internally by ESP32-P4 and looped back via GPIOs. For GPIO configurations, please refer to Chapter 7 *GPIO Matrix and IO MUX*.
- Provided by the external crystal and obtained at GPIOs.

The RMII clock is shown in Figure 37.4-6.

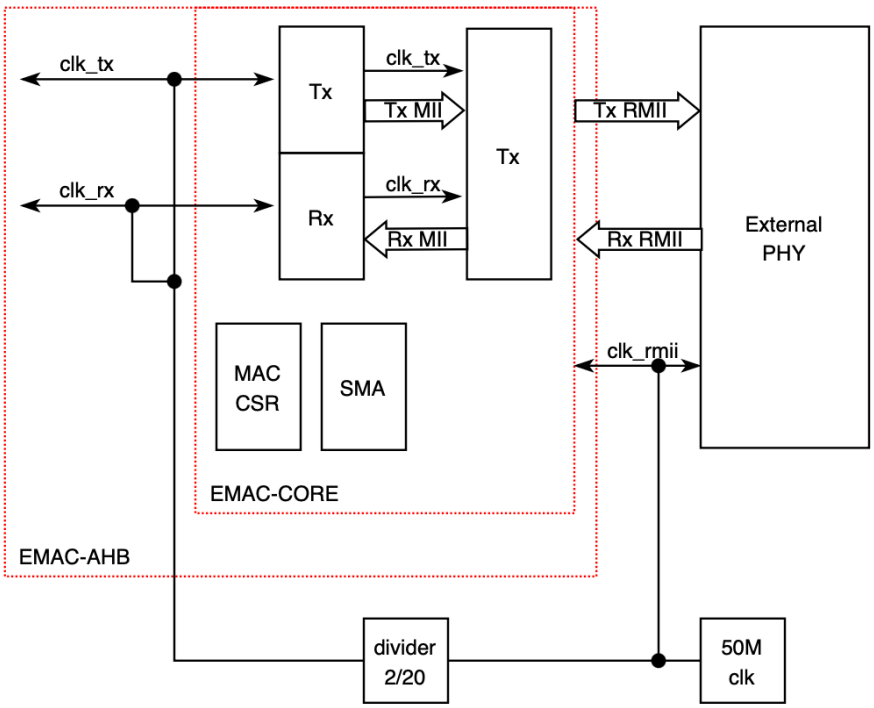


Figure 37.4-6. RMII Clock

37.4.4.3 Station Management Agent (SMA) Interface

As Figure 37.4-3 and Figure 37.4-5 show, the MAC uses MDC and MDIO signals to transfer control and data information to the PHY. The MDC clock signal is generated from the application clock by a clock divider at the maximum frequency of 2.5 MHz. The MDIO data signal transfers to and fro PHY synchronously with the MDC clock signal. Meanwhile, the PHY transmits register data.

- Read a PHY register:
  - Read MIIBUSY until the MII is in an idle state.
  - Configure MIIDEV to specify the PHY device to access.
  - Configure MIIREG to specify the address of the PHY register to access.
  - Configure MIICSRCLK to set the APB clock frequency.
  - Set MIIWRITE to 0 for read access.
  - Read MIIBUSY until the MII is in an idle state.
  - Read EMACMIIDATA_REG to retrieve the value of the PHY register.
- Write a PHY register:
  - Read MIIBUSY until the MII is in an idle state.
  - Configure EMACMIIDATA_REG to set the value to be written to the PHY register.
  - Configure MIIDEV to specify the PHY device to access.
  - Configure MIIREG to specify the address of the PHY register to access.
  - Configure MIICSRCLK to set the APB clock frequency.

- Set MIIWRITE to 1 for write access.
- Read MIIBUSY until the MII is in an idle state, which indicates the completion of the write operation.

37.4.5 MAC Address Filtering

Address filtering checks the destination and source addresses on all received frames and reports the address filtering status accordingly. Address checking is based on different parameters (Frame filter register) chosen by the application. The filtered frame can also be identified: multicast or broadcast frame. Address filtering uses the station's physical (MAC) address for address checking.

37.4.5.1 Unicast Destination Address Filtering

The MAC supports up to 9 MAC addresses for unicast perfect filtering. MAC compares all 48 bits of the received unicast address with the programmed MAC address for any match. By default, EMACADDR0 is always enabled, and the other addresses (EMACADDR0 ~ EMACADDR8) are selected with an individual enable bit. Each byte of these other addresses (EMACADDR0 ~ EMACADDR8) can be masked during comparison with the corresponding received DA byte by setting the corresponding Mask Byte Control bit in the register. This helps group address filtering for the DA.

37.4.5.2 Multicast Destination Address Filtering

The MAC can be programmed to pass all multicast frames by setting the Pass All Multicast (PAM) bit in the Frame Filter Register to 1. If the PAM bit is reset, the MAC the filtering for multicast addresses.

In perfect filtering mode, the multicast address is compared with the programmed MAC Destination Address Registers (EMACADDR0 ~ EMACADDR8). Group address filtering is also supported.

37.4.5.3 Broadcast Address Filter

The MAC does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames by setting the Disable Broadcast Frames (DBF) bit in the Frame Filter Register to 1, all broadcast frames will be dropped.

37.4.5.4 Unicast Source Address Filter

The MAC can also perform perfect filtering based on the source address field of the received frames. By default, the Address Filtering Module (AFM) compares the Source Address (SA) field with the values programmed in the SA register. The MAC address registers [1:3] can be configured to contain SA instead of DA for comparison, by setting bit[30] in the corresponding register. Group filtering with SA is also supported. The frames that fail the SA filter are dropped by the MAC if the Source Address Filter Enable (SAFE) bit in the Frame filter register is set. Otherwise, the result of the SA filter is given as a status bit in the Receive Status word (see Table 37.4-7).

When the SAFE bit is set to 1, the result of the SA filtering and DA filtering is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter failure results will drop the frame. Both filters have to pass the frame for the frame to be forwarded to the application.

37.4.5.5 Inverse Filtering Operation

For both destination and source address filtering, there is an option to invert the filter-match result at the final output. These are controlled by the DAIF and SAIF bits in the Frame filter register, respectively. The DAIF bit is applicable for both Unicast and Multicast DA frames. When the DAIF bit is set, the result of the unicast/multicast destination address filter is inverted. Similarly, when the SAIF bit is set, the result of the unicast SA filter is inverted.

The following two tables summarize the destination address and source address filtering based on the type of the frames received.

Table 37.4-14. Destination Address Filtering

Frame Type	PM	PF	DAIF	PAM	DB	DA Filter Operation
Broadcast	1	X	X	X	X	Pass
	0	X	X	X	0	Pass
	0	X	X	X	1	Fail
Unicast	1	X	X	X	X	Pass all frames
	0	X	0	X	X	Pass on perfect/group filter match
	0	X	1	X	X	Fail on perfect/group filter match
	0	1	0	X	X	Pass on perfect/group filter match
	0	1	1	X	X	Fail on perfect/group filter match
Multicast	1	X	X	X	X	Pass all frames
	X	X	X	1	X	Pass all frames
	0	X	0	0	X	Pass on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	1	0	0	X	Pass on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	X	1	0	X	Fail on perfect/group filter match and drop pause control frames if PCF = 0x.
	0	1	1	0	X	Fail on perfect/group filter match and drop pause control frames if PCF = 0x.

The filtering parameters in the MAC Frame Filter Register described in Table 37.4-14 are as follows.

Parameter name		Parameter setting	
PM:	Pass All Multicast	1:	Set
PF:	Perfect Filter	0:	Cleared
DAIF:	Destination Address Inverse Filtering	X:	Don't care
PAM:	Pass All Multicast		
DB:	Disable Broadcast Frames		

Table 37.4-15. Source Address Filtering

Frame Type	PM	SAIF	SAF	Source Address Filter Operation
	1	X	X	Pass all frames
	0	0	0	Pass on perfect/group filter match but do not drop frames that fail
	0	0	0	Fail on perfect/group filter match but do not drop frames that fail
	0	0	0	Pass on perfect/group filter match and drop frames that fail
Unicast	0	0	0	Fail on perfect/group filter match and drop frames that fail

The filtering parameters in the MAC Frame Filter Register described in Table 37.4-15 are as follows.

Parameter name		Parameter setting	
PM:	Pass All Multicast	1:	Set
SAF:	Source Address Filtering	0:	Cleared
SAIF:	Source Address Inverse Filtering	X:	Don't care

37.4.6 Energy Efficient Ethernet (EEE)

Energy Efficient Ethernet (EEE) is an optional operating mode that enables the MAC sublayer along with a series of physical layers to operate in the Low Power Idle (LPI) mode. The EEE mode supports 100 Mbps MAC operations compliant with the IEEE 802.3az-2010 standard. The EEE is supported only in the full-duplex mode when MAC uses MII.

The LPI mode allows power saving by switching off parts of the communication device functionality when there is no data to be transmitted and received. The system on both sides of the link can disable some functionalities and save power during the period of low-link utilization. The MAC controls whether the system should enter or exit the LPI mode and communicate this to the PHY.

37.4.6.1 Transmission

To enter the LPI mode, the software must set the LPIEN bit to indicate to the MAC to stop transmission and initiate the LPI protocol. The MAC completes the transmission in progress, generates its transmission status, and then starts transmitting the LPI pattern instead of the IDLE pattern during the Interframe gap (IFG). The MAC then updates TLPIEN and generates an interrupt.

To exit the LPI mode, the software must clear the LPIEN bit to indicate to the MAC. The MAC stops transmitting the LPI pattern, and starts transmitting the IDLE pattern. When the LPI_TW_TIMER expires, the MAC updates TLPIEX and generates an interrupt.

37.4.6.2 Reception

When the PHY receives the signals from the link partner to enter into the LPI state, the PHY starts transmitting the LPI pattern. The MAC updates RLPIEN and generates an interrupt.

When the PHY receives signals from the link partner to exit the LPI state, the PHY stops transmitting the LPI pattern. The MAC updates RLPIEX and generates an interrupt.

37.4.7 Source Address, VLAN, and CRC Control

37.4.7.1 Source Address Control

The software can enable source address insertion or replacement for all or some frames:

- Enable source address insertion or replacement for all frames by configuring SAIRC.
- Enable source address insertion or replacement for the current frame by configuring TDES1[31:29] of the frame. When TDES1[31] is set, MAC_ADDRESS1_HI and EMACADDR1LOW_REG are used for insertion and replacement. When TDES1[31] is cleared, MAC_ADDRESSO_HI and EMACADDROLOW_REG are used for insertion and replacement. When ADDRESS_ENABLE1 is not set, MAC_ADDRESSO_HI and EMACADDROLOW_REG are used for insertion and replacement irrespective of TDES1[31].

37.4.7.2 VLAN Control

The software can enable VLAN insertion, replacement, or deletion for all or some frames:

- Enable VLAN insertion, replacement, or deletion for all frames by configuring VLC.
- Enable VLAN insertion, replacement, or deletion for the current frame by configuring TDESO[19:18] of the frame.

37.4.7.3 CRC Control

CRC replacement is a feature that the software can use to indicate the MAC to replace the FCS field of the frame being transmitted with the calculated CRC. This feature can be enabled for the current frame by configuring TDESO[24].

CRC replacement is valid only when MAC does not append CRC (TDESO[27] is set), i.e., when FCS is provided by software in frames transmitted. If source address or VLAN control is enabled, the MAC will replace or append FCS fields with CRC when TDESO[27] is set or cleared respectively.

37.4.8 Time Stamp

The IEEE 1588-2008 standard defines the Precision Time Protocol (PTP) that allows precise clock synchronization in measurement and control systems implemented with technologies such as local computing, network communication and distributed objects. it supports system-wide synchronization accuracy in the sub-microsecond range with a minimum network and local clock computing resources.

37.4.8.1 Transmit Path

The MAC captures a timestamp when the start-of-frame data (SFD) is sent on the MII interface. The frames for which timestamps are captured can be controlled on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp should be captured for that frame. The MAC does not process the transmitted to identity PTP frames. Rather, it captures the timestamp or not according to the configurations of TDESO[25] by software. The MAC returns the timestamp to the software inside the corresponding transmit descriptor, thus connecting the timestamp automatically to the specific PTP frame. The 64-bit timestamp information is written to the TDES6 and TDES7 fields.

37.4.8.2 Receive Path

The MAC processes the received frames to identify valid PTP frames. The snapshot of the time to be sent to the application can be controlled by software via EMACTSTPCTRL_REG.

The MAC returns the timestamp to the software inside the corresponding receive descriptor. The extended status (including the timestamp status and IPC status) is written to RDES4, and the timestamp snapshot is written to RDES6 and RDES7.

37.4.9 Remote Wakeup

The MAC supports remote wakeup. When the receiver receives a remote wakeup frame or a magic packet, an interrupt will be triggered, and the PMU will be notified to wake up the SoC system.

The software can control whether to use remote wakeup frames to trigger interrupts by configuring RWKPKTEN , whether to use magic packets to trigger interrupts by configuring MGKPKTEN, and whether the MAC drops all received frames in power-down mode by configuring PWRDWN.

37.4.10 Good Transmitted and Received Frames

A frame that has been successfully transmitted is considered a "good frame". In other words, a transmitted frame is considered to be "good", if the frame transmission is not aborted due to the following errors:

- Jabber timeout (TDES0[14])
- No carrier or loss of carrier (TDES0[11:10])
- Late collision (TDES0[9])
- Frame underflow (TDES0[1])
- Excessive deferral (TDES0[2])
- Excessive collision (TDES0[8])

A received frame is considered to be "good" if the following errors do not occur:

- CRC error (RDES0[1])
- Runt frames (frames shorter than 64 bytes)
- Alignment error (in 10/100 Mbps modes only) (RDES0[2])
- Length error (non-type frames only) (RDES0[12])
- Frame size over the maximum size (for non-type frames over the maximum frame size only)) (RDES0[7])
- MII_RXER input error (RDES0[3])

The maximum frame size depends on the frame type:

- The maximum size of untagged frames = 1518 bytes
- The maximum size of VLAN frames = 1522 bytes

37.5 Programming Procedures

37.5.1 MAC System Layer Configuration

1. Configure the IO interfaces. For detailed configurations about the MII and the RMII interface, see Chapter *7 GPIO Matrix and IO MUX*.

2. Configure the EMAC system clock by setting HP_SYS_CLKRST_EMAC_SYS_CLK_EN to 1.
3. Configure the PHY interface clock:
   - RMII interface: select the RMII interface by configuring HP_SYS_PHY_INTF_SEL to 4.
     - Select RMII reference clock source:
       * Reference clock sourced from the external crystal:
         · Configure HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN to disable PAD_EMAC_REF_CLK outputted to GPIO.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_EN to disable PLL_F50M_CLK.
       * Reference clock sourced from ESP32-P4:
         · Configure HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN to enable PAD_EMAC_REF_CLK outputted to GPIO.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM to specify the clock divisor for PLL_F50M_CLK.
         · Configure HP_SYS_CLKRST_REF_50M_CLK_EN to enable PLL_F50M_CLK.
     - Configure HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL to specify the clock source of EMAC_RMII_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL to specify the clock source of EMAC_RX_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM to specify the clock divisor for EMAC_RX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL to specify the clock source of EMAC_TX_CLK as PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM to specify the clock divisor for EMAC_TX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure LP_AONCLKRST_HP_PAD_EMAC_TXRX_CLK_EN to enable the clock from IO pin PAD_EMAC_TXRX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RMII_CLK_EN to enable EMAC_RMII_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_EN to enable EMAC_RX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_EN to enable EMAC_TX_CLK;
   - MII interface: select the MII interface by configuring HP_SYS_PHY_INTF_SEL to 0.
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL to specify EMAC_RX_CLK as PAD_EMAC_RX_CLK;
     - Configure HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM to specify the clock divisor for EMAC_RX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
     - Configure HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL to specify EMAC_TX_CLK as PAD_EMAC_TX_CLK;

  - Configure HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM to specify the clock divisor for EMAC_TX_CLK according to the EMAC line speed (10 Mbit/s or 100 Mbit/s);
  - Configure LP_AONCLKRST_HP_PAD_EMAC_RX_CLK_EN to enable the clock from IO pin PAD_EMAC_RX_CLK;
  - Configure LP_AONCLKRST_HP_PAD_EMAC_TX_CLK_EN to enable the clock from IO pin PAD_EMAC_TX_CLK;
  - Configure HP_SYS_CLKRST_EMAC_RX_CLK_EN to enable EMAC_RX_CLK;
  - Configure HP_SYS_CLKRST_EMAC_TX_CLK_EN to enable EMAC_TX_CLK;

37.5.2 EMAC Initial Configuration

Initialize EMAC_DMA:

- Reset EMAC by setting SW_RST to 1.
- Wait for the completion of reset until reading 0 from SW_RST, which indicates that the reset is done.
- Read AHB_ST; a value of 0 indicates that all AHB bus transactions have been completed.
- Configure the AHB bus mode for EMAC_DMA via DMABUSMODE_REG.
- Prepare the linked list and configure the linked list base address via DMARXBASEADDR_REG and DMATXBASEADDR_REG.
- Configure the operating mode for EMAC_DMA and EMAC_MTL via DMAOPERATION_MODE_REG.
- Configure whether to enable DMA interrupts via DMAIN_EN_REG.

Initialize EMAC_CORE:

- Read EMACMIIADDR_REG and EMACMIIDATA_REG for the PHY's connection status, operating frequency, and operating mode.
- Configure EMACADDROHIGH_REG and EMACADDROLOW_REG to specify the MAC address.
- Configure EMACFF_REG to select the packet filtering mode.
- Configure EMACFC_REG for flow control.
- Configure whether to mask MAC interrupts via EMACINTMASK_REG.
- Configure the MAC's operating mode for transmission and reception via EMACCONFIG_REG.

37.5.3 Starting Transmission

- Start EMAC_DMA transmission by setting START_STOP_TRANSMISSION_COMMAND to 1.
- Start EMAC_CORE transmission by setting EMACTX to 1.
- Detect the TRANS_INT interrupt if enabled, and wait for the frame transmission to complete.

37.5.4 Starting Reception

- Start EMAC_DMA reception by setting START_STOP_RX to 1.

- Start EMAC_CORE reception by setting EMACRX to 1.
- Detect the RECV_INT interrupt if enabled, and wait for the frame reception to complete.

37.5.5 TX Entering and Exiting the LPI State

- Configure the LPI timer counter via EMACLPITIMERSCONTROL_REG
- Make the transmitter to enter the LPI state via EMACLPI_CSR_REG
- Detect the LPI_INT interrupt, and wait for TLPIEN to be set, which indicates that the transmitter has entered the LPI state
- Bring the transmitter out of the LPI state by clearing LPIEN
- Detect the LPI_INT interrupt, and wait for TLPIEX to be set, which indicates that the transmitter has exited the LPI state

37.5.6 RX Entering and Exiting the LPI State

- Detect the LPI_INT interrupt, and wait for RLPIEN to be set, which indicates that the receiver has entered the LPI state
- Detect the LPI_INT interrupt, and wait for RLPIEX to be set, which indicates that the receiver has exited the LPI state

37.6 Interrupts

ESP32-P4's EMAC can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- ETH_MAC_INTR
- PMT_INTR
- LPT_INTR

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

The following interrupt sources can generate the ETH_MAC_INTR interrupt signal:

- ABN_INT_SUMM: Triggered when DMAIN_AISE is 1 and the following interrupts are triggered.
  - TRANS_PROC_STOP: Triggered when DMAIN_TSE is 1 and the transmit process stops.
  - TRANS_JABBER_TO: Triggered when DMAIN_TJTE is 1 and the frame size exceeds 2048 bytes (10240 bytes if the jumbo frame is enabled).
  - RECV_OVFLOW: Triggered whenDMAIN_OIE is 1 and the receive buffer overflows during frame reception.
  - TRANS_UNDFLOW: Triggered whenDMAIN_UIE is 1 and the transmit buffer underflows during frame transmission.
  - RECV_BUF_UNAVAIL: Triggered when DMAIN_RBUE is 1, the Host owns the Next Descriptor in the Receive List and the DMA cannot acquire it.
  - RECV_PROC_STOP: Triggered when DMAIN_RSE is 1 and the receive process stops.

  - RECV_WDT_TO: Triggered when DMAIN_RWTE is 1 and the Receive Watchdog Timer has expired while receiving the current frame.
  - EARLY_TRANS_INT: Triggered when DMAIN_ETIE is 1 and the frame has been completely transmitted to MTL TX FIFO.
  - FATAL_BUS_ERR_INT: Triggered when DMAIN_FBEE is 1 and a bus error described in ERROR_BITS occurs.
- NORM_INT_SUMM: Triggered when DMAIN_NISE is 1 and the following interrupts are triggered.
  - TRANS_INT: Triggered when DMAIN_TIE is 1 and the transmission completes.
  - TRANS_BUF_UNAVAIL: Triggered when DMAIN_TBUE is 1, the Host owns the Next Descriptor in the Transmit List and the DMA cannot acquire it.
  - RECV_INT: Triggered when DMAIN_RIE is 1 and the frame has been received.
  - EARLY_RECV_INT: Triggered when DMAIN_ERIE is 1 and the DMA has received the first buffer of the data packet.

  EMAC_PMT_INT: Triggered when PMTINTMASK is 0, and a magic packet or a remote wakeup frame has been received in power-down mode.

  TS_TRI_INT: Triggered when TSINTMASK is 0 and any of the following conditions is true:
  - The system time value equals or exceeds the value specified in the Target Time registers
  - There is an overflow in the System Time Seconds Register

  EMAC_LPI_INT: Triggered when LPIINTMASK is 0 and any of the following conditions is true:
  - The MAC Transmitter enters the LPI state
  - The MAC Transmitter exits the LPI state
  - The MAC Receiver enters the LPI state
  - The MAC Receiver exits the LPI state

The following interrupt sources can generate the PMT_INTR interrupt signal:

- RWKPRCVD: Triggered when a remove wakeup frame is received.
- MGKPRCVD: Triggered when a magic packet is received.

The following interrupt source can generate the LPI_INTR interrupt signal:

- RLPIEX: Triggered when the MAC receiver has stopped receiving the LPI pattern on PHY and has exited the LPI state.

37.7 Register Summary

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

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

Name	Description	Address	Access
DMA configuration and control registers			
DMABUSMODE_REG	Bus mode register	0x1000	R/WS/SC
DMATXPOLLDEMAND_REG	Transmit poll demand register	0x1004	RO/WT
DMARXPOLLDEMAND_REG	Receive poll demand register	0x1008	RO/WT
DMARXBASEADDR_REG	Base address of the first receive descriptor	0x100C	R/W
DMATXBASEADDR_REG	Base address of the first transmit descriptor	0x1010	R/W
DMASTATUS_REG	Base address of interrupt, error and other events	0x1014	R/SS/WC
DMAOPERATION_MODE_REG	Operation mode and command register	0x1018	R/SS/WC
DMAIN_EN_REG	Interrupt disable and enable register	0x101C	R/W
DMAMISSEDFR_REG	Missed frame and buffer overflow counter register	0x1020	R/W
DMARINTWDTIMER_REG	Receive watchdog timer counter register	0x1024	R/W
DMAAHBSTATUS_REG	AHB master interface status register	0x102C	RO
DMATXCURRDESC_REG	Current transmit descriptor register	0x1048	RO
DMARXCURRDESC_REG	Current receive descriptor register	0x104C	RO
DMATXCURRADDR_BUF_REG	Current transmit buffer address register	0x1050	RO
DMARXCURRADDR_BUF_REG	Current receive buffer address register	0x1054	RO
MAC configuration and control registers			
EMACCONFIG_REG	MAC configuration register	0x0000	R/W
EMACFF_REG	Frame filter register	0x0004	R/W
EMACMIIADDR_REG	PHY access permission configuration register	0x0010	R/WS/SC
EMACMIIDATA_REG	PHY write and read data register	0x0014	R/W
EMACFC_REG	Frame flow control register	0x0018	varies
EMACVLANTAG_REG	802.1Q VLAN tag register	0x001C	R/W
EMACDEBUG_REG	Status and debug register	0x0024	RO
PMT_RWUFFR_REG	Wake-up frame filter register	0x0028	RO

Name	Description	Address	Access
PMT_CSR_REG	Power management control and status register	0x002C	RO
EMACLPI_CSR_REG	LPI control and status register	0x0030	RO
EMACLPITIMERSCONTROL_REG	LPI timer control register	0x0034	RO
EMACINTS_REG	Interrupt status register	0x0038	RO
EMACINTMASK_REG	Interrupt mask register	0x003C	R/W
EMACADDR0HIGH_REG	The higher 16 bits of the first 6-byte MAC address	0x0040	R/W
EMACADDR0LOW_REG	The lower 32 bits of the first 6-byte MAC address	0x0044	R/W
EMACADDR1HIGH_REG	The higher 16 bits of the second 6-byte MAC address	0x0048	R/W
EMACADDR1LOW_REG	The lower 32 bits of the second 6-byte MAC address	0x004C	R/W
EMACADDR2HIGH_REG	The higher 16 bits of the third 6-byte MAC address	0x0050	R/W
EMACADDR2LOW_REG	The lower 32 bits of the third 6-byte MAC address	0x0054	R/W
EMACADDR3HIGH_REG	The higher 16 bits of the fourth 6-byte MAC address	0x0058	R/W
EMACADDR3LOW_REG	The lower 32 bits of the fourth 6-byte MAC address	0x005C	R/W
EMACADDR4HIGH_REG	The higher 16 bits of the fifth 6-byte MAC address	0x0060	R/W
EMACADDR4LOW_REG	The lower 32 bits of the fifth 6-byte MAC address	0x0064	R/W
EMACADDR5HIGH_REG	The higher 16 bits of the sixth 6-byte MAC address	0x0068	R/W
EMACADDR5LOW_REG	The lower 32 bits of the sixth 6-byte MAC address	0x006C	R/W
EMACADDR6HIGH_REG	The higher 16 bits of the seventh 6-byte MAC address	0x0070	R/W
EMACADDR6LOW_REG	The lower 32 bits of the seventh 6-byte MAC address	0x0074	R/W
EMACADDR7HIGH_REG	The higher 16 bits of the eighth 6-byte MAC address	0x0078	R/W
EMACADDR7LOW_REG	The lower 32 bits of the eighth 6-byte MAC address	0x007C	R/W
EMACADDR8HIGH_REG	The higher 16 bits of the nineth 6-byte MAC address	0x0080	R/W
EMACADDR8LOW_REG	The lower 32 bits of the nineth 6-byte MAC address	0x0084	R/W
EMACCSTATUS_REG	Interconnection status	0x00D8	RO

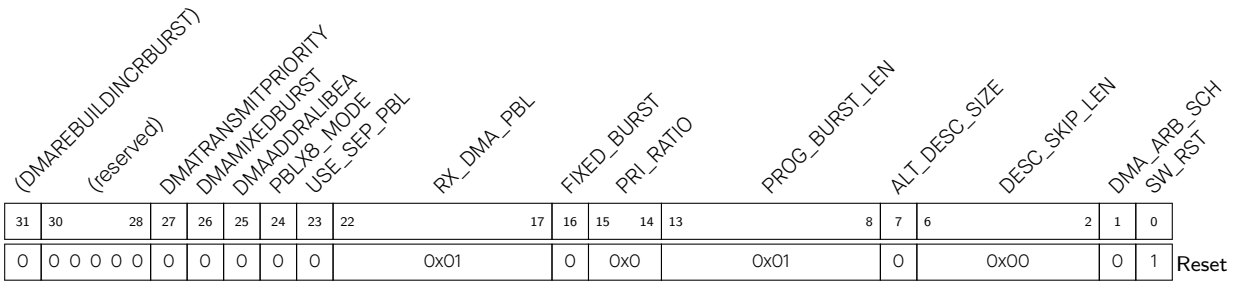
Name	Description	Address	Access
EMACWDOGTO_REG	Watchdog timer timeout control register	0x00DC	R/W
EMACTXVLTCTRL_REG	TX VLAN tag control register	0x0584	R/W
EMACTSTPCTRL_REG	Timestamp control register	0x0700	R/W
EMACSUB2NDINCR_REG	Sub-second increment register	0x0704	R/W
EMACSYSTIM2ND_REG	System time second register	0x0708	RO
EMACSYSTIMNS_REG	System time nanosecond register	0x070C	RO
EMAC2NDUPT_REG	System time second update register	0x0710	R/W
EMACNSUPT_REG	System time nanosecond update register	0x0714	R/W
EMACTSTPADDEND_REG	Fine update mode addend	0x0718	R/W
EMACTGTIME2ND_REG	Target time second register	0x071C	R/W
EMACTGTIMENS_REG	Target time nanosecond register	0x0720	R/W
EMACSYSTIMHIGHWORD2ND_REG	System time nanosecond register (bit 32 to bit 47)	0x0724	R/W/SU
EMACTSTPSTATUS_REG	Timestamp status register	0x0728	R/SS/RC

37.8 Registers

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

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

Register 37.1. DMABUSMODE_REG (0x1000)



DMAREBUILDINCRBURST Configures whether to rebuild INCRx burst.

0: Not rebuild

1: If the AHB master gets an EBT (retry, split, or losing bus grant), the AHB master interface rebuilds the pending beats of any burst transfer initiated with INCRx using INCRx and SINGLE.

(R/W)

DMATRANSMITPRIORITY Configures whether the transmit DMA has higher priority than the receive DMA during arbitration.

0: Lower priority

1: Higher priority

(R/W)

DMAMIXEDBURST Configures whether to use mixed burst.

0: Not use

1: If the FIXED_BURST bit is 1, the AHB Master interface starts all bursts of length more than 16 with INCR (undefined burst) whereas it reverts to fixed burst transfers (INCRx and SINGLE) for burst length of 16 and less.

(R/W)

DMAADDRALIBEA Configures whether bursts are aligned to address.

0: Not aligned to address

1: If the FIXED_BURST bit is 1, the AHB interface generates all bursts aligned to the start address LS bits. If the FIXED_BURST bit is 0, the first burst (accessing the data buffer's start address) is not aligned, but subsequent bursts are aligned to the address.

(R/W)

Continued on the next page...

Register 37.1. DMABUSMODE_REG (0x1000)

Continued from the previous page...

PBLX8_MODE Configures whether to multiply the PBL (programmable burst length) value by 8.
0: Disable
1: The programmed PBL value (PROG_BURST_LEN and RX_DMA_PBL) is multiplied by 8. Therefore, the DMA transfers the data in 8, 16, 32, 64, 128, and 256 beats depending on the PBL value.
(R/W)

USE_SEP_PBL Configures whether to use separate PBL.
0: PROG_BURST_LEN is applicable only for TX DMA, and RX DMA uses RX_DMA_PBL.
1: PROG_BURST_LEN is applicable for both TX and RX DMA.
(R/W)

RX_DMA_PBL Configure the maximum number of beats to be transferred in one RX DMA transaction, namely the maximum value that is used in a single block Read or Write.
Value range: 1, 2, 4, 8, 16 and 32. Any other value results in undefined behavior.
Valid only when USE_SEP_PBL is set.
(R/W)

FIXED_BURST Configures whether the AHB Master interface performs fixed burst transfers.
0: The AHB interface uses SINGLE and INCR burst transfer operations.
1: The AHB interface uses only SINGLE, INCR4, INCR8, or INCR16 during start of the normal burst transfers.
(R/W)

PRI_RATIO Configures the priority ratio in the weighted round-robin arbitration between the RX DMA and TX DMA.
0: 1:1
1: 2:0
2: 3:1
3: 4:1
Valid only when DMA_ARB_SCH is 0. (R/W)

PROG_BURST_LEN Configures the maximum number of beats to be transferred in one DMA transaction.
If the number of beats to be transferred is more than 32, then perform the following steps:
1. Set the PBLx8 mode with PBLX8_MODE
2. Set the PBL with PROG_BURST_LEN
(R/W)

ALT_DESC_SIZE Configures whether to increase the descriptor size to 32 bytes.
0: Keep 16 bytes
1: Increase to 32 bytes
(R/W)

DESC_SKIP_LEN Configures the number of Word, Dword, or Lword (depending on the 32-bit, 64-bit, or 128-bit bus) to skip between two unchained descriptors.
The address skipping starts from the end of the current descriptor to the start of next descriptor.
When this field is 0, the descriptor table is taken as contiguous by the DMA in Ring mode. (R/W)

Continued on the next page...

Register 37.1. DMABUSMODE_REG (0x1000)

Continued from the previous page...

DMA_ARB_SCH Configures the arbitration scheme between the transmit and receive paths.

0: Weighted round-robin with RX:TX or TX:RX. In this case, the priority between the paths is according to the priority specified in PRI_RATIO.

1: Fixed priority (The transmit path has priority over receive path).

(R/W)

SW_RST Configures whether the MAC DMA Controller resets the logic and all internal registers of the MAC.

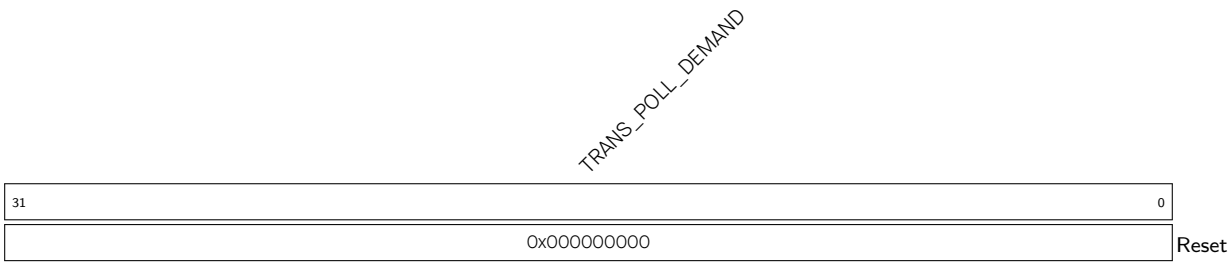
0: Release from reset

1: Reset

This bit is cleared automatically after the reset operation has completed in all of the ETH_MAC clock domains.

(R/WS/SC)

Register 37.2. DMATXPOLLDEMAND_REG (0x1004)

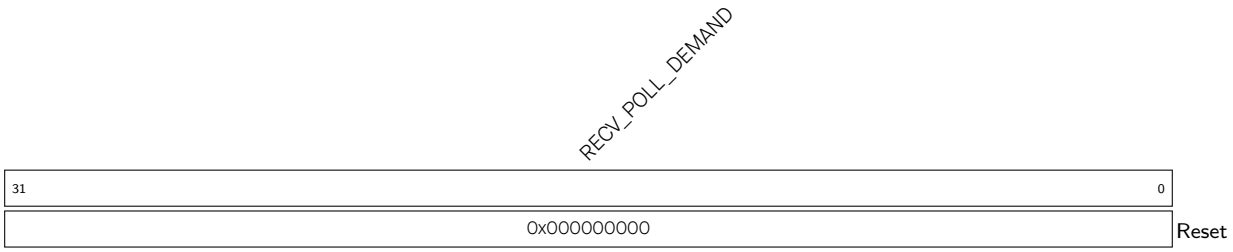


TRANS_POLL_DEMAND Configures whether to enable the TX DMA to check if the DMA owns the current descriptor.

Any value: Enable

When this field is written, the DMA reads the current descriptor pointed to by DMATXCURRDESC_REG. If that descriptor is not available (owned by the Host), the transmission returns to the Suspend state and the Bit 2 (TU) of Register 5 (Status Register) is asserted. If the descriptor is available, the transmission resumes. (RO/WT)

Register 37.3. DMARXPOLLDEMAND_REG (0x1008)

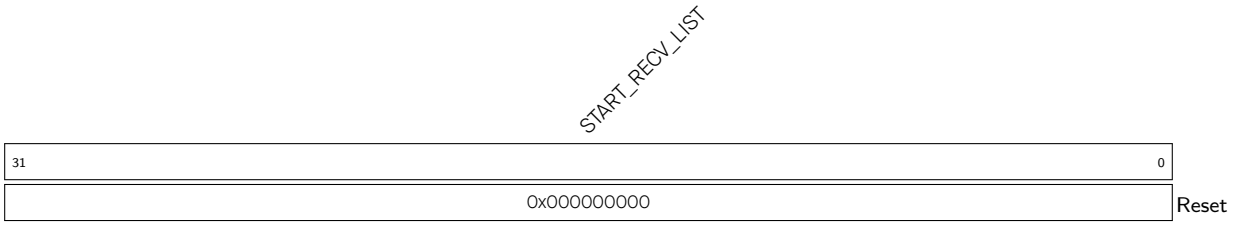


RECV_POLL_DEMAND Configures whether to enable the RX DMA to check if the DMA owns the current descriptor.

Any value: Enable

When this field is written, the DMA reads the current descriptor pointed to by DMARXCURRDESC_REG. If that descriptor is not available (owned by the Host), the reception returns to the Suspended state and the Bit 7 (RU) of Register 5 (Status Register) is not asserted. If the descriptor is available, the RX DMA returns to the active state.(RO/WT)

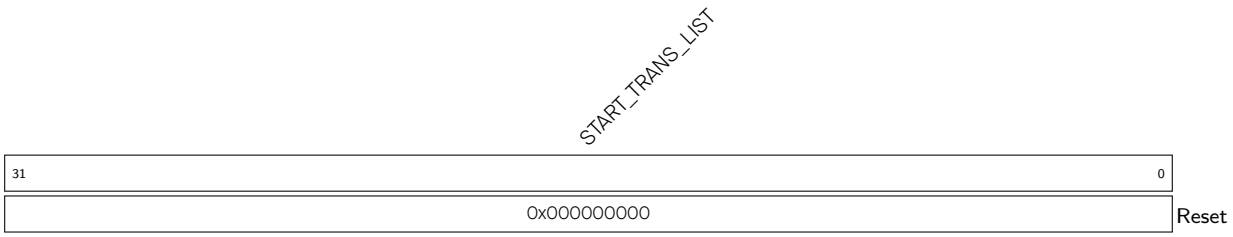
Register 37.4. DMARXBASEADDR_REG (0x100C)



START_RECV_LIST Configures the base address of the first descriptor in the receive descriptor linked list.

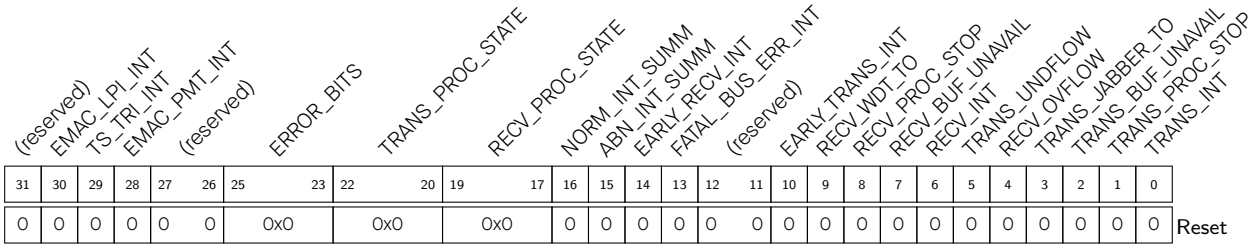
The LSB bits (1:0) are ignored (32-bit wide bus) and internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO). (R/W)

Register 37.5. DMATXBASEADDR_REG (0x1010)



START_TRANS_LIST Configures the base address of the first descriptor in the Transmit Descriptor list. The LSB bits (1:0) are ignored (32-bit wide bus) and are internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO). (R/W)

Register 37.6. DMASTATUS_REG (0x1014)



EMAC_LPI_INT The raw interrupt status of EMAC_LPI_INT. (RO)

TS_TRI_INT The raw interrupt status of TS_TRI_INT. (RO)

EMAC_PMT_INT The raw interrupt status of EMAC_PMT_INT. (RO)

ERROR_BITS Represents the type of error that caused a Bus Error, for example, error response on the AHB interface.

0: Error during RX DMA Write Data Transfer

1: Error during TX DMA Read Data Transfer

2: Error during RX DMA Descriptor Write Access

3: Error during TX DMA Descriptor Write Access

4: Error during RX DMA Descriptor Read Access

5: Error during TX DMA Descriptor Read Access

Valid only when Bit 13 (FBI) is 1. This field does not generate an interrupt. (RO)

TRANS_PROC_STATE Represents the Transmit DMA FSM state.

0: Stopped; Reset or Stop Transmit Command issued

1: Running; Fetching Transmit Transfer Descriptor

2: Running; Waiting for status

3: Running; Reading Data from Host memory buffer and queuing it to transmit buffer (TX FIFO)

4: TIME_STAMP write state

5: Reserved for future use

6: Suspended; Transmit Descriptor Unavailable or Transmit Buffer Underflow

7: Running; Closing Transmit Descriptor

This field does not generate an interrupt. (RO)

Continued on the next page...

Register 37.6. DMASTATUS_REG (0x1014)

Continued from the previous page...

NORM_INT_SUMM Represents the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register.
Bit[0]: Transmit Interrupt
Bit[2]: Transmit Buffer Unavailable
Bit[6]: Receive Interrupt
Bit[14]: Early Receive Interrupt
Only unmasked interrupts affect this bit.
This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes this set to be set, is cleared. (R/SS/WC)

ABN_INT_SUMM Represents the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register.
Bit[1]: Transmit Process Stopped
Bit[3]: Transmit Jabber Timeout
Bit[4]: Receive FIFO Overflow
Bit[5]: Transmit Underflow
Bit[7]: Receive Buffer Unavailable
Bit[8]: Receive Process Stopped
Bit[9]: Receive Watchdog Timeout
Bit[10]: Early Transmit Interrupt
Bit[13]: Fatal Bus Error
Only unmasked interrupts affect this bit.
This is a sticky bit and must be cleared each time a corresponding bit, which causes this bit to be set, is cleared. (R/SS/WC)

EARLY_RECV_INT The raw interrupt status of EARLY_RECV_INT.(R/SS/WC)

FATAL_BUS_ERR_INT The raw interrupt status of FATAL_BUS_ERR_INT. (R/SS/WC)

EARLY_TRANS_INT The raw interrupt status of EARLY_TRANS_INT. (R/SS/WC)

RECV_WDT_TO Represents whether the Receive Watchdog Timer expired while receiving the current frame and the current frame is truncated after the watchdog timeout.
0: Not expired
1: Expired
(R/SS/WC)

RECV_PROC_STOP Represents whether the Receive Process enters the Stopped state.
0: Not stopped
1: Stopped
(R/SS/WC)

Continued on the next page...

Register 37.6. DMASTATUS_REG (0x1014)

Continued from the previous page...

RECV_BUF_UNAVAIL Represents whether the Receive Buffer is available.

0: Available

1: Unavailable

This bit is set only when the previous Receive Descriptor is owned by the DMA.

When set, the Host owns the Next Descriptor in the Receive List and the DMA cannot acquire it. The Receive Process is suspended.

To resume processing Receive descriptors, the Host should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, the Receive Process resumes when the next recognized incoming frame is received. (R/SS/WC)

RECV_INT The raw interrupt status of RECV_INT. (R/SS/WC)

TRANS_UNDFLOW Represents whether the Transmit Buffer had an Underflow during frame transmission.

0: No underflow

1: Underflow

When this bit is 1, transmission is suspended and an Underflow Error TDES0[1] is set. (R/SS/WC)

RECV_OVFLOW Represents whether the Receive Buffer had an Overflow during frame reception.

0: No overflow

1: Overflow

If the partial frame is transferred to the application, the overflow status is set in RDES0[11]. (R/SS/WC)

TRANS_JABBER_TO Represents whether the Transmit Jabber Timer expired, which happens when the frame size exceeds 2,048 (10,240 bytes when the Jumbo frame is enabled).

0: Not expired

1: Expired

When the Jabber Timeout occurs, the transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert. (R/SS/WC)

TRANS_BUF_UNAVAIL Represents whether the Transmit Buffer is available.

0: Available

1: Unavailable

When this bit is 1, the Host owns the Next Descriptor in the Transmit List and the DMA cannot acquire it. Transmission is suspended. Bits[22:20] explain the Transmit Process state transitions.

To resume processing Transmit descriptors, the Host should change the ownership of the descriptor by setting TDES0[31] and then issue a Transmit Poll Demand command. (R/SS/WC)

TRANS_PROC_STOP Represents whether the transmission is stopped.

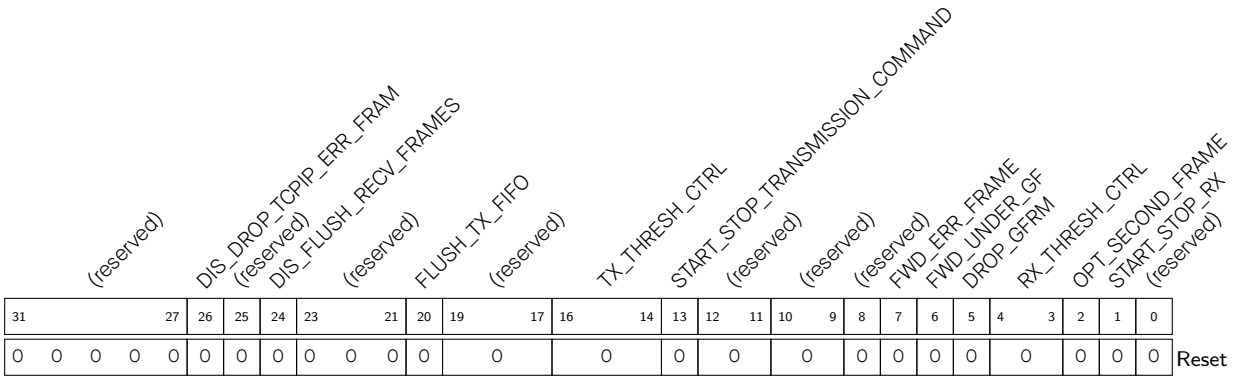
0: Not stopped

1: Stopped

(R/SS/WC)

TRANS_INT The raw interrupt status of TRANS_INT. (R/SS/WC)

Register 37.7. DMAOPERATION_MODE_REG (0x1018)



DIS_DROP_TCPIP_ERR_FRAM Configures whether to disable dropping of TCP/IP checksum error frames.

0: Enable

1: Disable

When FWD_ERR_FRAME is 0, this bit is ignored and all error frames are dropped. (R/W)

DIS_FLUSH_RECV_FRAMES Configures whether to disable flushing of received frames because of the unavailability of receive descriptors or buffers.

0: Enable

1: Disable

(R/W)

FLUSH_TX_FIFO Configures whether to flush TX FIFO to default values.

0: Not flush

1: Flush

(R/WS/SC)

TX_THRESH_CTRL Configures the threshold of TX FIFO. Transmission starts when the frame size within the TX FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted.

0: 64

1: 128

2: 192

3: 256

4: 40

5: 32

6: 24

7: 16

Transmission starts when the frame size within the TX FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted.

(R/W)

Continued on the next page...

Register 37.7. DMAOPERATION_MODE_REG (0x1018)

Continued from the previous page...

START_STOP_TRANSMISSION_COMMAND Configures whether to start or stop the transmission.

0: Stop

1: Start

(R/W)

FWD_ERR_FRAME Configures whether RX FIFO forward frames with error status (CRC error, collision error, giant frame, watchdog timeout, or overflow)

0: Drop

1: Forward

(R/W)

FWD_UNDER_GF Configures whether the RX FIFO forward undersized (frames with no Error and length less than 64 bytes) good frames including PAD and CRC.

0: Drop

1: Forward

(R/W)

DROP_GFRM Configures whether to the ETH_MAC drops the received giant frames in the RX FIFO.

0: Not drop

1: Drop

(R/W)

RX_THRESH_CTRL Configures the threshold of the RX FIFO.

0: 64

1: 32

2: 96

3: 128

Transfer (request) to DMA starts when the frame size within the RX FIFO is larger than the threshold. (R/W)

OPT_SECOND_FRAME Configures whether the DMA processes the second frame of the Transmit data even before the status for the first frame is obtained.

0: Not process

1: Process

(R/W)

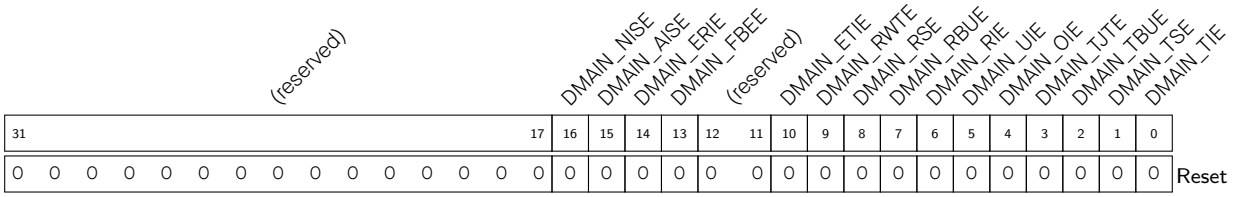
START_STOP_RX Configures whether to start or stop RX DMA reception.

0: Stop reception after the transfer of the current frame

1: Start reception

(R/W)

Register 37.8. DMAIN_EN_REG (0x101C)



DMAIN_NISE Write 1 to enable and write 0 to disable the summary of the following normal interrupt enable bits:
Bit[0]: Transmit Interrupt
Bit[2]: Transmit Buffer Unavailable
Bit[6]: Receive Interrupt
Bit[14]: Early Receive Interrupt
(R/W)

DMAIN_AISE Write 1 to enable and write 0 to disable the summary of the following abnormal interrupt enable bis:
Bit[1]: Transmit Process Stopped
Bit[3]: Transmit Jabber Timeout
Bit[4]: Receive FIFO Overflow
Bit[5]: Transmit Underflow
Bit[7]: Receive Buffer Unavailable
Bit[8]: Receive Process Stopped
Bit[9]: Receive Watchdog Timeout
Bit[10]: Early Transmit Interrupt
Bit[13]: Fatal Bus Error
(R/W)

DMAIN_ERIE Write 1 to enable and write 0 to disable EARLY_RECV_INT. (R/W)

DMAIN_FBEE Write 1 to enable and write 0 to disable FATAL_BUS_ERR_INT.(R/W)

DMAIN_ETIE Write 1 to enable and write 0 to disable EARLY_TRANS_INT. (R/W)

DMAIN_RWTE Write 1 to enable and write 0 to disable RECV_WDT_TO_INT. (R/W)

DMAIN_RSE Write 1 to enable and write 0 to disable RECV_PROC_STOP_INT. (R/W)

DMAIN_RBUE Write 1 to enable and write 0 to disable RECV_BUF_UNAVAIL_INT. (R/W)

DMAIN_RIE Write 1 to enable and write 0 to disable RECV_INT. (R/W)

Continued on the next page...

Register 37.8. DMAIN_EN_REG (0x101C)

Continued from the previous page...

DMAIN_UIE Write 1 to enable and write 0 to disable TRANS_UNDFLOW_INT. (R/W)

DMAIN_OIE Write 1 to enable and write 0 to disable RECV_OVFLOW_INT. (R/W)

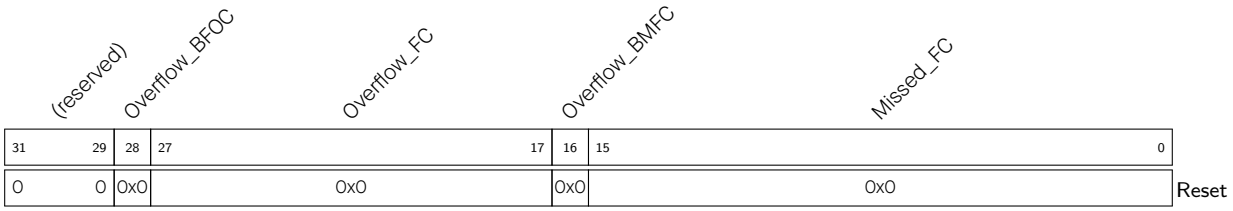
DMAIN_TJTE Write 1 to enable and write 0 to disable TRANS_JABBER_TO_INT. (R/W)

DMAIN_TBUE Write 1 to enable and write 0 to disable TRANS_BUF_UNAVAIL_INT. (R/W)

DMAIN_TSE Write 1 to enable and write 0 to disable TRANS_PROC_STOP_INT.(R/W)

DMAIN_TIE Write 1 to enable and write 0 to disable TRANS_INT.(R/W)

Register 37.9. DMAMISSEDFR_REG (0x1020)



Overflow_BFOC Represents the status of the overflow frame counter.

0: No overflow.

1: The overflow frame counter (Overflow_FC) overflows, that is, the RX FIFO overflows with the overflow frame counter at maximum value. In such a scenario, the overflow frame counter is reset to all-zeros.

(R/SS/RC)

Overflow_FC Represents the number of frames missed by the application.

This counter is incremented each time the MTL FIFO overflows. (R/SS/RC)

Overflow_BMFC Represents the status of the missed frame counter.

0: No overflow.

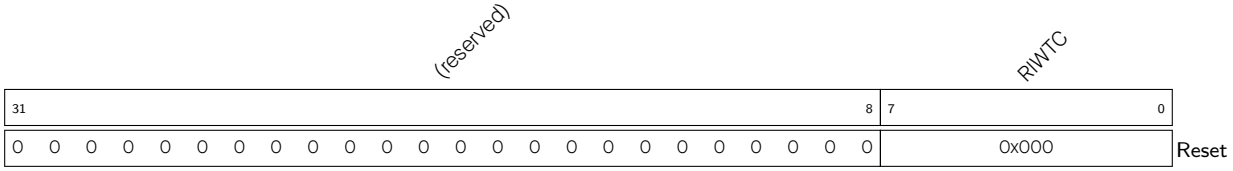
1: The missed frame counter (Missed_FC) overflows, that is, the Host Receive Buffer being unavailable with the missed frame counter at maximum value. In such a scenario, the Missed frame counter is reset to all-zeros.

(R/SS/RC)

Missed_FC Represents the number of frames missed by the controller because of the Host Receive Buffer being unavailable.

This counter is incremented each time the DMA discards an incoming frame. (R/SS/RC)

Register 37.10. DMARINTWDTIMER_REG (0x1024)

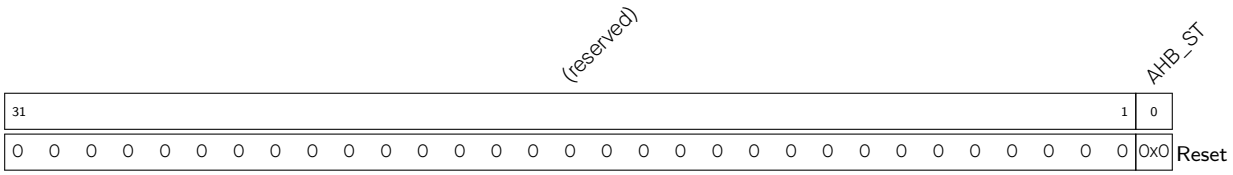


RIWTC Configures the number of system clock cycles multiplied by 256.

The watchdog timer gets triggered with the programmed value after the RX DMA completes the transfer of a frame for which the RI (RECV_INT) status bit is not set because of the setting in the corresponding descriptor RDES1[31].

When the watchdog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when the RI bit is set high because of the automatic setting of RI as per RDES1[31] of any received frame. (R/W)

Register 37.11. DMAAHBSTATUS_REG (0x102C)



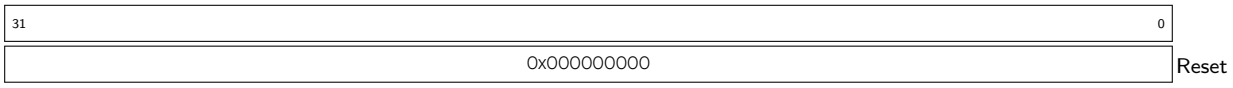
AHB_ST Configures the state of the AHB master interface.

0: Idle

1: Active

(R/W)

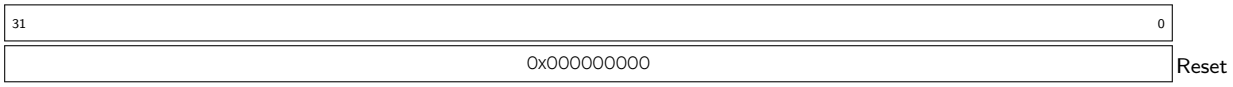
Register 37.12. DMATXCURRDESC_REG (0x1048)



TRANS_DECR_ADDR_PTR Represents the address of the current transmit descriptor list updated by the DMA during operation.

This field is cleared on reset. (RO)

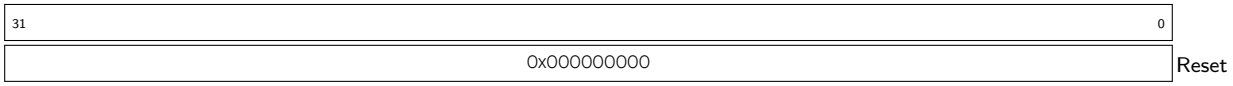
Register 37.13. DMARXCURRDESC_REG (0x104C)



RECV_DECR_ADDR_PTR Represents the address of the current receive descriptor list updated by the DMA during operation.

This field is cleared on reset. (RO)

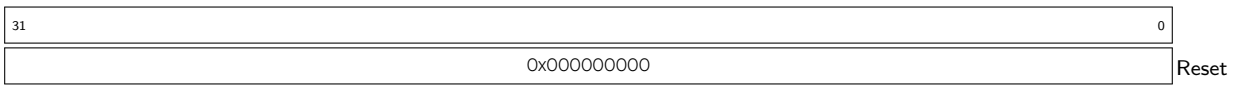
Register 37.14. DMATXCURRADDR_BUF_REG (0x1050)



TRANS_BUFF_ADDR_PTR Represents the address of the Transmit Buffer updated by the DMA during operation.

This field is cleared on reset. (RO)

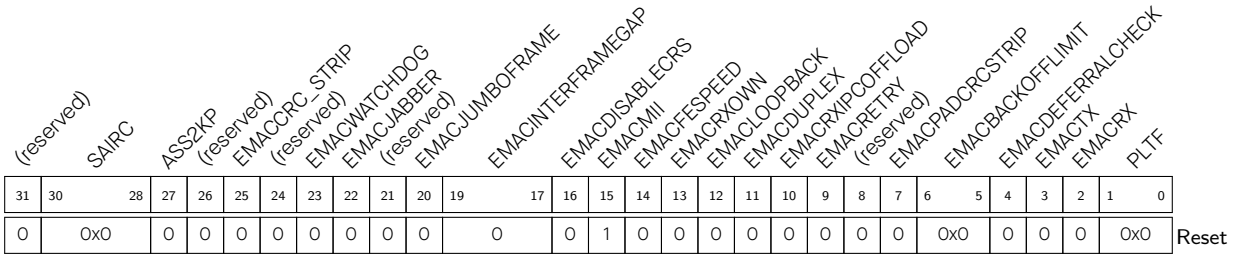
Register 37.15. DMARXCURRADDR_BUF_REG (0x1054)



RECV_BUFF_ADDR_PTR Represents the address of the Receive Buffer updated by the DMA during operation.

This field is cleared on reset. (RO)

Register 37.16. EMACCONFIG_REG (0x0000)



SAIRC Configures the source address insertion or replacement for all transmitted frames.

Bit 30 specifies which MAC Address register (0 or 1) is used for source address insertion or replacement based on the values of Bits [29:28].

Bits [29:28]:

0, 1: The input mti_sa_ctrl_i and ati_sa_ctrl_i control the SA field generation

2: Insert the content of the MAC Address register (0 or 1) in the SA field of all transmitted frames

3: Replace the the content of MAC Address register (0 or 1) in the SA field of all transmitted frames

Bit 30:

0: Use MAC Address 0 register

1: Use MAC Address 1 register

(R/W)

ASS2KP Configures whether the MAC considers received frames of more than 2000 bytes as normal packets or giant frames.

0: Giant frames. In this case, if Bit[20] (JE) is 0, the MAC considers all received frames of size more than 1,518 bytes (1,522 bytes for tagged) as Giant frames.

1: Normal packets.

(R/W)

EMACCRC_STRIP Configures whether to strip and drop the last 4 bytes (FCS) of all frames of Ether type (Length/Type field greater than or equal to 1,536) before forwarding the frame to the application.

0: Not strip and drop

1: Strip and drop

(R/W)

EMACWATCHDOG Configures whether to disable the watchdog timer.

0: Enable. In this case, the MAC does not allow a receive frame with more than 2,048 bytes (10,240 if JE is set high) or the value programmed in Watchdog Timeout Register. The MAC cuts off any bytes received after the watchdog limit number of bytes.

1: Disable. In this case, the MAC can receive frames of up to 16,384 bytes.

(R/W)

EMACJABBER Configures whether to disable the jabber timer on the transmitter.

0: Enable. In this case, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if JE is set high) during transmission.

1: Disable. In this case, the MAC can transfer frames of up to 16,384 bytes.

(R/W)

Continued on the next page...

Register 37.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACJUMBOFRAME Configures whether the MAC allows Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames) without reporting a giant frame error in the receive frame status.
0: Not allow
1: Allow
(R/W)

EMACINTERFRAMEGAP Configures the minimum interframe gap (IFG) between frames during transmission.
0: 96 bit times
1: 88 bit times
2: 80 bit times
3: 72 bit times
4: 64 bit times
5: 56 bit times
6: 48 bit times
7: 40 bit times
In the half-duplex mode, the minimum IFG can be configured only for 64 bit times (IFG = 100).
(R/W)

EMACDISABLECRS Configures whether to disable Carrier Sense during transmission.
0: The MAC transmitter generates Loss of Carrier or No Carrier errors because of Carrier Sense and can even abort the transmissions.
1: The MAC transmitter ignores the MII CRS signal during frame transmission in the half-duplex mode. This request results in no errors generated because of Loss of Carrier or No Carrier during such transmission.
(R/W)

EMACMII Configures the Ethernet line speed.
0: For 1000 Mbps operations
1: For 10 or 100 Mbps operations
In 10 or 100 Mbps operations, the exact line speed is determined by both this bit and the FES (EMACFESPEED) bit. (R/W)

EMACFESPEED Configures the speed in the MII and RMII interface.
0: 10 Mbps
1: 100 Mbps
(R/W)

EMACRXOWN Configures whether to disable receive own.
0: The MAC receives all packets that are given by the PHY while transmitting.
1: the MAC disables the reception of frames when TX_EN triggers an interrupt in the half-duplex mode.
This bit is not applicable if the MAC is operating in the full-duplex mode. (R/W)

Continued on the next page...

Register 37.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACLOOPBACK Configures whether to enable the loopback mode.

0: Disable.

1: Enable. In this case, the MII Receive clock input (CLK_RX) is required for the loopback to work properly, because the Transmit clock is not looped-back internally.

(R/W)

EMACDUPLEX Configures whether to enable full-duplex mode, so that the MAC can transmit and receive simultaneously.

0: Disable

1: Enable

This bit is RO with a default value of 1 in the full-duplex-only configuration. (R/W)

EMACRXIPCOFFLOAD Configures whether to enable the checksum offload function

0: Disable

1: Enable

When the checksum offload function is enabled, the MAC calculates the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25/26 or 29/30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The MAC also appends the 16-bit checksum calculated for the IP header datagram payload (bytes after the IPv4 header) and appends it to the Ethernet frame transferred to the application (when Type 2 COE is deselected). (R/W)

EMACRETRY Configures whether to disable retry, so that the MAC attempts only one transmission, and when a collision occurs on the MII interface the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status.

0: Enable

1: Disable

Valid only in the half-duplex mode. (R/W)

EMACPADCRCSTRIP Configures whether to enable automatic Pad or CRC stripping.

0: Disable. In this case, the MAC passes all incoming frames, without modifying them, to the Host.

1: Enable. In this case, the MAC strips the Pad or FCS field on the incoming frames only if the value of the length field is less than 1,536 bytes. All received frames with length field greater than or equal to 1,536 bytes are passed to the application without stripping the Pad or FCS field. (R/W)

Continued on the next page...

Register 37.16. EMACCONFIG_REG (0x0000)

Continued from the previous page...

EMACBACKOFFLIMIT Configures the back-off limit determining the random integer number (r) of slot time delays (4,096 bit times for 1000 Mbps and 512 bit times for 10/100 Mbps) for which the MAC waits before rescheduling a transmission attempt during retries after a collision.
00: k = min (n, 10)
01: k = min (n, 8)
10: k = min (n, 4)
11: k = min (n, 1)
where n = retransmission attempt.
The random integer r takes the value in the range 0 <= r < kth power of 2
Valid only in the half-duplex mode. (R/W)

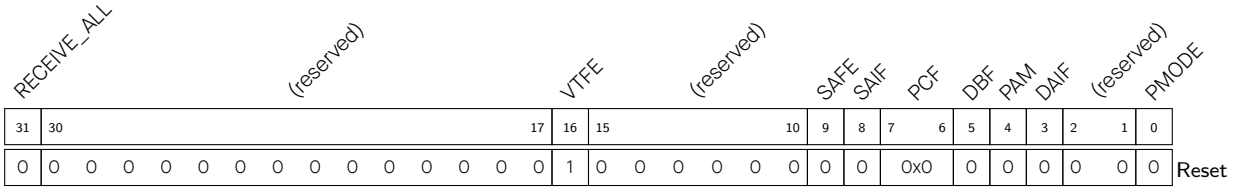
EMACDEFERRALCHECK Configures whether to enable the deferral check function.
0: Disable
1: Enable
(R/W)

EMACTX Configures whether to enable the transmit state machine of the MAC.
0: Disabled after the completion of the transmission of the current frame, and does not transmit any further frames
1: Enabled for transmission on the MII
(R/W)

EMACRX Configures whether to enable the receiver state machine of the MAC. 0: Disabled after the completion of the reception of the current frame, and does not receive any further frames from the MII
1: Enabled for receiving frames from the MII
(R/W)

PLTF Configures the number of preamble bytes that are added to the beginning of every Transmit frame.
0: 7 bytes of preamble
1: 5 byte of preamble
2: 3 bytes of preamble
3: Reserved
The preamble reduction occurs only when the MAC is operating in the full-duplex mode. (R/W)

Register 37.17. EMACFF_REG (0x0004)



RECEIVE_ALL Configures whether to pass all received modules.

0: The MAC Receiver module passes only those frames to the Application that pass the SA or DA address filter.

1: The MAC Receiver module passes all received frames, irrespective of whether they pass the address filter or not, to the Application. The result of the SA or DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word.

(R/W)

VTFE Configures whether to enable the VLAN Tag filter.

0: Disable. In this case, the MAC forwards all frames irrespective of the match status of the VLAN Tag.

1: Enable. In this case, the MAC drops VLAN-tagged frames that do not match the VLAN Tag comparison.

(R/W)

SAFE Configures whether to enable the source address filter.

0: Disable. In this case, the MAC forwards the received frame to the application with an updated SAF bit of the RX Status depending on the SA address comparison.

1: Enable. In this case, the MAC compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison fails, the MAC drops the frame.

(R/W)

SAIF Configures whether to enable SA inverse filtering.

0: Disable. In this case, frames whose SA does not match the SA registers are marked as failing the SA Address filter.

1: Enable. In this case, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA Address filter.

(R/W)

Continued on the next page...

Register 37.17. EMACFF_REG (0x0004)

Continued from the previous page...

PCF Configures the forwarding of all control frames (including unicast and multicast PAUSE frames).

0: The MAC filters all control frames.

1: The MAC forwards all control frames except PAUSE control frames to application even if they fail the Address filter.

2: The MAC forwards all control frames to application even if they fail the Address Filter.

3: The MAC forwards control frames that pass the Address Filter.

The following conditions should be true for the PAUSE control frames processing:

Condition 1: The MAC is in the full-duplex mode and flow control is enabled by setting Bit 2 (RFE) of Register 6 (Flow Control Register) to 1.

Condition 2: The destination address (DA) of the received frame matches the special multicast address or EMACADDR0 when Bit 3 (UP) of the Register 6 (Flow Control Register) is set.

Condition 3: The Type field of the received frame is 0x8808 and the OPCODE field is 0x0001.

(R/W)

DBF Configures whether to disable broadcast frames.

0: Enable. The AFM (Address Filtering Module) module passes all received broadcast frames.

1: Disable. The AFM module filters all incoming broadcast frames. In addition, it overrides all other filter settings.

(R/W)

PAM Configures whether to pass all multicast frames.

0: Filtering of multicast frame depends on HMC bit

1: All received frames with a multicast destination address (first bit in the destination address field is '1') are passed

(R/W)

DAIF Configures whether to enable DA inverse filtering for both unicast and multicast frames.

0: Disable

1: Enable

(R/W)

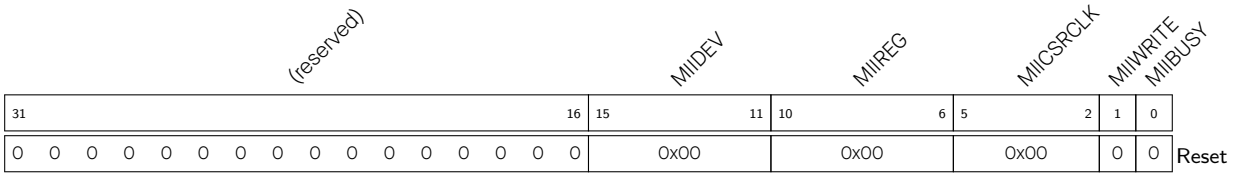
PMODE Configures whether to enable the promiscuous mode.

0: Disable.

1: Enabled. In this case, the Address Filter module passes all incoming frames regardless of its destination or source address. The SA or DA Filter Fails status bits of the Receive Status Word are always cleared when PR (PRI_RATIO) is set.

(R/W)

Register 37.18. EMACMIIADDR_REG (0x0010)



MIIDEV Configures which of the 32 possible PHY devices are being accessed. (R/W)

MIIREG Configures the desired address register in the selected PHY device. (R/W)

MIICSRCLK Configures the CSR clock frequency.

0: APB clock frequency is 80 MHz, and MDC clock frequency is APB_CLK/42

3: APB clock frequency is 40 MHz, and MDC clock frequency is APB_CLK/26

Other values are reserved

(R/W)

MIIWRITE Configures the direction of the operation that uses MII_DATA.

0: Read operation

1: Write operation

(R/W)

MIIBUSY Configures the state of the MII.

0: Idle

1: Busy

This bit is used in combination with MIIREG and MII_DATA.

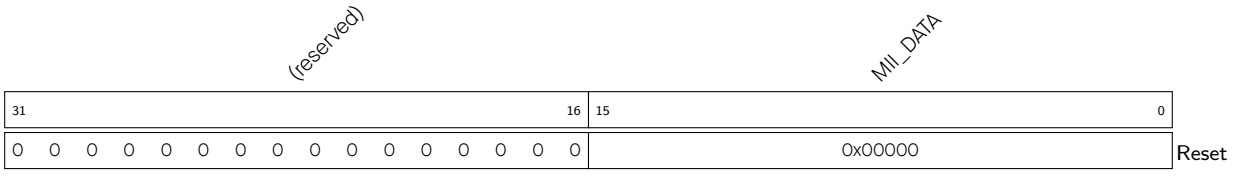
This bit should read logic 0 (default) before writing to MIIREG and MII_DATA.

To read or write to MIIREG and MII_DATA, software (the user) should set this field to 1.

MII_DATA should be kept valid (data remains unchanged) when it is accessed until this field is cleared by hardware (the MAC).

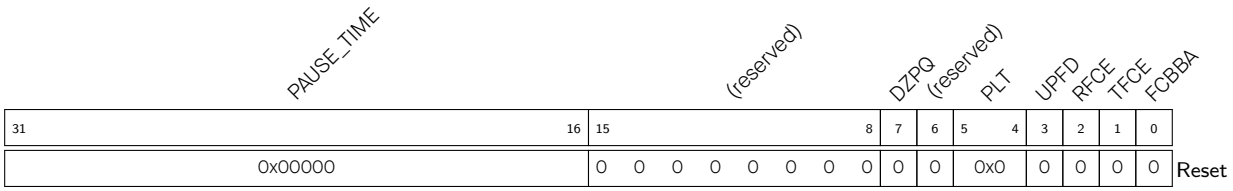
Note that ESP32-P4 MAC does not receive ACK from PHY during a read or write access to MIIREG and MII_DATA. (R/WS/SC)

Register 37.19. EMACMIIDATA_REG (0x0014)



MII_DATA Configures the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.(R/W)

Register 37.20. EMACFC_REG (0x0018)



PAUSE_TIME Configures the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the (G)MII clock domain, then consecutive writes to this register should be performed only after at least four clock cycles in the destination clock domain. (R/W)

DZPQ Configures whether to disable the automatic generation of the Zero-Quanta Pause Control frames.

0: Enable

1: Disable

(R/W)

PLT Configures the threshold of the PAUSE timer at which the PAUSE Frame is automatically retransmitted.

0: The threshold is Pause time minus 4 slot times (PT - 4 slot times).

1: The threshold is Pause time minus 28 slot times (PT - 28 slot times).

2: The threshold is Pause time minus 144 slot times (PT - 144 slot times).

3: The threshold is Pause time minus 256 slot times (PT - 256 slot times).

The threshold values should be always less than the Pause Time configured in Bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second PAUSE frame is automatically transmitted 228 (256 - 28) slot times after the first PAUSE frame is transmitted.

(R/W)

UPFD Configures whether to enable unicast pause frame detection.

0: Disable. The MAC only detects Pause frames with a unique multicast address.

1: Enable. The MAC can also detect Pause frames with unicast address of the station. This unicast address should be as specified in the EMACADDR0 High Register and EMACADDR0 Low Register.

(R/W)

RFCE Configures whether to enable the decode function of the Pause time.

0: Disable

1: Enable. The MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time

(R/W)

TFCE Configures whether to enable transmit flow control in full-duplex mode, and whether to enable the back-pressure feature in half-duplex mode.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 37.20. EMACFC_REG (0x0018)

Continued from the previous page...

FCBBA Configures whether to enable a Pause Control frame in the full-duplex mode, and whether to activate the back-pressure function in the half-duplex mode if the TFCE bit is set.

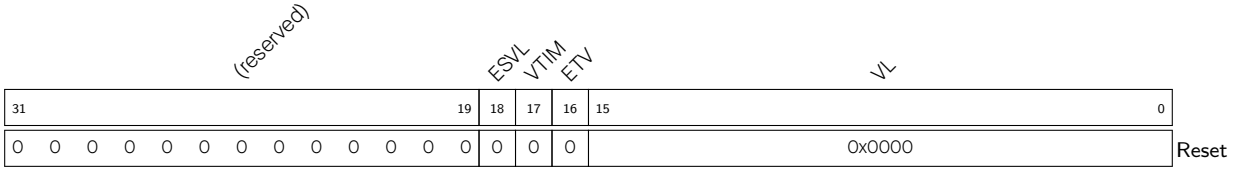
0: Disable

1: Enable

In the full-duplex mode, this bit should be read as 0 before writing to the Flow Control register. To initiate a Pause control frame, the Application must set this bit to 1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to 0. The Flow Control register should not be written to until this bit is cleared. (R/WS/SC)

In the half-duplex mode, when this bit is set (and TFCE is set), then backpressure is asserted by the MAC. During backpressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. When the MAC is configured for the full-duplex mode, the BPA is automatically disabled. (R/W)

Register 37.21. EMACVLANTAG_REG (0x001C)



ESVL Configures whether to enable S-VLAN.

0: Disable.

1: Enable. The MAC transmitter and receiver also consider the S-VLAN (Type = 0x88A8) frames as valid VLAN tagged frames.

(R/W)

VTIM Configures whether to enable VLAN Tag inverse match.

0: Disable. The frames with matched VLAN Tag are marked as matched.

1: Enable. The frames that do not have matching VLAN Tag are marked as matched.

(R/W)

ETV Configures whether to enable 12-bit VLAN Tag comparison.

0: Disable. All 16 bits of the 15th and 16th bytes of the received VLAN frame are used for comparison and VLAN hash filtering.

1: Enable. A 12-bit VLAN identifier is used for comparing and filtering instead of the complete 16-bit VLAN tag. Bits [11:0] of VLAN tag are compared with the corresponding field in the received VLAN-tagged frame.

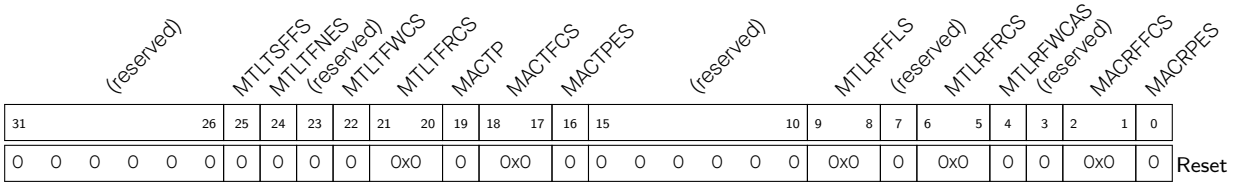
(R/W)

VL Configures the 802.1Q VLAN tag to identify the VLAN frames and is compared to the 15th and 16th bytes of the frames being received for VLAN frames.

When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison.

If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and 16th bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 or 0x88a8 as VLAN frames.(R/W)

Register 37.22. EMACDEBUG_REG (0x0024)



MTLTSFFS Represents whether MTL TX Status FIFO is full and can accept any more frames for transmission.

0: Not Full

1: Full

(RO)

MTLTFNES Represents whether MTL TX FIFO is not empty and some data is left for transmission.

0: Empty

1: Not empty

(RO)

MTLTFWCS Represents whether MTL TX FIFO Write Controller is active and transferring data to the TX FIFO.

0: Inactive

1: Active

(RO)

MTLTFRCS Represents the state of the TX FIFO Read Controller.

0: IDLE state

1: READ state (transferring data to MAC transmitter)

2: Waiting for TXStatus from MAC transmitter

3: Writing the received TXStatus or flushing the TX FIFO

(RO)

MACTP Represents whether the MAC transmitter is in the PAUSE condition (in the full-duplex only mode) and hence does not schedule any frame for transmission.

0: Not in pause

1: In pause

(RO)

MACTFCS Represents the state of the MAC Transmit Frame Controller module.

0: IDLE state

1: Waiting for Status of the previous frame or IFG or backoff period to be over

2: Generating and transmitting a PAUSE control frame (in the full-duplex mode)

3: Transferring input frame for transmission

(RO)

MACTPES Represents the status of the MAC MII transmit protocol engine.

0: IDLE state

1: Actively transmitting data

(RO)

Continued on the next page...

Register 37.22. EMACDEBUG_REG (0x0024)

Continued from the previous page...

MTLRFFLS Represents the status of the fill-level of the RX FIFO.

0: RX FIFO Empty

1: RX FIFO fill level is below the flow-control deactivate threshold

2: RX FIFO fill level is above the flow-control activate threshold

3: RX FIFO Full

(RO)

MTLRFRCS Represents the state of the RX FIFO read Controller.

0: IDLE state

1: Reading frame data

2: Reserved

3: Flushing the frame data and status

(RO)

MTLRFWCAS Represents the state of the MTL RX FIFO Write Controller.

0: Inactive

1: Active and is transferring a received frame to the FIFO.

(RO)

MACRFFCS Represents the state of the small FIFO Read (bit 1) and Write (bit 0) controllers of the MAC Receive Frame Controller Module.

0: Inactive

1: Active

(RO)

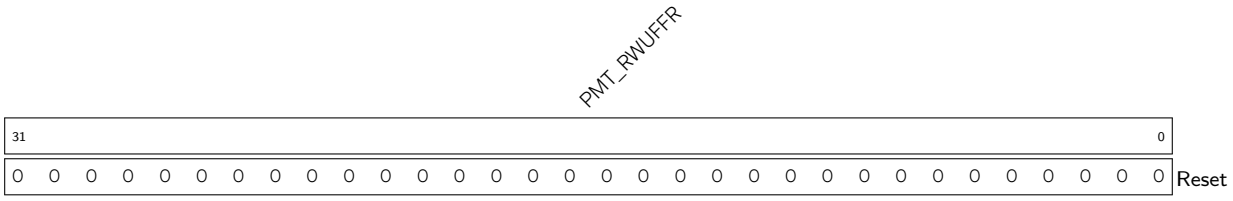
MACRPES Represents the state of the MAC GMII or MII receive protocol engine.

0: IDLE

1: Actively receiving data

(RO)

Register 37.23. PMT_RWUFFR_REG (0x0028)



WKUPPKTFILTER When RWKPTR is 0 ~ 3, configures the masking of filter 0 ~ 3 wakeup frame.

Bit[30:0] corresponds to offset+j (j = 0 ~ 30).

Bit[31] has to be 0. 0: Unmask

1: Mask

When RWKPTR is 4, bit[3]/bit[11]/bit[19]/bit[27] configure the target address type, and bit[0]/bit[8]/bit[16]/bit[24] configure whether to enable filter 0 ~ 3.

0: Unicast/Disable

1: Multicast/Enable

When RWKPTR is 5, configures the offset of the filter 0 ~ 3 masked byte.

Bit[7:0] configures the offset of filter 0 masked byte, and [31:24] configures the offset of filter 3 masked byte.

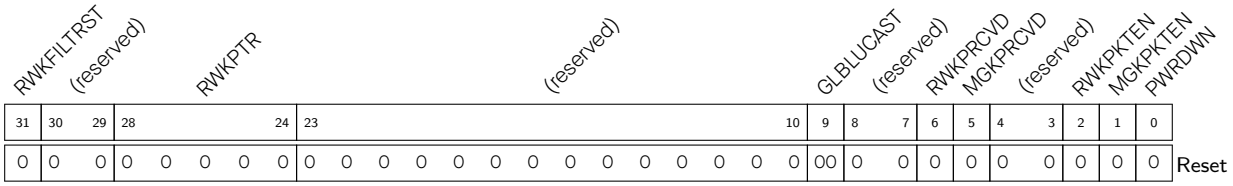
When RWKPTR is 6 ~ 7, configures the 16-bit CRC value of filter 0 ~ 3 masked byte starting from offset based on the formula:

$G(x) = x^{16} + x^{15} + x^2 + 1$

Bit[7:0] configures the CRC value of filter 0 or filter 2 masked byte when RWKPTR is 6 or 7 respectively, and bit[31:16] configures the CRC value of filter 1 or filter 3 masked byte when RWKPTR is 6 or 7 respectively.

(R/W)

Register 37.24. PMT_CSR_REG (0x002C)



RWKFILTRST Configures whether to reset the RWKPTR register.

0: Release from reset

1: Reset

(R/WS/SC)

RWKPTR Configures the purpose of the PMT_RWUFFR register. For details, see the description of PMT_RWUFFR. (RO)

GLBLUCAST Configures whether to enable the unicast frame of target address filter as the wakeup frame.

0: Disable

1: Enable

(R/W)

RWKPRCVD Represents whether a remote wakeup frame event has been received.

0: Not received

1: Received

(R/SS/RC)

MGKPRCVD Represents whether a magic frame event has been received.

0: Not received

1: Received

(R/SS/RC)

RWKPKTEN Configures whether to enable power management events using remote wakeup frames.

0: Disable

1: Enable

(R/W)

MGKPKTEN Configures whether to enable power management events using magic frames.

0: Disable

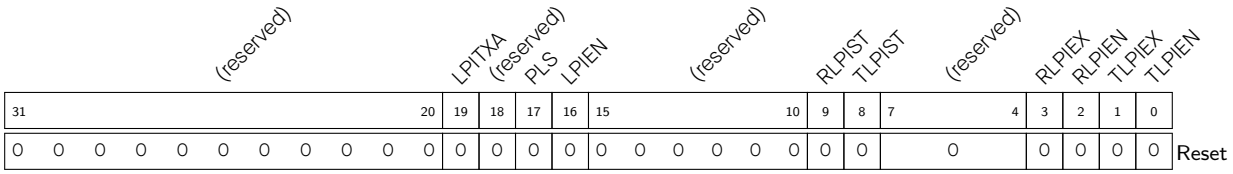
1: Enable

(R/W)

PWRDWN Configures whether the EMAC receiver drops all received frames until it receives the expected magic frame or remote wakeup frame.

Valid only when MGKPKTEN, GLBLUCAST or RWKPKTEN is 1. (R/WS/SC)

Register 37.25. EMACLPI_CSR_REG (0x0030)



LPITXA Configures the behavior of the MAC when it is entering or coming out of the LPI mode on the transmit side.

0: This bit directly controls behavior of the MAC when it is entering or coming out of the LPI mode.

1: This bit controls behavior of the MAC when it is entering or coming out of the LPI mode in combination with LPIEN. When LPIEN is also 1, the MAC enters the LPI mode only after all outstanding frames (in the core) and pending frames (in the application interface) have been transmitted. The MAC comes out of the LPI mode when the application sends any frame for transmission

(R/W)

PLS Configures the link status of the PHY.

0: Link up

1: Link down

(R/W)

LPIEN Configures whether the EMAC transmitter enters the LPI state.

0: Exit

1: Enter

This bit is cleared when the LPITXA bit is set and the MAC exits the LPI state because of the arrival of a new packet for transmission.

(R/W/SC)

RLPIST Represents the receive LPI state.

0: Inactive

1: The EMAC is receiving the LPI pattern

(RO)

TLPIST Represents the transmit LPI state.

0: Inactive

1: The EMAC is transmitting the LPI pattern

(RO)

RLPIEX Represents the receive LPI exit state.

0: Inactive

1: The EMAC Receiver has stopped receiving the LPI pattern, exited the LPI state, and resumed the normal reception

(R/SS/RC)

Continued on the next page...

Register 37.25. EMACLPI_CSR_REG (0x0030)

Continued from the previous page...

RLPIEN Represents the receive LPI entry state.

0: Inactive

1: The EMAC Receiver has received an LPI pattern and entered the LPI state

(R/SS/RC)

TLPIEX Represents the transmit LPI exit state.

0: Inactive

1: The EMAC transmitter has exited the LPI state after the user has cleared the LPIEN bit and the LPI_TW_TIMER has expired

(R/SS/RC)

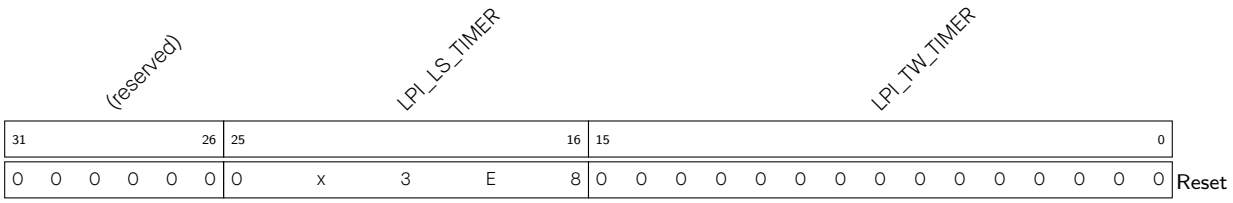
TLPIEN Represents the transmit LPI entry state.

0: Inactive

1: The EMAC Transmitter has entered the LPI state because of the setting of the LPIEN bit

(R/SS/RC)

Register 37.26. EMACLPITIMERSCONTROL_REG (0x0034)



LPI_LS_TIMER Configures the minimum time for which the link status from the PHY should be up.

Measurement unit: millisecond.

The default value is 1000 (1 second) as defined in the IEEE standard.

(R/W)

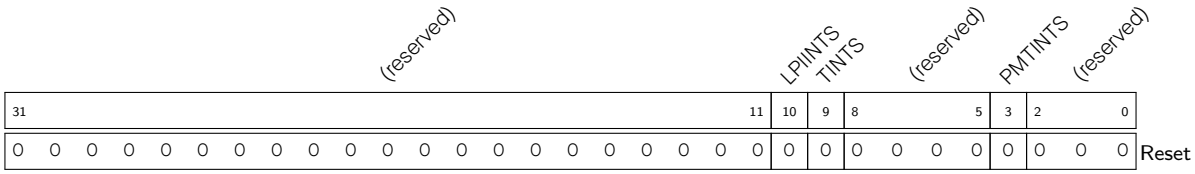
LPI_TW_TIMER Configures the minimum time for which the MAC waits after it stops transmitting the LPI pattern to the PHY and before it resumes the normal transmission.

Measurement unit: millisecond.

The TLPIEX status bit is set after the expiry of this timer.

(R/W)

Register 37.27. EMACINTS_REG (0x0038)

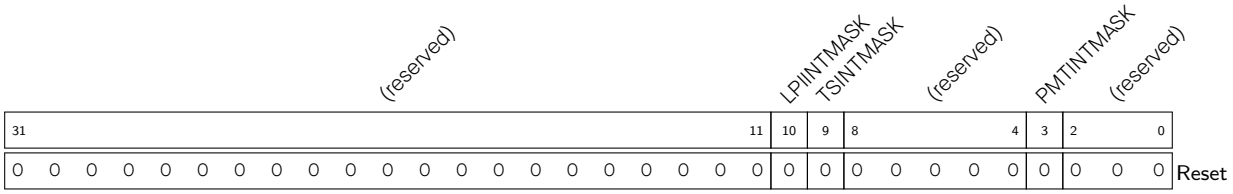


LPIINTS The raw interrupt status of EMAC_LPI_INT. (RO)

TINTS The raw interrupt status of TS_TRI_INT. (R/SS/RC)

PMTINTS The raw interrupt status of EMAC_PMP_INT. (RO)

Register 37.28. EMACINTMASK_REG (0x003C)

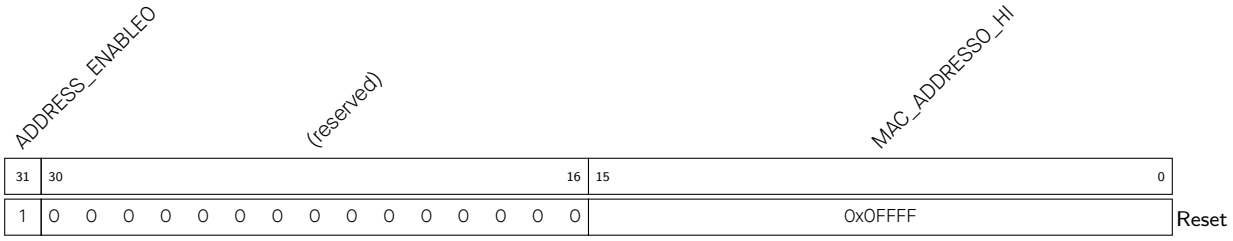


LPIINTMASK Write 1 to mask EMAC_LPI_INT.(R/W)

TSINTMASK Write 1 to mask TS_TRI_INT. (R/W)

PMTINTMASK Write 1 to mask EMAC_PMP_INT. (R/W)

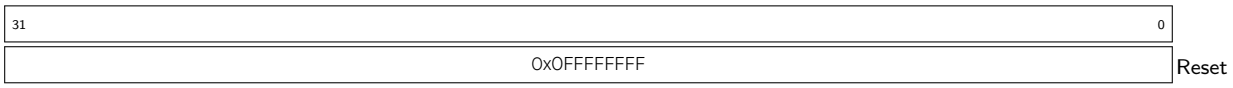
Register 37.29. EMACADDROHIGH_REG (0x0040)



ADDRESS_ENABLEO This bit is always set to 1. (RO)

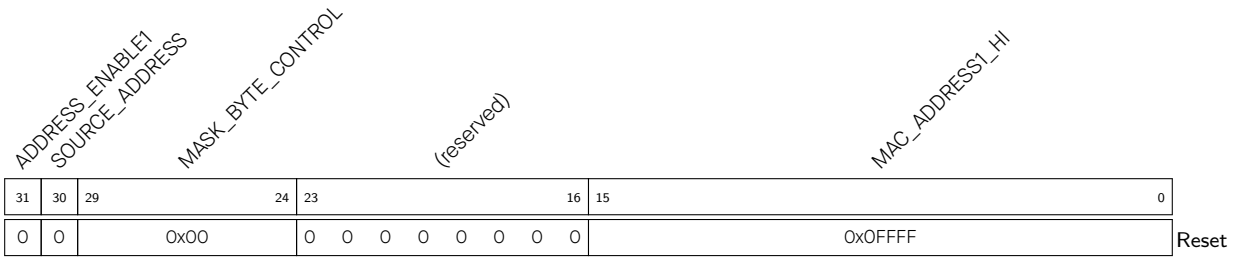
MAC_ADDRESSO_HI Configures the upper 16 bits of the first 6-byte MAC address [47:32]. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames. (R/W)

Register 37.30. EMACADDR0LOW_REG (0x0044)



EMACADDR0LOW_REG Configures the lower 32 bits of the first 6-byte MAC address. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames. (R/W)

Register 37.31. EMACADDR1HIGH_REG (0x0048)



ADDRESS_ENABLE1 Configures whether to enable perfect filtering using the second MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS Configures whether to which fields of the received frame EMACADDR1 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR1.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR1HIGH_REG[15:8]

Bit[28]: EMACADDR1HIGH_REG[7:0]

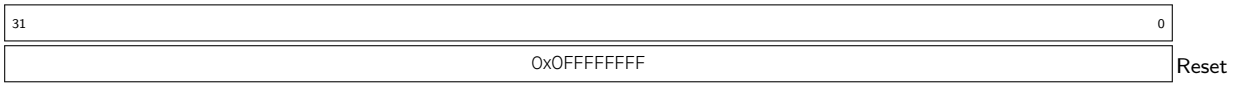
Bit[27]: EMACADDR1LOW_REG[31:24]

Bit[24]: EMACADDR1LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

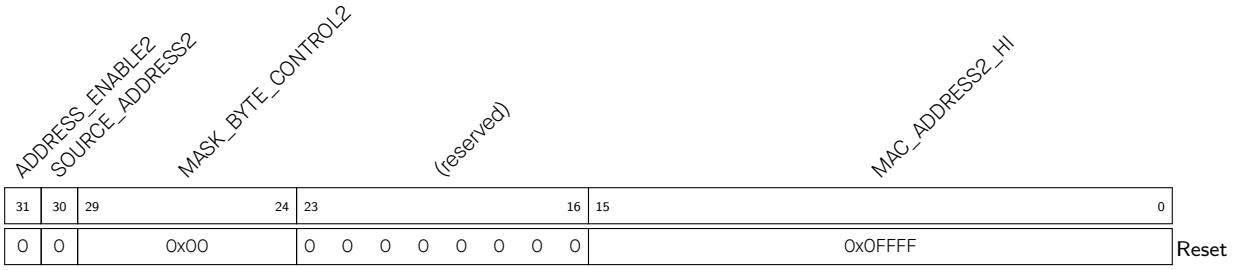
MAC_ADDRESS1_HI Configures the upper 16 bits of the second 6-byte MAC address [47:32]. (R/W)

Register 37.32. EMACADDR1LOW_REG (0x004C)



EMACADDR1LOW_REG Configures the lower 32 bits of the second 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 37.33. EMACADDR2HIGH_REG (0x0050)



ADDRESS_ENABLE2 Configures whether to enable perfect filtering using the third MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS2 Configures whether to which fields of the received frame EMACADDR2 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL2 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR2.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR2HIGH_REG[15:8]

Bit[28]: EMACADDR2HIGH_REG[7:0]

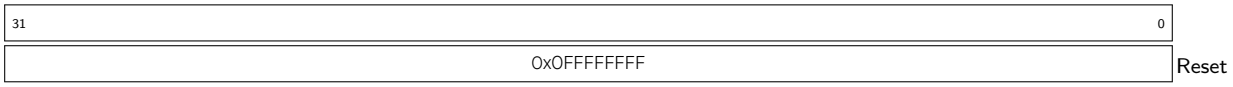
Bit[27]: EMACADDR2LOW_REG[31:24]

Bit[24]: EMACADDR2LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

MAC_ADDRESS2_HI Configures the upper 16 bits of the third 6-byte MAC address [47:32]. (R/W)

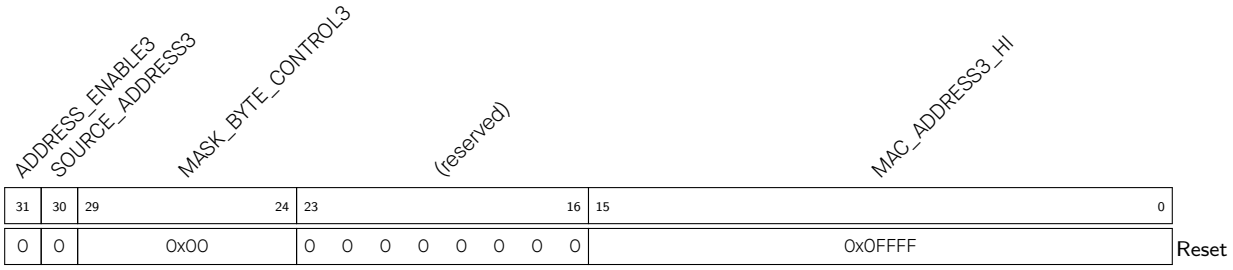
Register 37.34. EMACADDR2LOW_REG (0x0054)



EMACADDR2LOW_REG Configures the lower 32 bits of the third 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 37.35. EMACADDR3HIGH_REG (0x0058)



ADDRESS_ENABLE3 Configures whether to enable perfect filtering using the fourth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS3 Configures whether to which fields of the received frame EMACADDR3 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL3 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR3.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR3HIGH_REG[15:8]

Bit[28]: EMACADDR3HIGH_REG[7:0]

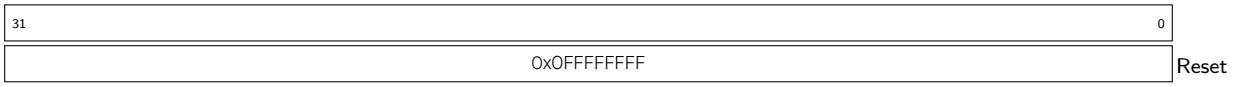
Bit[27]: EMACADDR3LOW_REG[31:24]

Bit[24]: EMACADDR3LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS3_HI Configures the upper 16 bits of the fourth 6-byte MAC address [47:32]. (R/W)

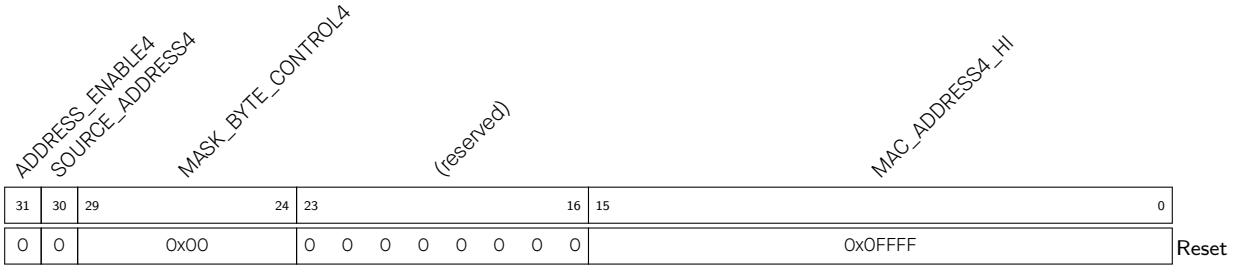
Register 37.36. EMACADDR3LOW_REG (0x005C)



EMACADDR3LOW_REG Configures the lower 32 bits of the fourth 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 37.37. EMACADDR4HIGH_REG (0x0060)



ADDRESS_ENABLE4 Configures whether to enable perfect filtering using the fifth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS4 Configures whether to which fields of the received frame EMACADDR4 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL4 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR4.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR4HIGH_REG[15:8]

Bit[28]: EMACADDR4HIGH_REG[7:0]

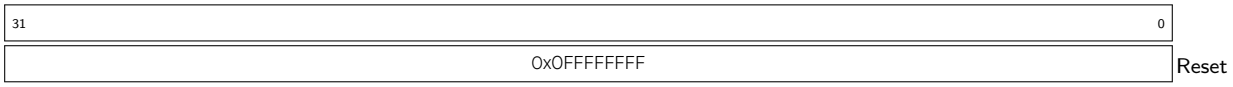
Bit[27]: EMACADDR4LOW_REG[31:24]

Bit[24]: EMACADDR4LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

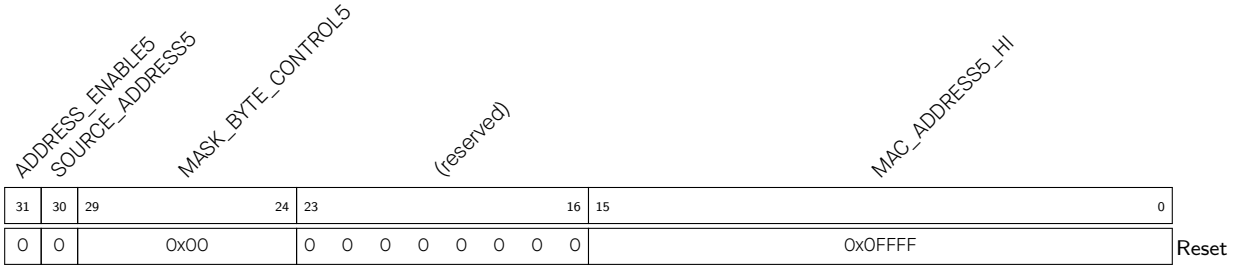
MAC_ADDRESS4_HI Configures the upper 16 bits of the fifth 6-byte MAC address [47:32]. (R/W)

Register 37.38. EMACADDR4LOW_REG (0x0064)



EMACADDR4LOW_REG Configures the lower 32 bits of the fifth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 37.39. EMACADDR5HIGH_REG (0x0068)



ADDRESS_ENABLE5 Configures whether to enable perfect filtering using the sixth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS5 Configures whether to which fields of the received frame EMACADDR5 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL5 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR5.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR5HIGH_REG[15:8]

Bit[28]: EMACADDR5HIGH_REG[7:0]

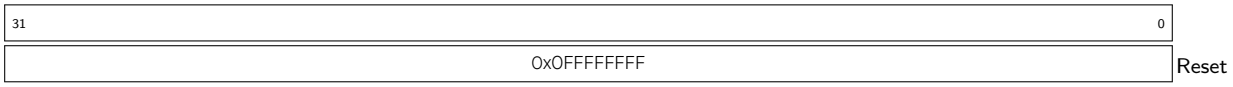
Bit[27]: EMACADDR5LOW_REG[31:24]

Bit[24]: EMACADDR5LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.(R/W)

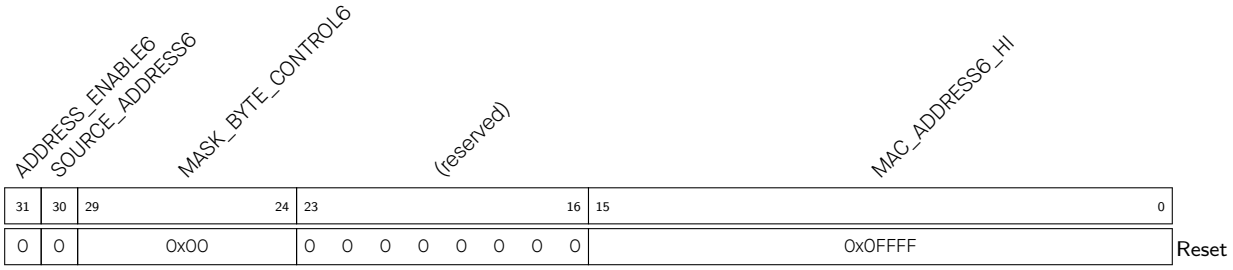
MAC_ADDRESS5_HI Configures the upper 16 bits of the sixth 6-byte MAC address [47:32]. (R/W)

Register 37.40. EMACADDR5LOW_REG (0x006C)



EMACADDR5LOW_REG Configures the lower 32 bits of the sixth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 37.41. EMACADDR6HIGH_REG (0x0070)



ADDRESS_ENABLE6 Configures whether to enable perfect filtering using the seventh MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS6 Configures whether to which fields of the received frame EMACADDR 6[47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL6 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR6.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDR6HIGH_REG[15:8]

Bit[28]: EMACADDR6HIGH_REG[7:0]

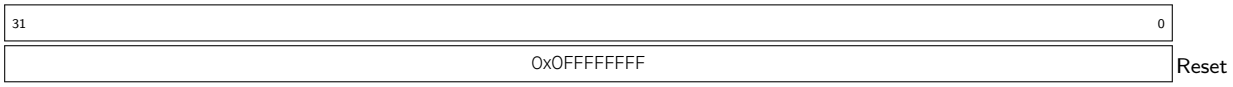
Bit[27]: EMACADDR6LOW_REG[31:24]

Bit[24]: EMACADDR6LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS6_HI Configures the upper 16 bits of the seventh 6-byte MAC address [47:32]. (R/W)

Register 37.42. EMACADDR6LOW_REG (0x0074)



EMACADDR6LOW_REG Configures the lower 32 bits of the seventh 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 37.43. EMACADDR7HIGH_REG (0x0078)



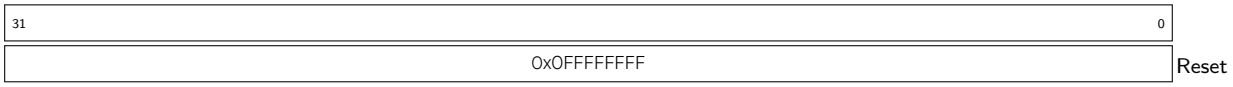
ADDRESS_ENABLE7 Configures whether to enable perfect filtering using the eighth MAC address.
0: Disable
1: Enable
(R/W)

SOURCE_ADDRESS7 Configures whether to which fields of the received frame EMACADDR7 [47:0] compares.
0: The DA fields
1: The SA fields
(R/W)

MASK_BYTE_CONTROL7 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR7.
0: Unmask
1: Mask
The correspondence between the mask control bits and the bytes:
Bit[29]: EMACADDR7HIGH_REG[15:8]
Bit[28]: EMACADDR7HIGH_REG[7:0]
Bit[27]: EMACADDR7LOW_REG[31:24]
Bit[24]: EMACADDR7LOW_REG[7:0]
You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

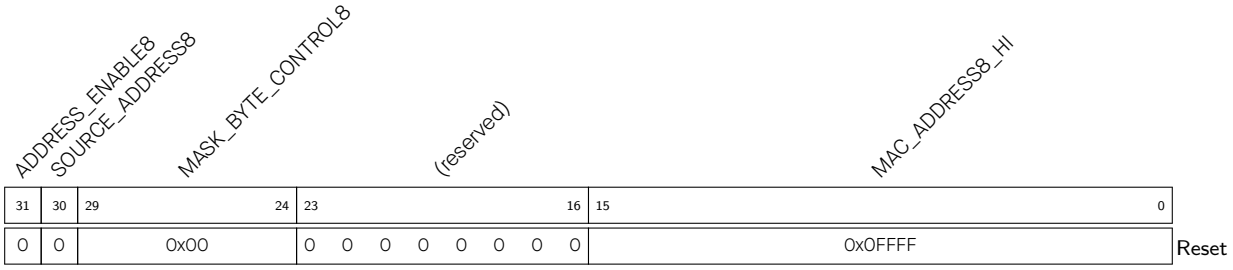
MAC_ADDRESS7_HI Configures the upper 16 bits of the eighth 6-byte MAC address [47:32]. (R/W)

Register 37.44. EMACADDR7LOW_REG (0x007C)



EMACADDR7LOW_REG Configures the lower 32 bits of the eighth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.(R/W)

Register 37.45. EMACADDR8HIGH_REG (0x0080)



ADDRESS_ENABLE8 Configures whether to enable perfect filtering using the ninth MAC address.

0: Disable

1: Enable

(R/W)

SOURCE_ADDRESS8 Configures whether to which fields of the received frame EMACADDR8 [47:0] compares.

0: The DA fields

1: The SA fields

(R/W)

MASK_BYTE_CONTROL8 Configures whether to mask MAC address byte bytes when comparing the received DA or SA with the contents of EMACADDR8.

0: Unmask

1: Mask

The correspondence between the mask control bits and the bytes:

Bit[29]: EMACADDRH8IGH_REG[15:8]

Bit[28]: EMACADDR8HIGH_REG[7:0]

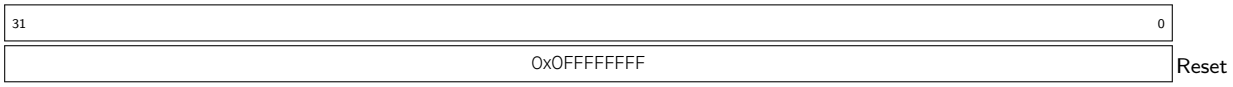
Bit[27]: EMACADDR8LOW_REG[31:24]

Bit[24]: EMACADDR8LOW_REG[7:0]

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS8_HI Configures the upper 16 bits of the ninth 6-byte MAC address [47:32]. (R/W)

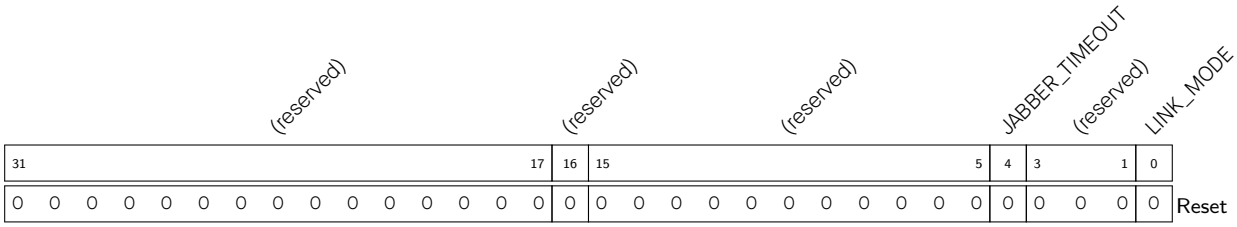
Register 37.46. EMACADDR8LOW_REG (0x0084)



EMACADDR8LOW_REG Configures the lower 32 bits of the ninth 6-byte MAC address.

The content of this field is undefined until loaded by the Application after the initialization process. (R/W)

Register 37.47. EMACCSTATUS_REG (0x00D8)



JABBER_TIMEOUT Represents whether a receive jabber error occurs.

0: Not occur

1: Occur

(RO)

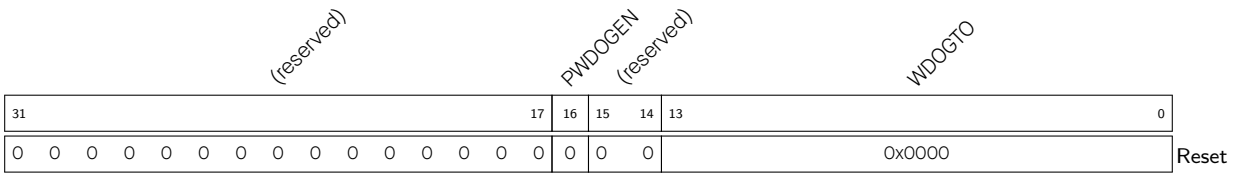
LINK_MODE Represents the current mode of operation of the link.

0: Half-duplex

1: Full-duplex

(RO)

Register 37.48. EMACWDOGTO_REG (0x00DC)



PWDOGEN Configures whether the watchdog timeout is programmble.

0: Not programmable. The watchdog timeout for a received frame is controlled by the setting of EMACCONFIG_REG[23] (WD) and EMACCONFIG_REG[20] (JE).

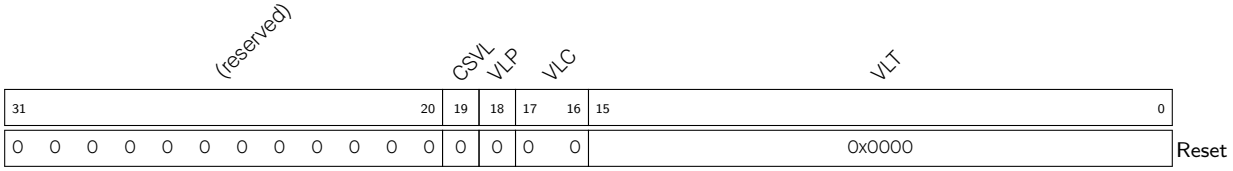
1: Programmable via the WDOGTO field if EMACCONFIG_REG[23] (WD) is 0.

(R/W)

WDOGTO Configures the watchdog timeout for a received frame when PWDOGEN is 1 and EMACCONFIG_REG[23] (WD) is 0.

If the length of a received frame exceeds the value of this field, such frame is terminated and declared as an error frame. (R/W)

Register 37.49. EMACTXVLTCTRL_REG (0x0584)



CSVL Configures which type is inserted or replaced in the 13th and 14th bytes of transmitted frames.

0: C-VLAN type (0x8100)

1: S-VLAN type (0x88A8)

(R/W)

VLP Configures what controls VLAN deletion, insertion, or replacement.

0: The control input signal

1: VLC

(R/W)

VLC Configures the VLAN tag in transmit frames.

0: No VLAN tag deletion, insertion, or replacement.

1: VLAN tag deletion. The MAC removes the VLAN type (bytes 13 and 14) and VLAN tag (bytes 15 and 16) of all transmitted frames with VLAN tags.

2: VLAN tag insertion. The MAC inserts VLT in bytes 15 and 16 of the frame after inserting the Type value (0x8100/0x88a8) in bytes 13 and 14. This operation is performed on all transmitted frames, irrespective of whether they already have a VLAN tag.

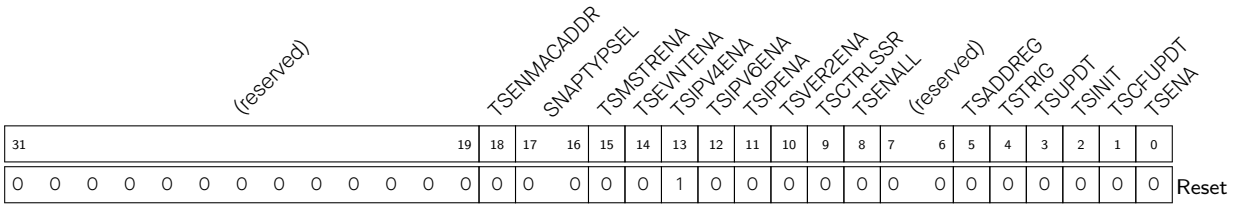
3: VLAN tag replacement. The MAC replaces VLT in bytes 15 and 16 of all VLAN-type transmitted frames (Bytes 13 and 14 are 0x8100/0x88a8).

(R/W)

VLT Configures the value of the VLAN tag to be inserted or replaced.

The value must only be changed when the transmit lines are inactive or during the initialization phase. (R/W)

Register 37.50. EMACTSTPCTRL_REG (0x0700)



TSENMACADDR Configures whether to enable the DA MAC address (that matches any MAC Address register) for PTP frame filtering when PTP is directly sent over Ethernet.

0: Disable

1: Enable

(R/W)

SNAPTYPSEL Configures the set of PTP packet types for which snapshot needs to be taken, in combination with TSMSTRENA and TSEVNTENA.

SNAPTYPSEL, TSMSTRENA, TSEVNTENA:

00x0: SYNC, Follow_Up, Delay_Req, Delay_Resp

0001: SYNC

0011: Delay_Req

01x0: SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up

0101: SYNC, Pdelay_Req, Pdelay_Resp

0111: Delay_Req, Pdelay_Req, Pdelay_Resp

10xx: SYNC, Delay_Req

11xx: Pdelay_Req, Pdelay_Resp

(R/W)

TSMSTRENA Configures whether to enable snapshot only for messages relevant to the master node.

0: Disable. The snapshot is taken for the messages relevant to the slave node

1: Enable

(R/W)

TSEVNTENA Configures whether to enable timestamp snapshot only for event messages (SYNC, Delay_Req, Pdelay_Req, Pdelay_Resp).

0: Disable. The snapshot is taken for all messages except Announce, Management, and Signaling

1: Enable

(R/W)

TSIPV4ENA Configures whether to enable the processing of PTP frames sent over IPv4-UDP.

0: Disable. The MAC ignores the PTP transported over UDP-IPv4 packets

1: Enable. The MAC receiver processes the PTP packets encapsulated in UDP over IPv4 packets

(R/W)

Continued on the next page...

Register 37.50. EMACTSTPCTRL_REG (0x0700)

Continued from the previous page...

TSIPV6ENA Configures whether to enable the processing of PTP frames sent over IPv6-UDP.
0: Disable. The MAC ignores the PTP transported over UDP-IPv6 packets
1: Enable. The MAC receiver processes the PTP packets encapsulated in UDP over IPv6 packets
(R/W)

TSIPENA Configures whether to enable the processing of PTP sent over Ethernet frames.
0: Disable. the MAC ignores the PTP over Ethernet packets
1: Enable. The MAC receiver processes the PTP packets encapsulated directly in the Ethernet frames
(R/W)

TSVER2ENA Configures whether to enable PTP packet processing for version 2 format.
0: Disable. The PTP packets are processed using the version 1 format
1: Enable. The PTP packets are processed using the 1588 version 2 format
(R/W)

TSCTRLSSR Configures the timeout digital or binary rollover.
0: The Timestamp Low register rolls over after the value of the sub-second register 0x7FFF_FFFF. The sub-second increment has to be programmed correctly depending on the PTP reference clock frequency and the value of this bit.
1: The Timestamp Low register rolls over after 0x3B9A_C9FF value (that is, 1 nanosecond accuracy) and increments the timestamp (High) seconds.
(R/W)

TSENALL Configures whether to enable timestamp for all frames received by the MAC.
0: Disable
1: Enable
(R/W)

TSADDREG Configures whether to update the content of the EMACTSTPADDEND_REG register in the PTP block for fine correction.
0: Not update
1: Update
This bit should be zero before setting it. (R/WS/SC)

Continued on the next page...

Register 37.50. EMACTSTPCTRL_REG (0x0700)

Continued from the previous page...

TSTRIG Configures whether to enable the timestamp interrupt trigger.

0: Disable

1: Enable. The timestamp interrupt is generated when the System Time becomes greater than the value written in the Target Time register

(R/WS/SC)

TSUPDT Configures whether to update (add or substract) the system time with the value specified in EMAC2NDUPT_REG and EMACNSUPT_REG.

0: Not update

1: Update

This bit should be read as zero before updating it.

EMACSYSTIMHIGHWORD2ND_REG is not updated. (R/WS/SC)

TSINIT Configures whether to initialize (overwrite) the system time with the value specified in EMAC2NDUPT_REG and EMACNSUPT_REG.

0: Not initialize

1: Initialize

This bit should be read as zero before updating it.

EMACSYSTIMHIGHWORD2ND_REG will be initialized. (R/WS/SC)

TSCFUPDT Configures the method used to update the system time.

0: Coarse method

1: Find update method

(R/W)

TSENA Configures whether to add the timestamp for the transmit and receive frames.

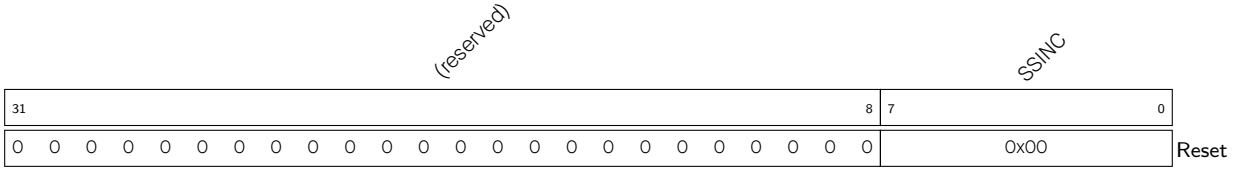
0: Not add the timestamp, and receive frames and the Timestamp Generator is also suspended

1: Add the timestamp

You need to initialize the Timestamp (system time) after setting this bit to 1.

On the receive side, the MAC processes the 1588 frames only if this bit is set. (R/W)

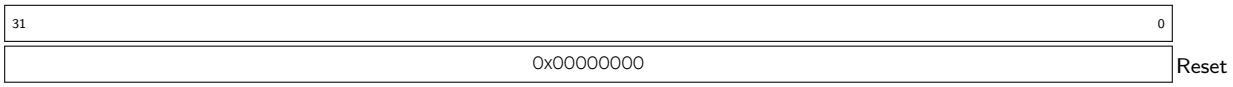
Register 37.51. EMACSUB2NDINCR_REG (0x0704)



SSINC Configures the value accumulated every clock cycle (PTP clock) with the contents of the sub-second register.

For example, when PTP clock is 50 MHz (the period is 20 ns), you should program 20 (0x14) when the System Time-Nanoseconds register has an accuracy of 1 ns (EMACTSTPCTRL_REG[9], i.e., the TSCTRLSSR bit is set). When TSCTRLSSR is cleared, the Nanoseconds register has a resolution of 0.465 ns. In this case, you should program a value of 43 (0x2B) that is derived by 20 ns/0.465. (R/W)

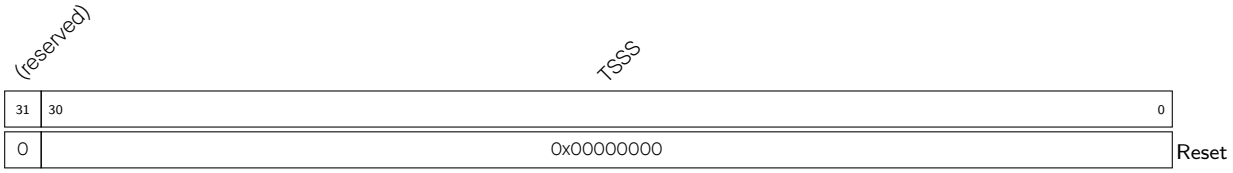
Register 37.52. EMACSYSTIM2ND_REG (0x0708)



TSS Represents the current value of the System Time maintained by the MAC.

Measurement unit: second. (RO)

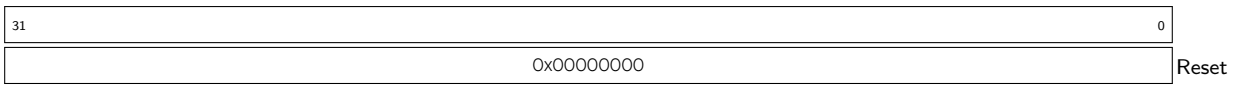
Register 37.53. EMACSYSTIMNS_REG (0x070C)



TSSS Represents the sub-second representation of time, with an accuracy of 0.46 ns.

When EMACTSTPCTRL_REG[9] (TSCTRLSSR) is set, each bit represents 1 ns and the maximum value is 0x3B9A_C9FF. (RO)

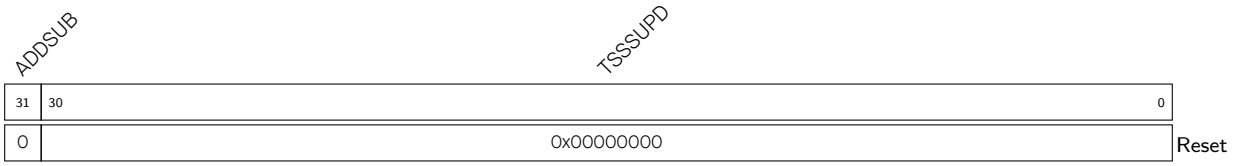
Register 37.54. EMAC2NDUPT_REG (0x0710)



TSSSUPD Configures the time to be initialized or added to the system time.

Measurement unit: second. (R/W)

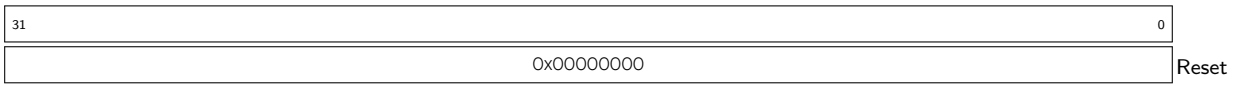
Register 37.55. EMACNSUPT_REG (0x0714)



ADDSUB with Configures whether to substract or add the system time value the contents of the update register.
0: Add
1: Substract
(R/W)

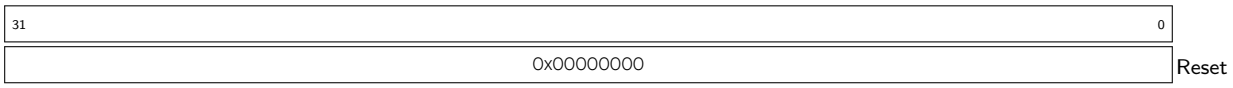
TSSSUPD Configures the sub-second representation of time, with an accuracy of 0.46 ns.
When EMACTSTPCTRL_REG[9] (TSCTRLSSR) is set, each bit represents 1 ns and the programmed value should not exceed 0x3B9A_C9FF. (R/W)

Register 37.56. EMACTSTPADDEND_REG (0x0718)



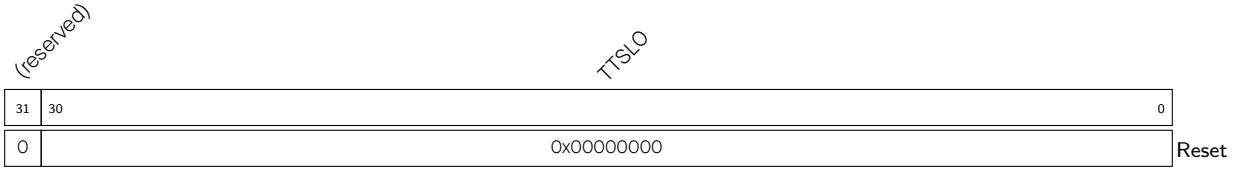
TSAR Configures the 32-bit time value to be added to the Accumulator register to achieve time synchronization. (R/W)

Register 37.57. EMACTGTIME2ND_REG (0x071C)



TSTR Configures the target time.
Measurement unit: second.
When the system time matches or exceeds the target time, an interrupt is generated (if enabled). (R/W)

Register 37.58. EMACTGTIMENS_REG (0x0720)

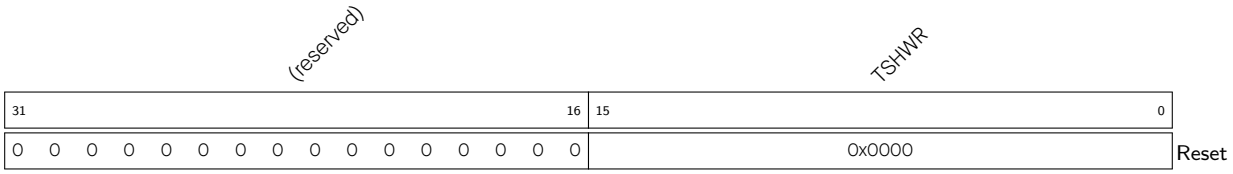


TTSLO Configures the signed target time.

Measurement unit: nanosecond.

When the system time matches or exceeds the target time, an interrupt is generated (if enabled). (R/W)

Register 37.59. EMACSYSTIMHIGHWORD2ND_REG (0x0724)

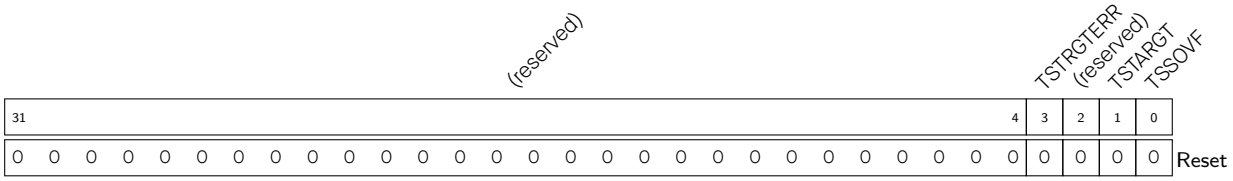


TSHWR Configures the most significant 16-bits of the timestamp seconds value.

The register is directly written to initialize the value.

This register is incremented when there is an overflow from the 32 bits of the System Time Seconds register. (R/W/SU)

Register 37.60. EMACTSTPSTATUS_REG (0x0728)



TSTRGTERR Represents whether the target time, being programmed in EMACTGTIME2ND_REG and EMACTGTIMENS_REG, is already elapsed.

0: Not elapsed

1: Elapsed

(R/SS/RC)

TSTARGT Represents whether the system time is greater or equal to the value specified in EMACTGTIME2ND_REG and EMACTGTIMENS_REG.

0: Less than the value

1: Greater or equal to the value

(R/SS/RC)

TSSOVF Represents whether the second value of the timestamp has overflowed beyond 0xFFFF_FFFF.

0: No overflow

1: Overflow

(R/SS/RC)

Chapter 38

Two-Wire Automotive Interface (TWAI)

The Two-wire Automotive Interface (TWAI®) is a multi-master, multi-cast communication protocol with functions such as error detection and signaling and inbuilt message priorities and arbitration. The TWAI protocol is suited for automotive and industrial applications (see Section 38.2 for more details).

ESP32-P4 contains three TWAI controllers, TWAI 0, TWAI 1, and TWAI 2. Each controller can individually be connected to a TWAI bus via an external transceiver. The TWAI controllers have many advanced features, and can be utilized in a wide range of use cases, such as automotive products, industrial automation controls, building automation, etc.

38.1 Features

Each TWAI controller on ESP32-P4 supports the following features:

- Compatibility with ISO 11898-1 protocol (CAN Specification 2.0)
- Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation:
  - Normal
  - Listen-only (no influence on bus)
  - Self-test (no acknowledgment required during data transmission)
- 64-byte Receive FIFO
- Special transmissions:
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self-reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports Single and Dual-filter modes)
- Error detection and handling:
  - Error counters
  - Configurable error warning limit
  - Error code capture
  - Arbitration lost capture
  - Automatic transceiver standby

38.2 Protocol Overview

38.2.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows nodes to exchange messages with deterministic delay. A TWAI bus has the following properties:

Single Channel and Non-Return-to-Zero: The bus has only one transmission line for single-channel communication, which is half-duplex. Synchronization is also implemented in this channel, so extra channels (e.g., clock or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a dominant or recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting data in a dominant state always overrides the other node transmitting data in a recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential pair or a single wire).

Bit Stuffing: Certain fields of TWAI messages are bit-stuffed. A transmitter that transmits five consecutive bits of the same value (e.g., dominant value or recessive value) should automatically insert a complementary bit. Likewise, a receiver that receives five consecutive bits of the same value should treat the next bit as a stuffed bit. Bit stuffing is applied to the following fields: SOF, arbitration field, control field, data field, and CRC sequence (see Section 38.2.2 for more details).

Multi-Cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (see Section 38.2.3 for more details).

Multi-Master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before initiating a new transmission.

Message Priority and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The arbitration field of the message transmitted by each node is used to determine which node will win the arbitration.

Error Detection and Signaling: Each node actively monitors the bus for errors, and signals the detected errors by transmitting an error frame.

Fault Confinement: Each node maintains a set of error counters that are incremented/decremented according to a set of rules. When the error counters surpass a certain threshold, the node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes on the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a transmitter or a receiver.

- A node generating a message is a transmitter. The node remains a transmitter until the bus is idle or until the node loses arbitration. Please note that there could be multiple nodes that act as transmitters during arbitration.
- All nodes that are not transmitters can work as receivers.

38.2.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes when detecting errors on the bus. Messages are split into various frame types, and different frame types have different frame formats.

The TWAI protocol has the following frame types:

- Data frame
- Remote frame
- Error frame
- Overload frame
- Interframe space

The TWAI protocol supports the following frame formats:

- Standard Frame Format (SFF) that uses a 11-bit identifier
- Extended Frame Format (EFF) that uses a 29-bit identifier

38.2.2.1 Data Frames and Remote Frames

Data frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote frames are used for nodes to request a data frame with the same identifier from other nodes, and thus they do not contain any data bytes. However, data frames and remote frames share many fields. Figure 38.2-1 illustrates the fields and sub-fields of different frames and formats.

Arbitration Field

When two or more nodes transmit data or remote frames simultaneously, the arbitration field is used to determine which node will win arbitration of the bus. In the arbitration field, if a node transmits a recessive bit while detecting a dominant bit, it indicates that another node has overridden its recessive bit. Therefore, the node transmitting the recessive bit has lost arbitration of the bus and should immediately switch to be a receiver.

The arbitration field primarily consists of a frame identifier that is transmitted from the most significant bit first. Given that a dominant bit represents a logical 0, and a recessive bit represents a logical 1:

- A frame with the smallest ID value always wins arbitration.
- Given the same ID and format, data frames always prevail over remote frames due to their RTR bits being dominant.
- Given the same first 11 bits of ID, a Standard Format Data Frame always prevails over an Extended Format Data Frame due to its SRR bits being recessive.

Control Field

The control field primarily consists of the Data Length Code (DLC) which indicates the number of payload data bytes for a data frame, or the number of requested data bytes for a remote frame. The DLC is transmitted from the most significant bit first.

Data Field

The data field contains the actual payload data bytes of a data frame. Remote frames do not contain any data field.

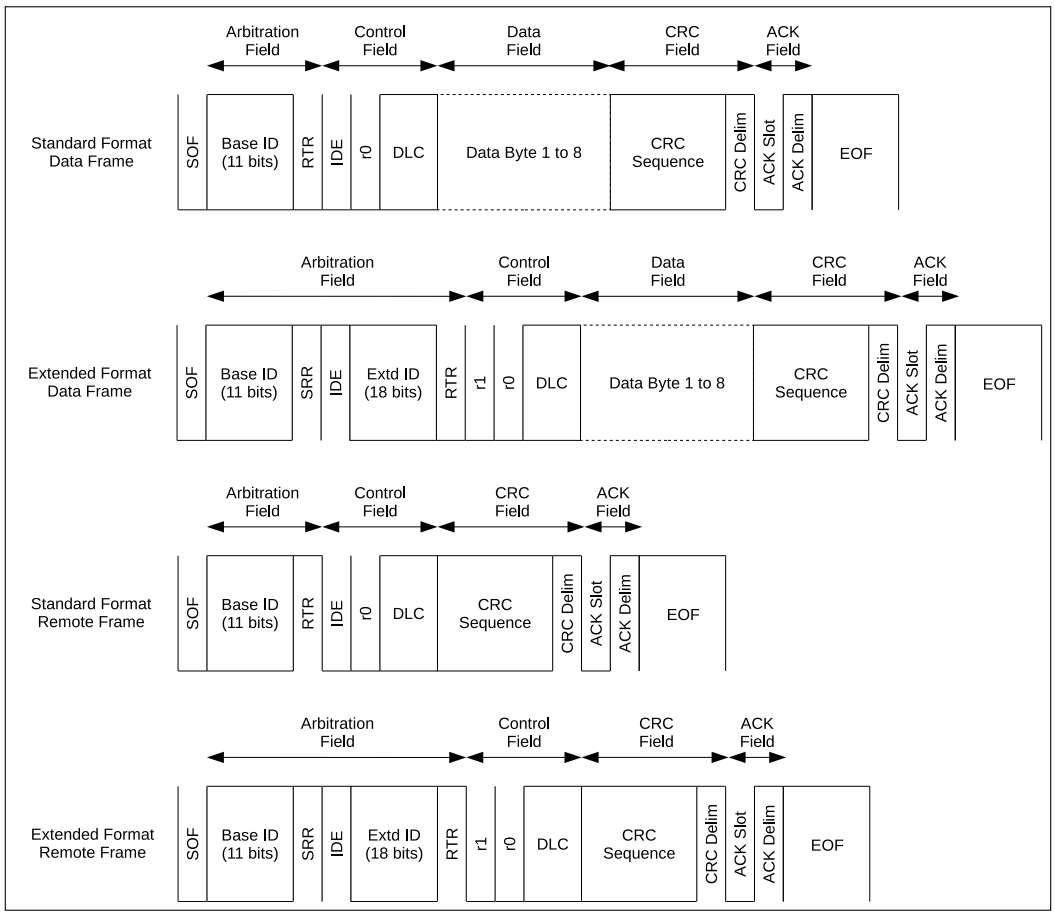


Figure 38.2-1. Bit Fields in Data Frames and Remote Frames

CRC Field

The CRC field primarily consists of a CRC sequence. The CRC sequence is a 15-bit cyclic redundancy code calculated from the de-stuffed contents (everything from the SOF to the end of the data field) of a data or remote frame.

ACK Field

The ACK field primarily consists of an ACK Slot and an ACK Delim. The ACK field indicates that the receiver has received an effective message from the transmitter.

Table 38.2-1. Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit identifier for SFF, or the first 11 bits of the 29-bit identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a data frame (dominant) or a remote frame (recessive). This means that a remote frame will always lose arbitration to a data frame if they have the same ID.

Cont'd on next page

Table 38.2-1 – cont'd from previous page

Data/Remote Frames	Description
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The IDE (Identifier Extension) bit indicates whether the message is SFF (dominant) or EFF (recessive). This means that a SFF frame will always win arbitration over an EFF frame if they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18 bits of the 29-bit identifier for EFF.
r1	The r1 bit (reserved bit 1) is always dominant.
r0	The r0 bit (reserved bit 0) is always dominant.
DLC	The DLC (Data Length Code) is 4-bit long and should contain any value from 0 to 8. Data frames use the DLC to indicate the number of data bytes in the data frame. Remote frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of data frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted from the most significant bit first.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.
CRC Delim	The CRC Delim (CRC Delimiter) is a single recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) is intended for receiver nodes to indicate that the data or remote frame was received without any issue. The transmitter node will send a recessive bit in the ACK Slot and receiver nodes should override the ACK Slot with a dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimiter) is a single recessive bit.
EOF	The EOF (End of Frame) marks the end of a data or remote frame, and consists of seven recessive bits.

38.2.2.2 Error and Overload Frames

Error Frames

Error frames are transmitted when a node detects a bus error. Error frames notably include an Error Flag which is made up of six consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a bus error and transmits an error frame, all other nodes will then detect a stuff error and transmit their own error frames in response. This has the effect of propagating the detection of a bus error across all nodes on the bus.

When a node detects a bus error, it will transmit an error frame starting from the next bit. However, when a node detects a CRC error, the error frame will start at the bit following the ACK Delim (see Section 38.2.3 for more details). The following Figure 38.2-2 shows different fields of an error frame:

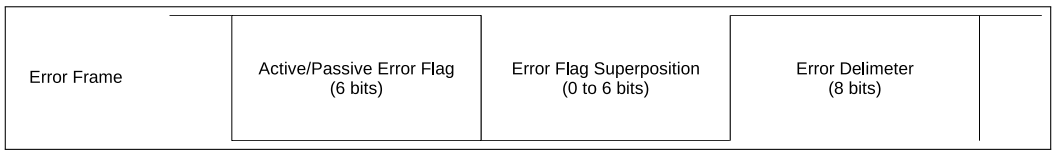


Figure 38.2-2. Fields of an Error Frame

Table 38.2-2. Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of six dominant bits and the Passive Error Flag consisting of six recessive bits (unless overridden by dominant bits of other nodes). Active Error Flags are sent by error active nodes, whilst Passive Error Flags are sent by error passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first recessive bit is detected (i.e., the first it of the Delimiter).
Error Delimeter	The Delimiter field marks the end of the error/overload frame, and consists of eight recessive bits.

Overload Frames

An overload frame has the same bit fields as an error frame containing an Active Error Flag. The key difference is in the cases that can trigger the transmission of an overload frame. Figure 38.2-3 below shows the bit fields of an overload frame.

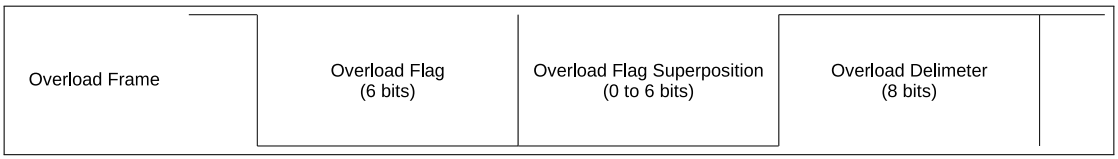


Figure 38.2-3. Fields of an Overload Frame

Table 38.2-3. Overload Frame

Overload Flag	Description
Overload Flag	Consists of six dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimiter	Consists of eight recessive bits. Same as an Error Delimiter.

Overload frames will be transmitted in the following cases:

1. A receiver requires a delay of the next data or remote frame.
2. A dominant bit is detected at the first and second bit of intermission.

3. A dominant bit is detected at the eighth (last) bit of an Error Delimiter. Note that in this case, TEC and REC will not be incremented (see Section 38.2.3 for more details).

Transmitting an overload frame due to one of the above cases must also satisfy the following rules:

- The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission. In this case, a maximum of two overload frames may be generated.
- The start of an overload frame due to case 2 and 3 is only allowed to be started one bit after detecting the dominant bit.

38.2.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Before sending, data frames and remote frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (data frame, remote frame, error frame, or overload frame). However, error frames and overload frames do not need to be separated from preceding frames.

Figure 38.2-4 shows the fields within an Interframe Space:

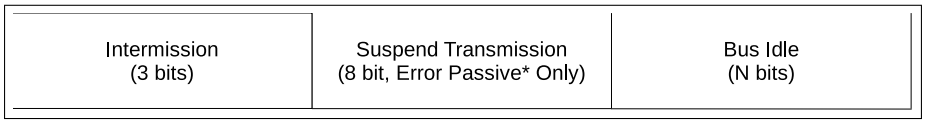


Figure 38.2-4. The Fields within an Interframe Space (*Error Passive)

Table 38.2-4. Interframe Space

Interframe Space	Description
Intermission	The Intermission consists of three recessive bits.
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of eight recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following Suspend Transmission.

38.2.3 TWAI Errors

38.2.3.1 Error Types

Bus Errors in TWAI are categorized into the following types:

Bit Error

A Bit Error occurs when a node transmits a bit value (i.e., dominant or recessive) but detects an opposite bit (e.g., a dominant bit is transmitted but a recessive is detected). However, if the transmitted bit is recessive and is located in the Arbitration Field, ACK Slot, or Passive Error Flag, then detecting a dominant bit will not be considered as a Bit Error.

Stuff Error

A stuff error occurs when six consecutive bits of the same value are detected (which violates the bit-stuffing encoding rules).

CRC Error

The receiver will calculate the CRC value based on the valid bits of received data frames and remote frames (before bit stuffing). A CRC Error occurs when the CRC calculated by the receiver does not match the CRC sequence in the received data frames and remote frames.

Format Error

A Format Error occurs when a format-fixed bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be dominant.

ACK Error

An ACK Error occurs when a transmitter does not detect a dominant bit at the ACK Slot.

38.2.3.2 Error States

TWAI nodes implement fault confinement by maintaining two error counters in each node, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states:

Error Active

An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication and transmit a Passive Error Flag when it detects an error. Error Passive nodes that have transmitted data or remote frames must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit data).

38.2.3.3 Error State Transition

1. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. Though the node becomes Error Passive, it still sends an Active Error Flag. Note that once the REC has reached 128, any further increases to its value are invalid until the REC returns to a value less than 128.
2. A node becomes Bus Off when its TEC is greater than or equal to 256.
3. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
4. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive recessive bits on the bus.

38.2.3.4 Error Counter Rules

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply to a given message transfer.**

TEC increment/decrement rules:

1. When a transmitter sends an Error Flag, the TEC is increased by 8. However, the following scenarios are exempt from this rule:
   - A transmitter is Error Passive and no dominant bit is detected when an Acknowledgment Error is detected and the Passive Error Flag is sent. In this case, the TEC should not be increased.
   - A transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the stuffed bit should have been recessive but was monitored as dominant, then the TEC should not be increased.
2. If a transmitter detects a Bit Error while sending an Active Error Flag or Overload Flag, the TEC is increased by 8.
3. A node can tolerate up to 7 consecutive dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit when sending an Active Error Flag or Overload Flag, or the 8th consecutive dominant bit following a Passive Error Flag, a transmitter will increase its TEC by 8 and a receiver will increase its REC by 8. Every additional 8 consecutive dominant bits will also increase the TEC for transmitters or REC for receivers by 8 as well.
4. When a transmitter has transmitted a message, which means getting ACK and no errors until the EOF is completed, the TEC is decremented by 1, unless the TEC is already at 0.

REC increment/decrement rules:

1. When a receiver detects a dominant bit as the first bit after sending an Error Flag, the REC is increased by 8.
2. When a receiver detects an error, the REC is increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
3. If a receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
4. When a receiver successfully receives a message, which means getting no errors before ACK Slot and successfully sending ACK, the REC is decremented accordingly.
   - If the REC is between 1 and 127, it will be decremented by 1.
   - If the REC is greater than 127, it will be set to 127.
   - If the REC is 0, it will remain 0.

38.2.4 TWAI Bit Timing

38.2.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- **The Nominal Bit Rate** is defined as the number of bits transmitted per second.
- **The Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a minimum unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 38.2-5 illustrates the segments within a single Nominal Bit Time.

TWAI controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed. If the bus states in two consecutive Time Quanta are different (i.e., recessive to dominant or vice versa), an edge is generated. The intersection of PBS1 and PBS2 is considered the Sample Point and the sampled bus value is considered the value of that bit.

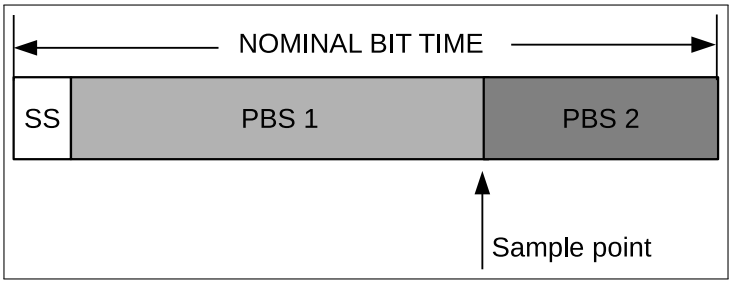


Figure 38.2-5. Layout of a Bit

Table 38.2-5. Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.
PBS2	PBS2 (Phase Buffer Segment 2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

38.2.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules:

- Only one synchronization may occur in a single bit time.
- Synchronizations only occur on recessive to dominant edges.

Hard Synchronization

Hard Synchronization occurs on the recessive to dominant (i.e., the first SOF bit after Bus Idle) edges when the bus is idle. All nodes will restart their internal bit timings so that the recessive to dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on recessive to dominant edges when the bus is not idle. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is programmable.

- When the magnitude of the Phase Error (**e**) is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by the **e** number of Time Quanta. This has the same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater than the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

38.3 Architectural Overview

The major functional blocks of the TWAI controller are shown in Figure 38.3-1.

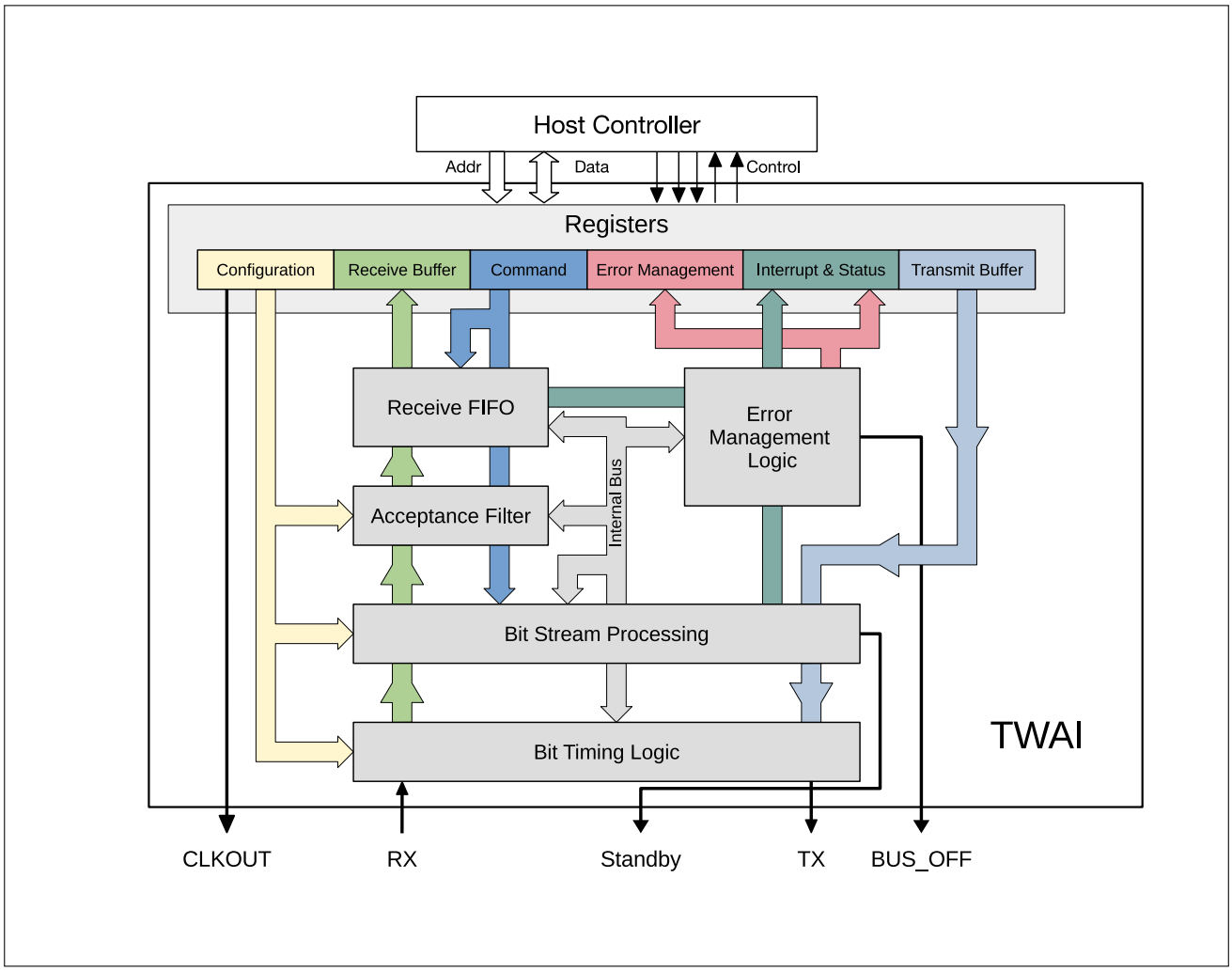


Figure 38.3-1. TWAI Overview Diagram

38.3.1 Bit Timing Logic

The Bit Timing Logic (BTL) module transmits and receives messages at the configured bit rate. The BTL module also handles bit timing synchronization so that communication remains stable. A single bit rate consists of multiple programmable segments that allows users to set the number of time quanta to adjust propagation delay and controller processing time, etc.

38.3.2 Bit Stream Processor

The Bit Stream Processing (BSP) module handles the frame processing of data from the Transmit Buffer (e.g., bit stuffing and additional CRC fields) and generates a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

38.3.3 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter can be configured to Single-filter mode or Dual-filter mode through registers.

38.3.4 Receive FIFO

The Receive FIFO is a 64-byte buffer (inside the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13 bytes). When the Receive FIFO is full (or does not have enough space to store the currently received message in its entirety), the Overrun Interrupt will be triggered, and any subsequently received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After being cleared, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

38.3.5 Error Management Logic

The Error Management Logic (EML) module updates the TEC and REC, records error information like error types and positions, and updates the error state of the TWAI controller to ensure that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

38.3.6 Registers Block

The ESP32-P4 CPU accesses peripherals using 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration items for the TWAI controller such as bit rates, Operation mode, Acceptance Filter, etc. Configuration registers can only be modified whilst the TWAI controller is in Reset mode (See Section 38.4.1).

Command Registers

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation mode (see Section 38.4.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management registers include error counters and capture registers. The error counter registers represent TEC and REC values. The capture registers will record information about instances where the TWAI controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window of Receive FIFO, whose first message will be mapped into the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset mode, all reads and writes to the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation mode:
  - All reads to the address range maps to the Receive Buffer registers.
  - All writes to the address range maps to the Transmit Buffer registers.

38.4 Functional Description

38.4.1 Modes

The ESP32-P4 TWAI controller has two working modes: Reset mode and Operation mode. Reset mode and Operation mode are entered by setting or clearing the TWAI_RESET_MODE bit.

38.4.1.1 Reset Mode

Entering Reset mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset mode, the TWAI controller will not be able to transmit any messages (including error signals). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will not be able to receive any messages either.

38.4.1.2 Operation Mode

In Operation mode, the TWAI controller connects to the bus and write-protects all configuration registers to ensure consistency during operation. When in Operation mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operation sub-mode the TWAI controller was configured with. The TWAI controller supports the following operation sub-modes:

- **Normal mode:** The TWAI controller can transmit and receive messages including error signals (such as Error and Overload Frames).
- **Self-test mode:** Self-test mode is similar to Normal mode, but the TWAI controller will consider the transmission of a data or remote frame successful and do not generate an ACK error even if it was not acknowledged. This mode is commonly used during the self-test of a TWAI controller.
- **Listen-only mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitoring.

Note that when exiting Reset mode (i.e., entering Operation mode), the TWAI controller must wait for 11 consecutive recessive bits to occur before fully connecting to the TWAI bus (i.e., being able to transmit or receive).

38.4.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset mode. The bit rate is configured using TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG.

The following Table 38.4-1 illustrates the bit fields of TWAI_BUS_TIMING_0_REG. The frequency of the TWAI core clock has multiple clock sources that can be configured by the user as needed. See Chapter 8 *Reset and Clock* for detailed configuration instructions.

Table 38.4-1. Bit Information of TWAI_BUS_TIMING_0_REG (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

Notes:

- BRP: The TWAI Time Quanta clock is derived from the XTAL clock (the default is 40 MHz and is configured). The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock cycle and t_{CLK} is TWAI core clock cycle:
  t_{Tq} = 2 × t_{CLK} × (BRP + 1)
- SJW (t_{SJW}): Synchronization Jump Width (SJW) t_{SJW} is configured in SJW[1:0] where t_{SJW} = (SJW[1:0] + 1).

The following Table 38.4-2 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 38.4-2. Bit Information of TWAI_BUS_TIMING_1_REG (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined based on PBS1 + 1.
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined based on PBS2 + 1.
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses to filter spikes on the bus line.

38.4.3 Transmit and Receive Buffers

38.4.3.1 Overview of Buffers

Table 38.4-3. Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
Offset Address	Content	Offset Address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 38.4-3 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation mode. The CPU accesses Transmit Buffer registers for write operations, and Receive Buffer registers for read operations. Both buffers share the exact same register layout and fields to store a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQUEST bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQUEST bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQUEST and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUFFER bit in TWAI_CMD_REG to clear the Receive Buffer registers. If there are still messages in the Receive FIFO, the Receive Buffer registers will map the first of the remaining messages again.

38.4.3.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 38.4-4.

Table 38.4-4. TX/RX Frame Information (SFF/EFF); TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	DLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

1. FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when the FF bit is 1, and SFF when the FF bit is 0.
2. RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a data frame or a remote frame. The message is a remote frame when the RTR bit is 1, and a data frame when the RTR bit is 0.
3. X: Don't care, can be any value.
4. DLC: The Data Length Code (DLC) field specifies the number of data bytes for a data frame, or the number of data bytes to request in a remote frame. TWAI data frames are limited to a maximum payload of 8 data bytes, and thus the DLC should range from 0 to 8.

38.4.3.3 Frame Identifier

The Frame Identifier fields occupy two-byte (11-bit) long if the message is SFF, and four-byte (29-bit) long if the message is EFF.

The Frame Identifier fields for an SFF (11-bit) message are shown in Table 38.4-5 ~ 38.4-6.

Table 38.4-5. TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

Table 38.4-6. TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).

2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 38.4-7 ~ 38.4-10.

Table 38.4-7. TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 38.4-8. TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 38.4-9. TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 38.4-10. TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X^1	X^2	X^2

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self-reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self-reception functionality (or together with self-test functionality).

38.4.3.4 Frame Data

The Frame Data field contains the payloads of transmitted or received data frame, and can range from 0 to 8 bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than eight bytes, the number of valid bytes would still be limited to eight. Remote frames do not have data payloads, so their Frame Data fields will be unused.

For example, when transmitting a data frame with five bytes, the CPU should write five to the DLC field, and then write data to the corresponding register of the first to the fifth data field. Likewise, when the CPU receives a data frame with a DLC of five data bytes, only the first to the fifth data byte will contain valid payload data for the CPU to read.

38.4.4 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13 bytes of space in the Receive FIFO, and their

endianness is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO.

When the TWAI controller receives the first message, the value of TWAI_RX_MESSAGE_COUNTER increases to 1, and the RXI interrupt is activated, automatically updating the first message into the buffer. Subsequently, with each new message received by the TWAI controller, the value of TWAI_RX_MESSAGE_COUNTER increments by 1, reaching a maximum of 64. When this limit is reached and there is sufficient space remaining in the Receive FIFO, the message will be written into the FIFO.

Once the software reads a message from the Receive Buffer, it can free up the space occupied by the message in the Receive FIFO by setting TWAI_RELEASE_BUFFER to 1. This will also decrement TWAI_RX_MESSAGE_COUNTER by 1. The Receive Buffer will then map to the next message in the Receive FIFO. The software should repeat the above operations until TWAI_RX_MESSAGE_COUNTER reaches 0. At this point, the RXI interrupt stops being triggered, the data in the receive buffer becomes invalid, and TWAI_STATUS_RECEIVE_BUFFER is set to 0.

A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs:

- The free space left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- When data in the Receive FIFO overruns for the first time, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The Receive FIFO will internally mark overrun messages as invalid. The TWAI_STATUS_MISS bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUFFER must be called repeatedly until TWAI_RX_MESSAGE_COUNTER is 0. This requires users to read all valid messages in the Receive FIFO and clear all overrun messages.

38.4.5 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) since the TWAI Controller only needs to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset mode, since they share the same address spaces with the Transmit Buffer and Receive Buffer registers.

The configuration registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Acceptance Code value specifies a bit pattern which each filtered bit of the message must match in order for the message to be accepted. The Acceptance Mask Value is able to mask out certain bits of the Code value (i.e., set as "Don't Care" bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 38.4-1.

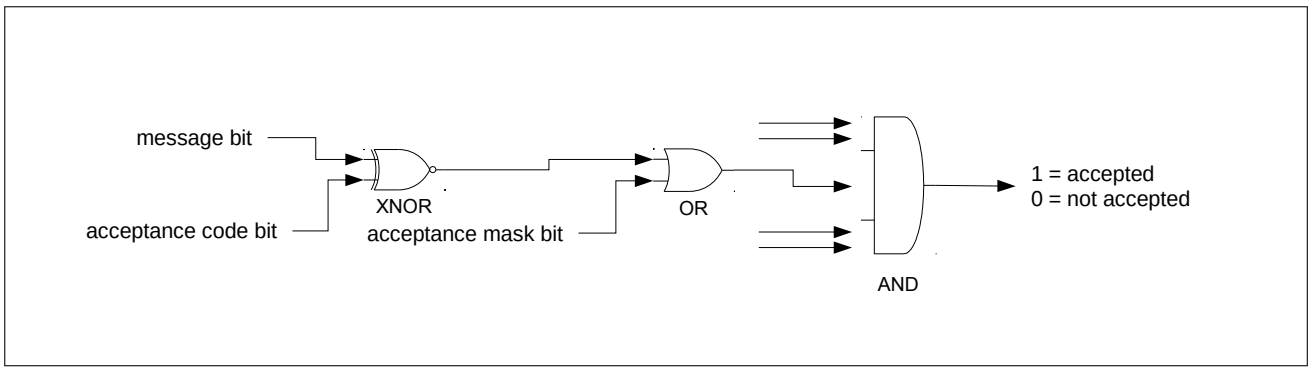


Figure 38.4-1. Acceptance Filter

The TWAI controller Acceptance Filter allows the 32-bit Acceptance Code and Mask Values to either define a single filter (i.e., Single-filter mode), or two filters (i.e., Dual-filter mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on the filter mode and the format of received messages (i.e., SFF or EFF).

38.4.5.1 Single-Filter Mode

Single Filter mode is enabled by setting the TWAI_ACCEPTANCE_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of data or remote frames:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 38.4-2 illustrates how the 32-bit code and mask values will be interpreted under Single-filter mode.

38.4.5.2 Dual-Filter Mode

Dual-filter mode is enabled by clearing the TWAI_ACCEPTANCE_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define a two separate filters referred to as filter 1 or filter 2. Under Dual-filter mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of data or remote frames:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)

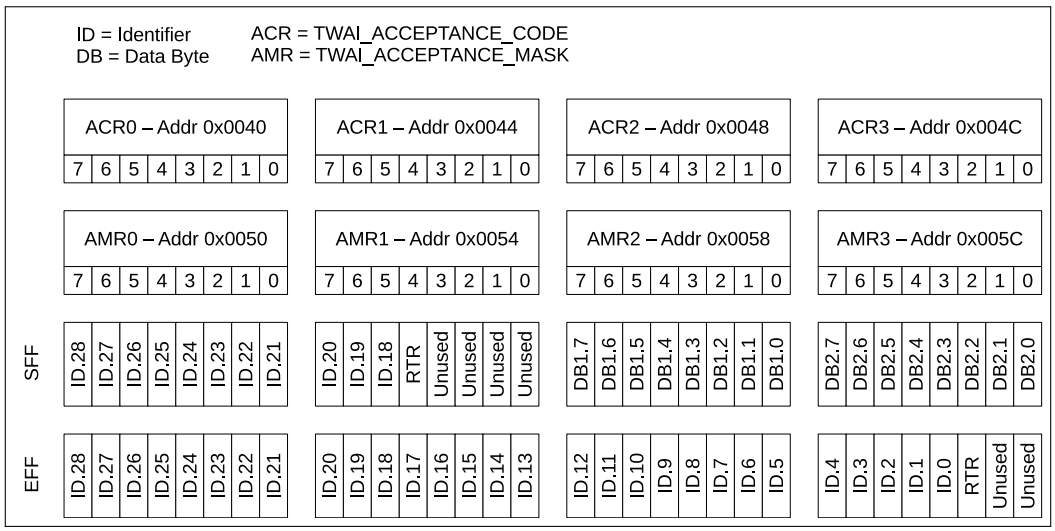


Figure 38.4-2. Single-Filter Mode

- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 38.4-3 illustrates how the 32-bit code and mask values will be interpreted in Dual-filter mode.

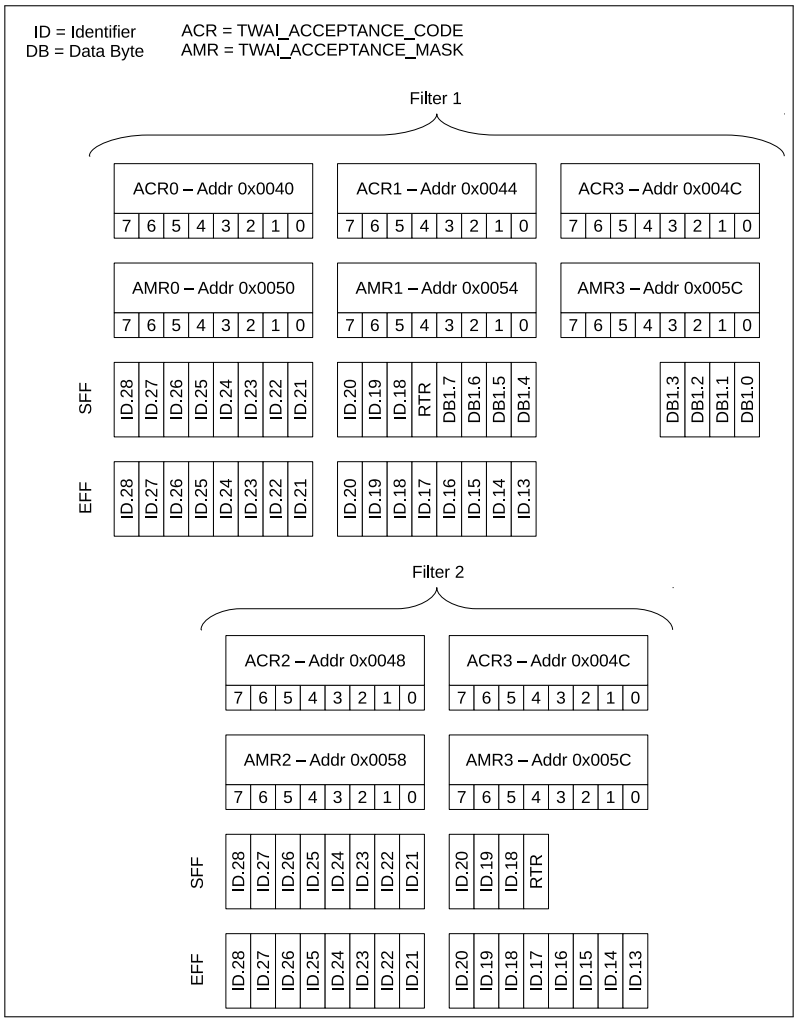


Figure 38.4-3. Dual-Filter Mode

38.4.6 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Counter (TEC) and Receive Error Counter (REC). The value of both error counters determines the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and they can be read by the CPU anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn users of the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_STATUS_ERR, and TWAI_STATUS_NODE_BUS_OFF. Certain changes to these values and bits will also trigger interrupts, so that users are notified of error state transitions (see section 38.5). The following figure 38.4-4 shows the relation between the error states, values and bits, and error state related interrupts.

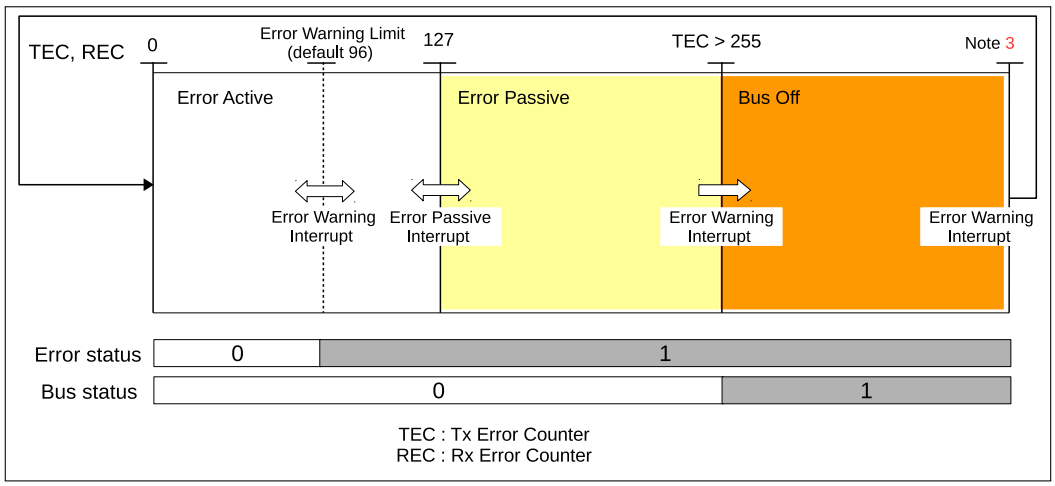


Figure 38.4-4. Error State Transition

38.4.6.1 Error Warning Limit

The Error Warning Limit (EWL) is a configurable threshold value for the TEC and REC, which will trigger an interrupt when exceeded. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96.

When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_STATUS_ERR bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_STATUS_ERR bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_STATUS_ERR bit (or the TWAI_STATUS_NODE_BUS_OFF) changes.

38.4.6.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

38.4.6.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_STATUS_NODE_BUS_OFF bit to 1
- Enter Reset mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_STATUS_NODE_BUS_OFF bit (or the TWAI_STATUS_ERR bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off Recovery. Bus-Off Recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive recessive bits on the bus. To initiate Bus-Off Recovery (after entering the Bus-Off state), the TWAI controller should enter Operation mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off Recovery by decrementing the TEC each time when the TWAI controller observes 11 consecutive recessive bits. When Bus-Off Recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_STATUS_NODE_BUS_OFF bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

38.4.7 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from TWAI_ERR_CODE_CAP_REG.

The following Table 38.4-11 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 38.4-11. Bit Information of TWAI_ERR_CODE_CAP_REG (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error; 00 for bit error, 01 for format error, 10 for stuff error, and 11 for other types of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error occurred; 0 for the transmitter, 1 for the receiver.
- SEG: The Error Segment (SEG) indicates the segment of the TWAI message at which the bus error occurred.

The following Table 38.4-12 shows how to interpret the SEG.0 to SEG.4 bits.

Table 38.4-12. Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0

Cont'd on next page

Table 38.4-12 – cont'd from previous page

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	ACK slot
1	1	0	1	1	ACK delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit SRTR: under Standard Frame Format.
- Bit IDE: Identifier Extension Bit, 0 for Standard Frame Format.

38.4.8 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in TWAI_ARB_LOST_CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent losses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in TWAI_ARB_LOST_CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 38.4-13 illustrates bits and fields of TWAI_ERR_CODE_CAP_REG whilst Figure 38.4-5 illustrates the bit positions of a TWAI message.

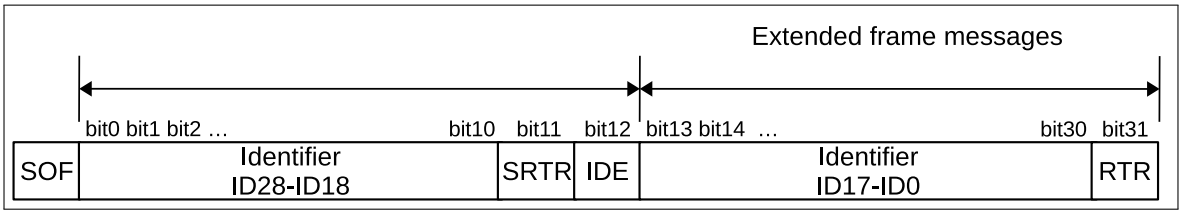


Figure 38.4-5. Positions of Arbitration Lost Bits

Table 38.4-13. Bit Information of TWAI_ARB_LOST_CAP_REG (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

38.4.9 Transceiver Auto-Standby

It is common for TWAI transceivers to support a Standby mode to lower power consumption. TWAI transceivers will usually generate a standby signal that is asserted by the connected TWAI controller, thus allowing the controller to place the transceiver into standby when appropriate (e.g., when the bus will be idle for an extended period of time). Transceivers will exit Standby mode if the controller de-asserts the standby signal, or if the transceiver detects bus activity (also known as a wake-up feature).

ESP32-P4's TWAI controller supports both hardware control (i.e., automatic) and software control (i.e., manual) of the standby signal to control the switching of TWAI transceivers connected to the chip. When hardware controlled, the TWAI controller will automatically assert the standby signal when the bus remains idle for longer than a configurable amount of time. When software controlled, the standby signal can be manually asserted/de-asserted directly by the software.

- Hardware output:
  1. Set the TWAI_HW_STANDBY_EN field in the TWAI_HW_CFG_REG register to enable standby function for hardware.
  2. Configure the TWAI_HW_STANDB_CNT_REG register. This register indicates the time required before hardware triggers the standby signal after entering idle status, in which the value indicates the number of cycles of the TWAI controller operating clock (40 MHz by default).
- Software output:
  1. Set the TWAI_SW_STANDBY_EN field in the TWAI_SW_STANDBY_CFG_REG register to generate standby signals in the TWAI controller.

The standby signal generated using either of the above methods will be pulled down (cleared) when either of the following conditions is met:

1. The standby signal will be automatically cleared when the TWAI controller's receiver exits the idle status (such as when receiving a new message).
2. Users can also pull down the standby signal by setting the TWAI_SW_STANDBY_CLR field in the TWAI_SW_STANDBY_CFG_REG register.

38.5 Interrupts

ESP32-P4's TWAI can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- TWAI_INT

There are several internal interrupt sources from the TWAI that can generate the above interrupt signal. The interrupt sources from the TWAI are listed with their trigger conditions and the resulted interrupt signal in Table 38.5-1.

Table 38.5-1. TWAI's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_RECEIVE_INT (RXI)	The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to be read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_COUNTER_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUFFER command bit.	TWAI_INT
TWAI_TRANSMIT_INT (TXI)	The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer for transmission. The Transmit Buffer becomes free under the following scenarios: • A message transmission has completed successfully, i.e., acknowledged without any errors. Any failed messages will automatically be resent. • A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_STATUS_TRANSMISSION_COMPLETE bit). • A message transmission was aborted using the TWAI_ABORT_TX command bit.	TWAI_INT

Cont'd on next page

Table 38.5-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_ERR_WARNING_INT (EWI)	The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_STATUS_ERR and TWAI_STATUS_NODE_BUS_OFF bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_STATUS_ERR and TWAI_STATUS_NODE_BUS_OFF at the moment when the EWI is triggered. • If TWAI_STATUS_ERR = 0 and TWAI_STATUS_NODE_BUS_OFF = 0: – If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG. – If the TWAI controller was previously in the Bus Off Recovery state, it indicates that Bus Recovery has completed successfully. • If TWAI_STATUS_ERR = 1 and TWAI_STATUS_NODE_BUS_OFF = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG. • If TWAI_STATUS_ERR = 1 and TWAI_STATUS_NODE_BUS_OFF = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256). • If TWAI_STATUS_ERR = 0 and TWAI_STATUS_NODE_BUS_OFF = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.	TWAI_INT

Cont'd on next page

Table 38.5-1 – cont'd from previous page

Internal Interrupt Source	Trigger Condition	Interrupt Signal
TWAI_DATA_OVERRUN_INT (DOI)	The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages. The DOI is only triggered by the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI could be triggered again when all received messages (valid or overrun) have been cleared.	TWAI_INT
TWAI_TS_COUNTER_OVFL_INT	The Timestamp Overflow Interrupt is triggered whenever the 32-bit counter of the TWAI controller timestamp has overflowed.	TWAI_INT
TWAI_ERR_PASSIVE_INT (EPI)	The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller switches from Error Active to Error Passive, or vice versa.	TWAI_INT
TWAI_ARBITRATION_LOST_INT (ALI)	The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB_LOST_CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record the new bit location until it is cleared (via CPU reading this register).	TWAI_INT
TWAI_BUS_ERR_INT (BEI)	The Bus Error Interrupt (BEI) is triggered whenever the TWAI controller detects an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).	TWAI_INT
TWAI_IDLE_INT (BISI)	The Bus Idle Status Interrupt (BISI) is triggered when the number of clock cycles of the TWAI controller in the idle status exceeds the pre-configured value in the TWAI_IDLE_INTR_CNT_REG register. Users can configure this interrupt to get the TWAI controller idle status and further decide whether to turn off the external TWAI receiver to reduce the overall power consumption (see Section 38.4.9).	TWAI_INT

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INTERRUPT_REG, and de-asserted when all bits in TWAI_INTERRUPT_REG are cleared. The majority of interrupt bits in TWAI_INTERRUPT_REG are automatically cleared when the register is read, except for the Receive Interrupt which can only be cleared when all the messages are released by setting the TWAI_RELEASE_BUFFER bit.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 38.6 *Register Summary*.

38.6 Register Summary

'|' here means a separate line to distinguish between TWAI working modes discussed in Section 38.4.1 *Modes*. The left describes the access in Operation mode. The right belongs to Reset mode and is marked in red. The addresses in this section are relative to Two-wire Automotive Interface base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_CMD_REG	Command Register	0x0004	WO
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	R/W
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	R/W
TWAI_SW_STANDBY_CFG_REG	Software Standby Register	0x0080	R/W
TWAI_HW_CFG_REG	Hardware Standby Register	0x0084	R/W
TWAI_HW_STANDBY_CNT_REG	Standby Time Length Register	0x0088	R/W
TWAI_IDLE_INTR_CNT_REG	Idle Status Time Length Register	0x008C	R/W
Status Registers			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB_LOST_CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	R/W
TWAI_RX_MESSAGE_COUNTER_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INTERRUPT_REG	Interrupt Mask Register	0x000C	RO
TWAI_INTERRUPT_ENABLE_REG	Interrupt Enable Register	0x0010	varies
Data Registers			
TWAI_DATA_0_REG	Data Register 0	0x0040	R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	R/W | R/O
TWAI_DATA_9_REG	Data Register 9	0x0064	R/W | R/O
TWAI_DATA_10_REG	Data Register 10	0x0068	R/W | R/O
TWAI_DATA_11_REG	Data Register 11	0x006C	R/W | R/O
TWAI_DATA_12_REG	Data Register 12	0x0070	R/W | R/O

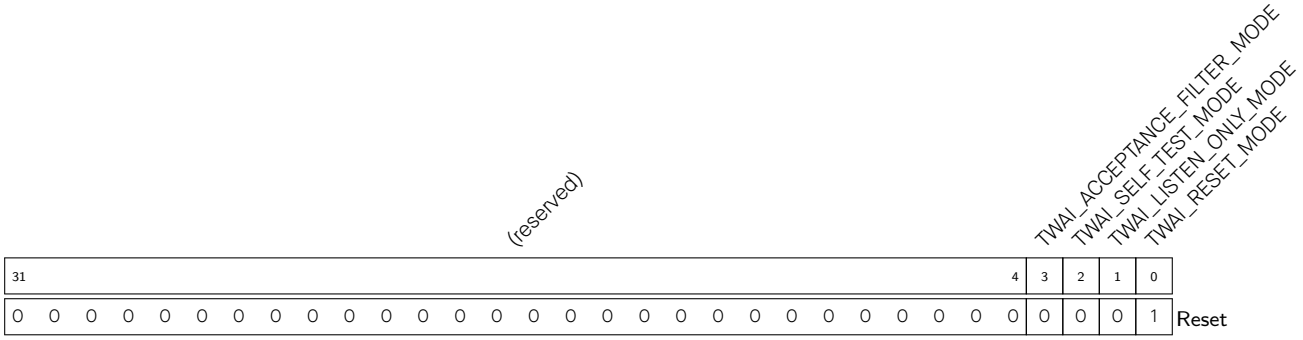
Name	Description	Address	Access
Timestamp Register			
TWAI_TIMESTAMP_DATA_REG	Timestamp Data Register	0x0094	RO
TWAI_TIMESTAMP_PRESCALER_REG	Timestamp Configuration Register 0	0x0098	R/W
TWAI_TIMESTAMP_CFG_REG	Timestamp Configuration Register 1	0x009C	R/W

38.7 Registers

'|' here means a separate line. The left describes the access in Operation mode. The right belongs to Reset mode with red color. The addresses in this section are relative to Two-wire Automotive Interface base address (each TWAI 0 and TWAI 1 has an individual base address) provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 38.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE Configures the Operation mode of the TWAI Controller.

0: Operation mode

1: Reset mode

(R/W)

TWAI_LISTEN_ONLY_MODE Configures whether to enter the Listen-only mode.

0: No effect

1: Listen-only mode. In this mode, the nodes will only receive messages from the bus, without generating the acknowledge signal or updating the RX error counter.

(R/W)

TWAI_SELF_TEST_MODE Configures whether to enter the Self-test mode.

0: No effect

1: Enter the Self-test mode. In this mode, the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self-reception request command.

(R/W)

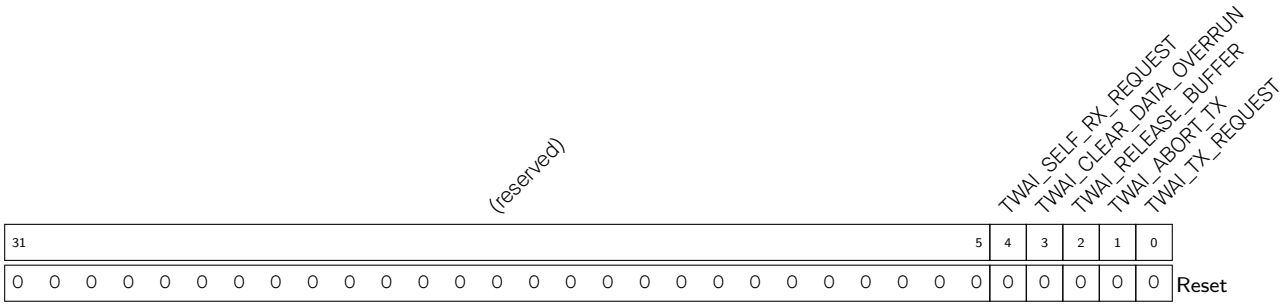
TWAI_ACCEPTANCE_FILTER_MODE Configures the filter mode.

0: Dual-filter mode

1: Single-filter mode

(R/W)

Register 38.2. TWAI_CMD_REG (0x0004)



TWAI_TX_REQUEST Configures whether to drive nodes to start transmission.

0: No effect

1: Drive nodes to start transmission

(WO)

TWAI_ABORT_TX Configures whether to cancel a pending transmission request.

0: No effect

1: Cancel a pending transmission request

(WO)

TWAI_RELEASE_BUFFER Configures whether to release the RX buffer.

0: No effect

1: Release the RX buffer

(WO)

TWAI_CLEAR_DATA_OVERRUN Configures whether to clear the data overrun status bit.

0: No effect

1: Clear the data overrun status bit

(WO)

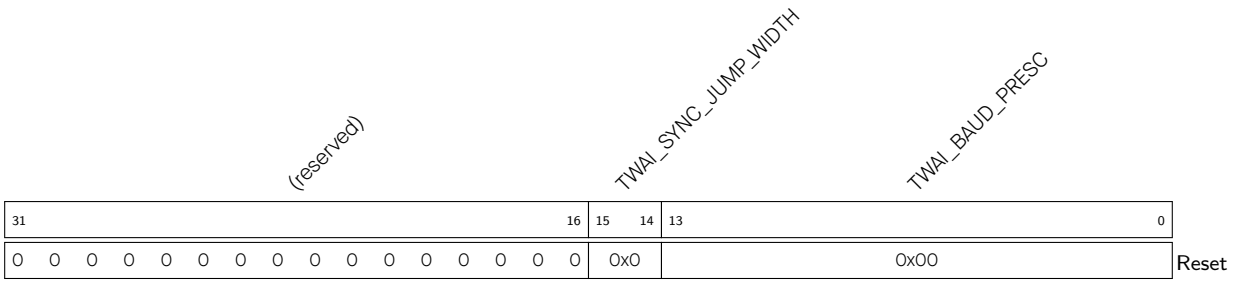
TWAI_SELF_RX_REQUEST Configures whether to drive nodes to start transmitting messages while also receiving messages on the bus simultaneously.

0: No effect

1: Drive nodes to start transmitting messages while also receiving messages on the bus (set the value of TWAI_TX_REQUEST to 0, or only transmission is allowed)

(WO)

Register 38.3. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Configures baud rate prescaler value, determining the frequency dividing ratio.

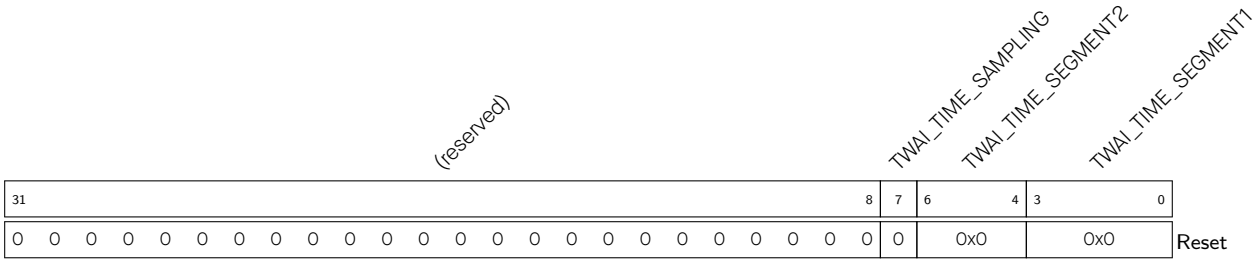
0: Low

1: High

(R/W)

TWAI_SYNC_JUMP_WIDTH Configures Synchronization Jump Width (SJW), ranging from 1 ~ 4 Tq wide. (R/W)

Register 38.4. TWAI_BUS_TIMING_1_REG (0x001C)



TWAI_TIME_SEGMENT1 Configures the width of PBS1. (R/W)

TWAI_TIME_SEGMENT2 Configures the width of PBS2. (R/W)

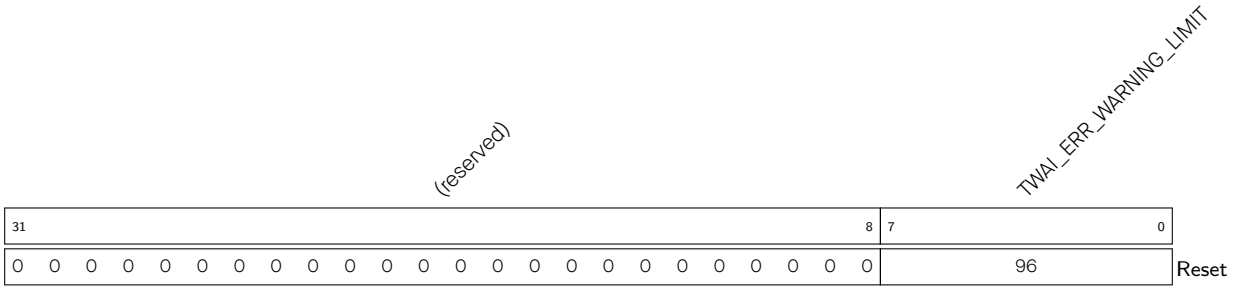
TWAI_TIME_SAMPLING Configures the number of sample points.

0: The bus is sampled once

1: The bus is sampled three times

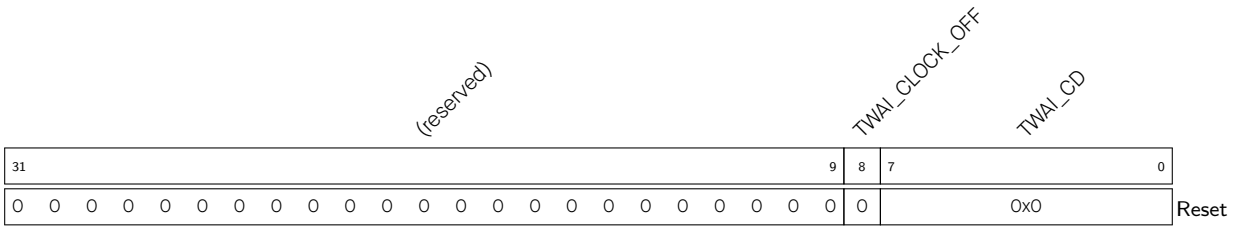
(R/W)

Register 38.5. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Configures error warning threshold. When any of the error counter values (TWAI_RX_ERR_CNT or TWAI_TX_ERR_CNT) exceeds the threshold, the TWAI controller enters an error state; when all the error counter values are below the threshold, the TWAI controller exits the error state. An error warning interrupt will be triggered if the TWAI controller enters or exits an error state. (R/W)

Register 38.6. TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD Configures the divisor of the external CLKOUT pin. (R/W)

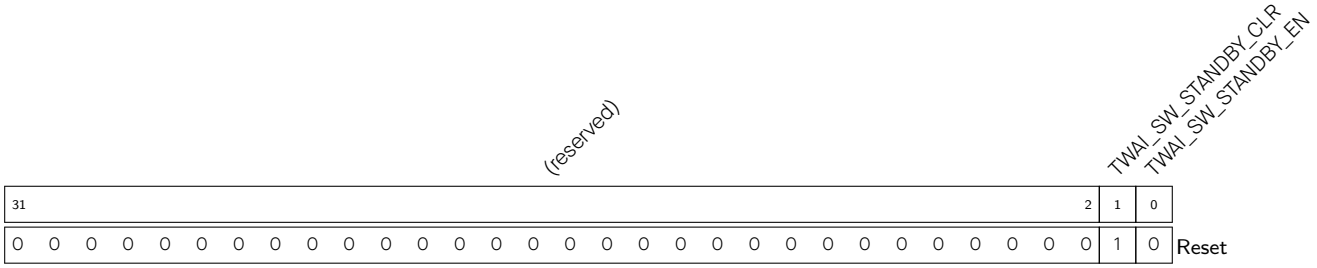
TWAI_CLOCK_OFF Configures whether or not to enable the external CLKOUT pin in Reset mode.

0: Enable the external CLKOUT pin

1: Disable the external CLKOUT pin

(R/W)

Register 38.7. TWAI_SW_STANDBY_CFG_REG (0x0080)



TWAI_SW_STANDBY_EN Configures whether to set standby signals with software.

0: No effect

1: Set standby signals

(R/W)

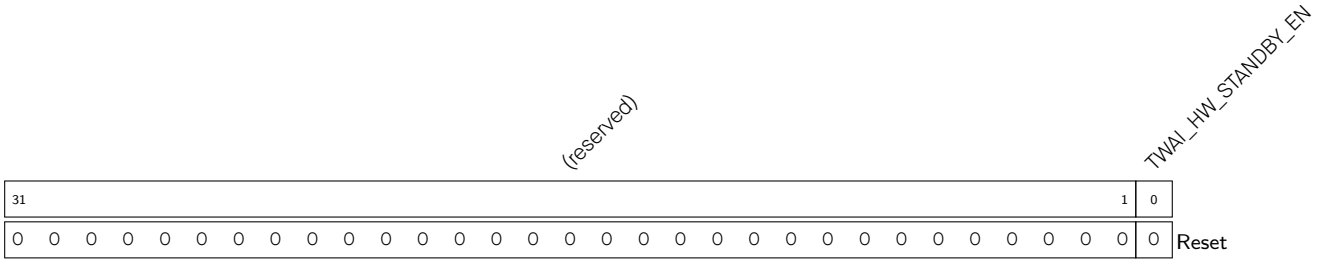
TWAI_SW_STANDBY_CLR Configures whether to clear standby signals with software.

0: No effect

1: Clear standby signals

(R/W)

Register 38.8. TWAI_HW_CFG_REG (0x0084)



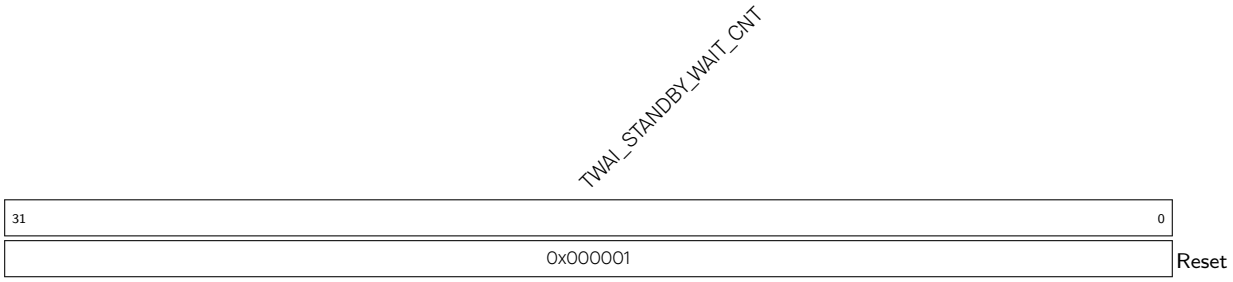
TWAI_HW_STANDBY_EN Configures whether to enable the automatic standby function for hardware.

0: No effect

1: Enable the automatic standby function for hardware

(R/W)

Register 38.9. TWAI_HW_STANDBY_CNT_REG (0x0088)

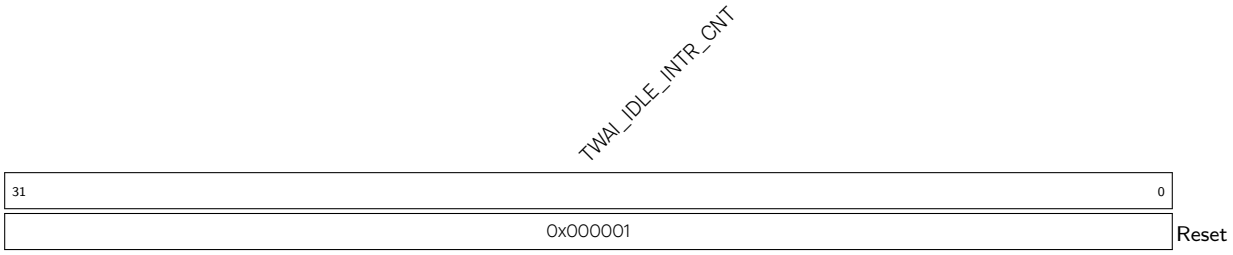


TWAI_STANDBY_WAIT_CNT Configures the time required before hardware triggers the standby signal after entering idle status.

Measurement unit: TWAI controller clock cycles.

(R/W)

Register 38.10. TWAI_IDLE_INTR_CNT_REG (0x008C)

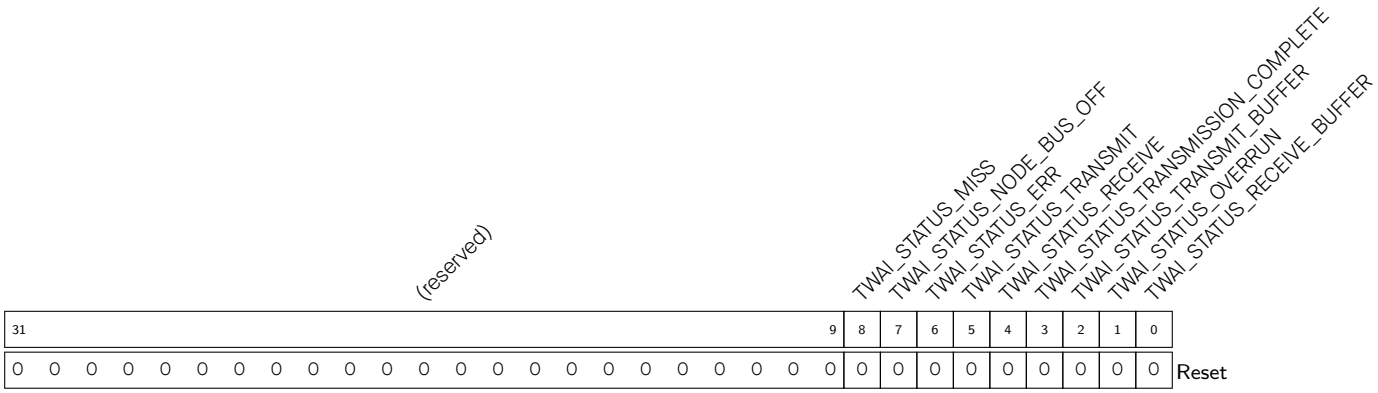


TWAI_IDLE_INTR_CNT Configures the time required before hardware generates the bus idle status interrupt signal after entering idle status.

Measurement unit: TWAI controller clock cycles.

(R/W)

Register 38.11. TWAI_STATUS_REG (0x0008)



TWAI_STATUS_RECEIVE_BUFFER Represents whether or not the RX buffer is empty.

0: Empty

1: Not empty, with at least one received data packet.

(RO)

TWAI_STATUS_OVERRUN Represents whether or not the RX FIFO is full.

0: Not full

1: Full, and data overrun has occurred

(RO)

TWAI_STATUS_TRANSMIT_BUFFER Represents whether or not the TX buffer is empty.

0: Not empty

1: Empty, and the CPU may write a message into it

(RO)

TWAI_STATUS_TRANSMISSION_COMPLETE Represents whether or not the TWAI controller has sent an entire packet to the bus.

0: Not sent

1: Sent

(RO)

TWAI_STATUS_RECEIVE Represents whether or not the TWAI Controller is receiving a message from the bus.

0: Not receiving

1: Receiving

(RO)

TWAI_STATUS_TRANSMIT Represents whether or not the TWAI Controller is transmitting a message to the bus.

0: Not transmitting

1: Transmitting

(RO)

TWAI_STATUS_ERR Represents at least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

Continued on the next page...

Register 38.11. LP_UART_INT_RAW_REG (0x0004)

Continued from the previous page...

TWAI_STATUS_NODE_BUS_OFF Represents whether or not the TWAI Controller involves in bus activities in bus-off status.

0: Involved

1: No longer involved

(RO)

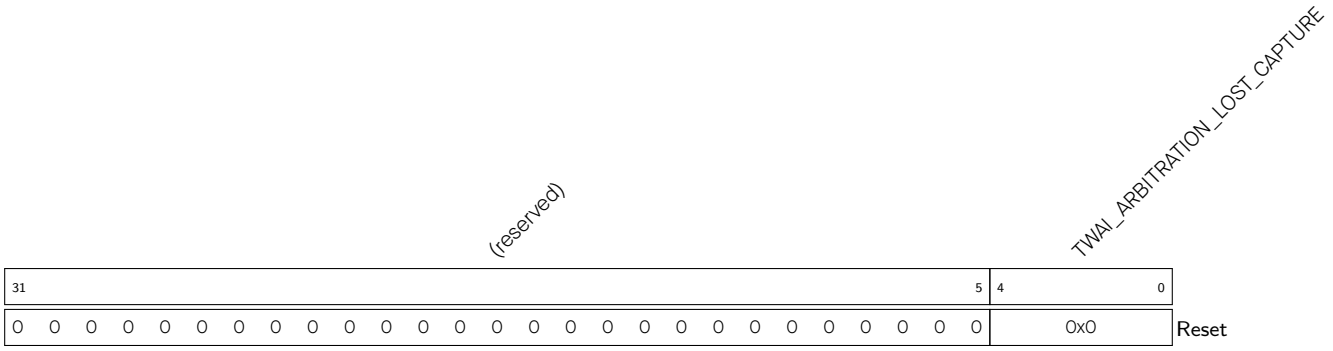
TWAI_STATUS_MISS Represents whether or not the data packet in the RX FIFO is complete.

0: The current packet is complete

1: The current packet is missing

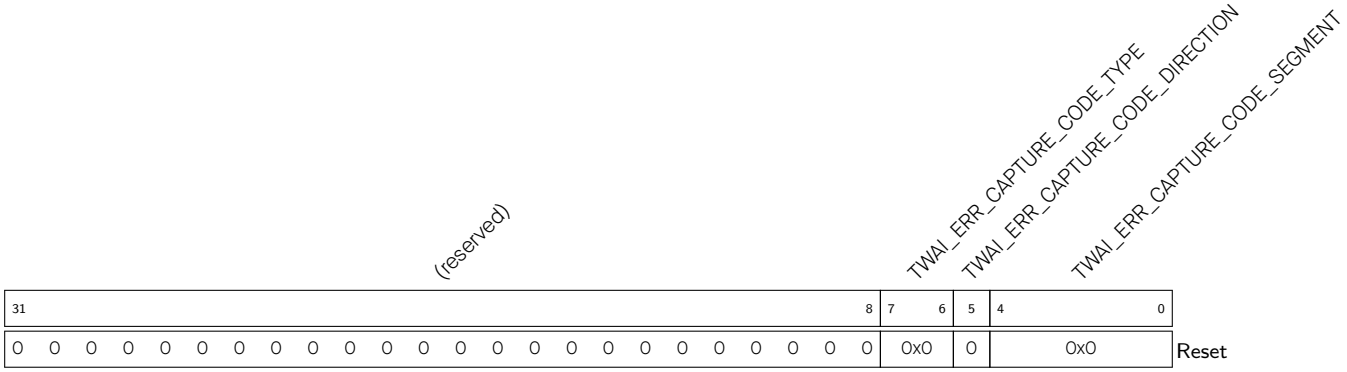
(RO)

Register 38.12. TWAI_ARB_LOST_CAP_REG (0x002C)



TWAI_ARBITRATION_LOST_CAPTURE Represents the bit position of lost arbitration. (RO)

Register 38.13. TWAI_ERR_CODE_CAP_REG (0x0030)



TWAI_ERR_CAPTURE_CODE_SEGMENT Represents the location of errors, see Table 38.4-11 for details. (RO)

TWAI_ERR_CAPTURE_CODE_DIRECTION Represents transmission direction of the node when an error occurs.

0: Error occurs when transmitting a message

1: Error occurs when receiving a message

(RO)

TWAI_ERR_CAPTURE_CODE_TYPE Represents error types.

0: Bit error

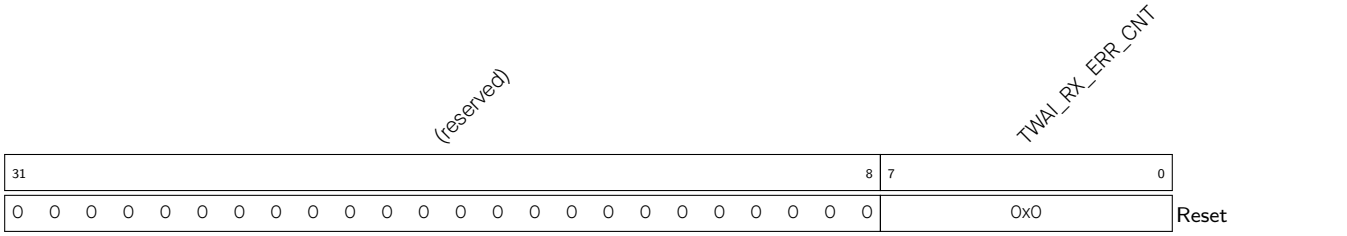
1: Form error

2: Stuff error

3: Others

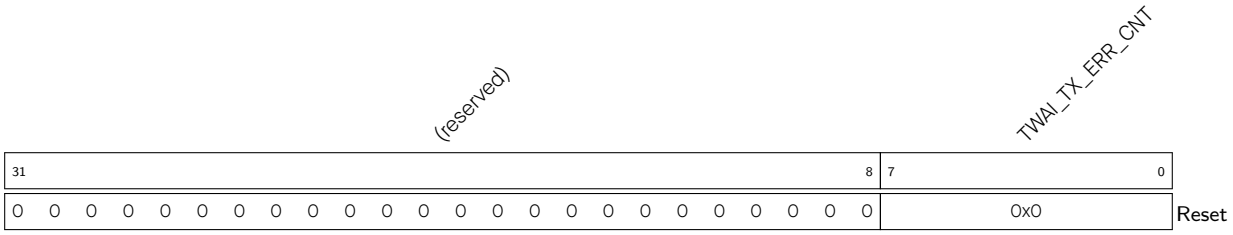
(RO)

Register 38.14. TWAI_RX_ERR_CNT_REG (0x0038)



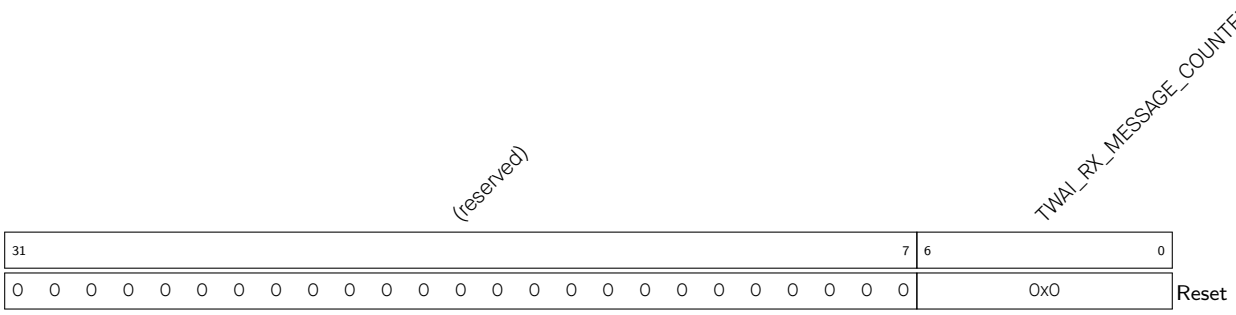
TWAI_RX_ERR_CNT The RX error counter register, which reflects value changes in reception status. (R/W)

Register 38.15. TWAI_TX_ERR_CNT_REG (0x003C)



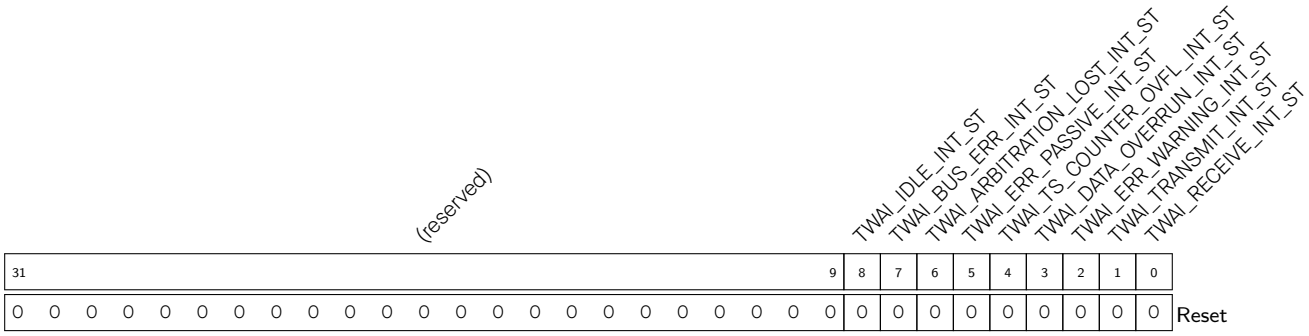
TWAI_TX_ERR_CNT The TX error counter register, which reflects value changes in transmission status. (R/W)

Register 38.16. TWAI_RX_MESSAGE_COUNTER_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER Represents the number of messages available within the RX FIFO. (RO)

Register 38.17. TWAI_INTERRUPT_REG (0x000C)



TWAI_RECEIVE_INT_ST The masked interrupt status of TWAI_RX_INT. (RO)

TWAI_TRANSMIT_INT_ST The masked interrupt status of TWAI_TX_INT. (RO)

TWAI_ERR_WARNING_INT_ST The masked interrupt status of TWAI_ERR_WARN_INT. (RO)

TWAI_DATA_OVERRUN_INT_ST The masked interrupt status of TWAI_OVERRUN_INT. (RO)

TWAI_TS_COUNTER_OVFL_INT_ST The masked interrupt status of TWAI_TS_COUNTER_OVFL_INT. (RO)

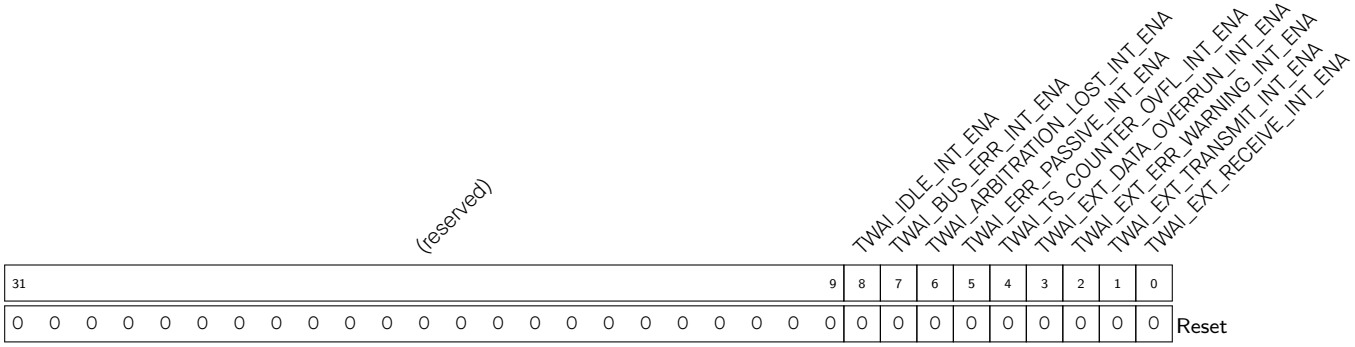
TWAI_ERR_PASSIVE_INT_ST The masked interrupt status of TWAI_ERR_PASSIVE_INT. (RO)

TWAI_ARBITRATION_LOST_INT_ST The masked interrupt status of TWAI_ARB_LOST_INT. (RO)

TWAI_BUS_ERR_INT_ST The masked interrupt status of TWAI_BUS_ERR_INT. (RO)

TWAI_IDLE_INT_ST The masked interrupt status of TWAI_BUS_STATE_INT. (RO)

Register 38.18. TWAI_INTERRUPT_ENABLE_REG (0x0010)



TWAI_EXT_RECEIVE_INT_ENA Write 1 to enable the TWAI_RX_INT interrupt. (R/W)

TWAI_EXT_TRANSMIT_INT_ENA Write 1 to enable the TWAI_TX_INT interrupt. (R/W)

TWAI_EXT_ERR_WARNING_INT_ENA Write 1 to enable the TWAI_ERR_WARN_INT interrupt. (R/W)

TWAI_EXT_DATA_OVERRUN_INT_ENA Write 1 to enable the TWAI_OVERRUN_INT interrupt. (R/W)

TWAI_TS_COUNTER_OVFL_INT_ENA Write 1 to enable the TWAI_TS_COUNTER_OVFL_INT interrupt. (R/W)

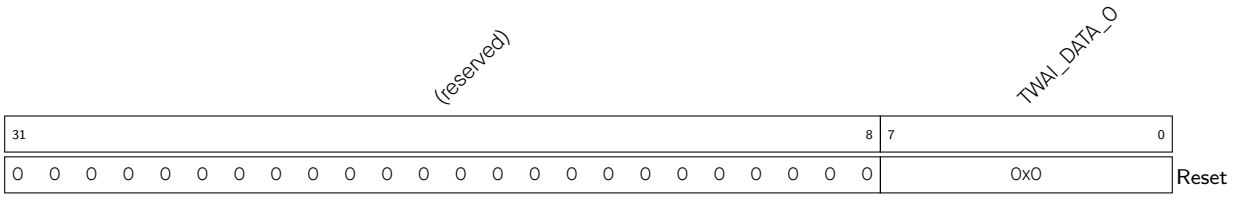
TWAI_ERR_PASSIVE_INT_ENA Write 1 to enable the TWAI_ERR_PASSIVE_INT interrupt. (R/W)

TWAI_ARBITRATION_LOST_INT_ENA Write 1 to enable the TWAI_ARB_LOST_INT interrupt. (R/W)

TWAI_BUS_ERR_INT_ENA Write 1 to enable the TWAI_BUS_ERR_INT interrupt. (R/W)

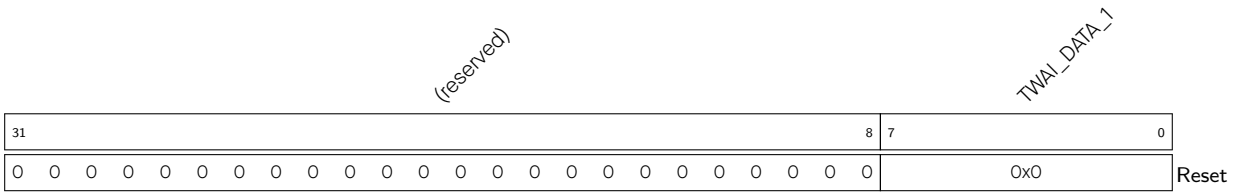
TWAI_IDLE_INT_ENA Write 1 to enable the TWAI_BUS_STATE_INT interrupt. (RO)

Register 38.19. TWAI_DATA_0_REG (0x0040)



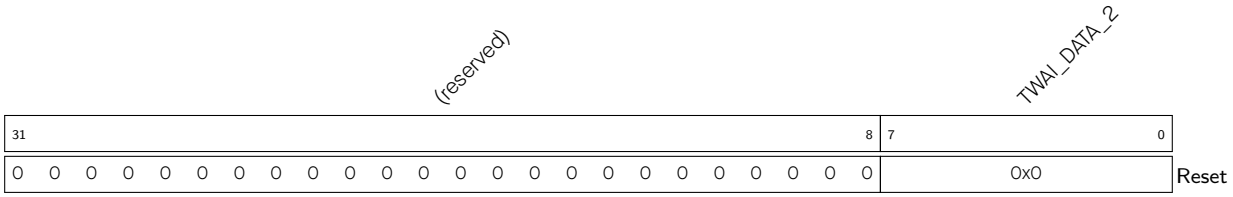
TWAI_DATA_0 In Operation mode, configures the 0th byte information of the data to be transmitted, or reads the 0th byte information of the data received. In Reset mode, configures the 1th byte of the filter code. (R/W)

Register 38.20. TWAI_DATA_1_REG (0x0044)



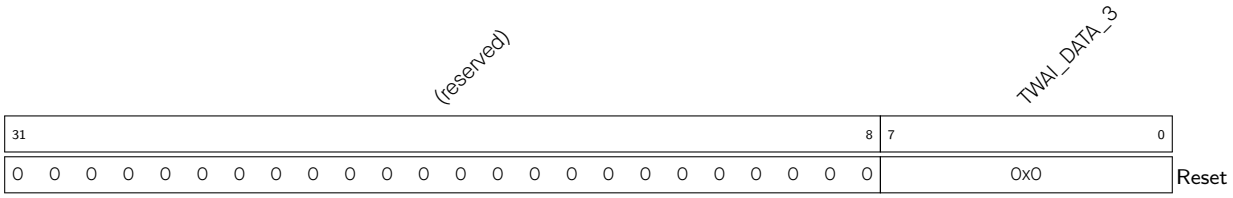
TWAI_DATA_1 In Operation mode, configures the 1st byte information of the data to be transmitted, or reads the 1st byte information of the data received. In Reset mode, configures the 2nd byte of the filter code. (R/W)

Register 38.21. TWAI_DATA_2_REG (0x0048)



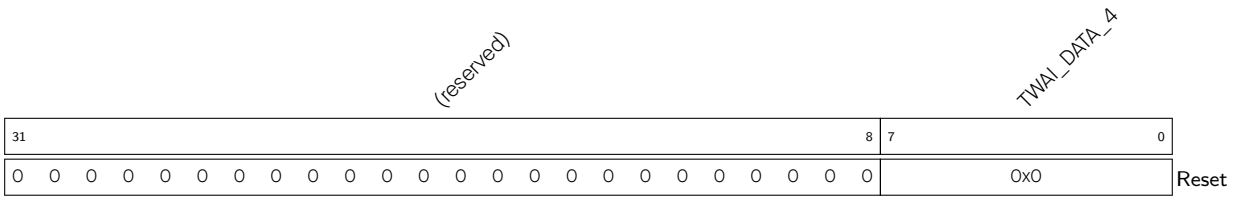
TWAI_DATA_2 In Operation mode, configures the 2nd byte information of the data to be transmitted, or reads the 2nd byte information of the data received. In Reset mode, configures the 3rd byte of the filter code. (R/W)

Register 38.22. TWAI_DATA_3_REG (0x004C)



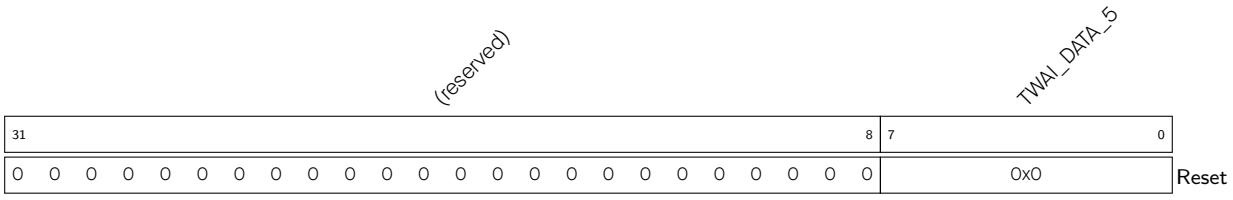
TWAI_DATA_3 In Operation mode, configures the 3rd byte information of the data to be transmitted, or reads the 3rd byte information of the data received. In Reset mode, configures the 4th byte of the filter code. (R/W)

Register 38.23. TWAI_DATA_4_REG (0x0050)



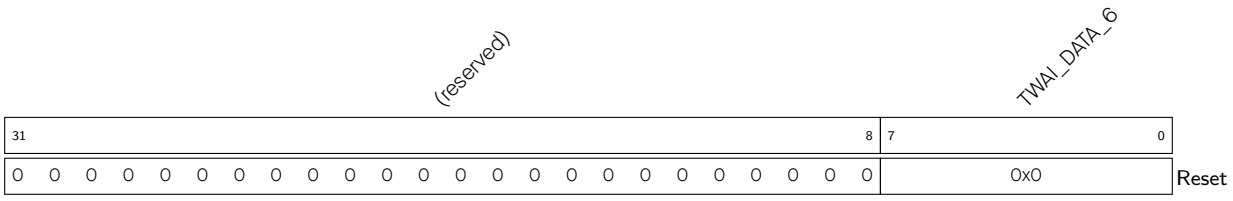
TWAI_DATA_4 In Operation mode, configures the 4th byte information of the data to be transmitted, or reads the 4th byte information of the data received. In Reset mode, configures the 1st byte of the masked filter code. (R/W)

Register 38.24. TWAI_DATA_5_REG (0x0054)



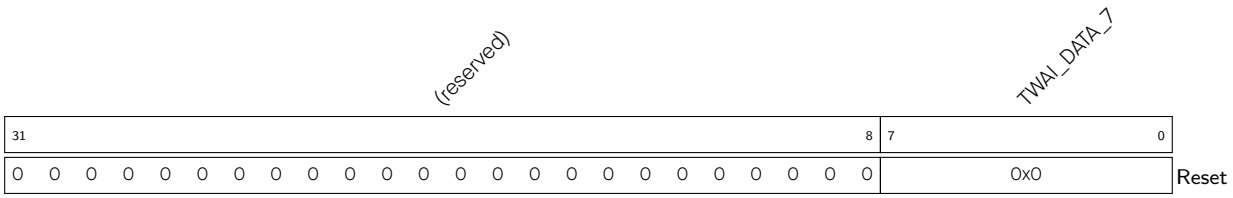
TWAI_DATA_5 In Operation mode, configures the 5th byte information of the data to be transmitted, or reads the 5th byte information of the data received. In Reset mode, configures the 2nd byte of the masked filter code. (R/W)

Register 38.25. TWAI_DATA_6_REG (0x0058)



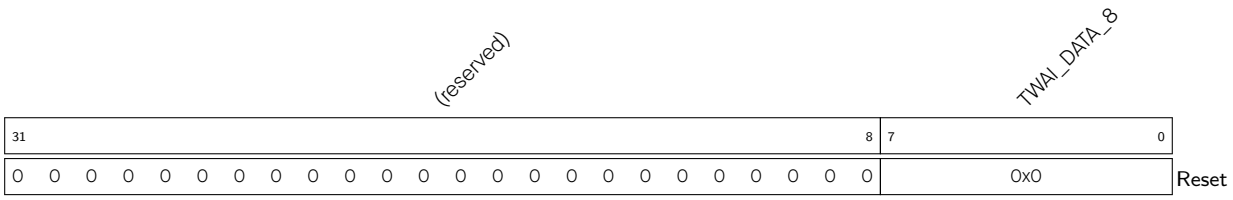
TWAI_DATA_6 In Operation mode, configures the 6th byte information of the data to be transmitted, or reads the 6th byte information of the data received. In Reset mode, configures the 3rd byte of the masked filter code. (R/W)

Register 38.26. TWAI_DATA_7_REG (0x005C)



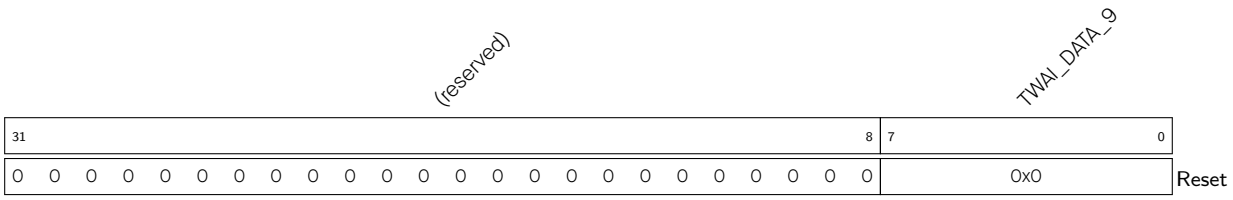
TWAI_DATA_7 In Operation mode, configures the 7th byte information of the data to be transmitted, or reads the 7th byte information of the data received. In Reset mode, configures the 4th byte of the masked filter code. (R/W)

Register 38.27. TWAI_DATA_8_REG (0x0060)



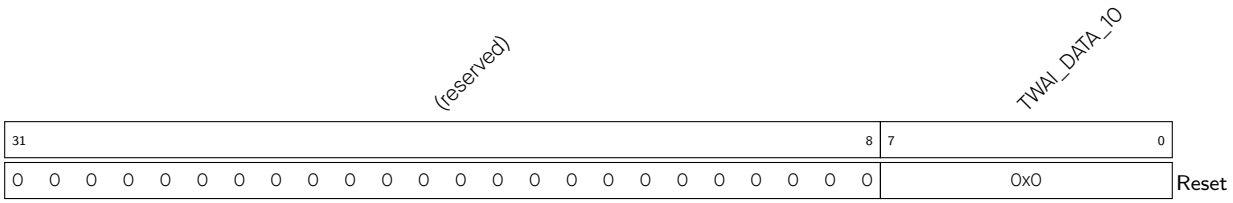
TWAI_DATA_8 In Operation mode, configures the 8th byte information of the data to be transmitted, or reads the 8th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 38.28. TWAI_DATA_9_REG (0x0064)



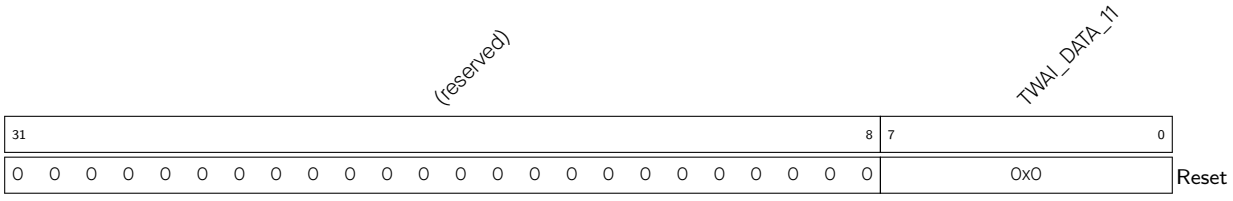
TWAI_DATA_9 In Operation mode, configures the 9th byte information of the data to be transmitted, or reads the 9th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 38.29. TWAI_DATA_10_REG (0x0068)



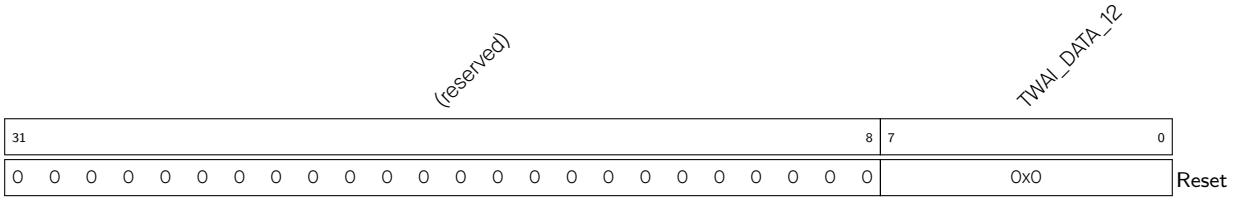
TWAI_DATA_10 In Operation mode, configures the 10th byte information of the data to be transmitted, or reads the 10th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 38.30. TWAI_DATA_11_REG (0x006C)



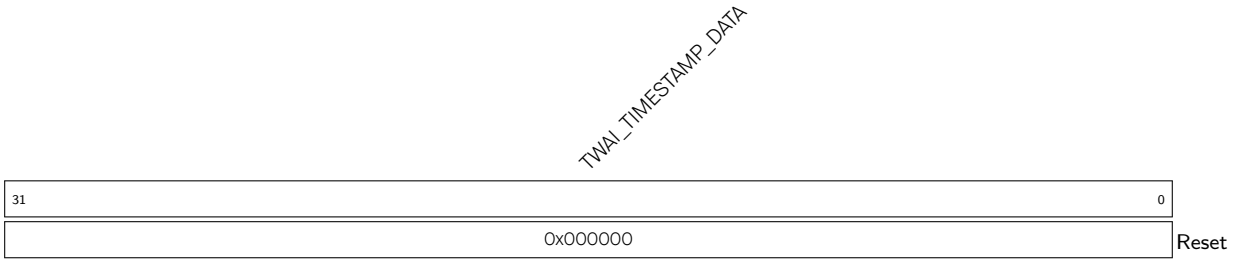
TWAI_DATA_11 In Operation mode, configures the 11th byte information of the data to be transmitted, or reads the 11th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 38.31. TWAI_DATA_12_REG (0x0070)



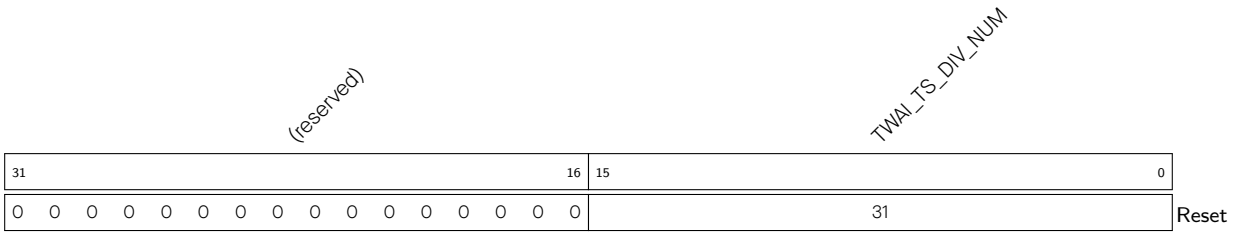
TWAI_DATA_12 In Operation mode, configures the 12th byte information of the data to be transmitted, or reads the 12th byte information of the data received. (R/W)
In Reset mode, reserved. (RO)

Register 38.32. TWAI_TIMESTAMP_DATA_REG (0x0094)



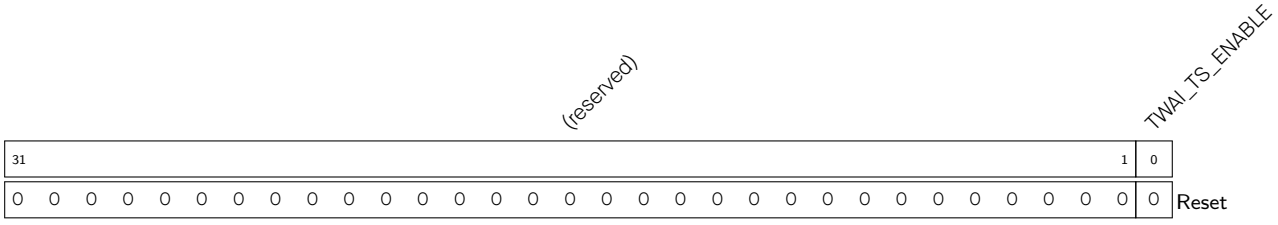
TWAI_TIMESTAMP_DATA The timestamp of the TWAI frame indicated by the RX FIFO read pointer corresponds to the time when the current TWAI frame was received. Valid only when TWAI_TS_ENABLE is enabled. (RO)

Register 38.33. TWAI_TIMESTAMP_PRESCALER_REG (0x0098)



TWAI_TS_DIV_NUM Configures the clock divisor of the timestamp counter. (R/W)

Register 38.34. TWAI_TIMESTAMP_CFG_REG (0x009C)



TWAI_TS_ENABLE Configures whether to enable the timestamp collection function.

0: Disable

1: Enable

(R/W)

Chapter 39

SD/MMC Host Controller (SDHOST)

39.1 Overview

The ESP32-P4 memory card interface controller provides a hardware interface between the Advanced Peripheral Bus (APB) and an external memory device. The memory card interface allows the ESP32-P4 to be connected to SDIO (secure digital I/O) memory cards, MMC (multimedia cards) and devices with a CE-ATA (Consumer Electronics Advanced Transport Architecture) interface. It supports two external cards (Card0 and Card1). All SD/MMC module interface signals only connect to GPIO pins via GPIO matrix.

39.2 Features

This module supports the following features:

- Two external cards
- SD memory Card specification V3.0 and V3.01
- MMC: V4.41, V4.5, and V4.51
- CE-ATA: V1.1
- 1-bit, 4-bit, and 8-bit modes

The SD/MMC controller topology is shown in Figure 39.2-1. The controller supports two peripherals, but they cannot function at the same time.

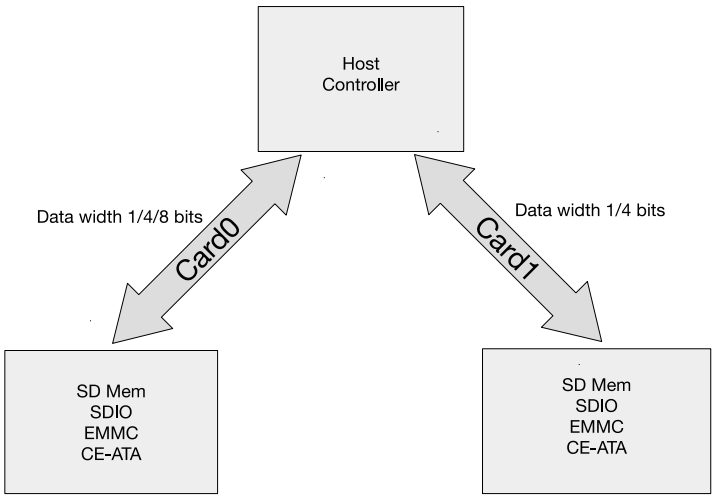


Figure 39.2-1. SD/MMC Controller Topology

39.3 SD/MMC External Interface Signals

The primary external interface signals, which enable the SD/MMC controller to communicate with an external device, are clock (sdhost_cclk_out_1.eg:card1), command (sdhost_ccmd_out_1), and data signals (sdhost_cdata_in_1[7:0]/sdhost_cdata_out_1[7:0]). Additional signals include the card interrupt, card detect, and write protect signals. The direction of each signal is shown in Figure 39.3-1. The direction and description of each pin are listed in Table 39.3-1.

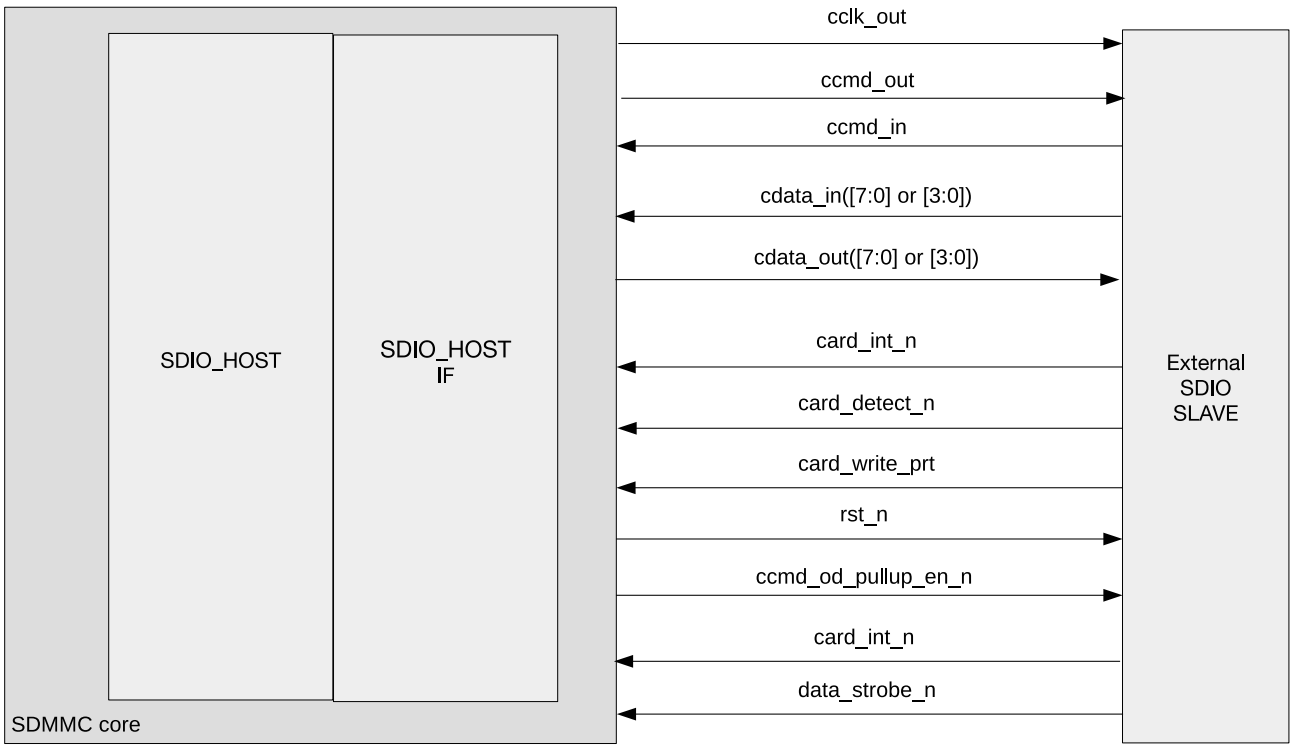


Figure 39.3-1. SD/MMC Controller External Interface Signals

Table 39.3-1. SD/MMC Signal Description

Signal	Direction	Description
sdhost_cclk_out	Output	Clock signals for slave device
sdhost_ccmd	Duplex	Duplex command/response lines
sdhost_cdata	Duplex	Duplex data read/write lines
sdhost_card_detect_n	Input	Card detection input line
sdhost_card_write_prt	Input	Card write protection status input

39.4 Functional Description

39.4.1 SD/MMC Host Controller Architecture

The SD/MMC host controller consists of two main functional blocks, as shown in Figure 39.4-1:

- Bus Interface Unit (BIU): Provides the APB interface for registers, access to RAM data, and DMA data read and write operations.

- Card Interface Unit (CIU): Handles external memory card interface protocols and provides clock control.

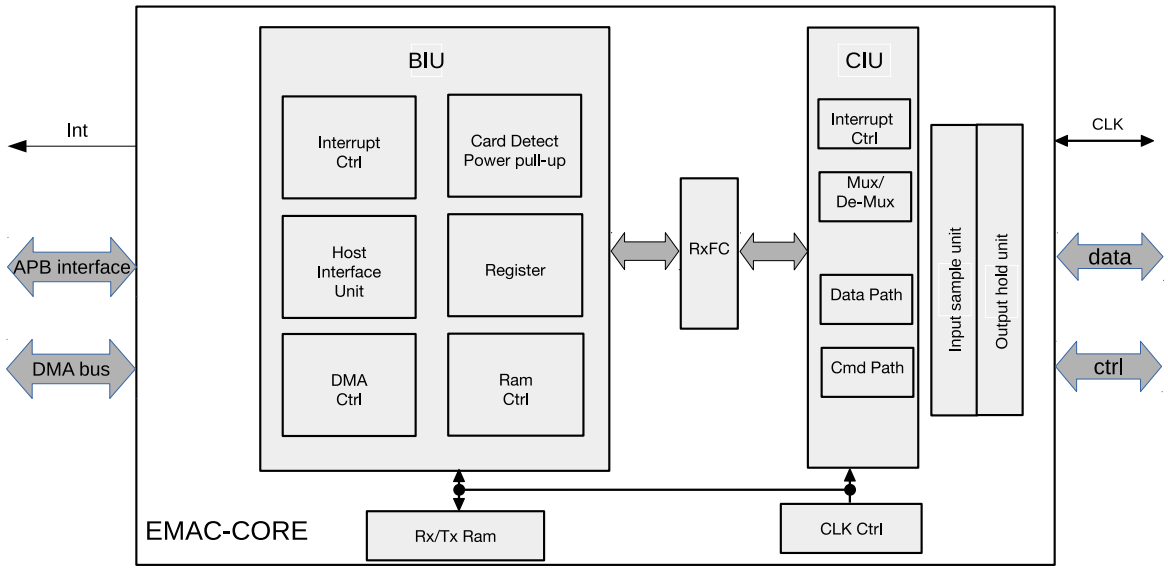


Figure 39.4-1. SDIO Host Block Diagram

39.4.1.1 Bus Interface Unit (BIU)

The BIU provides access to registers and RAM data through the Host Interface Unit (HIU). Additionally, it provides a method to access to memory data through a DMA interface. Figure 39.4-1 illustrates the internal components of the BIU.

The BIU provides the following functions:

- Host interface
- DMA interface
- Interrupt control
- Register access
- FIFO access
- Power-up/pull-up control and card detection

39.4.1.2 Card Interface Unit (CIU)

The CIU module implements the dedicated card slot specifications. Within the CIU, the command path control unit and data path control unit are used to interface with the command and data ports, respectively, of the SD/MMC/CE-ATA cards. The CIU also provides clock control. Figure 39.4-1 illustrates the internal structure of the CIU, which consists of the following primary functional blocks:

- Command path
- Data path
- SDIO interrupt control
- Clock control
- MUX (multiplexer) and De-MUX (de-multiplexer) unit

39.4.2 Command Path

The command path performs the following functions:

- Configures clock parameters
- Configures card command parameters
- Sends commands to card bus (sdhost_ccmd_out line)
- Receives responses from card bus (sdhost_ccmd_in line)
- Sends responses to BIU
- Drives the P-bit on the command line

The command path state machine is shown in Figure 39.4-2.

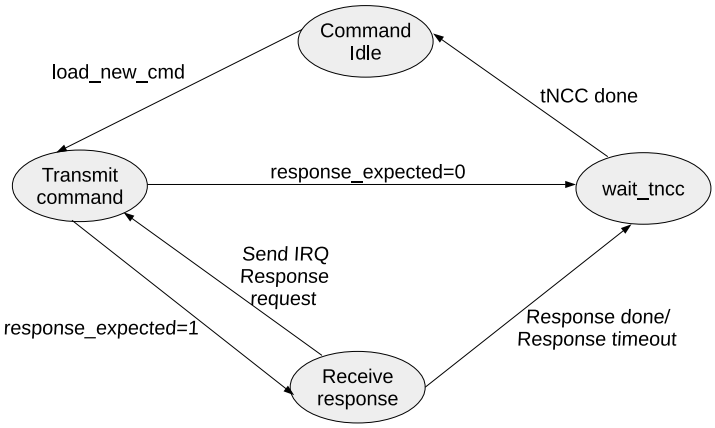


Figure 39.4-2. Command Path State Machine

39.4.3 Data Path

The data path block pops RAM data and transmits them on sdhost_cdata_out during a write-data transfer, or it receives data on sdhost_cdata_in and pushes them into RAM during a read-data transfer. The data path loads new data parameters, i.e., expected data, read/write data transfer, stream/block transfer, block size, byte count, card type, timeout registers, etc., whenever a data transfer command is not in progress.

If the SDHOST_DATA_EXPECTED bit is set in the SDHOST_CMD_REG register, the new command is a data transfer command and the data path starts one of the following operations:

- Transmitting data if the SDHOST_READ_WRITE bit is 1
- Receiving data if the SDHOST_READ_WRITE bit is 0

39.4.3.1 Data Transmit

The module starts data transmission two clock cycles after a response for the data write command is received. This occurs even if the command path detects a response error or a cyclic redundancy check (CRC) error in a response. If no response is received from the card until the response timeout, no data will be transmitted. Depending on the value of the SDHOST_TRANSFER_MODE bit in the SDHOST_CMD_REG register, the data transmit state machine adds data to the card's data bus in a stream or in block(s). The data transmit state machine is shown in Figure 39.4-3.

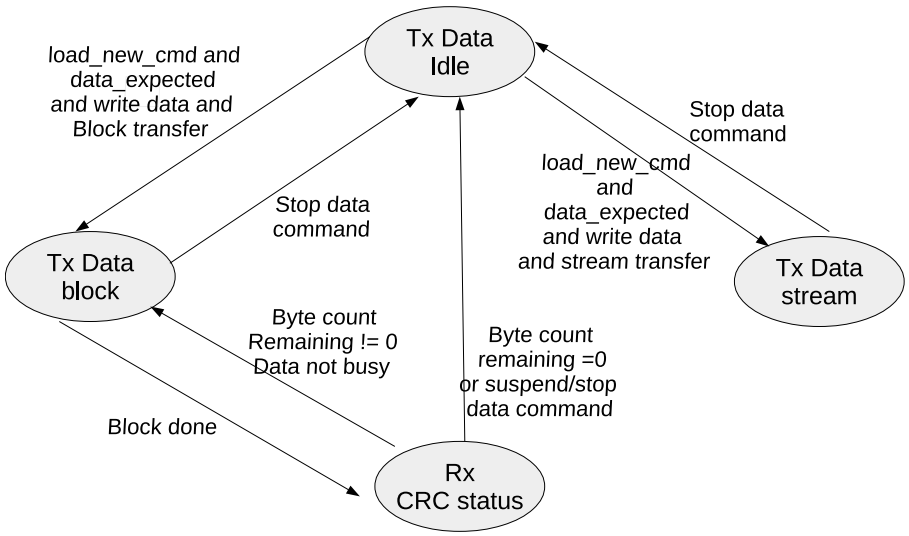


Figure 39.4-3. Data Transmit State Machine

39.4.3.2 Data Receive

The module receives data two clock cycles after the end bit of a data read command, even if the command path detects a response error or a CRC error. If no response is received from the card and a response timeout occurs, the BIU does not receive a signal about the completion of the data transfer. If the command sent by the CIU is an illegal operation for the card, it would prevent the card from starting a read-data transfer, and the BIU will not receive a signal about the completion of the data transfer.

If no data is received by the data timeout, the data path signals a data timeout to the BIU, which marks an end to the data transfer. Based on the value of the SDHOST_TRANSFER_MODE bit in the SDHOST_CMD_REG register, the data receive state machine gets data from the card's data bus in a stream or block(s). The data receive state machine is shown in Figure 39.4-4.

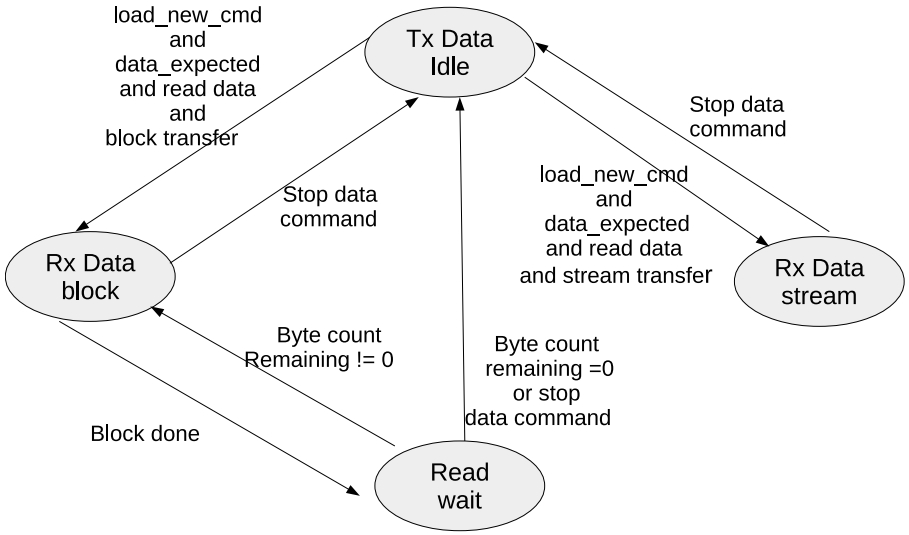


Figure 39.4-4. Data Receive State Machine

39.5 Software Restrictions for CIU Operations

- Only one card at a time can be selected to execute a command or data transfer. For example, when data are being transferred to or from a card, a new command must not be issued to another card. A new command, however, can be issued to the same card, allowing it to read the device status or stop the transfer.
- Only one data transfer command can be issued at a time.
- During an open-ended card write operation, if the card clock is stopped because RAM is empty, the software must first fill RAM with data, and start the card clock. Only then can it issue a stop/abort command to the card.
- During an SDIO/Combo card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset and then release RAM by configuring the SDHOST_FIFO_RESET bit, and then issue the resume command as if it were a new data transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52_reset) while a card data transfer is in progress, the software must set the SDHOST_STOP_ABORT_CMD bit in the SDHOST_CMD_REG register, so that the CIU can stop the data transfer after issuing the card reset command.
- When the controlling bit for end bit error is set in the SDHOST_RINTSTS_REG register, the CIU cannot control SDIO interrupts. In such a case, the software must ignore SDIO interrupts and issue a stop/abort command to the card, so that the card stops transmitting data.
- If the card clock is stopped because RAM is full during a card read, the software must read at least two RAM addresses to restart the card clock.
- Only one CE-ATA device at a time can be selected for a command or data transfer. For example, when data are transferred from a CE-ATA device, a new command must not be sent to another CE-ATA device.
- If the CE-ATA device interrupts are enabled (nIEN=0), a new SDHOST_RW_BLK command must not be sent to the same device if the execution of a SDHOST_RW_BLK command is already in progress. Only the Command Completion Signal Disable (CCSD) command can be sent while waiting for the Command Completion Signal (CCS).
- If, however, the CE-ATA device interrupts are disabled (nIEN=1), a new command can be issued to the same device, allowing it to read status information.
- Open-ended transfers are not supported for CE-ATA devices.
- The sdhost_send_auto_stop signal is not supported (software must not set the SDHOST_SEND_AUTO_STOP bit) for CE-ATA transfers.

After configuring the command start bit to 1, the values of the following registers cannot be changed before a command has been issued:

- CMD - command
- CMDARG - command argument
- BYTCNT - byte count
- BLKSIZ - block size
- CLKDIV - clock divider

- CKLENA - clock enable
- CLKSRC - clock source
- TMOUT - timeout
- CTYPE - card type

39.6 RAM for Receiving and Transmitting Data

The submodule RAM is a buffer area for transmitting and receiving data. It can be divided into two units: one is for transmitting data, and the other is for receiving data. The process of transmitting and receiving data can be achieved by the CPU and DMA for reading and writing. The latter method is described in detail in Section 39.8.

39.6.1 TX RAM Module

There are two ways to enable a write operation: CPU and DMA write.

For CPU write operations to TX RAM, write the data directly into the SDHOST_BUFFIFO_REG register via the APB interface. The corresponding FIFO width is 32 bits and the depth is 512.

For DMA write operations to TX RAM, set up DMA linked list descriptors as described in Section 39.8.

39.6.2 RX RAM Module

There are two ways to enable a read operation: CPU and DMA read.

For CPU read operations from RX RAM, read the data directly from the SDHOST_BUFFIFO_REG register via the APB interface. The corresponding FIFO width is 32 bits and the depth is 512.

For DMA read operations from RX RAM, set up DMA linked list descriptors as described in Section 39.8.

39.7 DMA Linked List

Each unit in the linked list structure consists of two parts: a descriptor and a data buffer. In other words, each descriptor points to a unique data buffer and to the next descriptor in the linked list. Figure 39.7-1 shows the linked list.

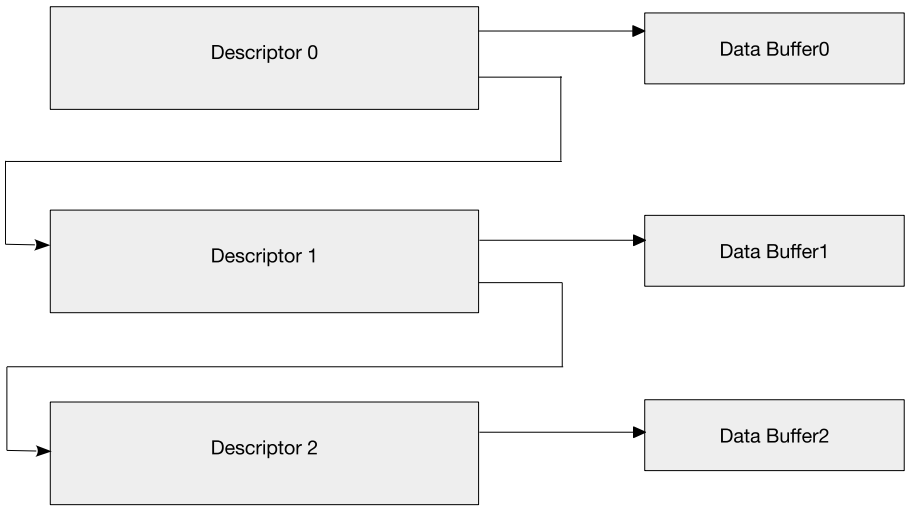


Figure 39.7-1. Linked List Structure

39.8 DMA Descriptor Format

Each descriptor consists of four words as shown in 39.8-1. Table 39.8-1, Table 39.8-2, Table 39.8-3, Table 39.8-4 provide the descriptions of these four words.

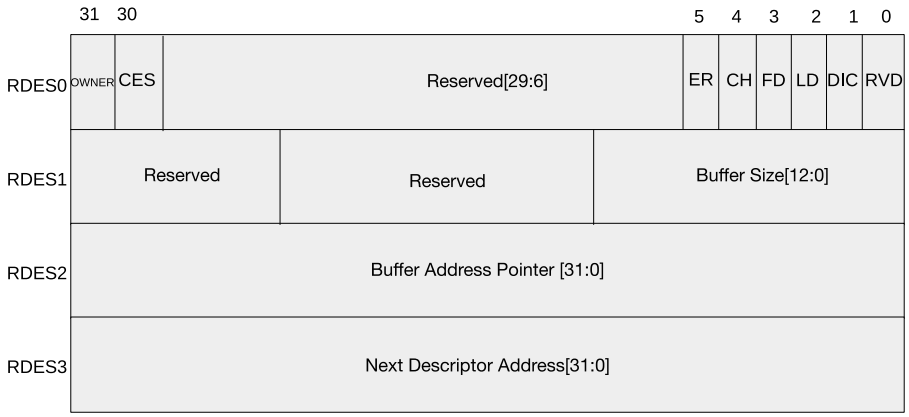


Figure 39.8-1. Descriptor Format

The DES0 field contains control and status information.

Table 39.8-1. DES0 Descriptor Field

Bits	Name	Description
31	OWNER	When set to 1, this bit indicates that the descriptor is owned by the DMA Controller. When reset to 0, it indicates that the descriptor is owned by the host. The DMA clears this bit when it completes the data transfer.

Bits	Name	Description
30	CES (Card Error Summary)	This bit indicates the status of the card's read/write operations. This bit is the logical OR of the following bits in the SDHOST_RINTSTS_REG register. • EBE: End Bit Error • RTO: Response Timeout • RCRC: Response CRC • SBE: Start Bit Error • DRTO: Data Read Timeout • DCRC: Data CRC for Receive • RE: Response Error
29:6	Reserved	Reserved
5	ER (End of Ring)	When set to 1, this bit indicates that the linked list has reached its final descriptor. The DMA Controller then returns to the base address of the linked list, creating a linked list ring.
4	CH (Second Address Chained)	When set to 1, this bit indicates that the second address in the descriptor is the next descriptor address. When this bit is set to 1, BS2 (DES1[25:13]) must be all zeros.
3	FD (First Descriptor)	When set to 1, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, the next descriptor contains the beginning of the data.
2	LD (Last Descriptor)	This bit is associated with the last data block of a DMA transfer. When set to 1, the bit indicates that the buffers pointed by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count should be 0.
1	DIC (Disable Interrupt on Completion)	When set to 1, this bit prevents the setting of the TI/RI bit of the DMA Status Register (SDHOST_IDSTS_REG) for the data that ends in the buffer pointed by this descriptor.
0	Reserved	Reserved

The DES1 field contains the buffer size.

Table 39.8-2. DES1 Descriptor Field

Bits	Name	Description
31:26	Reserved	Reserved
25:13	Reserved	Reserved

Bits	Name	Description
12:0	BS (Buffer Size)	These bits indicate the size of the data buffer byte size, which must be a multiple of four. When the buffer size is not a multiple of four, the resulting behavior is undefined. This field must not be zero.

The DES2 field contains the address pointer to the data buffer.

Table 39.8-3. DES2 Descriptor Field

Bits	Name	Description
31:0	Buffer Address Pointer	These bits indicate the address of the data buffer. The buffer address must be word-aligned.

The DES3 field contains the address pointer to the next descriptor if the present descriptor is not the last one in a linked list structure.

Table 39.8-4. DES3 Descriptor Field

Bits	Name	Description
31:0	Next Descriptor Address	If CH (DESO[4]) is set to 1, these bits contains the pointer to the next descriptor. If this is not the last descriptor in a linked list structure, bits 1 and 0 must be zero, i.e., DES3[1:0] = 0.

39.9 Programming Procedures

39.9.1 Initializing Registers

To initialize registers, perform the following steps:

1. Write to the control register SDHOST_CTRL_REG, the raw interrupt register SDHOST_RINTSTS_REG, and SDIO interrupt mask register SDHOST_INTMASK_REG to clear pending interrupts and configure controlling and interrupt parameters.
2. Write clock divider configuration register SDHOST_CLKDIV_REG, clock source selection register SDHOST_CLKSRC_REG, and clock enable register SDHOST_CLKENA_REG to configure the card clock.
3. Write other configuration registers based on card parameters. For example, configure
   - card bus width (SDHOST_CTYPE_REG)
   - user ID (SDHOST_USRID_REG)
   - UHS mode voltage and DDR (SDHOST_UHS_REG)
   - EMMC mode start bit (SDHOST_EMMCDDR_REG)
   - SDIO mode (SDHOST_CLK_EDGE_SEL_REG)
   - timeout value (SDHOST_TMOUT_REG)

- debounce filter (SDHOST_DEBNCE_REG)
- internal phase shift (SDHOST_ENSHIFT_REG)

You can reset the card with SDHOST_RST_N_REG.

39.9.2 Sending Commands

To send commands, perform the following steps:

1. Write the command argument register SDHOST_CMDARG_REG with the appropriate command argument parameter.
2. For data transfer, write the data size register SDHOST_BYTCNT_REG, the block size register SDHOST_BLKSIZ_REG register, and the card threshold control register SDHOST_CARDTHRCTL_REG to configure data transfer parameters.
3. Data transfer can be achieved by DMA or FIFO, and DMA is preferable.
   - To transmit and receive data via DMA, configure the DMA Controller according to Section 39.9.3, Section 39.9.4, and Section 39.9.5.
   - To transmit and receive data via FIFO, configure the FIFO threshold via FIFO configuration register SDHOST_FIFOTH_REG. For writing to a card, software should write data to the FIFO via SDHOST_BUFFIFO_REG continuously during command execution.
4. Configure and start the command via the SDHOST_CMD_REG.
5. Wait until the SDHOST_START_CMD bit in the SDHOST_CMD_REG register is cleared by hardware upon command reception. When hardware cannot receive the command, it will set the 12th bit HLE in the SDHOST_RINTSTS_REG register to 1.
6. Read the SDHOST_RINTSTS_REG register or enable interrupts to check command execution status. If needed, read the SDHOST_RESP*n*_REG (*n* ranges from 0 ~ 3) register to get command response from the card.
7. If data is read from the card via FIFO, software can get the data returned by the card via reading the SDHOST_BUFFIFO_REG register.

Please refer to the card specification for the command format, and configure register field values depending on the command format.

39.9.3 Initializing DMA

To initialize the DMA Controller, perform the following steps:

1. Write to the DMA bus mode register (SDHOST_BMOD_REG) to configure the host bus's access parameters.
2. Write to the DMA interrupt enable register (SDHOST_IDINTEN_REG) to mask unnecessary interrupt causes.
3. Create either a transmit or receive linked list. Then write the starting address of the first descriptor in the linked list to the DMA Controller's linked list base address register (SDHOST_DBADDR_REG).
4. The DMA Controller engine proceeds to get descriptors from the linked list.

39.9.4 Initializing DMA Transmission

To initialize DMA transmission, perform the following steps:

1. The host sets up the descriptor fields (DES0-DES3) for transmission, and sets the OWNER bit (DES0[31]) to 1. The host also prepares the data buffer.
2. The host writes the write data command to the command register SDHOST_CMD_REG.
3. The host sets the required transmit threshold via the SDHOST_TX_WMARK field in the SDHOST_FIFOTH_REG register.
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA Controller enters a suspend state and asserts the Descriptor Unable interrupt via the SDHOST_IDSTS_REG register. The host then needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. The DMA Controller waits for the Command Done (CMDD) bit in the SDHOST_RINTSTS_REG register to be set to 1 with no errors from BIU, which indicates that a transfer has completed.
6. The DMA Controller engine waits for a DMA interface request from BIU. This request is generated based on the configured transmit threshold value. For the last byte of data that cannot be accessed using a burst, a single transfer is performed on the AHB bus.
7. The DMA Controller fetches the transmit data from the data buffer in the host memory and transfers them to RAM in preparation for transmission to the card.
8. When data span across multiple descriptors, the DMA Controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit indicates whether the data span multiple descriptors or not.
9. When data transmission is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting the SDHOST_IDSTS_TI bit to 1 if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by updating the DES0 field.

39.9.5 Initializing DMA Reception

To initialize DMA reception, perform the following steps:

1. The host sets up the descriptor fields (DES0-DES3) for reception, and sets the OWNER bit (DES0[31]) to 1.
2. The host writes the read data command in the SDHOST_CMD_REG register in BIU.
3. The host sets the required receive threshold level via the SDHOST_RX_WMARK field in the SDHOST_FIFOTH_REG register.
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA enters suspend state and asserts the Descriptor Unable interrupt via the SDHOST_IDSTS_REG register. The host then needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. The DMA Controller waits for the Command Done (CMDD) bit in the SDHOST_RINTSTS_REG register to be set to 1 with no errors from BIU, which indicates that a reception has been completed.

6. The DMA Controller engine waits for a DMA interface request from BIU. This request is generated based on the configured receive threshold value. For the last byte of data that cannot be accessed using a burst, a single transfer is performed on the AHB bus.
7. The DMA Controller fetches the data from RAM and transfers them to the host memory.
8. When data span across multiple descriptors, the DMA Controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit indicates whether the data span multiple descriptors or not.
9. When data reception is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting the SDHOST_IDSTS_RI bit to 1, if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by updating the DESO field.

39.10 Clock Phase Selection

The phase of the SD/MMC Host Controller clock source is configurable, so as to adjust the setup time sequence for the input or output data signals.

The clock source of the SD/MMC Host Controller can be a high-performance clock at a high frequency or a low-power clock at a low frequency. When the clock source is a high-performance clock, the internal signal clock phase, output signal driving clock phase, and input signal sampling clock phase can be configured via the SDHOST_DLL_CLK_CONF_REG, in the unit of 1/64 clock source period. When the clock source is a low-power clock, the internal signal clock phase, output signal driving clock phase, and input signal sampling clock phase can be configured via the system clock register. The phase options are 0 degrees, 90 degrees, 180 degrees, and 270 degrees, or in other words the unit is 1/4 clock source period.

For details, please see Chapter 8 *Reset and Clock*.

39.11 Interrupt

The SD/MMC Host Controller can generate interrupt signal SDIO_HOST_INTR triggered by the following interrupt sources, and send the interrupt signals to the Interrupt Matrix.

- SDIO_INT: Interrupts from SDIO cards
- EBE: Triggered when an error in end bit occurs, or no data CRC received
- ACD: Triggered when an automatically sent command has been executed
- SBE/BCI: Triggered when an error in the start bit occurs
- HLE: Triggered when a write error occurs during hardware-lock period
- FRUN: Triggered when the FIFO is empty or full
- HTO: Triggered when the host does not fill data before timeout period
- DRTO: Triggered when a data read operation times out
- RTO: Triggered when card response times out
- DCRC: Triggered when data CRC error occurs
- RCRC: Triggered when a card response CRC error occurs

- RXDR: Triggered during read operations from card when FIFO level is greater than Receive-Threshold level
- TXDR: Triggered during write operations to card when FIFO level reaches less than or equal to Transmit-Threshold level.
- DTO: Triggered when data transfer completes
- CMDD: Triggered when a command has been executed
- RE: Triggered when response error occurs
- CD: Triggered when a card is detected
- IDSTS_RI: Triggered when data pointed by a DMA descriptor list have been received
- IDSTS_TI: Triggered when data pointed by a DMA descriptor list have been transmitted
- IDSTS_FBE: Triggered when a fatal bus error occurs and the host is stopped during data transmission or reception. When triggered, the DMA disables all bus access.
- IDSTS_DU: Triggered when a DMA descriptor list is not available because the OWNER bit is 0 (DES0 [31] = 0)

The SDHOST_RINTSTS_REG and SDHOST_IDSTS_REG register contains all the bits that might cause an interrupt. The SDHOST_INTMASK_REG and SDHOST_IDINTEN_REG register contains an enable bit for each of the events that can cause an interrupt.

There are three interrupt summary bits in the SDHOST_IDSTS_REG register: the card error interrupt summary bit (bit 5 SDHOST_IDSTS_CES), the normal interrupt summary bit (bit 8 SDHOST_IDSTS_NIS), and the abnormal interrupt summary bit (bit 9 SDHOST_IDSTS_AIS). Interrupts are cleared by writing 1 to the corresponding bit. When all the enabled interrupts within a group are cleared, the corresponding summary bit is also cleared.

Interrupts are not queued. If another interrupt event occurs before the driver has responded to the previous interrupt, no additional interrupts are generated. For example, the SDHOST_IDSTS_RI indicates that one or more data were transferred to the host buffer.

An interrupt is generated only once for simultaneous multiple events. The driver must scan the SDHOST_RINTSTS_REG and SDHOST_IDSTS_REG register for the interrupt cause.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

39.12 Register Summary

The addresses in this section are relative to the SD/MMC Host Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
Configuration registers			
SDHOST_CTRL_REG	Control register	0x0000	R/W
SDHOST_CLKDIV_REG	Clock divider configuration	0x0008	R/W
SDHOST_CLKSRC_REG	Clock source selection	0x000C	R/W
SDHOST_CLKENA_REG	Clock enable register	0x0010	R/W
SDHOST_TMOUT_REG	Data and response timeout configuration	0x0014	R/W
SDHOST_CTYPE_REG	Card bus width configuration	0x0018	R/W
SDHOST_BLKSIZ_REG	Card data block size configuration	0x001C	R/W
SDHOST_BYTCNT_REG	Data transfer length configuration	0x0020	R/W
SDHOST_CMDARG_REG	Command argument data	0x0028	R/W
SDHOST_CMD_REG	Command and boot configuration	0x002C	R/W
SDHOST_FIFOTH_REG	FIFO configuration	0x004C	R/W
SDHOST_DEBNCE_REG	Debounce filter time configuration	0x0064	R/W
SDHOST_USRID_REG	User ID (scratchpad)	0x0068	R/W
SDHOST_UHS_REG	UHS-1	0x0074	R/W
SDHOST_RST_N_REG	Card reset	0x0078	R/W
SDHOST_BMOD_REG	Burst mode transfer configuration	0x0080	R/W
SDHOST_PLDMND_REG	Poll demand configuration	0x0084	WO
SDHOST_DBADDR_REG	Linked-list base address	0x0088	R/W
SDHOST_CARDTHRCTL_REG	Card Threshold Control	0x0100	R/W
SDHOST_EMMCDDR_REG	eMMC DDR register	0x010C	R/W
SDHOST_ENSHIFT_REG	Enable Phase Shift	0x0110	R/W
SDHOST_CLK_EDGE_SEL_REG	SDIO control register	0x0800	R/W
SDHOST_DLL_CLK_CONF_REG	SDIO DLL clock control	0x0808	R/W
Interrupt registers			
SDHOST_INTMASK_REG	SDIO interrupt mask register	0x0024	R/W
SDHOST_MINTSTS_REG	Masked interrupt status register	0x0040	RO
SDHOST_RINTSTS_REG	Raw interrupt register	0x0044	R/W1C
SDHOST_IDSTS_REG	DMA interrupt register	0x008C	R/W1C
SDHOST_IDINTEN_REG	DMA interrupt enable register	0x0090	R/W
Status registers			
SDHOST_RESP0_REG	Response data	0x0030	RO
SDHOST_RESP1_REG	Long response data	0x0034	RO
SDHOST_RESP2_REG	Long response data	0x0038	RO
SDHOST_RESP3_REG	Long response data	0x003C	RO
SDHOST_STATUS_REG	SD/MMC status	0x0048	RO
SDHOST_CDETECT_REG	Card detect	0x0050	RO

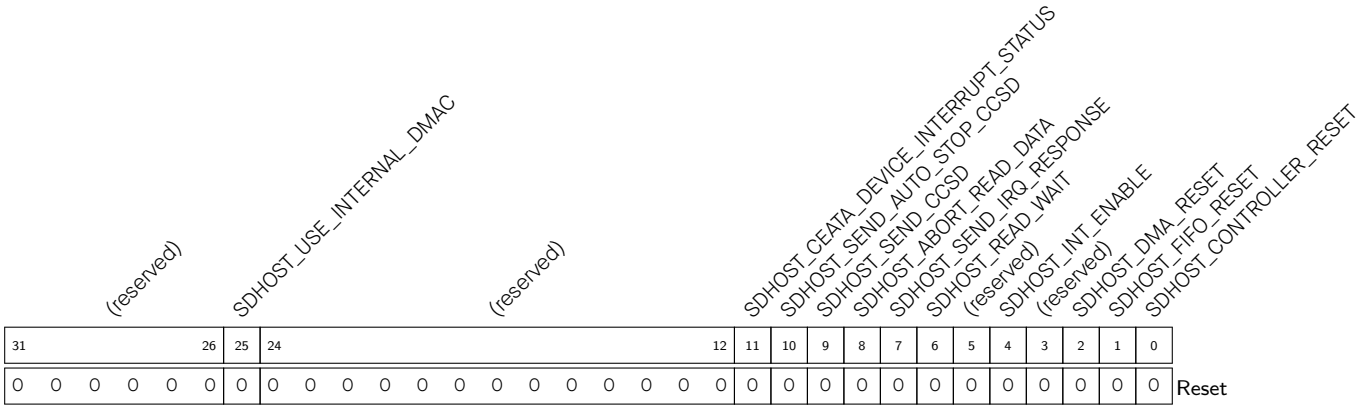
Name	Description	Address	Access
SDHOST_WRTPRT_REG	Card write protection (WP) status	0x0054	RO
SDHOST_TCBCNT_REG	Transferred byte count	0x005C	RO
SDHOST_TBBCNT_REG	Transferred byte count	0x0060	RO
SDHOST_DSCADDR_REG	Host descriptor address pointer	0x0094	RO
SDHOST_BUFADDR_REG	Host buffer address pointer	0x0098	RO
Transmit and receive data registers			
SDHOST_BUFFIFO_REG	Software write and read transmit data by FIFO	0x0200	R/W

39.13 Registers

The addresses in this section are relative to the SD/MMC Host Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 39.1. SDHOST_CTRL_REG (0x0000)



SDHOST_CONTROLLER_RESET Configures whether to reset the controller. This bit is auto-cleared after controller reset.

0: No effect

1: Reset

(R/W)

SDHOST_FIFO_RESET Configures whether to reset FIFO. This bit is auto-cleared after FIFO reset.

0: No effect

1: Reset

(R/W)

SDHOST_DMA_RESET Configures whether to reset DMA interface. This bit is auto-cleared after DMA reset.

0: No effect

1: Reset

(R/W)

SDHOST_INT_ENABLE Write 1 to enable global interrupt port. (R/W)

SDHOST_READ_WAIT Configures whether to send read-wait to SDIO cards.

0: Clear read wait

1: Assert read wait

(R/W)

Continued on the next page...

Register 39.1. CTRL_REG (0x0000)

Continued from the previous page...

SDHOST_SEND_IRQ_RESPONSE Configures whether to send auto interrupt request (IRQ) response. This bit automatically clears once a response is sent.
To wait for MMC card interrupts, the host issues a CMD40 command and waits for an interrupt response from MMC card(s). In the meantime, if the host wants SD/MMC to exit waiting for the interrupt state, it can set this bit, at which time SD/MMC command state-machine sends a CMD40 response on the bus and returns to the idle state.
0: No effect
1: Send auto IRQ response
(R/W)

SDHOST_ABORT_READ_DATA Configures whether to abort read data. This bit is automatically cleared once the data state machine is reset to idle.
After a suspend-command is issued during a read-operation, software polls the card to find when the suspend-event occurred. Once the suspend-event has occurred, software sets the bit which will reset the data state machine that is waiting for the next block of data.
0: No effect
1: Abort read data
(R/W)

SDHOST_SEND_CCSD Configures whether to send Command Completion Signal Disable (CCSD) to CE-ATA device. Once the CCSD pattern is sent to the device, SD/MMC automatically clears the SDHOST_SEND_CCSD bit. It also sets the Command Done (CD) bit in the SDHOST_RINTSTS_REG register.
Software sets this bit only if the current command is expecting CCS (that is, RW_BLK), and if interrupts are enabled for the CE-ATA device.
0:No effect
1: Send CCSD to CE-ATA device
(R/W)

SDHOST_SEND_AUTO_STOP_CCSD Configures whether to send an internally-generated STOP command (CMD12) to the CE-ATA device. After sending the CCSD, SD/MMC automatically clears the SDHOST_SEND_AUTO_STOP_CCSD bit. After sending this internally-generated STOP command, the Auto Command Done (ACD) bit in SDHOST_RINTSTS_REG is set.
Always Set SDHOST_SEND_AUTO_STOP_CCSD and SDHOST_SEND_CCSD bits together, SDHOST_SEND_AUTO_STOP_CCSD should not be set independently of SDHOST_SEND_CCSD.
0:No effect
1: Send internally-generated STOP command (CMD12) to the CE-ATA device
(R/W)

Continued on the next page...

Register 39.1. CTRL_REG (0x0000)

Continued from the previous page...

SDHOST_CEATA_DEVICE_INTERRUPT_STATUS Write 1 to enable interrupts in CE-ATA device (nIEN = 0 in ATE control register).

Software should appropriately write to this bit after the power-on reset or any other reset to the CE-ATA device. After reset, the CE-ATA device's interrupt is usually disabled (nIEN = 1). If the host enables the CE-ATA device's interrupt, then software should set this bit. (R/W)

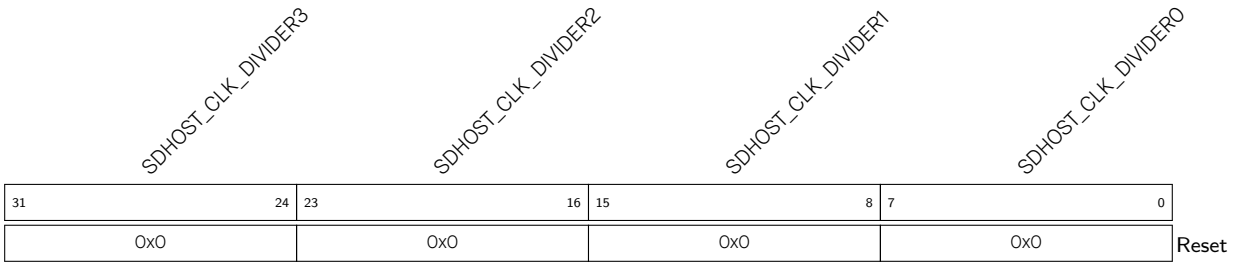
SDHOST_USE_INTERNAL_DMAC Configures whether to use DMA for data transfer.

0: Not use

1: Use

(R/W)

Register 39.2. SDHOST_CLKDIV_REG (0x0008)



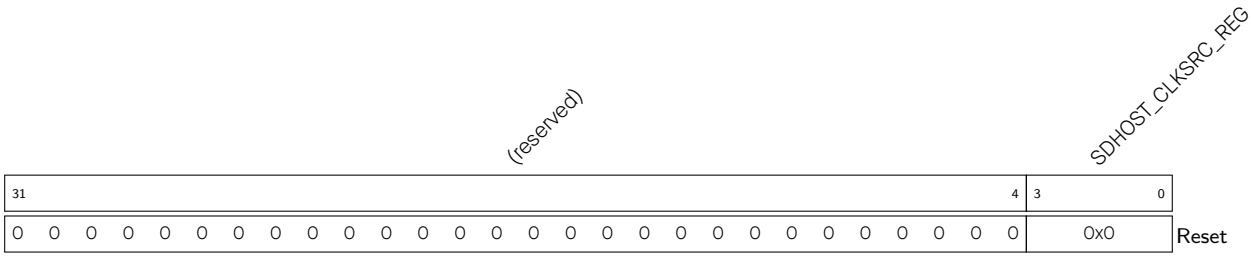
SDHOST_CLK_DIVIDER0 Configures clock divider0 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER1 Configures clock divider1 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER2 Configures clock divider2 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

SDHOST_CLK_DIVIDER3 Configures clock divider3 value. Clock divisor is 2*n, where n = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. (R/W)

Register 39.3. SDHOST_CLKSRC_REG (0x000C)



SDHOST_CLKSRC_REG Configures clock divider source for two SD cards. Each card has two bits assigned to it. For example, bit[1:0] are assigned to card 0, and bit[3:2] are assigned to card 1. Card 1/0 maps and internally routes clock divider[0:3] outputs to cclk_out[1:0] pins, depending on bit value. For every card's clock divider source:

0x0: Clock divider 0

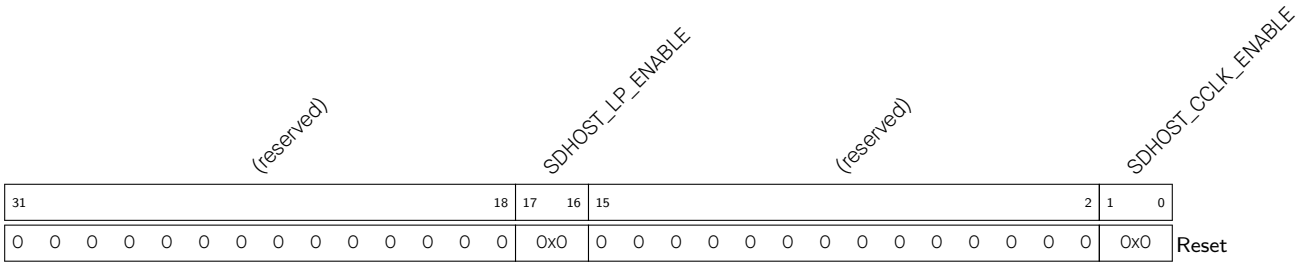
0x1: Clock divider 1

0x2: Clock divider 2

0x3: Clock divider 3

(R/W)

Register 39.4. SDHOST_CLKENA_REG (0x0010)



SDHOST_CCLK_ENABLE Configures whether to enable two SD card clocks or one MMC card clock. One bit per card. For every card's bit:

0: Clock disabled

1: Clock enabled

(R/W)

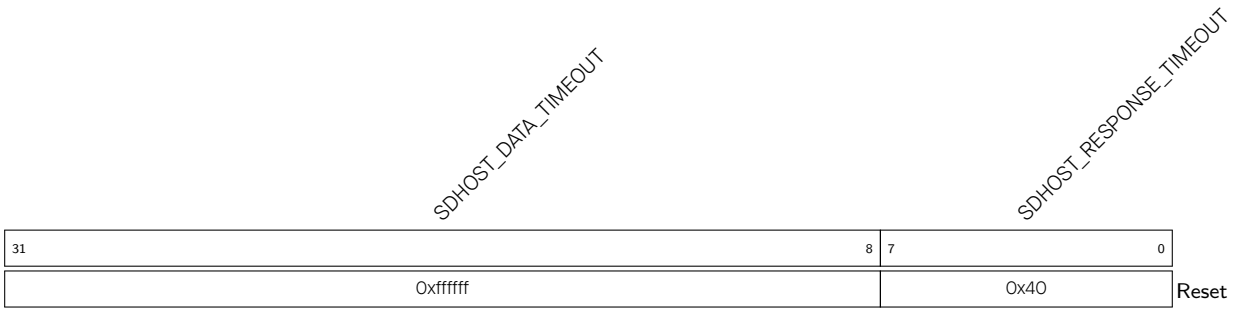
SDHOST_LP_ENABLE Configures whether to disable the clock when the card is in an IDLE state. One bit per card. For every card's bit:

0: Clock disabled

1: Clock enabled

(R/W)

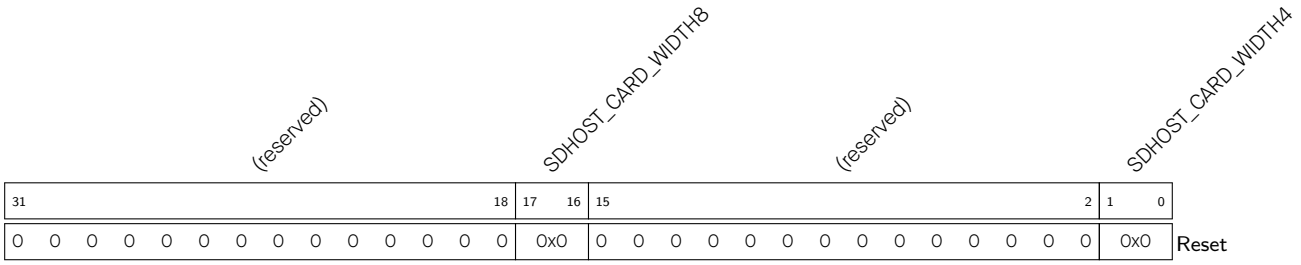
Register 39.5. SDHOST_TMOUT_REG (0x0014)



SDHOST_RESPONSE_TIMEOUT Configures response timeout value. Measurement unit: Card output clock period. (R/W)

SDHOST_DATA_TIMEOUT Configures card data read timeout value. This value is also used for data starvation by host timeout. The timeout counter is started only after the card clock is stopped. Measurement unit: Card output clock period.
NOTE: The software timer should be used if the timeout value is in the order of 100 ms. In this case, read data timeout interrupt (DRTO) needs to be disabled.
(R/W)

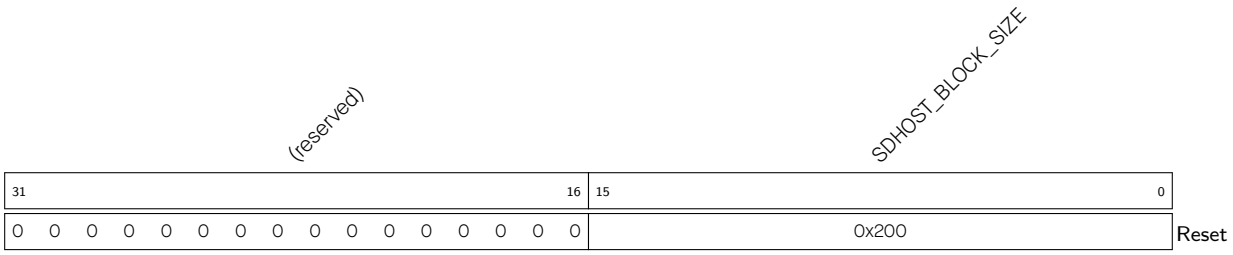
Register 39.6. SDHOST_CTYPE_REG (0x0018)



SDHOST_CARD_WIDTH4 Configures whether the card is 1-bit or 4-bit mode. One bit per card. Bit[0] corresponds to card[0]. For every card's bit:
0: 1-bit mode
1: 4-bit mode
(R/W)

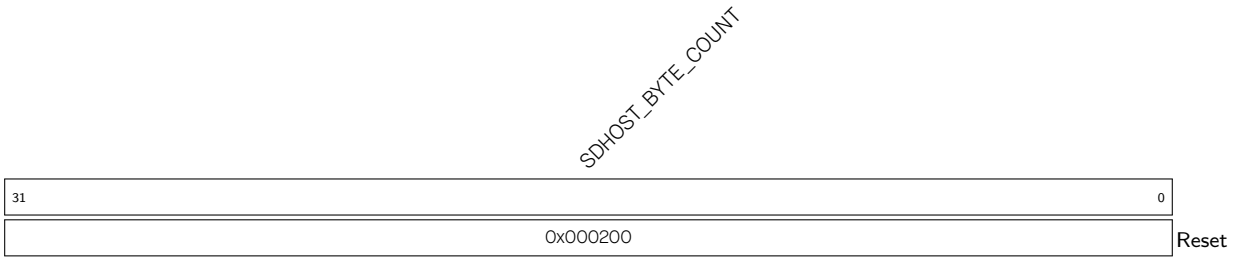
SDHOST_CARD_WIDTH8 Configures whether the card is 8-bit mode. One bit per card. Bit[0] corresponds to card[0]. For every card's bit:
0: Non 8-bit mode
1: 8-bit mode
(R/W)

Register 39.7. SDHOST_BLKSIZ_REG (0x001C)



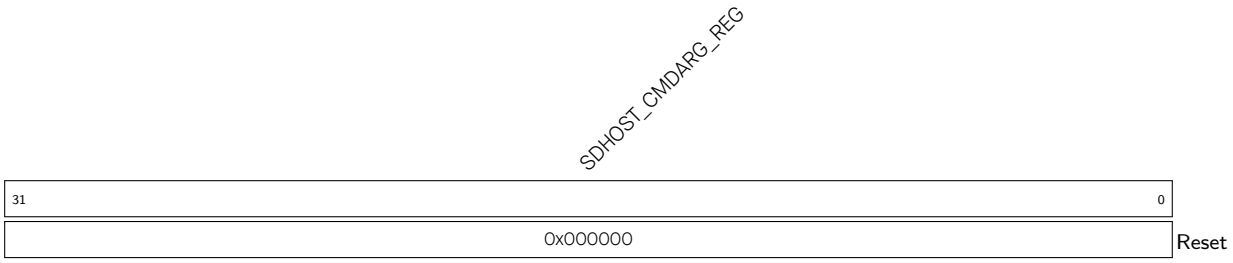
SDHOST_BLOCK_SIZE Configures the Block Size. (R/W)

Register 39.8. SDHOST_BYTCNT_REG (0x0020)



SDHOST_BYTE_COUNT Configures the number of bytes to be transferred, should be an integral multiple of Block Size for block transfers. For data transfers of undefined byte lengths, the byte count should be set to 0. When the byte count is set to 0, it is the responsibility of the host to explicitly send a stop/abort command to terminate data transfer. (R/W)

Register 39.9. SDHOST_CMDARG_REG (0x0028)



SDHOST_CMDARG_REG Configures the value indicates the command argument to be passed to the card. (R/W)

Register 39.10. SDHOST_CMD_REG (0x002C)

Bits	Field	Reset
31	SDHOST_START_CMD	0
30	(reserved)	0
29	SDHOST_USE_HOLE_REG	1
28	SDHOST_VOLT_SWITCH	0
27:24	(reserved)	0 0 0 0
23	SDHOST_CCS_EXPECTED	0
22	SDHOST_READ_CEATA_DEVICE	0
21	SDHOST_UPDATE_CLOCK_REGISTERS_ONLY	0
20:16	SDHOST_CARD_NUMBER	0x0
15	SDHOST_SEND_INITIALIZATION	0
14	SDHOST_STOP_ABORT_CMD	0
13	SDHOST_WAIT_PRVDATA_COMPLETE	0
12	SDHOST_SEND_AUTO_STOP	0
11	SDHOST_TRANSFER_MODE	0
10	SDHOST_READ_WRITE	0
9	SDHOST_DATA_EXPECTED	0
8	SDHOST_CHECK_RESPONSE_CRC	0
7	SDHOST_RESPONSE_LENGTH	0
6	SDHOST_RESPONSE_EXPECT	0
5:0	SDHOST_CMD_INDEX	0x0

SDHOST_CMD_INDEX Configures the Command index send to card. (R/W)

SDHOST_RESPONSE_EXPECT Configures whether to expect response from card.

0: No response expected from card

1: Response expected from card

(R/W)

SDHOST_RESPONSE_LENGTH Configures response length from card.

0: Short response expected from card

1: Long response expected from card

(R/W)

SDHOST_CHECK_RESPONSE_CRC Configures whether to check response CRC.

0: Do not check

1: Check response CRC

(R/W)

SDHOST_DATA_EXPECTED Configures whether to expect data transfer.

0: No data transfer expected

1: Data transfer expected

(R/W)

SDHOST_READ_WRITE Configures data transfer direction.

0: Read from card

1: Write to card

(R/W)

SDHOST_TRANSFER_MODE Configures data transfer mode.

0: Block data transfer command

1: Stream data transfer command

(R/W)

SDHOST_SEND_AUTO_STOP Configures whether to automatically send stop command.

0: No stop command is sent at the end of data transfer

1: Send stop command at the end of data transfer

(R/W)

Continued on the next page...

Register 39.10. SDHOST_CMD_REG (0x002C)

Continued from the previous page...

SDHOST_WAIT_PRVDATA_COMPLETE Configures whether to send command at once.

0: Send command at once, even if previous data transfer has not completed

1: Wait for previous data transfer to complete before sending Command

(R/W)

SDHOST_STOP_ABORT_CMD Configures whether the stop/abort command stops data transfer.

0: Neither stop nor abort command can stop the current data transfer. If abort is sent to the card device currently selected, or the card device is not in data-transfer mode, then bit should be set to 0

1: Stop or abort command intended to stop current data transfer in progress

(R/W)

SDHOST_SEND_INITIALIZATION Configures whether to send initialization sequence (80 clocks of 1) before sending this command. After powered on, 80 clocks must be sent to card for initialization before sending any commands to card. Bit should be set while sending the first command to card so that the controller will initialize clocks before sending command to card.

0: Do not send initialization sequence before sending this command

1: Send initialization sequence before sending this command

(R/W)

SDHOST_CARD_NUMBER Configures card number in use, the physical slot number of the card being accessed. In SD-only mode, up to two cards are supported. (R/W)

SDHOST_UPDATE_CLOCK_REGISTERS_ONLY Configures whether to only update clock register value into card clock domain.

Following register values are transferred into card clock domain: SDHOST_CLKDIV_REG, SDHOST_CLRSRC_REG, and SDHOST_CLKENA_REG.

When this bit is set, there will be no Command Done interrupts because no command is sent to SD_MMC_CEATA cards.

0: Normal command sequence

1: Do not send commands, just update clock register value into card clock domain

(R/W)

SDHOST_READ_CEATA_DEVICE Configures whether to read access CE-ATA device.

This bit is used to disable read data timeout indication while performing CE-ATA read transfers.

0: Host is not performing read access (RW_REG or RW_BLK)towards CE-ATA device

1: Host is performing read access (RW_REG or RW_BLK) towards CE-ATA device

(R/W)

Continued on the next page...

Register 39.10. SDHOST_CMD_REG (0x002C)

Continued from the previous page...

SDHOST_CCS_EXPECTED Configures whether to expect Command Completion Signal (CCS).

0: Interrupts are not enabled in CE-ATA device (nIEN = 1 in ATA control register), or command does not expect CCS from CE-ATA device

1: Interrupts are enabled in CE-ATA device (nIEN = 0), and RW_BLK command expects CCS from CE-ATA device

(R/W)

SDHOST_VOLT_SWITCH Configures whether to enable voltage switching.

0: No voltage switching

1: Voltage switching enabled, must be set for CMD11 only

(R/W)

SDHOST_USE_HOLE_REG Configures whether to use Hold Register.

0: CMD and DATA sent to card bypassing HOLD Register

1: CMD and DATA sent to card through the HOLD Register

(R/W)

SDHOST_START_CMD Configures whether to to start the command.

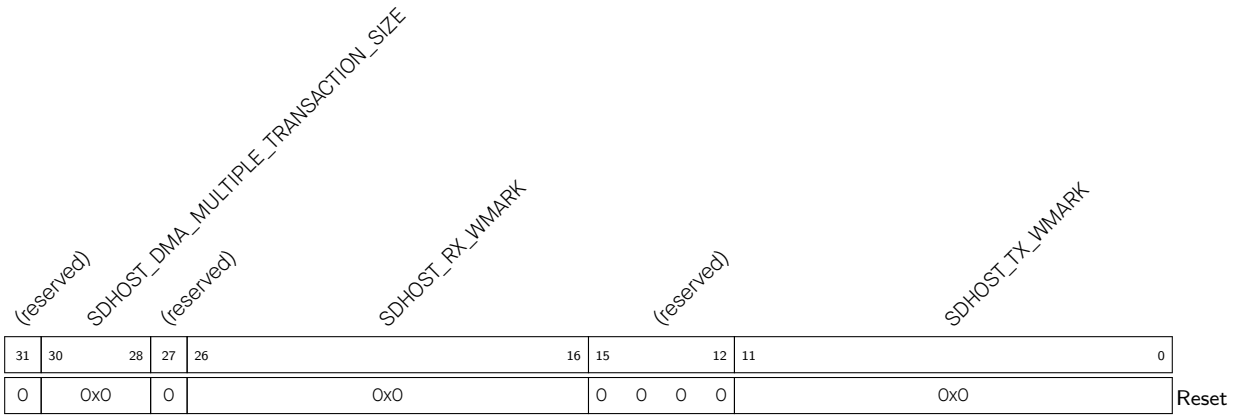
0: Not start

1: Start

Once command is served by the CIU, this bit is automatically cleared.

When this bit is set, host should not attempt to write to any command registers. If a write is attempted, hardware lock error is set in raw interrupt register. (R/W/SC)

Register 39.11. SDHOST_FIFOTH_REG (0x004C)

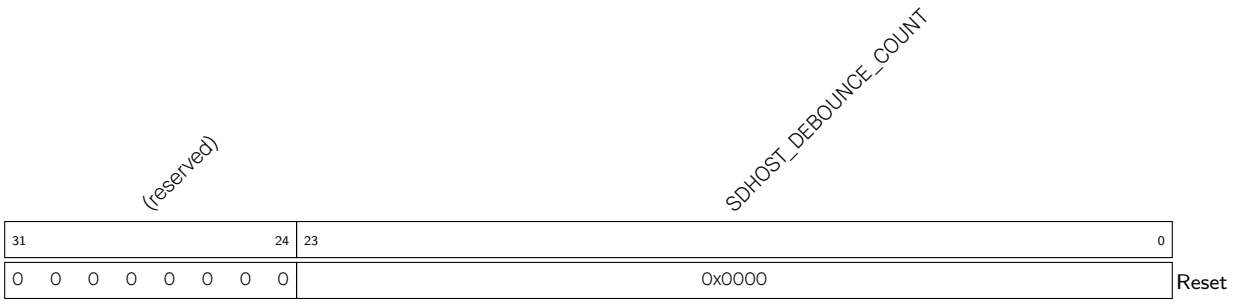


SDHOST_TX_WMARK Configures FIFO threshold watermark level when transmitting data to card. When FIFO data count is less than or equal to this number, DMA/FIFO request is raised. During end of packet, request or interrupt is generated, regardless of threshold programming.
In non-DMA mode, when transmitting FIFO threshold (TXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, on last interrupt, host is responsible for filling FIFO with only the required remaining bytes (not before FIFO is full or after CIU completes data transfers, because FIFO may not be empty).
In DMA mode, at end of packet, if last transfer is less than burst size, DMA controller does single cycles until required bytes are transferred. (R/W)

SDHOST_RX_WMARK Configures FIFO threshold watermark level when receiving data from card. When FIFO data count reaches greater than this number, DMA/FIFO request is raised. During end of packet, request is generated regardless of threshold programming in order to complete any remaining data.
In non-DMA mode, when receiver FIFO threshold (RXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, interrupt is not generated if threshold programming is larger than any remaining data. It is responsibility of host to read remaining bytes on seeing Data Transfer Done interrupt.
In DMA mode, at end of packet, even if remaining bytes are less than threshold, DMA request does single transfers to flush out any remaining bytes before Data Transfer Done interrupt is set. (R/W)

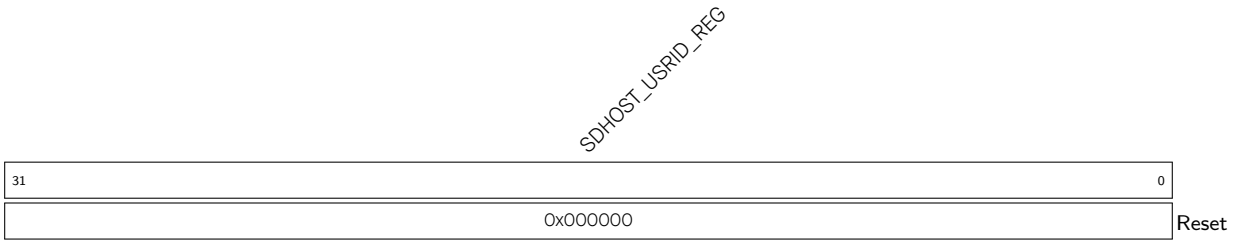
SDHOST_DMA_MULTIPLE_TRANSACTION_SIZE Configures burst size of multiple transaction, should be programmed same as DMA controller multiple-transaction-size SDHOST_SRC/DEST_MSIZE.
0x0: 1-byte transfer
0x1: 4-byte transfer
0x2: 8-byte transfer
0x3: 16-byte transfer
0x4: 32-byte transfer
0x5: 64-byte transfer
0x6: 128-byte transfer
0x7: 256-byte transfer
(R/W)

Register 39.12. SDHOST_DEBNCE_REG (0x0064)



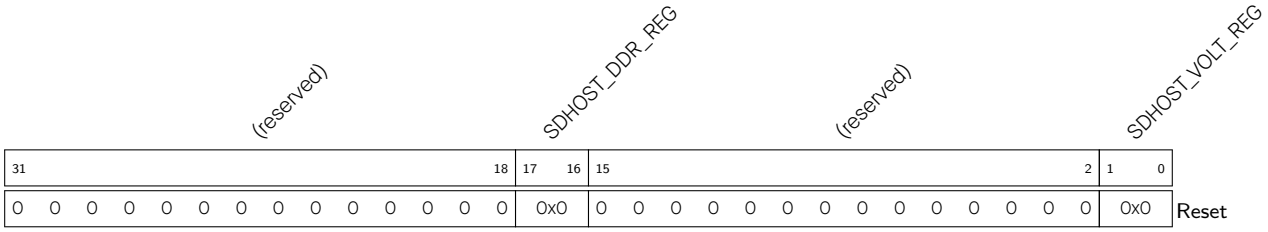
SDHOST_DEBOUNCE_COUNT Configures the number of host clocks (clk) used by debounce filter logic. The typical debounce time is 5 ~ 25 ms to prevent the card instability when the card is inserted or removed. (R/W)

Register 39.13. SDHOST_USRID_REG (0x0068)



SDHOST_USRID_REG Configures user identification, value set by users. Can also be used as a scratchpad register by users. (R/W)

Register 39.14. SDHOST_UHS_REG (0x0074)



SDHOST_VOLT_REG Configures High Voltage mode per card. Bit[0] corresponds to card[0]. For every card bit:

0: 3.3 V VDD

1: 1.8 V VDD

(R/W)

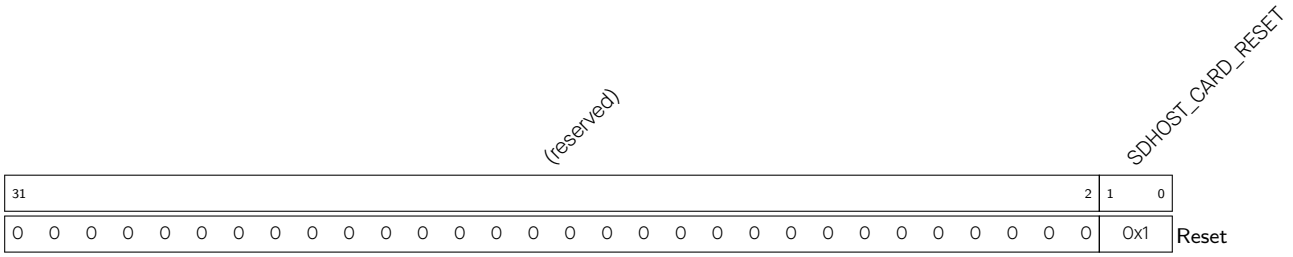
SDHOST_DDR_REG Configures DDR (double data rate) mode selecton per card. Bit[0] corresponds to card[0]. For every card bit:

0: Non-DDR mdoe

1: DDR mdoe

(R/W)

Register 39.15. SDHOST_RST_N_REG (0x0078)



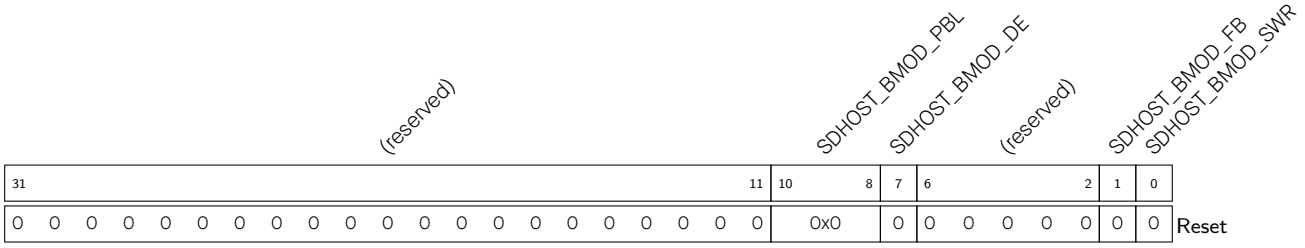
SDHOST_CARD_RESET Configures hardware reset per card. Bit[0] corresponds to card[0]. These bits cause the cards to enter the pre-idle state, which requires them to be re-initialized. For every card bit:

0: Reset

1: Active mode

(R/W)

Register 39.16. SDHOST_BMOD_REG (0x0080)



SDHOST_BMOD_SWR Configures whether to reset all DMA internal registers. It is automatically cleared after one clock cycle.

0: No effect

1: Reset

(R/W)

SDHOST_BMOD_FB Configures whether the AHB Master interface performs fixed burst transfers.

0: Use SINGLE and INCR burst transfer operations

1: Only use SINGLE, INCR4, INCR8 or INCR16 burst transfers

(R/W)

SDHOST_BMOD_DE Configures whether to enable the DMA.

0: Not enable

1: Enable

(R/W)

SDHOST_BMOD_PBL Configures the maximum number of beats to be performed in one DMA transaction.

0x0: 1-beat transfer

0x1: 4-beat transfer

0x2: 8-beat transfer

0x3: 16-beat transfer

0x4: 32-beat transfer

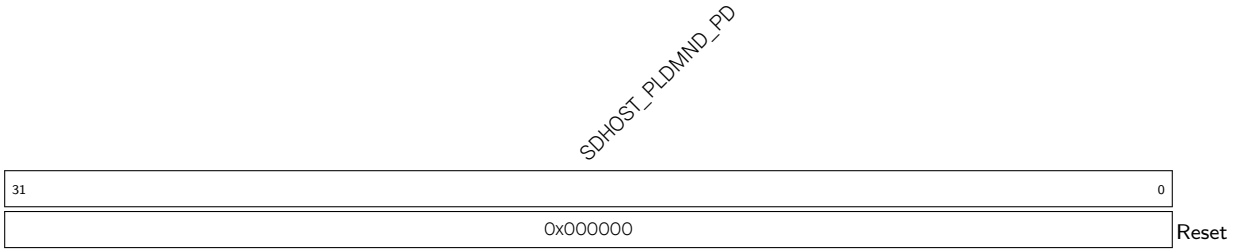
0x5: 64-beat transfer

0x6: 128-beat transfer

0x7: 256-beat transfer

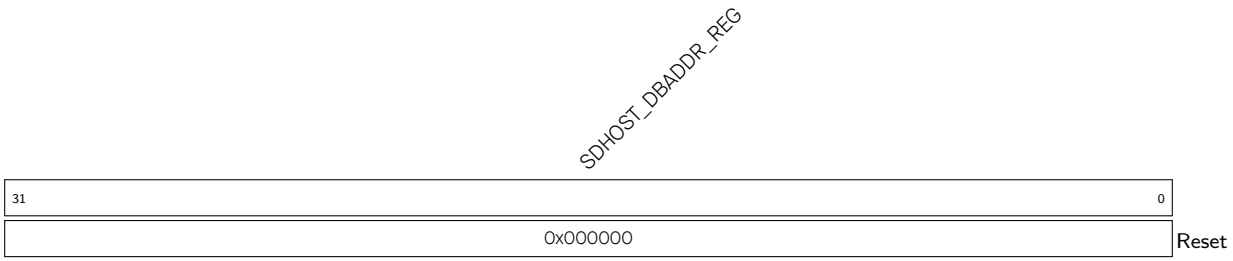
(R/W)

Register 39.17. SDHOST_PLDMND_REG (0x0084)



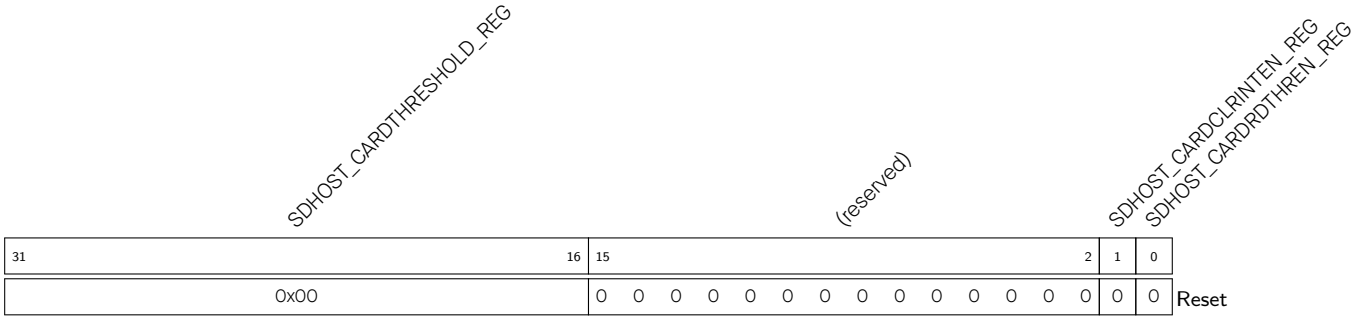
SDHOST_PLDMND_PD Configures the DMA FSM to resume normal descriptor fetch operation. If the OWNER bit of a descriptor is not set, the FSM goes to the Suspend state, software needs to write any value into this field for the FSM to resume normal descriptor fetch operation. (WO)

Register 39.18. SDHOST_DBADDR_REG (0x0088)



SDHOST_DBADDR_REG Configures the base address of the linked list (First Descriptor). Bit[1:0] are ignored and taken as all-zero by the DMA internally. (R/W)

Register 39.19. SDHOST_CARDTHRCTL_REG (0x0100)



SDHOST_CARDRDTHREN_REG Configures whether to enable card read threshold.

0: Not enable

1: Enable

(R/W)

SDHOST_CARDCLRINTEN_REG Configures whether to enable busy clear interrupt generation.

0: Not enable

1: Enable

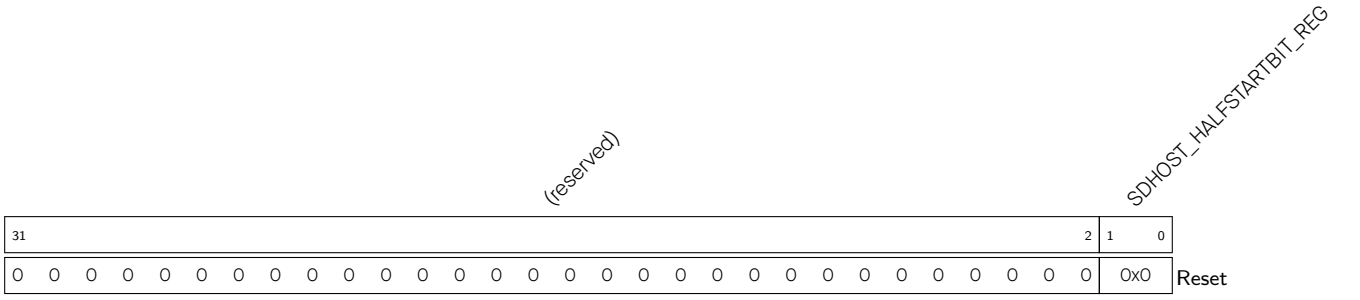
(R/W)

SDHOST_CARDTHRESHOLD_REG Configures the card read threshold. Measurement unit: byte.

This field is valid only when SDHOST_CARDRDTHREN_REG is set to 1.

The value should be less than the FIFO size 512.(R/W)

Register 39.20. SDHOST_EMMCDDR_REG (0x010C)



SDHOST_HALFSTARTBIT_REG Configures the start bit detection mechanism duration of start bit. Each bit refers to one card. Set this bit to 1 for eMMC4.5 and above, set to 0 for SD applications.

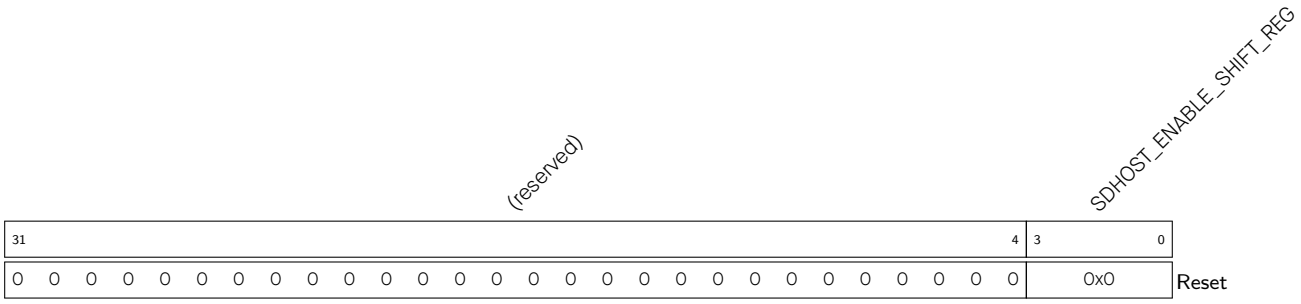
For eMMC4.5, start bit can be:

0: Full cycle

1: Less than one full cycle

(R/W)

Register 39.21. SDHOST_ENSHIFT_REG (0x0110)



SDHOST_ENABLE_SHIFT_REG Configures the amount of phase shift provided on the default enables in the design. Two bits are assigned for each card, e.g., bit[1:0] are assigned to card 0. For every card:

0x0: Default phase shift

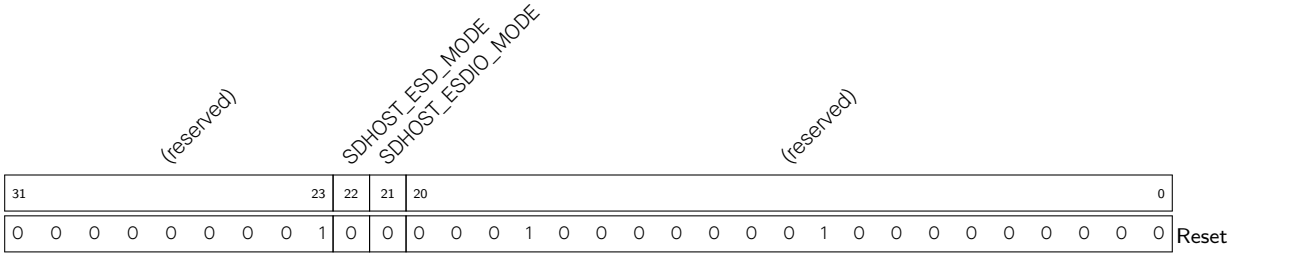
0x1: Enables shifted to next immediate positive edge

0x2: Enables shifted to next immediate negative edge

0x3: Reserved

(R/W)

Register 39.22. SDHOST_CLK_EDGE_SEL_REG (0x0800)



SDHOST_ESDIO_MODE Configures whether to enable eSDIO mode.

0: Not enable

1: Enable

(R/W)

SDHOST_ESD_MODE Configures whether to enable eSD mode.

0: Not enable

1: Enable

(R/W)

Register 39.23. SDHOST_DLL_CLK_CONF_REG (0x0808)

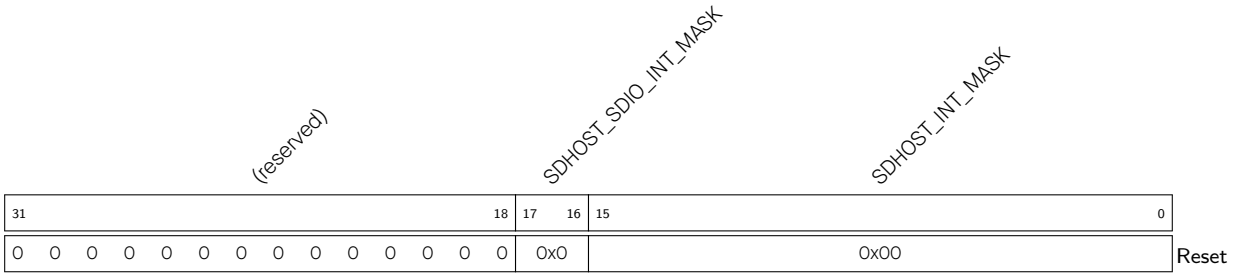
(reserved)	SDHOST_DLL_CCLK_IN_SAM_PHASE	SDHOST_DLL_CCLK_IN_DRV_PHASE	SDHOST_DLL_CCLK_IN_SLF_PHASE	(reserved)	
31 21	20 15	14 9	8 3	2 0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0	Reset

SDHOST_DLL_CCLK_IN_SLF_PHASE Configures the clock phase of the internal signal when selecting high frequency clock. Unit is 1/64 of clock cycle. (R/W)

SDHOST_DLL_CCLK_IN_DRV_PHASE Configures the clock phase of the output signal when selecting high frequency clock. Unit is 1/64 of clock cycle. (R/W)

SDHOST_DLL_CCLK_IN_SAM_PHASE Configures the clock phase of the input signal when selecting high frequency clock. Unit is 1/64 of clock cycle. (R/W)

Register 39.24. SDHOST_INTMASK_REG (0x0024)



SDHOST_INT_MASK Write 1 to correspond bit to enable correspond interrupt. For every bit:

Bit 15 (EBE): End-bit error/no CRC error

Bit 14 (ACD): Auto command done

Bit 13 (SBE/BCI): Rx Start Bit Error

Bit 12 (HLE): Hardware locked write error

Bit 11 (FRUN): FIFO underrun/overrun error

Bit 10 (HTO): Data starvation-by-host timeout

Bit 9 (DRTO): Data read timeout

Bit 8 (RTO): Response timeout

Bit 7 (DCRC): Data CRC error

Bit 6 (RCRC): Response CRC error

Bit 5 (RXDR): Receive FIFO data request

Bit 4 (TXDR): Transmit FIFO data request

Bit 3 (DTO): Data transfer over

Bit 2 (CMDD): Command done

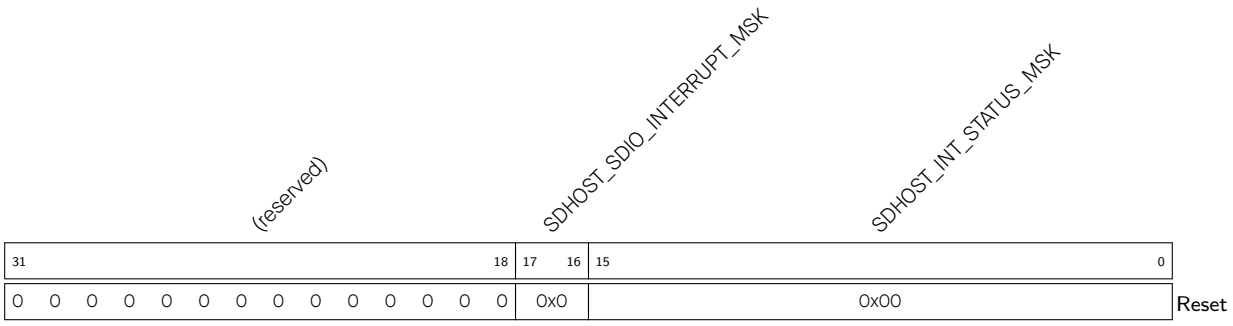
Bit 1 (RE): Response error

Bit 0 (CD): Card detect

(R/W)

SDHOST_SDIO_INT_MASK Write 1 to correspond bit to enable interrupt from correspond card. One bit per card. Bit[0] corresponds to card[0]. (R/W)

Register 39.25. SDHOST_MINTSTS_REG (0x0040)



SDHOST_INT_STATUS_MSK The masked interrupt status. For every bit:

Bit 15 (EBE): End-bit error/no CRC error

Bit 14 (ACD): Auto command done

Bit 13 (SBE/BCI): RX Start Bit Error

Bit 12 (HLE): Hardware locked write error

Bit 11 (FRUN): FIFO underrun/overrun error

Bit 10 (HTO): Data starvation by host timeout

Bit 9 (DTRO): Data read timeout

Bit 8 (RTO): Response timeout

Bit 7 (DCRC): Data CRC error

Bit 6 (RCRC): Response CRC error

Bit 5 (RXDR): Receive FIFO data request

Bit 4 (TXDR): Transmit FIFO data request

Bit 3 (DTO): Data transfer over

Bit 2 (CMDD): Command done

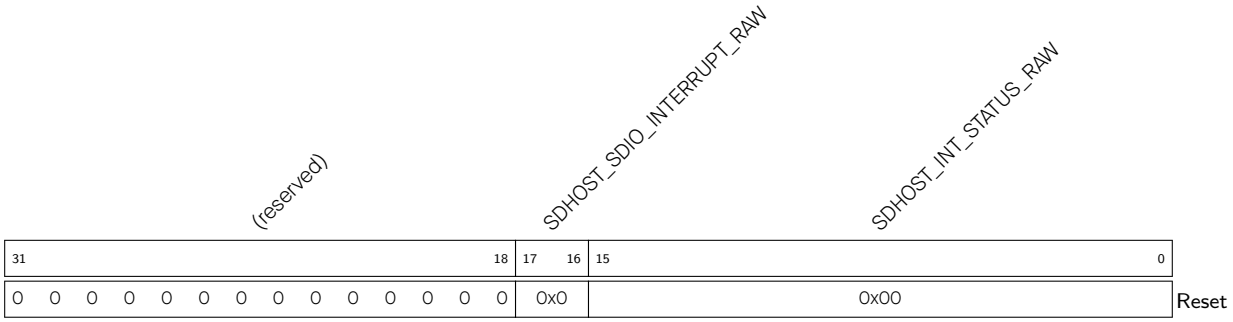
Bit 1 (RE): Response error

Bit 0 (CD): Card detect

(RO)

SDHOST_SDIO_INTERRUPT_MSK The masked interrupt status from SDIO card. One bit for each card. Bit[0] corresponds to card[0]. (RO)

Register 39.26. SDHOST_RINTSTS_REG (0x0044)

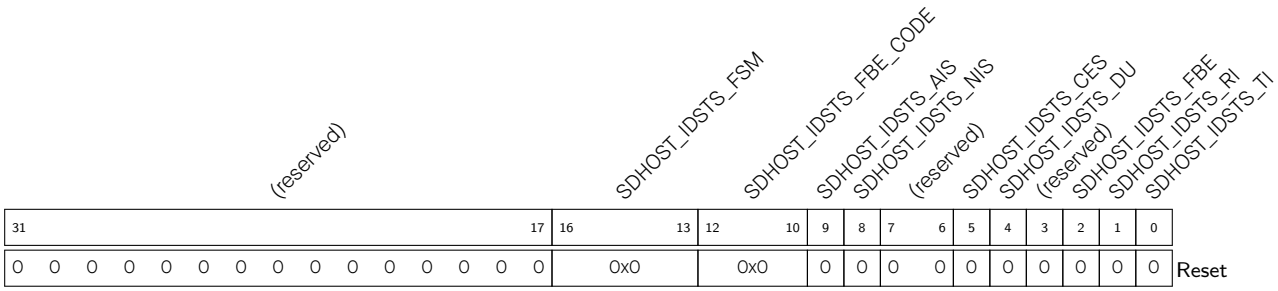


SDHOST_INT_STATUS_RAW The raw interrupt status. For every bit:

Bit 15 (EBE): End-bit error/no CRC error
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): RX Start Bit Error
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation by host timeout
Bit 9 (DTRO): Data read timeout
Bit 8 (RTO): Response timeout
Bit 7 (DCRC): Data CRC error
Bit 6 (RCRC): Response CRC error
Bit 5 (RXDR): Receive FIFO data request
Bit 4 (TXDR): Transmit FIFO data request
Bit 3 (DTO): Data transfer over
Bit 2 (CMDD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect
(R/W1C)

SDHOST_SDIO_INTERRUPT_RAW The raw interrupt status from SDIO card. One bit per card. Bit[0] corresponds to card[0]. (R/W1C)

Register 39.27. SDHOST_IDSTS_REG (0x008C)



SDHOST_IDSTS_TI The raw interrupt status of Transmit Interrupt. Indicates that data transmission is finished for a descriptor. (R/W1C)

SDHOST_IDSTS_RI The raw interrupt status of Receive Interrupt. Indicates the completion of data reception for a descriptor. (R/W1C)

SDHOST_IDSTS_FBE The raw interrupt status of Fatal Bus Error Interrupt. Indicates that a Bus Error occurred (SDHOST_IDSTS_REG[12:10]). (R/W1C)

SDHOST_IDSTS_DU The raw interrupt status of Descriptor Unavailable Interrupt. This bit is set when the descriptor is unavailable due to OWNER bit = 0 (DES0[31] = 0). (R/W1C)

SDHOST_IDSTS_CES The raw interrupt status of Card Error Summary. Indicates the status of the transaction to/from the card, also present in SDHOST_RINTSTS_REG. Indicates the logical OR of the following bits:
EBE: End-bit error/no CRC error
SBE/BCI: RX Start Bit Error
DRTO: Data read timeout
RTO: Response timeout
DCRC: Data CRC error
RCRC: Response CRC error
RE: Response error
The abort condition of the DMA depends on the setting of this field. If this field is enabled, then the DMA aborts on a response error. (R/W1C)

SDHOST_IDSTS_NIS The raw interrupt status of Normal Interrupt Summary. Logical OR of SDHOST_IDSTS_REG[1:0]. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes this field to be set is cleared by software. (R/W1C)

SDHOST_IDSTS_AIS The raw interrupt status of Abnormal Interrupt Summary. Logical OR of SDHOST_IDSTS_REG[2], SDHOST_IDSTS_REG[4]. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes this field to be set is cleared by software. (R/W1C)

Continued on the next page...

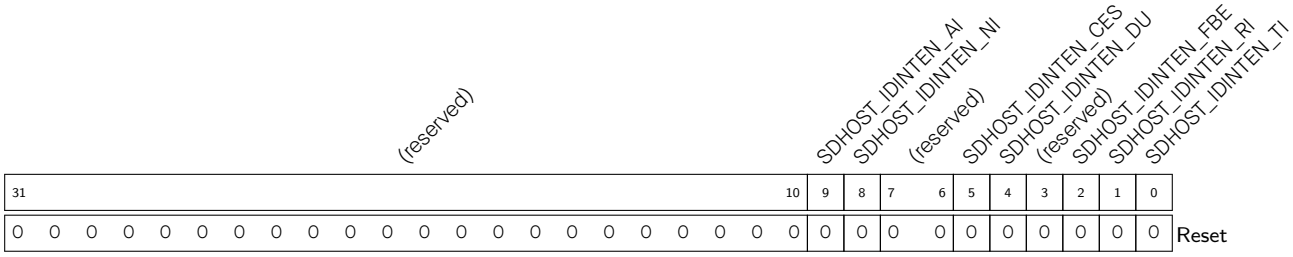
Register 39.27. SDHOST_IDSTS_REG (0x008C)

Continued from the previous page...

SDHOST_IDSTS_FBE_CODE Represents the type of error that caused a Bus Error, valid only when SDHOST_IDSTS_REG[2] is set.
0x1: Host Abort received during transmission
0x2: Host Abort received during reception
Others: Reserved
(RO)

SDHOST_IDSTS_FSM Represents DMA FSM present state.
0x0: DMA_IDLE (idle state)
0x1: DMA_SUSPEND (suspend state)
0x2: DESC_RD (descriptor reading state)
0x3: DESC_CHK (descriptor checking state)
0x4: DMA_RD_REQ_WAIT (read-data request waiting state)
0x5: DMA_WR_REQ_WAIT (write-data request waiting state)
0x6: DMA_RD (data-read state)
0x7: DMA_WR (data-write state)
0x8: DESC_CLOSE (descriptor close state)
(RO)

Register 39.28. SDHOST_IDINTEN_REG (0x0090)



SDHOST_IDINTEN_TI Write 1 to enable interrupt of Transmit.(R/W)

SDHOST_IDINTEN_RI Write 1 to enable interrupt of Receive. (R/W)

SDHOST_IDINTEN_FBE Write 1 to enable interrupt of Fatal Bus Error. (R/W)

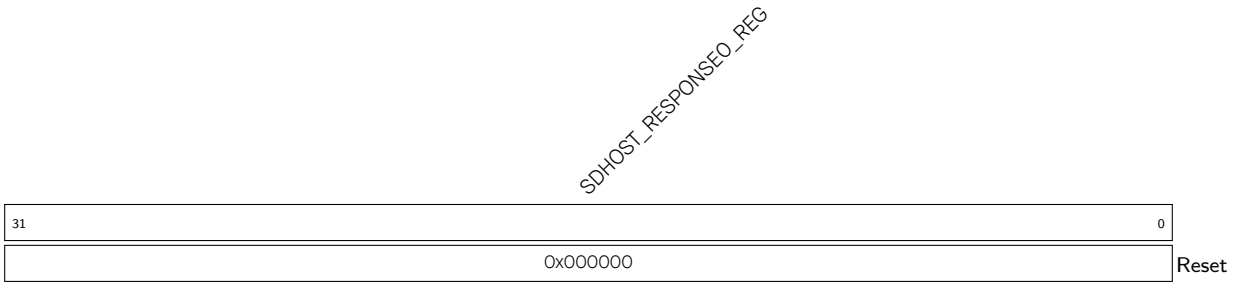
SDHOST_IDINTEN_DU Write 1 to enable interrupt of Descriptor Unavailable. (R/W)

SDHOST_IDINTEN_CES Write 1 to enable interrupt of Card Error summary. (R/W)

SDHOST_IDINTEN_NI Write 1 to enable interrupt of Normal Interrupt Summary. (R/W)

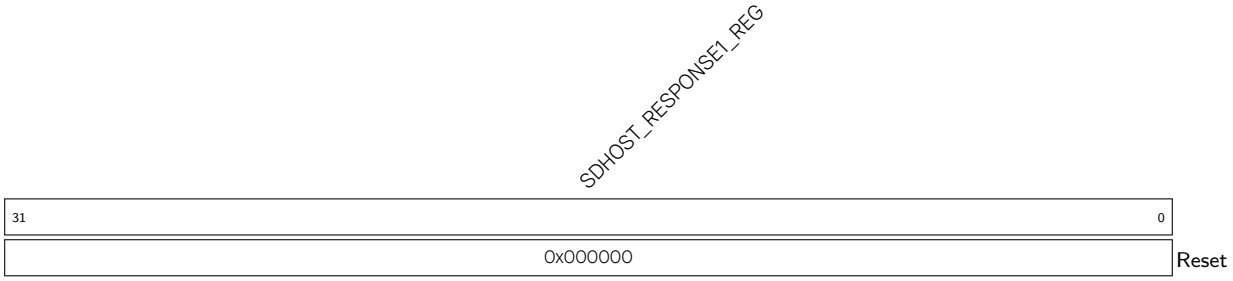
SDHOST_IDINTEN_AI Write 1 to enable interrupt of Abnormal Interrupt Summary. (R/W)

Register 39.29. SDHOST_RESP0_REG (0x0030)



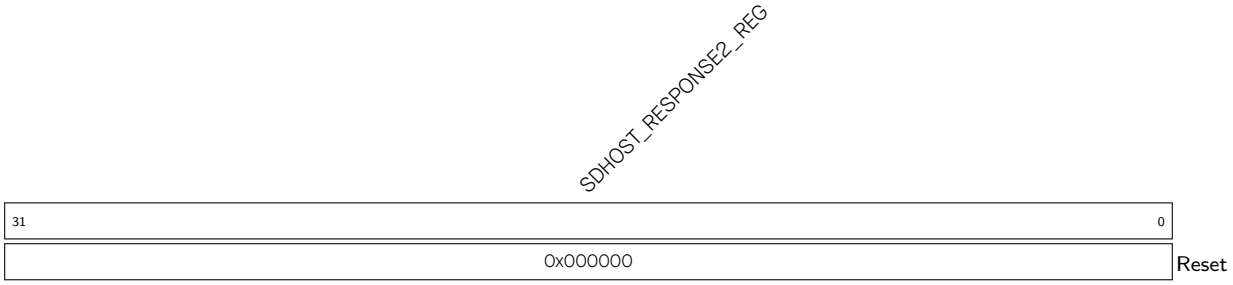
SDHOST_RESPONSE0_REG Represents bit[31:0] of response. (RO)

Register 39.30. SDHOST_RESP1_REG (0x0034)



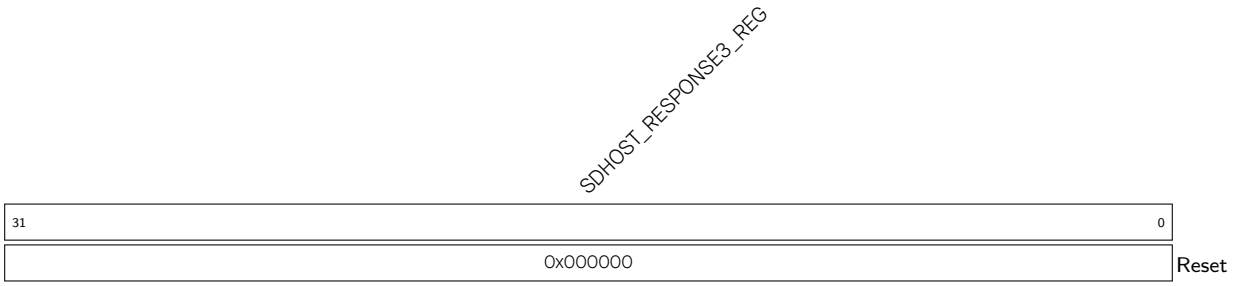
SDHOST_RESPONSE1_REG Represents bit[63:32] of long response. (RO)

Register 39.31. SDHOST_RESP2_REG (0x0038)



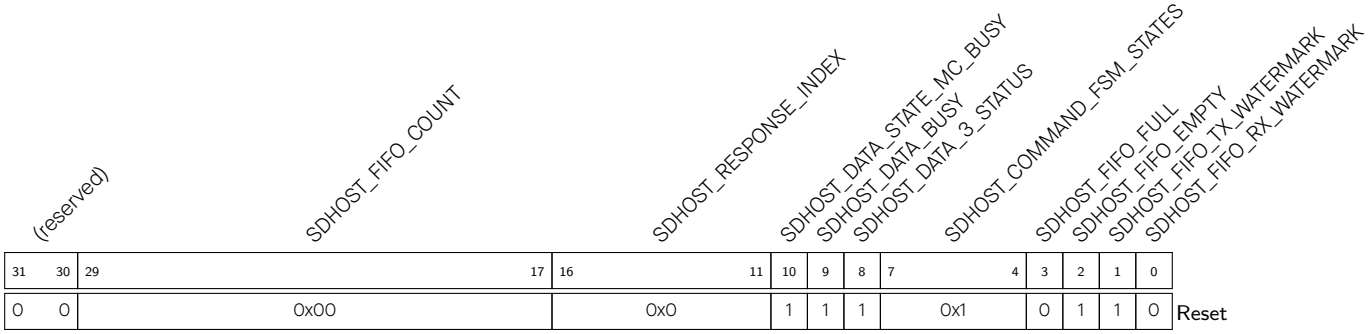
SDHOST_RESPONSE2_REG Represents bit[95:64] of long response. (RO)

Register 39.32. SDHOST_RESP3_REG (0x003C)



SDHOST_RESPONSE3_REG Represents bit[127:96] of long response. (RO)

Register 39.33. SDHOST_STATUS_REG (0x0048)



SDHOST_FIFO_RX_WATERMARK Represents whether the FIFO reached Receive watermark level, not qualified with data transfer.

0: Not reach

1: Reach

(RO)

SDHOST_FIFO_TX_WATERMARK Represents whether the FIFO reached Transmit watermark level, not qualified with data transfer.

0: Not reach

1: Reach

(RO)

SDHOST_FIFO_EMPTY Represents whether the FIFO is empty.

0: Not empty

1: Empty

(RO)

SDHOST_FIFO_FULL Represents whether the FIFO is full.

0: Not full

1: Full

(RO)

Continued on the next page...

Register 39.33. SDHOST_STATUS_REG (0x0048)

Continued from the previous page...

SDHOST_COMMAND_FSM_STATES Represents the command FSM states.
0x0: Idle
0x1: Send init sequence
0x2: Send cmd start bit
0x3: Send cmd tx bit
0x4: Send cmd index + arg
0x5: Send cmd crc7
0x6: Send cmd end bit
0x7: Receive resp start bit
0x8: Receive resp IRQ response
0x9: Receive resp tx bit
0xA: Receive resp cmd idx
0xB: Receive resp data
0xC: Receive resp crc7
0xD: Receive resp end bit
0xE: Cmd path wait NCC
0xF: Wait, cmd-to-response turnaround
(RO)

SDHOST_DATA_3_STATUS Represents the value of the sdhost_card_data[3] siganl, which indicates whether card is present.
0: Card not present
1: Card present
(RO)

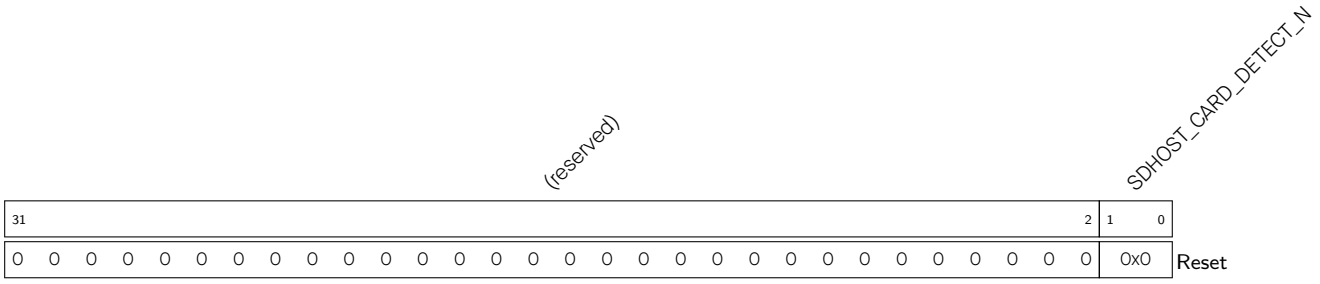
SDHOST_DATA_BUSY Represents inverted value of the sdhost_card_data[0] signal, which indicates card data is busy.
0: Card data not busy
1: Card data busy
(RO)

SDHOST_DATA_STATE_MC_BUSY Represents whether the data transmit or receive state-machine is busy.
0: Not busy
1: Busy
(RO)

SDHOST_RESPONSE_INDEX Represents index of previous response, including any auto-stop sent by core. (RO)

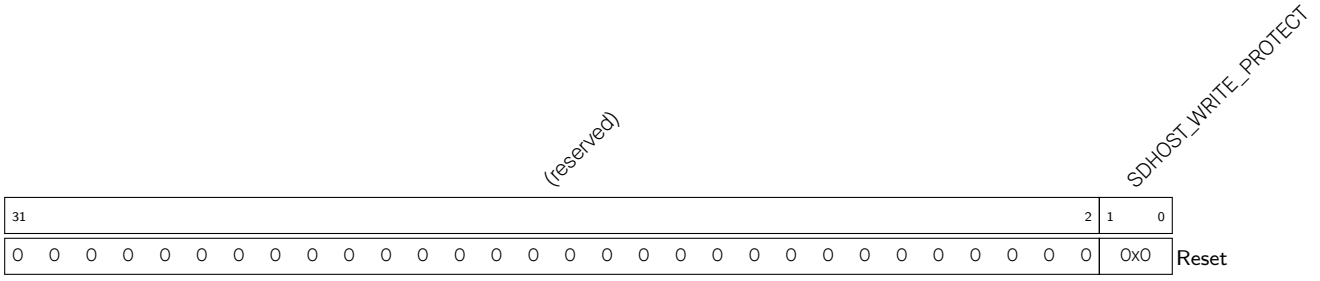
SDHOST_FIFO_COUNT Represents FIFO count, number of filled locations in FIFO. (RO)

Register 39.34. SDHOST_CDETECT_REG (0x0050)



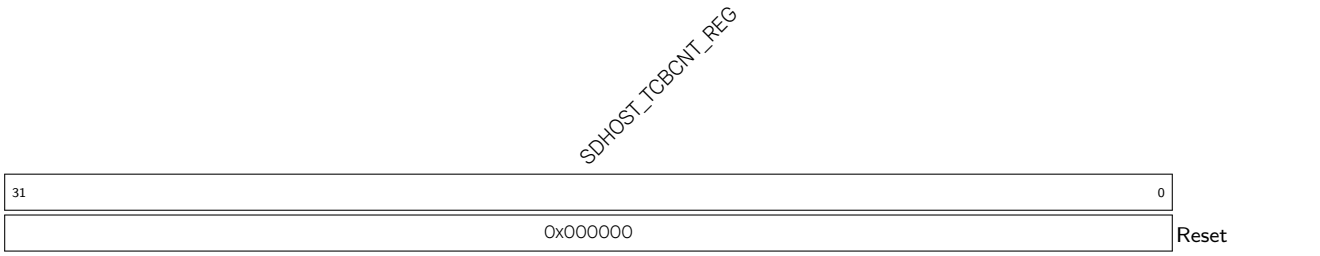
SDHOST_CARD_DETECT_N Represents value on sdhost_card_detect_n input ports. One bit per card. Bit[0] corresponds to card[0]. (RO)

Register 39.35. SDHOST_WRTPRT_REG (0x0054)



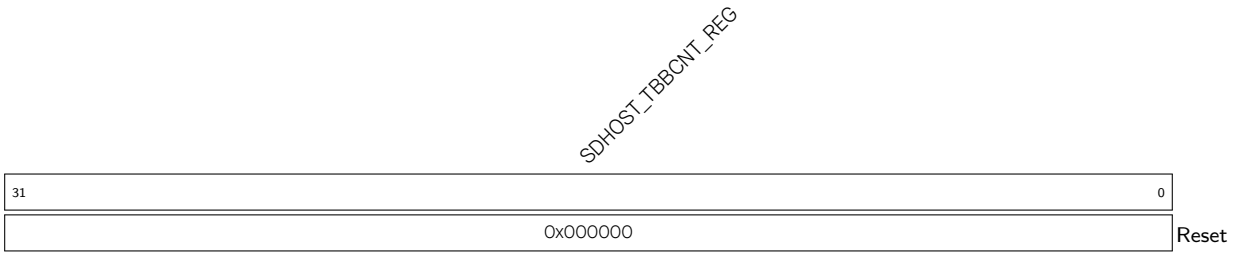
SDHOST_WRITE_PROTECT Represents value on sdhost_card_write_prt input ports, one bit per card. Bit[0] corresponds to card[0]. (RO)

Register 39.36. SDHOST_TCBCNT_REG (0x005C)



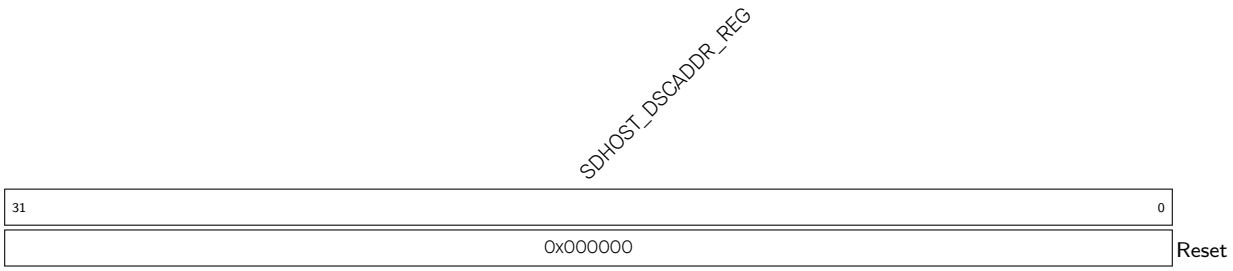
SDHOST_TCBCNT_REG Represents the number of bytes transferred by the CIU unit to the card. (RO)

Register 39.37. SDHOST_TBBCNT_REG (0x0060)



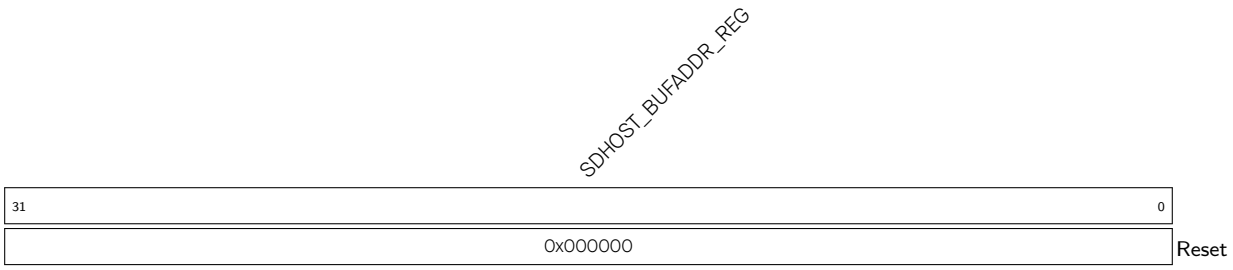
SDHOST_TBBCNT_REG Represents the number of bytes transferred between Host/DMA memory and BIU FIFO. (RO)

Register 39.38. SDHOST_DSCADDR_REG (0x0094)



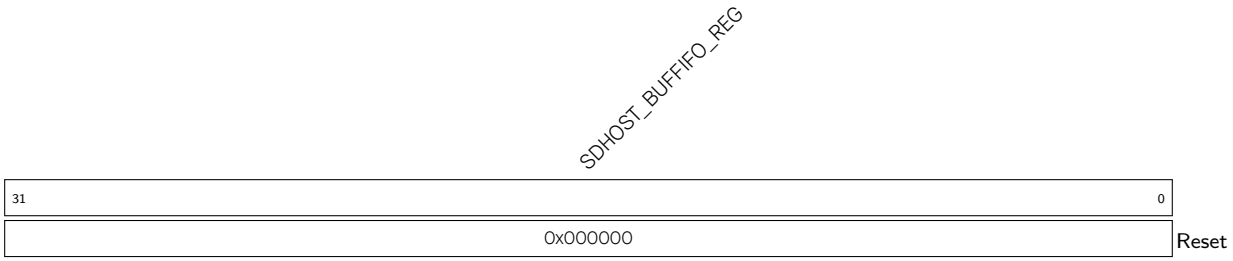
SDHOST_DSCADDR_REG Represents the Descriptor Address Pointer, points to the start address of current descriptor read by DMA. (RO)

Register 39.39. SDHOST_BUFADDR_REG (0x0098)



SDHOST_BUFADDR_REG Represents the Buffer Address Pointer, points to the current Data Buffer Address being accessed by the DMA. (RO)

Register 39.40. SDHOST_BUFFIFO_REG (0x0200)



SDHOST_BUFFIFO_REG Date received or to be transmitted between FIFO and software directly. This register points to the current Data FIFO. (R/W)

Chapter 40

LED PWM Controller (LEDC)

40.1 Overview

The LED PWM Controller is a peripheral designed to generate PWM signals for LED control. It has specialized features such as automatic duty cycle fading. However, the LED PWM Controller can also be used to generate PWM signals for other purposes.

40.2 Features

The LED PWM Controller has the following features:

- Eight independent PWM generators (i.e., eight channels)
- Maximum PWM duty cycle resolution: 20 bits
- Four independent timers that support fractional division
- Adjustable phase of PWM signal output
- PWM duty cycle dithering
- Automatic duty cycle fading — gradual increase/decrease of a PWM's duty cycle without interference from the processor. An interrupt will be generated upon fade completion
- Up to 16 duty cycle ranges for each PWM generator to generate gamma curve signals - each range can be independently configured in terms of fading direction (increase or decrease), fading amount (the amount by which the duty cycle increases or decreases each time), the number of fades (how many times the duty cycle fades in one range), and fading frequency
- PWM signal output in low-power mode (Light-sleep mode)
- Event generation and task response related to the Event Task Matrix (ETM) peripheral

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the eight PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 7).

40.3 Architectural Overview

Figure 40.3-1 shows the architecture of the LED PWM Controller.

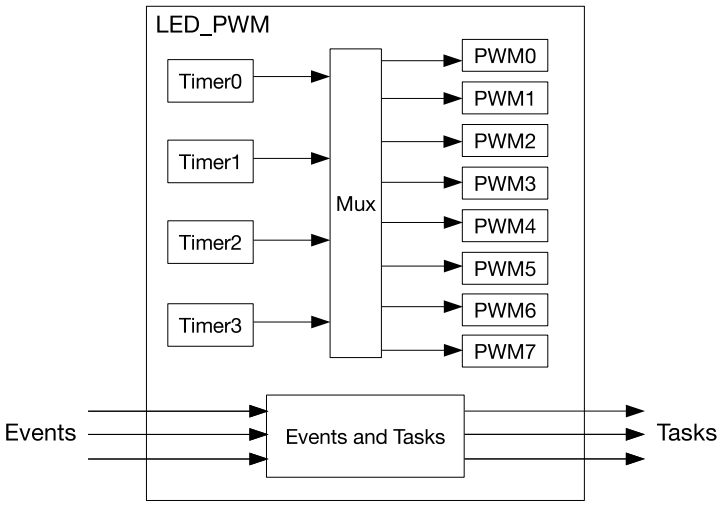


Figure 40.3-1. LED PWM Architecture

Each of the four timers has an internal timebase counter (i.e., a counter that counts on cycles of a reference clock) and thus can be independently configured (i.e., configurable clock divider, and counter overflow value). Each PWM generator selects one of the timers by configuring the LEDC_TIMER_SEL_CH*n*, and uses the timer's counter value timer*x*_cnt as a reference to generate its PWM signal.

Figure 40.3-2 illustrates the main functional blocks of the timer and the PWM generator.

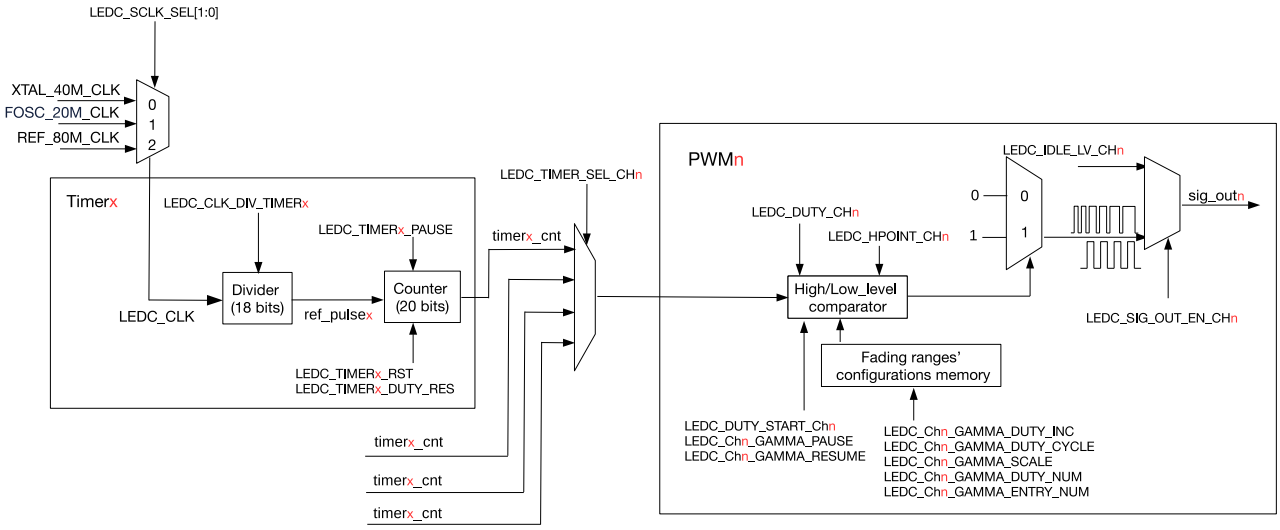


Figure 40.3-2. Timer and PWM Generator Block Diagram

40.4 Functional Description

40.4.1 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 40.3-2, this clock signal used by the timebase counter is named ref_pulse*x*. All timers use the same clock source LEDC_CLK, which is then passed through a clock divider to generate ref_pulse*x* for the counter.

40.4.1.1 Clock Source

LED PWM registers configured by software are clocked by APB_CLK. To use the LED PWM peripheral, the APB_CLK signal going to the LED PWM has to be enabled. The APB_CLK signal to LED PWM can be enabled by setting the HP_SYS_CLKRST_LEDC_APB_CLK_EN field in the HP_SYS_CLKRST_SOC_CLK_CTRL3_REG register. The LEDC_CLK signal to LED PWM can be enabled by setting the HP_SYS_CLKRST_LEDC_CLK_EN field in the HP_SYS_CLKRST_PERI_CLK_CTRL22_REG register. The LED PWM peripheral can be reset via software by setting the HP_SYS_CLKRST_RST_EN_LEDC field in the HP_SYS_CLKRST_HP_RST_EN1_REG register.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: XTAL_40M_CLK, FOSC_20M_CLK, and REF_80M_CLK. The procedure for selecting a clock source signal for LEDC_CLK is described below:

- XTAL_40M_CLK: Set HP_SYS_CLKRST_LEDC_CLK_SRC_SEL[1:0] to 0
- FOSC_20M_CLK: Set HP_SYS_CLKRST_LEDC_CLK_SRC_SEL[1:0] to 1
- REF_80M_CLK: Set HP_SYS_CLKRST_LEDC_CLK_SRC_SEL[1:0] to 2

The LEDC_CLK signal will then be passed through the clock divider.

For more information, please refer to Chapter 8 *Reset and Clock*.

40.4.1.2 Clock Divider Configuration

The LEDC_CLK signal is passed through a clock divider to generate the ref_pulse*x* signal for the counter. The frequency of ref_pulse*x* is equal to the frequency of LEDC_CLK divided by the divisor LEDC_CLK_DIV (see Figure 40.3-2).

The clock divider is a fractional divider. Thus, the divisor LEDC_CLK_DIV can be non-integer values. LEDC_CLK_DIV is configured according to the following equation.

$$\text{LEDC\_CLK\_DIV} = A + \frac{B}{256}$$

- The integer part *A* corresponds to the most significant 10 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[22:13])
- The fractional part *B* corresponds to the least significant 8 bits of LEDC_CLK_DIV_TIMER*x* (i.e., LEDC_TIMER*x*_CONF_REG[12:5])

When the fractional part *B* is 0, LEDC_CLK_DIV is equivalent to an integer divisor (i.e., an integer prescaler). In other words, a ref_pulse*x* clock pulse is generated after every *A* number of LEDC_CLK clock pulses.

However, when *B* is not 0, LEDC_CLK_DIV becomes a non-integer divisor. The clock divider implements non-integer frequency division by alternating between *A* and (*A*+1) LEDC_CLK clock pulses per ref_pulse*x* clock pulse. In this way, the average frequency of ref_pulse*x* clock pulse will be the desired frequency (i.e., the non-integer divided frequency). For every 256 ref_pulse*x* clock pulses:

- A number of *B* ref_pulse*x* clock pulses are generated every (*A*+1) LEDC_CLK clock pulses
- A number of (256-*B*) ref_pulse*x* clock pulses are generated every *A* LEDC_CLK clock pulses
- The ref_pulse*x* clock pulses generated every (*A*+1) pulses are evenly distributed amongst those generated every *A* pulses

Figure 40.4-1 illustrates the relation between LEDC_CLK clock pulses and ref_pulse*x* clock pulses when LEDC_CLK_DIV is a non-integer value.

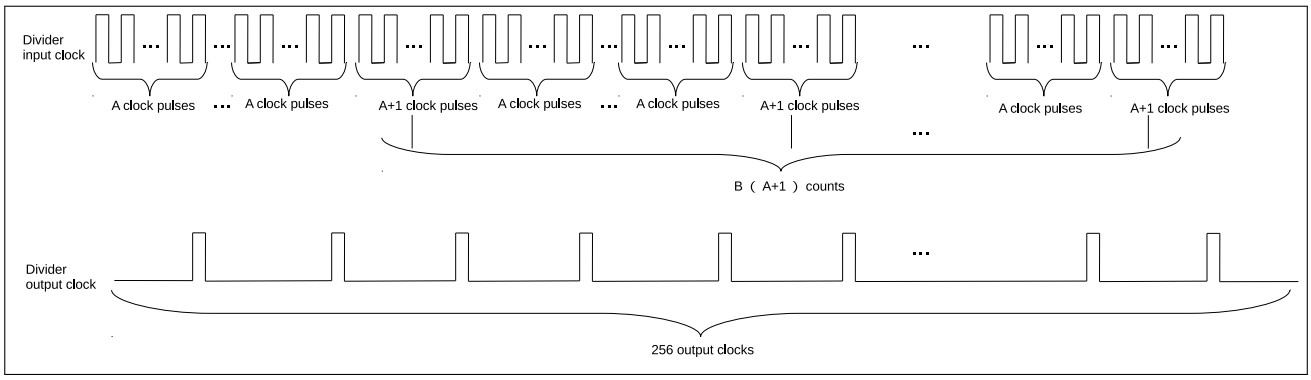


Figure 40.4-1. Frequency Division When LEDC_CLK_DIV is a Non-Integer Value

To change the timer's clock divisor at runtime, first configure the LEDC_CLK_DIV_TIMER*x* field, and then set the LEDC_TIMER*x*_PARA_UP field to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. The LEDC_TIMER*x*_PARA_UP field will be automatically cleared by hardware.

40.4.1.3 20-Bit Counter

Each timer contains a 20-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 40.3-2). 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 40.4-2 shows the relationship between the counter and PWM resolution.

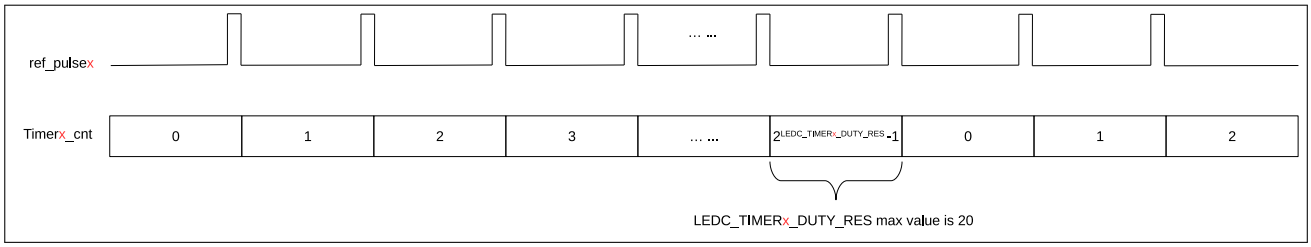


Figure 40.4-2. Relationship Between Counter And Resolution

Every time the counter overflows, it can trigger the LEDC_TIMER*x*_OVF_INT interrupt (generated automatically by hardware without configuration). It can also be configured to trigger LEDC_OVF_CNT_CH*n*_INT interrupt after overflowing LEDC_OVF_NUM_CH*n* + 1 times. To configure LEDC_OVF_CNT_CH*n*_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CH*n* to select the timer for the PWM generator
2. Enable the overflow counter by setting LEDC_OVF_CNT_EN_CH*n*
3. Configure LEDC_OVF_NUM_CH*n* with the number of counter overflows (that triggers an interrupt) minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CH*n*_INT_ENA
5. Configure LEDC_TIMER*x*_DUTY_RES to specify the counter bit width for the selected Timer*x* and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP or LEDC_PARA_UP_CH*n*. LEDC_TIMER*x*_PARA_UP and LEDC_PARA_UP_CH*n* will be automatically cleared by hardware.

Referring to Figure 40.3-2, 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 40.4-1 lists the commonly-used frequencies and their corresponding resolutions.

Table 40.4-1. Commonly-used Frequencies and Resolutions

LEDC_CLK	PWM Frequency	Highest Resolution (bit) [1]	Lowest Resolution (bit) [2]
REF_80M_CLK (80 MHz)	1 kHz	16	6
REF_80M_CLK (80 MHz)	5 kHz	13	3
REF_80M_CLK (80 MHz)	10 kHz	12	2
XTAL_40M_CLK (40 MHz)	1 kHz	15	5
XTAL_40M_CLK (40 MHz)	4 kHz	13	3
FOSC_20M_CLK (20 MHz)	1 kHz	14	4
FOSC_20M_CLK (20 MHz)	2 kHz	13	3

[1] The highest resolution is calculated when the clock divisor LEDC_CLK_DIV is 1. If the highest resolution calculated by the formula is higher than the counter's width 20 bits, then the highest resolution should be 20 bits.

[2] The lowest resolution is calculated when the clock divisor LEDC_CLK_DIV is $1023 + \frac{255}{256}$. If the lowest resolution calculated by the formula is lower than 0, then the lowest resolution should be 1.

40.4.2 PWM Generators

To generate a PWM signal, a PWM generator (PWM*n*) needs a timer (Timer*x*). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CH*n* to use one of four timers to generate the PWM output.

As shown in Figure 40.3-2, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 20-bit counter value (Timer*x*_cnt) to two trigger values Hpoint*n* and Lpoint*n*. When the timer's counter value is equal to Hpoint*n* or Lpoint*n*, the PWM signal is high or low, respectively, as described below:

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Figure 40.4-3 illustrates how Hpoint*n* and Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

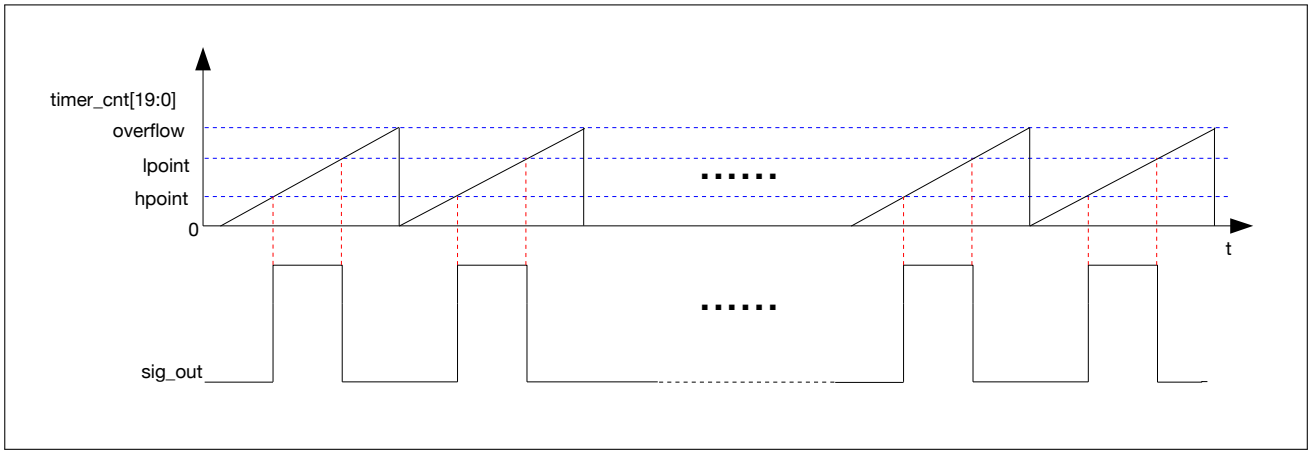


Figure 40.4-3. LED PWM Output Signal Diagram

For a particular PWM generator (PWM*n*), its Hpoint*n* is sampled from the LEDC_HPOINT_CH*n* field each time the selected timer's counter overflows. Likewise, Lpoint*n* is also sampled on every counter overflow and is calculated from the sum of the LEDC_DUTY_CH*n*[24:4] and LEDC_HPOINT_CH*n* fields. By setting Hpoint*n* and Lpoint*n* via the LEDC_HPOINT_CH*n* and LEDC_DUTY_CH*n*[24:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig_out*n*) is enabled by setting LEDC_SIG_OUT_EN_CH*n*. When LEDC_SIG_OUT_EN_CH*n* is cleared, PWM signal output is disabled, and the output signal (sig_out*n*) will output a constant level specified by LEDC_IDLE_LV_CH*n*.

The bits LEDC_DUTY_CH*n*[3:0] are used to dither the duty cycles of the PWM output signal (sig_out*n*) by periodically altering the duty cycle of sig_out*n*. When LEDC_DUTY_CH*n*[3:0] is not 0, then for every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC_DUTY_CH*n*[3:0]) cycles. For instance, if LEDC_DUTY_CH*n*[24:4] is set to 10 and LEDC_DUTY_CH*n*[3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC_TIMER_SEL_CH*n*, LEDC_HPOINT_CH*n*, LEDC_DUTY_CH*n*[24:4], and LEDC_SIG_OUT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

40.4.3 Duty Cycle Fading

The PWM generators can fade the duty cycle of a PWM output signal (i.e., gradually change the duty cycle from one value to another). Each PWM generator can have up to 16 duty cycle ranges, which can be independently configured in terms of fading direction (increase or decrease), fading amount, the number of fades, and fading frequency. If Duty Cycle Fading is enabled, every range's Lpoint*n* value will change

according to its fading configuration.

40.4.3.1 Linear Duty Cycle Fading

Linear fading PWM signals can be generated by configuring the direction, fading amount, the number of fades, and fading frequency of the first duty cycle range.

The programming procedures will be described in detail in the section 40.6.

After the procedures, the PWM generator can fade the duty cycle of a PWM signal once per LEDC_CH*n*_GAMMA_DUTY_CYCLE times of counter overflows. Every time when the PWM signal is faded, Lpoint*n* increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by LEDC_CH*n*_GAMMA_SCALE, and the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}$$

The duty cycle is faded for LEDC_CH*n*_GAMMA_DUTY_NUM times. After that, the PWM generator stops fading and keeps outputting signals at this duty cycle. Upon each fading the duty cycle increases or decreases by the same amount, and therefore the PWM signal is a linear fading signal.

Figure 40.4-4 shows a linear fading PWM signal.

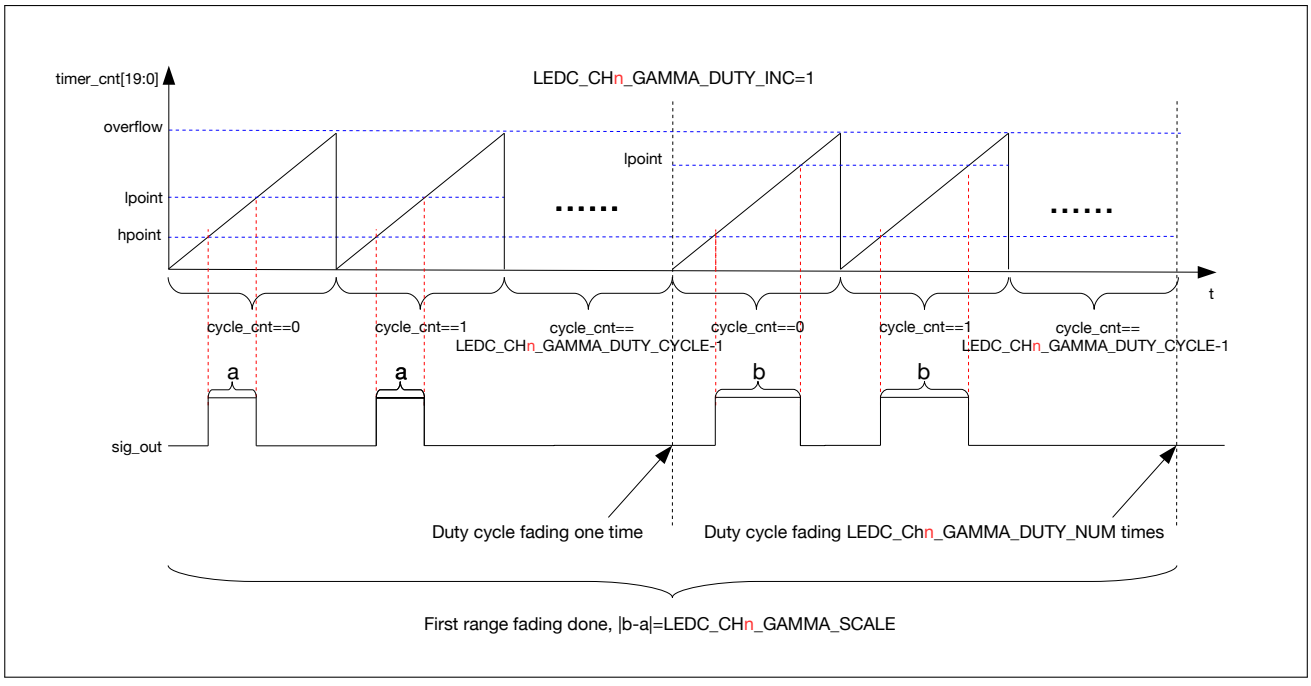


Figure 40.4-4. Output Signal of Linear Duty Cycle Fading

40.4.3.2 Gamma Curve Fading

Gamma curve fading PWM signals can be generated by configuring the fading direction, fading amount, the number of fades and fading frequency of multiple duty cycle fading ranges.

The programming procedures will be described in detail in the section 40.6.

After the procedures, the PWM generator can generate a PWM signal with LEDC_CH*n*_GAMMA_ENTRY_NUM ranges. The duty cycle of the PWM signal fades according to the configurations of range 0 first, and then range 1, till range (LEDC_CH*n*_GAMMA_ENTRY_NUM − 1) (the last range) where Duty Cycle Fading ends. The PWM signal fades independently in each range. According to the configuration of each range, every time when the counter overflows for LEDC_CH*n*_GAMMA_DUTY_CYCLE times, Lpoint*n* increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by LEDC_CH*n*_GAMMA_SCALE, and accordingly the duty cycle increases or decreases (configured by LEDC_CH*n*_GAMMA_DUTY_INC) by

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}$$

After the duty cycle fades for LEDC_CH*n*_GAMMA_DUTY_NUM times in a range, Duty Cycle Fading in this range finishes.

When Duty Cycle Fading finishes in all ranges (the number of ranges is specified by LEDC_CH*n*_GAMMA_ENTRY_NUM), the PWM signal stops fading and keeps the duty cycle of the last fade. Given that the duty cycle fades differently and linearly in each range, several linear fading ranges would be fitted to a gamma curve.

Figure 40.4-5 illustrates a gamma curve fading PWM signal.

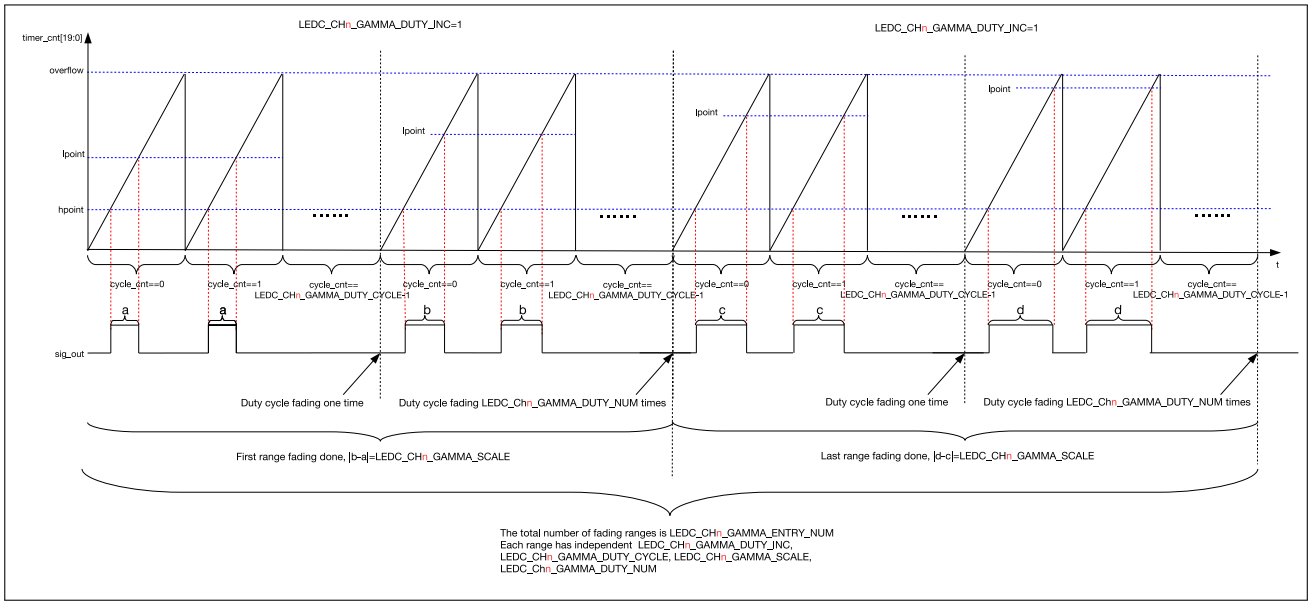


Figure 40.4-5. Output Signal of Gamma Curve Fading

40.4.3.3 Suspend and Resume Duty Cycle Fading

To suspend Duty Cycle Fading that has already been started, write 1 to the LEDC_CH*n*_GAMMA_PAUSE field of the LEDC_CH*n*_GAMMA_CONF_REG register. Once LEDC_CH*n*_GAMMA_PAUSE is set to 1, the PWM signal keeps the duty cycle of the most recent fade.

To resume Duty Cycle Fading, write 1 to the LEDC_CH*n*_GAMMA_RESUME field of the LEDC_CH*n*_GAMMA_CONF_REG register. Once LEDC_CH*n*_GAMMA_RESUME is set to 1, the PWM signal resumes fading from the range where the suspension occurs, until fading in the last range finishes. The fading will continue from the state when it was paused until all the ranges complete duty cycle fading (when

LEDC_CH*n*_GAMMA_RESUME is set to 1, LEDC_CH*n*_GAMMA_PAUSE is cleared automatically by hardware.

40.4.4 Event Task Matrix Feature

The LEDC on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows LEDC's ETM tasks to be triggered by any peripherals' ETM events, or LEDC's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to LEDC. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

ETM-related events and tasks are enabled by configuring corresponding fields of LEDC_EVT_TASK_EN0_REG, LEDC_EVT_TASK_EN1_REG and LEDC_EVT_TASK_EN2_REG registers. For the correspondence between events, tasks, and fields, Please refer to Section 40.8).

LEDC can receive the following ETM tasks:

- LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*: If the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task, PWM*n* generates fading PWM signals according to the newly configured LEDC_CH*n*_GAMMA_SCALE field.
- LEDC_TASK_TIMER*x*_RES_UPDATE: If the LEDC_TASK_TIMER*x*_RES_UPDATE_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RES_UPDATE task, Timer*x* updates its counter's overflow value to the value configured in the LEDC_TIMER*x*_DUTY_RES field at the next overflow of the counter.
- LEDC_TASK_TIMER*x*_CAP: If the LEDC_TASK_TIMER*x*_CAP_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_CAP task, Timer*x* captures its counter's value, and stores the value into the LEDC_TIMER*x*_CNT_CAP field of register LEDC_TIMER*x*_CNT_CAP_REG.
- LEDC_TASK_SIG_OUT_DIS_CH*n*: If the LEDC_TASK_SIG_OUT_DIS_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_SIG_OUT_DIS_CH*n* task, PWM*n*'s signal output is disabled, and the output signal (sig_out*n*) outputs a constant level as specified by field LEDC_IDLE_LV_CH*n*, as shown in Figure 40.3-2.
- LEDC_TASK_OVF_CNT_RST_CH*n*: If the LEDC_TASK_OVF_CNT_RST_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_OVF_CNT_RST_CH*n* task, PWM*n* timer's overflow counter is reset to 0.
- LEDC_TASK_TIMER*x*_RST: If the LEDC_TASK_TIMER*x*_RST_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RST task, Timer*x*'s counter is reset to 0.
- LEDC_TASK_TIMER*x*_RESUME and LEDC_TASK_TIMER*x*_PAUSE: If the LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN field is enabled, upon receiving the LEDC_TASK_TIMER*x*_RESUME and LEDC_TASK_TIMER*x*_PAUSE task, Timer*x* is suspended and resumed alternately. That is, when the task is received, Timer*x* is paused; and when the task is received again, Timer*x* is resumed.
- LEDC_TASK_GAMMA_RESTART_CH*n*: If the LEDC_TASK_GAMMA_RESTART_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_RESTART_CH*n* task, the PWM*n* restarts to generate the fading PWM signal.
- LEDC_TASK_GAMMA_PAUSE_CH*n*: If the LEDC_TASK_GAMMA_PAUSE_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_PAUSE_CH*n* task, PWM*n* suspends Duty Cycle Fading at the next timer overflow. That is, after the task has been received, PWM*n* keeps the duty cycle of the last fade.

- LEDC_TASK_GAMMA_RESUME_CH*n*: If the LEDC_TASK_GAMMA_RESUME_CH*n*_EN field is enabled, upon receiving the LEDC_TASK_GAMMA_RESUME_CH*n* task, PWM*n* resumes Duty Cycle Fading at the next counter overflow. That is, after the task has been received, PWM*n* resumes fading from the range where the suspension occurs.

LEDC can generate the following ETM events:

- LEDC_EVT_DUTY_CHNG_END_CH*n*: Generated when the LEDC_EVT_DUTY_CHNG_END_CH*n*_EN field is enabled, and PWM*n* has finished Duty Cycle Fading.
- LEDC_EVT_OVF_CNT_PLS_CH*n*: Generated when the LEDC_EVT_OVF_CNT_PLS_CH*n*_EN field is enabled and when PWM*n* timer's counter overflows for LEDC_OVF_NUM_CH*n* + 1 times.
- LEDC_EVT_TIME_OVF_TIMER*x*: Generated when the LEDC_EVT_TIME_OVF_TIMER*x*_EN field is enabled and Timer*x*'s counter overflows.
- LEDC_EVT_TIMER*x*_CMP: Generated when the LEDC_EVT_TIMER*x*_CMP_EN field is enabled and the value of Timer*x*)'s counter reaches that of the LEDC_TIMER*x*_CMP field of register LEDC_TIMER*x*_CMP_REG.

In practical applications, LEDC's ETM events can trigger its own ETM tasks. For example, LEDC_EVT_DUTY_CHNG_END_CH*n* event can trigger the LEDC_TASK_GAMMA_RESTART_CH*n* task, thus starting the next fading directly after the current fading is completed.

40.5 Interrupts

ESP32-P4's LEDC can generate the LEDC_INT interrupt signal, which will be sent to the Interrupt Matrix.

Interrupt signal LEDC_INT can be generated by the following internal interrupt sources:

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for LEDC_OVF_NUM_CH*n* + 1 times and the register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on PWM*n* has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when Timer*x* has reached its maximum counter value.

The above interrupt sources can only be triggered when the interrupt enable register *XXX*_ENA is set to 1. Write 1 to *XXX*_CLR will clear the interrupt (*XXX* is the interrupt source name).

40.6 Programming Procedures

The configuration process of the LED PWM controller to generate PWM signals is as follows:

1. Configure the Timer*x* according to section 40.4.1. After that, the clock source, divider, and counter are ready.
2. Configure the PWM*n* generator according to section 40.4.2. After that, PWM*n* selects one of the four timers and the phase and duty cycle of the PWM*n* output is set.
3. Choose the corresponding configuration process based on the different duty cycle fading types:
   (a) Linear duty cycle fading:
      i. Configure the LEDC_DUTY_CH*n* field with the initial value of Lpoint*n*.

ii. Set the LEDC_DUTY_START_CH*n* field to enable Duty Cycle Fading. When this field is cleared, Duty Cycle Fading will be disabled.

iii. Write the linear duty cycle fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4), the LEDC_CH*n*_MEM starting address is provided in Table 40.7-1. Please note that the following bit names are just for easier description, and there are no such register fields.

- Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increase or decrease in the current configured range.
- Bit 1 to 10 of the configuration data is used to configure the number of times the counter overflows per an increment or decrement of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
- Bit 11 to 20 of the configuration data is used to configure the amount by which Lpoint*n* increase or decrease in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
- Bit 21 to 30 of the configuration data is used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
- The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For linear duty cycle fading only the first range needs to be configured, so the duty cycle range number should be configured as 0.

iv. Configure the number of ranges per each fading (1 in this case) via the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG. Once the specified number of ranges have been faded, Duty Cycle Fading stops and the PWM generator triggers the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. For linear duty cycle fading there is only one duty cycle range (i.e., the first one), so configure LEDC_CH*n*_GAMMA_ENTRY_NUM as 1.

v. Set the LEDC_PARA_UP_CH*n* field to apply the above configurations. After this field is set, the configurations for Duty Cycle Fading will take effect upon the next overflow of the counter, and the PWM generator will output a linear fading PWM signal following configurations. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

(b) Gamma curve fading:

i. The same as Step 1 in Section 40.4.3.1.

ii. The same as Step 2 in Section 40.4.3.1.

iii. Configure multiple duty cycle ranges with the following steps. Write the gamma curve fading configuration to (LEDC_CH*n*_MEM starting address + duty cycle range number * 4), the LEDC_CH*n*_MEM starting address is provided in Table 40.7-1.

- Bit 0 of the configuration data is used to configure the direction (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_INC). When it is set or cleared, the Lpoint*n* will increment or decrement in the current configured range.

- Bit 1 to 10 of the configuration data is used to configure the number of times the counter overflows per an increase or decrease of Lpoint*n* (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_CYCLE). In other words, Lpoint*n* will increase or decrease after the counter overflows for the configured number of times.
- Bit 11 to 20 of the configuration data is used to configure the amount by which Lpoint*n* increase or decrease in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_SCALE).
- Bit 21 to 30 of the configuration data is used to configure the number of fades in the current configured range (hereafter referred to as LEDC_CH*n*_GAMMA_DUTY_NUM).
- The duty cycle range number (from 0 to 15) specifies to which range the above configuration data apply. For gamma curve fading, it must start from 0 and increase by 1 for the next range to be configured.
- Once the above procedures are finished, the configuration for one range is complete. Other ranges are configured by repeating the same set of procedures. You can configure any number of ranges from 0 to 15, and each can be configured independently.

iv. After all required ranges are configured, write the total number of ranges configured in Step 3 to the LEDC_CH*n*_GAMMA_ENTRY_NUM field of the LEDC_CH*n*_GAMMA_CONF_REG register.

v. Set the LEDC_PARA_UP_CH*n* field to apply the above configuration. After this field is set, the configurations for duty cycle fading will take effect upon the next overflow of the counter, and the PWM generator will output a gamma curve fading PWM signal following the configurations. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

After the above procedures, the LED PWM controller will generate the desired PWM signals.

At any time, duty cycle fading can be suspended or resumed, more detials can be found in setction 40.4.3.3. If the duty cycle fading configurations need to be changed before the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has not been triggered), the duty cycle fading needs to be suspended before changing the configurations. However, if the duty cycle fading configurations need to be changed after the PWM signals stop fading (that is, the LEDC_DUTY_CHNG_END_CH*n*_INT has been triggered), the configurations can be changed directly.

At any time, PWM signal output can be enabled or disabled by software, and the output signal level can be changed, more details can be found in section 40.4.2.

40.7 Memory Blocks

Each LEDC channel has a memory to store duty cycle fading configuration data. Each memory can store configuration data for up to 16 ranges. The addresses in this section are relative to LED PWM Controller base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

Table 40.7-1. LEDC Memory Blocks

Name	Description	Size	Starting Address	Ending Address	Access
LEDC_CH*n*_MEM	Memory of PWM*n*	64 B	0x400 + PWM*n* * 64	0x400 + (PWM*n* + 1) * 64	WR

40.8 Register Summary

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

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

Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH0_DUTY_REG	Initial duty cycle register for channel 0	0x0008	R/W
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W/SC
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH1_DUTY_REG	Initial duty cycle register for channel 1	0x001C	R/W
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	R/W/SC
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH2_DUTY_REG	Initial duty cycle register for channel 2	0x0030	R/W
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	R/W/SC
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH3_DUTY_REG	Initial duty cycle register for channel 3	0x0044	R/W
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	R/W/SC
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH4_DUTY_REG	Initial duty cycle register for channel 4	0x0058	R/W
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	R/W/SC
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
LEDC_CH5_DUTY_REG	Initial duty cycle register for channel 5	0x006C	R/W
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W/SC
LEDC_CH6_CONF0_REG	Configuration register 0 for channel 6	0x0078	varies
LEDC_CH6_HPOINT_REG	High point register for channel 6	0x007C	R/W
LEDC_CH6_DUTY_REG	Initial duty cycle register for channel 6	0x0080	R/W
LEDC_CH6_CONF1_REG	Configuration register 1 for channel 6	0x0084	R/W/SC
LEDC_CH7_CONF0_REG	Configuration register 0 for channel 7	0x008C	varies
LEDC_CH7_HPOINT_REG	High point register for channel 7	0x0090	R/W
LEDC_CH7_DUTY_REG	Initial duty cycle register for channel 7	0x0094	R/W
LEDC_CH7_CONF1_REG	Configuration register 1 for channel 7	0x0098	R/W/SC
LEDC_TIMER0_CONF_REG	Timer 0 configuration register	0x00A0	varies
LEDC_TIMER1_CONF_REG	Timer 1 configuration register	0x00A8	varies
LEDC_TIMER2_CONF_REG	Timer 2 configuration register	0x00B0	varies
LEDC_TIMER3_CONF_REG	Timer 3 configuration register	0x00B8	varies

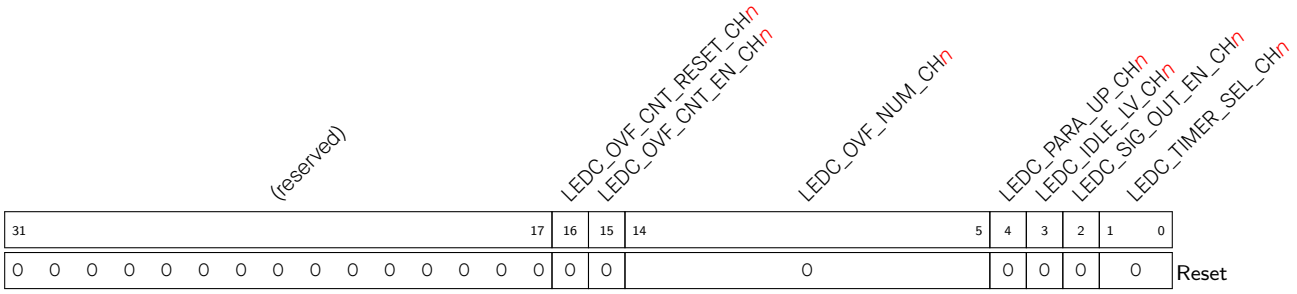
Name	Description	Address	Access
LEDC_CH0_GAMMA_CONF_REG	LEDC channel 0 gamma config register	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	LEDC channel 1 gamma config register	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	LEDC channel 2 gamma config register	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	LEDC channel 3 gamma config register	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	LEDC channel 4 gamma config register	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	LEDC channel 5 gamma config register	0x0114	varies
LEDC_CH6_GAMMA_CONF_REG	LEDC channel 6 gamma config register	0x0118	varies
LEDC_CH7_GAMMA_CONF_REG	LEDC channel 7 gamma config register	0x011C	varies
LEDC_EVT_TASK_EN0_REG	LEDC event task enable bit register 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC event task enable bit register 1	0x0124	R/W
LEDC_EVT_TASK_EN2_REG	LEDC event task enable bit register 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	LEDC timer 0 compare value register	0x0140	R/W
LEDC_TIMER1_CMP_REG	LEDC timer 1 compare value register	0x0144	R/W
LEDC_TIMER2_CMP_REG	LEDC timer 2 compare value register	0x0148	R/W
LEDC_TIMER3_CMP_REG	LEDC timer 3 compare value register	0x014C	R/W
LEDC_CONF_REG	LEDC global configuration register	0x0170	R/W
Status Register			
LEDC_CH0_DUTY_R_REG	Current duty cycle register for channel 0	0x0010	RO
LEDC_CH1_DUTY_R_REG	Current duty cycle register for channel 1	0x0024	RO
LEDC_CH2_DUTY_R_REG	Current duty cycle register for channel 2	0x0038	RO
LEDC_CH3_DUTY_R_REG	Current duty cycle register for channel 3	0x004C	RO
LEDC_CH4_DUTY_R_REG	Current duty cycle register for channel 4	0x0060	RO
LEDC_CH5_DUTY_R_REG	Current duty cycle register for channel 5	0x0074	RO
LEDC_CH6_DUTY_R_REG	Current duty cycle register for channel 6	0x0088	RO
LEDC_CH7_DUTY_R_REG	Current duty cycle register for channel 7	0x009C	RO
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value register	0x00A4	RO
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value register	0x00AC	RO
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value register	0x00B4	RO
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value register	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	LEDC timer 0 captured count value register	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	LEDC timer 1 captured count value register	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	LEDC timer 2 captured count value register	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	LEDC timer 3 captured count value register	0x015C	RO
Interrupt Register			
LEDC_INT_RAW_REG	Interrupt raw status register	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	Interrupt masked status register	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable register	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear register	0x00CC	WT
Version Register			
LEDC_DATE_REG	Version control register	0x0174	R/W

40.9 Registers

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

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

Register 40.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* Configures which timer is channel *n* selected.

0: Select Timer 0

1: Select Timer 1

2: Select Timer 2

3: Select Timer 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* Configures whether to enable signal output on channel *n*.

0: Signal output disable

1: Signal output enable

(R/W)

LEDC_IDLE_LV_CH*n* Configures the output value when channel *n* is inactive. Valid only when LEDC_SIG_OUT_EN_CH*n* is 0.

0: Output level is low

1: Output level is high

(R/W)

LEDC_PARA_UP_CH*n* Configures whether to update LEDC_HPOINT_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_SIG_OUT_EN_CH*n*, LEDC_TIMER_SEL_CH*n*, LEDC_OVF_CNT_EN_CH*n* fields and duty cycle range configurations for channel *n*, and will be automatically cleared by hardware.

0: Invalid. No effect

1: Update

(WT)

LEDC_OVF_NUM_CH*n* Configures the maximum times of overflow minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* Configures whether to enable the ovf_cnt of channel *n*.

0: Disable

1: Enable

(R/W)

Continued on the next page...

Register 40.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14**n*)

Continued from the previous page...

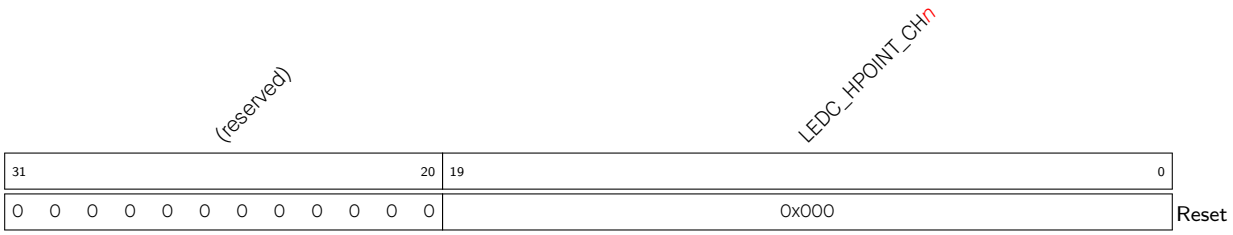
LEDC_OVF_CNT_RESET_CH*n* Configures whether to reset the ovf_cnt of channel *n*.

0: Invalid. No effect

1: Reset the ovf_cnt

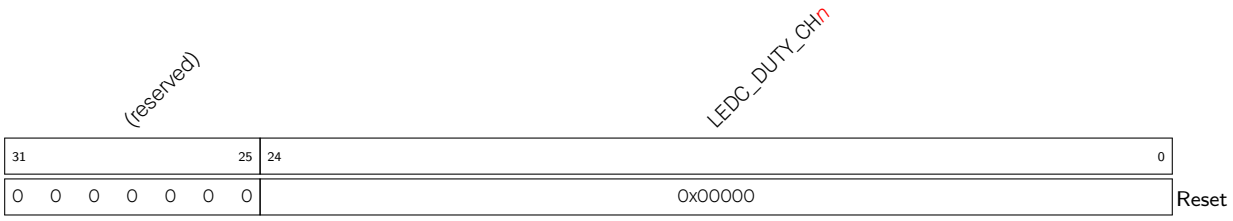
(WT)

Register 40.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+0x14**n*)



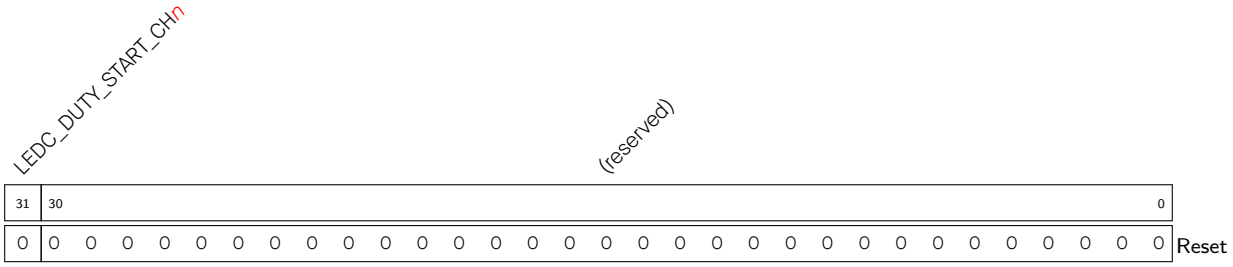
LEDC_HPOINT_CH*n* Configures high point of signal output on channel *n*. The output value changes to high when the selected timers have reached the value specified by this register. (R/W)

Register 40.3. 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 40.4. LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+0x14**n*)



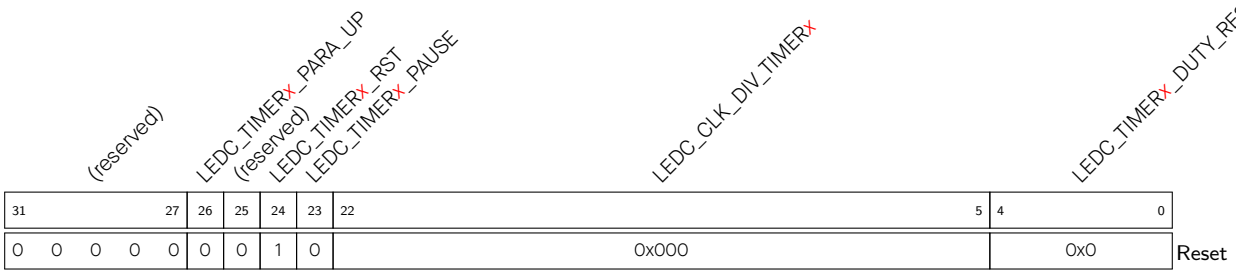
LEDC_DUTY_START_CH*n* Configures whether the duty cycle fading configurations take effect.

0: Not take effect

1: Take effect

(R/W/SC)

Register 40.5. LEDC_TIMER*n*_CONF_REG (*n*: 0-3) (0x00A0+0x8**n*)



LEDC_TIMER*x*_DUTY_RES Configures the bit width of the counter in timer *x*. Valid values are 1 to 20. (R/W)

LEDC_CLK_DIV_TIMER*x* Configures the divisor for the divider in timer *x*.

The least significant eight bits represent the fractional part. The most significant ten bits represent the integer part. (R/W)

LEDC_TIMER*x*_PAUSE Configures whether to pause the counter in timer *x*.

0: Normal

1: Pause

(R/W)

LEDC_TIMER*x*_RST Configures whether to reset timer *x*. The counter will show 0 after reset.

0: Not reset

1: Reset

(R/W)

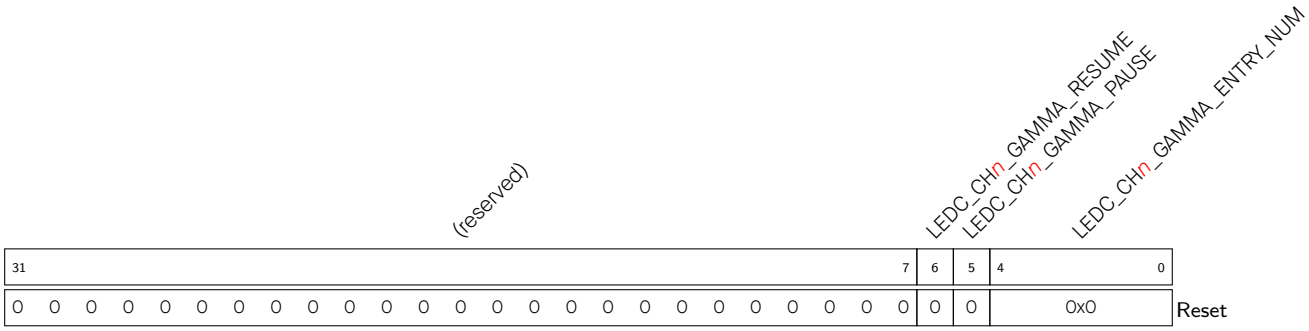
LEDC_TIMER*x*_PARA_UP Configures whether to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES.

0: Invalid. No effect

1: Update

(WT)

Register 40.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-7) (0x0100+0x4n*)**



LEDC_CH*n*_GAMMA_ENTRY_NUM Configures the number of duty cycle fading ranges for LEDC channel *n*. (R/W)

LEDC_CH*n*_GAMMA_PAUSE Configures whether to pause duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Pause

(WT)

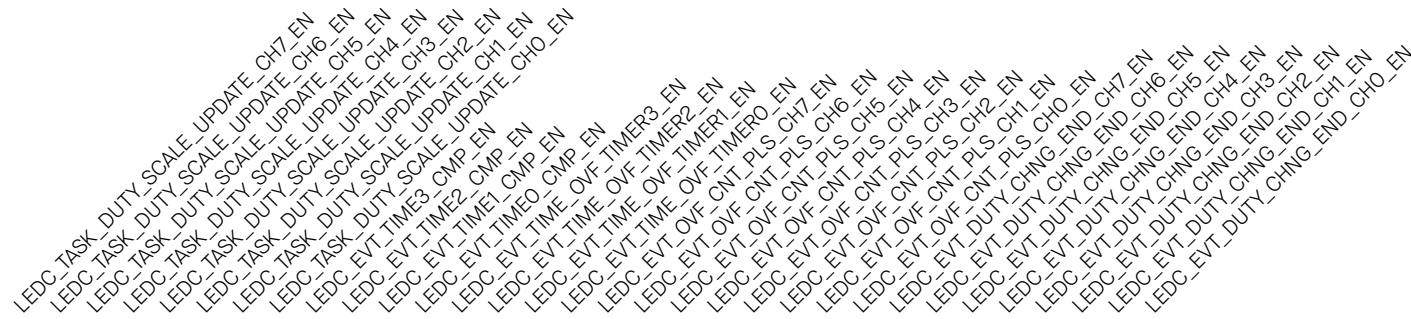
LEDC_CH*n*_GAMMA_RESUME Configures whether to resume duty cycle fading of LEDC channel *n*.

0: Invalid. No effect

1: Resume

(WT)

Register 40.7. LEDC_EVT_TASK_EN0_REG (0x0120)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_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)

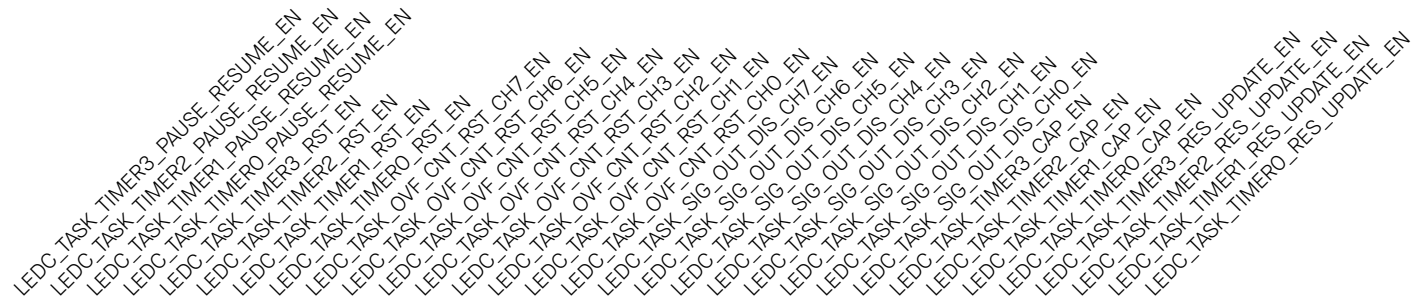
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-7) Configures whether to enable the LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* task.

0: Disable

1: Enable

(R/W)

Register 40.8. LEDC_EVT_TASK_EN1_REG (0x0124)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_TASK_TIMERx_RES_UPDATE_EN (x: 0-3) Configures whether to enable LEDC_TASK_TIMERx_RES_UPDATE task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMERx_CAP_EN (x: 0-3) Configures whether to enable LEDC_TASK_TIMERx_CAP task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_SIG_OUT_DIS_CHn_EN (n: 0-7) Configures whether to enable LEDC_TASK_SIG_OUT_DIS_CHn task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_OVF_CNT_RST_CHn_EN (n: 0-7) Configures whether to enable LEDC_TASK_OVF_CNT_RST_CHn task.

0: Disable

1: Enable

(R/W)

LEDC_TASK_TIMERx_RST_EN (x: 0-3) Configures whether to enable LEDC_TASK_TIMERx_RST task.

0: Disable

1: Enable

(R/W)

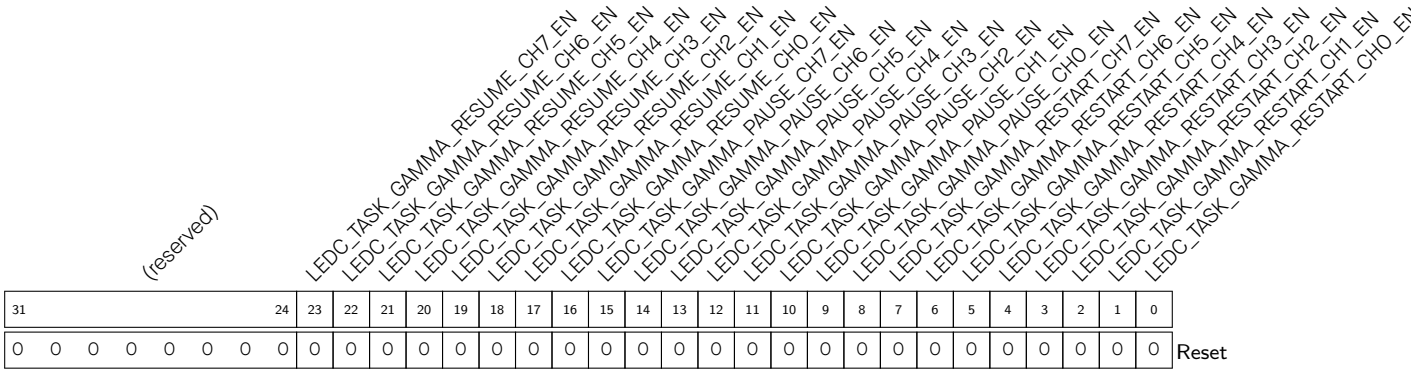
LEDC_TASK_TIMERx_PAUSE_RESUME_EN (x: 0-3) Configures whether to enable LEDC_TASK_TIMERx_PAUSE and LEDC_TASK_TIMERx_RESUME task.

0: Disable

1: Enable

(R/W)

Register 40.9. 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)

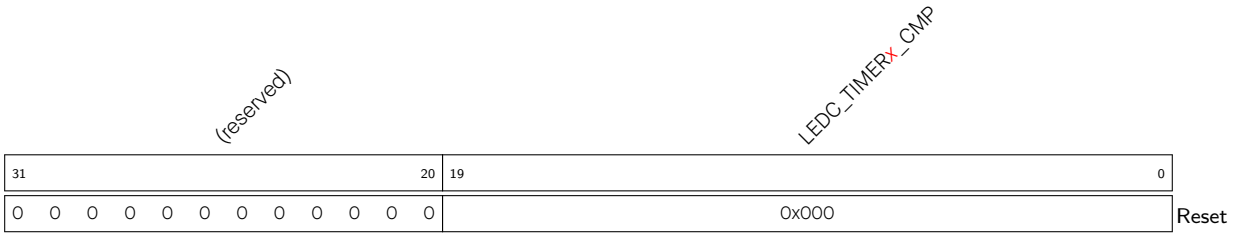
LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-7) Configures whether to enable LEDC_TASK_GAMMA_RESUME_CH*n* task.

0: Disable

1: Enable

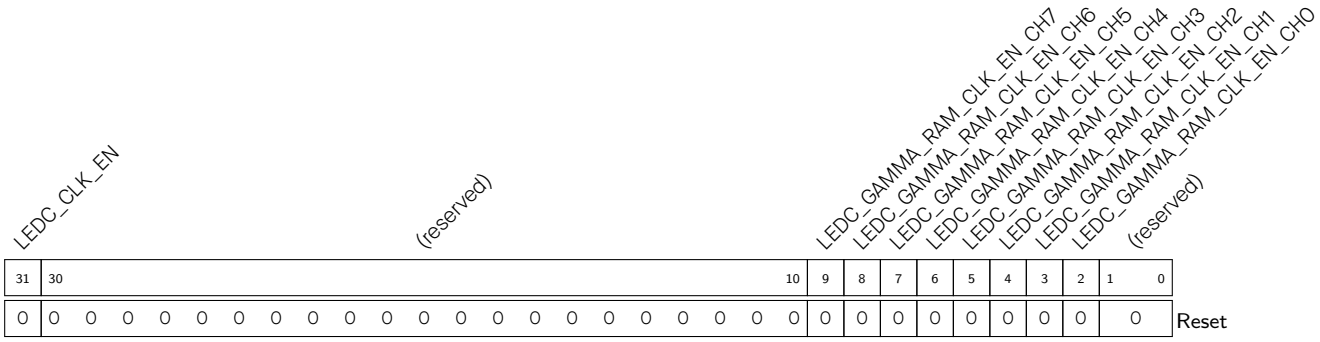
(R/W)

Register 40.10. LEDC_TIMER*n*_CMP_REG (*n*: 0-3) (0x0140+0x4*n)



LEDC_TIMER*x*_CMP Configures the comparison value for LEDC timer *x*. (R/W)

Register 40.11. LEDC_CONF_REG (0x0170)



LEDC_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-7) Configures whether to open LEDC channel *n* gamma RAM clock gate.

0: Open the clock gate only when an application writes or reads LEDC channel *n* gamma RAM

1: Force open the clock gate for LEDC channel *n* gamma RAM

(R/W)

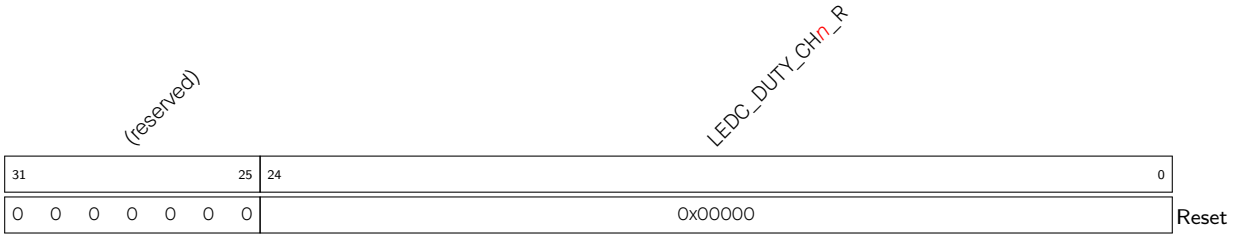
LEDC_CLK_EN Configures whether to open the register clock gate.

0: Open the clock gate only when an application writes registers

1: Force open the clock gate for register

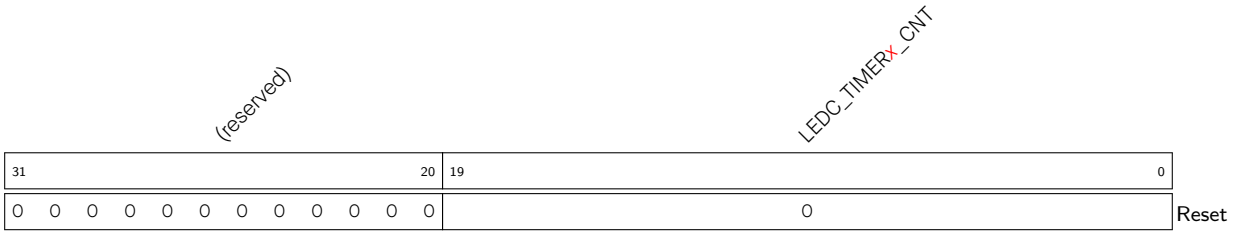
(R/W)

Register 40.12. LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+0x14*n)



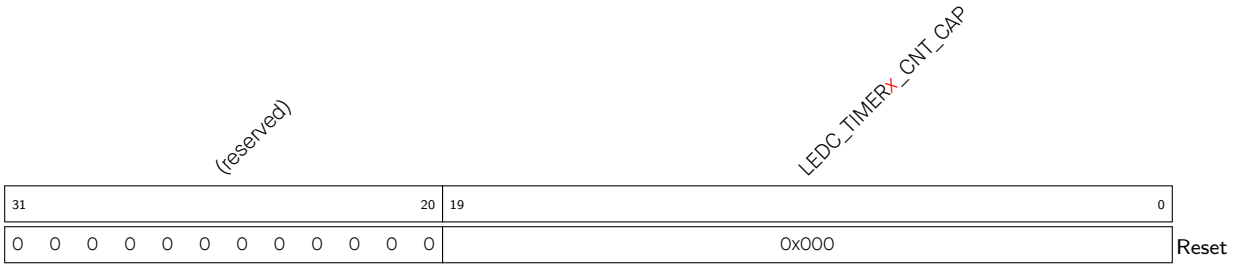
LEDC_DUTY_CH*n*_R Represents the current duty cycle of output signal on channel *n*. (RO)

Register 40.13. LEDC_TIMER*n*_VALUE_REG (*n*: 0-3) (0x00A4+0x8*n)



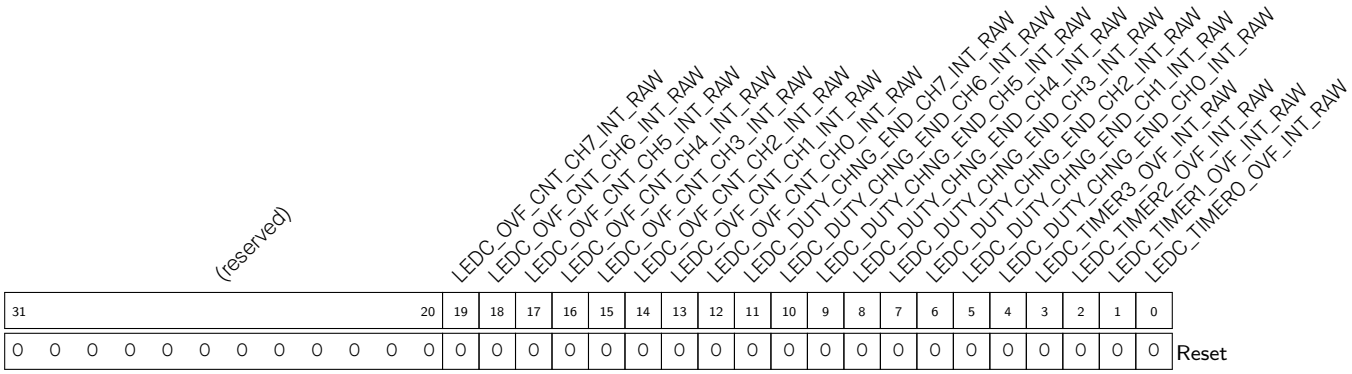
LEDC_TIMER*x*_CNT Represents the current counter value of timer *x*. (RO)

Register 40.14. LEDC_TIMER*n*_CNT_CAP_REG (*n*: 0-3) (0x0150+0x4n*)**



LEDC_TIMER*x*_CNT_CAP Represents the captured LEDC timer *x* count value. (RO)

Register 40.15. LEDC_INT_RAW_REG (0x00C0)

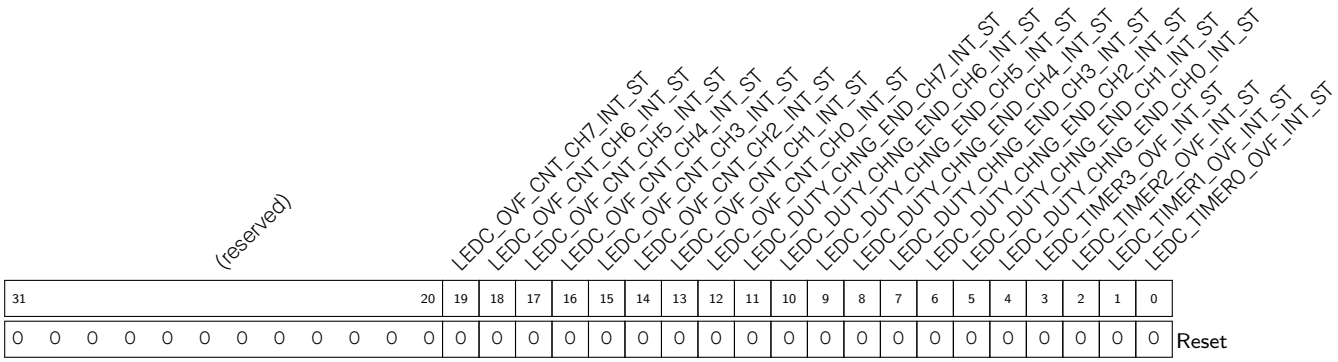


LEDC_TIMER*x*_OVF_INT_RAW (*x*: 0-3) Raw status bit: The raw interrupt status of LEDC_TIMER*x*_OVF_INT. Triggered when the timer *x* has reached its maximum counter value. (R/WTC/SS)

LEDC_DUTY_CHNG_END_CH*n*_INT_RAW (*n*: 0-7) Raw status bit: The raw interrupt status of LEDC_DUTY_CHNG_END_CH*n*_INT. Triggered when the fading of duty has finished. (R/WTC/SS)

LEDC_OVF_CNT_CH*n*_INT_RAW (*n*: 0-7) Raw status bit: The raw interrupt status of LEDC_OVF_CNT_CH*n*_INT. Triggered when the ovf_cnt has reached the value specified by LEDC_OVF_NUM_CH*n*. (R/WTC/SS)

Register 40.16. LEDC_INT_ST_REG (0x00C4)

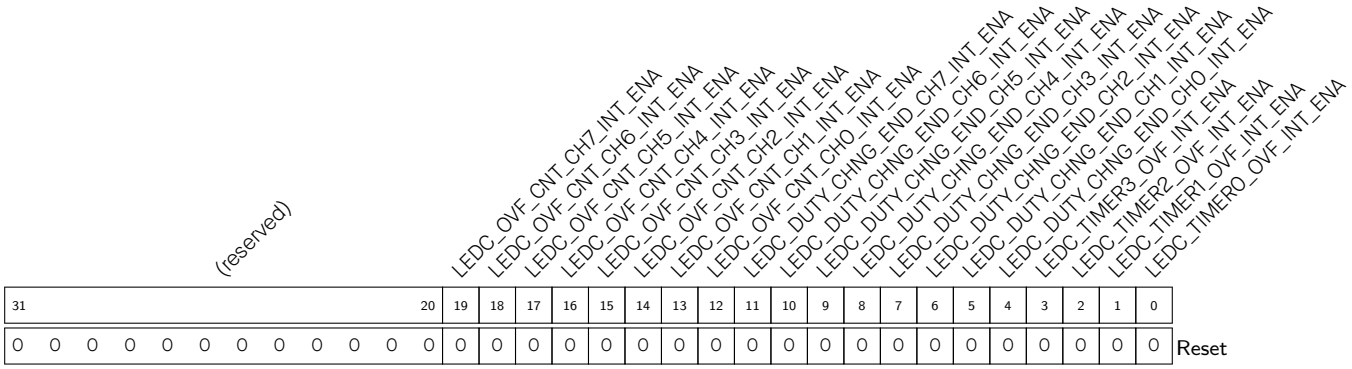


LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3) Masked status bit: The masked interrupt status of LEDC_TIMER*x*_OVF_INT. Valid only when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_DUTY_CHNG_END_CH*n*_INT. Valid only when LEDC_DUTY_CHNG_END_CH*n*_INT_ENA is set to 1. (RO)

LEDC_OVF_CNT_CH*n*_INT_ST (*n*: 0-7) Masked status bit: The masked interrupt status of LEDC_OVF_CNT_CH*n*_INT. Valid only when LEDC_OVF_CNT_CH*n*_INT_ENA is set to 1. (RO)

Register 40.17. LEDC_INT_ENA_REG (0x00C8)

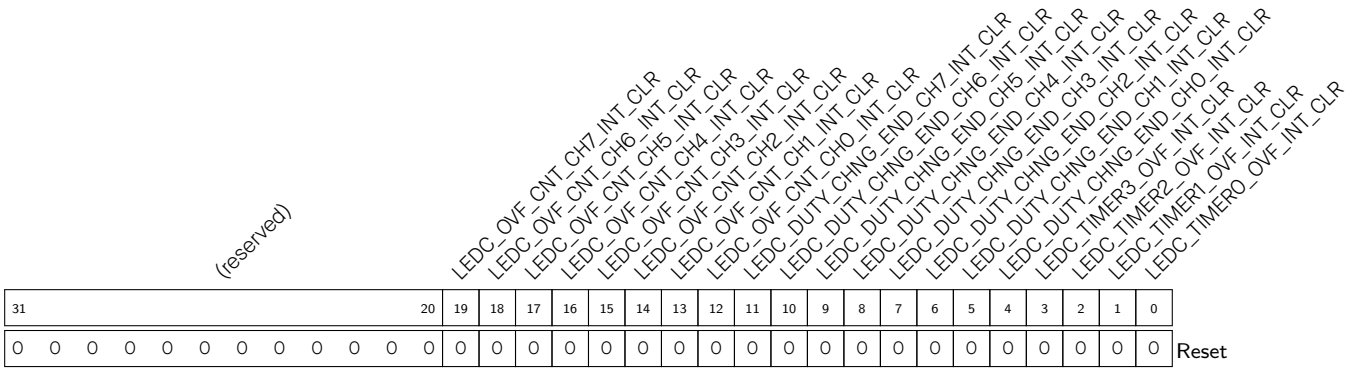


LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3) Enable bit: Write 1 to enable LEDC_TIMER*x*_OVF_INT. (R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_DUTY_CHNG_END_CH*n*_INT. (R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-7) Enable bit: Write 1 to enable LEDC_OVF_CNT_CH*n*_INT. (R/W)

Register 40.18. LEDC_INT_CLR_REG (0x00CC)

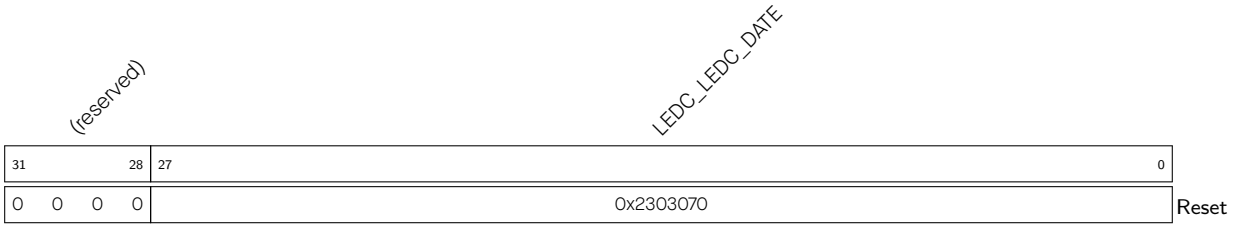


LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3) Clear bit: Write 1 to clear LEDC_TIMER*x*_OVF_INT. (WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_DUTY_CHNG_END_CH*n*_INT. (WT)

LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-7) Clear bit: Write 1 to clear LEDC_OVF_CNT_CH*n*_INT. (WT)

Register 40.19. LEDC_DATE_REG (0x0174)



LEDC_LEDC_DATE Configures the version. (R/W)

Chapter 41

Motor Control PWM (MCPWM)

41.1 Overview

The **M**otor **C**ontrol **P**ulse **W**idth **M**odulator (MCPWM) peripheral is intended for motor and power control. It provides six PWM outputs that can be set up to operate in several topologies. One common topology uses a pair of PWM outputs driving an H-bridge to control motor rotation speed and rotation direction.

The MCPWM can be divided into five main modules: PWM timers, PWM operators, Capture module, Event Task Matrix (ETM) module, and Fault Detection module. Each PWM timer provides timing references that can either run freely or be synced to other timers or external sources. Each PWM operator has all the necessary control resources to generate waveform pairs for one PWM channel. The Capture module is used for systems that need to accurately time external events. The ETM module responds to tasks received by the MCPWM, generating corresponding events depending on the state of motion. The Fault Detection module is used to capture external faults, allowing the system to respond by choice.

ESP32-P4 has two MCPWM peripherals, which are MCPWM0 and MCPWM1.

41.2 Features

An MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, a Capture module, an ETM module, and a Fault Detection module. MCPWM's core clock can be selected from three clock sources: PLL_F160M_CLK, XTAL_CLK, and RC_FAST_CLK (configured by the HP_SYS_CLKRST_MCPWM*n*_CLK_SRC_SEL field of the HP_SYS_CLKRST_PERI_CLK_CTRL20_REG register). Figure 41.2-1 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.

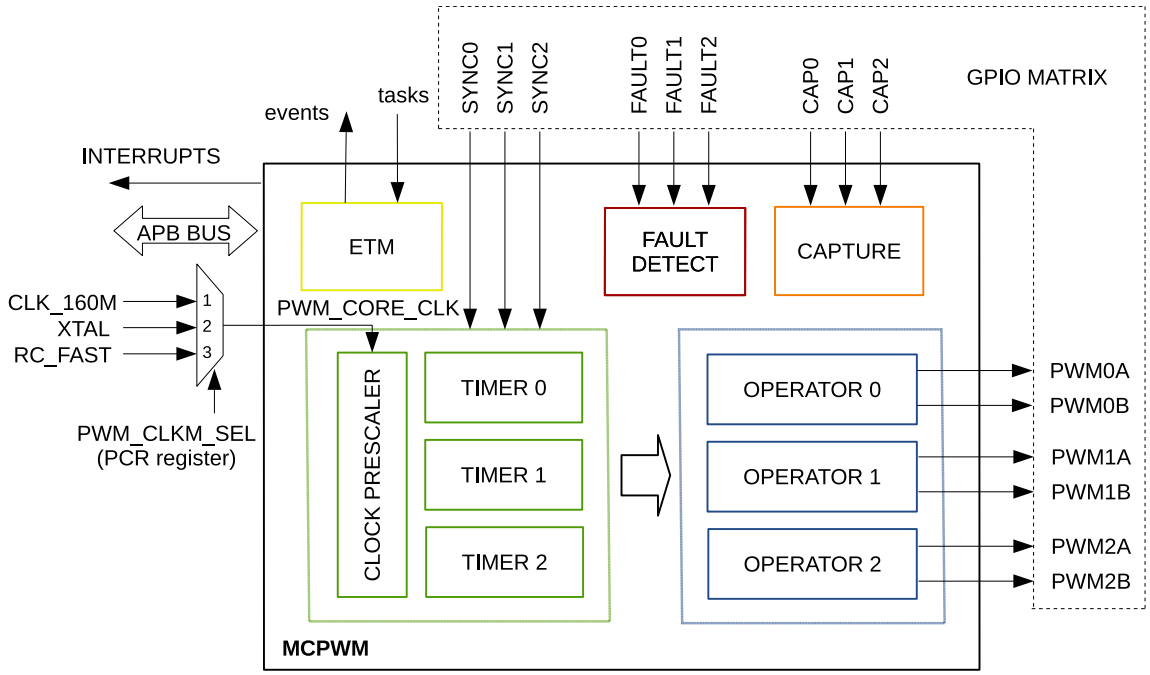


Figure 41.2-1. MCPWM Module Overview

Below is an overview of the submodules' functionality in Figure 41.2-1:

- PWM Timers 0, 1, and 2:
  - Every PWM timer has a dedicated 8-bit clock prescaler.
  - The 16-bit counter in the PWM timer can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode.
  - A hardware sync or software sync can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler's restart, so that the timer's clock can also be synced. The source of the hard sync can come from any GPIO or any other PWM timer's sync_out. The source of the soft sync comes from writing toggle value to the MCPWM_TIMER*n*_SYNC_SW bit.
- PWM Operators 0, 1, and 2:
  - Every PWM operator has two PWM outputs: PWM*x*A and PWM*x*B. They can work independently, in symmetric or asymmetric configurations.
  - The control of the PWM signal can be updated asynchronously.
  - Configurable dead time on rising and falling edges; each set up independently.
  - All events can trigger CPU interrupts.
  - Modulating of PWM output by high-frequency carrier signals, useful when gate drivers are insulated with a transformer.
  - Period, time stamps, and important control registers have shadow registers with flexible updating methods.
- Fault Detection Module:
  - Programmable fault handling in both cycle-by-cycle mode and one-shot mode.

  - A fault condition can force the PWM output to either high or low logic levels.

- Capture Module:
  - Clock of the capture module is the same as MCPWM's core clock.
  - Speed measurement of rotating machinery.
  - Measurement of elapsed time between position sensor pulses
  - Period and duty cycle measurement of pulse train signals
  - Decoding current or voltage amplitude derived from duty-cycle-encoded signals of current/voltage sensors
  - Three individual capture channels, each of which with a time-stamp register (32-bit)
  - Selection of edge polarity and prescaling of input capture signals
  - The capture timer can sync with a PWM timer or external signals.
  - Interrupt on each of the three capture channels
- ETM Module:
  - Generation of different events depending on the different running states of each timer and operator.
  - Each timer and operator responds to its corresponding task and automatically performs the corresponding operation.
  - Each event and task can be enabled independently. When an event is not enabled, the corresponding event will not be generated. When a task is not enabled, the corresponding task will not be responded to.

41.3 Modules

41.3.1 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.

41.3.1.1 Timer Module

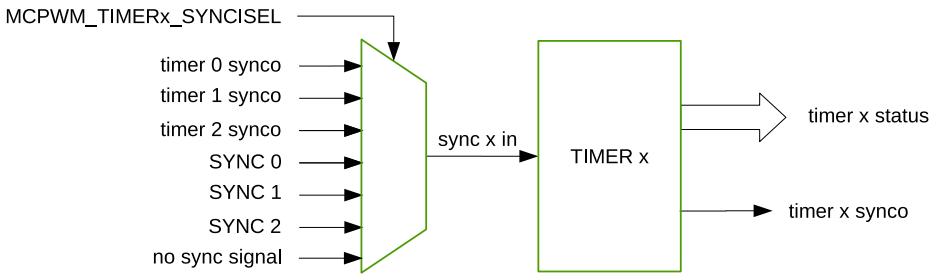


Figure 41.3-1. Timer Module

This module generates a timer, which can count at a specified period. It can work in Count-Up Mode, Count-Down Mode, or Count-Up-Down Mode. It supports different synchronization input sources (a total of seven optional input sources) for reloading the count value and direction, allowing synchronization among multiple timers. Furthermore, it provides synchronization output options (a total of four optional output sources) for use by other timers.

The timer*x* status in the figure represents the timer status output by the timer module, such as the counting mode, and count value equal to zero or period.

41.3.1.2 Operator Module

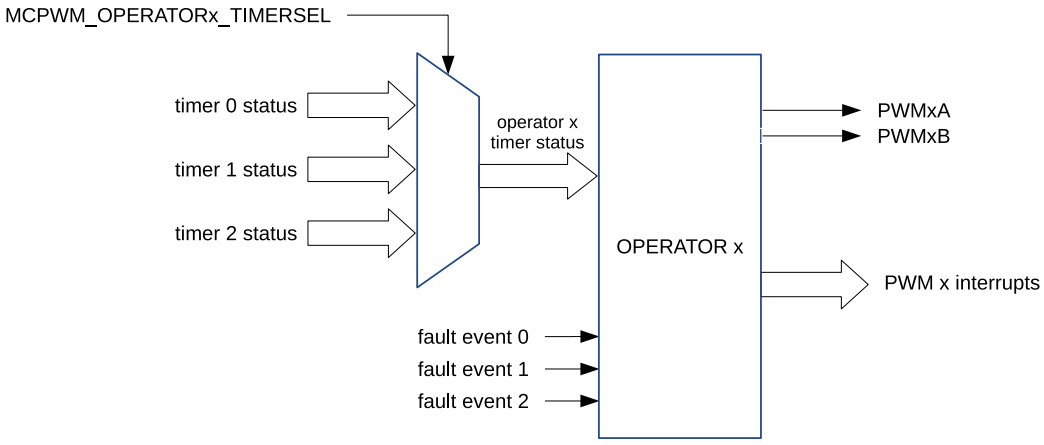


Figure 41.3-2. Operator Module

A PWM operator contains a PWM generator, a dead time generator, and a PWM carrier module. Their functions are shown in Table 41.3-1.

Table 41.3-1. 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.

41.3.1.3 Capture Module

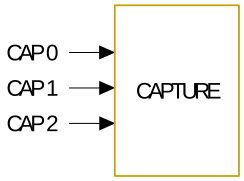


Figure 41.3-3. Capture Module

The module contains a timer that can be synchronized. When the input capture signal is valid, the count value of the timer is captured.

41.3.1.4 ETM Module

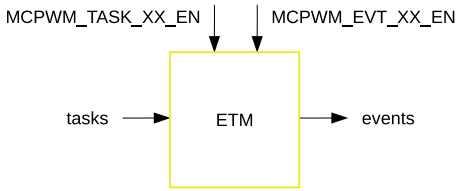


Figure 41.3-4. ETM Module

The module responds to received tasks, or generate and output events. Each event and task can be enabled independently.

41.3.2 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.

41.3.2.1 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.

41.3.2.2 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 41.3-5 to 41.3-8 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.

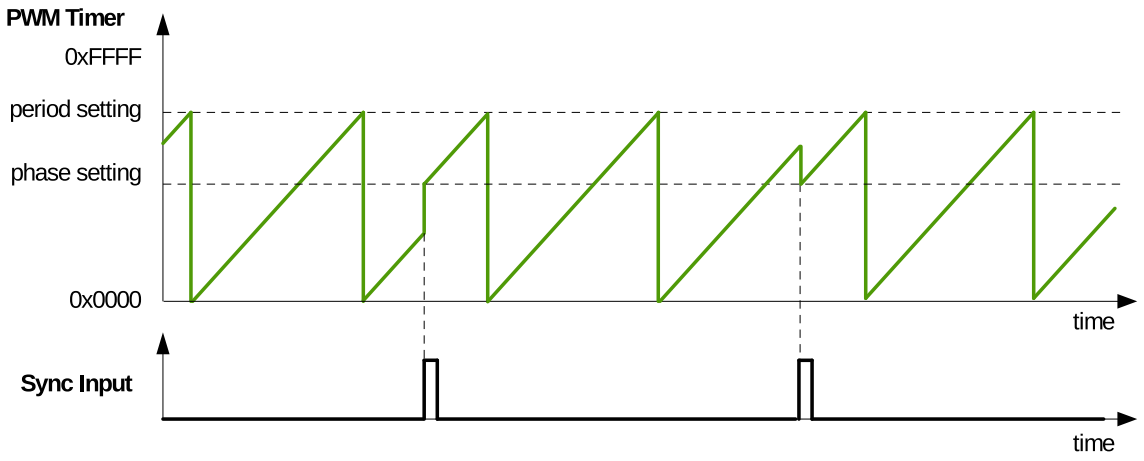


Figure 41.3-5. Count-Up Mode Waveform

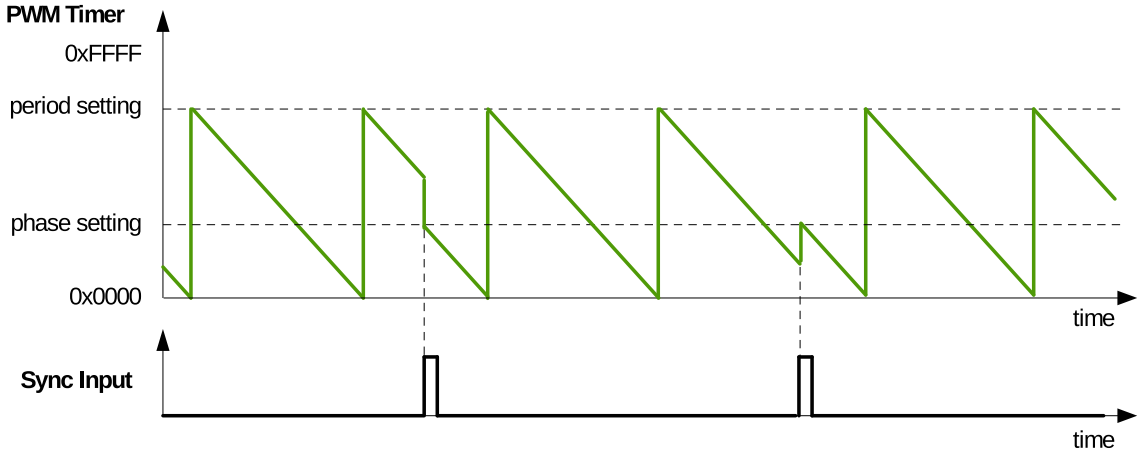


Figure 41.3-6. Count-Down Mode Waveforms

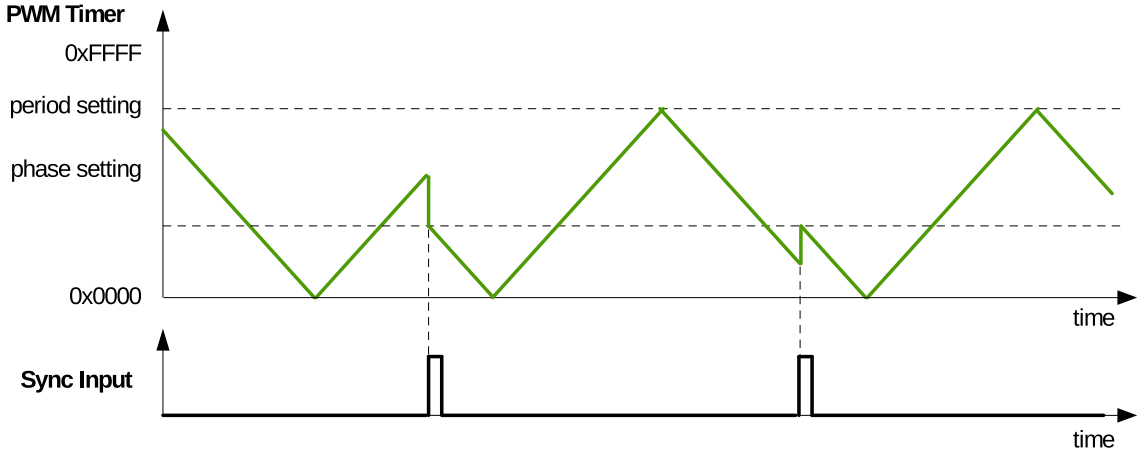


Figure 41.3-7. Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

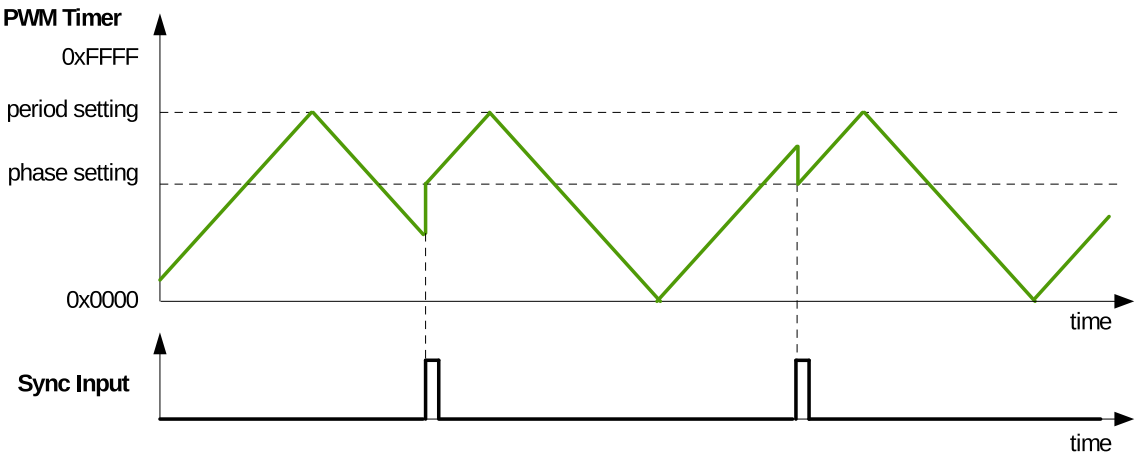


Figure 41.3-8. Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

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 41.3-9 to 41.3-11 show the timing waveforms of U/DTEP and U/DTEZ.

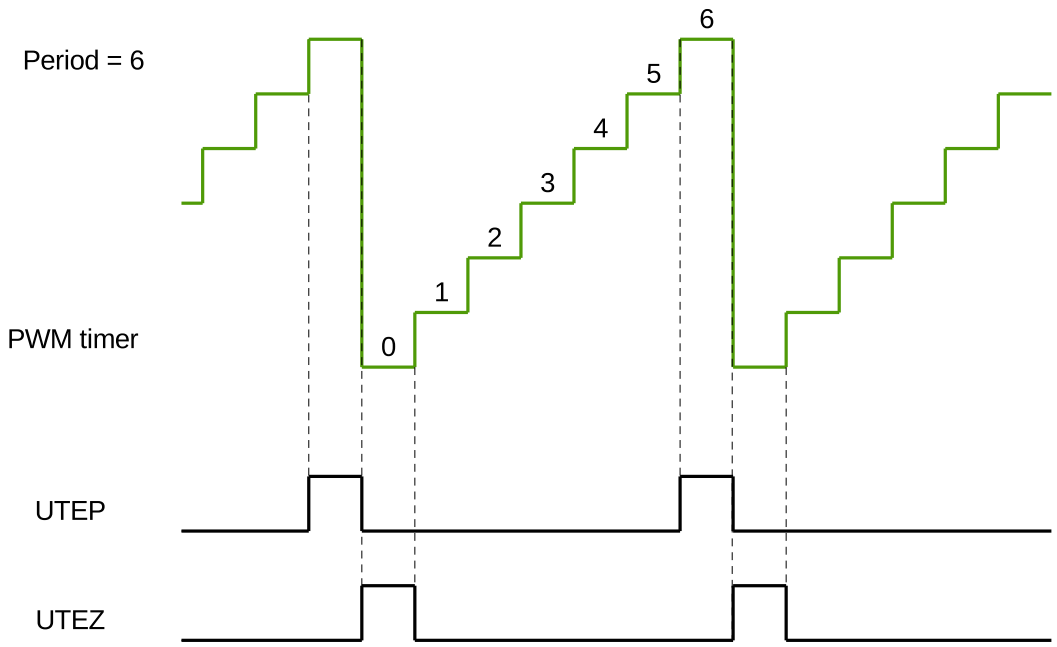


Figure 41.3-9. UTEP and UTEZ Generation in Count-Up Mode

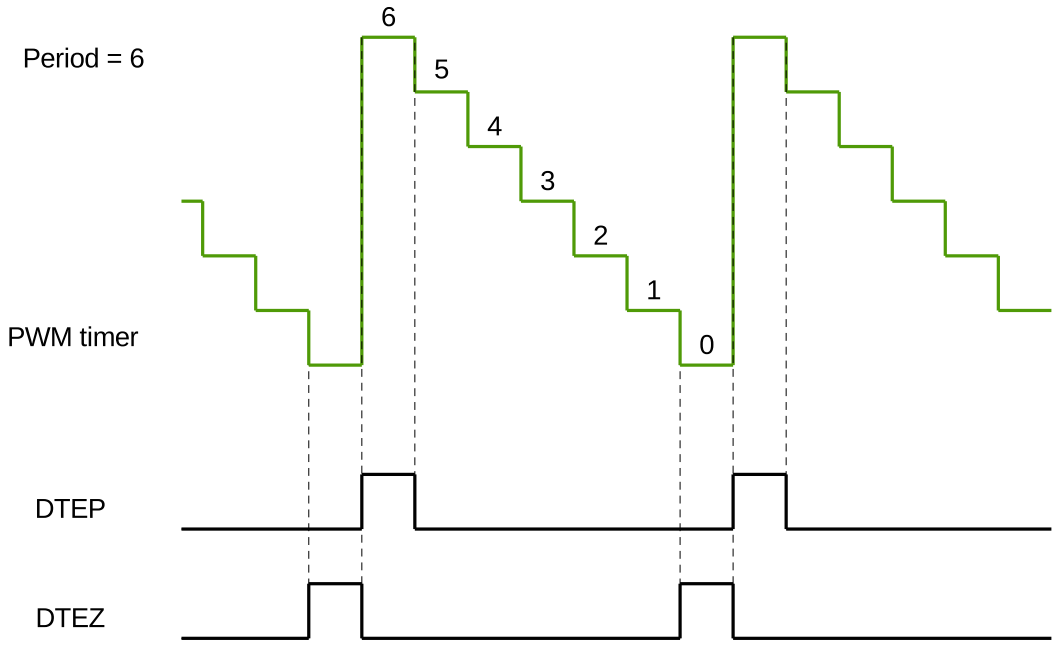


Figure 41.3-10. DTEP and DTEZ Generation in Count-Down Mode

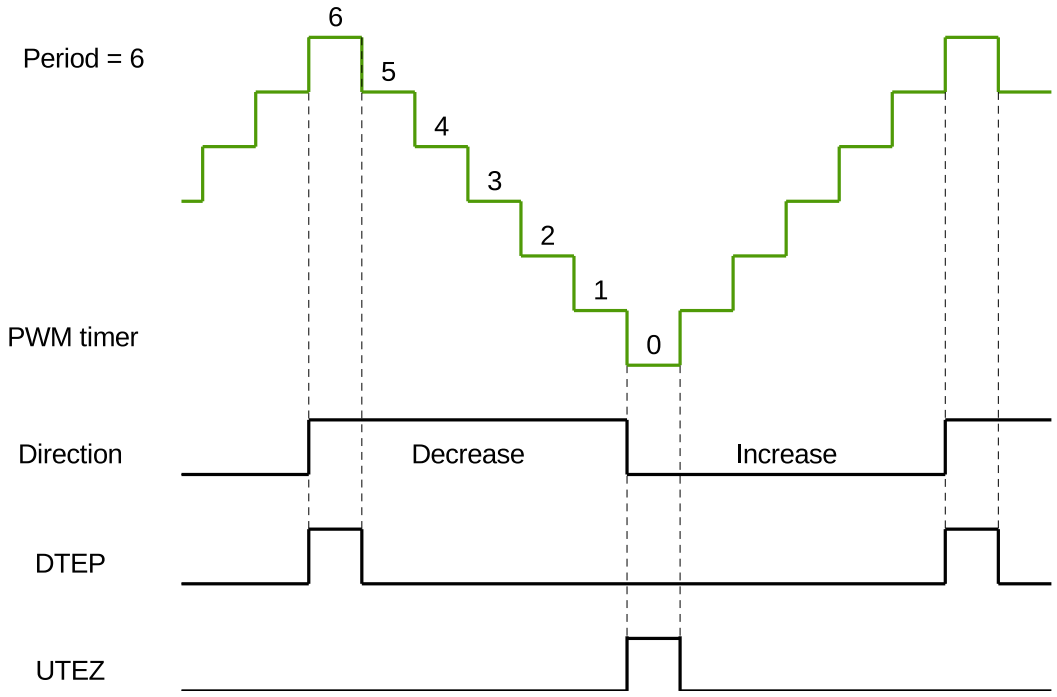


Figure 41.3-11. DTEP and UTEZ Generation in Count-Up-Down Mode

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.

41.3.2.3 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.

41.3.2.4 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.

41.3.3 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 41.3-12 shows the block diagram of a PWM operator.

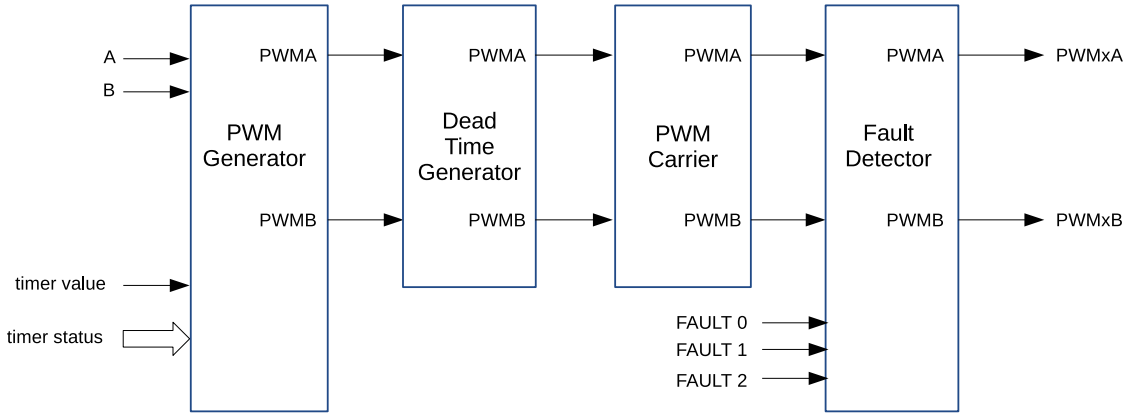


Figure 41.3-12. Block Diagram of A PWM Operator

41.3.3.1 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 41.3.3.1):
  - 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 41.3.2.3.

Timing Events

For convenience, all timing signals and events are summarized in Table 41.3-2.

Table 41.3-2. 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 41.3-3 and Table 41.3-4. 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 41.3-3. Timing Events Priority When PWM Timer Increments

Priority Level	Event
1 (highest)	Software-forced event
2	UTEP

Priority Level	Event
3	UT0
4	UT1
5	UTEB
6	UTEA
7 (lowest)	UTEZ

Table 41.3-4. 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 41.3-9, 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 41.3-10, 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 41.3-2 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 41.5.

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 41.3.2. A and B denote the time stamp registers A and B used by the hardware. For register configuration, please refer to the section 41.3.3.1. 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 41.3.3.1. 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 41.3.3.1 for configuration. PWM*x*A and PWM*x*B are the output signals of PWM generator*x*.

Figure 41.3-13 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.

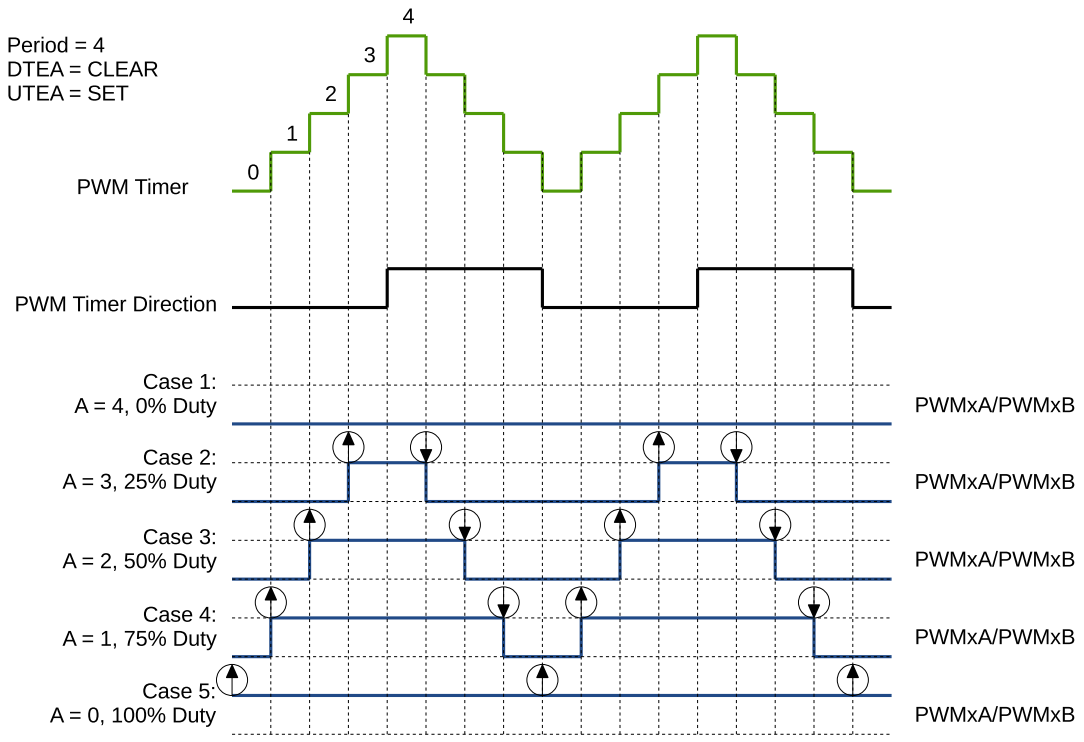


Figure 41.3-13. Symmetrical Waveform in Count-Up-Down Mode

The PWM waveforms in Figures 41.3-14 to 41.3-17 show some common PWM generator configurations.

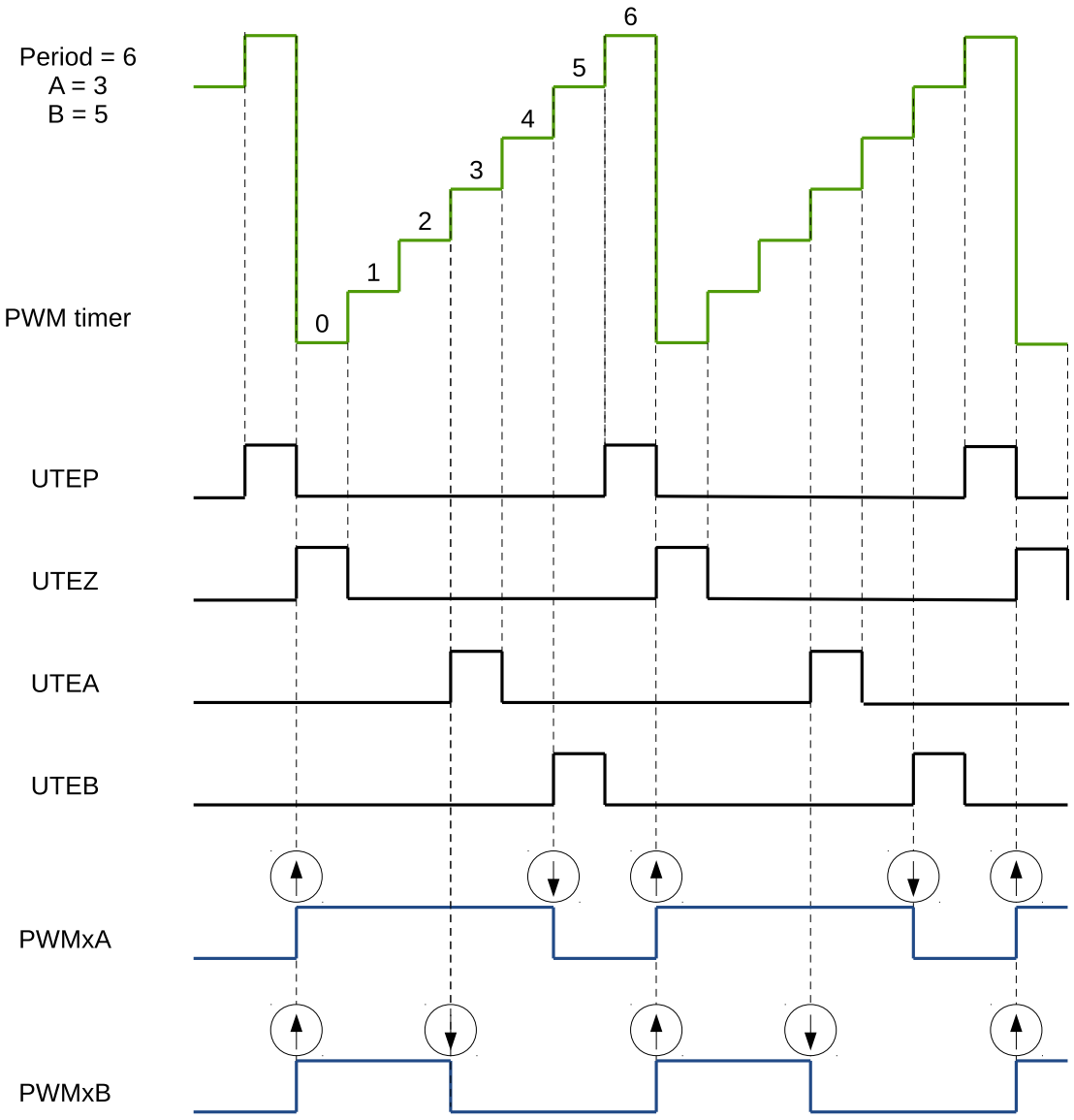


Figure 41.3-14. Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

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.

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

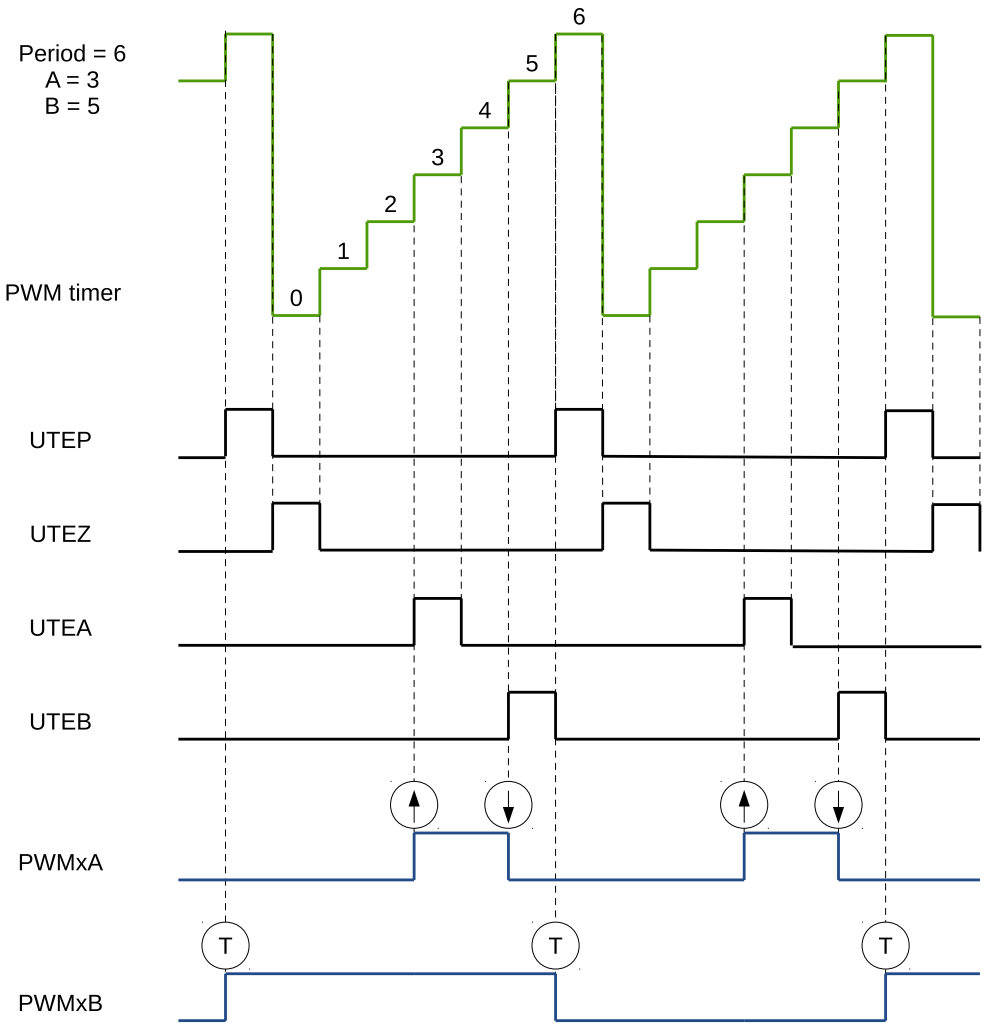


Figure 41.3-15. Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A

Pulses may be generated anywhere within the PWM cycle (zero to period).

PWM*x*A's high time duty is proportional to (B – A).

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

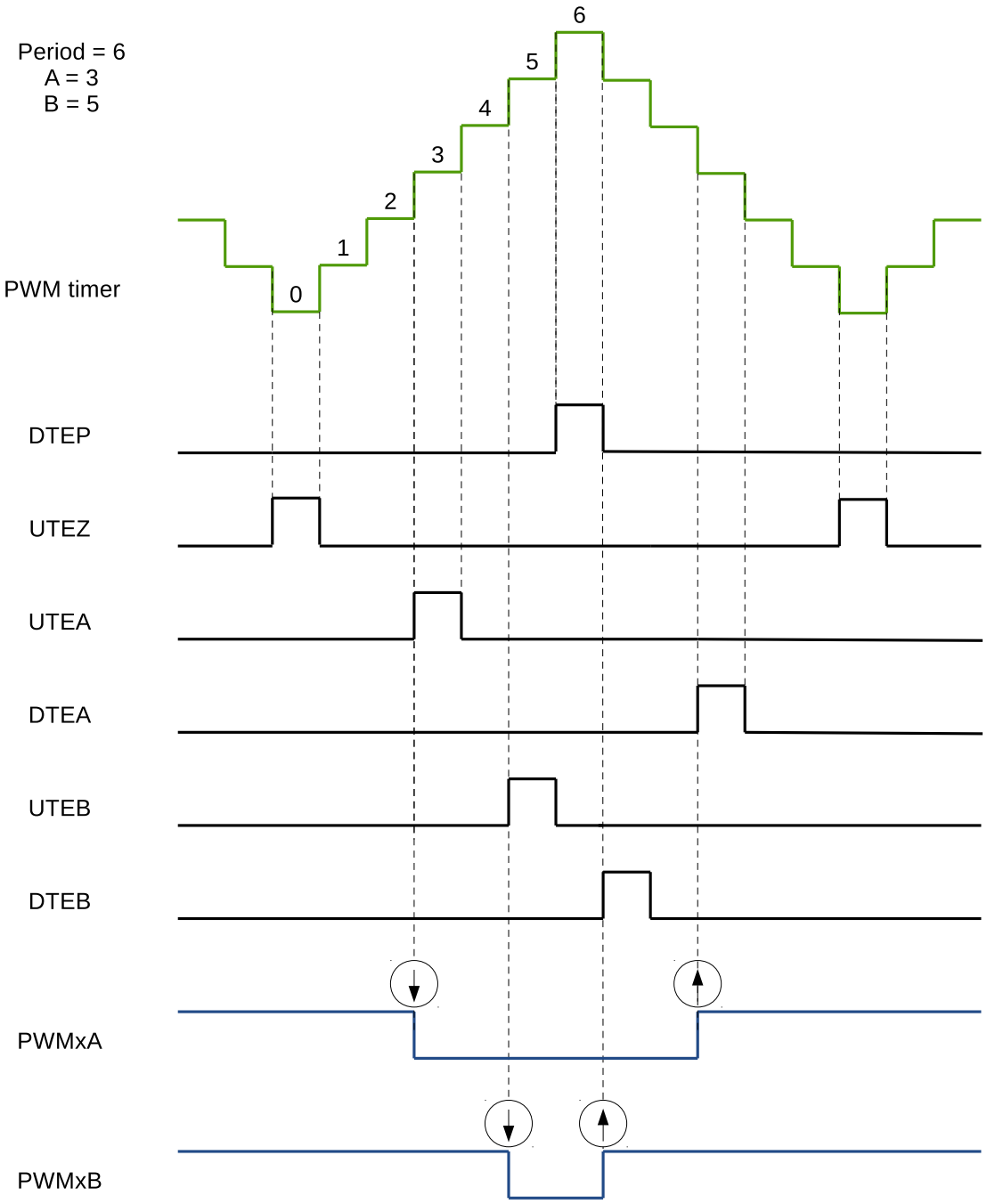


Figure 41.3-16. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

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.

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

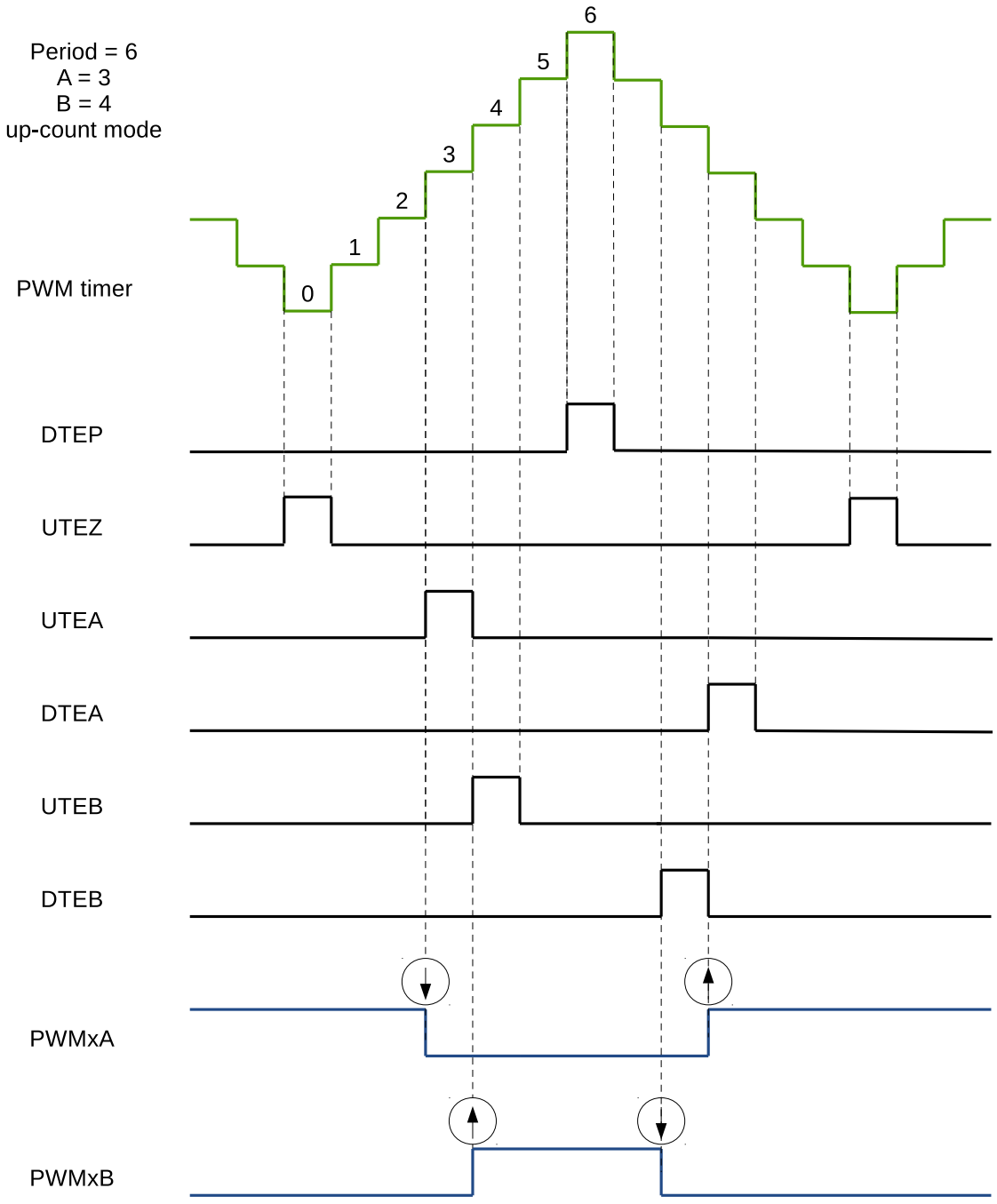


Figure 41.3-17. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary

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.

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

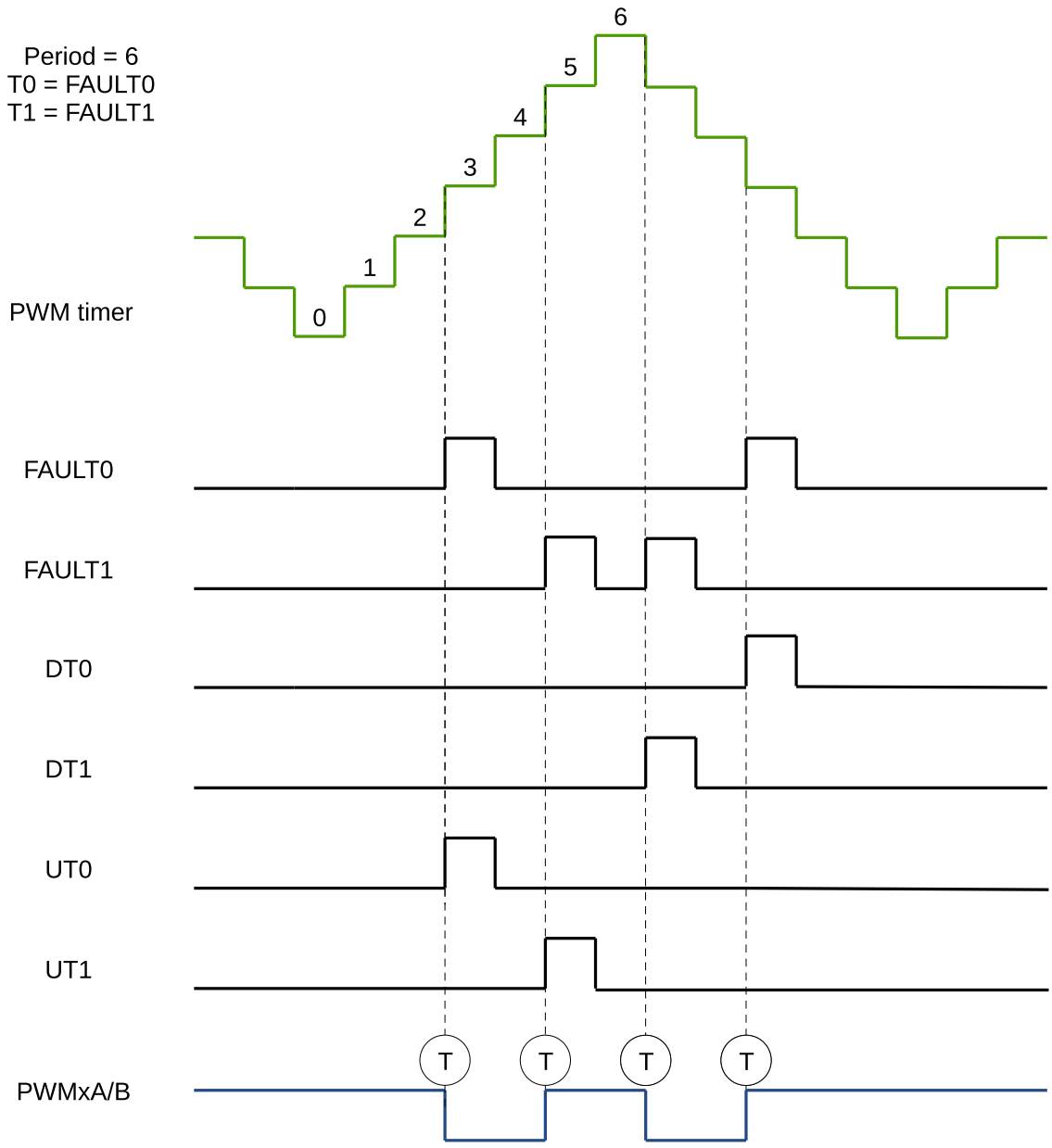


Figure 41.3-18. Count-Up-Down, Fault or Synchronization Events, with Same Modulation on PWM*x*A and PWM*x*B

Figure 41.3-18 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 41.3.3.1.

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 41.3-19 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.

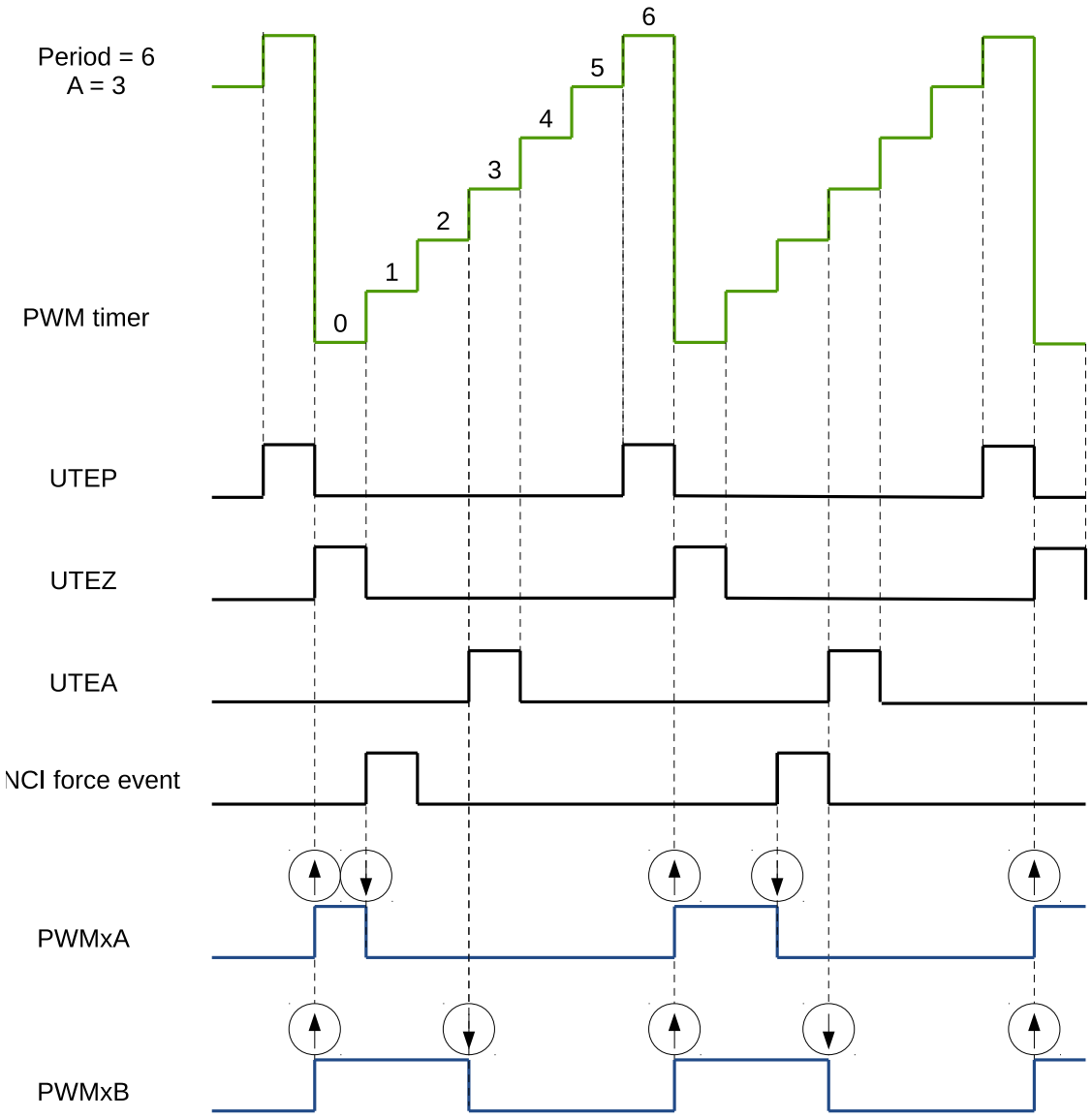


Figure 41.3-19. Example of an NCI Software-Force Event on PWM*x*A

Figure 41.3-20 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.

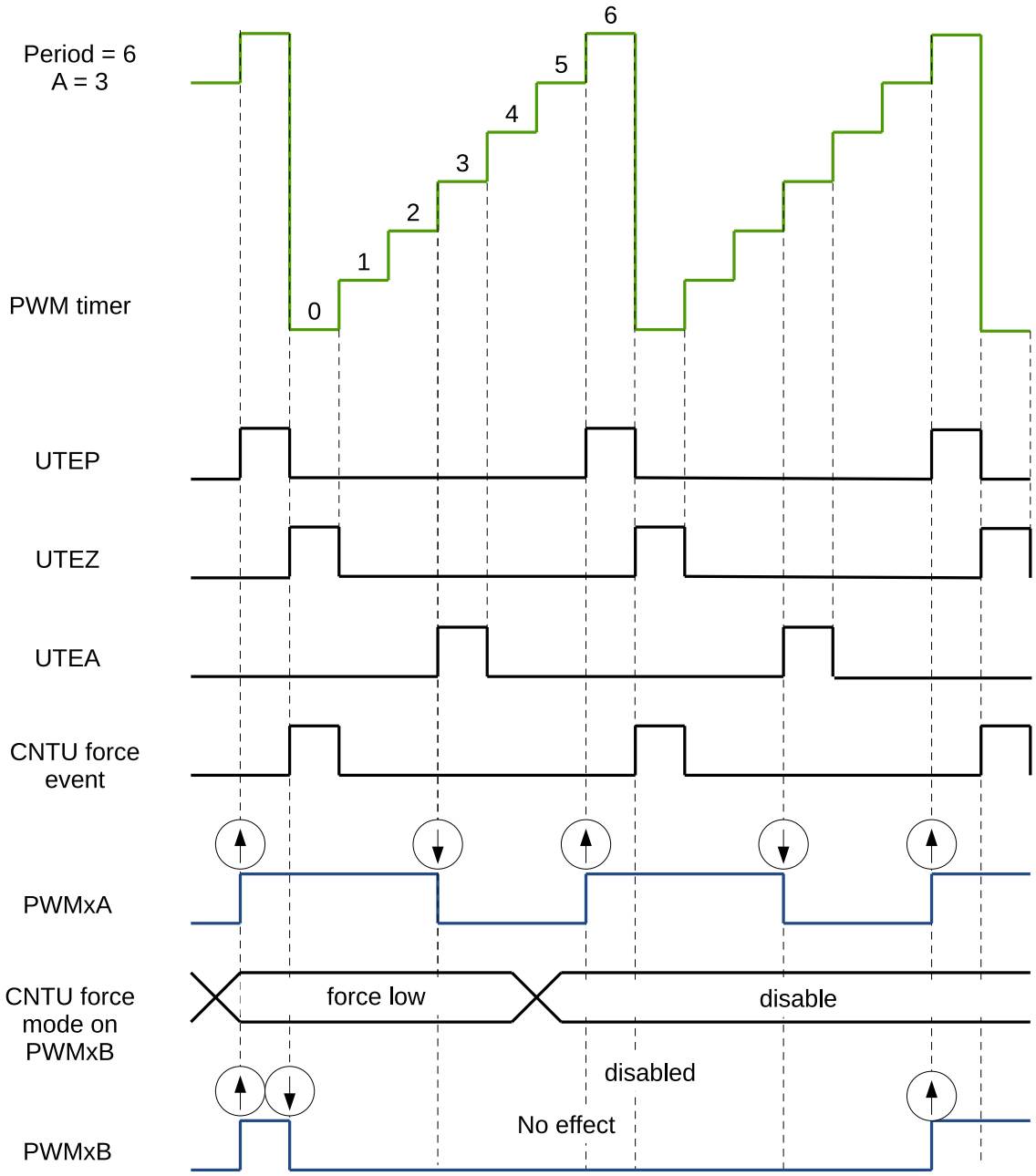


Figure 41.3-20. Example of a CNTU Software-Force Event on PWM*x*B

41.3.3.2 Dead Time Generator Module

Purpose of the Dead Time Generator Module

Section 41.3.3.1 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 41.3.2.3.

Highlights for Operation of the Dead Time Generator

Options for setting up the dead time module are shown in Figure 41.3-21.

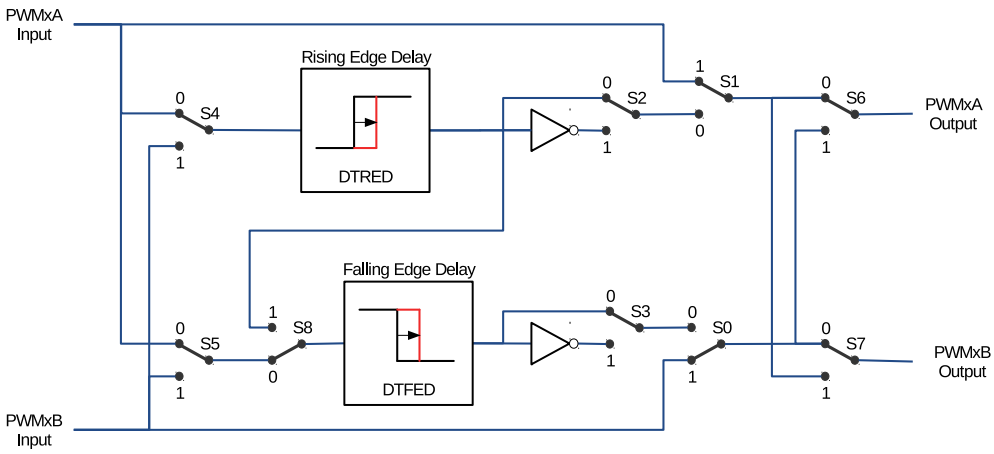


Figure 41.3-21. Options for Setting up the Dead Time Generator Module

S0-S8 in the figure above are switches controlled by fields in register MCPWM_DT*n*_CFG_REG shown in Table 41.3-5.

Table 41.3-5. 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 41.3-6 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 41.3-6 may be categorized as follows:

Table 41.3-6. 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 41.3-22	0	0	0	1
3	Active Low Complementary (ALC), see Figure 41.3-23	0	0	1	0
4	Active High (AH), see Figure 41.3-24	0	0	0	0

Mode	Mode Description	S0	S1	S2	S3
5	Active Low (AL), see Figure 41.3-25	0	0	1	1
6	PWM*x*A Output = PWM*x*A In (No Delay) PWM*x*B Output = PWM*x*A Input with Falling Edge Delay	0	1	0 or 1	0 or 1
7	PWM*x*A Output = PWM*x*A Input with Rising Edge Delay PWM*x*B Output = PWM*x*B Input with No Delay	1	0	0 or 1	0 or 1

Note:
For all the modes above, the position of the binary switches S4 to S8 is set to 0.

- **Mode 1: Bypass delays on both FED and RED**
  In this mode, the dead time module is disabled. Signals of PWM*x*A and PWM*x*B pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**
  These four modes represent typical configurations of polarity and should cover the active-high/low modes in available industry power switch gate drivers. The typical waveforms are shown in Figures 41.3-22 to 41.3-25.
- **Modes 6 and 7: Bypass delay on the falling edge (FED) or rising edge (RED)**
  In these two modes, either RED or FED is bypassed. As a result, the corresponding delay is not applied.

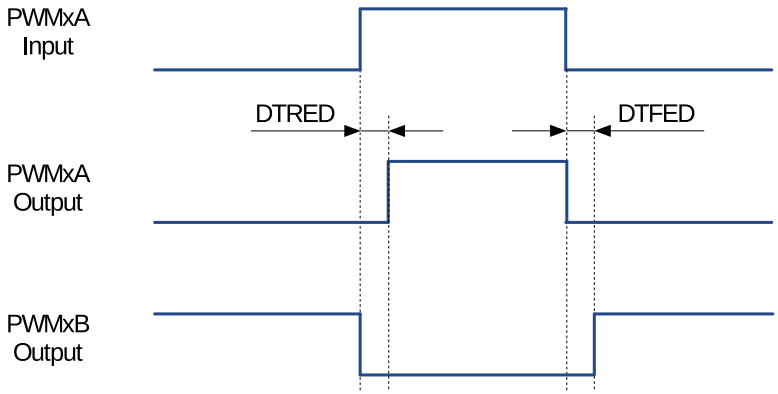


Figure 41.3-22. Active High Complementary (AHC) Dead Time Waveforms

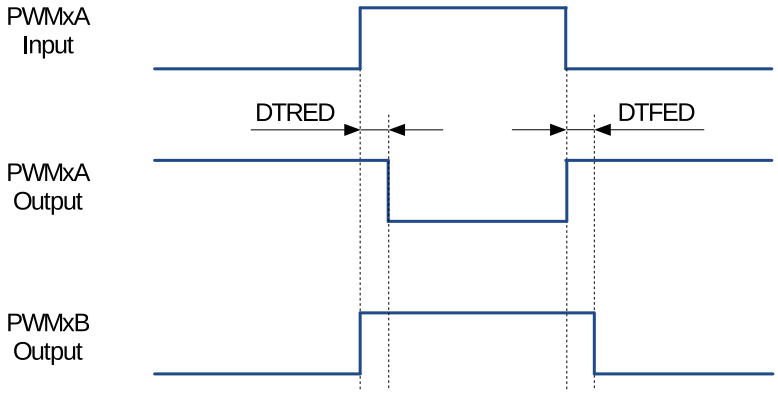


Figure 41.3-23. Active Low Complementary (ALC) Dead Time Waveforms

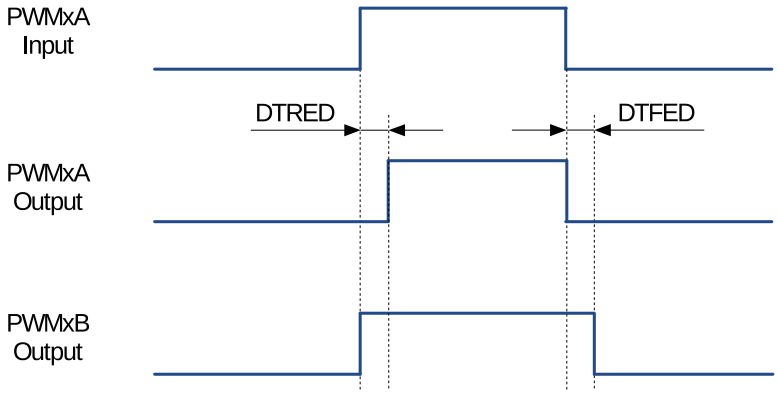


Figure 41.3-24. Active High (AH) Dead Time Waveforms

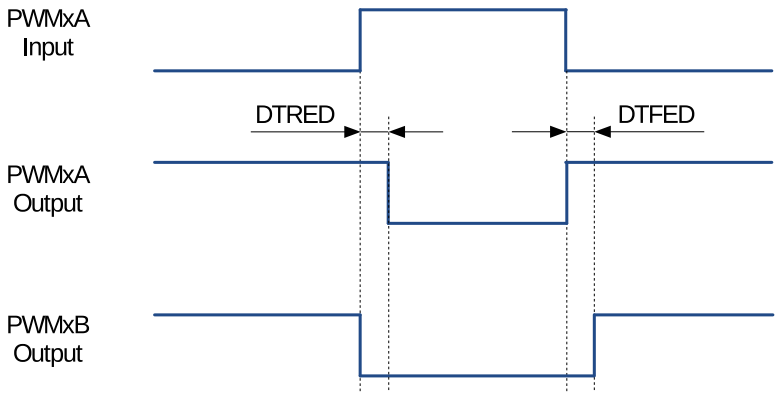


Figure 41.3-25. Active Low (AL) Dead Time Waveforms

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:

$$FED = MCPWM\_DTn\_FED \times T_{DT\_clk}$$

$$RED = MCPWM\_DTn\_RED \times T_{DT\_clk}$$

41.3.3.3 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 41.3-26 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.

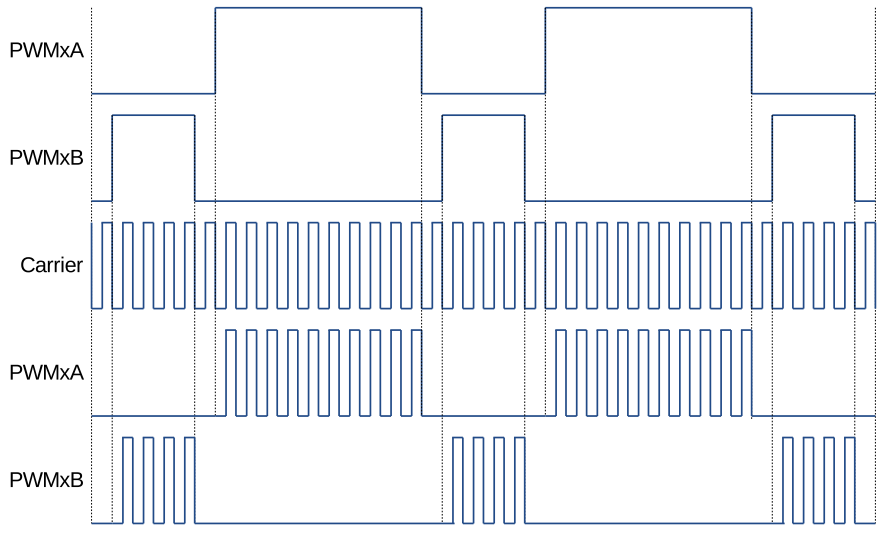


Figure 41.3-26. Example of Waveforms Showing PWM Carrier Action

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 41.3-27.

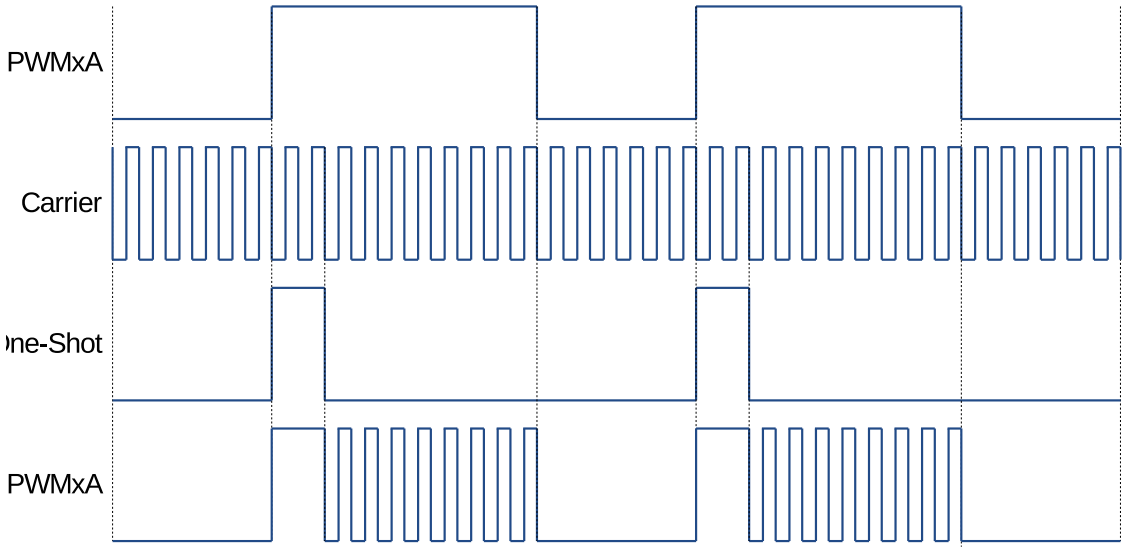


Figure 41.3-27. Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule

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 41.3-28.

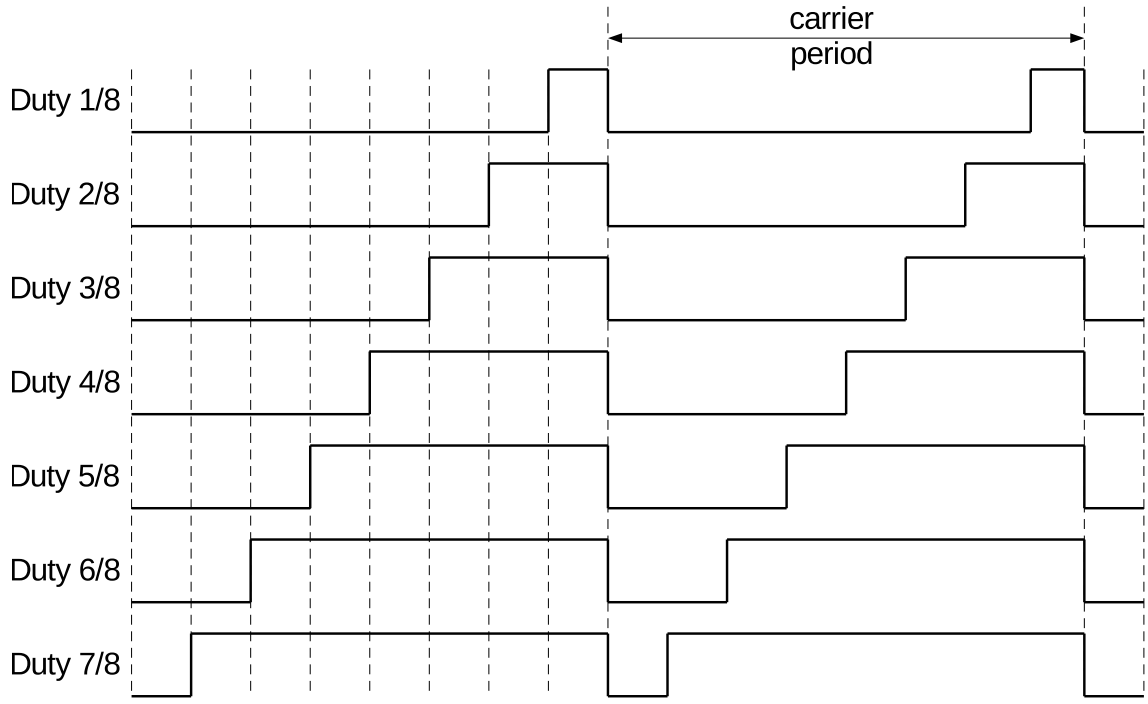


Figure 41.3-28. Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

41.3.3.4 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.

41.3.4 Capture Module

41.3.4.1 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.

41.3.4.2 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.

41.3.4.3 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.

41.3.5 ETM Module

41.3.5.1 Overview

The MCPWM peripheral on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows MCPWM's ETM tasks to be triggered by any peripherals' ETM events, or MCPWM's ETM events to trigger any peripherals' ETM tasks. The capture module, the fault detection module, three timers, and three operators can generate events and respond to tasks independently. This section introduces the ETM tasks and events related to MCPWM. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

41.3.5.2 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 41.3-7 below:

Table 41.3-7. MCPWM-Related ETM Events

Enable Field	Generation Condition	Event Generated
MCPWM_EVT_CAP*n*_EN	CAP*x* capture event occurs	MCPWM_EVT_CAP*x*
MCPWM_EVT_TZ*n*_OST_EN	PWM operator *x* performs a One-Shot trip (OST) operation	MCPWM_EVT_TZ*n*_OST
MCPWM_EVT_TZ*n*_CBC_EN	PWM operator *x* performs a cycle-by-cycle trip (CBC) operation	MCPWM_EVT_TZ*n*_CBC
MCPWM_EVT_F*n*_CLR_EN	Fault event fault_event*x* is cleared	MCPWM_EVT_F*n*_CLR

[1] See Section 41.3.3.1 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*x* is generated	MCPWM_EVT_F*x*
MCPWM_EVT_OP*n*_TEB_EN	The count value of the timer that PWM operator *x* selects is equal to the value of timer stamp B[1]	MCPWM_EVT_OP*n*_TEB
MCPWM_EVT_OP*n*_TEA_EN	The count value of the timer that PWM operator *x* selects is equal to the value of timer stamp A[1]	MCPWM_EVT_OP*n*_TEA
MCPWM_EVT_OP*n*_TEE1_EN	The count value of the timer that PWM operator *x* selects is equal to the value of register MCPWM_OP*x*_TSTMP_E1_REG	MCPWM_EVT_OP*n*_TEE1
MCPWM_EVT_OP*n*_TEE2_EN	The count value of the timer that PWM operator *x* selects is equal to the value of register MCPWM_OP*x*_TSTMP_E2_REG	MCPWM_EVT_OP*n*_TEE2
MCPWM_EVT_TIMER*n*_TEP_EN	The count value of timer *x* is equal to the period value MCPWM_TIMER*n*_PERIOD	MCPWM_EVT_TIMER*n*_TEP
MCPWM_EVT_TIMER*n*_TEZ_EN	The count value of timer *x* is equal to 0	MCPWM_EVT_TIMER*n*_TEZ
MCPWM_EVT_TIMER*n*_STOP_EN	Timer *x*'s count stops	MCPWM_EVT_TIMER*n*_STOP

[1] See Section 41.3.3.1 for a detailed description of timer stamp A and B.

41.3.5.3 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 41.3-8 below:

Table 41.3-8. ETM Related Tasks

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CAP*n*_EN	MCPWM_TASK_CAP*x*	CAP*x* channel performs a capture operation
MCPWM_TASK_CLR*n*_OST_EN	MCPWM_TASK_CLR*n*_OST	PWM operator *x* clears the One-Shot Trip operation
MCPWM_TASK_TZ*n*_OST_EN	MCPWM_TASK_TZ*n*_OST	PWM operator *x* performs a One-Shot Trip (OST) operation
MCPWM_TASK_TIMER*n*_PERIOD_UP_EN	MCPWM_TASK_TIMER*n*_PERIOD_UP	The period of timer *x* is updated to the value configured in the period register MCPWM_TIMER*n*_PERIOD
MCPWM_TASK_TIMER*n*_SYNC_EN	MCPWM_TASK_TIMER*n*_SYN	Timer *x* performs a sync operation
MCPWM_TASK_GEN_STOP_EN	MCPWM_TASK_GEN_STOP	All the timers stop counting and the PWM signals output by all PWM operators remain unchanged

Enable Field	Valid Task Input	Response Operation
MCPWM_TASK_CMPR*n*_B_UP_EN	MCPWM_TASK_CMPR*n*_B_UP	Timer stamp B of the PWM operator *x* is updated to the value of the shadow register MCPWM_GEN*n*_B
MCPWM_TASK_CMPR*n*_A_UP_EN	MCPWM_TASK_CMPR*n*_A_UP	Timer stamp A of the PWM operator *x* is updated to the value of the shadow register MCPWM_GEN*n*_A

41.4 Interrupts

ESP32-P4's MCPWM0 can generate the PWM0_INTR interrupt signal and MCPWM1 can generate the PWM1_INTR interrupt signal. The PWM*n*_INTR will be sent to the Interrupt Matrix.

Interrupt signal PWM*n*_INTR can be generated by the following internal interrupt sources:

- MCPWM_TIMER*x*_STOP_INT: Triggered when timer*x* stops.
- MCPWM_TIMER*x*_TEZ_INT: Triggered by the TEZ event of PWM timer*x*.
- MCPWM_TIMER*x*_TEP_INT: Triggered by the TEP event of PWM timer*x*.
- MCPWM_FAULT*x*_INT: Triggered when fault_event*x* starts.
- MCPWM_FAULT*x*_CLR_INT: Triggered after fault_event*x* ends.
- MCPWM_CMPR*x*_TEA_INT: Triggered by the TEA event of PWM operator*x*.
- MCPWM_CMPR*x*_TEB_INT: Triggered by the TEB event of PWM operator*x*.
- MCPWM_TZ*x*_CBC_INT: Triggered by the CBC action of PWM*x*.
- MCPWM_TZ*x*_OST_INT: Triggered by the OST action of PWM*x*.
- MCPWM_CAP*x*_INT: Triggered by the capture event on channel*x*.

The above interrupt sources can only be triggered when the interrupt enable register *interrupt_source_name*_ENA is set to 1. Write 1 to *interrupt_source_name*_CLR will clear the interrupt.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 41.5 *Register Summary*.

41.5 Register Summary

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

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

Name	Description	Address	Access
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 configuratoin	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 configuratoin	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 configuratoin	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

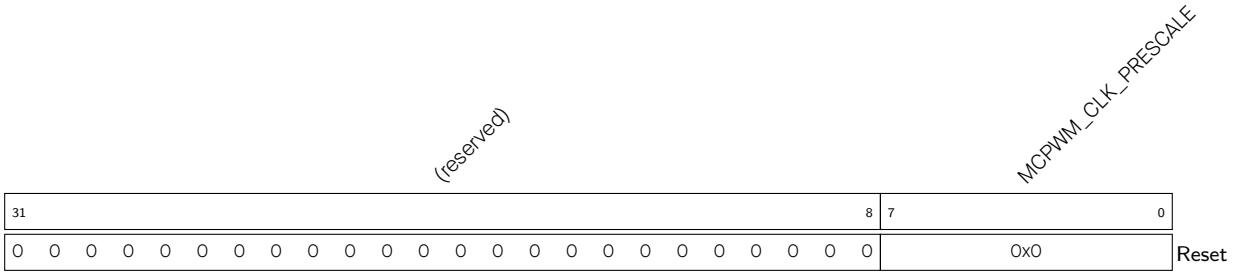
Name	Description	Address	Access
Fault Detection Configuration and Status			
MCPWM_FAULT_DETECT_REG	Fault detection configuration and status	0x00E4	varies
Capture Configuration and Status			
MCPWM_CAP_TIMER_CFG_REG	Configure capture timer	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	Phase for capture timer 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

41.6 Registers

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

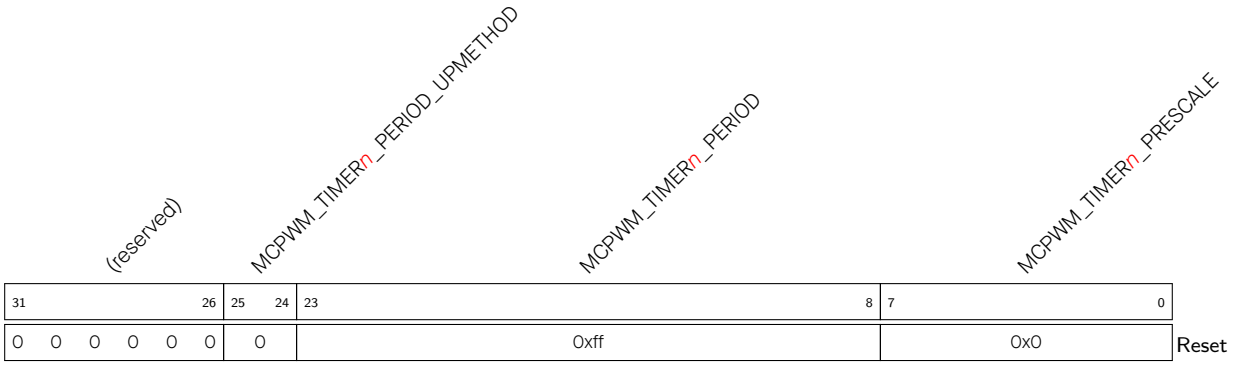
For how to program reserved fields, please refer to Section Programming Reserved Register Field.

Register 41.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE Configures the prescaler value of the clock, so that the period of PWM_clk = 6.25ns * (PWM_CLK_PRESCALE + 1). (R/W)

Register 41.2. MCPWM_TIMER*n*_CFG0_REG (*n*: 0-2) (0x0004+0x10n*)**

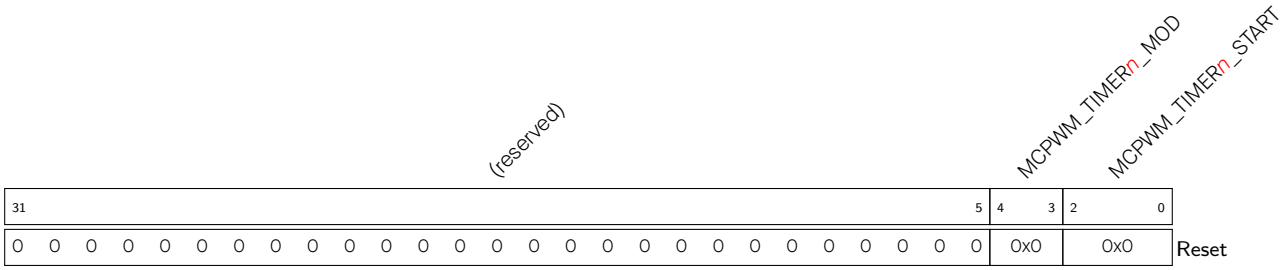


MCPWM_TIMER*n*_PRESCALE Configures the prescaler value of timer*n*, so that the period of PTO_clk = Period of PWM_clk * (PWM_TIMER*n*_PRESCALE + 1). (R/W)

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 41.3. 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

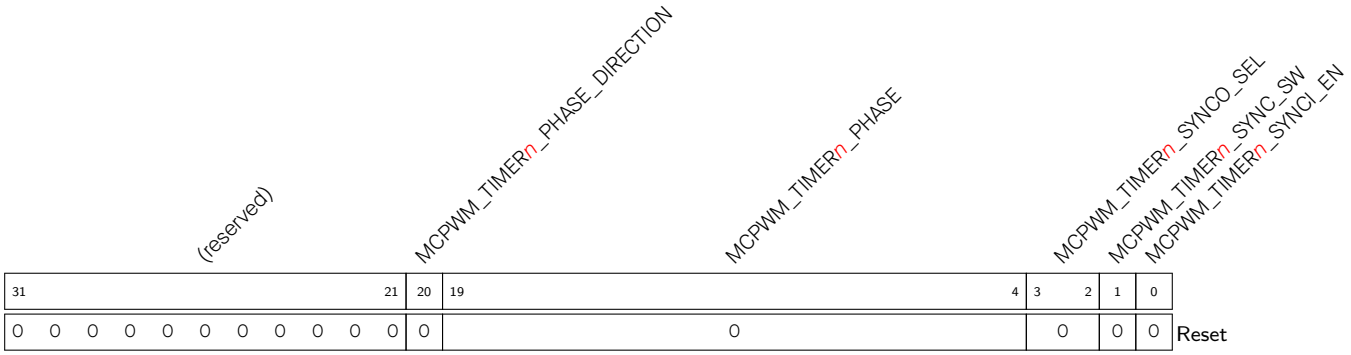
1: Increase mode

2: Decrease mode

3: Up-down mode

(R/W)

Register 41.4. MCPWM_TIMER*n*_SYNC_REG (*n*: 0-2) (0x000C+0x10**n*)



MCPWM_TIMER*n*_SYNCI_EN Configures whether to enable timer*n* reloading with phase on sync input event.

0: Disable

1: Enable

(R/W)

MCPWM_TIMER*n*_SYNC_SW Configures whether to trigger a software sync.

0: No effect

1: Trigger a software sync

(R/W)

MCPWM_TIMER*n*_SYNCO_SEL Configures PWM timer*n* sync out selection.

0: sync_in. The sync out will always generate when toggling the MCPWM_TIMER0_SYNC_SW bit.

1: TEZ

2: TEP

3: No effect

(R/W)

MCPWM_TIMER*n*_PHASE Configures the phase for timer*n* reload on sync event.

(R/W)

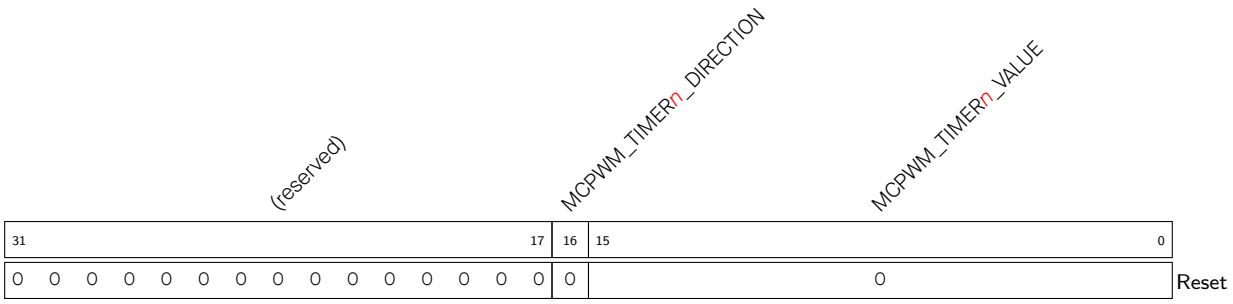
MCPWM_TIMER*n*_PHASE_DIRECTION Configures the PWM timer*n*'s direction when timer*n* mode is up-down mode.

0: Increase

1: Decrease

(R/W)

Register 41.5. MCPWM_TIMER*n*_STATUS_REG(*n*: 0-2) (0x0010+0x10**n*)



MCPWM_TIMER*n*_VALUE Represents current PWM timer*n* counter value. (RO)

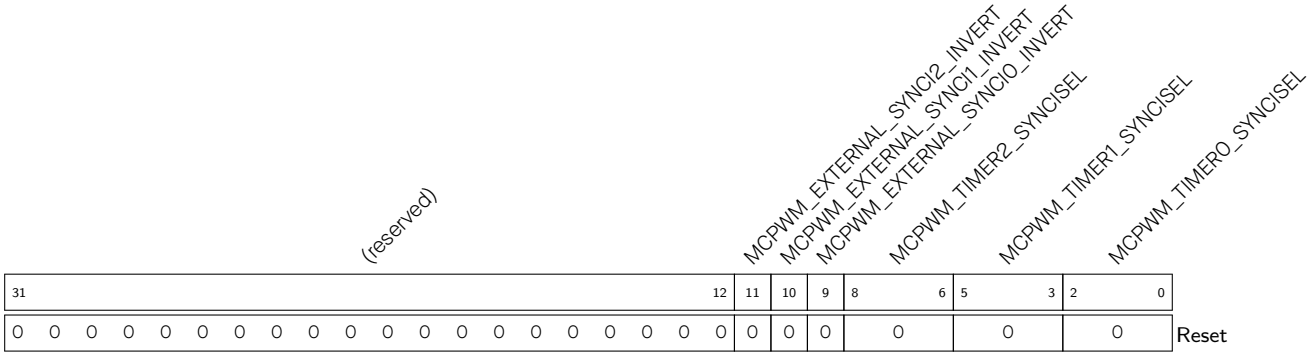
MCPWM_TIMER*n*_DIRECTION Represents current PWM timer*n* counter direction.

0: Increment

1: Decrement

(RO)

Register 41.6. 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)

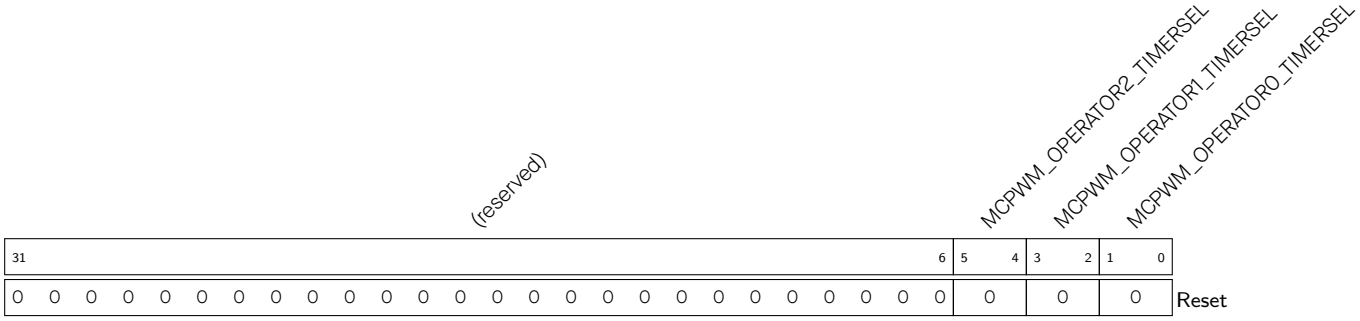
MCPWM_EXTERNAL_SYNCI*n*_INVERT Configures whether to invert SYNC*n* from GPIO matrix.

0: Not invert

1: Invert

(R/W)

Register 41.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL Configures which PWM timer will be the timing reference for PWM operator*n*.

0: timer0

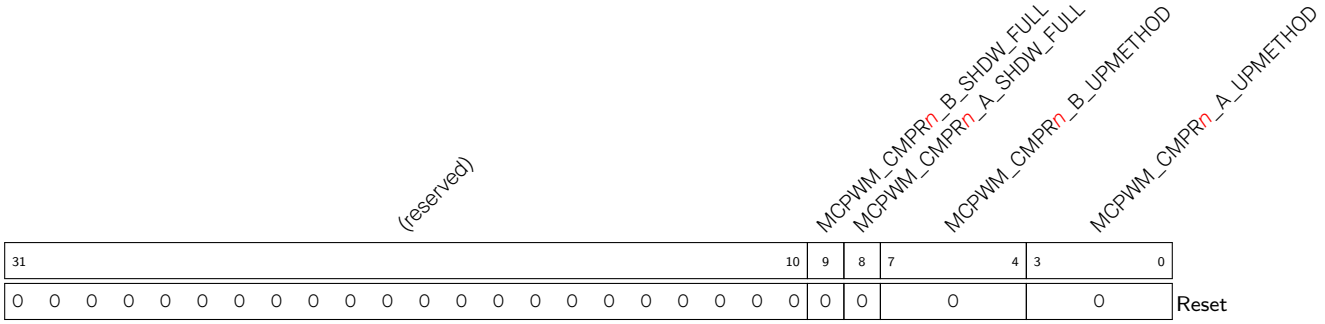
1: timer1

2: timer2

3: Invalid, will select timer2

(R/W)

Register 41.8. 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)

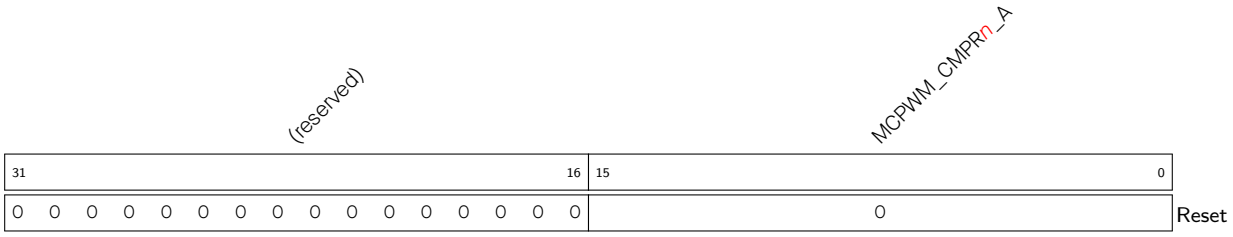
MCPWM_CMPR*n*_B_SHDW_FULL Configures whether the value in the shadow register is written to the corresponding active register at a specific time. This field is set and reset by hardware.

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

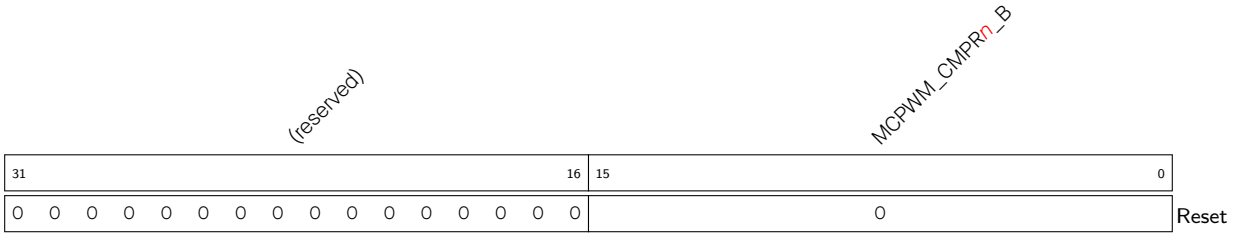
(R/SC/WTC)

Register 41.9. MCPWM_GEN*n*_TSTMP_A_REG(*n*: 0-2) (0x0040+0x38**n*)



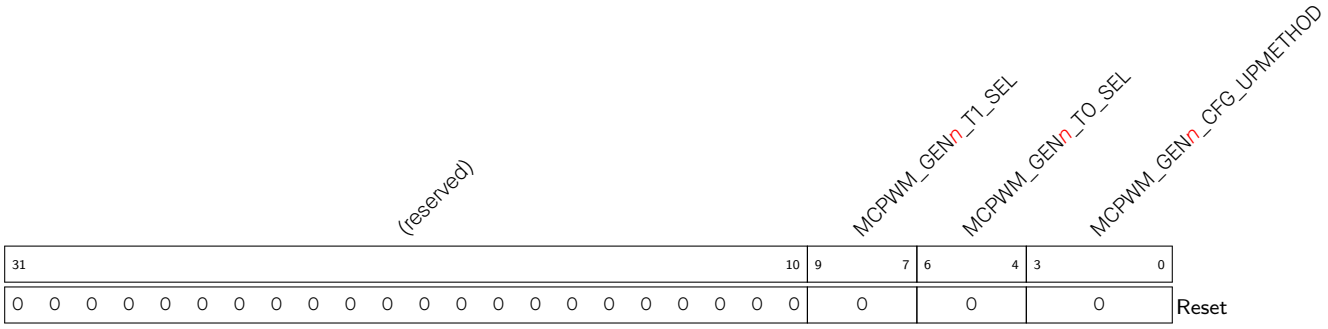
MCPWM_CMPR*n*_A Configures the value of PWM generator *n* time stamp A's shadow register. (R/W)

Register 41.10. MCPWM_GEN*n*_TSTMP_B_REG(*n*: 0-2) (0x0044+0x38**n*)



MCPWM_CMPR*n*_B Configures the value of PWM generator *n* time stamp B's shadow register. (R/W)

Register 41.11. MCPWM_GEN*n*_CFG0_REG(*n*: 0-2) (0x0048+0x38n*)**



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

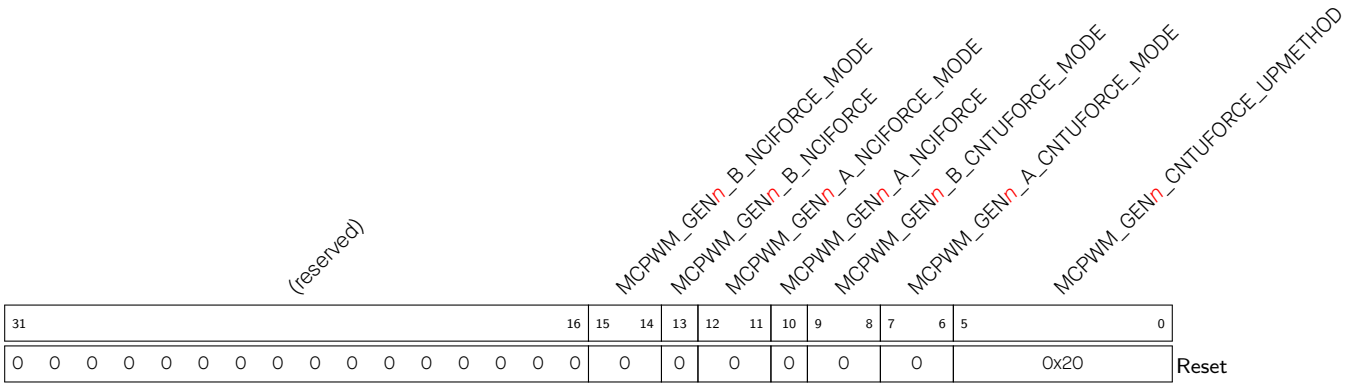
2: fault_event2

3: sync_taken

4: Invalid, selects nothing

(R/W)

Register 41.12. 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 41.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)

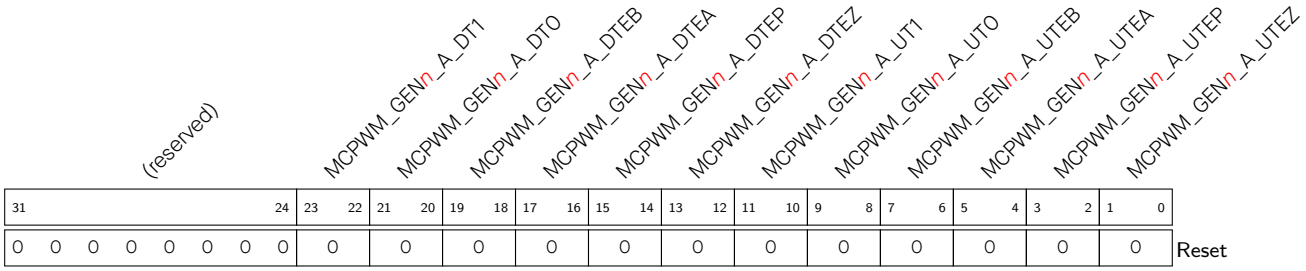
Continued from the previous page...

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 41.13. 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 41.13. 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 41.13. 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

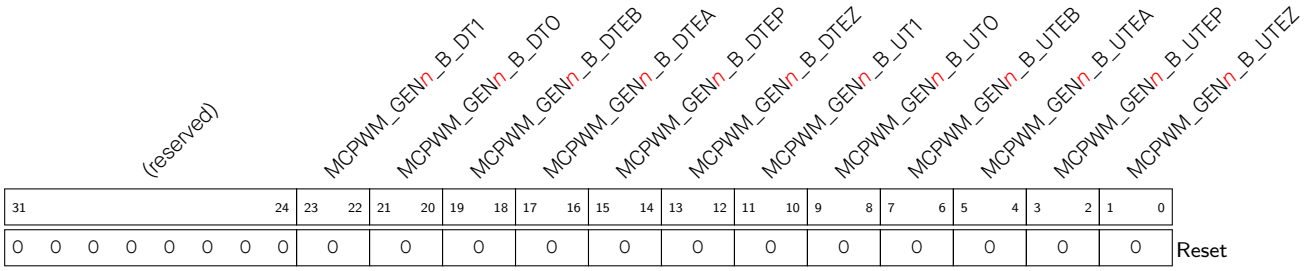
1: Low

2: High

3: Toggle

(R/W)

Register 41.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38n*)**



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 41.14. 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 41.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38n*)**

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

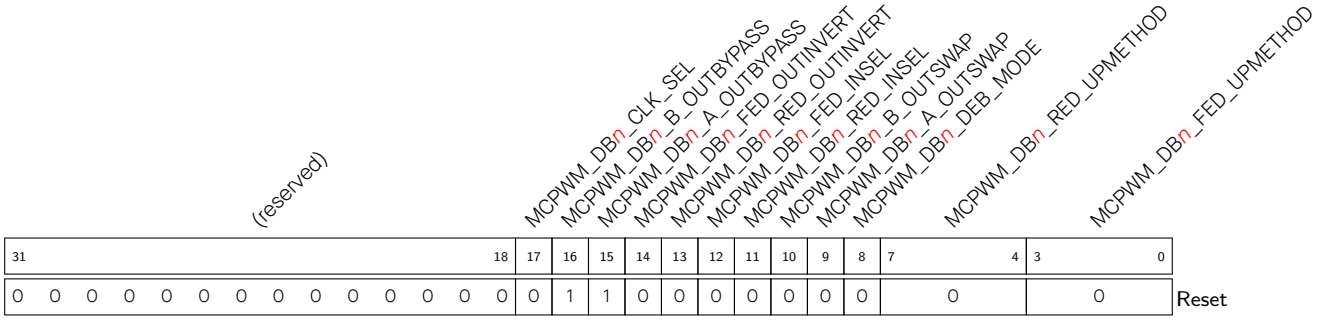
1: Low

2: High

3: Toggle

(R/W)

Register 41.15. 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 41.3-5.

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 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_B_OUTSWAP Configures the S7 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_RED_INSEL Configures the S4 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_FED_INSEL Configures the S5 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_RED_OUTINVERT Configures the S2 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

Continued on the next page...

Register 41.15. 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 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_A_OUTBYPASS Configures the S1 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

MCPWM_DB*n*_B_OUTBYPASS Configures the S0 switch in Table 41.3-5. For typical configurations, please refer to Table 41.3-6. (R/W)

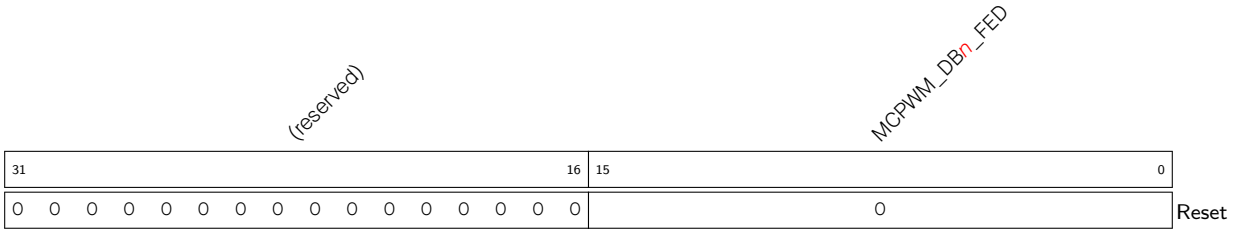
MCPWM_DB*n*_CLK_SEL Configures dead time generator *n* clock selection.

0: PWM_CLK

1: PT_CLK

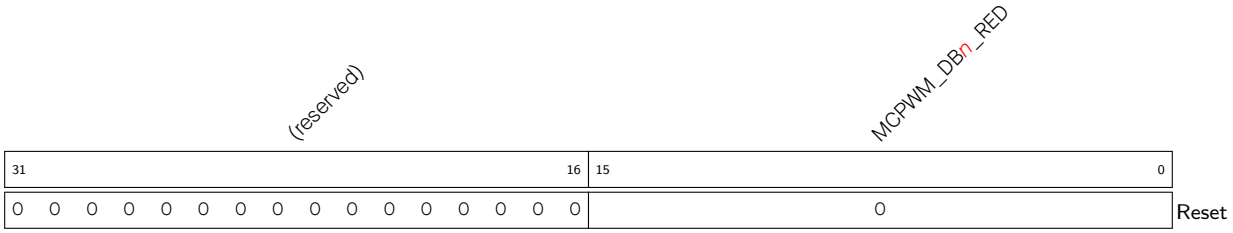
(R/W)

Register 41.16. MCPWM_DT*n*_FED_CFG_REG(*n*: 0-2) (0x005C+0x38**n*)



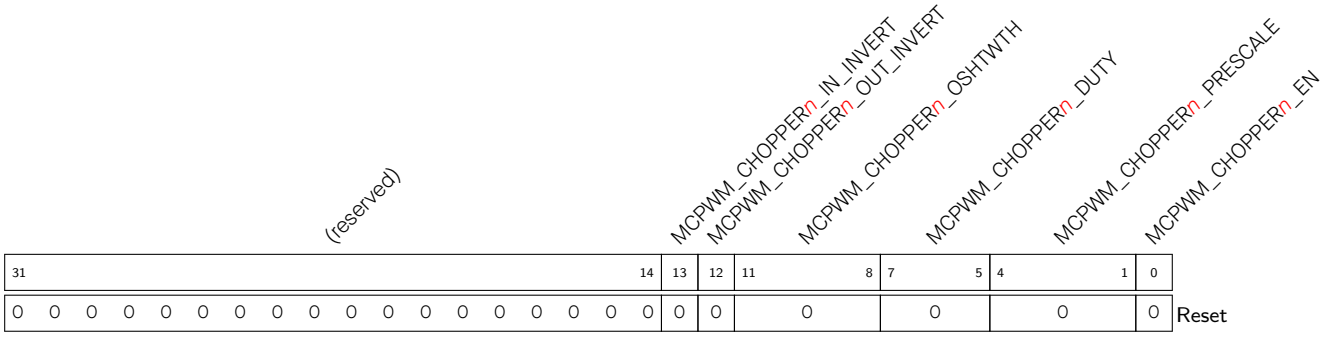
MCPWM_DB*n*_FED Configures the shadow register for FED. (R/W)

Register 41.17. 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 41.18. 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)

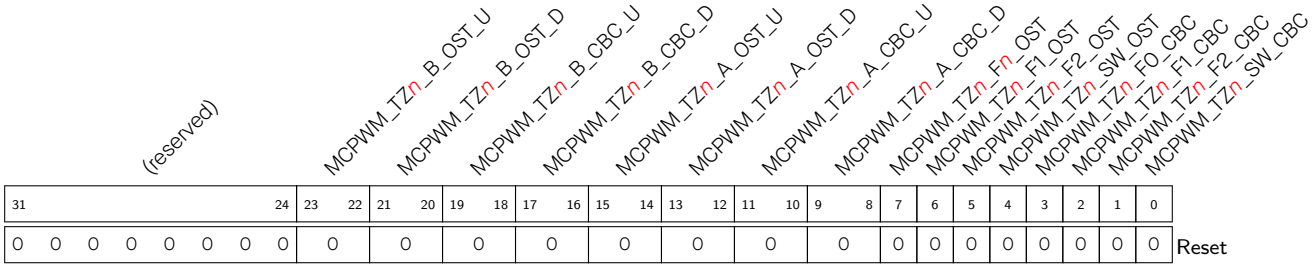
MCPWM_CHOPPER*n*_IN_INVERT Configures whether to invert the input of PWM*n* A and PWM*n* B for this submodule.

0: Normal

1: Invert

(R/W)

Register 41.19. 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 41.19. 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 41.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

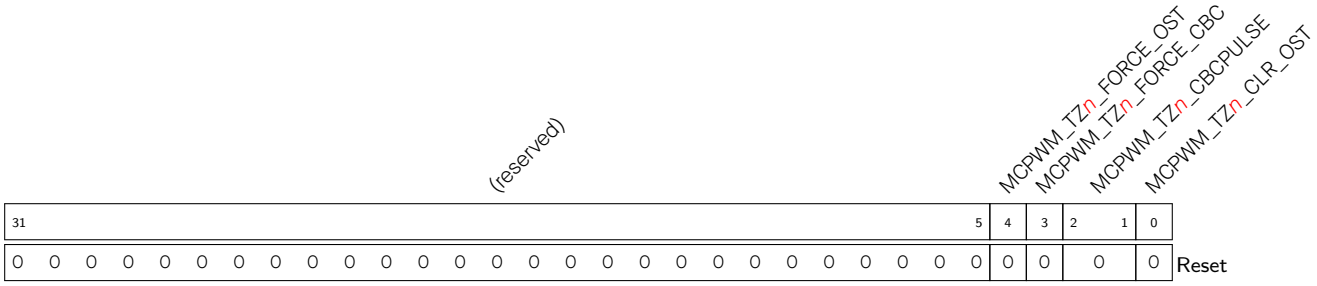
Continued from the previous page...

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 41.20. MCPWM_FH*n*_CFG1_REG(*n*: 0-2) (0x006C+0x38n*)**



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)

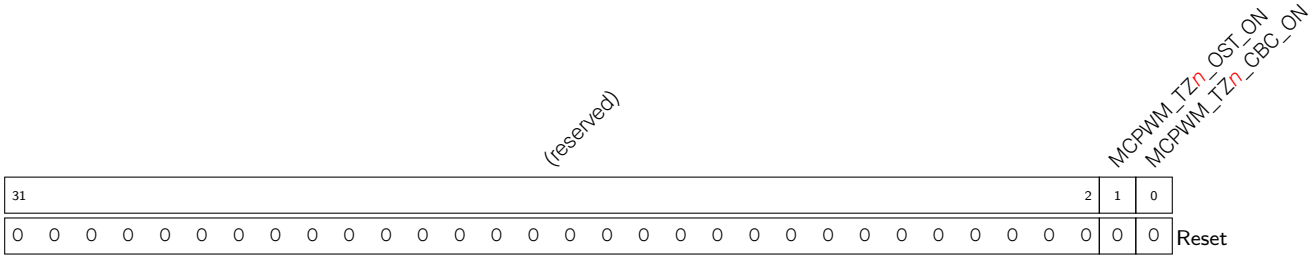
MCPWM_TZ*n*_FORCE_OST Configures whether to generate a software one-shot mode action.

0: No effect

1: Triggers a one-shot mode action by software

(R/W)

Register 41.21. MCPWM_FH*n*_STATUS_REG(*n*: 0-2) (0x0070+0x38n*)**



MCPWM_TZ*n*_CBC_ON Represents whether an cycle-by-cycle mode action is on going.

0: No action

1: Ongoing

(RO)

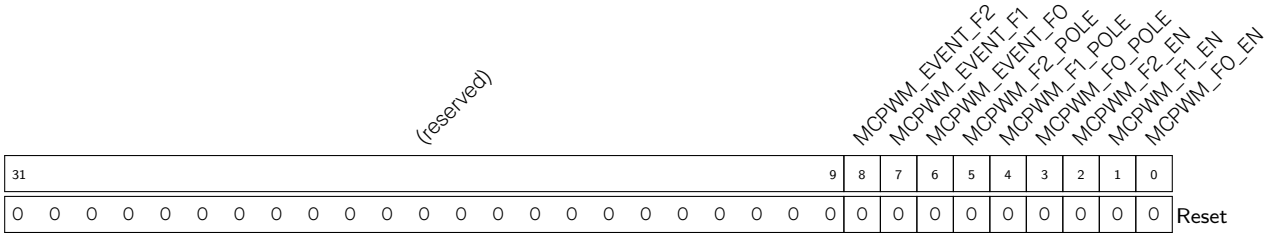
MCPWM_TZ*n*_OST_ON Represents whether a one-shot mode action is ongoing.

0: No action

1: Ongoing

(RO)

Register 41.22. 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)

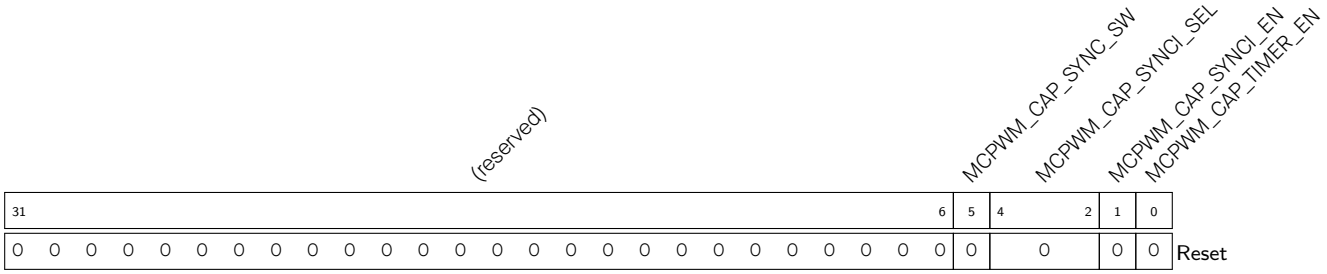
MCPWM_EVENT_F*n* Represents whether an event_f*n* is ongoing. This field is set and reset by hardware.

0: No action

1: Ongoing

(RO)

Register 41.23. 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)

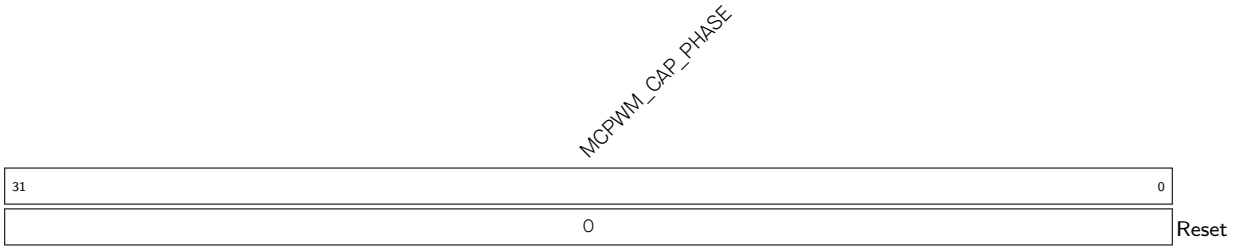
MCPWM_CAP_SYNC_SW When MCPWM_CAP_SYNCI_EN is set to 1, configures whether to trigger a capture timer sync so that capture timer is loaded with value in phase register.

0: No effect

1: Trigger a capture timer sync

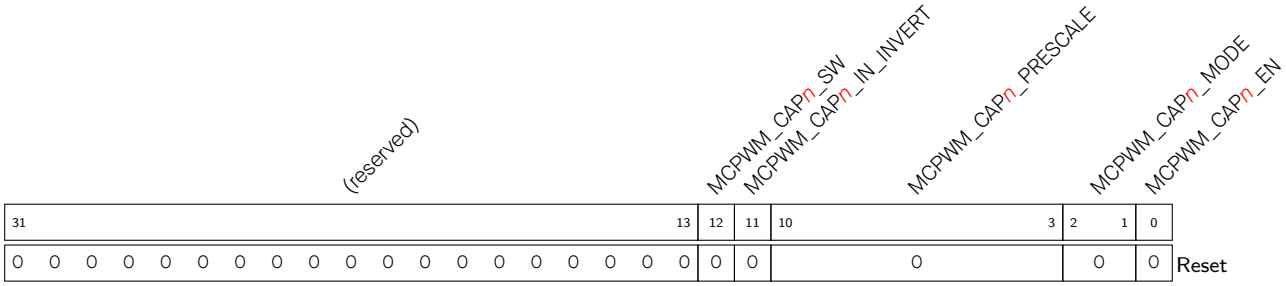
(WT)

Register 41.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_PHASE Configures phase value for capture timer sync operation. (R/W)

Register 41.25. 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)

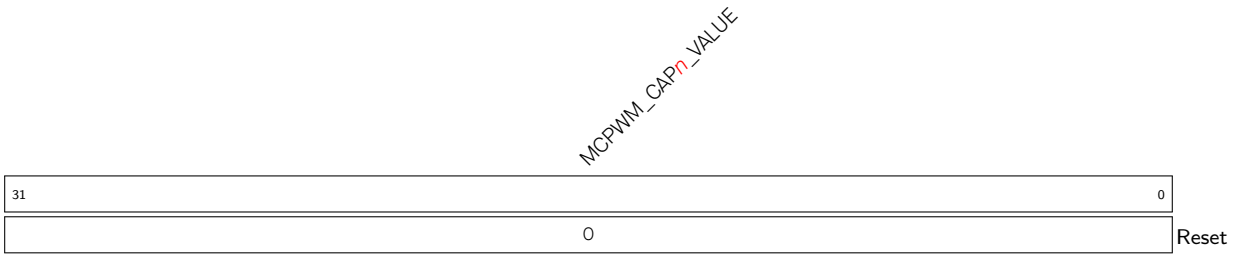
MCPWM_CAP*n*_SW Configures whether to trigger a software-forced capture on channel 0.

0: Not trigger

1: Trigger

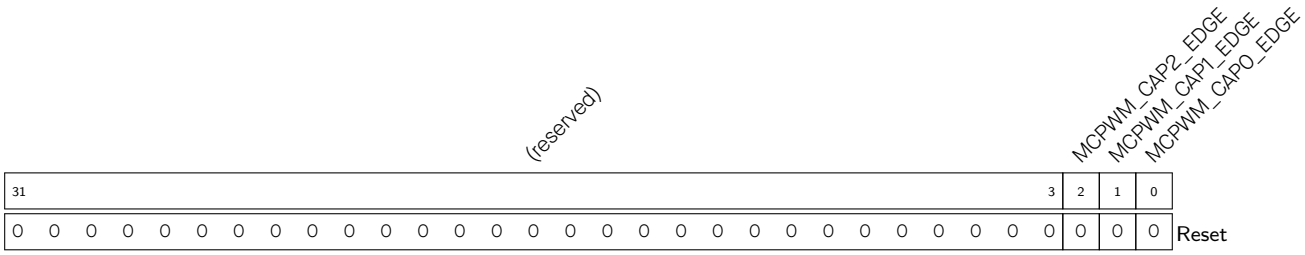
(WT)

Register 41.26. 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 41.27. MCPWM_CAP_STATUS_REG (0x0108)



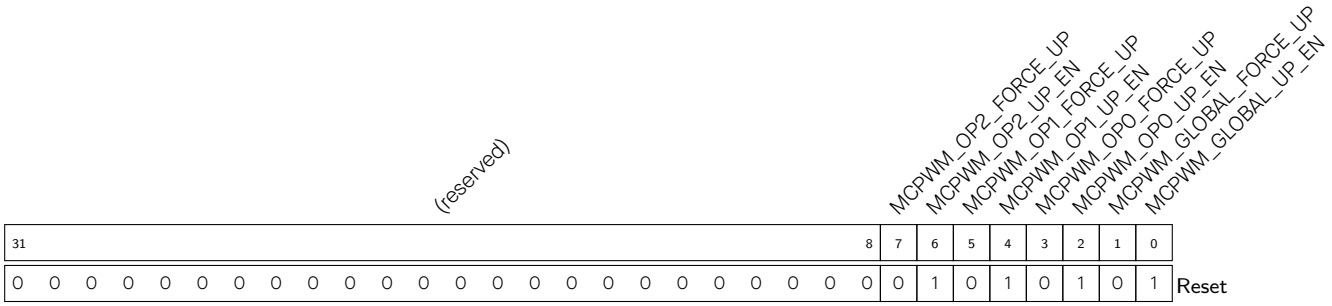
MCPWM_CAP*n*_EDGE Represents the edge of the last capture trigger on channel *n*.

0: Rising edge

1: Falling edge

(RO)

Register 41.28. 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 41.28. 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)

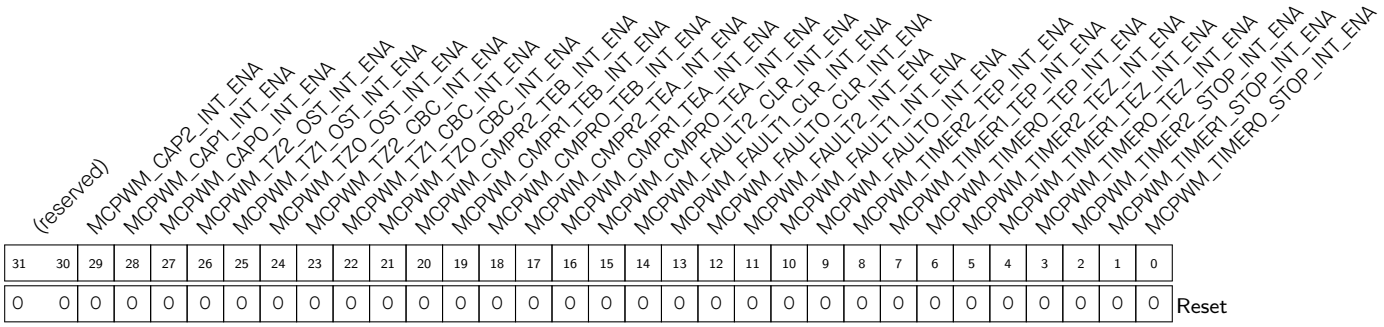
MCPWM_OP2_FORCE_UP Configures whether to trigger a forced update of active registers in PWM operator 2.

0: No effect

1: Trigger a forced update

(R/W)

Register 41.29. 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 41.29. 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)

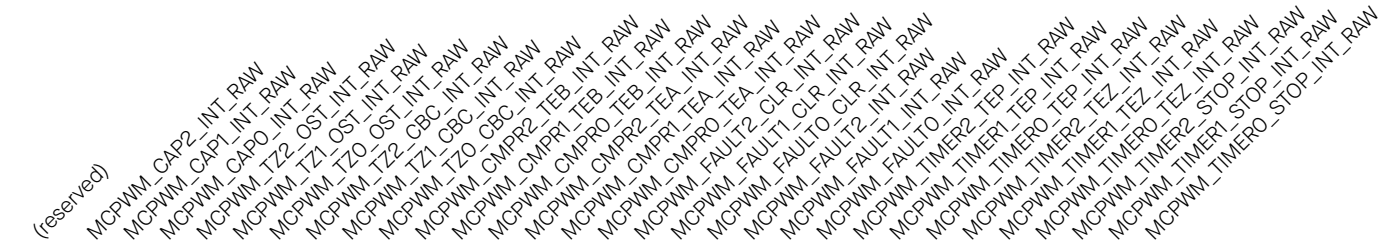
MCPWM_TZ2_OST_INT_ENA Write 1 to enable MCPWM_TZ2_OST_INT. (R/W)

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 41.30. 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 41.30. 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)

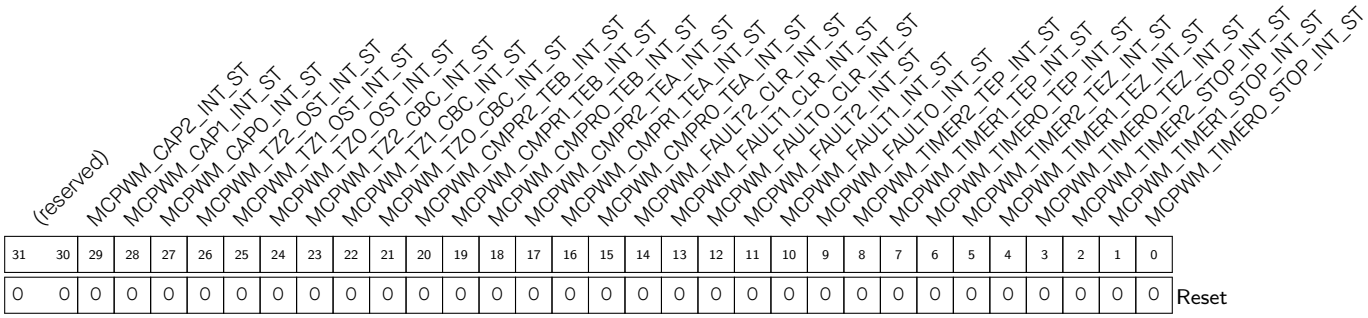
MCPWM_TZ2_OST_INT_RAW Represents the raw status of MCPWM_TZ2_OST_INT. (R/WTC/SS)

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 41.31. 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 41.31. 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)

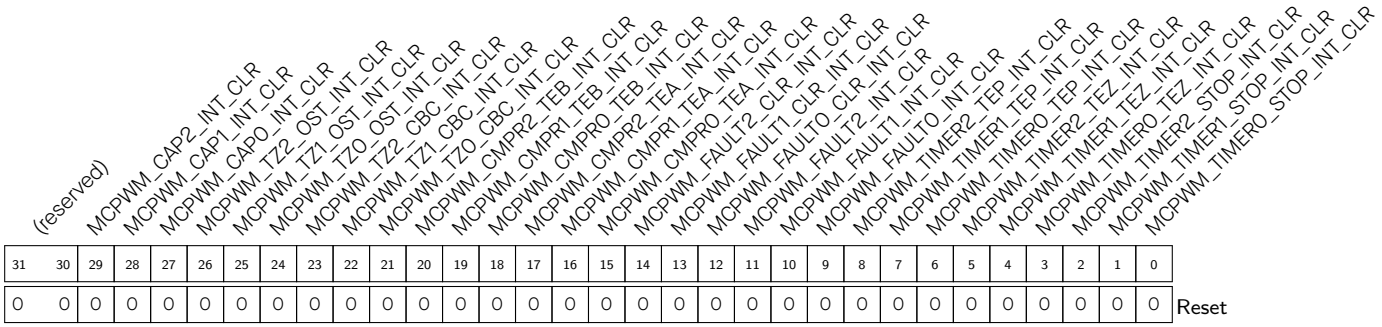
MCPWM_TZ2_OST_INT_ST Represents the masked status of MCPWM_TZ2_OST_INT. (RO)

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 41.32. 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 41.32. 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)

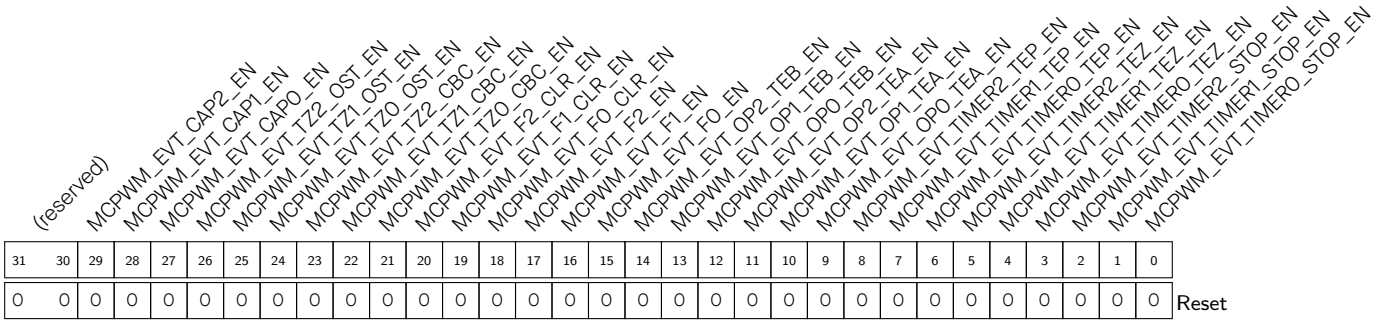
MCPWM_TZ2_OST_INT_CLR Write 1 to clear MCPWM_TZ2_OST_INT. (WT)

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 41.33. 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 41.33. 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 41.33. 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 41.33. 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)

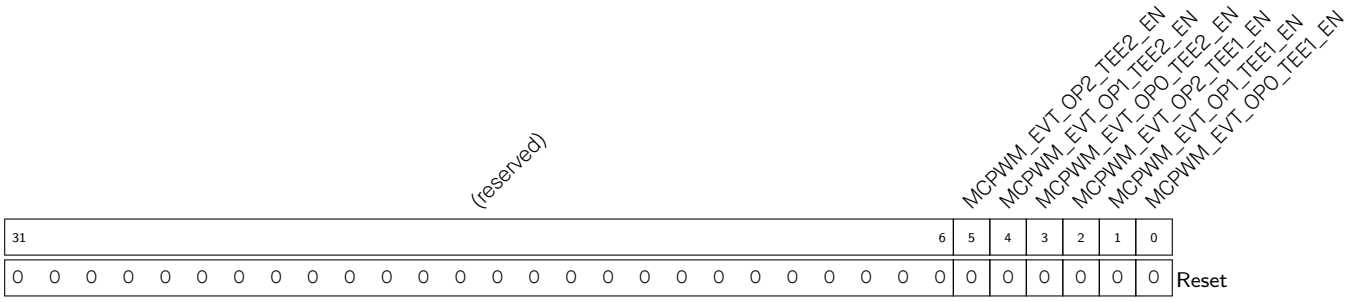
MCPWM_EVT_TZ2_OST_EN Configures whether to enable one shot trip2 event generation.
0: Disable
1: Enable
(R/W)

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 41.34. 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)

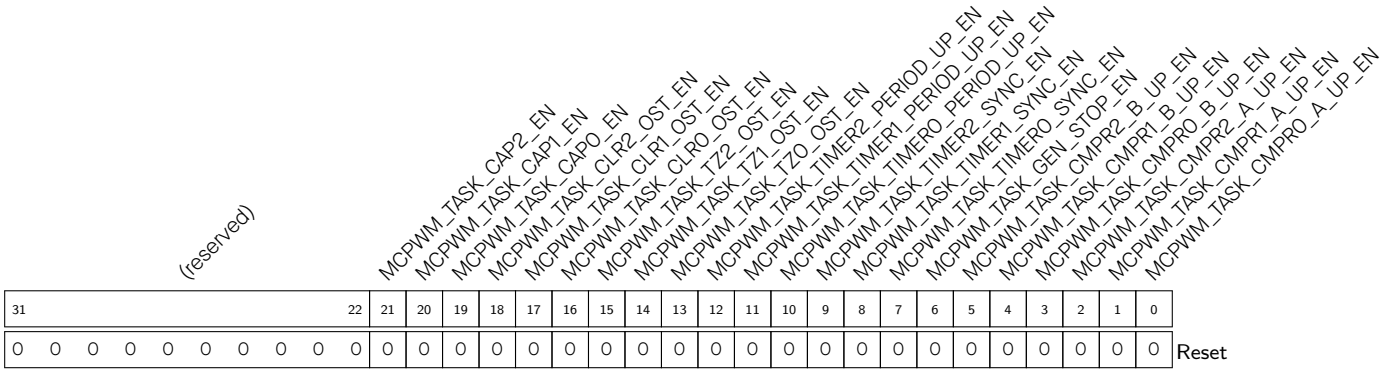
MCPWM_EVT_OP*n*_TEE2_EN Configures whether to generate the MCPWM_EVT_OP*n*_TEE2 event when the PWM generator*n* timer equals OP*n*_TSTMP_E2_REG.

0: Not generate

1: Generate

(R/W)

Register 41.35. 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 41.35. 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 41.35. 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)

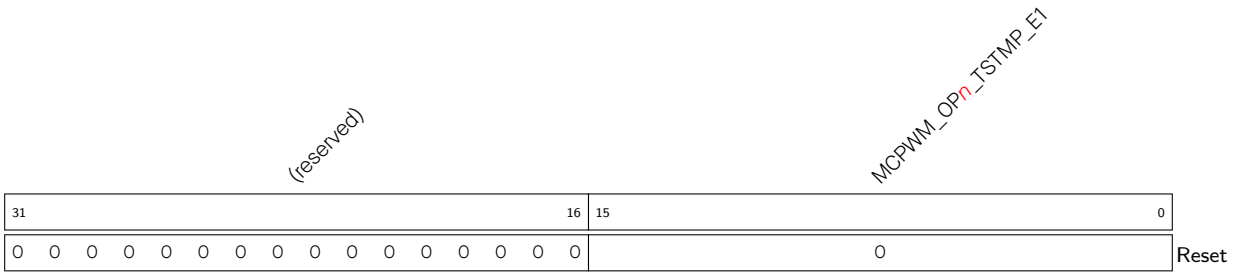
MCPWM_TASK_CAP2_EN Configures whether to receive capture2 task.

0: No effect

1: Receive

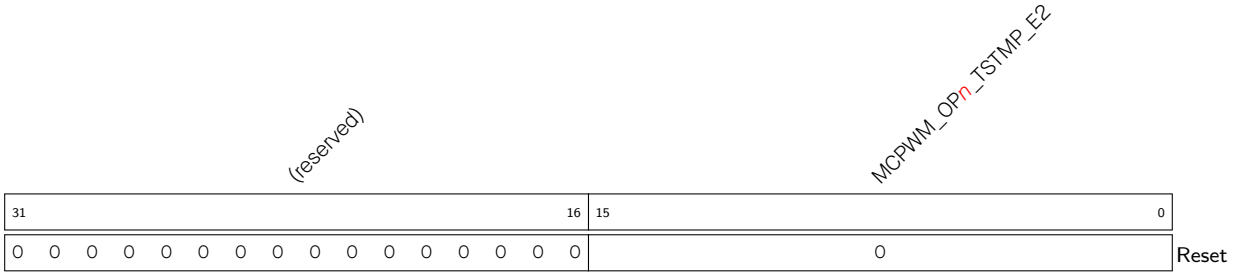
(R/W)

Register 41.36. MCPWM_OP*n*_TSTMP_E1_REG (*n*: 0-2) (0x012C+0x8**n*)



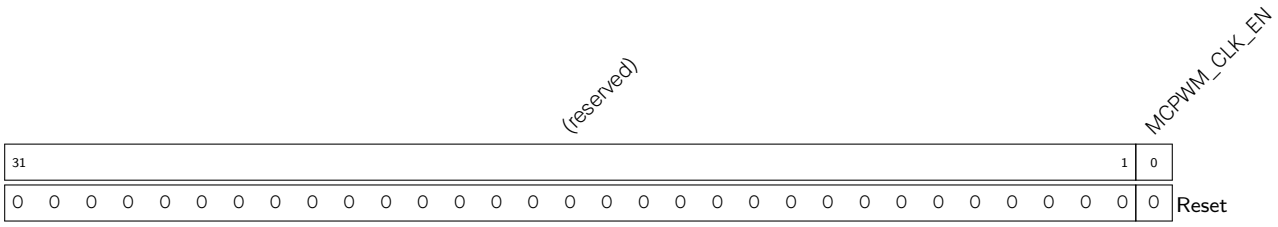
MCPWM_OP*n*_TSTMP_E1 Configures the time stamp E1 value of the generator*n*. (R/W)

Register 41.37. 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 41.38. MCPWM_CLK_REG (0x0144)



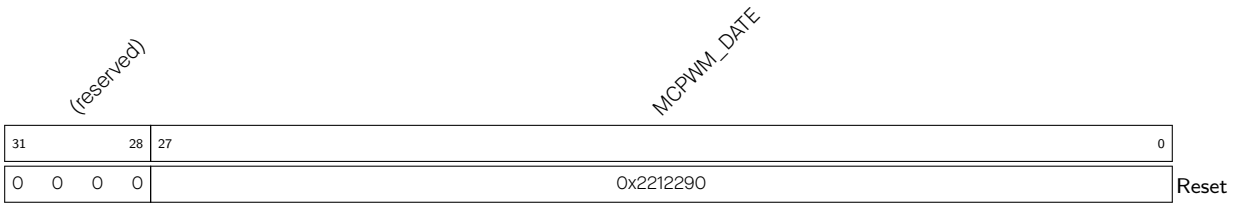
MCPWM_CLK_EN Configures whether to force open register clock gate.

0: Open the clock gate only when application writes registers

1: Force open the clock gate for register

(R/W)

Register 41.39. MCPWM_VERSION_REG (0x0148)



MCPWM_DATE Version control register. (R/W)

Chapter 42

Remote Control Peripheral (RMT)

42.1 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.

42.2 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

42.3 Functional Description

42.3.1 Architecture

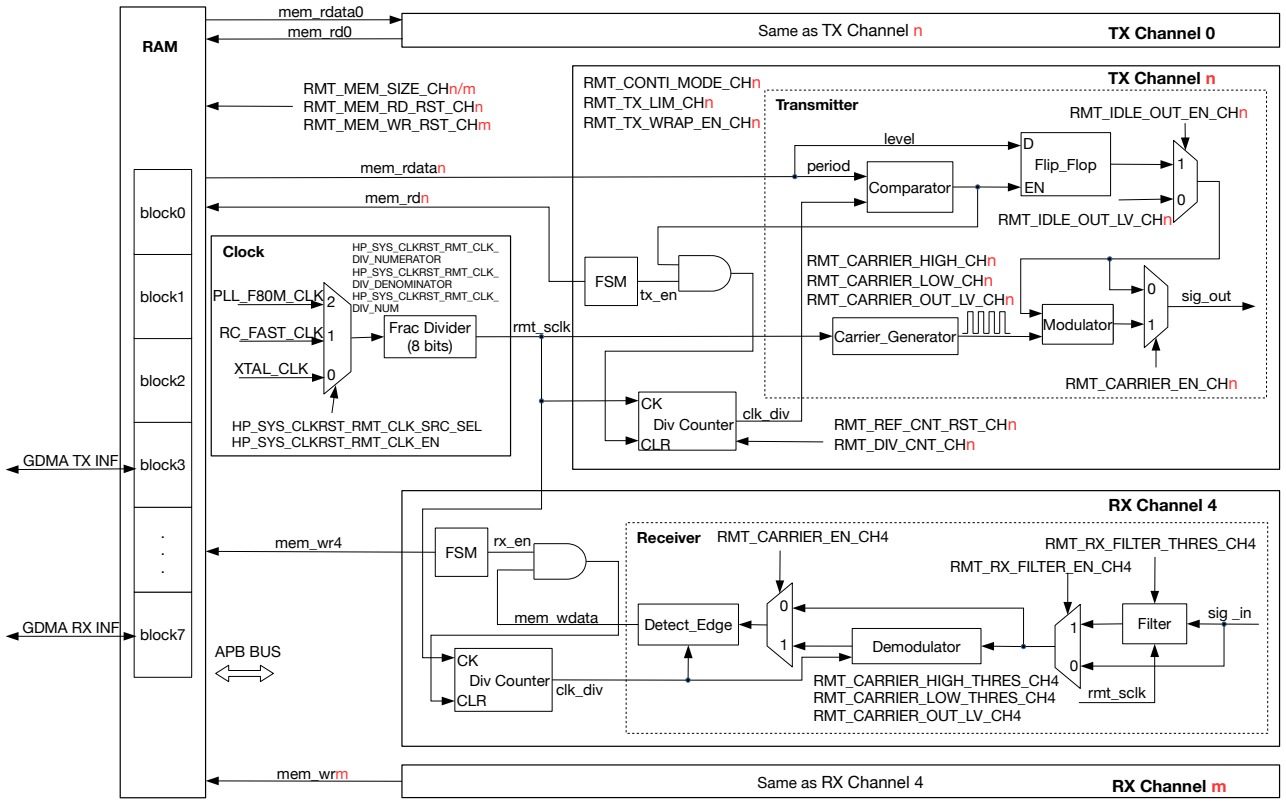


Figure 42.3-1. RMT Architecture

As shown in Figure 42.3-1, 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.

42.3.2 RAM

42.3.2.1 Structure of RAM

Figure 42.3-2 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 42.3-1) that the level lasts for.

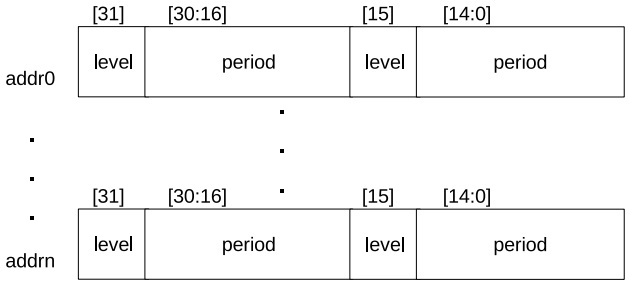


Figure 42.3-2. Format of Pulse Code in RAM

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:

$$3 \times T_{apb\_clk} + 5 \times T_{rmt\_sclk} < period \times T_{clk\_div} \qquad (1)$$

Note:

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 $(3 \times T_{apb\_clk} + 5 \times T_{rmt\_sclk})$.
- 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 42.3.3, or Chapter 8 *Reset and Clock*.

42.3.2.2 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.

42.3.2.3 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* - 1) x 48. The access address for the second data and the following data are RMT base address + 0x800 + (*n* - 1) x 48 + 0x4, and so on, incremented by 0x4.
- The read-starting address of RX channel *m* is: RMT base address + 0x8C0 + (*m* - 1) x 48. The access address for the second data and the following data are RMT base address + 0x8C0 + (*m* - 1) x 48 + 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.

42.3.3 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 42.3-1) is obtained by dividing the selected clock source with a fractional divider. The divider is HP_SYS_CLKRST_RMT_CLK_DIV_NUM + 1 + HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR or HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR.

For more information, see Chapter 8 *Reset and Clock*. RMT_DIV_CNT_CH*n*/*m* is used to configure the divider coefficient of internal clock divider for RMT channels. The coefficient is normally equal to the value of RMT_DIV_CNT_CH*n*/*m*, except value 0 that represents divider 256. The clock divider can be reset by setting RMT_REF_CNT_RST_CH*n*/*m*. The clock generated from the divider can be used by the counter (see Figure 42.3-1).

42.3.4 Transmitter

Note:
Updating the configuration described in this and subsequent sections requires setting RMT_CONF_UPDATE_CH*n*/*m* first. See Section 42.3.6.

42.3.4.1 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*.

42.3.4.2 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.

42.3.4.3 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.

42.3.4.4 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:

$$6 \times T_{apb\_clk} + 12 \times T_{rmt\_sclk} < period \times T_{clk\_div} \qquad (2)$$

The period of the other data only need to satisfy relation (1).

42.3.4.5 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.

42.3.5 Receiver

42.3.5.1 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.

42.3.5.2 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.

42.3.5.3 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.

42.3.5.4 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.

42.3.6 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 42.3-1 should follow this rule.

Table 42.3-1. 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*

42.4 Interrupts

ESP32-P4's RMT can generate the RMT_INTR interrupt signal that will be sent to the Interrupt Matrix.

The following interrupt sources can generate the RMT_INTR interrupt signal:

- RMT_CH*n*_ERR_INT: Triggered when channel *n* does not read or write data correctly. For example, the transmitter still tries to read data from RAM when the RAM is empty.
- RMT_CH*m*_ERR_INT: Triggered when channel *m* does not read or write data correctly. For example, the receiver still tries to write data into RAM when the RAM is full.
- RMT_CH*n*_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the value of RMT_CH*n*_TX_LIM_REG.
- RMT_CH*m*_RX_THR_EVENT_INT: Triggered each time when the amount of data received by the receiver reaches the value set in RMT_CH*m*_RX_LIM_REG.
- RMT_CH*n*_TX_END_INT: Triggered when the transmitter has finished transmitting signals.

- RMT_CH*m*_RX_END_INT: Triggered when the receiver has finished receiving signals.
- RMT_CH*n*_TX_LOOP_INT: Triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CH*n* in continuous TX mode.
- RMT_CH3_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries written to channel 3's RAM - the entries transmitted by channel 3) is larger than channel 3's RAM size, but DAM keeps writing data to this channel.
- RMT_CH7_DMA_ACCESS_FAIL_INT: Triggered when the result of (the entries received by channel 7's RAM - the entries read by GDMA) is larger than channel 7's RAM size, but channel 7 keeps receiving data.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 42.5 *Register Summary*.

42.5 Register Summary

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

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

Name	Description	Address	Access
FIFO 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	Cconfiguration 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

42.6 Registers

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

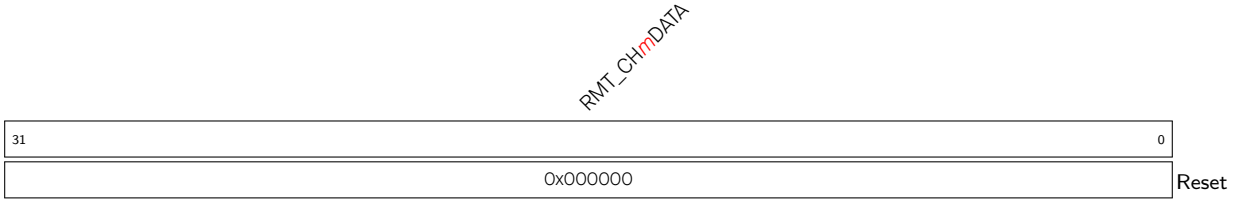
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

Register 42.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4n*)**



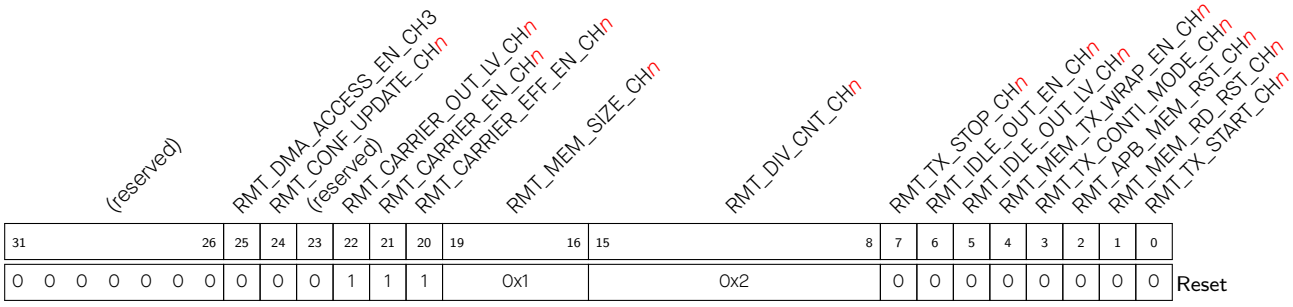
RMT_CH*n*DATA Read and write data for channel *n* via APB FIFO. (HRO)

Register 42.2. 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 42.3. 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 42.3. 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)

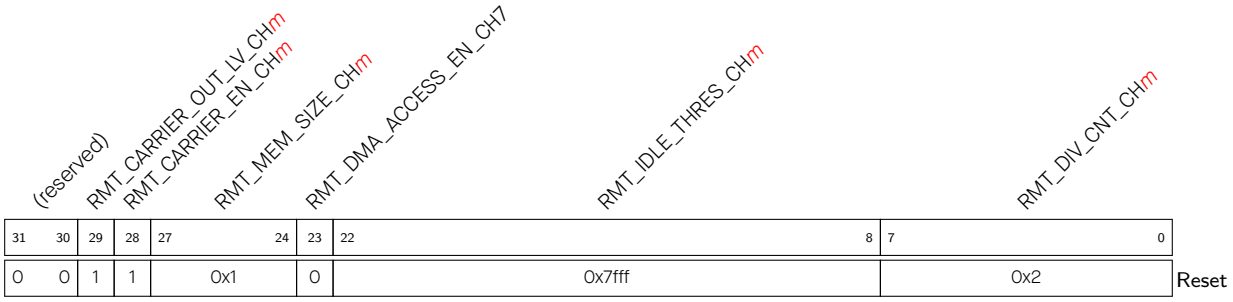
RMT_CARRIER_EN_CH*n* Configures whether to enable the carrier modulation on output signal for channel *n*.
0: Disable
1: Enable
(R/W)

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 42.4. 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)

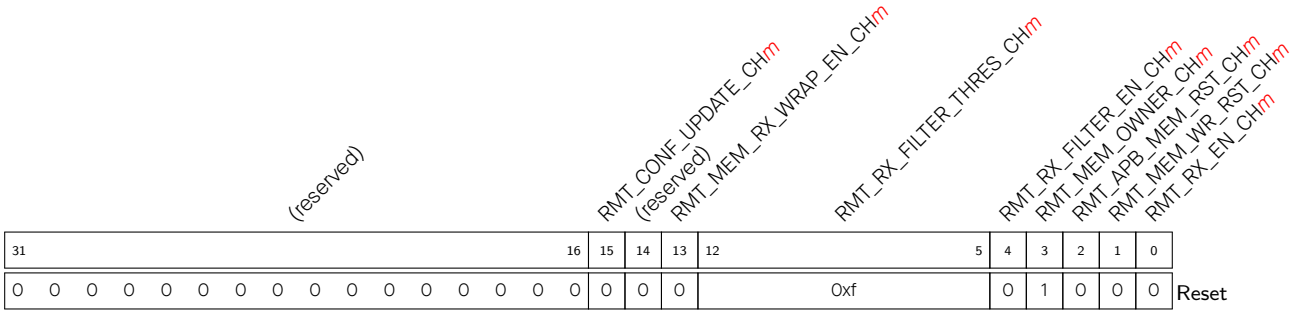
RMT_CARRIER_OUT_LV_CH*m* Configures the position of carrier wave for channel *m*.

0: Demodulate carrier wave on low level

1: Demodulate carrier wave on high level

(R/W)

Register 42.5. 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)

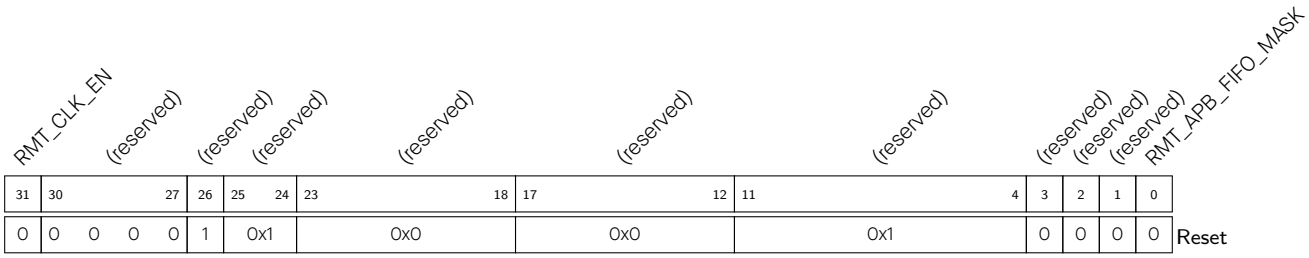
Register 42.6. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 4-7) (0x0090, 0x0094, 0x0098, 0x009C)



RMT_CARRIER_LOW_THRES_CH*m* Configures the low level period in a carrier modulation mode for channel *m*, which is euqal 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 42.7. 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)

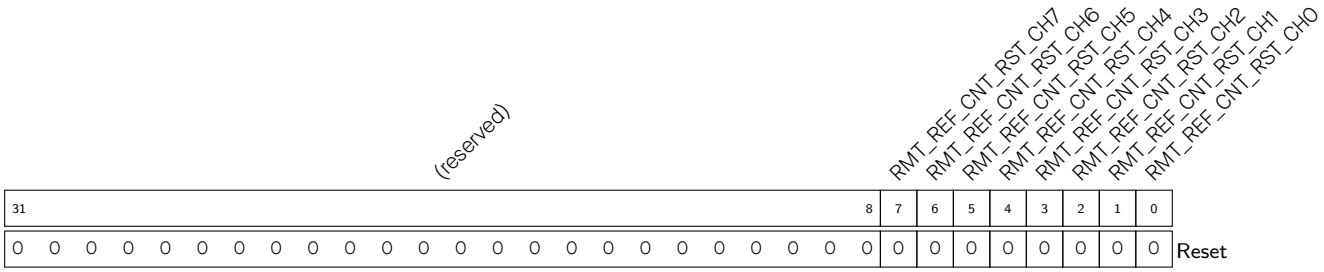
RMT_CLK_EN Configures whether to enable signal of RMT register clock gate.

0: Power down the drive clock of registers

1: Power up the drive clock of registers

(R/W)

Register 42.8. 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)

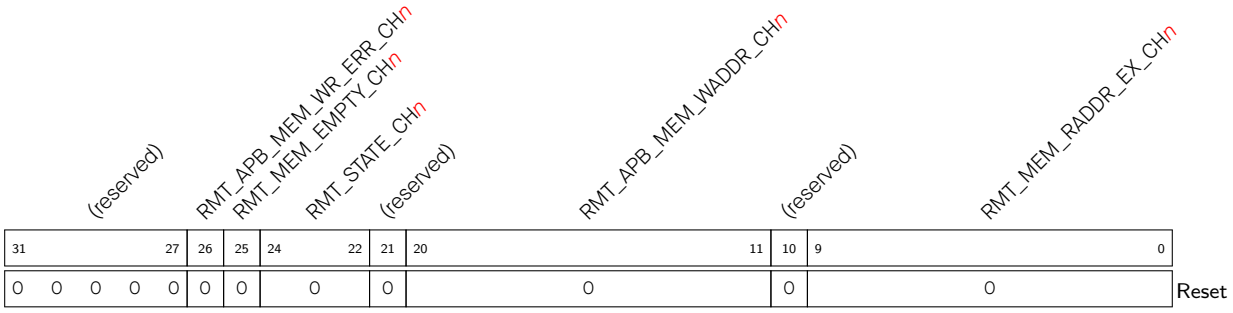
RMT_REF_CNT_RST_CH*m* (*m*: 4-7) Configures whether to reset the clock divider of channel *m*.

0: No effect

1: Reset

(WT)

Register 42.9. 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 42.10. 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)

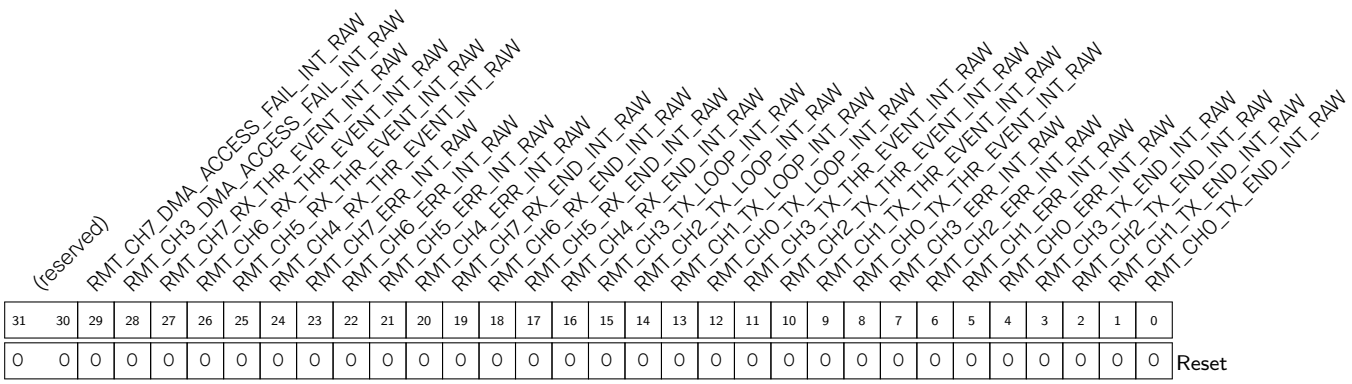
RMT_APB_MEM_RD_ERR_CH*m* Represents whether the offset address exceeds memory size (overflows) when reads RAM via APB bus.

0: Not exceed

1: Exceed

(RO)

Register 42.11. 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)

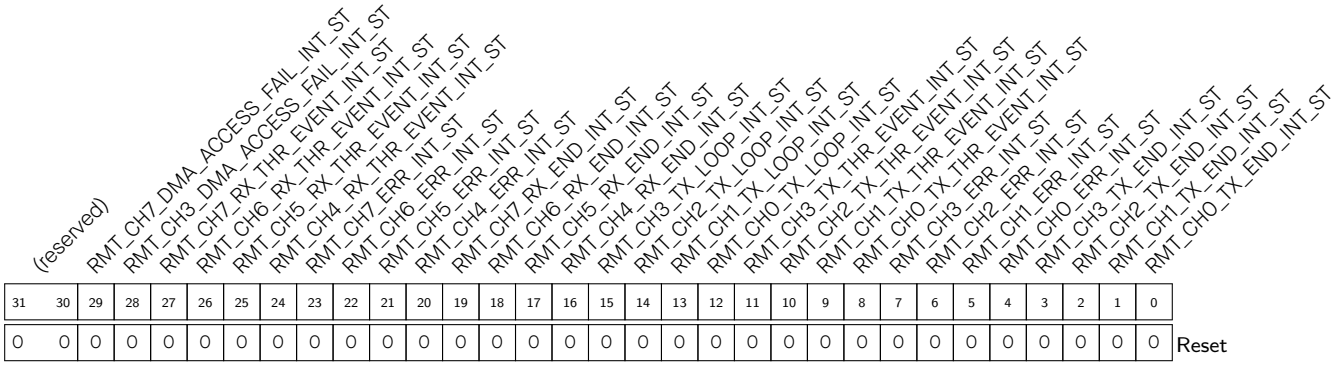
RMT_CH*m*_ERR_INT_RAW (*m*: 4-7) The raw interrupt status of RMT_CH*m*_ERR_INT. (R/WTC/SS)

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 42.12. 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)

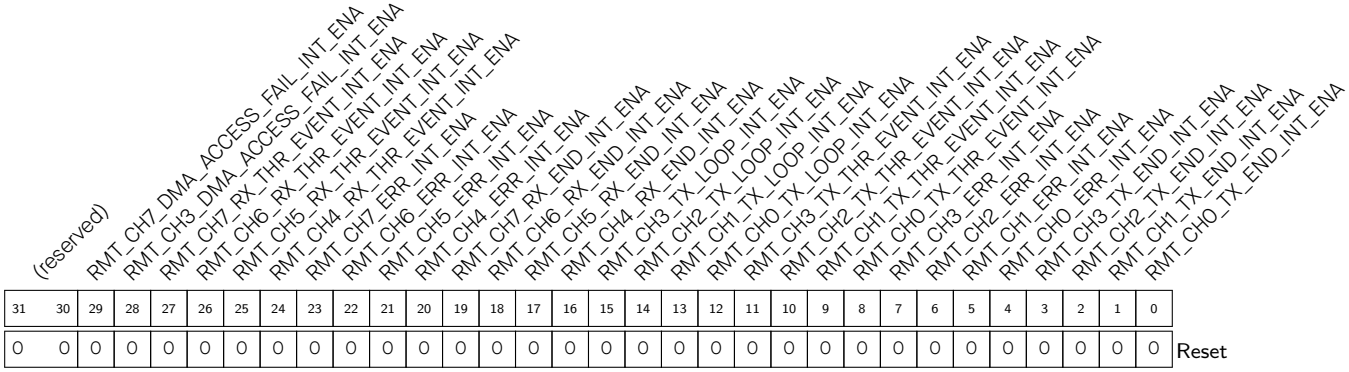
RMT_CH*m*_ERR_INT_ST (*m*: 4-7) The masked interrupt status of RMT_CH*m*_ERR_INT. (RO)

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 42.13. 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)

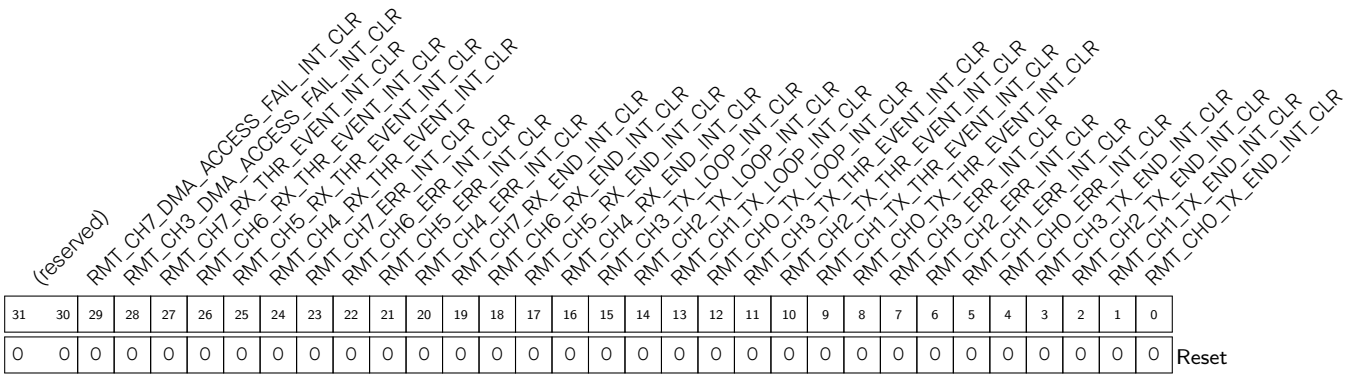
RMT_CH*m*_ERR_INT_ENA (*m*: 4-7) Write 1 to enable RMT_CH*m*_ERR_INT. (R/W)

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 42.14. 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)

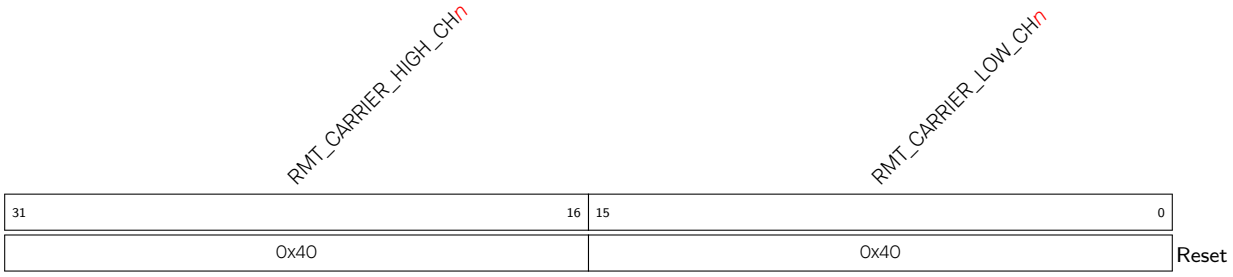
RMT_CH*m*_ERR_INT_CLR (*m*: 4-7) Write 1 to clear RMT_CH*m*_ERR_INT. (WT)

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 42.15. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-3) (0x0080+0x4n*)**



RMT_CARRIER_LOW_CH*n* Configures carrier wave's low level clock period for channel *n*.

Measurement unit: rmt_sclk

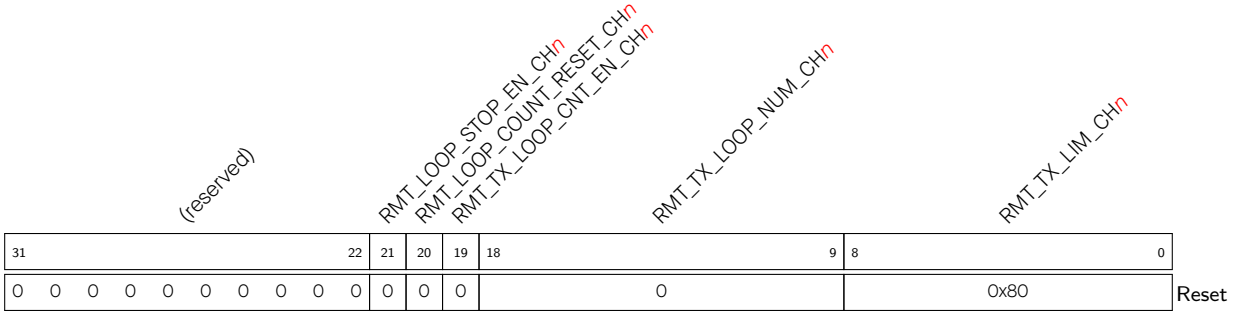
(R/W)

RMT_CARRIER_HIGH_CH*n* Configures carrier wave's high level clock period for channel *n*.

Measurement unit: rmt_sclk

(R/W)

Register 42.16. 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)

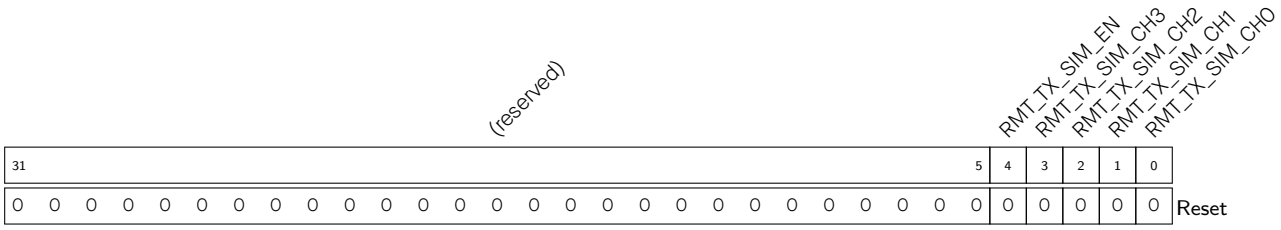
RMT_LOOP_STOP_EN_CH*n* Configures whether to enable the loop send stop function after the loop counter counts to loop number for channel *n*.

0: No effect

1: Enable

(R/W)

Register 42.17. 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)

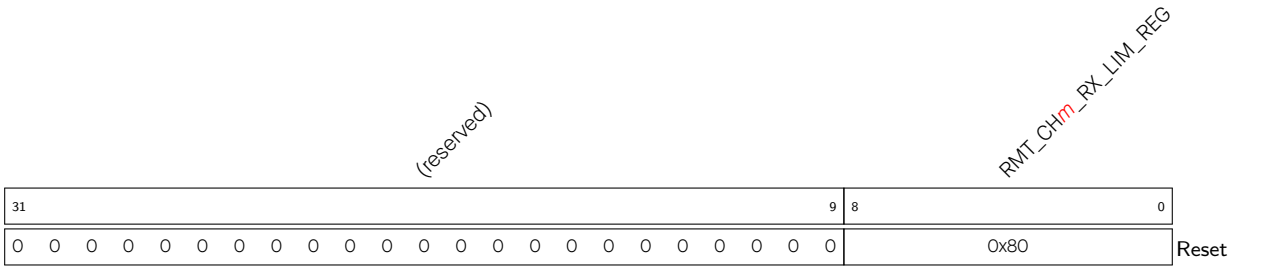
RMT_TX_SIM_EN Configures whether to enable multiple of channels to start sending data synchronously.

0: No effect

1: Enable

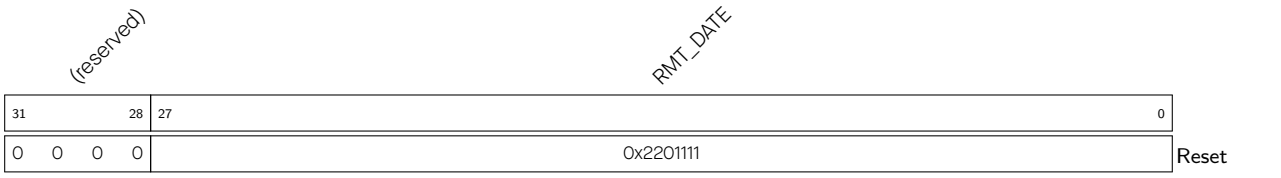
(R/W)

Register 42.18. 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 42.19. RMT_DATE_REG (0x00CC)



RMT_DATE Version control register. (R/W)

Chapter 43

Parallel IO Controller (PARLIO)

43.1 Introduction

ESP32-P4 contains a Parallel IO controller (PARLIO) capable of transferring data between external devices and internal memory on a parallel bus through General Direct Memory Access (GDMA). The PARLIO consists of a TX unit and an RX unit, serving as a transmitter and a receiver respectively. With the two units combined, PARLIO achieves full-duplex communication.

Due to its flexibility, PARLIO can function as a general interface to connect various peripherals. For example, with SPI as the master device and PARLIO as the slave device, a peer-to-peer transfer can be achieved. For detailed application examples, refer to Section 43.8.

43.2 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.

43.3 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:
  - Output clock gating
  - TX unit input and output clock inverse
  - Configurable TX EOF generation
  - Valid signal output
  - Configurable bus idle value

43.4 Architectural Overview

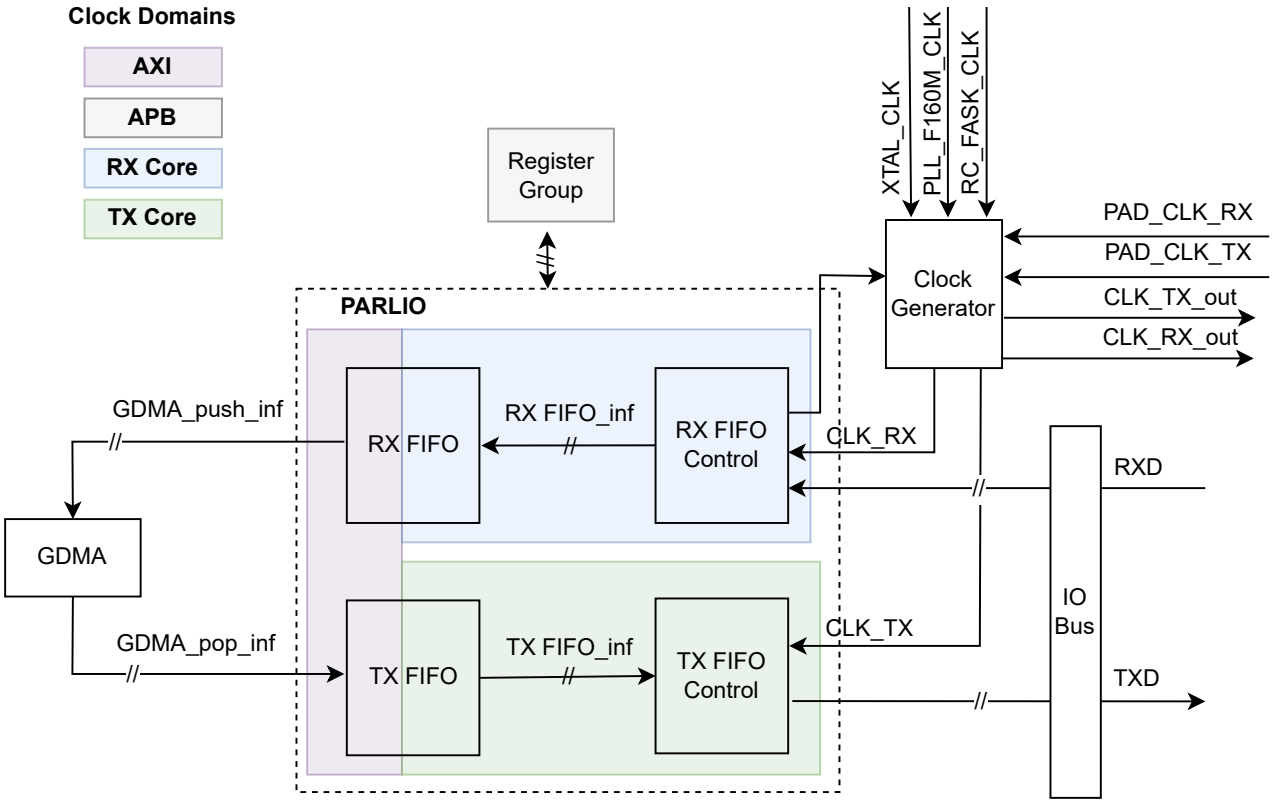


Figure 43.4-1. PARLIO Architecture

Figure 43.4-1 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 synchronizes RXD to the AXI clock domain through an asynchronous FIFO interface, converts RXD to standard GDMA interface data, and sends it to the internal memory.

The TX unit fetches data from internal memory through GDMA, synchronizes the data to the TX Core clock domain through an asynchronous FIFO interface, and converts the data to TXD for parallel IO bus output.

43.5 Functional Description

43.5.1 Clock Generator

There are four input clock domains in PARLIO, namely, RX Core, TX Core, AXI, and APB, as shown in Figure 43.4-1.

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 43.5-1. 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 43.5-1. 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.

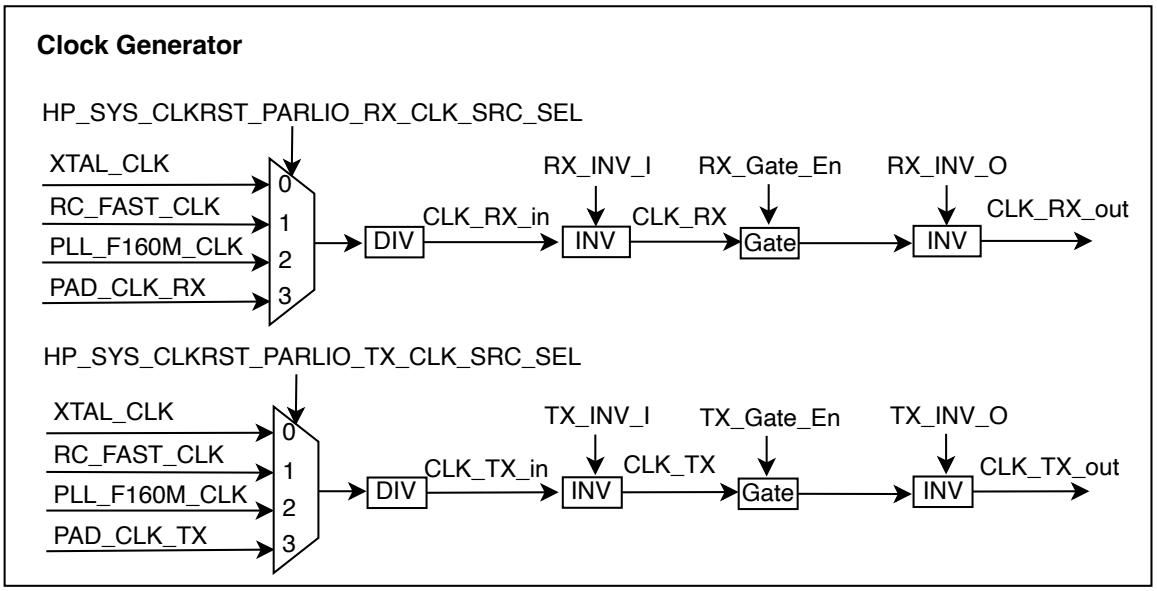


Figure 43.5-1. PARLIO Clock Generation

43.5.2 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 43.5-1. 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.

43.5.3 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 43.5-2. 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 43.5-2)	The master clock should sample TXD at the falling edge.
Not free-running clock	Negative waveform (Figure 43.5-3)	The master device should invert the original clock and convert it to the waveform as Figure 43.5-2 shows before output.

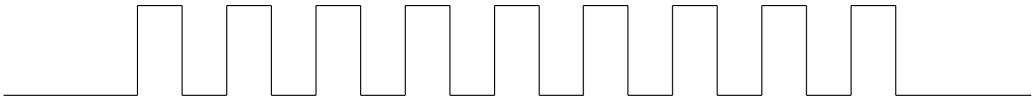


Figure 43.5-2. Positive Waveform

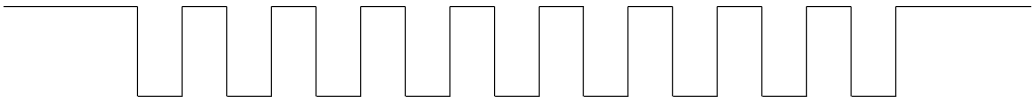


Figure 43.5-3. Negative Waveform

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 43.5-3. 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 43.5-2)	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 43.5-3)	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).

43.5.4 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.

43.5.4.1 Level Enable Mode

Figure 43.5-4 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.

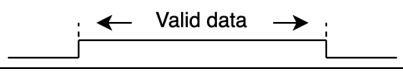
Mode	Sub-mode	Description
LEVEL_ENABLE	\	***signal level high*** 

Figure 43.5-4. Level Enable Mode for RX Unit

43.5.4.2 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 43.5-5.

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.

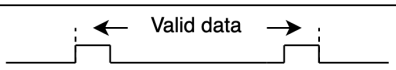
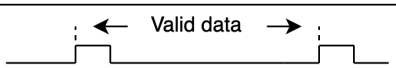
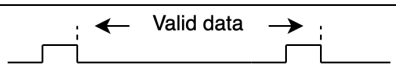
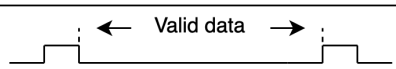
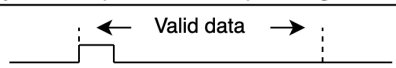
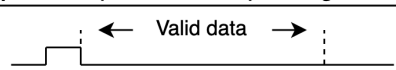
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	***pulse start(data bit include) && pulse end(data bit include)*** 
	sub-mode2	***pulse start(data bit include) && pulse end(data bit exclude)*** 
	sub-mode3	***pulse start(data bit exclude) && pulse end(data bit include)*** 
	sub-mode4	***pulse start(data 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 43.5-5. Sub-Modes of Pulse Enable Mode for RX Unit

43.5.4.3 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.

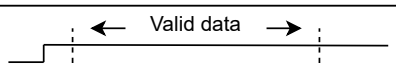
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

Figure 43.5-6. Software Enable Mode for RX Unit

43.5.5 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.

43.5.6 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.

43.5.7 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.

43.5.8 Output Clock Gating of TX Unit

The TX unit supports output clock gating. The clock gating function is disabled by default, and can be enabled by software via setting PARL_IO_TX_GATING_EN. The gating signal is fixed as the highest bit of TXD, and when the bit is at high level, the clock signal can be toggled. There are currently two configurable control sources for the highest bit of TXD, i.e., the GDMA output data and the valid signal. Write 0 to PARL_IO_TX_VALID_OUTPUT_EN to select the GDMA output data as the control source. In this case, it is

required to configure the bit width of the TX unit to a maximum. Write 1 to PARL_IO_TX_VALID_OUTPUT_EN to select the valid signal as the control source. In this case, there is no limit to the bit width.

When the gating function is enabled, there are at most 15 IO pins left usable for TXD as the gating signal occupies one IO.

43.5.9 Valid Signal Output of TX Unit

The TX unit can generate a valid signal aligned with TXD. Configure PARL_IO_TX_VALID_OUTPUT_EN to choose whether to output it to TXD. The polarity of the valid signal is fixed to active high.

The valid output function is disabled by default. When enabled, the output valid signal occupies the most significant bit (MSB) of the TXD, which means that no matter what the original value is, the 15th bit of TXD remains high and is output as the valid signal. However, the valid signal pin does not affect the bus width configuration. For example, if the data bus width is 1 bit, the valid output function can still be enabled with a fixed pin as TXD[15] while the data pin is TXD[0].

Note:
When the valid signal output function and the clock gating function are enabled at the same time, the highest bit of TXD (i.e., the gating signal) is continuously at high level. In cases where software requires the bit not to continuously stay at high level, do not enable the two functions simultaneously.

43.5.10 Bus Idle Value of TX Unit

The TX unit is regarded as in idle state when it is not transmitting data. It supports a configurable bus idle value.

The bus idle value is 0x0 by default, and its maximum configurable value is 0xFF. Note that the configured idle value should not conflict with other enabled functions. For example, when the MSB of TXD is used as the valid signal, users should avoid configuring the MSB of the idle value as 1.

43.5.11 Data Transfer in a Single Frame

The RX unit and the TX unit transfer data in the unit of bits, i.e., a single frame transfers 1 bit of data at least.

When the RX unit generates GDMA EOF signals through bit length, the maximum length of the single-frame transmission is $(2^{19} - 1)$ bits. When the RX unit generates GDMA EOF signals through the enable signal from an external device, there is no limit to the amount of bits of the single-frame transmission.

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.

43.5.12 Bit Reversal in One Byte

The sequence of data within one byte can be reversed when data bus width is 1/2/4-bit. Taking the RX unit as an example, when the configured bus width is 2 bits, the data needs to be packed into one byte before being written into the RX FIFO.

Presume that the original bit sequence is:

{ { b_0, b_1 }, { b_2, b_3 }, { b_4, b_5 }, { b_6, b_7 } }

If the bit reversal is enabled, the sequence will be reordered to:

{ { b_6, b_7 }, { b_4, b_5 }, { b_2, b_3 }, { b_0, b_1 } }

43.6 Interrupts

ESP32-P4's PARLIO module can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

There are several internal interrupt sources from PARLIO that can generate the above interrupt signals. The interrupt sources from PARLIO are listed with their trigger conditions and the resulted interrupt signals in Table 43.6-1.

Table 43.6-1. 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 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 43.9 *Register Summary*.

43.7 Programming Procedures

43.7.1 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 43.5.2.

1. Reset the RX unit. For specific reset scenarios and sequences, refer to Section 43.5.2.
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 43.3 and 43.5.
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.

43.7.2 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 43.5.2.

1. Reset the TX unit. For specific reset scenarios and sequences, refer to Section 43.5.2.
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 43.5.

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.

43.8 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.

43.8.1 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.

43.8.2 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.

43.8.3 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.

43.9 Register Summary

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

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

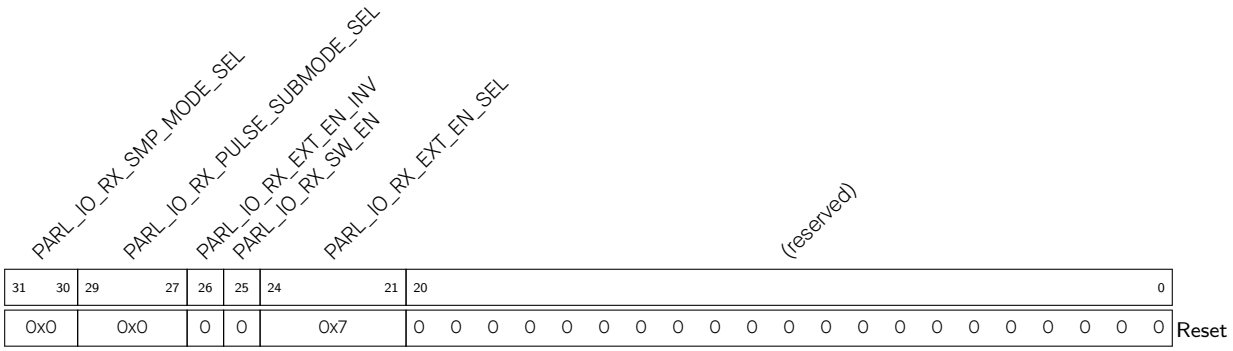
Name	Description	Address	Access
PARLIO RX Configuration Registers			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX sampling mode configuration register	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX data configuration register	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX general configuration register	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX start configuration register	0x000C	R/W
PARLIO TX Configuration Registers			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX data configuration register	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX start configuration register	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX general configuration register	0x0018	R/W
PARLIO Configuration and Status Registers			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO configuration register	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO register update configuration register	0x0020	WT
PARL_IO_ST_REG	PARLIO module status register	0x0024	RO
PARLIO Interrupt Configuration and Status Registers			
PARL_IO_INT_ENA_REG	PARLIO interrupt enable signal configuration register	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO interrupt raw signal status register	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO interrupt signal status register	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO interrupt clear signal configuration register	0x0034	WT
PARLIO RX/TX Status Registers			
PARL_IO_RX_ST0_REG	PARLIO RX status register 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX status register 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX status register 0	0x0040	RO
PARLIO Clock Configuration Registers			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX clock configuration register	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX clock configuration register	0x0048	R/W
PARL_IO_CLK_REG	PARLIO clock configuration register	0x0120	R/W
PARLIO Version Register			
PARL_IO_VERSION_REG	Version control register	0x03FC	R/W

43.10 Registers

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

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

Register 43.1. 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.

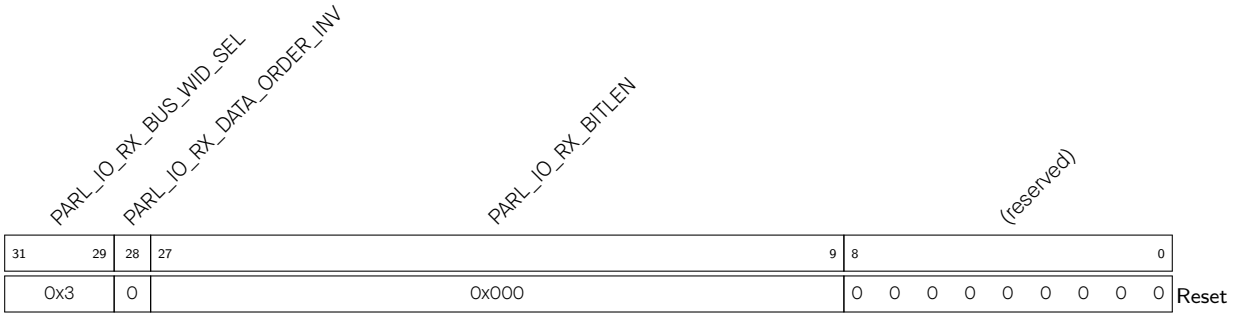
0: External Level Enable mode

1: External Pulse Enable mode

2: Internal Software Enable mode

(R/W)

Register 43.2. 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

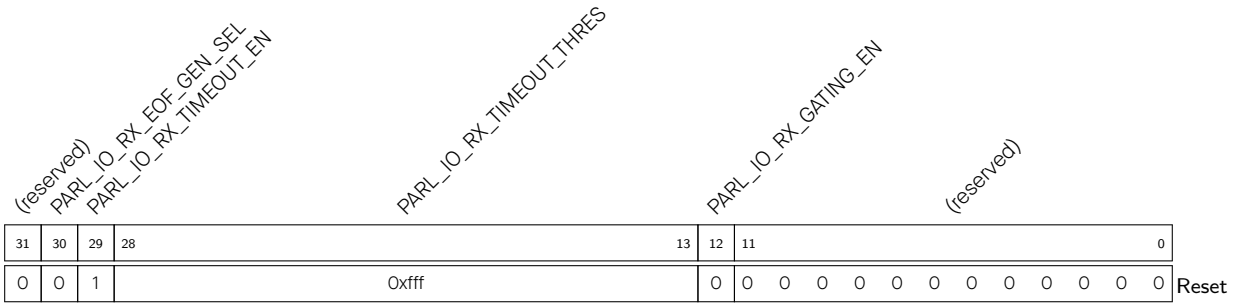
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 43.3. 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)

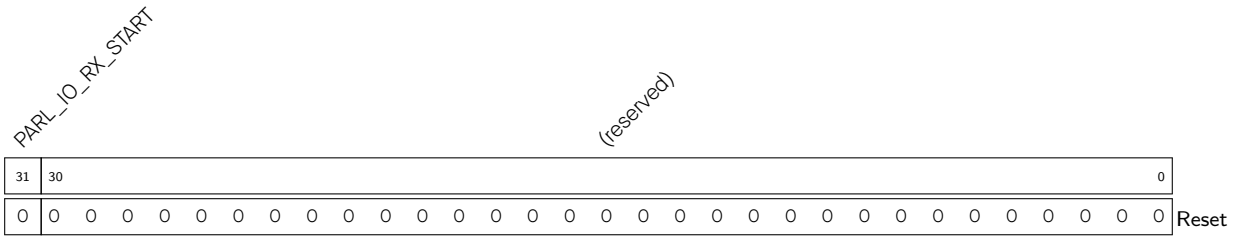
PARL_IO_RX_EOF_GEN_SEL Configures the generation mechanism of GDMA SUC EOF.

0: Generate GDMA SUC EOF by the configured data bit length

1: Generate GDMA SUC EOF by the external enable signal

(R/W)

Register 43.4. PARL_IO_RX_START_CFG_REG (0x000C)



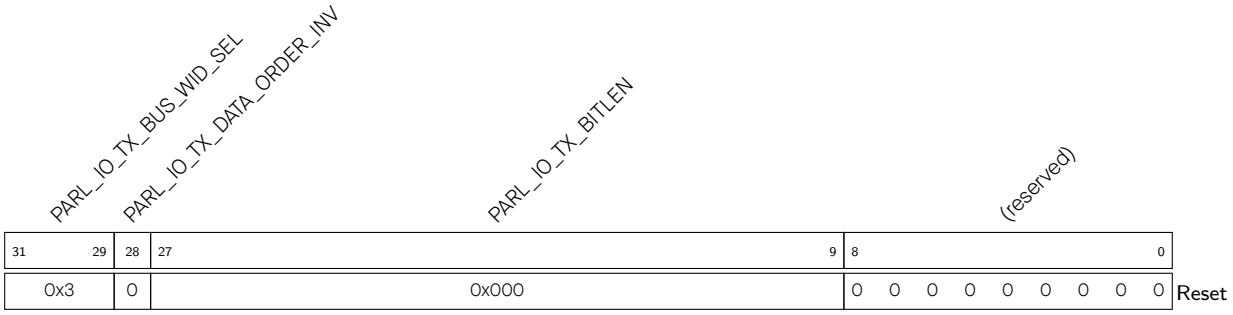
PARL_IO_RX_START Configures whether to start RX data sampling.

0: No effect

1: Start

(R/W)

Register 43.5. 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

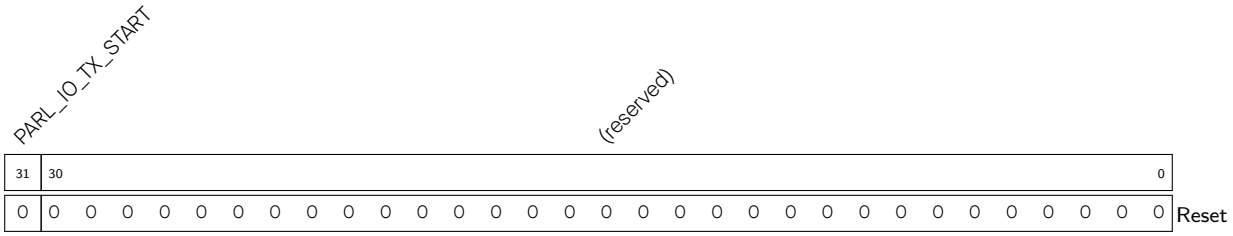
3: 8 bits

4: 16 bits

5 ~ 15: Invalid and will incur error

(R/W)

Register 43.6. PARL_IO_TX_START_CFG_REG (0x0014)



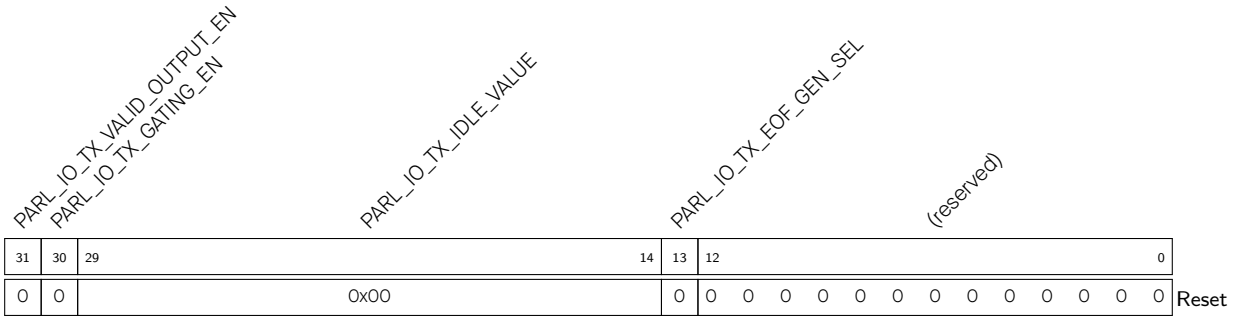
PARL_IO_TX_START Configures whether to start TX data transmission.

0: No effect

1: Start

(R/W)

Register 43.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL Configures the generation mechanism of TX EOF.

0: Generate TX EOF by the configured data bit length

1: Generate TX EOF by the GDMA EOF signal.

(R/W)

PARL_IO_TX_IDLE_VALUE Configures the data value on TX bus in idle state. (R/W)

PARL_IO_TX_GATING_EN Configures whether to enable the clock gating of the TX output clock.

0: Disable

1: Enable

(R/W)

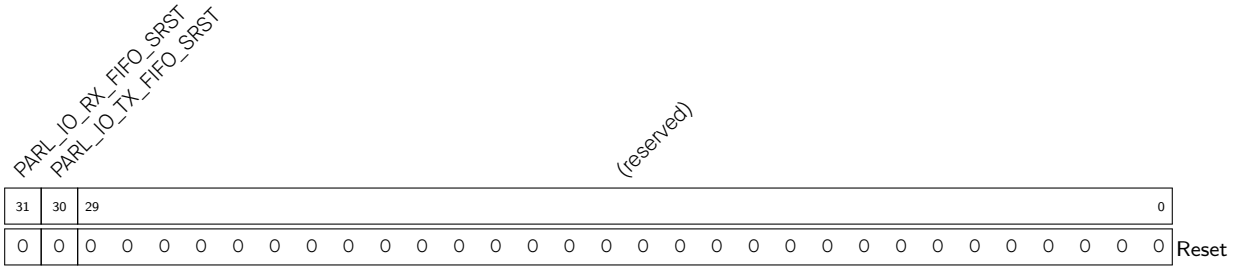
PARL_IO_TX_VALID_OUTPUT_EN Configures whether to enable the output of TX data valid signal.

0: Disable

1: Enable

(R/W)

Register 43.8. 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)

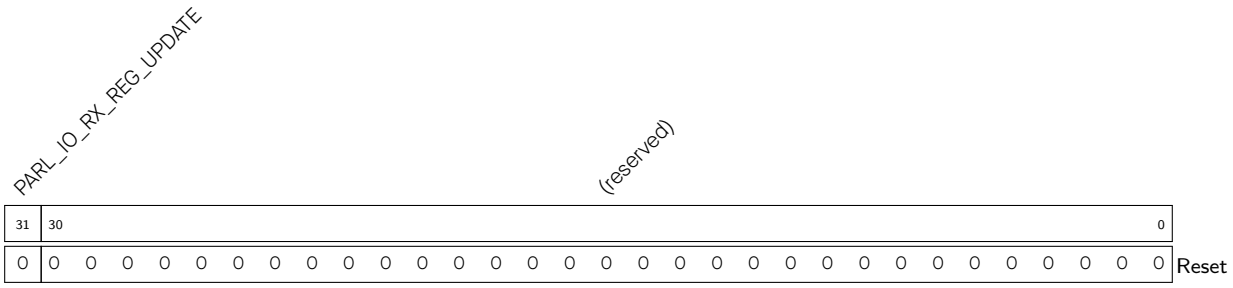
PARL_IO_RX_FIFO_SRST Configures whether to reset async FIFO in the RX unit.

0: No effect

1: Reset

(R/W)

Register 43.9. PARL_IO_REG_UPDATE_REG (0x0020)



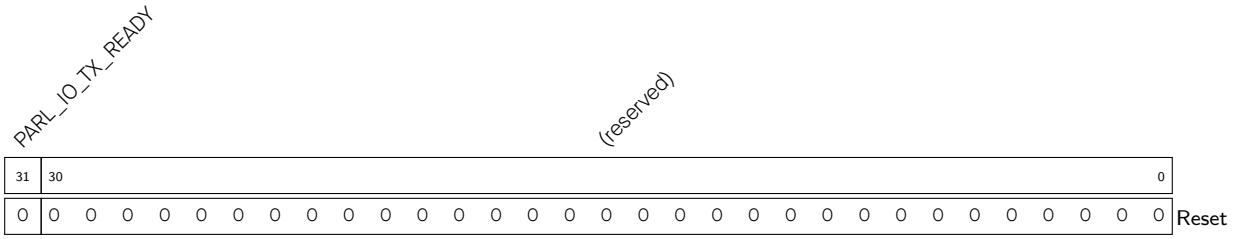
PARL_IO_RX_REG_UPDATE Configures whether to update RX register configuration.

0: No effect

1: Update

(WT)

Register 43.10. PARL_IO_ST_REG (0x0024)



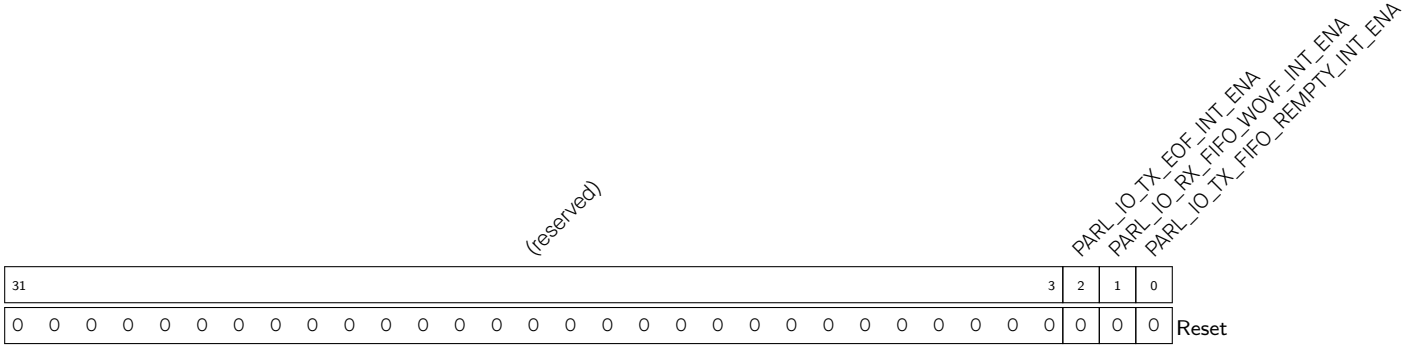
PARL_IO_TX_READY Represents whether TX is ready to transmit data.

0: Not ready

1: Ready

(RO)

Register 43.11. PARL_IO_INT_ENA_REG (0x0028)

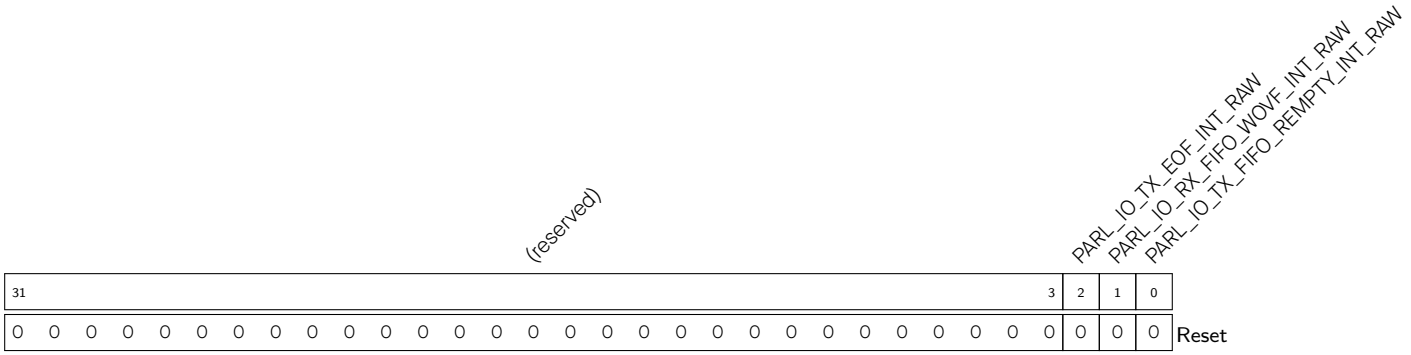


PARL_IO_TX_FIFO_REMPTY_INT_ENA Write 1 to enable TX_FIFO_REMPTY_INT. (R/W)

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 43.12. PARL_IO_INT_RAW_REG (0x002C)

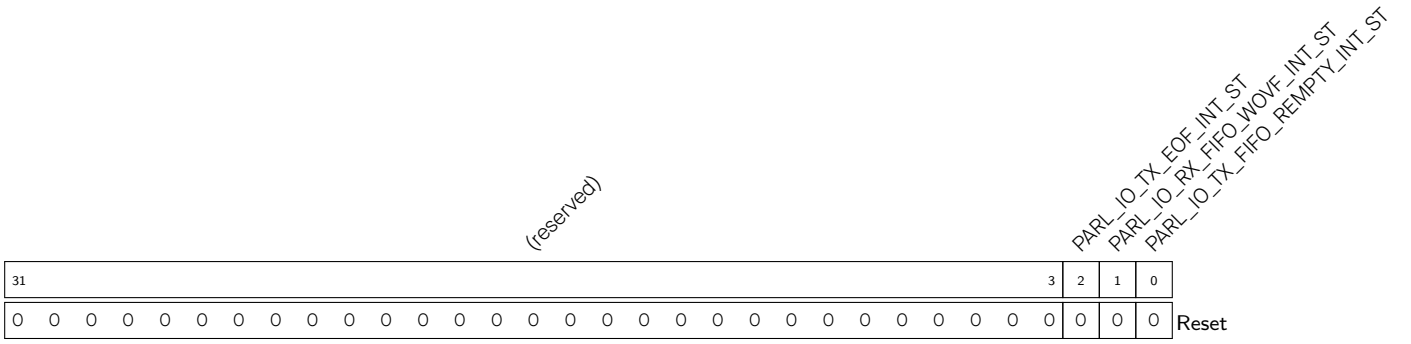


PARL_IO_TX_FIFO_REMPTY_INT_RAW The raw interrupt status of TX_FIFO_REMPTY_INT. (R/WTC/SS)

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 43.13. PARL_IO_INT_ST_REG (0x0030)

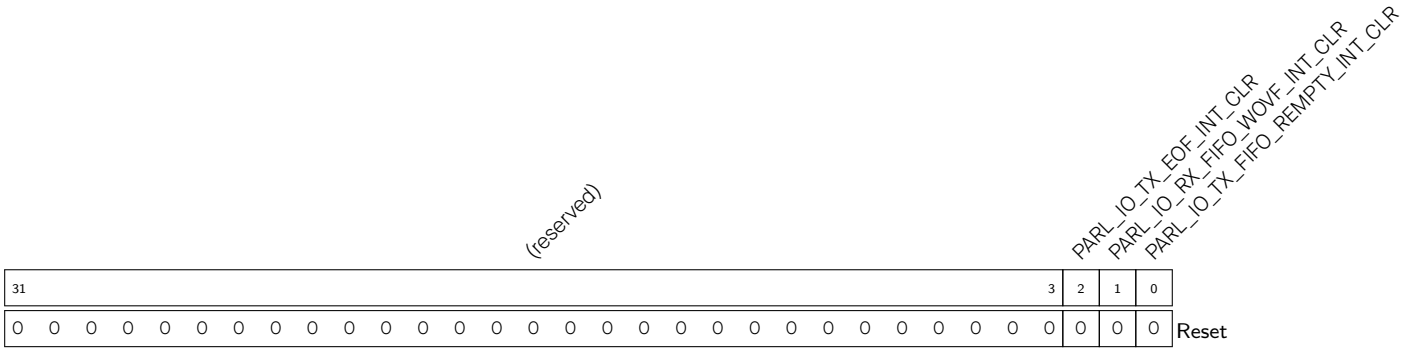


PARL_IO_TX_FIFO_REMPTY_INT_ST The masked interrupt status of TX_FIFO_REMPTY_INT. (RO)

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 43.14. PARL_IO_INT_CLR_REG (0x0034)

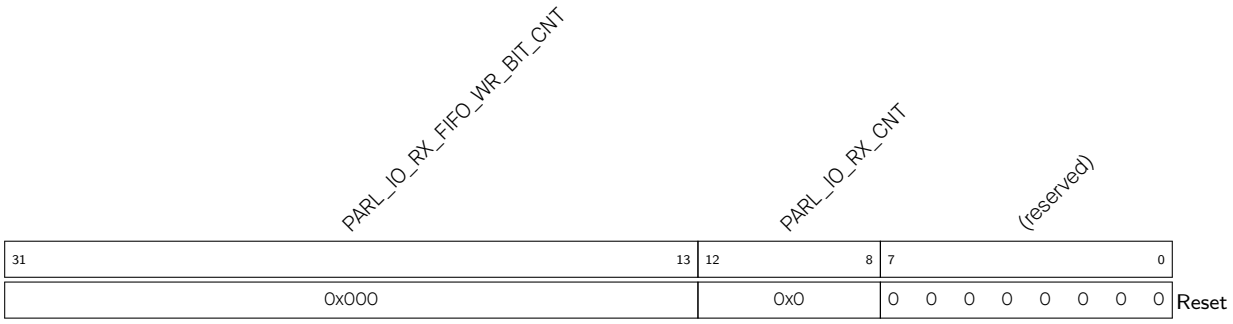


PARL_IO_TX_FIFO_REMPTY_INT_CLR Write 1 to clear TX_FIFO_REMPTY_INT. (WT)

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)

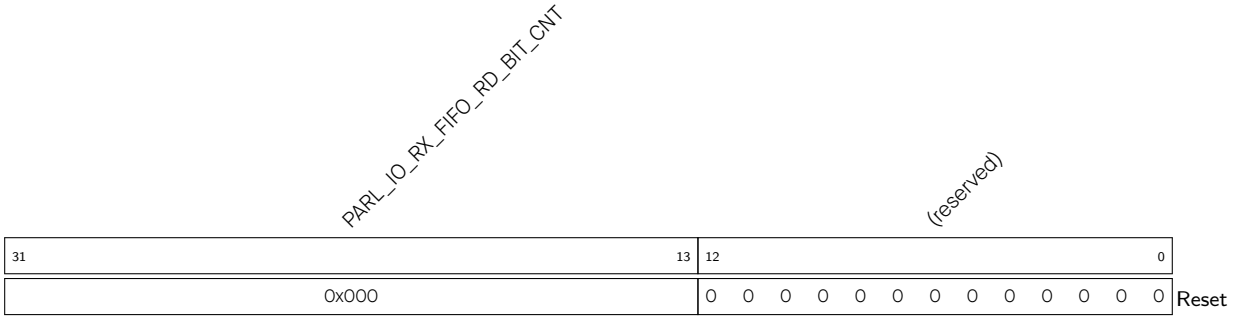
Register 43.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT Represents the clock cycle number of reading the RX FIFO. (RO)

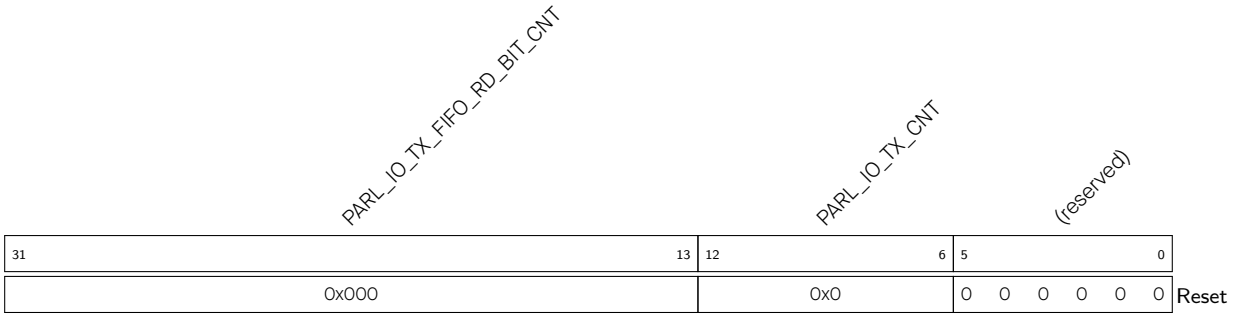
PARL_IO_RX_FIFO_WR_BIT_CNT Represents the bit number currently written into the RX FIFO. (RO)

Register 43.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT Represents the bit number currently read from the RX FIFO. (RO)

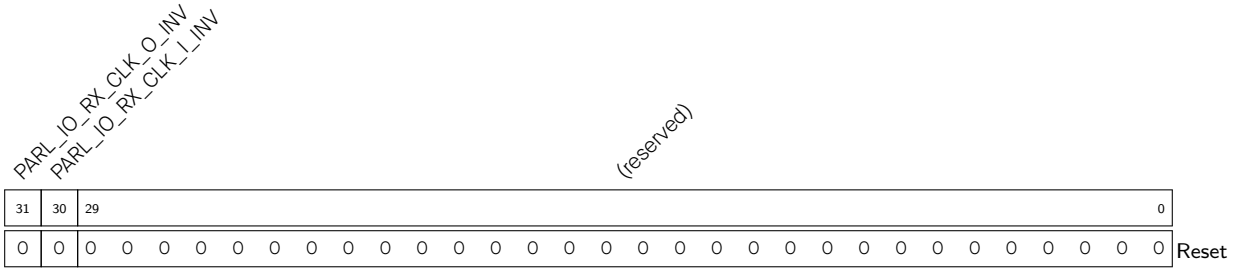
Register 43.17. PARL_IO_TX_ST0_REG (0x0040)



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 43.18. 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)

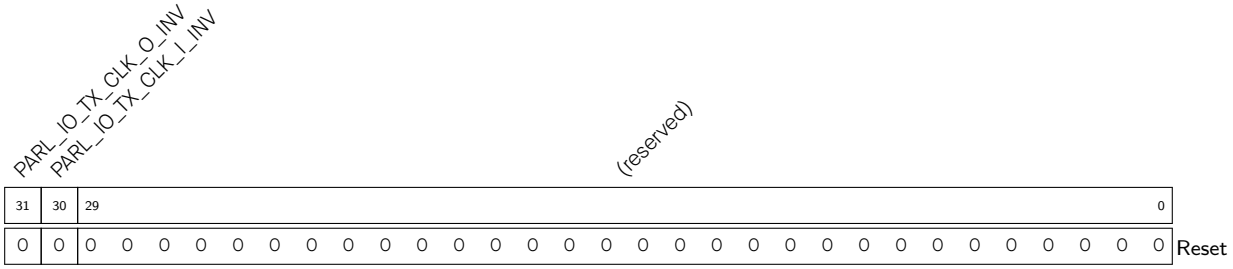
PARL_IO_RX_CLK_O_INV Configures whether to invert the RX output Core clock.

0: No effect

1: Invert

(R/W)

Register 43.19. 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)

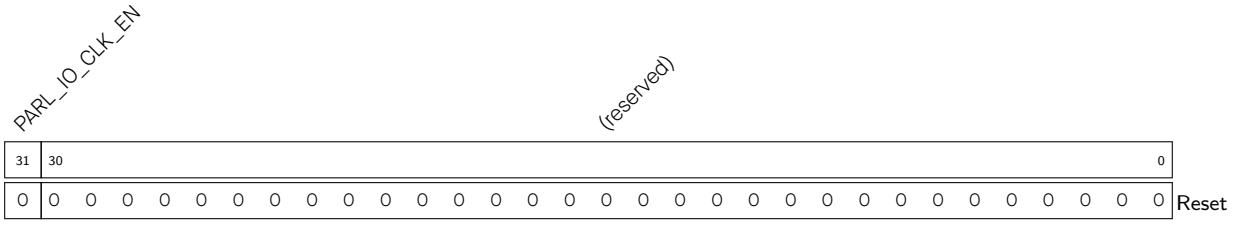
PARL_IO_TX_CLK_O_INV Configures whether to invert the TX output Core clock.

0: No effect

1: Invert

(R/W)

Register 43.20. PARL_IO_CLK_REG (0x0120)



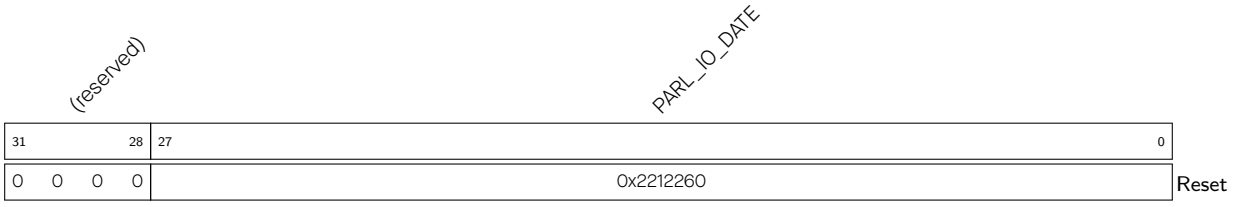
PARL_IO_CLK_EN Register clock gating.

0: Enable the register clock only when reading or writing registers

1: Always enable the register clock

(R/W)

Register 43.21. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE Version control register. (R/W)

Part VIII

Analog Signal Processing

This part describes components related to analog-to-digital conversion, voltage comparison, on-chip sensors, and features such as temperature and touch sensing, demonstrating the system's capabilities in handling analog signals.

Chapter 44

Temperature Sensor (TSENS)

44.1 Overview

ESP32-P4 provides a temperature sensor for real-time monitoring of temperature changes within the chip. The sensor converts analog voltage to digital values and provides compensation for temperature offsets.

44.2 Features

The temperature sensor has the following features:

- Software-triggered temperature measurement, which once triggered, the sensor continuously measures temperature. Software can read the data at any time.
- Hardware-triggered automatic temperature monitoring, supporting two wake-up modes
- Configurable temperature offset based on the application scenario for improved accuracy
- Configurable temperature measurement range
- Support for Event Task Matrix (ETM)-related events and tasks

44.3 Architecture

Figure 44.3-1 shows the internal structure of the temperature sensor.

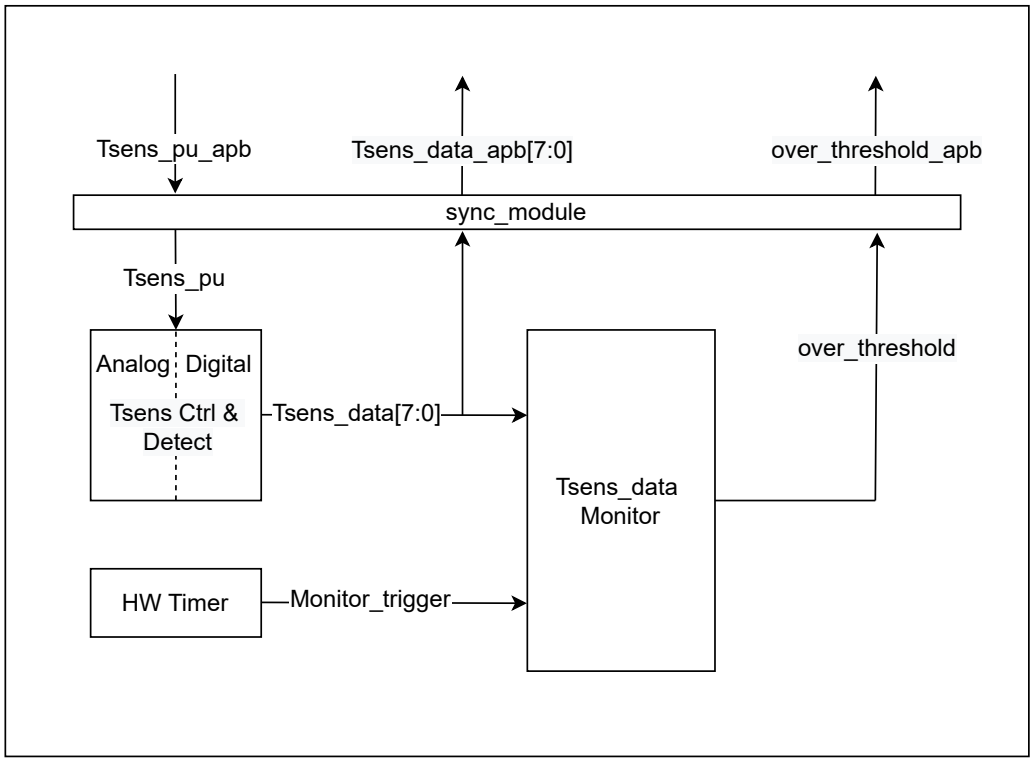


Figure 44.3-1. Temperature Sensor Architecture

As Figure 44.3-1 shows, the temperature sensor module contains the following major blocks:

- Tsens Ctrl & Detect: temperature sensor
- HW Timer: triggers automatic temperature monitoring
- Tsens_data Monitor: monitors whether the temperature is outside the threshold range
- sync_module: Synchronization block between APB clock domain and temperature sensor clock domain

44.4 Functional Description

44.4.1 Temperature Sensor Power Up

The temperature sensor can be powered up by setting the TSENS_POWER_UP register.

44.4.2 Temperature Sensor Clock

The temperature sensor has only one clock source: LP_PERI_CLK, and there is no frequency divider inside the module.

44.4.3 Wake-Up Modes for Automatic Temperature Monitoring

There are two wake-up modes for temperature monitoring, selected by TSENS_WAKEUP_MODE:

- Absolute value mode:

  - Monitors the absolute value of the current temperature. Configure TSENS_WAKEUP_TH_LOW and TSENS_WAKEUP_TH_HIGH to set the temperature thresholds. Wake-up will be triggered if the sampled value is above the high threshold or below the low threshold.

- Change value mode:
  - Monitors the temperature changes inside the chip. If the temperature increment of two consecutive samplings exceeds the high threshold configured in TSENS_WAKEUP_TH_HIGH, or the temperature decrement of two consecutive samplings exceeds the low threshold configured in TSENS_WAKEUP_TH_LOW, a wake-up will be triggered. For example, when TSENS_WAKEUP_TH_LOW is set to 8, and the two consecutive sampling values are 28 and 19, resulting in a temperature decrement of 9, a wake-up will be triggered.

44.4.4 Temperature Measurement Range and Offset

The temperature sensor is designed with five measurement ranges, as displayed in Table 44.4-1, to enhance accuracy. Each range has its own measurement errors, but users can select particular offsets to obtain calibrated data, so there is no need to be concerned about the errors.

Table 44.4-1. Temperature Measurement Range and Offset

Temperature Measurement Range (°C)	Temperature Offset
50 ~ 125	−2
20 ~ 100	−1
−10 ~ 80	0
−30 ~ 50	1
−40 ~ 20	2

44.4.5 Data Conversion

The sensor output value ($VALUE$) is stored in TSENS_OUT. To calculate the actual temperature T (°C) based on $VALUE$, use the following equation:

$$T = 0.4386 * VALUE - 27.88 * offset - 20.52$$

where $offset$ is the temperature offset displayed in Table 44.4-1.

44.5 Event Task Matrix Feature

The temperature sensor on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows the temperature sensor's ETM tasks to be triggered by any peripherals' ETM events, or temperature sensor's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to the temperature sensor. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The temperature sensor can receive the following ETM tasks:

- TMPSNSR_TASK_START: The temperature sensor starts sampling when this task is triggered.
- TMPSNSR_TASK_STOP: The temperature sensor stops sampling when this task is triggered.

The temperature sensor can generate the following ETM events:

- TMPSNSR_EVT_OVER_LIMIT: Generated when the temperature is beyond the threshold.

In practical applications, the temperature sensor's ETM events can trigger its own ETM tasks. For example, the TMPSNSR_EVT_OVER_LIMIT event can trigger the TMPSNSR_TASK_STOP task.

44.6 Interrupts

ESP32-P4's Temperature Sensor can generate the following interrupt signal that will be sent to the Interrupt Matrix.

- LP_TSENS_INTR

There is one internal interrupt source from the Temperature Sensor that can generate the above interrupt signal:

- TSENS_COCPU_TSENS_WAKE_INT: Triggered when the temperature sample value exceeds the threshold.

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 44.8 *Register Summary*.

44.7 Programming Procedure

The temperature sensor can be started by software as follows:

1. Set TSENS_POWER_UP to power up the temperature sensor.
2. Set LPPERI_CK_EN_LP_TSENS to enable the temperature sensor clock.
3. Wait for TSENS_XPD_WAIT clock cycles till the temperature sensor releases from reset and starts measuring the temperature.
4. Wait for 100 μs (so that the output value gradually approaches the actual temperature) and then read the data from TSENS_OUT.

To enable hardware-triggered automatic temperature monitoring, add the following programming steps before powering up the sensor:

1. Configure TSENS_SAMPLE_RATE to set sampling rate.
2. Configure TSENS_WAKEUP_MODE to select wake-up mode for temperature monitoring.
3. Configure TSENS_WAKEUP_TH_HIGH/LOW to set the high and low temperature monitoring thresholds.
4. Set TSENS_WAKEUP_EN to start temperature monitoring.
5. Set TSENS_SAMPLE_EN to enable automatic temperature monitoring.

In automatic temperature monitoring mode, the output value will not be saved. Nevertheless, users can get the value at any moment by accessing TSENS_OUT.

44.8 Register Summary

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

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

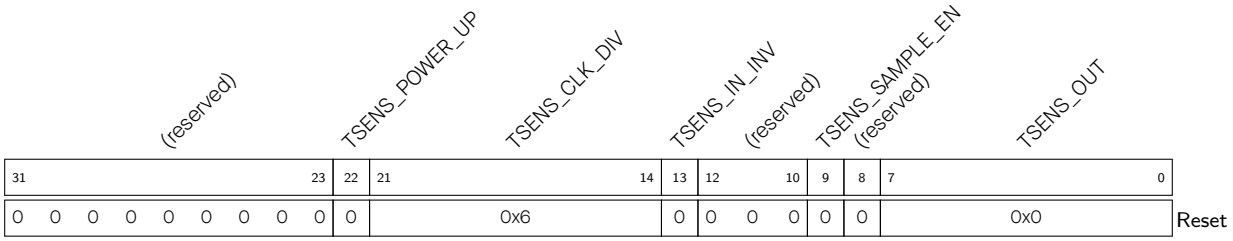
Name	Description	Address	Access
Control Registers			
TSENS_CTRL_REG	Temperature sensor control register 1.	0x0000	varies
TSENS_CTRL2_REG	Temperature sensor control register 2.	0x0004	R/W
Interrupt Registers			
TSENS_INT_RAW_REG	Raw register of temperature sensor interrupt.	0x0008	R/WTC/SS
TSENS_INT_ST_REG	State register of temperature sensor interrupt.	0x000C	RO
TSENS_INT_ENA_REG	Enable register of temperature sensor interrupt.	0x0010	R/WTC
TSENS_INT_CLR_REG	Clear register of temperature sensor interrupt.	0x0014	WT
TSENS_INT_ENA_W1TS_REG	TSENS_INT_ENA_REG configuration register	0x001C	WT
TSENS_INT_ENA_W1TC_REG	TSENS_INT_ENA_REG configuration register	0x0020	WT
Wake-up Control Registers			
TSENS_WAKEUP_CTRL_REG	Temperature sensor wake-up control registers.	0x0024	varies
TSENS_SAMPLE_RATE_REG	Hardware automatic sampling control registers.	0x0028	R/W

44.9 Registers

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

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

Register 44.1. TSENS_CTRL_REG (0x0000)



TSENS_OUT Stores the temperature sensor output value. (RO)

TSENS_SAMPLE_EN Configures whether to enable automatic temperature monitoring.

0: Disable

1: Enable

(R/W)

TSENS_IN_INV Configures whether to invert temperature sensor data.

0: No effect

1: Invert

(R/W)

TSENS_CLK_DIV Configures the clock division of the temperature sensor, which affects the frequency of temperature data generation by the analog circuits. (R/W)

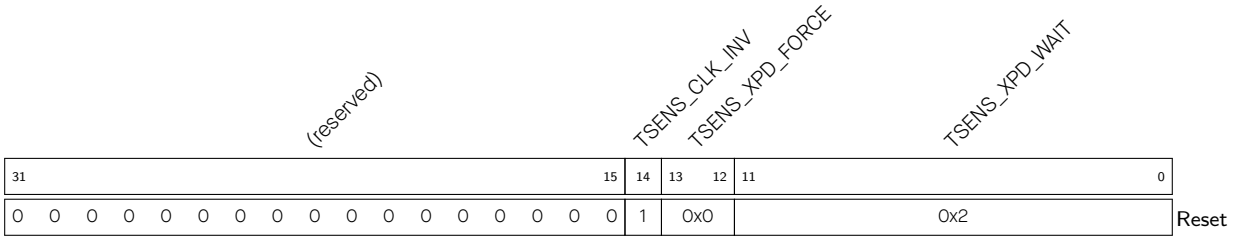
TSENS_POWER_UP Configures whether to power up temperature sensor.

0: No effect

1: Power up

(R/W)

Register 44.2. TSENS_CTRL2_REG (0x0004)



TSENS_XPD_WAIT Configures the wait time (in clock cycles) for the sensor to release from reset. (R/W)

TSENS_XPD_FORCE Configures whether to enable force power up/down the temperature sensor.

0: Disable force power up function

1: Disable force power down function

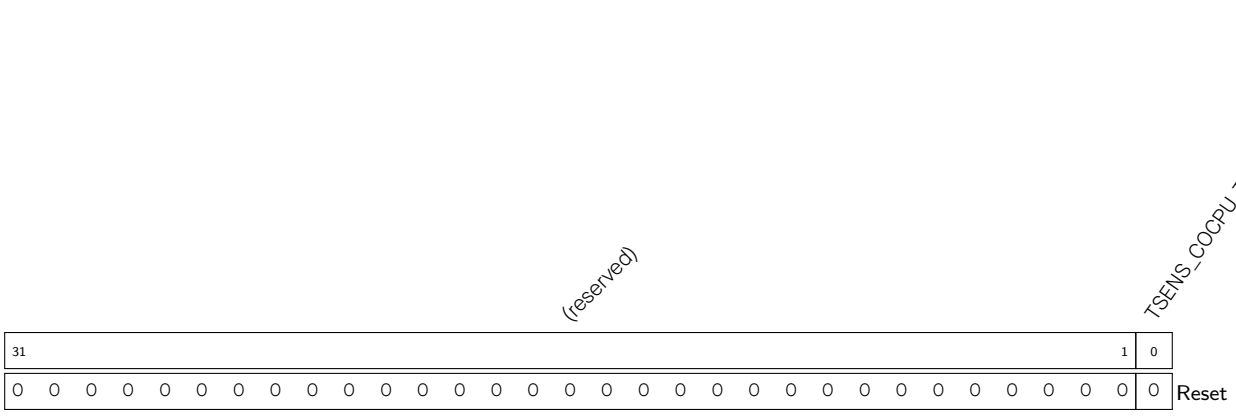
2: Enable force power up temperature sensor

3: Enable force power down temperature sensor

(R/W)

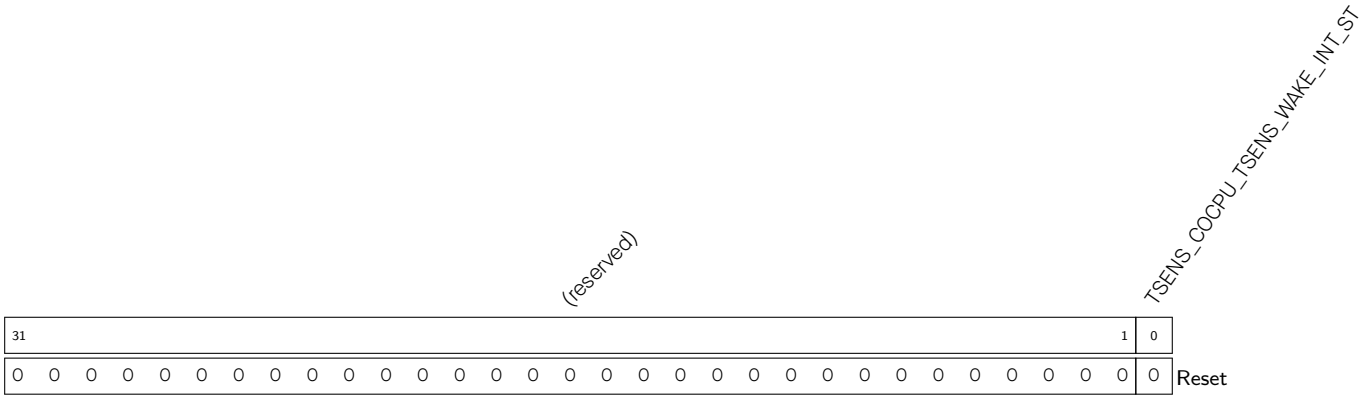
TSENS_CLK_INV Configures the phase of the temperature sensor clock. (R/W)

Register 44.3. TSENS_INT_RAW_REG (0x0008)



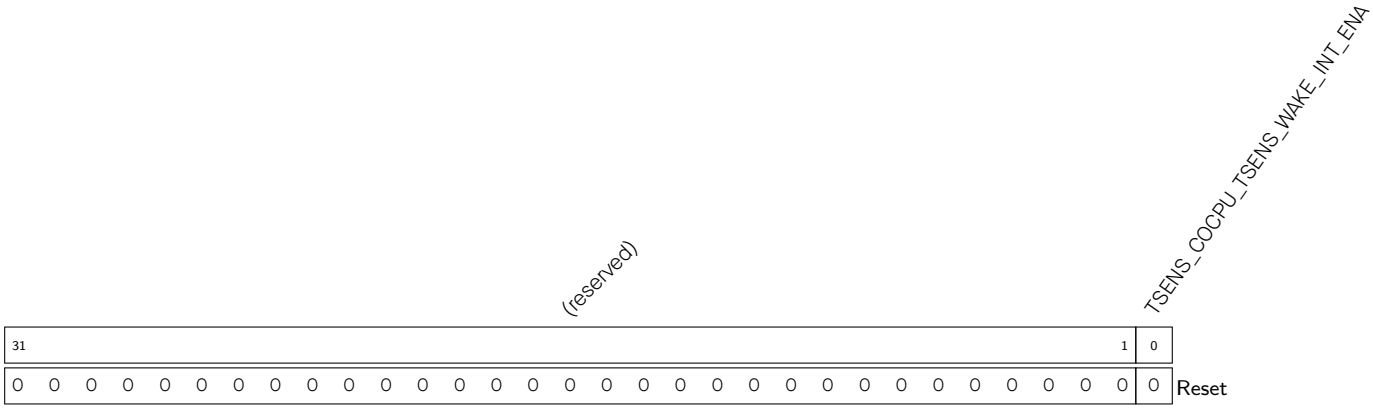
TSENS_COCPU_TSENS_WAKE_INT_RAW The raw interrupt status of the TSENS_COCPU_TSENS_WAKE_INT. (R/WTC/SS)

Register 44.4. TSENS_INT_ST_REG (0x000C)



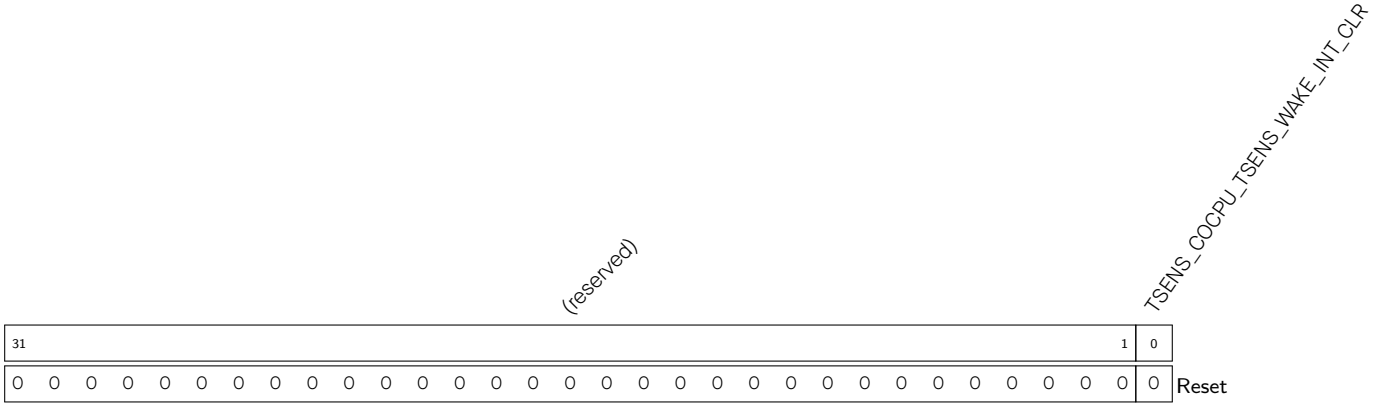
TSENS_COCPU_TSENS_WAKE_INT_ST The masked interrupt status of the TSENS_COCPU_TSENS_WAKE_INT. (RO)

Register 44.5. TSENS_INT_ENA_REG (0x0010)



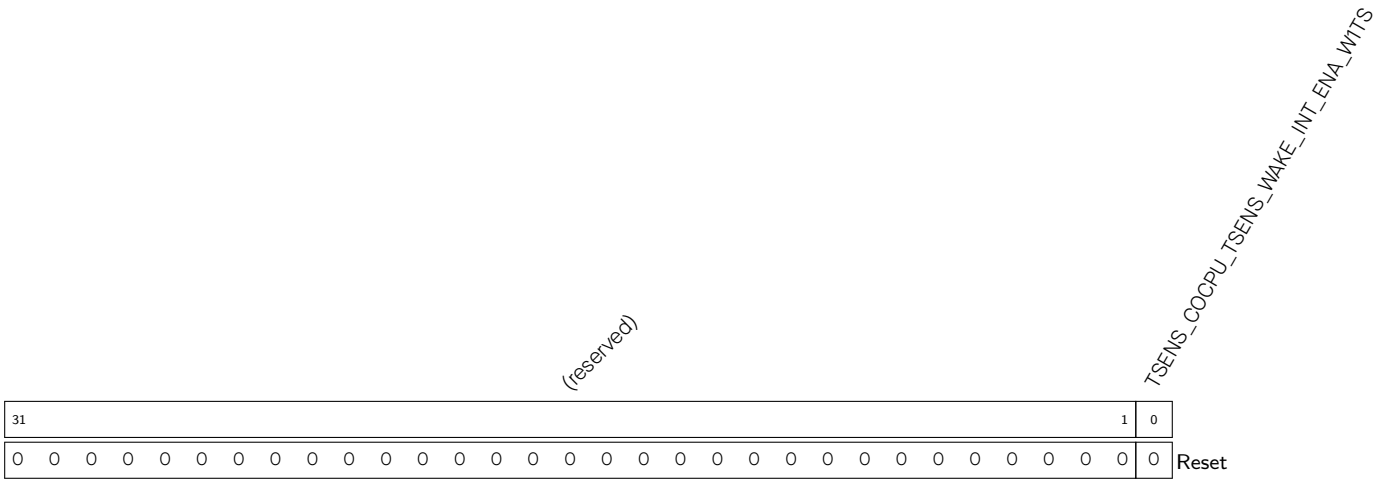
TSENS_COCPU_TSENS_WAKE_INT_ENA Write 1 to enable TSENS_COCPU_TSENS_WAKE_INT. (R/WTC)

Register 44.6. TSENS_INT_CLR_REG (0x0014)



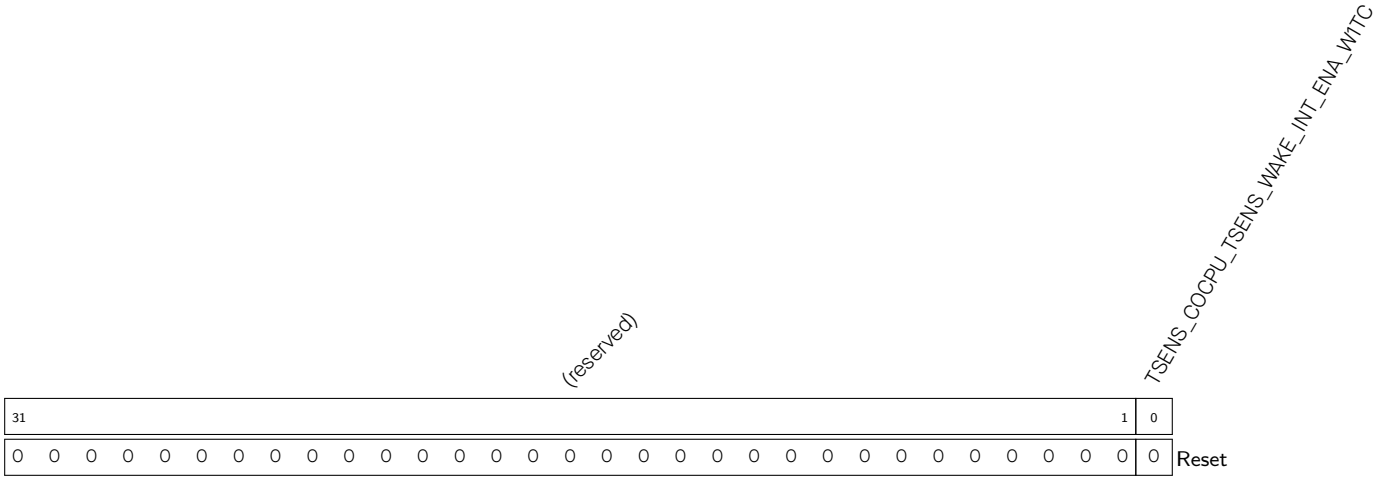
TSENS_COCPU_TSENS_WAKE_INT_CLR Write 1 to clear TSENS_COCPU_TSENS_WAKE_INT. (WT)

Register 44.7. TSENS_INT_ENA_W1TS_REG (0x001C)



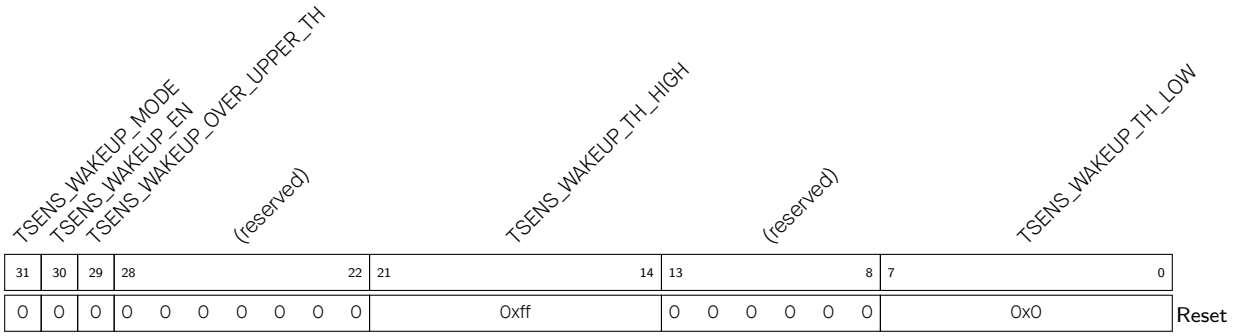
TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TS Write 1 to set TSENS_INT_ENA_REG. (WT)

Register 44.8. TSENS_INT_ENA_W1TC_REG (0x0020)



TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TC Write 1 to clear TSENS_INT_ENA_REG. (WT)

Register 44.9. TSENS_WAKEUP_CTRL_REG (0x0024)



TSENS_WAKEUP_TH_LOW Configures the low threshold for temperature monitoring wake-up function. (R/W)

TSENS_WAKEUP_TH_HIGH Configures the high threshold for temperature monitoring wake-up function. (R/W)

TSENS_WAKEUP_OVER_UPPER_TH Represents whether the temperature output value exceeds the threshold.

0: The temperature output value is below the low threshold.

1: The temperature output value is above the high threshold.

(RO)

TSENS_WAKEUP_EN Configures whether to enable temperature monitoring wake-up function.

0: Disable

1: Enable

(R/W)

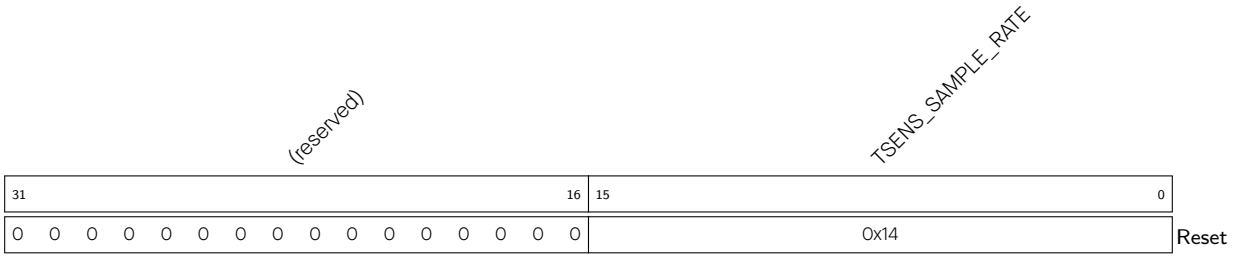
TSENS_WAKEUP_MODE Selects the wake-up mode for temperature monitoring. Valid only when TSENS_WAKEUP_EN = 1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 44.10. TSENS_SAMPLE_RATE_REG (0x0028)



TSENS_SAMPLE_RATE Configures the sampling rate for hardware-triggered temperature monitoring. The sampling period = configured value × sensor's working clock cycle. (R/W)

Chapter 45

ADC Controller (ADC)

45.1 Overview

ESP32-P4 integrates two 12-bit successive approximation ADCs (SAR ADCs) for measuring analog signals from up to 14 pins. Figure 45.1-1 shows the basic architecture of the SAR ADC module on ESP32-P4.

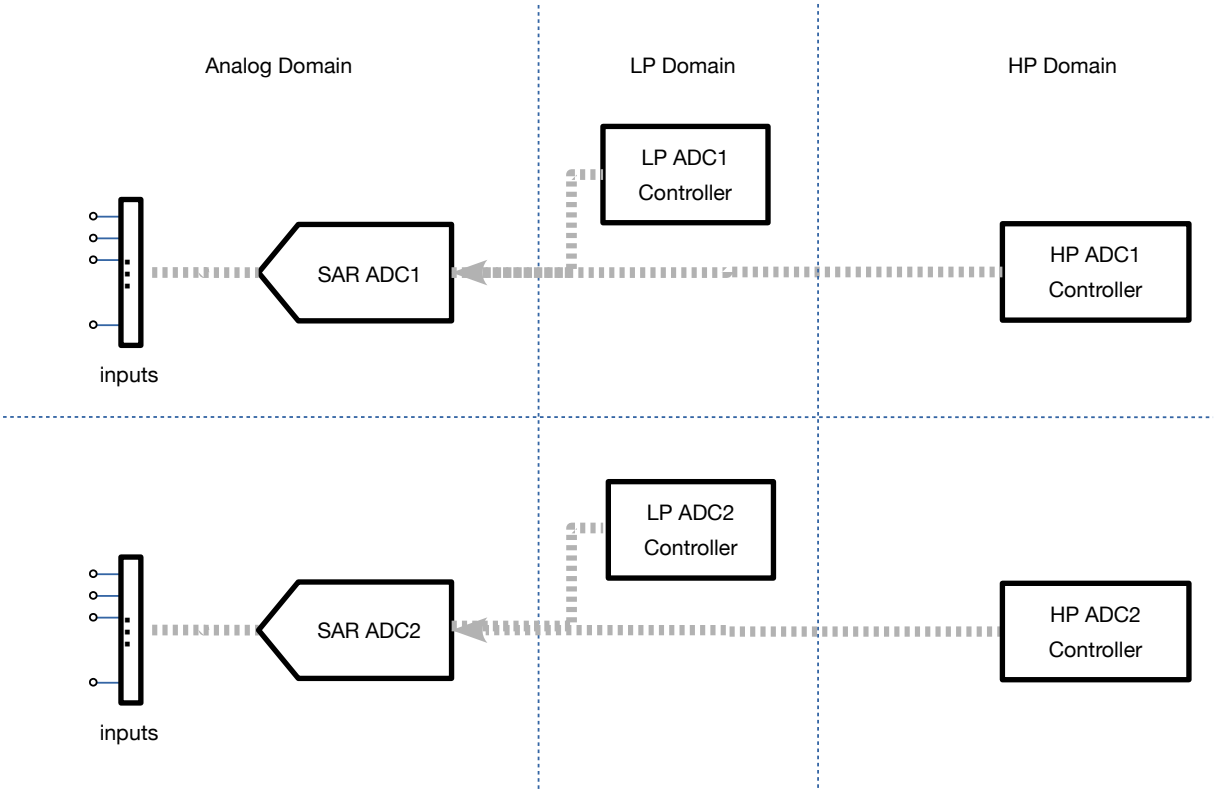


Figure 45.1-1. SAR ADC Simplified Architecture

As the figure shows, the SAR ADCs are managed by four dedicated controllers:

- Two HP (high-performance) controllers: HP ADC1 Controller and HP ADC2 Controller, designed for multi-channel sampling mode, supporting continuous transfer of conversion results to memory via the GDMA interface.
- Two LP (low-power) controllers: LP ADC1 Controller and LP ADC2 Controller, supporting operation in sleep mode and one-shot sampling mode.

45.2 Terminology

The following terms related to SAR ADC are defined in the context of the *ESP32-P4 Technical Reference Manual* to help readers better understand this document:

SAR ADC: Including analog ADC circuit and digital ADC controller. The HP ADC and LP ADC in the following text share the analog ADC circuit, but have their own controllers, namely the HP ADC controllers and LP ADC controllers. The SAR ADC mentioned in the text refers to both HP ADC and LP ADC.

HP ADC: High-performance ADC, supporting multi-channel sampling, GDMA data transfer, etc., not available in sleep mode.

LP ADC: Low-power ADC, supporting one-shot sampling mode, can be used in sleep mode.

One-shot sampling mode: In this mode, LP ADC samples one channel at a time.

Multi-channel sampling mode: In this mode, HP ADC sequentially samples a group of channels or continuously samples a single channel.

Dual HP ADC sampling: It means two HP ADCs sample either simultaneously or alternatively.

Conversion result: The conversion result is the binary digital value obtained after the analog-to-digital conversion process. It is sometimes written as conversion data.

Filtered data: Filtered data is the result of processing conversion data through a filter.

Sampling: The term typically refers to the entire process of sampling analog inputs, converting the sampled data, and transferring the conversion results to memory. In certain contexts and stages of the process, sampling may also refer to the SAR ADC capturing data points from an analog signal.

45.3 Features

The SAR ADCs have the following features:

- Support HP ADC*x* controllers and LP ADC*x* controllers to get control of SAR ADC*x* via software, *x* = 1, 2
- 12-bit resolution
- Analog inputs sampling from up to 14 pins
- HP ADC*x* controllers:
  - Provide multi-channel sampling control module, supporting multi-channel sampling mode, with configurable channel sampling sequence
  - Provide mode control module, supporting dual HP ADC sampling
  - Provide two filters with configurable filter coefficients

  - Provide two threshold monitors that can trigger an interrupt when the filtered data is above a high threshold or below a low threshold
  - Support continuous transfer of conversion results to memory via the GDMA interface
- LP ADC*x* controllers:
  - Support one-shot sampling mode
  - Support sampling in sleep mode (e.g., Deep-sleep)
- Support for several Event Task Matrix (ETM) related events and tasks

45.4 Architecture

The major components of SAR ADCs and their interconnections are shown in Figure 45.4-1.

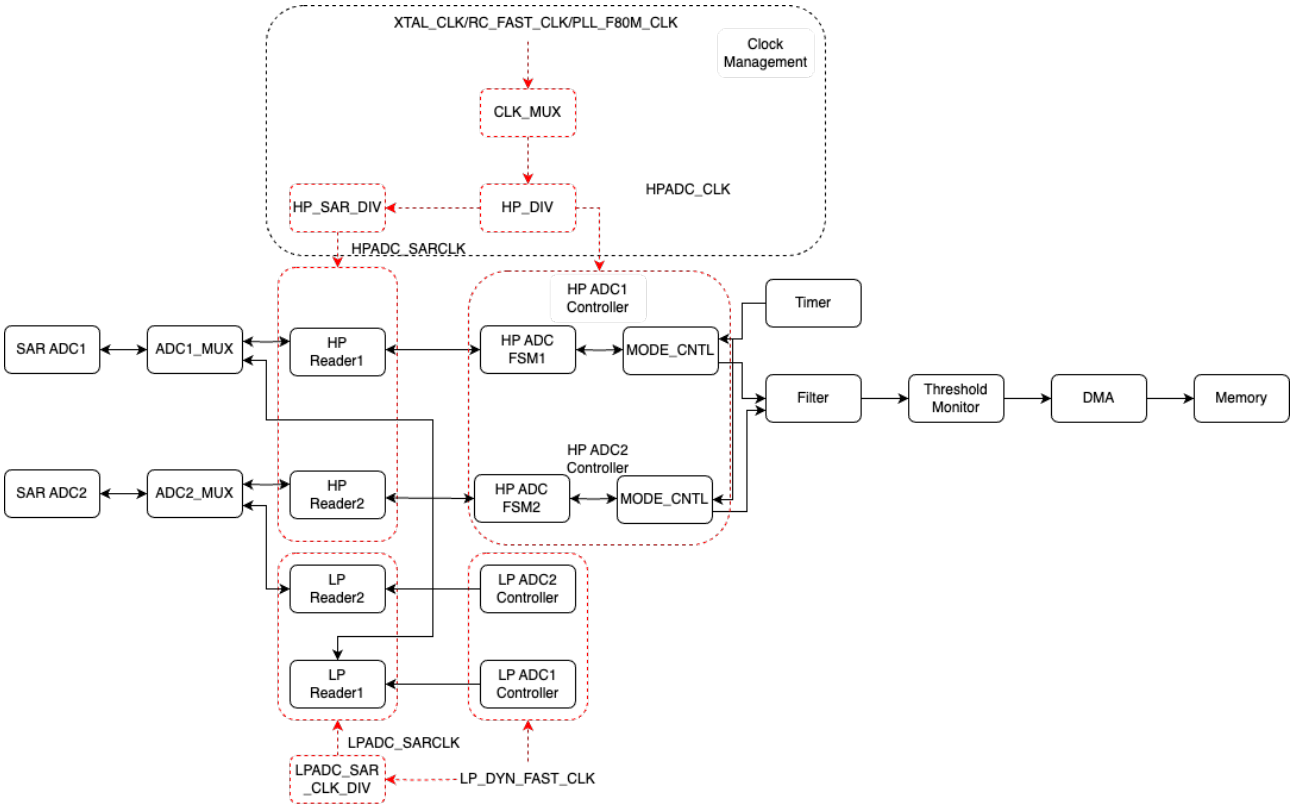


Figure 45.4-1. SAR ADC Architecture

- —>: clock signals
- : clock divider, clock mux, and the blocks the clock works for

As Figure 45.4-1 shows, the SAR ADC module contains the following major functional blocks:

- SAR ADC1: Samples analog inputs from up to 8 pins
- SAR ADC2: Samples analog inputs from up to 6 pins
- Clock Management: Selects clock source and division
- Timer: Dedicated timer for the HP ADC*x* controllers to generate sampling enable signal.
- HP ADC FSM*x*: FSM1 and FSM2 that can:

  - receive the sampling enable signal from the timer.
  - determine the conversion rules defined in the patterns for HP ADC*x*.
  - drive the HP Reader*x* module to read conversion data.
  - transfer conversion data to the filter.
- Filter: Filter0 and filter1, used for filtering conversion results in multi-channel sampling mode.
- Threshold Monitor: Threshold monitor 0 and threshold monitor 1 that trigger an interrupt when the filtered data is above a high threshold or below a low threshold.
- MODE_CNTL: Used for filtering the sampling signal triggered by the timer, supporting dual HP ADC sampling mode.
- HP Reader*x*: HP reader 1 and HP reader 2, driven by HP ADC FSM*x* to read data from HP ADC*x*.
- LP Reader*x*: LP reader 1 and LP reader 2, driven by LP ADC*x* Controller to read data from LP ADC*x*.
- LP ADC*x* Controller: LP ADC1 Controller and LP ADC2 Controller. The controllers provide sampling enable signal, drive the LP Reader*x* module to read the conversion results of LP ADC*x*, and store the conversion results.

45.5 Functional Description

45.5.1 SAR ADC Power Up

The SAR ADCs can be powered up by setting PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RST. The ADC sampling can be conducted immediately after power-up. Users need not be concerned about wait time, as it has been accounted for in the hardware design.

45.5.2 SAR ADC Channels

The SAR ADCs have 14 channels that are connected to 14 pins on the chip. To sample an analog signal, the SAR ADCs must first select the analog pin to measure via an internal multiplexer.

Table 45.5-1 shows the pins used as SAR ADC channels.

Table 45.5-1. SAR ADC Channels and Chip Pins

Chip Pins	SAR ADC Channel	SAR ADC Selection
GPIO16	0	SAR ADC1
GPIO17	1	
GPIO18	2	
GPIO19	3	
GPIO20	4	
GPIO21	5	
GPIO22	6	
GPIO23	7	
GPIO49	2	SAR ADC2
GPIO50	3	

Chip Pins	SAR ADC Channel	SAR ADC Selection
GPIO51	4	
GPIO52	5	
GPIO53	6	
GPIO54	7	

45.5.3 SAR ADC Clock

The clock for the HP ADC*x* Controller has three possible sources, selected by HP_SYS_CLKRST_ADC_CLK_SRC_SEL:

- XTAL_CLK
- RC_FAST_CLK
- PLL_F80M_CLK

The following clocks can be divided from the HP ADC*x* Controller's clock:

- HPADC_SARCLK: It is the operating clock of HP ADC*x* and HP Reader*x*.
- HPADC_CLK: It is the operating clock of HP ADC FSM*x*.

The frequency of HPADC_SARCLK can affect the sampling accuracy. If the HPADC_SARCLK frequency is higher than 5 MHz, the sampling accuracy will decrease. HPADC_SARCLK is divided from HPADC_CLK via a dedicated divider. The division is set in ADC_SAR_CLK_DIV.

It takes 25 HPADC_SARCLK clock cycles to complete one sampling, so the maximum sampling rate is limited by the frequency of HPADC_SARCLK.

The LP ADC*x* Controller is clocked from LP_DYN_FAST_CLK.

The following clocks can be divided from the LP ADC*x* Controller's clock:

- LPADC_SARCLK: It is the operating clock of LP ADC*x* and LP Reader*x*. The division set in LPADC_SAR1_CLK_DIV and LPADC_SAR2_CLK_DIV should be at least 2. The frequency of LPADC_SARCLK should not exceed 5 MHz.
- LPADC_CLK: It is the operating clock of LP ADC FSM*x*.

For more information about clocks, please refer to Chapter 8 *Reset and Clock*.

45.5.4 SAR ADC Conversion and Attenuation

The SAR ADCs can measure analog voltages from 0 mV to V_{ref}. V_{ref} is the SAR ADCs' internal reference voltage. The conversion result ($data$) is a 12-bit digital value, which is the raw data. To calculate the voltage V_{data} based on the raw data, the following formula can be used:

$$V_{data} = \frac{V_{ref}}{4095} \times data$$

To convert voltages larger than V_{ref}, apply attenuation to the input signals. The attenuation can be configured to 0 dB, 2.5 dB, 6 dB, or 12 dB.

45.5.5 HP ADC FSM

In multi-channel sampling mode HP ADC FSM (hereinafter referred to as FSM) generates all types of signals used in the sampling process. Figure 45.5-1 illustrates how the FSM works.

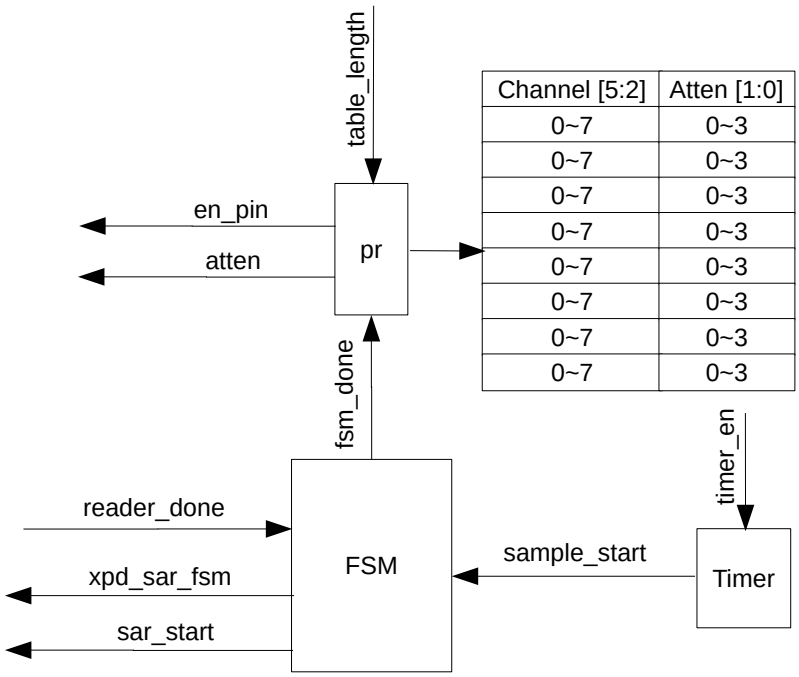


Figure 45.5-1. HP ADC FSM Block Diagram

Wherein:

- Timer: a dedicated timer for the HP ADC controller to generate `sample_start` signal.
- pr: a pointer to the pattern table that defines the conversion rules for HP ADC. FSM sends out corresponding signals based on the conversion rules.

Set ADC_TIMER_EN to enable the timer. The timeout event of this timer triggers a `sample_start` signal. This signal drives the FSM module to start sampling. When the FSM module receives the `sample_start` signal, it starts the following operations:

- Powers up HP ADC.
- Determines the conversion rules (selected sample channels and attenuations) defined in the patterns that the current pr points to.
- Outputs the `en_pin` and `atten` signals corresponding to the conversion rules to the analog side.
- Initiates the `sar_start` signal and starts sampling.

When FSM receives the `reader_done` signal from HP Reader, it starts the following operations:

- Stops sampling.
- Transfers the conversion result to the filter. Then the threshold monitor transfers the filtered result to memory via GDMA (see Figure 45.4-1).

- Updates pr and waits for the next sampling. The pointer pr counts cyclically between 0 and ADC_SAR1/2_PATT_LEN (table_length).

45.5.6 HP ADC Pattern Table

HP ADC FSM1/2 each contain a pattern table configured by ADC_SAR*x*_PATT_TAB*m*_REG, with *m* indicating registers 1 ~ 4. Each register contains four patterns and each pattern is 6 bits wide, as below shows.

(reserved)								cmd0		cmd1		cmd2		cmd3	
31							24	23	18	17	12	11	6	5	0
0	0	0	0	0	0	0	0	0x0000		0x0000		0x0000		0x0000	

cmd *n* (*n* = 0 - 3) represents patterns 0 – 3.

Figure 45.5-2. ADC_SAR*x*_PATT_TAB1_REG Contains Patterns 0 - 3

(reserved)								cmd4		cmd5		cmd6		cmd7	
31							24	23	18	17	12	11	6	5	0
0	0	0	0	0	0	0	0	0x0000		0x0000		0x0000		0x0000	

cmd *n* (*n* = 4 - 7) represents patterns 4 ~ 7.

Figure 45.5-3. ADC_SAR*x*_PATT_TAB2_REG Contains Patterns 4 - 7

(reserved)								cmd8		cmd9		cmd10		cmd11	
31							24	23	18	17	12	11	6	5	0
0	0	0	0	0	0	0	0	0x0000		0x0000		0x0000		0x0000	

cmd *n* (*n* = 8 - 11) represents patterns 8 ~ 11.

Figure 45.5-4. ADC_SAR*x*_PATT_TAB3_REG Contains Patterns 8 - 11

(reserved)								cmd12		cmd13		cmd14		cmd15	
31							24	23	18	17	12	11	6	5	0
0	0	0	0	0	0	0	0	0x0000		0x0000		0x0000		0x0000	

cmd *n* (*n* = 12 - 15) represents patterns 12 ~ 15.

Figure 45.5-5. ADC_SAR*x*_PATT_TAB4_REG Contains Patterns 12 - 15

Each pattern is 6 bits wide, consisting of two fields where conversion rules are stored. Figure 45.5-6 shows the pattern format and is followed by the descriptions of each field.

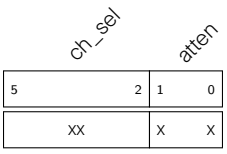


Figure 45.5-6. Pattern Structure

atten Configures attenuation:

0: 0 dB

1: 2.5 dB

2: 6 dB

3: 12 dB

ch_sel Configures channel. For HP ADC1, values 0-7 correspond to channels 0-7 individually, for HP ADC2, values 2-7 correspond to channels 2-7 individually.

45.5.7 HP ADC Pattern Configuration Example for Multi-channel Sampling

Assuming you want to implement the following sequence of HP ADC1 multi-channel sampling:

- Sample channel 2 of HP ADC1 with attenuation of 12 dB;
- Sample channel 0 of HP ADC1 with attenuation of 2.5 dB.

Then, the detailed configuration is as follows:

- Configure the first pattern (cmd0) of ADC_SAR1_PATT_TAB1_REG[5:0]:

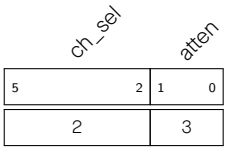


Figure 45.5-7. cmd0 configuration

atten write 3 to this field, to set the attenuation to 12 dB.

ch_sel write 2 to this field, to select channel 2.

- Configure the second pattern (cmd1) of ADC_SAR1_PATT_TAB1_REG[11:6]:

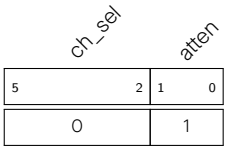


Figure 45.5-8. cmd1 Configuration

atten write 1 to this field, to set the attenuation to 2.5 dB.

ch_sel write 0 to this field, to select channel 0.

- Configure ADC_SAR1_PATT_LEN to 1, i.e., set pattern table size to (this value + 1) = 2. Then patterns cmd0 and cmd1 will be used.
- Enable the timer so that the HP ADC1 Controller starts sampling the two channels periodically.

45.5.8 Dual HP ADC Sampling Control

ADC_WORK_MODE controls how two HP ADCs work together:

- 0: Single HP ADC sampling, i.e., only one HP ADC samples. When a sampling request is triggered by the timer, ADC_SAR_SEL controls which FSM receives the sampling request:
  - 0: HP ADC FSM1, i.e., using HP ADC1 for sampling.
  - 1: HP ADC FSM2, i.e., using HP ADC2 for sampling.
- 1: Dual HP ADC simultaneous sampling. When a sampling request is triggered by the timer, both HP ADC FSM1 and HP ADC FSM2 receive the sampling request.
- 2: Dual HP ADC alternative sampling. When an odd number of sampling requests are triggered by the timer, HP ADC FSM1 receives the sampling request. When an even number of sampling requests are triggered by the timer, HP ADC FSM2 receives the sampling request.

45.5.9 HP ADC Filters

The HP ADC*x* controllers provide two filters for filtering conversion results in multi-channel sampling mode. Both filters can be configured to any HP ADC channel, but cannot be configured to the same channel. If the two filters are configured to the same channel, the first one takes effect.

The filtered data is determined by the following equation:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} - 0.5$$

- $data_{cur}$: the filtered data
- $data_{in}$: the conversion result
- $data_{prev}$: the last filtered data
- k: the filter coefficient

The filters are configured as follows:

- Configure ADC_FILTER_CHANNEL*x* to select the HP ADC channel for filter *x*, *x*=0, 1.
- Configure ADC_FILTER_FACTOR*x* to set the coefficient k for filter *x*, *x*=0, 1.

45.5.10 HP ADC Threshold Monitors

The HP ADC*x* controllers provide two threshold monitors to monitor the filtered data in multi-channel sampling mode. When the data is above the high threshold, a high threshold interrupt is triggered; when the data is below the low threshold, a low threshold interrupt is triggered. Both monitors can be configured to any HP ADC channel, but cannot be configured to the same channel.

Threshold monitors are configured as follows:

- Set ADC_THRES*x*_EN to enable threshold monitor *x* (*x*=0, 1).
- Configure ADC_THRES*x*_LOW to set a low threshold.
- Configure ADC_THRES*x*_HIGH to set a high threshold.

- Configure ADC_THRESx_CHANNEL to select the channel to monitor.

45.5.11 HP ADC GDMA Support

When HP ADC converts data on multiple channels, it is necessary to enable GDMA so that the data can be transferred to memory continuously.

GDMA support is triggered by the timer in the HP ADCx controllers. Users can switch the GDMA data path to the HP ADC controller by configuring ADC_APB_ADC_TRANS. For specific GDMA configuration, please refer to Chapter 3 *GDMA Controller (GDMA-AHB, GDMA-AXI)*.

The HP ADC eventually passes 32-bit data to GDMA. The data format is shown in Figure 45.5-9.

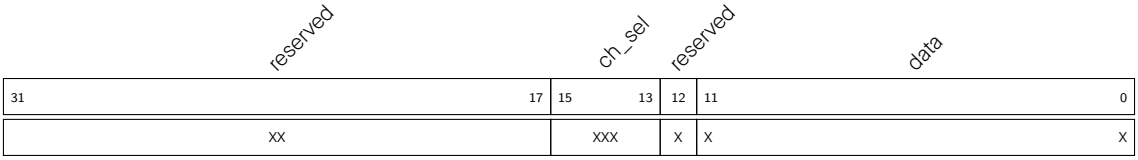


Figure 45.5-9. GDMA Data Format

data 12-bit HP ADC conversion result

ch_sel 3-bit channel information

45.5.12 LP ADC Wake-Up Modes for Automatic Monitoring

The LP ADCx controllers support automatic monitoring of the conversion data. A dedicated timer generates LP ADC sampling signals periodically based on the value of LPADC_ADCx_HW_READ_RATE_I. Then the conversion data is fed into a threshold monitor (not the HP ADC threshold monitors). If the data exceeds the threshold, an interrupt will be generated.

There are two wake-up modes for automatic monitoring, selected by LPADC_SARx_WAKEUP_MODE:

- 0: Absolute value mode
  Monitors the absolute value of the conversion data. Configure LPADC_SARx_WAKEUP_TH_LOW and LPADC_SARx_WAKEUP_TH_HIGH to set the thresholds. Wake-up will be triggered if the data is above the high threshold or below the low threshold.
- 1: Change value mode
  Monitors the changes of the conversion data. If the increment of two consecutive samplings exceeds the high threshold configured in LPADC_SARx_WAKEUP_TH_HIGH, or the decrement of two consecutive samplings exceeds the low threshold configured in LPADC_SARx_WAKEUP_TH_LOW, a wake-up will be triggered. For example, when LPADC_SARx_WAKEUP_TH_LOW is set to 8, and the two consecutive conversion results are 28 and 19, resulting in a decrement of 9, a wake-up will be triggered.

45.6 Event Task Matrix Feature

The SAR ADC on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows SAR ADC's ETM tasks to be triggered by any peripherals' ETM events, or SAR ADC's ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM tasks and events related to SAR ADC. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The SAR ADC can receive the following ETM tasks:

- ADC_TASK_SAMPLE: LP ADC starts one-shot sampling when this task is triggered.
- ADC_TASK_START: HP ADC starts multi-channel sampling when this task is triggered.
- ADC_TASK_STOP: SAR ADC stops sampling when this task is triggered.

The SAR ADC can generate the following ETM events:

- ADC_EVT_CONV_CMPLT: Generated each time SAR ADC completes a sampling in either one-shot sampling mode or multi-channel sampling mode.
- ADC_EVT_EQ_ABOVE_THRESH*x*: Generated when the HP ADC's filtered data is above the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_EQ_BELOW_THRESH*x*: Generated when the HP ADC's filtered data is below the threshold. *x* = 0, 1, representing threshold monitor 0, 1.
- ADC_EVT_STARTED: Generated when SAR ADC begins sampling; one-shot sampling will not trigger this event.
- ADC_EVT_STOPPED: Generated when SAR ADC stops sampling, one-shot sampling will not trigger this event.

In practical applications, SAR ADC's ETM events can trigger its own ETM tasks. For example, the ADC_EVT_EQ_ABOVE_THRESH*x* event can trigger the ADC_TASK_STOP task.

45.7 Interrupts

ESP32-P4's SAR ADC can generate the following interrupt signals that will be sent to the Interrupt Matrix.

- ADC_INTR
- LP_ADC_INTR

The following internal interrupt sources from SAR ADC can generate the interrupt signal ADC_INTR:

- ADC_SAR*x*_DONE_INT: Triggered when HP ADC*x* completes one sampling.
- ADC_THRES*x*_HIGH_INT: Triggered when the filtered data is above the high threshold.
- ADC_THRES*x*_LOW_INT: Triggered when the filtered data is below the low threshold.

The following internal interrupt sources from SAR ADC can generate the interrupt signal LP_ADC_INTR:

- LPADC_COCPU_SARADC*x*_DONE_INT: Triggered when LP ADC*x* completes one sampling.
- LPADC_COCPU_SARADC*x*_ERROR_INT: Triggered when sampling error occurs.
- LPADC_COCPU_SARADC*x*_WAKE_INT: Triggered when LP ADC*x* generates a wake-up event (such as conversion data exceeding the threshold).

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 45.9 *Register Summary*.

45.8 Programming Procedure

45.8.1 HP ADC Multi-Channel Sampling Mode

The multi-channel sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RST to power up HP ADC.
2. Configure HP_SYS_CLKRST_ADC_CLK_SRC_SEL to select HP ADC clock source.
3. Configure HP_SYS_CLKRST_ADC_CLK_DIV_NUM and HP_SYS_CLKRST_ADC_SAR*x*_CLK_DIV_NUM to set clock division.
4. Set HP_SYS_CLKRST_ADC_CLK_EN to enable HP ADC clock.
5. Set LPADC_SAR*x*_DIG_FORCE to enable the sampling channel.
6. Configure the pattern table as described in Section 45.5.6 *HP ADC Pattern Table*.
7. Configure channels and filtering coefficients as described in Section 45.5.9 *HP ADC Filters*.
8. Configure threshold monitoring as described in Section 45.5.10 *HP ADC Threshold Monitors*, as needed.
9. Set ADC_APB_ADC_TRANS to use GDMA.
10. Configure ADC_TIMER_TARGET to set trigger target for HP ADC timer.
11. Set ADC_TIMER_EN to enable the timer.

The timer timeout will trigger HP ADC FSM to start sampling according to the pattern table. The conversion result will be automatically stored in memory. When the sampling reaches the specified limit in ADC_APB_ADC_EOF_NUM, it will terminate.

Once sampling is complete, an ADC_SAR*x*_DONE_INT_RAW interrupt will be generated.

45.8.2 LP ADC One-shot Sampling Mode

The one-shot sampling mode can be configured with the following procedure:

1. Set PMU_XPD_PERIF_I2C and PMU_PERIF_I2C_RSTB to power up LP ADC.
2. Configure LPPERI_LPADC_FUNC_DIV_NUM and LPPERI_LPADC_SAR*x*_DIV_NUM to set the frequency of LPADC_CLK and LPADC*x*_SARCLK.
3. Configure LPADC_SAR*x*_EN_PAD and LPADC_SAR*x*_ATTEN to select sampling channels and attenuation.
4. Set LPADC_SAR_MEAS*x*_START_SAR to enable the one-shot sampling mode of LP ADC*x*.

Once sampling is complete, an LPADC_COCPU_SARADC*x*_INT_RAW interrupt is generated. Software can read the conversion result from LPADC_SAR_MEAS*n*_DATA_SAR.

45.8.3 LP ADC Automatic Monitoring

The configuration process for the automatic monitoring of the LP ADC controller is as follows:

1. Configure LPADC_ADC*x*_HW_READ_RATE_I to set the sampling rate.
2. Configure LPADC_SAR*x*_WAKEUP_MODE to select the wake-up mode.

3. Configure LPADC_SAR*x*_WAKEUP_TH_HIGH and LPADC_SAR*x*_WAKEUP_TH_LOW to set the high and low thresholds.
4. Set LPADC_SAR*x*_WAKEUP_EN to enable automatic monitoring.
5. Set LPADC_ADC*x*_HW_READ_EN_I to enable periodic monitoring.

During the automatic monitoring process, the conversion data will not be stored, but software can always read the last conversion data from LPADC_SAR_MEAS*n*_DATA_SAR.

45.9 Register Summary

45.9.1 HP ADC Register Summary

The addresses in the table below are relative to the base address of HP ADC (written as ADC Controller) in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
HP ADC Configuration Registers			
ADC_CTRL_REG	Configuration register for HP ADC Controller	0x0000	R/W
ADC_CTRL2_REG	Configuration register for HP ADC Controller.	0x0004	R/W
ADC_FILTER_CTRL1_REG	Configuration register 1 for HP ADC filter.	0x0008	R/W
ADC_SAR1_PATT_TAB1_REG	Pattern table register 1 for HP ADC1.	0x0018	R/W
ADC_SAR1_PATT_TAB2_REG	Pattern table register 2 for HP ADC1.	0x001C	R/W
ADC_SAR1_PATT_TAB3_REG	Pattern table register 3 for HP ADC1.	0x0020	R/W
ADC_SAR1_PATT_TAB4_REG	Pattern table register 4 for HP ADC1.	0x0024	R/W
ADC_SAR2_PATT_TAB1_REG	Pattern table register 1 for HP ADC2.	0x0028	R/W
ADC_SAR2_PATT_TAB2_REG	Pattern table register 2 for HP ADC2.	0x002C	R/W
ADC_SAR2_PATT_TAB3_REG	Pattern table register 3 for HP ADC2.	0x0030	R/W
ADC_SAR2_PATT_TAB4_REG	Pattern table register 4 for HP ADC2.	0x0034	R/W
ADC_FILTER_CTRL0_REG	Configuration register 0 for HP ADC filter.	0x003C	R/W
ADC_THRES0_CTRL_REG	Filtered data threshold control register 0.	0x0044	R/W
ADC_THRES1_CTRL_REG	Filtered data threshold control register 1.	0x0048	R/W
ADC_THRES_CTRL_REG	Filter configuration register.	0x004C	R/W
ADC_INT_ENA_REG	Enable register of HP ADC interrupts.	0x0050	R/W
ADC_INT_RAW_REG	Raw register of HP ADC interrupts.	0x0054	R/WTC/SS
ADC_INT_ST_REG	State register of HP ADC interrupts.	0x0058	RO
ADC_INT_CLR_REG	Clear register of HP ADC interrupts.	0x005C	WT
ADC_DMA_CONF_REG	GDMA configuration register for HP ADC.	0x0060	R/W
HP ADC Version Control Registers			
ADC_CTRL_DATE_REG	HP ADC version control register	0x03FC	RO

45.9.2 LP ADC Register Summary

The addresses in the table below are relative to LP ADC's base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Name	Description	Address	Access
LP ADC Configuration Registers			
LPADC_READER1_CTRL_REG	LP ADC1 Reader control register.	0x0000	R/W

Name	Description	Address	Access
LPADC_MEAS1_CTRL2_REG	LP ADC1 configuration registers.	0x000C	varies
LPADC_MEAS1_MUX_REG	LP or HP ADC controller selection register for SAR ADC1.	0x0010	R/W
LPADC_ATTEN1_REG	LP ADC1 attenuation registers.	0x0014	R/W
LPADC_READER2_CTRL_REG	LP ADC2 Reader control register.	0x0024	R/W
LPADC_MEAS2_CTRL2_REG	SAR ADC2 configuration registers.	0x0030	varies
LPADC_MEAS2_MUX_REG	LP or HP controller selection register for SAR ADC2.	0x0034	R/W
LPADC_ATTEN2_REG	LP ADC2 attenuation registers.	0x0038	R/W
LP ADC Power Control Registers			
LPADC_FORCE_WPD_SAR_REG	LP ADC Controllers power control registers.	0x003C	R/W
LP ADC Interrupt Registers			
LPADC_COCPU_INT_RAW_REG	Raw register of LP ADC interrupts.	0x0048	R/WTC/SS
LPADC_INT_ENA_REG	Enable register of LP ADC interrupts.	0x004C	R/WTC
LPADC_INT_ST_REG	Status register of LP ADC interrupts.	0x0050	RO
LPADC_INT_CLR_REG	Clear register of LP ADC interrupts.	0x0054	WT
LPADC_INT_ENA_W1TS_REG	LPADC_INT_ENA_REG configuration register.	0x0058	WT
LPADC_INT_ENA_W1TC_REG	LPADC_INT_ENA_REG configuration register.	0x005C	WT
LP ADC Wake-up Control Registers			
LPADC_WAKEUP1_REG	LP ADC1 wake-up configuration registers.	0x0060	varies
LPADC_WAKEUP2_REG	LP ADC2 wake-up configuration registers.	0x0064	varies
LPADC_WAKEUP_SEL_REG	Wake-up source selection register.	0x0068	R/W
LPADC_SAR1_HW_WAKEUP_REG	LP ADC1 automatic monitoring configuration registers.	0x006C	R/W
LPADC_SAR2_HW_WAKEUP_REG	LP ADC2 automatic monitoring configuration registers.	0x0070	R/W

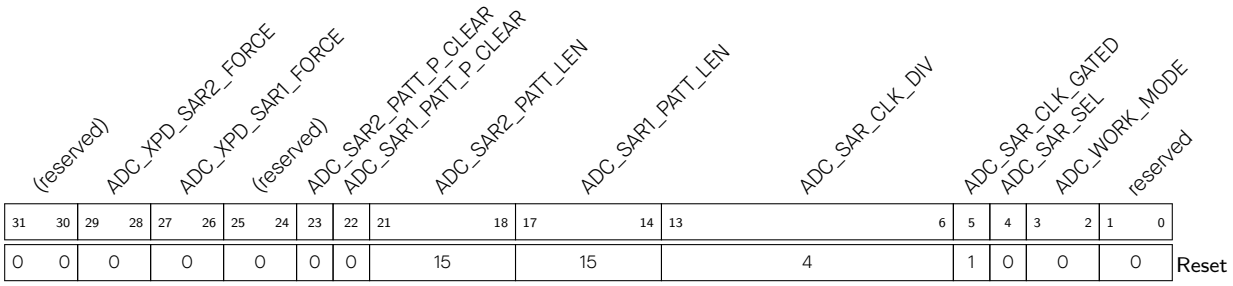
45.10 Registers

45.10.1 HP ADC Registers

The addresses in the following section are relative to the base address of HP ADC (written as ADC Controller) provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 45.1. ADC_CTRL_REG (0x0000)



ADC_WORK_MODE Configures dual HP ADC sampling.

0: Single HP ADC sampling

1: Dual HP ADC simultaneous sampling

2: Dual HP ADC alternative sampling.

(R/W)

ADC_SAR_SEL Configures which HP ADC to use for single HP ADC sampling.

0: Use HP ADC1

1: Use HP ADC2

(R/W)

ADC_SAR_CLK_GATED Enable HP ADC clock gate when HP ADC is in idle.

0: Disable

1: Enable

(R/W)

ADC_SAR_CLK_DIV Configures HP ADC clock division. Division = configured value + 1 . (R/W)

ADC_SAR1_PATT_LEN Configures how many pattern table entries will be used for HP ADC1.

0: Use only cmd0

1: Use cmd0 and cmd1

n: Use cmd0 to cmdn, the maximum n is 7

(R/W)

ADC_SAR2_PATT_LEN Configures how many pattern table entries will be used for HP ADC2.

0: Use only cmd0

1: Use cmd0 and cmd1

n: Use cmd0 to cmdn, the maximum n is 7

(R/W)

ADC_SAR1_PATT_P_CLEAR Clears the pointer of pattern table for HP ADC1 Controller.

0: No effect

1: Clear

(R/W)

ADC_SAR2_PATT_P_CLEAR Clears the pointer of pattern table for HP ADC2 Controller.

0: No effect

1: Clear

(R/W)

Continued on the next page...

Register 45.1. ADC_CTRL_REG (0x0000)

Continued from the previous page...

ADC_XPD_SAR1_FORCE Keep HP ADC1 powered on forcefully.

0: No effect

1: Power on forcefully

(R/W)

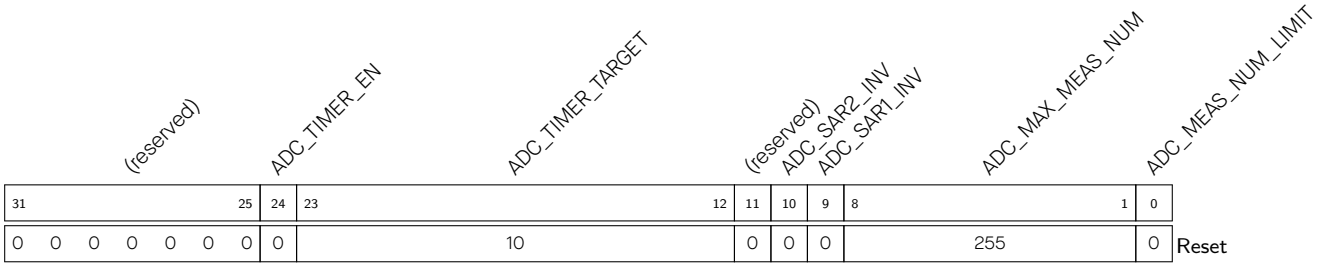
ADC_XPD_SAR2_FORCE Keep HP ADC2 powered on forcefully.

0: No effect

1: Power on forcefully

(R/W)

Register 45.2. ADC_CTRL2_REG (0x0004)



ADC_MEAS_NUM_LIMIT Configures whether to enable the limitation of HP ADC's maximum conversion times.

0: Disable

1: Enable

(R/W)

ADC_MAX_MEAS_NUM Configures the HP ADC's maximum conversion times. (R/W)

ADC_SAR1_INV Configures whether to invert the data of HP ADC1.

0: Do not invert.

1: Invert the date.

(R/W)

ADC_SAR2_INV Configures whether to to invert the data of HP ADC2.

0: Do not invert.

1: Invert the date.

(R/W)

ADC_TIMER_TARGET Configures HP ADC timer target. (R/W)

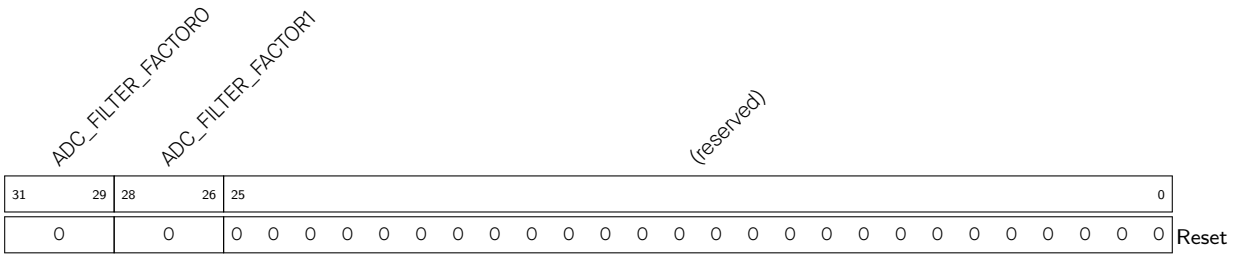
ADC_TIMER_EN Configures whether to enable HP ADC timer trigger.

0: Disable

1: Enable

(R/W)

Register 45.3. ADC_FILTER_CTRL1_REG (0x0008)



ADC_FILTER_FACTOR1 Configures the filter coefficient k for HP ADC filter 1.

0: k=0 (i.e., the filter is disabled)

1: k=2

2: k=4

3: k=8

4: k=16

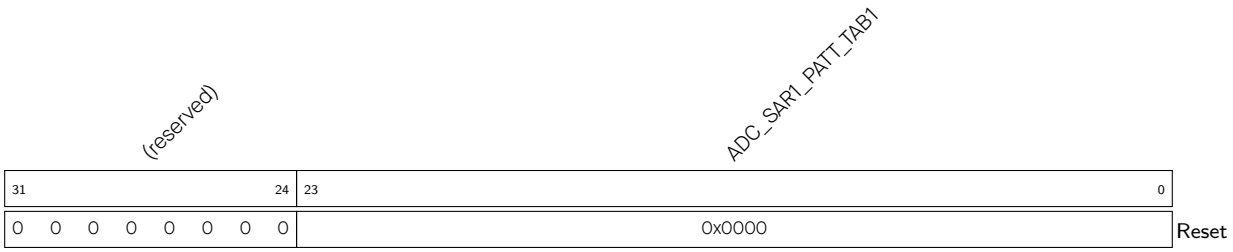
5: k=32

6: k=64

(R/W)

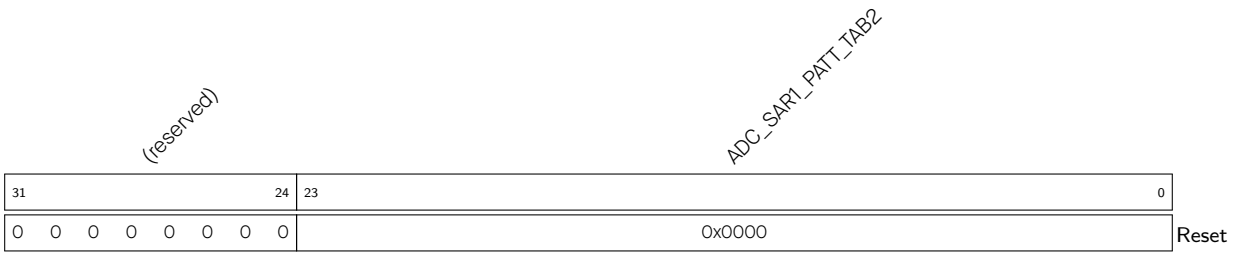
ADC_FILTER_FACTOR0 Configures the filter coefficient k for HP ADC filter 0 (same as above). (R/W)

Register 45.4. ADC_SAR1_PATT_TAB1_REG (0x0018)



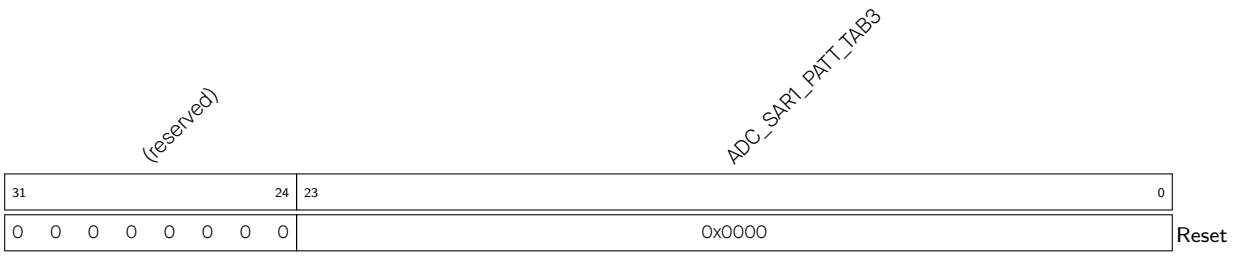
ADC_SAR1_PATT_TAB1 Configures entries 0 ~ 3 for HP ADC FSM1. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.5. ADC_SAR1_PATT_TAB2_REG (0x001C)



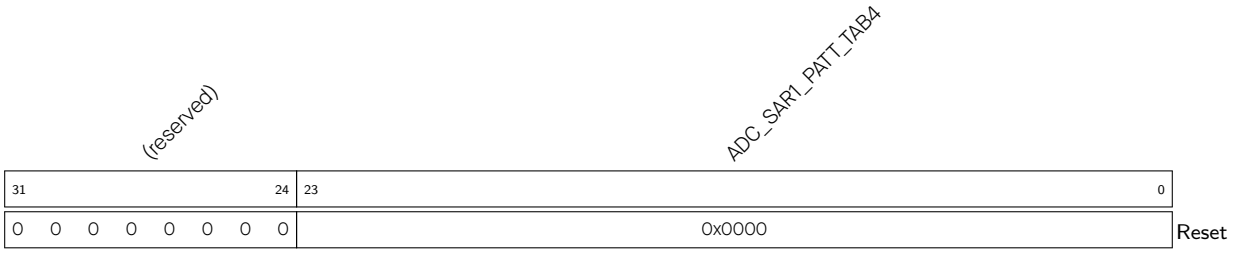
ADC_SAR1_PATT_TAB2 Configures entries 4 ~ 7 for HP ADC FSM1. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.6. ADC_SAR1_PATT_TAB3_REG (0x0020)



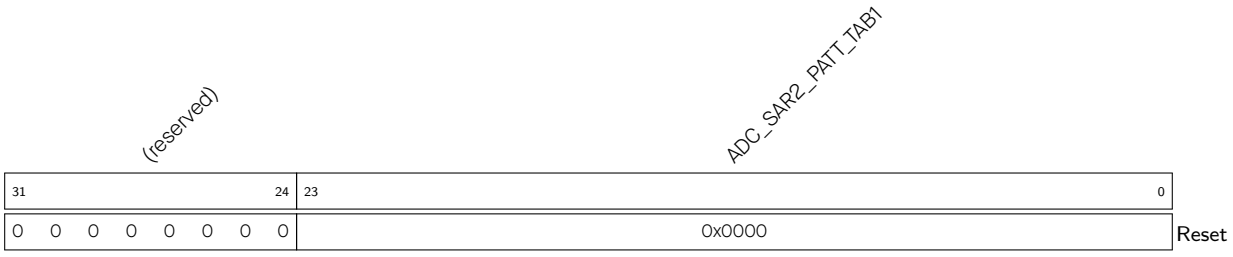
ADC_SAR1_PATT_TAB3 Configures entries 8 ~ 11 for HP ADC FSM1. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.7. ADC_SAR1_PATT_TAB4_REG (0x0024)



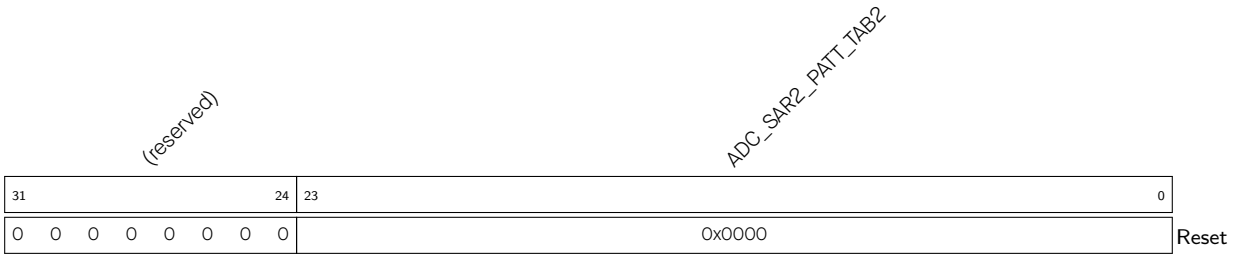
ADC_SAR1_PATT_TAB4 Configures entries 12 ~ 15 for HP ADC FSM1. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.8. ADC_SAR2_PATT_TAB1_REG (0x0028)



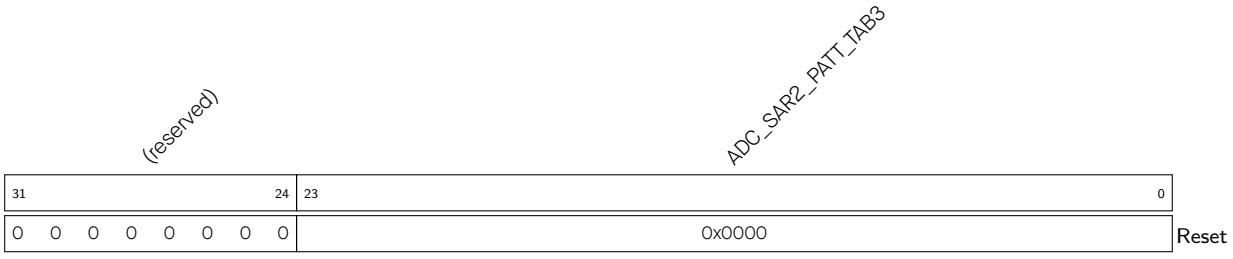
ADC_SAR2_PATT_TAB1 Configures entries 0 ~ 3 for HP ADC FSM2. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.9. ADC_SAR2_PATT_TAB2_REG (0x002C)



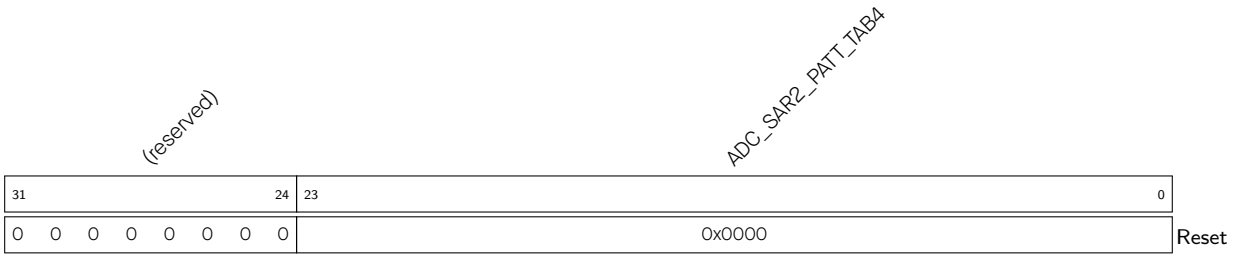
ADC_SAR2_PATT_TAB2 Configures entries 4 ~ 7 for HP ADC FSM2. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.10. ADC_SAR2_PATT_TAB3_REG (0x0030)



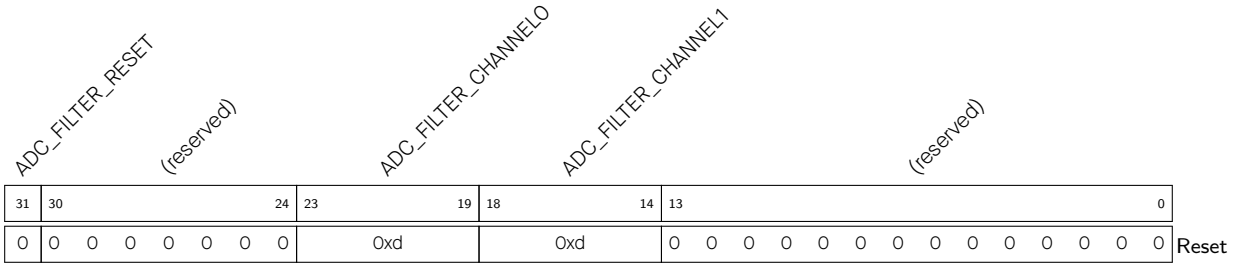
ADC_SAR2_PATT_TAB3 Configures entries 8 ~ 11 for HP ADC FSM2. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.11. ADC_SAR2_PATT_TAB4_REG (0x0034)



ADC_SAR2_PATT_TAB4 Configures entries 12 ~ 15 for HP ADC FSM2. Each entry is 6-bit wide. See Section 45.5.6 *HP ADC Pattern Table*. (R/W)

Register 45.12. ADC_FILTER_CTRL0_REG (0x003C)



ADC_FILTER_CHANNEL0 Configures the ADC channel for filter 0. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 2 ~ 7 in HP ADC2. (R/W)

ADC_FILTER_CHANNEL1 Configure the ADC channel for filter 1. Same as above. (R/W)

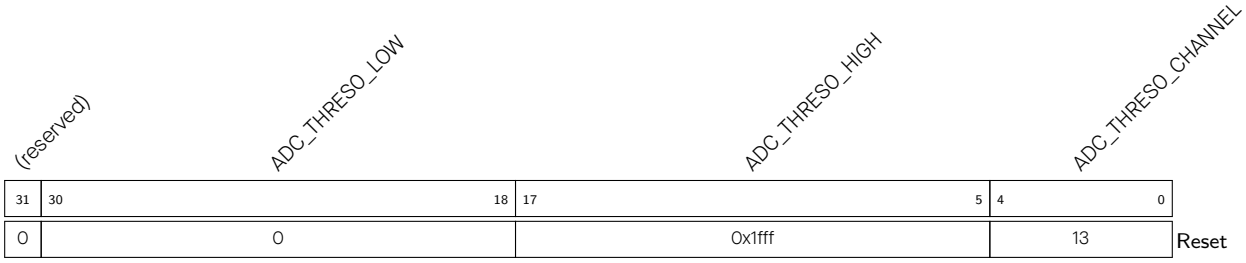
ADC_FILTER_RESET Configures whether to reset the filter.

0: No effect

1: Reset

(R/W)

Register 45.13. ADC_THRES0_CTRL_REG (0x0044)

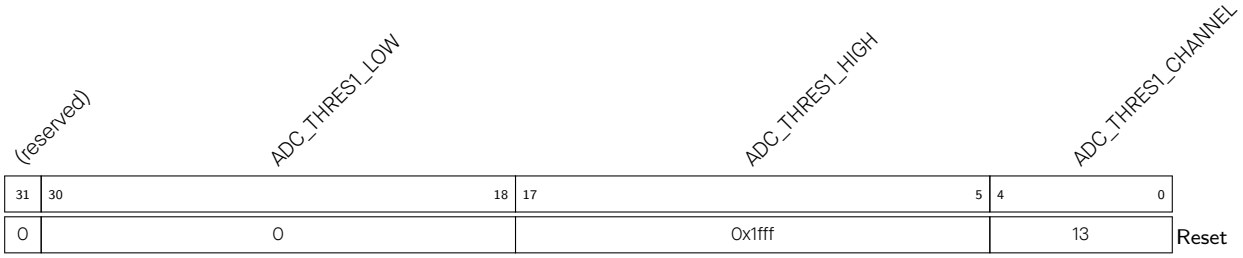


ADC_THRES0_CHANNEL Configures the ADC channel for threshold monitor 0. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 2 ~ 7 in HP ADC2. (R/W)

ADC_THRES0_HIGH Configures the high threshold for HP ADC threshold monitor 0. (R/W)

ADC_THRES0_LOW Configures the low threshold for HP ADC threshold monitor 0. (R/W)

Register 45.14. ADC_THRES1_CTRL_REG (0x0048)

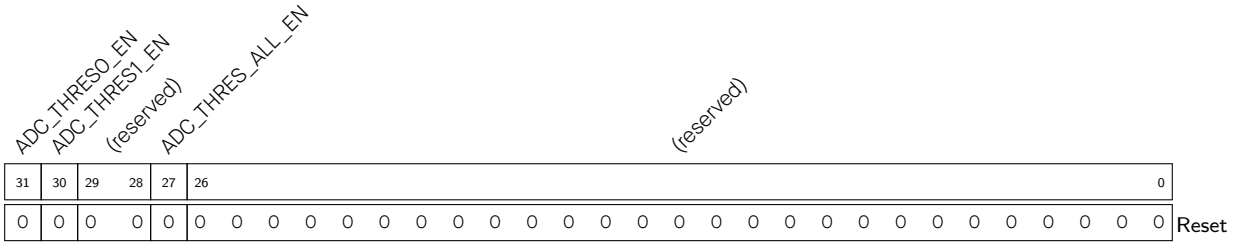


ADC_THRES1_CHANNEL Configures the ADC channel for threshold monitor 1. Value 0 ~ 7 corresponds to channel 0 ~ 7 in HP ADC1, value 10 ~ 15 corresponds to channel 2 ~ 7 in HP ADC2. (R/W)

ADC_THRES1_HIGH Configures the high threshold for HP ADC threshold monitor 1. (R/W)

ADC_THRES1_LOW Configures the low threshold for HP ADC threshold monitor 1. (R/W)

Register 45.15. ADC_THRES_CTRL_REG (0x004C)



ADC_THRES_ALL_EN Configures whether to enable the threshold monitoring for all configured channels.

0: Disable

1: Enable

(R/W)

ADC_THRES1_EN Configures whether to enable threshold monitor 1.

0: Disable

1: Enable

(R/W)

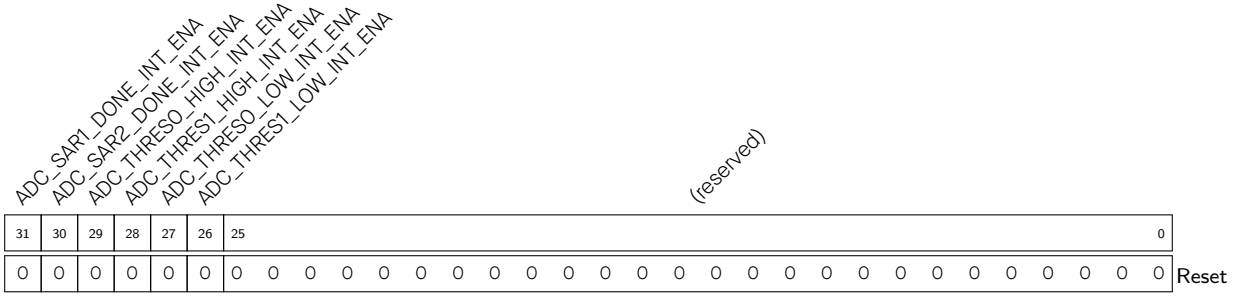
ADC_THRES0_EN Configures whether to enable threshold monitor 0.

0: Disable

1: Enable

(R/W)

Register 45.16. ADC_INT_ENA_REG (0x0050)



ADC_THRES1_LOW_INT_ENA Enable bit of ADC_THRES1_LOW_INT. (R/W)

ADC_THRES0_LOW_INT_ENA Enable bit of ADC_THRES0_LOW_INT. (R/W)

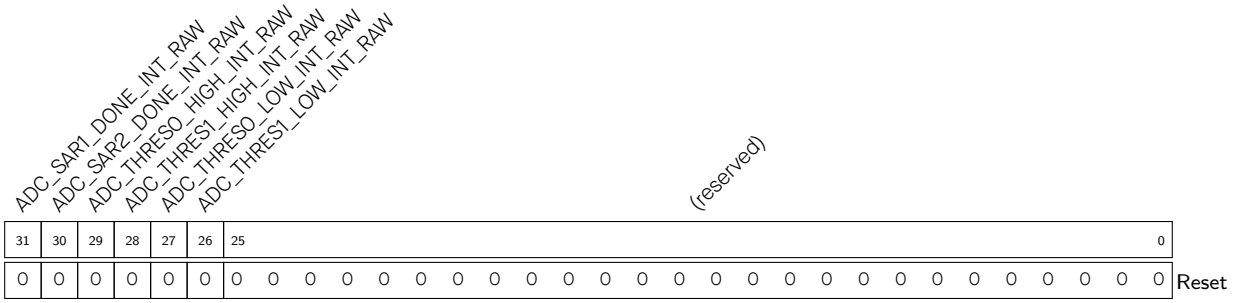
ADC_THRES1_HIGH_INT_ENA Enable bit of ADC_THRES1_HIGH_INT. (R/W)

ADC_THRES0_HIGH_INT_ENA Enable bit of ADC_THRES0_HIGH_INT. (R/W)

ADC_SAR2_DONE_INT_ENA Enable bit of ADC_SAR2_DONE_INT. (R/W)

ADC_SAR1_DONE_INT_ENA Enable bit of ADC_SAR1_DONE_INT. (R/W)

Register 45.17. ADC_INT_RAW_REG (0x0054)



ADC_THRES1_LOW_INT_RAW Raw bit of ADC_THRES1_LOW_INT. (R/WTC/SS)

ADC_THRES0_LOW_INT_RAW Raw bit of ADC_THRES0_LOW_INT. (R/WTC/SS)

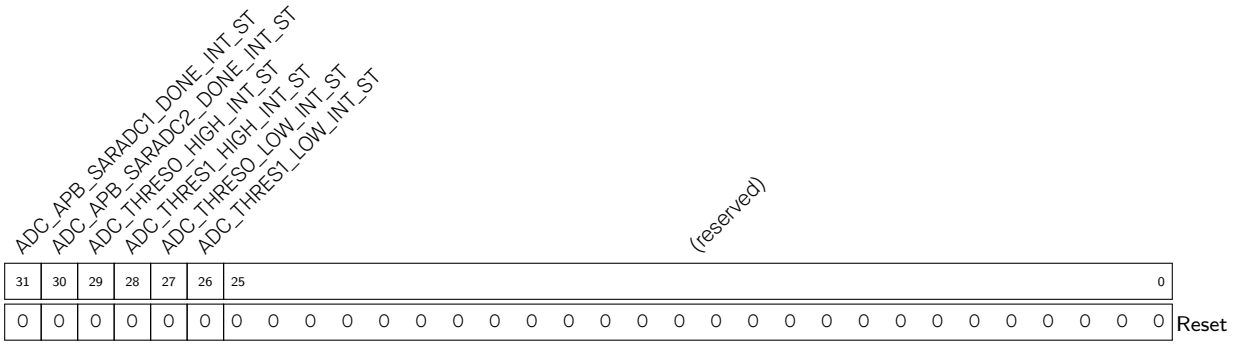
ADC_THRES1_HIGH_INT_RAW Raw bit of ADC_THRES1_HIGH_INT. (R/WTC/SS)

ADC_THRES0_HIGH_INT_RAW Raw bit of ADC_THRES0_HIGH_INT. (R/WTC/SS)

ADC_SAR2_DONE_INT_RAW Raw bit of ADC_SAR2_DONE_INT. (R/WTC/SS)

ADC_SAR1_DONE_INT_RAW Raw bit of ADC_SAR1_DONE_INT. (R/WTC/SS)

Register 45.18. ADC_INT_ST_REG (0x0058)



ADC_THRES1_LOW_INT_ST Status bit of ADC_THRES1_LOW_INT. (RO)

ADC_THRES0_LOW_INT_ST Status bit of ADC_THRES0_LOW_INT. (RO)

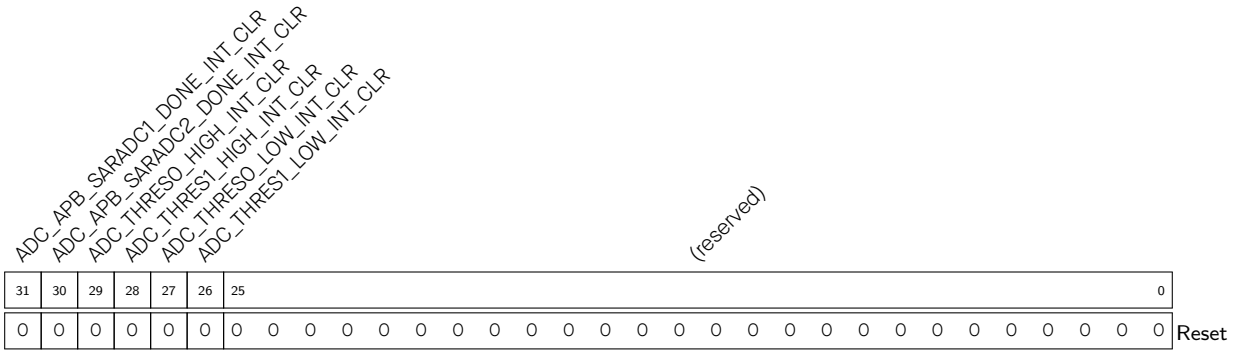
ADC_THRES1_HIGH_INT_ST Status bit of ADC_THRES1_HIGH_INT. (RO)

ADC_THRES0_HIGH_INT_ST Status bit of ADC_THRES0_HIGH_INT. (RO)

ADC_APB_SARADC2_DONE_INT_ST Status bit of ADC_SAR2_DONE_INT. (RO)

ADC_APB_SARADC1_DONE_INT_ST Status bit of ADC_SAR1_DONE_INT. (RO)

Register 45.19. ADC_INT_CLR_REG (0x005C)



ADC_THRES1_LOW_INT_CLR Clear bit of ADC_THRES1_LOW_INT. (WT)

ADC_THRES0_LOW_INT_CLR Clear bit of ADC_THRES0_LOW_INT. (WT)

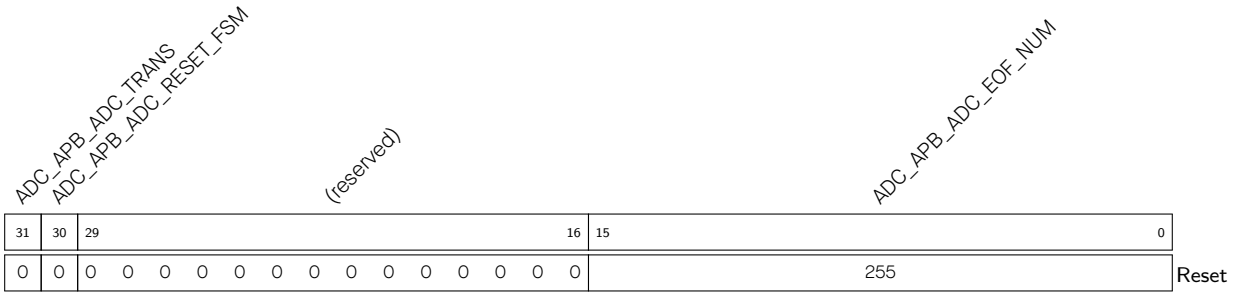
ADC_THRES1_HIGH_INT_CLR Clear bit of ADC_THRES1_HIGH_INT. (WT)

ADC_THRES0_HIGH_INT_CLR Clear bit of ADC_THRES0_HIGH_INT. (WT)

ADC_APB_SARADC2_DONE_INT_CLR Clear bit of ADC_SAR2_DONE_INT. (WT)

ADC_APB_SARADC1_DONE_INT_CLR Clear bit of ADC_SAR1_DONE_INT. (WT)

Register 45.20. ADC_DMA_CONF_REG (0x0060)



ADC_APB_ADC_EOF_NUM Configures the number of samples. When the sampling reaches the configured number, the EOF flag bit sent to GDMA will be pulled high. (R/W)

ADC_APB_ADC_RESET_FSM Configures whether to reset the status of HP ADC controller.

0: No effect

1: Reset

(R/W)

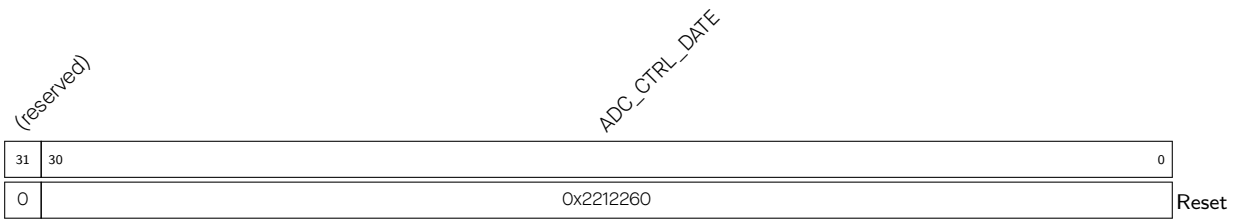
ADC_APB_ADC_TRANS Configures whether to let HP ADC controller use GDMA.

0: No effect

1: HP ADC controller uses DMA

(R/W)

Register 45.21. ADC_CTRL_DATE_REG (0x03FC)



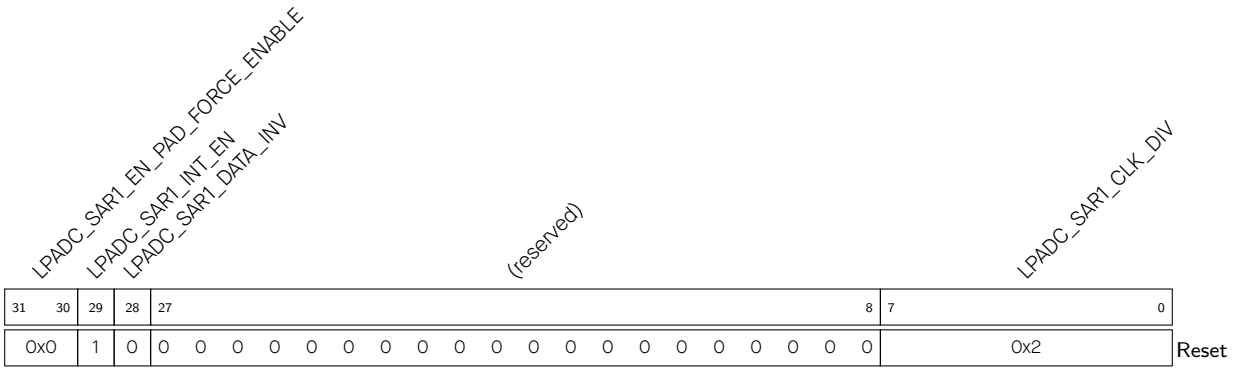
ADC_CTRL_DATE HP ADC version control register. (RO)

45.10.2 LP ADC Registers

The addresses in the following section are relative to LP ADC's base address provided in Table 5.3-2 in Chapter 5 *System and Memory*.

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

Register 45.22. LPADC_READER1_CTRL_REG (0x0000)



LPADC_SAR1_CLK_DIV Configures the division of LP ADC1's clock LPADC_SARCLK. (R/W)

LPADC_SAR1_DATA_INV Configures whether to to invert the data of LP ADC1.

0: Do not invert the data.

1: Invert the data.

(R/W)

LPADC_SAR1_INT_EN Enables LP ADC1 to send out interrupt. (R/W)

LPADC_SAR1_EN_PAD_FORCE_ENABLE Configures whether to use software to force enable LPADC_SAR1_EN_PAD.

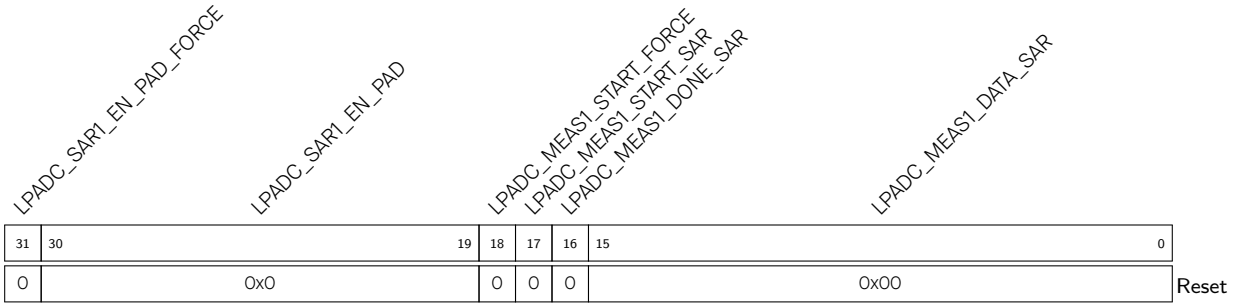
2: Force off

3: Force enable

Other values: Hardware control (enabled during sampling)

(R/W)

Register 45.23. LPADC_MEAS1_CTRL2_REG (0x000C)



LPADC_MEAS1_DATA_SAR Stores LP ADC1's conversion data in one-shot sampling mode. (RO)

LPADC_MEAS1_DONE_SAR Indicates whether LP ADC1 conversion is done. (RO)

LPADC_MEAS1_START_SAR Configures whether to start LP ADC1 by software.

0: No effect

1: Start LP ADC1 by software

Valid only when LPADC_MEAS1_START_FORCE = 1.

(R/W)

LPADC_MEAS1_START_FORCE Configures whether to use software to enable LP ADC1 sampling.

0: Select FSM to start LP ADC1 sampling

1: Select software to start LP ADC1 sampling

(R/W)

LPADC_SAR1_EN_PAD Configures the sampling channel for LP ADC1. Value 0 ~ 7 corresponds to channel 0 ~ 7. (R/W)

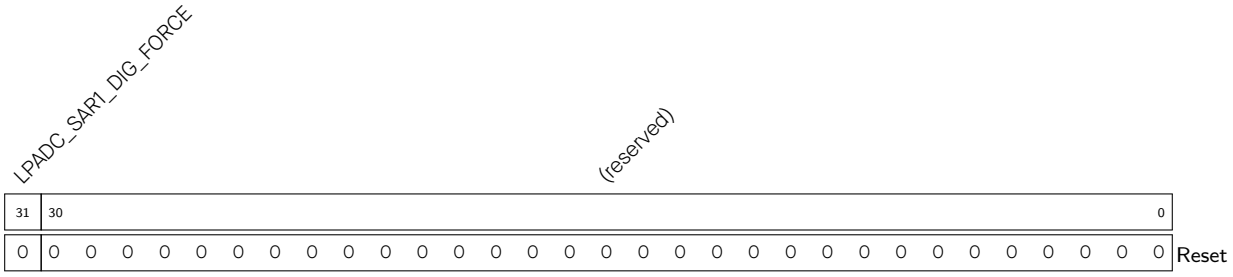
LPADC_SAR1_EN_PAD_FORCE Configures whether to use software to select the sampling channel for LP ADC1.

0: No effect

1: Use software to control

(R/W)

Register 45.24. LPADC_MEAS1_MUX_REG (0x0010)



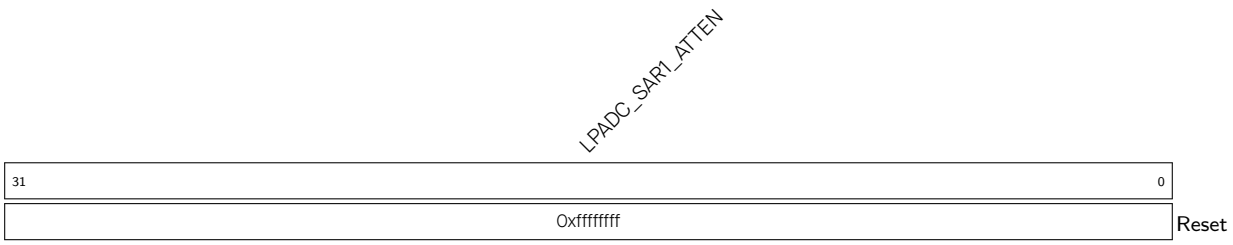
LPADC_SAR1_DIG_FORCE Configures whether to use HP ADC1 to control SAR ADC1.

0: SAR ADC1 controlled by LP ADC1 CTRL

1: SAR ADC1 controlled by HP ADC1 CTRL

(R/W)

Register 45.25. LPADC_ATTEN1_REG (0x0014)



LPADC_SAR1_ATTEN Configures the attenuation value for one-shot sampling of LP ADC1.

0: 0 dB

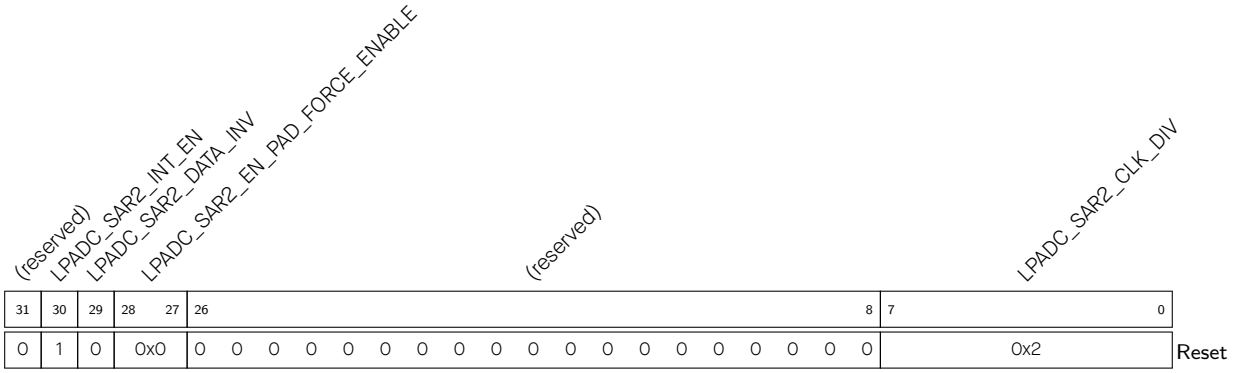
1: 2.5 dB

2: 6 dB

3: 12 dB

(R/W)

Register 45.26. LPADC_READER2_CTRL_REG (0x0024)



LPADC_SAR2_CLK_DIV Configures the division of LP ADC2's clock LPADC_SARCLK. (R/W)

LPADC_SAR2_EN_PAD_FORCE_ENABLE Configures whether to use software to force enable LPADC_SAR2_EN_PAD.

2: Force off

3: Force enable

Other values: Hardware control (enabled during sampling)

(R/W)

LPADC_SAR2_DATA_INV onfigures whether to to invert the data of LP ADC2.

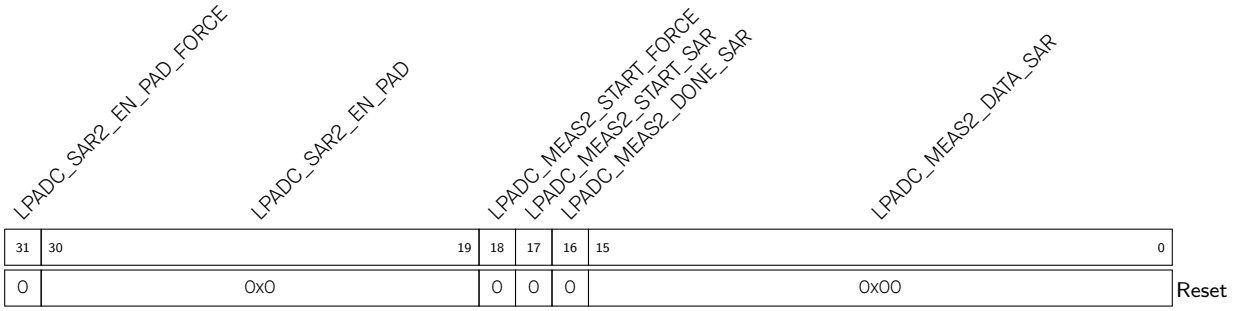
0: Do not invert the data.

1: Invert the data.

(R/W)

LPADC_SAR2_INT_EN Enable LP ADC2 to send out interrupt. (R/W)

Register 45.27. LPADC_MEAS2_CTRL2_REG (0x0030)



LPADC_MEAS2_DATA_SAR Stores LP ADC2's conversion data in one-shot sampling mode. (RO)

LPADC_MEAS2_DONE_SAR Indicates whether LP ADC2 conversion is done. (RO)

LPADC_MEAS2_START_SAR Configures whether to start LP ADC2 by software.

0: No effect

1: Start LP ADC2 by software

Valid only when LPADC_MEAS2_START_FORCE = 1.

(R/W)

LPADC_MEAS2_START_FORCE Configures whether to use software to enable LP ADC2 sampling.

0: Select FSM to start LP ADC2 sampling

1: Select software to start LP ADC2 sampling

(R/W)

LPADC_SAR2_EN_PAD Configures the sampling channel for LP ADC2. Value 2 ~ 7 corresponds to channel 2 ~ 7. (R/W)

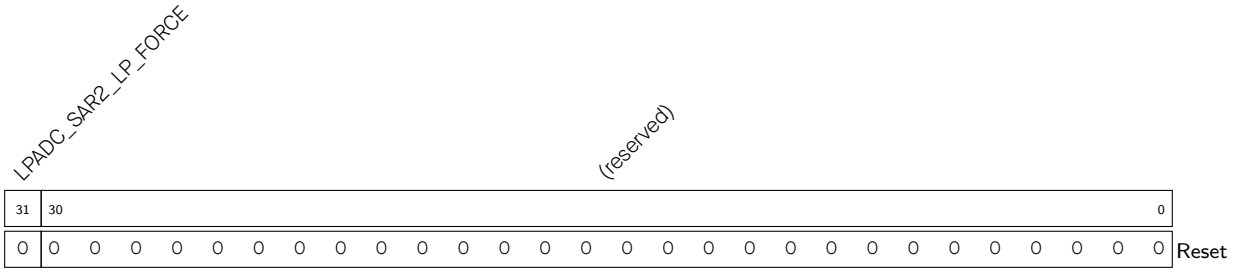
LPADC_SAR2_EN_PAD_FORCE Configures whether to use software to select the sampling channel for LP ADC1.

0: No effect

1: Use software to control

(R/W)

Register 45.28. LPADC_MEAS2_MUX_REG (0x0034)



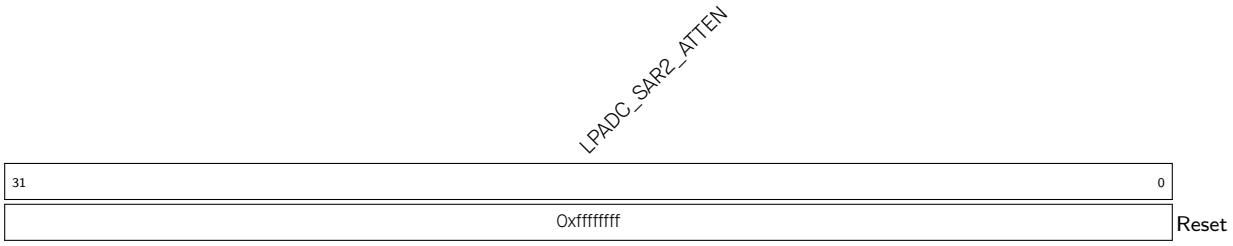
LPADC_SAR2_LP_FORCE Configures whether to disable HP ADC2 Controller to control SAR ADC2.

0: Do not disable HP ADC2 Controller.

1: Disable HP ADC2 Controller.

(R/W)

Register 45.29. LPADC_ATTEN2_REG (0x0038)



LPADC_SAR2_ATTEN Configures the attenuation value for one-shot sampling of LP ADC2.

0: 0 dB

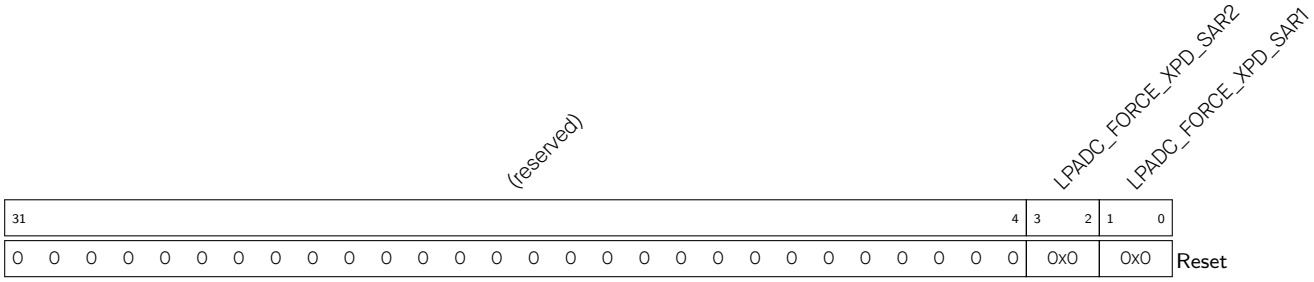
1: 2.5 dB

2: 6 dB

3: 12 dB

(R/W)

Register 45.30. LPADC_FORCE_WPD_SAR_REG (0x003C)



LPADC_FORCE_XPD_SAR1 Configures the power control for LP ADC1.

3: Software control, force on.

2: Software control, force off.

1/0: Hardware control.

(R/W)

LPADC_FORCE_XPD_SAR2 Configures the power control for LP ADC2.

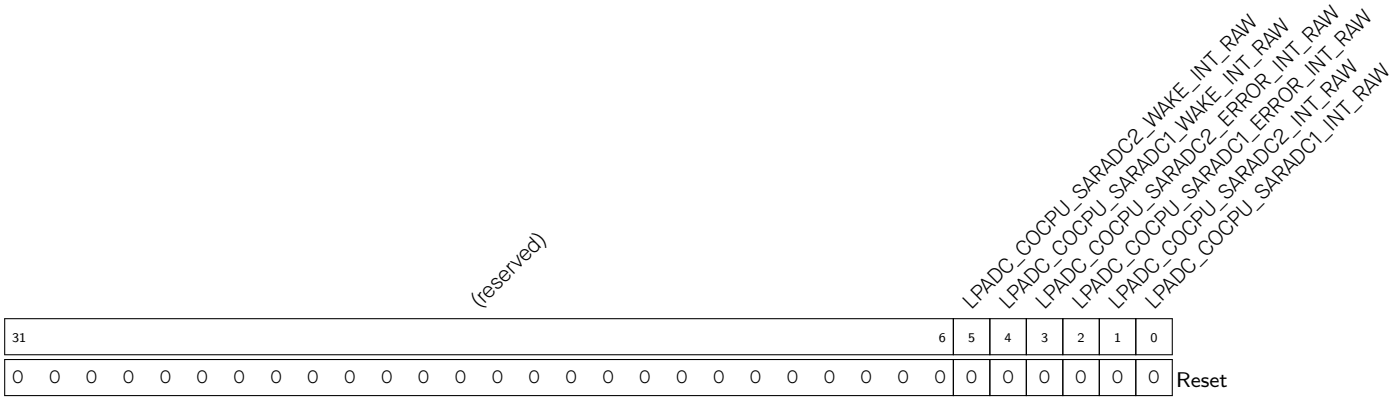
3: Software control, force on.

2: Software control, force off.

1/0: Hardware control.

(R/W)

Register 45.31. LPADC_COCPU_INT_RAW_REG (0x0048)



LPADC_COCPU_SARADC1_INT_RAW Raw bit of LPADC_COCPU_SARADC1_DONE_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_INT_RAW Raw bit of LPADC_COCPU_SARADC2_DONE_INT. (R/WTC/SS)

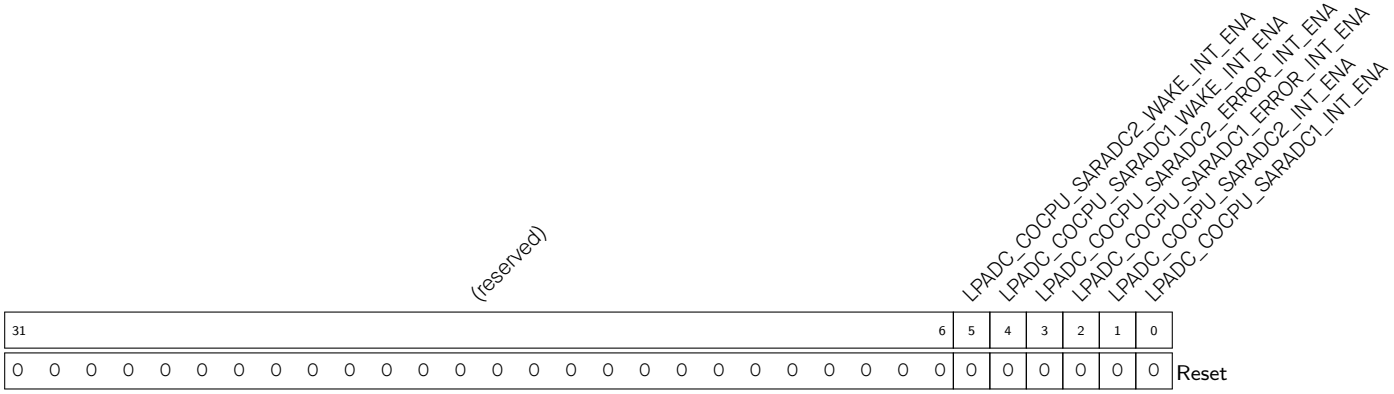
LPADC_COCPU_SARADC1_ERROR_INT_RAW Raw bit of LPADC_COCPU_SARADC1_ERROR_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_ERROR_INT_RAW Raw bit of LPADC_COCPU_SARADC2_ERROR_INT. (R/WTC/SS)

LPADC_COCPU_SARADC1_WAKE_INT_RAW Raw bit of LPADC_COCPU_SARADC1_WAKE_INT. (R/WTC/SS)

LPADC_COCPU_SARADC2_WAKE_INT_RAW Raw bit of LPADC_COCPU_SARADC2_WAKE_INT. (R/WTC/SS)

Register 45.32. LPADC_INT_ENA_REG (0x004C)



LPADC_COCPU_SARADC1_INT_ENA Enable bit of LPADC_COCPU_SARADC1_DONE_INT. (R/WTC)

LPADC_COCPU_SARADC2_INT_ENA Enable bit of LPADC_COCPU_SARADC2_DONE_INT. (R/WTC)

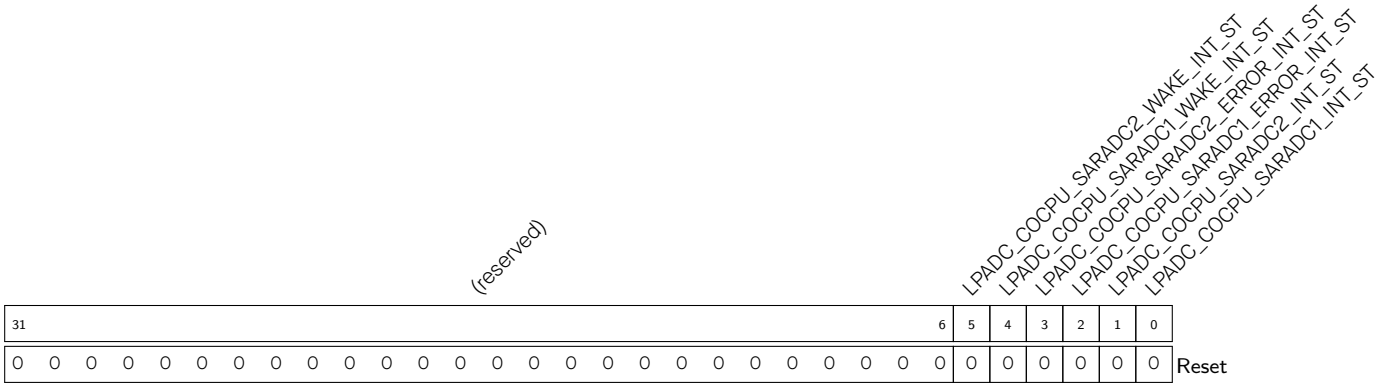
LPADC_COCPU_SARADC1_ERROR_INT_ENA Enable bit of LPADC_COCPU_SARADC1_ERROR_INT. (R/WTC)

LPADC_COCPU_SARADC2_ERROR_INT_ENA Enable bit of LPADC_COCPU_SARADC2_ERROR_INT. (R/WTC)

LPADC_COCPU_SARADC1_WAKE_INT_ENA Enable bit of LPADC_COCPU_SARADC1_WAKE_INT. (R/WTC)

LPADC_COCPU_SARADC2_WAKE_INT_ENA Enable bit of LPADC_COCPU_SARADC2_WAKE_INT. (R/WTC)

Register 45.33. LPADC_INT_ST_REG (0x0050)



LPADC_COCPU_SARADC1_INT_ST Status bit of LPADC_COCPU_SARADC1_DONE_INT. (RO)

LPADC_COCPU_SARADC2_INT_ST Status bit of LPADC_COCPU_SARADC2_DONE_INT. (RO)

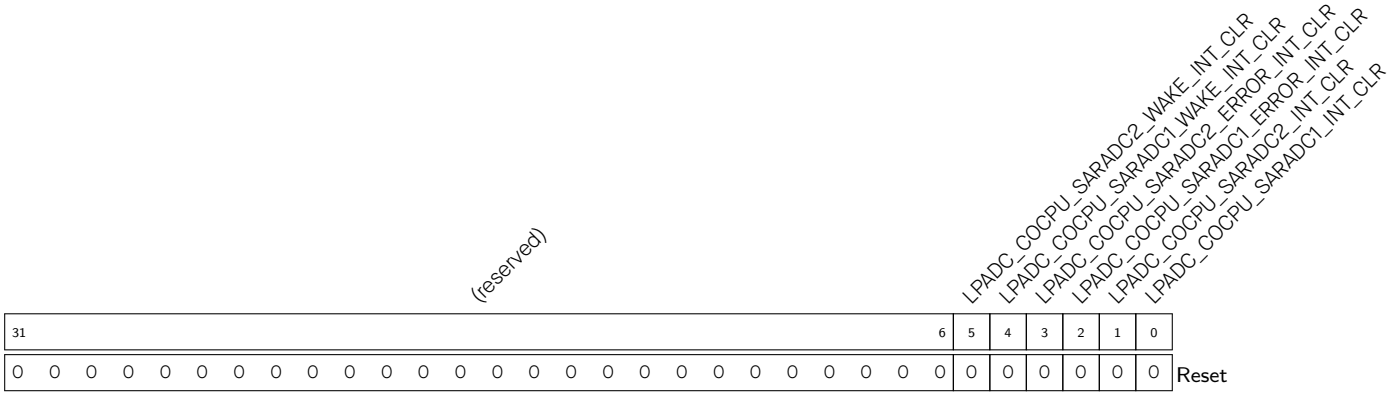
LPADC_COCPU_SARADC1_ERROR_INT_ST Status bit of LPADC_COCPU_SARADC1_ERROR_INT. (RO)

LPADC_COCPU_SARADC2_ERROR_INT_ST Status bit of LPADC_COCPU_SARADC2_ERROR_INT. (RO)

LPADC_COCPU_SARADC1_WAKE_INT_ST Status bit of LPADC_COCPU_SARADC1_WAKE_INT. (RO)

LPADC_COCPU_SARADC2_WAKE_INT_ST Status bit of LPADC_COCPU_SARADC2_WAKE_INT. (RO)

Register 45.34. LPADC_INT_CLR_REG (0x0054)



LPADC_COCPU_SARADC1_INT_CLR Clear bit of LPADC_COCPU_SARADC1_DONE_INT. (WT)

LPADC_COCPU_SARADC2_INT_CLR Clear bit of LPADC_COCPU_SARADC2_DONE_INT. (WT)

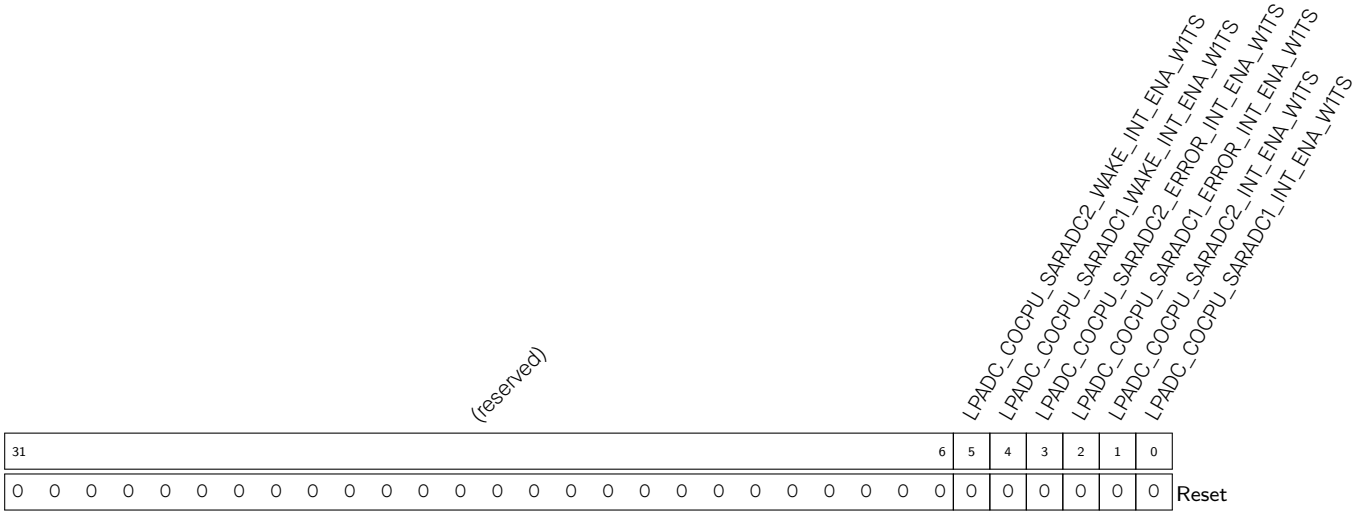
LPADC_COCPU_SARADC1_ERROR_INT_CLR Clear bit of LPADC_COCPU_SARADC1_ERROR_INT. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_CLR Clear bit of LPADC_COCPU_SARADC2_ERROR_INT. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_CLR Clear bit of LPADC_COCPU_SARADC1_WAKE_INT. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_CLR Clear bit of LPADC_COCPU_SARADC2_WAKE_INT. (WT)

Register 45.35. LPADC_INT_ENA_W1TS_REG (0x0058)



LPADC_COCPU_SARADC1_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_INT_ENA. (WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_INT_ENA. (WT)

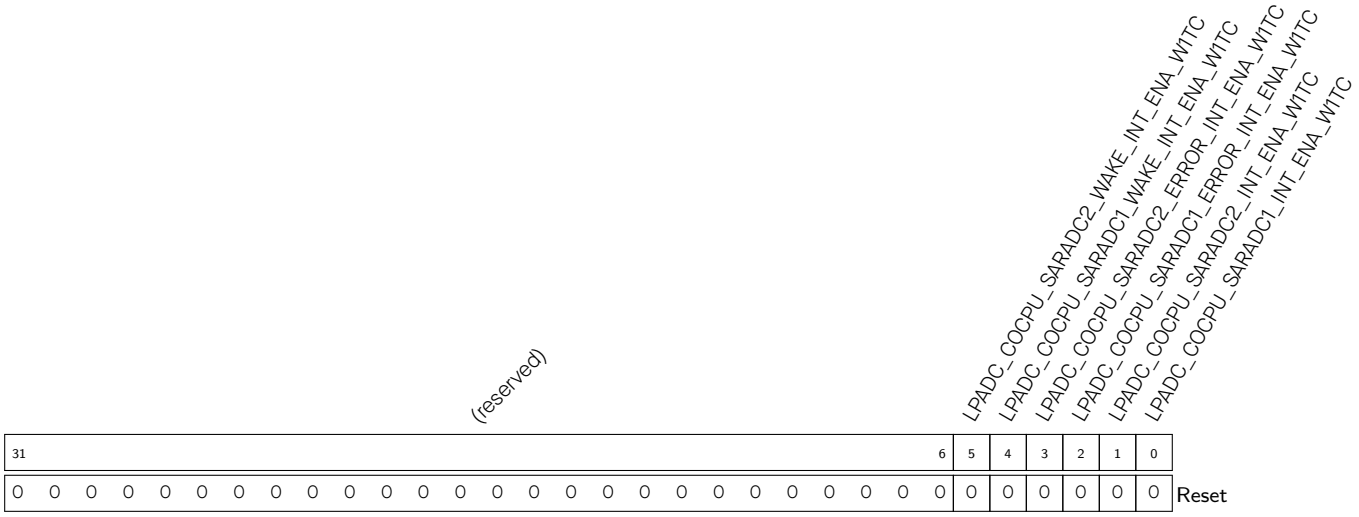
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC1_WAKE_INT_ENA. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TS Write 1 to enable LPADC_COCPU_SARADC2_WAKE_INT_ENA. (WT)

Register 45.36. LPADC_INT_ENA_W1TC_REG (0x005C)



LPADC_COCPU_SARADC1_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_INT_ENA. (WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_INT_ENA. (WT)

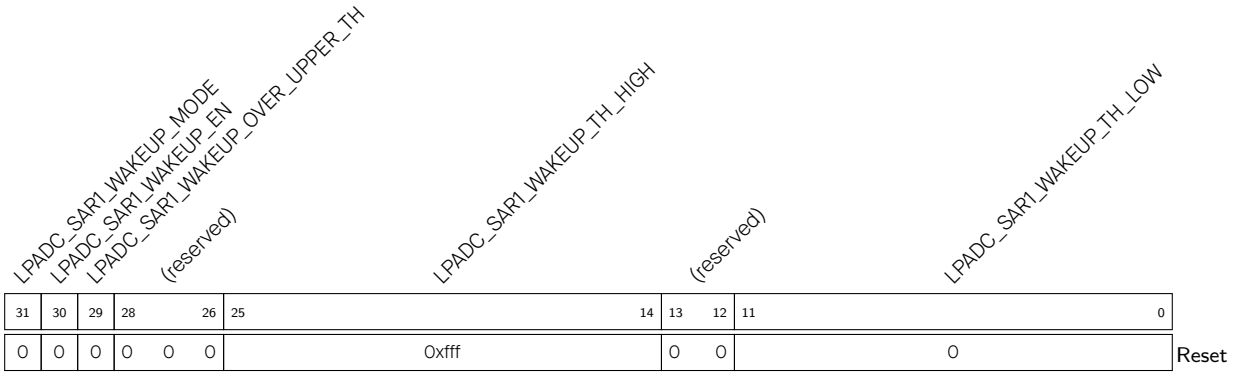
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_ERROR_INT_ENA. (WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC1_WAKE_INT_ENA. (WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TC Write 1 to clear LPADC_COCPU_SARADC2_WAKE_INT_ENA. (WT)

Register 45.37. LPADC_WAKEUP1_REG (0x0060)



LPADC_SAR1_WAKEUP_TH_LOW Configures the low threshold for LP ADC1's wake-up function. (R/W)

LPADC_SAR1_WAKEUP_TH_HIGH Configures the high threshold for LP ADC1's wake-up function. (R/W)

LPADC_SAR1_WAKEUP_OVER_UPPER_TH Indicates whether the wake-up event is triggered by the conversion data exceeding high threshold. (RO)

LPADC_SAR1_WAKEUP_EN Configures whether to enable wake-up function for LP ADC1.

0: Disable

1: Enable

(R/W)

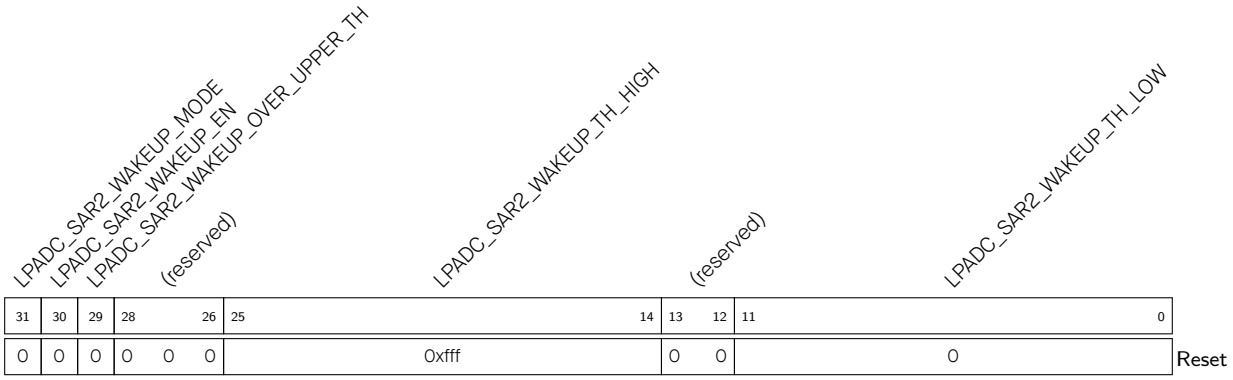
LPADC_SAR1_WAKEUP_MODE Selects the wake-up mode for automatic monitoring. Valid only when LPADC_SAR1_WAKEUP_EN=1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 45.38. LPADC_WAKEUP2_REG (0x0064)



LPADC_SAR2_WAKEUP_TH_LOW Configures the low threshold for LP ADC2's wake-up function. (R/W)

LPADC_SAR2_WAKEUP_TH_HIGH Configures the high threshold for LP ADC2's wake-up function. (R/W)

LPADC_SAR2_WAKEUP_OVER_UPPER_TH Indicates whether the wake-up event is triggered by the conversion data exceeding high threshold. (RO)

LPADC_SAR2_WAKEUP_EN Configures whether to enable wake-up function for LP ADC2.

0: Disable

1: Enable

(R/W)

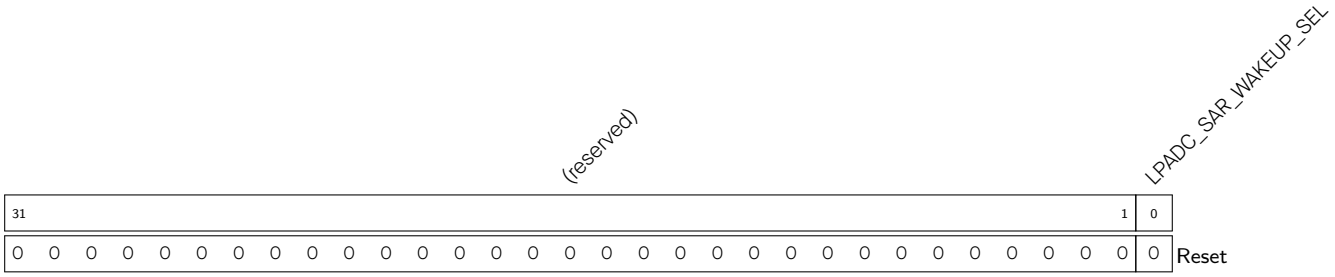
LPADC_SAR2_WAKEUP_MODE Selects the wake-up mode for automatic monitoring. Valid only when LPADC_SAR2_WAKEUP_EN=1.

0: Absolute value mode

1: Change value mode

(R/W)

Register 45.39. LPADC_WAKEUP_SEL_REG (0x0068)

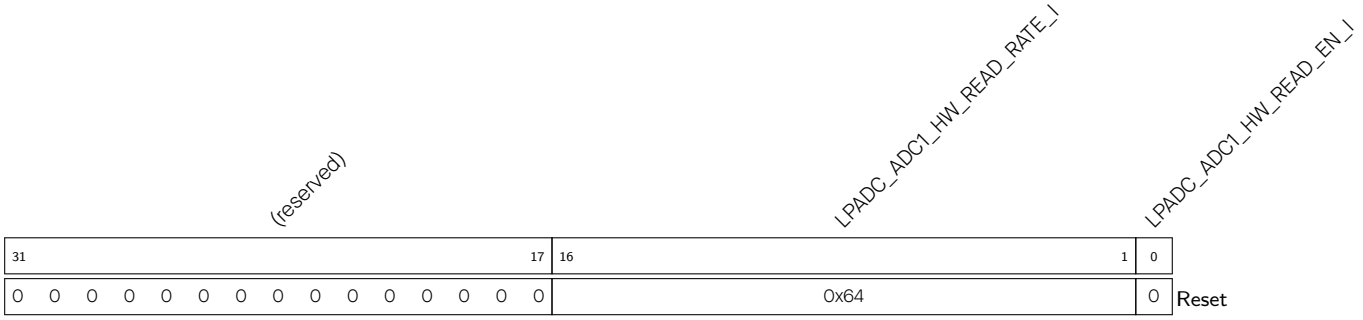


LPADC_SAR_WAKEUP_SEL Enables the wake-up function for LP ADC1 or LP ADC2. 0: LP ADC1.

1: LP ADC2.

(R/W)

Register 45.40. LPADC_SAR1_HW_WAKEUP_REG (0x006C)



LPADC_ADC1_HW_READ_EN_I Configures whether to enable automatic monitoring for LP ADC1.

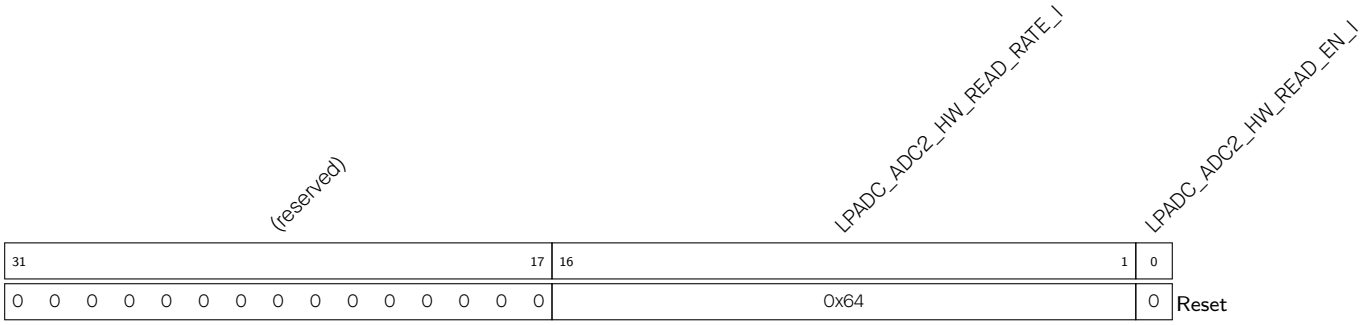
0: Disable

1: Enable

(R/W)

LPADC_ADC1_HW_READ_RATE_I Configures the sampling rate for hardware-triggered automatic monitoring for LP ADC1. The sampling period = configured value × LP ADC1's working clock cycle. (R/W)

Register 45.41. LPADC_SAR2_HW_WAKEUP_REG (0x0070)



LPADC_ADC2_HW_READ_EN_I Configures whether to enable automatic monitoring for LP ADC2.

0: Disable

1: Enable

(R/W)

LPADC_ADC2_HW_READ_RATE_I Configures the sampling rate for hardware-triggered automatic monitoring for LP ADC2. The sampling period = configured value × LP ADC2's working clock cycle. (R/W)

Chapter 46

Analog Voltage Comparator

46.1 Introduction

ESP32-P4 integrates two analog voltage comparators, i.e., analog voltage comparator *0* and analog voltage comparator *1*. These comparators rely on special pads that support voltage comparison functionality to monitor voltage changes on these pads. Each analog voltage comparator has two pads associated with it, for the main voltage and the reference voltage respectively. The voltage comparison result generated by the analog voltage comparator can be used as Event Task Matrix (ETM) events to drive ETM tasks of other peripherals or trigger interrupts.

Note:
The information provided in this chapter applies to both analog voltage comparator *0* and analog voltage comparator *1*. Unless otherwise indicated, the analog voltage comparator in this chapter refer to both analog voltage comparator *0* and analog voltage comparator *1*.

46.2 Feature List

The analog voltage comparator has the following features:

- Voltage comparison
  - Configurable voltage comparison mode
  - Configurable reference voltage
- Interrupt upon changes of voltage comparison result
- ETM event generation

46.3 Architectural Overview

Figure 46.3-1 shows the structure of the analog voltage comparator.

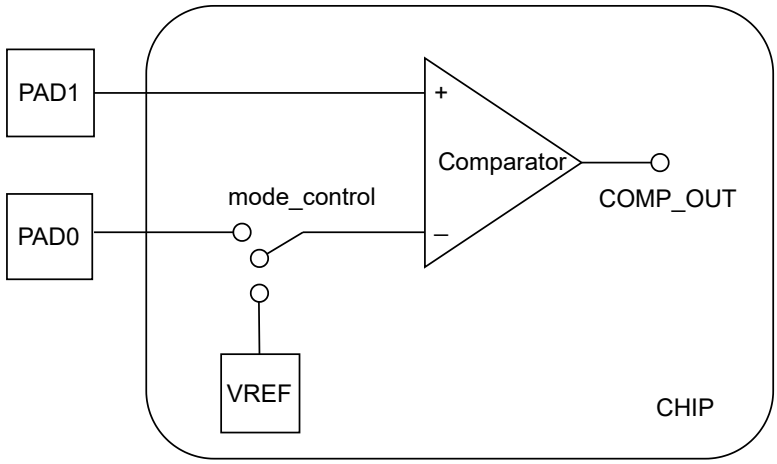


Figure 46.3-1. Analog Voltage Comparator Architecture

As shown in the figure above, the analog voltage comparator consists of:

- Two pad interfaces:
  - PAD1: Main voltage interface
  - PAD0: External reference voltage interface
- VREF: Internal reference voltage interface
- mode_control: Register-controlled switch for selecting the reference voltage source
- COMP_OUT: Comparator voltage comparison result, which is an on-chip internal signal that cannot be directly accessed externally
  - If the main voltage is higher than the reference voltage, the COMP_OUT signal is high.
  - If the main voltage is lower than the reference voltage, the COMP_OUT signal is low.

46.4 Functional Description

The analog voltage comparator has the following functions:

- **Voltage comparison**

  The analog voltage comparator relies on a specialized pads that support voltage comparison. For more information about general-purpose pads, please refer to Chapter 7 GPIO Matrix and IO MUX. Table below shows the mapping between the PAD of analog voltage comparator *0* and GPIO pins.

Table 46.4-1. Mapping Between PAD and GPIO

Analog Voltage Comparator	PAD0	PAD1
Analog voltage comparator *0*	GPIO51	GPIO52
Analog voltage comparator *1*	GPIO53	GPIO54

  The comparison mode and reference voltage are configurable. For detailed configuration procedures, see Section 46.7 Programming Procedures.

  The voltage comparison results are output as the COMP_OUT signal.

- **Voltage comparison mode control**

  The analog voltage comparator compares the main voltage with the reference voltage, which can either be internal or external. There are two voltage comparison modes depending on the reference voltage selection:

  - Comparing the main voltage with the external reference voltage.
  - Comparing the main voltage with the internal reference voltage.

  The voltage comparison mode can be configured through LPSYSREG_MODE_COMP*n* (*n* = 0 ~ 1).

- **Reference voltage configuration**

  The internal reference voltage range is 0 ~ (0.7 × VDDPST_6) V with a step of 0.1 × VDDPST_6 V, where VDDPST_6 is the supply voltage of the analog voltage comparator which is equal to the power supply voltage of the chip (usually 3.3 V). The internal reference voltage value can be configured through LPSYSREG_DREF_COMP*n* (n*n* = 0 ~ 1).

  The external reference voltage is accessed directly from the pad with an input range of 0 ~ 0.7 x VDDPST_6 V, same as the internal reference voltage, requiring no additional configuration.

- **Voltage comparison interrupt processing**

  The voltage comparison result, i.e., the COMP_OUT signal, can be either high or low. Whenever the output value changes, a corresponding interrupt signal is generated.

46.5 Event Task Matrix Feature

The analog voltage comparator on ESP32-P4 supports the Event Task Matrix (ETM) function, which allows analog voltage comparator ETM events to trigger any peripherals' ETM tasks. This section introduces the ETM events related to analog voltage comparator. For more information, please refer to Chapter 11 Event Task Matrix (ETM).

The analog voltage comparator can generate the following ETM events:

- GPIO_EVT_ZERO_DET_POS*n*: Indicates that the COMP_OUT signal changes from low level to high level, i.e., the main voltage changes from below to above the reference voltage.
- GPIO_EVT_ZERO_DET_NEG*n*: Indicates that the COMP_OUT signal changes from high level to low level, i.e., the main voltage changes from higher to lower than the reference voltage.

The analog voltage comparator does not support any ETM tasks.

46.6 Interrupts

ESP32-P4's analog voltage comparator can generate the GPIO_PAD_COMP_INT interrupt signal that will be sent to the Interrupt Matrix.

There are several internal interrupt sources from analog voltage comparator that can generate the above interrupt signal. The interrupt sources from analog voltage comparator are listed with their trigger conditions and the resulted interrupt signal(s) in Table 46.6-1.

Table 46.6-1. Analog Voltage Comparator's Internal Interrupt Sources

Internal Interrupt Source	Trigger Condition	Interrupt Signal
GPIO_COMP*n*_ALL_INT (*n* = 0 ~ 1)	Any COMP_OUT signal transition occurs	GPIO_PAD_COMP_INT
GPIO_COMP*n*_NEG_INT (*n* = 0 ~ 1)	The COMP_OUT signal changes from high level to low level	GPIO_PAD_COMP_INT
GPIO_COMP*n*_POS_INT (*n* = 0 ~ 1)	The COMP_OUT signal changes from low level to high level	GPIO_PAD_COMP_INT

Note:
For definitions of *interrupt*, *interrupt signal*, *interrupt source*, and their correlations, please refer to Chapter 10 *Interrupt Matrix* > Section 10.2 *Interrupt Terminology in ESP32-P4*.

Each interrupt source can be configured by a common set of registers that are described in Section *Interrupt Configuration Registers*. The specific registers can be found in Section 7.19.1 *HP GPIO Matrix Register Summary*.

46.7 Programming Procedures

The programming procedure for the analog voltage comparator is as follows:

Note:
For more information about the register fields mentioned in this section, refer to Chapter 6 System Registers [to be added later].

1. Enable the comparator by setting LPSYSREG_XPD_COMP*n* (*n* = 0 ~ 1) to 1.
2. Disable normal digital pad functions (including IE, OE, WPU, and WPD) for PAD1, and if using the external reference voltage, also for PAD0. For detailed configuration, see Chapter 7 GPIO Matrix and IO MUX.
3. Configure the comparison mode by setting LPSYSREG_MODE_COMP*n* (*n* = 0 ~ 1):
   - 0: Configures to compare the main voltage with the internal reference voltage;
   - 1: Configures to compare the main voltage with the external reference voltage.
4. Configure the internal reference voltage through LPSYSREG_DREF_COMP*n* (*n*n = 0 ~ 1), which offers a voltage range of 0 ~ (0.7 × VDDPST_6) V with a step of 0.1 × VDDPST_6 V.

Part IX

Appendix

This part contains the following information starting from the next page:

- *Related Documentation and Resources*
- *Glossary*
- *Programming Reserved Register Field*
- *Interrupt Configuration Registers*
- *Revision History*

Related Documentation and Resources

Related Documentation

- ESP32-P4 Series Datasheet – Specifications of the ESP32-P4 hardware.
- ESP32-P4 Hardware Design Guidelines – Guidelines on how to integrate the ESP32-P4 into your hardware product.
- *Certificates*
  https://espressif.com/en/support/documents/certificates
- *ESP32-P4 Product/Process Change Notifications (PCN)*
  https://espressif.com/en/support/documents/pcns?keys=ESP32-P4
- *ESP32-P4 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://espressif.com/en/support/documents/advisories?keys=ESP32-P4
- *Documentation Updates and Update Notification Subscription*
  https://espressif.com/en/support/download/documents

Developer Zone

- ESP-IDF Programming Guide for ESP32-P4 – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.
  https://github.com/espressif
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.
  https://espressif.com/en/support/download/sdks-demos

Products

- *ESP32-P4 Series SoCs* – Browse through all ESP32-P4 SoCs.
  https://espressif.com/en/products/socs?id=ESP32-P4
- *ESP32-P4 Series Modules* – Browse through all ESP32-P4-based modules.
  https://espressif.com/en/products/modules?id=ESP32-P4
- *ESP32-P4 Series DevKits* – Browse through all ESP32-P4-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32-P4
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector?language=en

Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.
  https://espressif.com/en/contact-us/sales-questions

Glossary

Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED Control PWM (Pulse Width Modulation)
MCPWM	Motor Control PWM (Pulse Width Modulation)
PCNT	Pulse Count Controller
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

Abbreviations Related to Registers

REG	**Register**.
SYSREG	**System registers** are a group of registers that control system reset, memory, clocks, software interrupts, power management, clock gating, etc.
ISO	**Isolation**. If a peripheral or other chip component is powered down, the pins, if any, to which its output signals are routed will go into a floating state. ISO registers isolate such pins and keep them at a certain determined value, so that the other non-powered-down peripherals/devices attached to these pins are not affected.

NMI **Non-maskable interrupt** is a hardware interrupt that cannot be disabled or ignored by the CPU instructions. Such interrupts exist to signal the occurrence of a critical error.

W1TS Abbreviation added to names of registers/fields to indicate that such register/field should be used to set a field in a corresponding register with a similar name. For example, the register `GPIO_ENABLE_W1TS_REG` should be used to set the corresponding fields in the register `GPIO_ENABLE_REG`.

W1TC Same as *W1TS*, but used to clear a field in a corresponding register.

Access Types for Registers

Sections *Register Summary* and *Register Description* in TRM chapters specify access types for registers and their fields.

Most frequently used access types and their combinations are as follows:

- RO
- WO
- WT
- R/W
- 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.

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

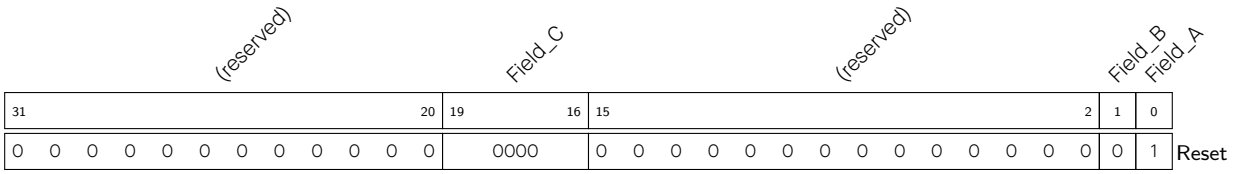
The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

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 46.1. Register X (Address)



Suppose you want to set Field_A, Field_B, and Field_C of Register X to 0x0, 0x1, and 0x2, you can:

- Use option 1 and fill in the reserved fields with the value you have just read. Suppose the register reads as 0x0000_0003. Then, you can modify the fields you want to configure, thus writing 0x0002_0002 to the register.
- Use option 2 and fill in the reserved fields with their defaults, thus writing 0x0002_0002 to the register.

Interrupt Configuration Registers

Generally, the peripherals' internal interrupt sources can be configured by the following common set of registers:

- **RAW** (Raw Interrupt Status) register: This register indicates the raw interrupt status. Each bit in the register represents a specific internal interrupt source. When an interrupt source triggers, its RAW bit is set to 1.
- **ENA** (Enable) register: This register is used to enable or disable the internal interrupt sources. Each bit in the ENA register corresponds to an internal interrupt source.

  By manipulating the ENA register, you can mask or unmask individual internal interrupt source as needed. When an internal interrupt source is masked (disabled), it will not generate an interrupt signal, but its value can still be read from the RAW register.
- **ST** (Status) register: This register reflects the status of enabled interrupt sources. Each bit in the ST register corresponds to a specific internal interrupt source. The ST bit being 1 means that both the corresponding RAW bit and ENA bit are 1, indicating that the interrupt source is triggered and not masked. The other combinations of the RAW bit and ENA bit will result in the ST bit being 0.

  The configuration of ENA/RAW/ST registers is shown in Table 46.7-4.
- **CLR** (Clear) register: The CLR register is responsible for clearing the internal interrupt sources. Writing 1 to the corresponding bit in the CLR register clears the interrupt source.

Table 46.7-4. Configuration of ENA/RAW/ST Registers

ENA Bit Value	RAW Bit Value	ST Bit Value
0	Ignored	0
1	0	0
	1	1

Revision History

Date	Version	Release notes
2023-	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 © 2024 Espressif Systems (Shanghai) Co., Ltd. All rights reserved.

www.espressif.com