# ESP32

## Technical Reference Manual Version 5.7

ESPRESSIF

www.espressif.com

## About This Document

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

## Navigation in This Document

Here are some tips on navigation through this extensive document:

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

# Release Status at a Glance

| No. | Chapter | Progress |
|---|---|---|
| | **Part I. Microprocessor and Master** | |
| 1 | ULP Coprocessor (ULP) | Published |
| 2 | DMA Controller (DMA) | Published |
| | **Part II. Memory Organization** | |
| 3 | System and Memory | Published |
| 4 | Memory Management and Protection Units (MMU, MPU) | Published |
| 5 | eFuse Controller (EFUSE) | Published |
| | **Part III. System Component** | |
| 6 | IO MUX and GPIO Matrix (GPIO, IO MUX) | Published |
| 7 | Reset and Clock | Published |
| 8 | Interrupt Matrix (INTERRUPT) | Published |
| 9 | Low-Power Management (RTC_CNTL) | Published |
| 10 | Timer Group (TIMG) | Published |
| 11 | Watchdog Timers (WDT) | Published |
| 12 | DPort Registers | Published |
| 13 | Process ID Controller (PID) | Published |
| | **Part IV. Cryptography/Security Component** | |
| 14 | AES Accelerator (AES) | Published |
| 15 | RSA Accelerator (RSA) | Published |
| 16 | SHA Accelerator (SHA) | Published |
| 17 | External Memory Encryption and Decryption (FLASH) | Published |
| 18 | Random Number Generator (RNG) | Published |
| | **Part V. Connectivity Interface** | |
| 19 | UART Controller (UART) | Published |
| 20 | SPI Controller (SPI) | Published |
| 21 | I2C Controller (I2C) | Published |
| 22 | I2S Controller (I2S) | Published |
| 23 | Pulse Count Controller (PCNT) | Published |
| 24 | Ethernet Media Access Controller (EMAC) | Published |
| 25 | Two-Wire Automotive Interface (TWAI) | Published |
| 26 | SDIO Slave Controller (SDIO) | Published |
| 27 | SD/MMC Host Controller (SDHOST) | Published |
| 28 | LED PWM Controller (LEDC) | Published |
| 29 | Motor Control PWM (MCPWM) | Published |
| 30 | Remote Control Peripheral (RMT) | Published |
| | **Part VI. Analog Signal Processing** | |
| 31 | On-Chip Sensors and Analog Signal Processing | 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_technical_reference_manual_en.pdf

# Contents


I Microprocessor and Master 28

1 ULP Coprocessor (ULP) 29
1.1 Introduction 29
1.2 Features 29
1.3 Functional Description 30
1.4 Instruction Set 30
1.4.1 ALU - Perform Arithmetic/Logic Operations 31
1.4.1.1 Operations Among Registers 31
1.4.1.2 Operations with Immediate Value 32
1.4.1.3 Operations with Stage Count Register 33
1.4.2 ST – Store Data in Memory 33
1.4.3 LD – Load Data from Memory 34
1.4.4 JUMP – Jump to an Absolute Address 35
1.4.5 JUMPR – Jump to a Relative Offset (Conditional upon R0) 35
1.4.6 JUMPS – Jump to a Relative Address (Conditional upon Stage Count Register) 36
1.4.7 HALT – End the Program 36
1.4.8 WAKE – Wake up the Chip 36
1.4.9 Sleep – Set the ULP Timer's Wake-up Period 37
1.4.10 WAIT – Wait for a Number of Cycles 37
1.4.11 ADC – Take Measurement with ADC 37
1.4.12 I2C_RD/I2C_WR – Read/Write I2C 38
1.4.13 REG_RD – Read from Peripheral Register 39
1.4.14 REG_WR – Write to Peripheral Register 39
1.5 ULP Program Execution 40
1.6 RTC_I2C Controller 41
1.6.1 Configuring RTC_I2C 42
1.6.2 Using RTC_I2C 42
1.6.2.1 I2C_RD - Read a Single Byte 42
1.6.2.2 I2C_WR - Write a Single Byte 43
1.6.2.3 Detecting Error Conditions 43
1.6.2.4 Connecting I2C Signals 44
1.7 Register Summary 45
1.7.1 SENS_ULP Address Space 45
1.7.2 RTC_I2C Address Space 45
1.8 Registers 47
1.8.1 SENS_ULP Address Space 47
1.8.2 RTC_I2C Address Space 49

2 DMA Controller (DMA) 56
2.1 Overview 56
2.2 Features 56

2.3 Functional Description 57
2.3.1 DMA Engine Architecture 57
2.3.2 Linked List 57
2.4 UART DMA (UDMA) 58
2.5 SPI DMA Interface 59
2.6 I2S DMA Interface 60

II Memory Organization 62

3 System and Memory 63
3.1 Introduction 63
3.2 Features 63
3.3 Functional Description 66
3.3.1 Address Mapping 66
3.3.2 Embedded Memory 66
3.3.2.1 Internal ROM 0 68
3.3.2.2 Internal ROM 1 68
3.3.2.3 Internal SRAM 0 68
3.3.2.4 Internal SRAM 1 69
3.3.2.5 Internal SRAM 2 69
3.3.2.6 DMA 70
3.3.2.7 RTC FAST Memory 70
3.3.2.8 RTC SLOW Memory 70
3.3.3 External Memory 70
3.3.4 Cache 71
3.3.5 Peripherals 72
3.3.5.1 Asymmetric PID Controller Peripheral 74
3.3.5.2 Non-Contiguous Peripheral Memory Ranges 74
3.3.5.3 Memory Speed 74

4 Memory Management and Protection Units (MMU, MPU) 75
4.1 Introduction 75
4.2 Features 75
4.3 Functional Description 75
4.3.1 PID Controller 75
4.3.2 MPU/MMU 76
4.3.2.1 Embedded Memory 76
4.3.2.2 External Memory 83
4.3.2.3 Peripheral 89

5 eFuse Controller (EFUSE) 91
5.1 Introduction 91
5.2 Features 91
5.3 Functional Description 91
5.3.1 Structure 91
5.3.1.1 System Parameter efuse_wr_disable 93

5.3.1.2 System Parameter efuse_rd_disable 93
5.3.1.3 System Parameter coding_scheme 94
5.3.1.4 BLK3_part_reserve 95
5.3.2 Programming of System Parameters 95
5.3.3 Software Reading of System Parameters 98
5.3.4 The Use of System Parameters by Hardware Modules 100
5.3.5 Interrupts 100
5.4 Register Summary 100
5.5 Registers 103

III System Component 114

6 IO MUX and GPIO Matrix (GPIO, IO MUX) 115
6.1 Overview 115
6.2 Peripheral Input via GPIO Matrix 117
6.2.1 Summary 117
6.2.2 Functional Description 117
6.2.3 Simple GPIO Input 118
6.3 Peripheral Output via GPIO Matrix 118
6.3.1 Summary 118
6.3.2 Functional Description 119
6.3.3 Simple GPIO Output 120
6.4 Direct I/O via IO MUX 120
6.4.1 Summary 120
6.4.2 Functional Description 120
6.5 RTC IO MUX for Low Power and Analog I/O 120
6.5.1 Summary 120
6.5.2 Analog Function Description 121
6.6 Light-sleep Mode Pin Functions 121
6.7 pin Hold Feature 121
6.8 I/O Pin Power Supplies 122
6.8.1 VDD_SDIO Power Domain 123
6.9 Peripheral Signal List 124
6.10 IO MUX Pin List 133
6.11 RTC_MUX Pin List 134
6.12 Register Summary 134
6.12.1 GPIO Matrix Register Summary 134
6.12.2 IO MUX Register Summary 136
6.12.3 RTC IO MUX Register Summary 137
6.13 Registers 138
6.13.1 GPIO Matrix Registers 138
6.13.2 IO MUX Registers 147
6.13.3 RTC IO MUX Registers 149

7 Reset and Clock 165
7.1 System Reset 165

7.1.1 Introduction 165
7.1.2 Reset Source 165
7.2 System Clock 166
7.2.1 Introduction 166
7.2.2 Clock Source 167
7.2.3 CPU Clock 168
7.2.4 Peripheral Clock 168
7.2.4.1 APB_CLK 169
7.2.4.2 REF_TICK 169
7.2.4.3 LEDC_SCLK Source 169
7.2.4.4 APLL_SCLK Source 170
7.2.4.5 PLL_F160M_CLK Source 170
7.2.4.6 Clock Source Considerations 170
7.2.5 Wi-Fi BT Clock 170
7.2.6 RTC Clock 170
7.2.7 Audio PLL 170
7.3 Register Summary 171
7.4 Registers 172

8 Interrupt Matrix (INTERRUPT) 174
8.1 Overview 174
8.2 Features 174
8.3 Functional Description 174
8.3.1 Peripheral Interrupt Source 175
8.3.2 CPU Interrupt 178
8.3.3 Allocate Peripheral Interrupt Sources to Peripheral Interrupt on CPU 178
8.3.4 CPU NMI Interrupt Mask 179
8.3.5 Query Current Interrupt Status of Peripheral Interrupt Source 179
8.4 Registers 179

9 Low-Power Management (RTC_CNTL) 180
9.1 Introduction 180
9.2 Features 180
9.3 Functional Description 181
9.3.1 Overview 181
9.3.2 Digital Core Voltage Regulator 181
9.3.3 Low-Power Voltage Regulator 182
9.3.4 Flash Voltage Regulator 183
9.3.5 Brownout Detector 184
9.3.6 RTC Module 184
9.3.7 Low-Power Clocks 186
9.3.8 Power-Gating Implementation 187
9.3.9 Predefined Power Modes 188
9.3.10 Wakeup Source 190
9.3.11 Reject Sleep 191
9.3.12 RTC Timer 192

9.3.13 RTC Boot 192
9.4 Register Summary 194
9.5 Registers 195

10 Timer Group (TIMG) 223
10.1 Introduction 223
10.2 Functional Description 223
10.2.1 16-bit Prescaler 223
10.2.2 64-bit Time-base Counter 223
10.2.3 Alarm Generation 224
10.2.4 MWDT 224
10.2.5 Interrupts 224
10.3 Register Summary 224
10.4 Registers 227

11 Watchdog Timers (WDT) 234
11.1 Introduction 234
11.2 Features 234
11.3 Functional Description 234
11.3.1 Clock 234
11.3.2 Operating Procedure 235
11.3.3 Write Protection 235
11.3.4 Flash Boot Protection 235
11.4 Registers 236

12 DPort Registers 237
12.1 Introduction 237
12.2 Features 237
12.3 Functional Description 237
12.3.1 System and Memory Register 237
12.3.2 Reset and Clock Registers 237
12.3.3 Interrupt Matrix Register 238
12.3.4 DMA Registers 238
12.3.5 MPU/MMU Registers 238
12.3.6 APP_CPU Controller Registers 238
12.3.7 Peripheral Clock Gating and Reset 238
12.4 Register Summary 239
12.5 Registers 245

13 Process ID Controller (PID) 271
13.1 Overview 271
13.2 Features 271
13.3 Functional Description 271
13.3.1 Interrupt Identification 271
13.3.2 Information Recording 272
13.3.3 Proactive Process Switching 273

13.4 Register Summary 275
13.5 Registers 275

IV Cryptography/Security Component 280

14 AES Accelerator (AES) 281
14.1 Introduction 281
14.2 Features 281
14.3 Functional Description 281
14.3.1 AES Algorithm Operations 281
14.3.2 Key, Plaintext and Ciphertext 281
14.3.3 Endianness 282
14.3.4 Encryption and Decryption Operations 284
14.3.5 Speed 284
14.4 Register Summary 284
14.5 Registers 285

15 RSA Accelerator (RSA) 287
15.1 Introduction 287
15.2 Features 287
15.3 Functional Description 287
15.3.1 Initialization 287
15.3.2 Large Number Modular Exponentiation 287
15.3.3 Large Number Modular Multiplication 289
15.3.4 Large Number Multiplication 289
15.4 Register Summary 290
15.5 Registers 291

16 SHA Accelerator (SHA) 293
16.1 Introduction 293
16.2 Features 293
16.3 Functional Description 293
16.3.1 Padding and Parsing the Message 293
16.3.2 Message Digest 293
16.3.3 Hash Operation 294
16.3.4 Speed 294
16.4 Register Summary 295
16.5 Registers 297

17 External Memory Encryption and Decryption (FLASH) 303
17.1 Overview 303
17.2 Features 303
17.3 Functional Description 304
17.3.1 Key Generator 304
17.3.2 Flash Encryption Block 305
17.3.3 Flash Decryption Block 305

17.4 Register Summary 306
17.5 Register 306

18 Random Number Generator (RNG) 308
18.1 Introduction 308
18.2 Feature 308
18.3 Functional Description 308
18.4 Programming Procedure 309
18.5 Register Summary 309
18.6 Register 309

V Connectivity Interface 310

19 UART Controller (UART) 311
19.1 Overview 311
19.2 UART Features 311
19.3 Functional Description 311
19.3.1 Introduction 311
19.3.2 UART Architecture 312
19.3.3 UART RAM 313
19.3.4 Baud Rate Detection 314
19.3.5 UART Data Frame 314
19.3.6 AT_CMD Character Structure 315
19.3.7 Flow Control 315
19.3.7.1 Hardware Flow Control 316
19.3.7.2 Software Flow Control 316
19.3.8 UART DMA 317
19.3.9 UART Interrupts 317
19.3.10 UHCI Interrupts 318
19.4 Register Summary 318
19.4.1 UART Register Summary 318
19.4.2 UHCI Register Summary 320
19.5 Registers 322
19.5.1 UART Registers 322
19.5.2 UHCI Registers 342

20 SPI Controller (SPI) 354
20.1 Overview 354
20.2 SPI Features 355
20.3 GP-SPI 355
20.3.1 GP-SPI Four-line Full-duplex Communication 356
20.3.2 GP-SPI Four-line Half-duplex Communication 356
20.3.3 GP-SPI Three-line Half-duplex Communication 357
20.3.4 GP-SPI Data Buffer 358
20.4 GP-SPI Clock Control 358
20.4.1 GP-SPI Clock Polarity (CPOL) and Clock Phase (CPHA) 358

20.4.2 GP-SPI Timing 360
20.5 Parallel QSPI 361
20.5.1 Communication Format of Parallel QSPI 361
20.6 GP-SPI Interrupt Hardware 362
20.6.1 SPI Interrupts 362
20.6.2 DMA Interrupts 362
20.7 Register Summary 363
20.8 Registers 366

21 I2C Controller (I2C) 389
21.1 Overview 389
21.2 Features 389
21.3 Functional Description 389
21.3.1 Introduction 389
21.3.2 Architecture 390
21.3.3 I2C Bus Timing 391
21.3.4 I2C cmd Structure 392
21.3.5 I2C Master Writes to Slave 393
21.3.6 Master Reads from Slave 397
21.3.7 Interrupts 399
21.4 Register Summary 401
21.5 Registers 403

22 I2S Controller (I2S) 415
22.1 Overview 415
22.2 Features 417
22.3 The Clock of I2S Module 417
22.4 I2S Mode 418
22.4.1 Supported Audio Standards 418
22.4.1.1 Philips Standard 419
22.4.1.2 MSB Alignment Standard 419
22.4.1.3 PCM Standard 419
22.4.2 Module Reset 420
22.4.3 FIFO Operation 420
22.4.4 Sending Data 420
22.4.5 Receiving Data 422
22.4.6 I2S Master/Slave Mode 424
22.4.7 I2S PDM 424
22.5 Camera-LCD Controller 426
22.5.1 LCD Master Transmitting Mode 426
22.5.2 Camera Slave Receiving Mode 427
22.5.3 ADC/DAC mode 428
22.6 I2S Interrupts 429
22.6.1 FIFO Interrupts 429
22.6.2 DMA Interrupts 430
22.7 Register Summary 430

22.8 Registers 432

23 Pulse Count Controller (PCNT) 450
23.1 Overview 450
23.2 Functional Description 450
23.2.1 Architecture 451
23.2.2 Counter Channel Inputs 451
23.2.3 Watchpoints 452
23.2.4 Examples 453
23.2.5 Interrupts 453
23.3 Register Summary 454
23.4 Registers 455

24 Ethernet Media Access Controller (EMAC) 461
24.1 Overview 461
24.2 EMAC_CORE 463
24.2.1 Transmit Operation 463
24.2.1.1 Transmit Flow Control 464
24.2.1.2 Retransmission During a Collision 464
24.2.2 Receive Operation 464
24.2.2.1 Reception Protocol 465
24.2.2.2 Receive Frame Controller 465
24.2.2.3 Receive Flow Control 465
24.2.2.4 Reception of Multiple Frames 466
24.2.2.5 Error Handling 466
24.2.2.6 Receive Status Word 466
24.3 MAC Interrupt Controller 467
24.4 MAC Address Filtering 467
24.4.1 Unicast Destination Address Filtering 467
24.4.2 Multicast Destination Address Filtering 467
24.4.3 Broadcast Address Filtering 467
24.4.4 Unicast Source Address Filtering 467
24.4.5 Inverse Filtering Operation 468
24.4.6 Good Transmitted Frames and Received Frames 469
24.5 EMAC_MTL (MAC Transaction Layer) 470
24.6 PHY Interface 470
24.6.1 MII (Media Independent Interface) 470
24.6.1.1 Interface Signals Between MII and PHY 470
24.6.1.2 MII Clock 471
24.6.2 RMII (Reduced Media-Independent Interface) 472
24.6.2.1 RMII Interface Signal Description 472
24.6.2.2 RMII Clock 473
24.6.3 Station Management Agent (SMA) Interface 473
24.6.4 RMII Timing 473
24.7 Ethernet DMA Features 474
24.8 Linked List Descriptors 475

24.8.1 Transmit Descriptors 475
24.8.2 Receive Descriptors 479
24.9 Register Summary 484
24.10 Registers 486

25 Two-Wire Automotive Interface (TWAI) 525
25.1 Overview 525
25.2 Features 525
25.3 Functional Protocol 526
25.3.1 TWAI Properties 526
25.3.2 TWAI Messages 527
25.3.2.1 Data Frames and Remote Frames 527
25.3.2.2 Error and Overload Frames 529
25.3.2.3 Interframe Space 531
25.3.3 TWAI Errors 531
25.3.3.1 Error Types 531
25.3.3.2 Error States 532
25.3.3.3 Error Counters 532
25.3.4 TWAI Bit Timing 533
25.3.4.1 Nominal Bit 533
25.3.4.2 Hard Synchronization and Resynchronization 534
25.4 Architectural Overview 534
25.4.1 Registers Block 535
25.4.2 Bit Stream Processor 536
25.4.3 Error Management Logic 536
25.4.4 Bit Timing Logic 536
25.4.5 Acceptance Filter 537
25.4.6 Receive FIFO 537
25.5 Functional Description 537
25.5.1 Modes 537
25.5.1.1 Reset Mode 537
25.5.1.2 Operation Mode 537
25.5.2 Bit Timing 538
25.5.3 Interrupt Management 538
25.5.3.1 Receive Interrupt (RXI) 539
25.5.3.2 Transmit Interrupt (TXI) 539
25.5.3.3 Error Warning Interrupt (EWI) 539
25.5.3.4 Data Overrun Interrupt (DOI) 540
25.5.3.5 Error Passive Interrupt (TXI) 540
25.5.3.6 Arbitration Lost Interrupt (ALI) 540
25.5.3.7 Bus Error Interrupt (BEI) 540
25.5.4 Transmit and Receive Buffers 540
25.5.4.1 Overview of Buffers 540
25.5.4.2 Frame Information 541
25.5.4.3 Frame Identifier 542
25.5.4.4 Frame Data 543

25.5.5 Receive FIFO and Data Overruns 543
25.5.6 Acceptance Filter 543
25.5.6.1 Single Filter Mode 544
25.5.6.2 Dual FIlter Mode 545
25.5.7 Error Management 545
25.5.7.1 Error Warning Limit 545
25.5.7.2 Error Passive 546
25.5.7.3 Bus-Off and Bus-Off Recovery 546
25.5.8 Error Code Capture 547
25.5.9 Arbitration Lost Capture 548
25.6 Register Summary 549
25.7 Registers 550

26 SDIO Slave Controller (SDIO) 563
26.1 Overview 563
26.2 Features 563
26.3 Functional Description 563
26.3.1 SDIO Slave Block Diagram 563
26.3.2 Sending and Receiving Data on SDIO Bus 564
26.3.3 Register Access 565
26.3.4 DMA 565
26.3.5 Packet-Sending/-Receiving Procedure 566
26.3.5.1 Sending Packets to SDIO Host 566
26.3.5.2 Receiving Packets from SDIO Host 568
26.3.6 SDIO Bus Timing 569
26.3.7 Interrupt 570
26.3.7.1 Host Interrupt 570
26.3.7.2 Slave Interrupt 570
26.4 Register Summary 571
26.5 SLC Registers 573
26.6 SLC Host Registers 581
26.7 HINF Registers 595

27 SD/MMC Host Controller (SDHOST) 597
27.1 Overview 597
27.2 Features 597
27.3 SD/MMC External Interface Signals 598
27.4 Functional Description 598
27.4.1 SD/MMC Host Controller Architecture 598
27.4.1.1 BIU 599
27.4.1.2 CIU 599
27.4.2 Command Path 599
27.4.3 Data Path 600
27.4.3.1 Data Transmit Operation 600
27.4.3.2 Data Receive Operation 601
27.5 Software Restrictions for Proper CIU Operation 601

27.6 RAM for Receiving and Sending Data 603
27.6.1 Transmit RAM Module 603
27.6.2 Receive RAM Module 603
27.7 Descriptor Chain 603
27.8 The Structure of a Linked List 603
27.9 Initialization 605
27.9.1 DMAC Initialization 606
27.9.2 DMAC Transmission Initialization 606
27.9.3 DMAC Reception Initialization 606
27.10 SD/MMC Timing 607
27.11 Clock Phase Selection 608
27.12 Interrupt 609
27.13 Register Summary 609
27.14 Registers 610

28 LED PWM Controller (LEDC) 629
28.1 Introduction 629
28.2 Functional Description 629
28.2.1 Architecture 629
28.2.2 Timers 630
28.2.3 Channels 631
28.2.4 Interrupts 633
28.3 Register Summary 633
28.4 Registers 637

29 Motor Control PWM (MCPWM) 647
29.1 Introduction 647
29.2 Features 647
29.3 Submodules 649
29.3.1 Overview 649
29.3.1.1 Prescaler Submodule 649
29.3.1.2 Timer Submodule 650
29.3.1.3 Operator Submodule 651
29.3.1.4 Fault Detection Submodule 652
29.3.1.5 Capture Submodule 653
29.3.2 PWM Timer Submodule 653
29.3.2.1 Configurations of the PWM Timer Submodule 653
29.3.2.2 PWM Timer's Working Modes and Timing Event Generation 653
29.3.2.3 PWM Timer Shadow Register 657
29.3.2.4 PWM Timer Synchronization and Phase Locking 658
29.3.3 PWM Operator Submodule 658
29.3.3.1 PWM Generator Submodule 658
29.3.3.2 Dead Time Generator Submodule 668
29.3.3.3 PWM Carrier Submodule 672
29.3.3.4 Fault Handler Submodule 675
29.3.4 Capture Submodule 676

29.3.4.1 Introduction 676
29.3.4.2 Capture Timer 677
29.3.4.3 Capture Channel 677
29.4 Register Summary 677
29.5 Registers 680

30 Remote Control Peripheral (RMT) 727
30.1 Introduction 727
30.2 Functional Description 727
30.2.1 RMT Architecture 727
30.2.2 RMT RAM 727
30.2.3 Clock 729
30.2.4 Transmitter 729
30.2.5 Receiver 729
30.2.6 Interrupts 729
30.3 Register Summary 730
30.4 Registers 731

VI Analog Signal Processing 736

31 On-Chip Sensors and Analog Signal Processing 737
31.1 Introduction 737
31.2 Capacitive Touch Sensor 737
31.2.1 Introduction 737
31.2.2 Features 737
31.2.3 Available GPIOs 738
31.2.4 Functional Description 738
31.2.5 Touch FSM 739
31.3 SAR ADC 740
31.3.1 Introduction 740
31.3.2 Features 741
31.3.3 Outline of Function 741
31.3.4 RTC SAR ADC Controllers 743
31.3.5 DIG SAR ADC Controllers 743
31.4 DAC 745
31.4.1 Introduction 745
31.4.2 Features 745
31.4.3 Structure 746
31.4.4 Cosine Waveform Generator 746
31.4.5 DMA support 747
31.5 Register Summary 747
31.5.1 Sensors 747
31.5.2 Advanced Peripheral Bus 748
31.5.3 RTC I/O 749
31.6 Registers 750
31.6.1 Sensors 750

31.6.2 Advanced Peripheral Bus 759
31.6.3 RTC I/O 763

VII Appendix 764

Related Documentation and Resources 765

Glossary 766
Abbreviations for Peripherals 766
Abbreviations Related to Registers 766
Access Types for Registers 768

Programming Reserved Register Field 770
Introduction 770
Programming Reserved Register Field 770

Interrupt Configuration Registers 771

Revision History 772

# List of Tables


1.4-1 ALU Operations Among Registers 32
1.4-2 ALU Operations with Immediate Value 33
1.4-3 ALU Operations with Stage Count Register 33
1.4-4 Input Signals Measured Using the ADC Instruction 38

3.3-1 Address Mapping 66
3.3-2 Embedded Memory Address Mapping 68
3.3-3 Module with DMA 70
3.3-4 External Memory Address Mapping 70
3.3-5 Cache memory mode 71
3.3-6 Peripheral Address Mapping 72

4.3-1 MPU and MMU Structure for Internal Memory 76
4.3-2 MPU for RTC FAST Memory 77
4.3-3 MPU for RTC SLOW Memory 77
4.3-4 Page Mode of MMU for the Remaining 128 KB of Internal SRAM0 and SRAM2 78
4.3-5 Page Boundaries for SRAM0 MMU 79
4.3-6 Page Boundaries for SRAM2 MMU 80
4.3-7 DPORT_DMMU_TABLE*n*_REG & DPORT_IMMU_TABLE*n*_REG 81
4.3-8 MPU for DMA 82
4.3-9 Virtual Address for External Memory 84
4.3-10 MMU Entry Numbers for PRO_CPU 84
4.3-11 MMU Entry Numbers for APP_CPU 84
4.3-12 MMU Entry Numbers for PRO_CPU (Special Mode) 85
4.3-13 MMU Entry Numbers for APP_CPU (Special Mode) 85
4.3-14 Virtual Address Mode for External SRAM 86
4.3-15 Virtual Address for External SRAM ( Normal Mode ) 87
4.3-16 Virtual Address for External SRAM ( Low-High Mode ) 87
4.3-17 Virtual Address for External SRAM (Even–Odd Mode) 87
4.3-18 MMU Entry Numbers for External RAM 88
4.3-19 MPU for Peripheral 89
4.3-20 DPORT_AHBLITE_MPU_TABLE_*X*_REG 90

5.3-1 System Parameters 91
5.3-2 BLOCK1/2/3 Encoding 94
5.3-3 Program Registers 95
5.3-4 Timing Configuration 97
5.3-5 Software Read Registers 98

6.6-1 IO_MUX Light-sleep Pin Function Registers 121
6.9-1 GPIO_Matrix 124
6.10-1 IO_MUX Pin Summary 133
6.11-1 RTC IO MUX Pin Summary 134
6.12-1 GPIO Matrix Register Summary 134
6.12-2 IO MUX Register Summary 136

6.12-3 RTC IO MUX Register Summary 137
6.13-1 Mapping of Bits to Pins 154

7.1-1 PRO_CPU and APP_CPU Reset Reason Values 166
7.2-1 CPU_CLK Source 168
7.2-2 CPU_CLK Derivation 168
7.2-3 Peripheral Clock Usage 168
7.2-4 APB_CLK 169
7.2-5 REF_TICK 169
7.2-6 LEDC_SCLK Derivation 170

8.3-1 PRO_CPU, APP_CPU Interrupt Configuration 176
8.3-2 CPU Interrupts 178

9.3-1 RTC Power Domains 188
9.3-2 Wake-up Source 191

13.3-1 Interrupt Vector Entry Address 272
13.3-2 Configuration of PIDCTRL_LEVEL_REG 272
13.3-3 Configuration of PIDCTRL_FROM_*n*_REG 273

14.3-1 Operation Mode 281
14.3-2 AES Text Endianness 282
14.3-3 AES-128 Key Endianness 283
14.3-4 AES-192 Key Endianness 283
14.3-5 AES-256 Key Endianness 283

20.1-1 Mapping Between SPI Bus Signals and Pin Function Signals 354
20.3-1 Command Definitions Supported by GP-SPI Slave in Half-duplex Mode 357
20.4-1 Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Master 359
20.4-2 Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Slave 359

21.3-1 SCL Frequency Configuration 391

22.1-1 I2S Signal Bus Description 416
22.4-1 Register Configuration 421
22.4-2 Send Channel Mode 421
22.4-3 Modes of Writing Received Data into FIFO and the Corresponding Register Configuration 423
22.4-4 The Register Configuration to Which the Four Modes Correspond 423
22.4-5 Upsampling Rate Configuration 425
22.4-6 Down-sampling Configuration 426

24.4-1 Destination Address Filtering 468
24.4-2 Source Address Filtering 469
24.6-1 Timing Parameters - Receiving Data 474
24.6-2 Timing Parameters – Transmitting Data 474
24.8-1 Transmit Descriptor 0 (TDES0) 475
24.8-2 Transmit Descriptor 1 (TDES1) 479
24.8-3 Transmit Descriptor 2 (TDES2) 479

24.8-4 Transmit Descriptor 3 (TDES3) 479
24.8-5 Receive Descriptor 0 (RDES0) 480
24.8-6 Receive Descriptor 1 (RDES1) 482
24.8-7 Receive Descriptor 2 (RDES2) 483
24.8-8 Receive Descriptor 3 (RDES3) 483
24.8-9 Receive Descriptor 4 (RDES4) 483

25.3-1 Data Frames and Remote Frames in SFF and EFF 528
25.3-2 Error Frame 530
25.3-3 Overload Frame 530
25.3-4 Interframe Space 531
25.3-5 Segments of a Nominal Bit Time 533
25.5-1 Bit Information of TWAI_CLOCK_DIVIDER_REG; TWAI Address 0x18 538
25.5-2 Bit Information of TWAI_BUS_TIMING_1_REG; TWAI Address 0x1c 538
25.5-3 Buffer Layout for Standard Frame Format and Extended Frame Format 540
25.5-4 TX/RX Frame Information (SFF/EFF); TWAI Address 0x40 541
25.5-5 TX/RX Identifier 1 (SFF); TWAI Address 0x44 542
25.5-6 TX/RX Identifier 2 (SFF); TWAI Address 0x48 542
25.5-7 TX/RX Identifier 1 (EFF); TWAI Address 0x44 542
25.5-8 TX/RX Identifier 2 (EFF); TWAI Address 0x48 542
25.5-9 TX/RX Identifier 3 (EFF); TWAI Address 0x4c 542
25.5-10 TX/RX Identifier 4 (EFF); TWAI Address 0x50 542
25.5-11 Bit Information of TWAI_ERR_CODE_CAP_REG; TWAI Address 0x30 547
25.5-12 Bit Information of Bits SEG.4 - SEG.0 548
25.5-13 Bit Information of TWAI_ARB LOST CAP_REG; TWAI Address 0x2c 549

27.3-1 SD/MMC Signal Description 598
27.8-1 DES0 604
27.8-2 DES1 605
27.8-3 DES2 605
27.8-4 DES3 605
27.10-1 SD/MMC Timing Requirements 607

28.2-1 Commonly-used Frequencies and Resolutions 631

29.3-1 Configuration Parameters of the Operator Submodule 651
29.3-2 Timing Events Used in PWM Generator 659
29.3-3 Timing Events Priority When PWM Timer Increments 660
29.3-4 Timing Events Priority when PWM Timer Decrements 660
29.3-5 Dead Time Generator Switches Control Registers 669
29.3-6 Typical Dead Time Generator Operating Modes 670

31.2-1 ESP32 Capacitive Sensing Touch Pads 738
31.3-1 Inputs of SAR ADC 742
31.3-2 ESP32 SAR ADC Controllers 743
31.3-3 Fields of the Pattern Table Register 744
31.3-4 Fields of Type I DMA Data Format 745
31.3-5 Fields of Type II DMA Data Format 745

31.6-4 Configuration of ENA/RAW/ST Registers 771

# List of Figures


1.2-1 ULP Coprocessor Diagram 30
1.4-1 The ULP Coprocessor Instruction Format 31
1.4-2 Instruction Type — ALU for Operations Among Registers 31
1.4-3 Instruction Type — ALU for Operations with Immediate Value 32
1.4-4 Instruction Type — ALU for Operations with Stage Count Register 33
1.4-5 Instruction Type — ST 33
1.4-6 Instruction Type — LD 34
1.4-7 Instruction Type — JUMP 35
1.4-8 Instruction Type — JUMPR 35
1.4-9 Instruction Type — JUMP 36
1.4-10 Instruction Type — HALT 36
1.4-11 Instruction Type — WAKE 36
1.4-12 Instruction Type — SLEEP 37
1.4-13 Instruction Type — WAIT 37
1.4-14 Instruction Type — ADC 37
1.4-15 Instruction Type — I2C 38
1.4-16 Instruction Type — REG_RD 39
1.4-17 Instruction Type — REG_WR 39
1.5-1 Control of ULP Program Execution 40
1.5-2 Sample of a ULP Operation Sequence 41
1.6-1 I2C Read Operation 43
1.6-2 I2C Write Operation 43

2.3-1 DMA Engine Architecture 57
2.3-2 Linked List Structure 57
2.4-1 Data Transfer in UDMA Mode 58
2.5-1 SPI DMA 59

3.2-1 System Structure 64
3.2-2 System Address Mapping 65
3.3-1 Cache Block Diagram 71

4.3-1 MMU Access Example 78

6.1-1 IO MUX, RTC IO MUX and GPIO Matrix Overview 115
6.1-2 Internal Structure of a Pad 116
6.2-1 Peripheral Input via IO MUX, GPIO Matrix 117
6.3-1 Output via GPIO Matrix 119
6.8-1 ESP32 I/O Pin Power Sources (QFN 6*6, Top View) 122
6.8-2 ESP32 I/O Pin Power Sources (QFN 5*5, Top View) 123

7.1-1 System Reset 165
7.2-1 System Clock 167

8.2-1 Interrupt Matrix Structure 174

9.2-1 ESP32 Power Control 181
9.3-1 Digital Core Voltage Regulator 182
9.3-2 Low-Power Voltage Regulator 183
9.3-3 Flash Voltage Regulator 184
9.3-4 Brownout Detector 184
9.3-5 RTC Structure 186
9.3-6 RTC Low-Power Clocks 187
9.3-7 RTC States 187
9.3-8 Power Modes 190
9.3-9 ESP32 Boot Flow 193

13.3-1 Interrupt Nesting 274

17.3-1 Flash Encryption/Decryption Module Architecture 304

18.3-1 Noise Source 308

19.3-1 UART Basic Structure 312
19.3-2 UART Shared RAM 313
19.3-3 UART Data Frame Structure 314
19.3-4 AT_CMD Character Format 315
19.3-5 Hardware Flow Control 316

20.1-1 SPI Architecture 354
20.3-1 SPI Master and Slave Full-duplex/Half-duplex Communication 356
20.3-2 SPI Data Buffer 358
20.4-1 GP-SPI Slave Data Output 361
20.5-1 Parallel QSPI 361
20.5-2 Communication Format of Parallel QSPI 362

21.3-1 I2C Master Architecture 390
21.3-2 I2C Slave Architecture 390
21.3-3 I2C Sequence Chart 391
21.3-4 Structure of The I2C Command Register 392
21.3-5 I2C Master Writes to Slave with 7-bit Address 393
21.3-6 I2C Master Writes to Slave with 10-bit Address 394
21.3-7 I2C Master Writes to addrM in RAM of Slave with 7-bit Address 395
21.3-8 Master Writes to Slave with 7-bit Address in Three Segments 396
21.3-9 Master Reads from Slave with 7-bit Address 397
21.3-10 Master Reads from Slave with 10-bit Address 398
21.3-11 Master Reads N Bytes of Data from addrM in Slave with 7-bit Address 398
21.3-12 Master Reads from Slave with 7-bit Address in Three Segments 399

22.1-1 I2S System Block Diagram 415
22.3-1 I2S Clock 418
22.4-1 Philips Standard 419
22.4-2 MSB Alignment Standard 419
22.4-3 PCM Standard 420

22.4-4 Tx FIFO Data Mode 421
22.4-5 The First Stage of Receiving Data 422
22.4-6 Modes of Writing Received Data into FIFO 423
22.4-7 PDM Transmitting Module 424
22.4-8 PDM Sends Signal 425
22.4-9 PDM Receives Signal 425
22.4-10 PDM Receive Module 426
22.5-1 LCD Master Transmitting Mode 426
22.5-2 LCD Master Transmitting Data Frame, Form 1 427
22.5-3 LCD Master Transmitting Data Frame, Form 2 427
22.5-4 Camera Slave Receiving Mode 427
22.5-5 ADC Interface of I2S0 428
22.5-6 DAC Interface of I2S 428
22.5-7 Data Input by I2S DAC Interface 429

23.2-1 PULSE_CNT Architecture 451
23.2-2 PULSE_CNT Upcounting Diagram 453
23.2-3 PULSE_CNT Downcounting Diagram 453

24.1-1 Ethernet MAC Functionality Overview 461
24.1-2 Ethernet Block Diagram 463
24.6-1 MII Interface 470
24.6-2 MII Clock 472
24.6-3 RMII Interface 472
24.6-4 RMII Clock 473
24.6-5 RMII Timing - Receiving Data 474
24.6-6 RMII Timing – Transmitting Data 474
24.8-1 Transmit Descriptor 475
24.8-2 Receive Descriptor 480

25.3-1 The bit fields of Data Frames and Remote Frames 528
25.3-2 Various Fields of an Error Frame 529
25.3-3 The Bit Fields of an Overload Frame 530
25.3-4 The Fields within an Interframe Space 531
25.3-5 Layout of a Bit 535
25.4-1 TWAI Overview Diagram 535
25.5-1 Acceptance Filter 544
25.5-2 Single Filter Mode 544
25.5-3 Dual Filter Mode 546
25.5-4 Error State Transition 547
25.5-5 Positions of Arbitration Lost Bits 549

26.3-1 SDIO Slave Block Diagram 564
26.3-2 SDIO Bus Packet Transmission 564
26.3-3 CMD53 Content 565
26.3-4 SDIO Slave DMA Linked List Structure 565
26.3-5 SDIO Slave Linked List 566

26.3-6 Packet Sending Procedure (Initiated by Slave) 567
26.3-7 Packet Receiving Procedure (Initiated by Host) 568
26.3-8 Loading Receiving Buffer 569
26.3-9 Sampling Timing Diagram 569
26.3-10 Output Timing Diagram 570

27.2-1 SD/MMC Controller Topology 597
27.3-1 SD/MMC Controller External Interface Signals 598
27.4-1 SDIO Host Block Diagram 599
27.4-2 Command Path State Machine 600
27.4-3 Data Transmit State Machine 601
27.4-4 Data Receive State Machine 601
27.7-1 Descriptor Chain 603
27.8-1 The Structure of a Linked List 604
27.10-1 SD/MMC Timing in HS Mode 607
27.11-1 Clock Phase Selection 608

28.2-1 LED_PWM Architecture 629
28.2-2 LED_PWM High-speed Channel Diagram 630
28.2-3 LED_PWM Divider 630
28.2-4 LED PWM Output Signal Diagram 632
28.2-5 Output Signal Diagram of Fading Duty Cycle 632

29.2-1 MCPWM Module Overview 648
29.3-1 Prescaler Submodule 649
29.3-2 Timer Submodule 650
29.3-3 Operator Submodule 651
29.3-4 Fault Detection Submodule 652
29.3-5 Capture Submodule 653
29.3-6 Count-Up Mode Waveform 654
29.3-7 Count-Down Mode Waveforms 654
29.3-8 Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event 655
29.3-9 Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event 655
29.3-10 UTEP and UTEZ Generation in Count-Up Mode 656
29.3-11 DTEP and DTEZ Generation in Count-Down Mode 656
29.3-12 DTEP and UTEZ Generation in Count-Up-Down Mode 657
29.3-13 Submodules Inside the PWM Operator 658
29.3-14 Symmetrical Waveform in Count-Up-Down Mode 662
29.3-15 Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 663
29.3-16 Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A 664
29.3-17 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High 665
29.3-18 Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary 666
29.3-19 Example of an NCI Software-Force Event on PWM*x*A 667
29.3-20 Example of a CNTU Software-Force Event on PWM*x*B 668

29.3-21 Options for Setting up the Dead Time Generator Submodule 669
29.3-22 Active High Complementary (AHC) Dead Time Waveforms 671
29.3-23 Active Low Complementary (ALC) Dead Time Waveforms 671
29.3-24 Active High (AH) Dead Time Waveforms 672
29.3-25 Active Low (AL) Dead Time Waveforms 672
29.3-26 Example of Waveforms Showing PWM Carrier Action 673
29.3-27 Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule 674
29.3-28 Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule 675

30.2-1 RMT Architecture 728
30.2-2 Data Structure 728

31.2-1 Touch Sensor 737
31.2-2 Touch Sensor Structure 739
31.2-3 Touch Sensor Operating Flow 739
31.2-4 Touch FSM Structure 740
31.3-1 SAR ADC Depiction 741
31.3-2 SAR ADC Outline of Function 742
31.3-3 RTC SAR ADC Outline of Function 743
31.3-4 Diagram of DIG SAR ADC Controllers 744
31.4-1 Diagram of DAC Function 746
31.4-2 Cosine Waveform (CW) Generator 747

# Part I

# Microprocessor and Master

This part covers the essential processing elements of the system, diving into the microprocessor architecture of the ultra-low-power processor. Details include controllers for Direct Memory Access (DMA).

# Chapter 1

# ULP Coprocessor (ULP)

## 1.1 Introduction

The ULP coprocessor is an ultra-low-power processor that remains powered on during the Deep-sleep mode of the main SoC. Hence, the developer can store in the RTC memory a program for the ULP coprocessor to access peripheral devices, internal sensors and RTC registers during deep sleep. This is useful for designing applications where the CPU needs to be woken up by an external event, or timer, or a combination of these, while maintaining minimal power consumption.

## 1.2 Features

- Contains up to 8 KB of SRAM for instructions and data
- Uses RTC_FAST_CLK, which is 8 MHz
- Works both in normal and deep sleep
- Is able to wake up the digital core or send an interrupt to the CPU
- Can access peripheral devices, internal sensors and RTC registers
- Contains four 16-bit general-purpose registers (R0, R1, R2, R3) for manipulating data and accessing memory
- Includes one 8-bit Stage_cnt register which can be manipulated by ALU and used in JUMP instructions

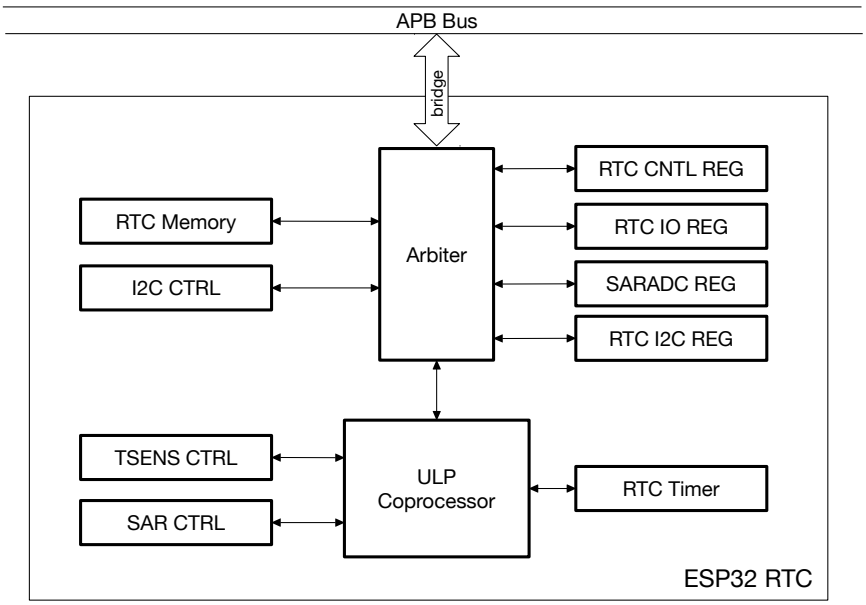



Figure 1.2-1. ULP Coprocessor Diagram

## 1.3 Functional Description

The ULP coprocessor is a programmable FSM (Finite State Machine) that can work during deep sleep. Like general-purpose CPUs, ULP coprocessor also has some instructions which can be useful for a relatively complex logic, and also some special commands for RTC controllers/peripherals. The 8 KB of SRAM RTC slow memory can be accessed by both the ULP coprocessor and the CPU; hence, it is usually used to store instructions and share data between the ULP coprocessor and the CPU.

The ULP coprocessor can be started by software or a periodically-triggered timer. The operation of the ULP coprocessor is ended by executing the HALT instruction. Meanwhile, it can access almost every module in RTC domain, either through built-in instructions or RTC registers. In many cases the ULP coprocessor can be a good supplement to, or replacement of, the CPU, especially for power-sensitive applications. Figure 1.2-1 shows the overall layout of a ULP coprocessor.

## 1.4 Instruction Set

The ULP coprocessor provides the following instructions:

- Perform arithmetic and logic operations - ALU
- Load and store data - LD, ST, REG_RD and REG_WR
- Jump to a certain address - JUMP
- Manage program execution - WAIT/HALT
- Control sleep period of ULP coprocessor - SLEEP

- Wake up/communicate with SoC - WAKE
- Take measurements - ADC
- Communicate using I2C - I2C_RD/I2C_WR

The ULP coprocessor's instruction format is shown in Figure 1.4-1.

| 31 28 | 27 0 |
|---|---|
| *OpCode* | *Operands* |

**Figure 1.4-1. The ULP Coprocessor Instruction Format**

An instruction, which has one *OpCode*, can perform various different operations, depending on the setting of *Operands* bits. A good example is the ALU instruction, which is able to perform 10 arithmetic and logic operations; or the JUMP instruction, which may be conditional or unconditional, absolute or relative.

Each instruction has a fixed width of 32 bits. A series of instructions can make a program be executed by the ULP coprocessor. The execution flow inside the program uses 32-bit addressing. The program is stored in a dedicated region called Slow Memory (RTC_SLOW_MEM), which is visible to the main CPUs as one that has an address range of 0x5000_0000 to 0x5000_1FFF (8 KB).

The *OpCode* in this chapter is represented by 4'd*x*, where 4 stands for 4-bit width, 'd is a decimal symbol, *x* stands for the value of *OpCode* (*x*: 0 ~ 15).

## 1.4.1 ALU - Perform Arithmetic/Logic Operations

The ALU (Arithmetic and Logic Unit) performs arithmetic and logic operations on values stored in ULP coprocessor registers, and on immediate values stored in the instruction itself.
The following operations are supported:

- Arithmetic: ADD and SUB
- Logic: bitwise logical AND and bitwise logical OR
- Bit shifting: LSH and RSH
- Moving data to register: MOVE
- Stage count register manipulation: STAGE_RST, STAGE_INC and STAGE_DEC

The ALU instruction, which has one *OpCode*, can perform various different arithmetic and logic operations, depending on the setting of the instruction's bits [27:21] accordingly.

### 1.4.1.1 Operations Among Registers

| 31 28 | 27 25 | 24 21 | 5 4 | 3 2 | 1 0 |
|---|---|---|---|---|---|
| 4'd7 | 3'b0 | ALU_sel | Rsrc2 | Rsrc1 | Rdst |

**Figure 1.4-2. Instruction Type — ALU for Operations Among Registers**

When bits [27:25] of the instruction in Figure 1.4-2 are set to 3'b0, ALU performs operations, using the ULP coprocessor register R[0-3]. The types of operations depend on the setting of the instruction's bits [24:21] presented in Table 1.4-1.

| **Operand** | **Description** - see Figure 1.4-2 |
|---|---|
| *ALU_sel* | Type of ALU operation |
| *Rdst* | Register R[0-3], destination |
| *Rsrc1* | Register R[0-3], source |
| *Rsrc2* | Register R[0-3], source |

| ALU_sel | Instruction | Operation | Description |
|---|---|---|---|
| 0 | ADD | *Rdst = Rsrc1 + Rsrc2* | Add to register |
| 1 | SUB | *Rdst = Rsrc1 - Rsrc2* | Subtract from register |
| 2 | AND | *Rdst = Rsrc1 & Rsrc2* | Bitwise logical AND of two operands |
| 3 | OR | *Rdst = Rsrc1 \| Rsrc2* | Bitwise logical OR of two operands |
| 4 | MOVE | *Rdst = Rsrc1* | Move to register |
| 5 | LSH | *Rdst = Rsrc1 << Rsrc2* | Bit shifting Left |
| 6 | RSH | *Rdst = Rsrc1 >> Rsrc2* | Bit shifting Right |

**Table 1.4-1. ALU Operations Among Registers**

**Note:**

- ADD/SUB operations can be used to set/clear the overflow flag in ALU.
- All ALU operations can be used to set/clear the zero flag in ALU.

### 1.4.1.2 Operations with Immediate Value

| 31–28 | 27–25 | 24–21 | | 19–4 | 3–2 | 1–0 |
|---|---|---|---|---|---|---|
| 4'd7 | 3'b1 | ALU_sel | | Imm | Rsrc1 | Rdst |

**Figure 1.4-3. Instruction Type — ALU for Operations with Immediate Value**

When bits [27:25] of the instruction in Figure 1.4-3 are set to 3'b1, ALU performs operations, using register R[0-3] and the immediate value stored in [19:4]. The types of operations depend on the setting of the instruction's bits [24:21] presented in Table 1.4-2.

| **Operand** | **Description** - see Figure 1.4-3 |
|---|---|
| *ALU_sel* | Type of ALU operation |
| *Rdst* | Register R[0-3], destination |
| *Rsrc1* | Register R[0-3], source |
| *Imm* | 16-bit signed value |

| ALU_sel | Instruction | Operation | Description |
|---|---|---|---|
| 0 | ADD | *Rdst = Rsrc1 + Imm* | Add to register |
| 1 | SUB | *Rdst = Rsrc1 - Imm* | Subtract from register |
| 2 | AND | *Rdst = Rsrc1 & Imm* | Bitwise logical AND of two operands |
| 3 | OR | *Rdst = Rsrc1 \| Imm* | Bitwise logical OR of two operands |
| 4 | MOVE | *Rdst = Imm* | Move to register |
| 5 | LSH | *Rdst = Rsrc1 << Imm* | Bit shifting left |
| 6 | RSH | *Rdst = Rsrc1 >> Imm* | Bit shifting right |

Table 1.4-2. ALU Operations with Immediate Value

**Note:**

- ADD/SUB operations can be used to set/clear the overflow flag in ALU.
- All ALU operations can be used to set/clear the zero flag in ALU.

#### 1.4.1.3 Operations with Stage Count Register

| 31 – 28 | 27 – 25 | 24 – 21 | | 11 – 4 | |
|---|---|---|---|---|---|
| 4'd7 | 3'b2 | ALU_sel | | Imm | |

Figure 1.4-4. Instruction Type — ALU for Operations with Stage Count Register

ALU is also able to increment/decrement by a given value, or reset the 8-bit register Stage_cnt. To do so, bits [27:25] of instruction in Figure 1.4-4 should be set to 3'b2. The type of operation depends on the setting of the instruction's bits [24:21] presented in Table 1.4-3. The Stage_cnt is a separate register and is not a part of the instruction in Figure 1.4-4.

| Operand | Description - see Figure 1.4-4 |
|---|---|
| *ALU_sel* | Type of ALU operation |
| *Stage_cnt* | Stage count register, a separate register [7:0] used to store variables, such as loop index |
| *Imm* | 8-bit value |

| ALU_sel | Instruction | Operation | Description |
|---|---|---|---|
| 0 | STAGE_INC | *Stage_cnt = Stage_cnt + Imm* | Increment stage count register |
| 1 | STAGE_DEC | *Stage_cnt = Stage_cnt - Imm* | Decrement stage count register |
| 2 | STAGE_RST | *Stage_cnt = 0* | Reset stage count register |

Table 1.4-3. ALU Operations with Stage Count Register

### 1.4.2 ST – Store Data in Memory

| 31 – 28 | 27 – 25 | 20 | 10 | 3 – 2 | 1 – 0 |
|---|---|---|---|---|---|
| 4'd6 | 3'b100 | 4'b0 | Offset | 6'b0 | Rdst | Rsrc |

Figure 1.4-5. Instruction Type — ST

| Operand | Description - see Figure 1.4-5 |
|---|---|
| *Offset* | 10-bit signed value, offset expressed in 32-bit words |
| *Rsrc* | Register R[0-3], 16-bit value to store |
| *Rdst* | Register R[0-3], address of the destination, expressed in 32-bit words |

**Description**

The instruction stores the 16-bit value of *Rsrc* in the lower half-word of memory with address *Rdst* + *Offset*. The upper half-word is written with the current program counter (PC) (expressed in words and shifted to the left by 5 bits) OR'd with Rdst (0..3):

Mem [ Rdst + Offset ]{31:0} = {PC[10:0], 3'b0, Rdst, Rsrc[15:0]}

The application can use the higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

**Note:**

- This instruction can only access 32-bit memory words.
- Data from *Rsrc* is always stored in the lower 16 bits of a memory word. Differently put, it is not possible to store *Rsrc* in the upper 16 bits of memory.
- The "Mem" written is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPUs.

### 1.4.3 LD – Load Data from Memory

| 31 – 28 | | 20 – 10 | | 3 – 2 | 1 – 0 |
|---|---|---|---|---|---|
| 4'd13 | | Offset | | Rsrc | Rdst |

**Figure 1.4-6. Instruction Type — LD**

| Operand | Description - see Figure 1.4-6 |
|---|---|
| *Offset* | 10-bit signed value, offset expressed in 32-bit words |
| *Rsrc* | Register R[0-3], address of destination memory, expressed in 32-bit words |
| *Rdst* | Register R[0-3], destination |

**Description**

The instruction loads the lower 16-bit half-word from memory with address *Rsrc* + *offset* into the destination register *Rdst*:

Rdst[15:0] = Mem[ Rsrc + Offset ][15:0]

**Note:**

- This instruction can only access 32-bit memory words.
- In any case, it is always the lower 16 bits of a memory word that are loaded. Differently put, it is not possible to read the upper 16 bits.
- The "Mem" loaded is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPUs.

### 1.4.4 JUMP – Jump to an Absolute Address

| 31 – 28 | 27 – 25 | 24 – 22 | 21 | 12 – 2 | 1 – 0 |
|---|---|---|---|---|---|
| 4'd8 | 3'b0 | Type | Sel | ImmAddr | Rdst |

**Figure 1.4-7. Instruction Type — JUMP**

| **Operand** | **Description** - see Figure 1.4-7 |
|---|---|
| *Rdst* | Register R[0-3], address to jump to |
| *ImmAddr* | 11-bit address, expressed in 32-bit words |
| *Sel* | Selects the address to jump to:<br>0 - jump to the address contained in *ImmAddr*<br>1 - jump to the address contained in *Rdst* |
| *Type* | Jump type:<br>0 - make an unconditional jump<br>1 - jump only if the last ALU operation has set the zero flag<br>2 - jump only if the last ALU operation has set the overflow flag |

**Description**

The instruction prompts a jump to the specified address. The jump can be either unconditional or based on the ALU flag.

**Note:**

All jump addresses are expressed in 32-bit words.

### 1.4.5 JUMPR – Jump to a Relative Offset (Conditional upon R0)

| 31 – 28 | 27 – 25 | 24 – 17 | 16 | 15 – 0 |
|---|---|---|---|---|
| 4'd8 | 3'b1 | Step | Cond | Threshold |

**Figure 1.4-8. Instruction Type — JUMPR**

| **Operand** | **Description** - see Figure 1.4-8 |
|---|---|
| *Step* | Relative shift from current position, expressed in 32-bit words:<br>if Step[7] = 0 then PC = PC + Step[6:0]<br>if Step[7] = 1 then PC = PC - Step[6:0] |
| *Threshold* | Threshold value for condition (see *Cond* below) to jump |
| *Cond* | Condition to jump:<br>0 - jump if R0 < *Threshold*<br>1 - jump if R0 >= *Threshold* |

**Description**

The instruction prompts a jump to a relative address, if the above-mentioned condition is true. The condition itself is the result of comparing the R0 register value and the *Threshold* value.

**Note:**

All jump addresses are expressed in 32-bit words.

### 1.4.6 JUMPS – Jump to a Relative Address (Conditional upon Stage Count Register)

| 31–28 | 27–25 | 24–17 | 16–15 | | 7–0 |
|---|---|---|---|---|---|
| 4'd8 | 3'b2 | Step | Cond | | Threshold |

Figure 1.4-9. Instruction Type — JUMP

| Operand | Description - see Figure 1.4-9 |
|---|---|
| *Step* | Relative shift from current position, expressed in 32-bit words:<br>if Step[7] = 0, then PC = PC + Step[6:0]<br>if Step[7] = 1, then PC = PC - Step[6:0] |
| *Threshold* | Threshold value for condition (see *Cond* below) to jump |
| *Cond* | Condition of jump:<br>1X - jump if *Stage_cnt* <= *Threshold*<br>00 - jump if *Stage_cnt* < *Threshold*<br>01 - jump if *Stage_cnt* >= *Threshold* |

**Note:**

- A description of how to set the stage count register is provided in section 1.4.1.3.
- All jump addresses are expressed in 32-bit words.

**Description**

The instruction prompts a jump to a relative address if the above-mentioned condition is true. The condition itself is the result of comparing the value of *Stage_cnt* (stage count register) and the *Threshold* value.

### 1.4.7 HALT – End the Program

| 31–28 | 0 |
|---|---|
| 4'd11 | |

Figure 1.4-10. Instruction Type — HALT

**Description**

The instruction ends the operation of the processor and puts it into power-down mode.

**Note:**

After executing this instruction, the ULP coprocessor timer gets started.

### 1.4.8 WAKE – Wake up the Chip

| 31–28 | 27–25 | | 0 |
|---|---|---|---|
| 4'd9 | 3'b0 | | 1'b1 |

Figure 1.4-11. Instruction Type — WAKE

**Description**

This instruction sends an interrupt from the ULP coprocessor to the RTC controller.

- If the SoC is in Deep-sleep mode, and the ULP wake-up is enabled, the above-mentioned interrupt will wake up the SoC.
- If the SoC is not in Deep-sleep mode, and the ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in register RTC_CNTL_INT_ENA_REG, a RTC interrupt will be triggered.

### 1.4.9 Sleep – Set the ULP Timer's Wake-up Period

| 31 – 28 | 27 – 25 | | 2 – 0 |
|---|---|---|---|
| 4'd9 | 3'b1 | | sleep_reg |

**Figure 1.4-12. Instruction Type — SLEEP**

| Operand | Description - see Figure 1.4-12 |
|---|---|
| *sleep_reg* | Selects one of five SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) as the wake-up period of the ULP coprocessor |

**Description**

The instruction selects which one of the SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) register values is to be used by the ULP timer as the wake-up period. By default, the value of SENS_ULP_CP_SLEEP_CYC0_REG is used.

### 1.4.10 WAIT – Wait for a Number of Cycles

| 31 – 28 | | 15 – 0 |
|---|---|---|
| 4'd4 | | Cycles |

**Figure 1.4-13. Instruction Type — WAIT**

| Operand | Description - see Figure 1.4-13 |
|---|---|
| *Cycles* | the number of cycles to wait between sleeps |

**Description**

The instruction will delay the ULP coprocessor from getting into sleep for a certain number of *Cycles*.

### 1.4.11 ADC – Take Measurement with ADC

| 31 – 28 | | 6 | 5 – 2 | 1 – 0 |
|---|---|---|---|---|
| 4'd5 | | Sel | Sar Mux | Rdst |

**Figure 1.4-14. Instruction Type — ADC**

| Operand | Description - see Figure 1.4-14 |
|---|---|
| *Rdst* | Destination Register R[0-3], results will be stored in this register. |
| *Sel* | Selected ADC: 0 = SAR ADC1, 1 = SAR ADC2, see Table 1.4-4. |
| *Sar Mux* | SARADC Pad [Sar_Mux - 1] is enabled, see Table 1.4-4. |

Table 1.4-4. Input Signals Measured Using the ADC Instruction

| Pad Name/Signal/GPIO | *Sar_Mux* | Processed by */Sel* |
|---|---|---|
| SENSOR_VP (GPIO36) | 1 | SAR ADC1/*Sel* = 0 |
| SENSOR_CAPP (GPIO37) | 2 | |
| SENSOR_CAPN (GPIO38) | 3 | |
| SENSOR_VN (GPIO39) | 4 | |
| 32K_XP (GPIO33) | 5 | |
| 32K_XN (GPIO32) | 6 | |
| VDET_1 (GPIO34) | 7 | |
| VDET_2 (GPIO35) | 8 | |
| GPIO4 | 1 | SAR ADC2/*Sel* = 1 |
| GPIO0 | 2 | |
| GPIO2 | 3 | |
| MTDO (GPIO15) | 4 | |
| MTCK (GPIO13) | 5 | |
| MTDI (GPIO12) | 6 | |
| MTMS (GPIO14) | 7 | |
| GPIO27 | 8 | |
| GPIO25 | 9 | |
| GPIO26 | 10 | |

**Description**

The instruction prompts the taking of measurements with the use of ADC. Pads/signals available for ADC measurement are provided in Table 1.4-4.

## 1.4.12 I2C_RD/I2C_WR – Read/Write I2C

| 31 – 28 | 27 | | 25 – 22 | 21 – 19 | 18 – 16 | 15 – 8 | 7 – 0 |
|---|---|---|---|---|---|---|---|
| 4'd3 | R/W | | I2C Sel | High | Low | Data | Sub-addr |

Figure 1.4-15. Instruction Type — I2C

| Operand | Description - see Figure 1.4-15 |
|---|---|
| *Sub-addr* | Slave register address |
| *Data* | Data to write in I2C_WR operation (not used in I2C_RD operation) |
| *Low* | High part of bit mask |
| *High* | Low part of bit mask |
| *I2C Sel* | Select register *n* of SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7), which contains the I2C slave address. |
| *R/W* | I2C communication direction:<br>1 - I2C write<br>0 - I2C read |

**Description**

Communicate (read/write) with external I2C slave devices. Details on using the RTC I2C peripheral are provided in section 1.6.

**Note:**

When working in master mode, RTC_I2C samples the SDA input on the negative edge of SCL.

### 1.4.13 REG_RD – Read from Peripheral Register

| 31 – 28 | 27 – 23 | 22 – 18 | | 9 – 0 |
|---|---|---|---|---|
| 4'd2 | High | Low | | Addr |

**Figure 1.4-16. Instruction Type — REG_RD**

| **Operand** | **Description** - see Figure 1.4-16 |
|---|---|
| *Addr* | Register address, expressed in 32-bit words |
| *High* | Register end bit number |
| *Low* | Register start bit number |

**Description**

The instruction prompts a read of up to 16 bits from a peripheral register into a general-purpose register R0:

R0 = REG[Addr][High:Low]

In case of more than 16 bits being requested, i.e. *High* - *Low* + 1 > 16, then the instruction will return [Low+15:Low].

**Note:**

- This instruction can access registers in RTC_CNTL, RTC_IO, SENS and RTC_I2C peripherals. The address of the register, as seen from the ULP coprocessor, can be calculated from the address of the same register on the DPORT bus, as follows:

  addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE)/4

- The *addr_ulp* is expressed in 32-bit words (not in bytes), and value 0 maps onto the DR_REG_RTCCNTL_BASE (as seen from the main CPUs). Thus, 10 bits of address cover a 4096-byte range of peripheral register space, including regions DR_REG_RTCCNTL_BASE, DR_REG_RTCIO_BASE, DR_REG_SENS_BASE and DR_REG_RTC_I2C_BASE.

### 1.4.14 REG_WR – Write to Peripheral Register

| 31 – 28 | 27 – 23 | 22 – 18 | 17 – 10 | 9 – 0 |
|---|---|---|---|---|
| 4'd1 | High | Low | Data | Addr |

**Figure 1.4-17. Instruction Type — REG_WR**

| **Operand** | **Description** - see Figure 1.4-17 |
|---|---|
| *Addr* | Register address, expressed in 32-bit words |
| *High* | Register end bit number |
| *Low* | Register start bit number |
| *Data* | Value to write, 8 bits |

**Description**

The instruction prompts the writing of up to 8 bits from an immediate data value into a peripheral register.

REG[Addr][High:Low] = Data

If more than 8 bits are requested, i.e. *High* - *Low* + 1 > 8, then the instruction will pad with zeros the bits above the eighth bit.

**Note:**
See notes regarding *addr_ulp* in section 1.4.13 above.

## 1.5 ULP Program Execution

The ULP coprocessor is designed to operate independently of the main CPUs, while they are either in deep sleep or running.

In a typical power-saving scenario, the ULP coprocessor operates while the main CPUs are in deep sleep. To save power even further, the ULP coprocessor can get into sleep mode, as well. In such a scenario, there is a specific hardware timer in place to wake up the ULP coprocessor, since there is no software program running at the same time. This timer should be configured in advance by setting and then selecting one of the SENS_ULP_CP_SLEEP_CYC*n*_REG registers that contain the expiration period. This can be done either by the main program, or the ULP program with the REG_WR and SLEEP instructions. Then, the ULP timer should be enabled by setting bit RTC_CNTL_ULP_CP_SLP_TIMER_EN in the RTC_CNTL_STATE0_REG register.

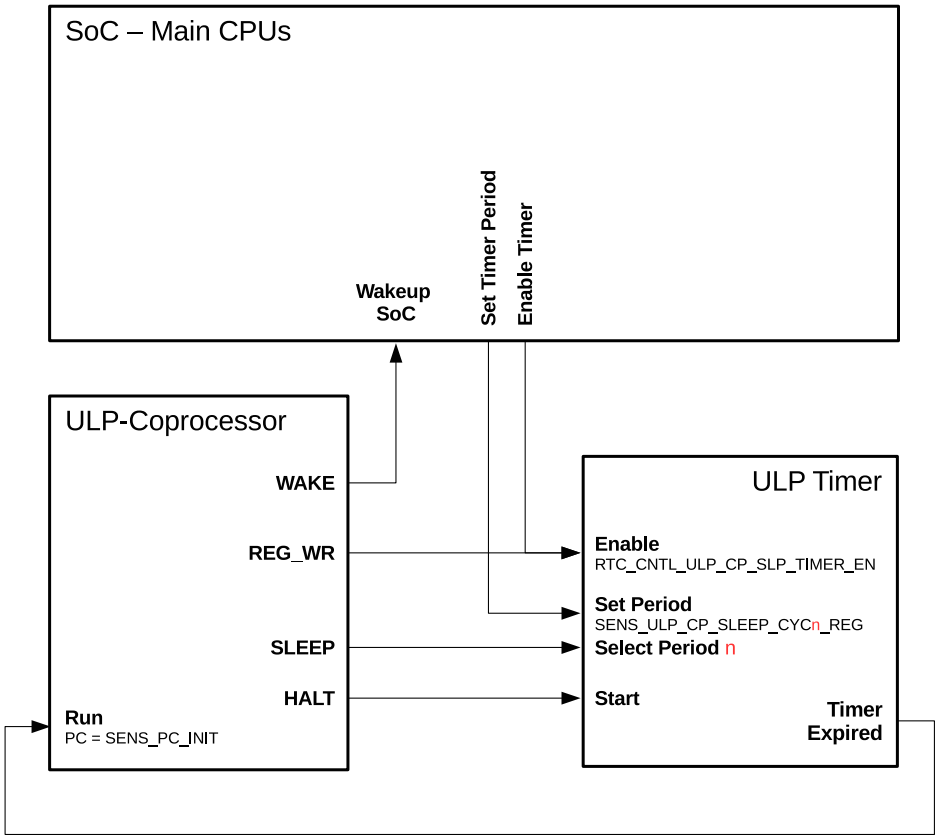



Figure 1.5-1. Control of ULP Program Execution

The ULP coprocessor puts itself into sleep mode by executing the HALT instruction. This also triggers the ULP timer to start counting RTC_SLOW_CLK ticks which, by default, originate from an internal 150 kHz RC oscillator. Once the timer expires, the ULP coprocessor is powered up and runs a program with the program counter

(PC) which is stored in register SENS_PC_INIT. The relationship between the described signals and registers is shown in Figure 1.5-1.

On reset or power-up the above-mentioned ULP program may start up only after the expiration of SENS_ULP_CP_SLEEP_CYC0_REG, which is the default selection period of the ULP timer.

A sample operation sequence of the ULP program is shown in Figure 1.5-2, where the following steps are executed:

1. Software enables the ULP timer by using bit RTC_CNTL_ULP_CP_SLP_TIMER_EN.
2. The ULP timer expires and the ULP coprocessor starts running the program at PC = SENS_PC_INIT.
3. The ULP program executes the HALT instruction; the ULP coprocessor is halted and the timer gets restarted.
4. The ULP program executes the SLEEP instruction to change the sleep timer period register.
5. The ULP program, or software, disables the ULP timer by using bit RTC_CNTL_ULP_CP_SLP_TIMER_EN.

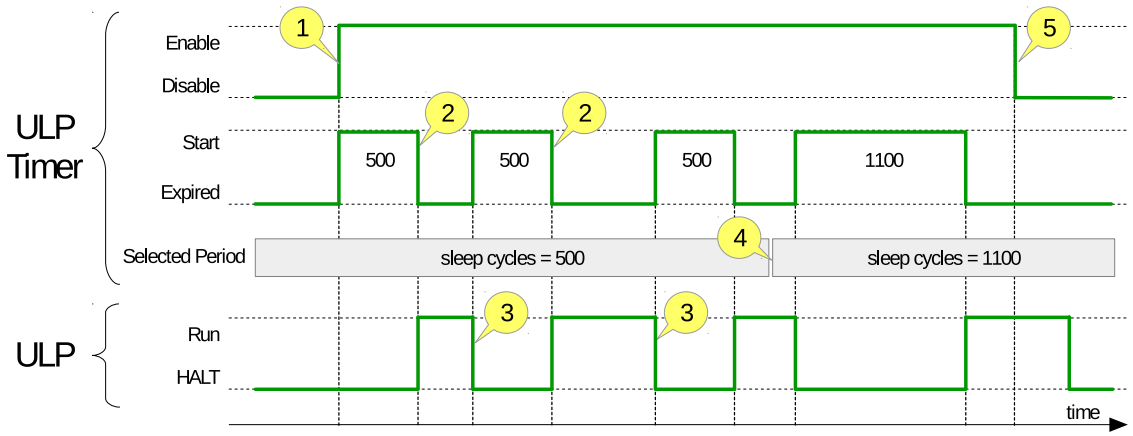



Figure 1.5-2. Sample of a ULP Operation Sequence

The specific timing of the wakeup, program execution and sleep sequence is governed by the ULP FSM as follows:

1. On the ULP timer expiration the FSM wakes up the ULP and this process takes two clock cycles.
2. Then, before executing the program, the FSM waits for the number of cycles configured in RTC_CNTL_ULPCP_TOUCH_START_WAIT field of the RTC_CNTL_TIMER2_REG register. This time is spent waiting for the 8 MHz clock to get stable.
3. The ULP program is executed.
4. After calling HALT instruction, the program is stopped. The FSM requires additional two clock cycles to put the ULP to sleep.

## 1.6 RTC_I2C Controller

The ULP coprocessor can use a separate I2C controller, located in the RTC domain, to communicate with external I2C slave devices. RTC_I2C has a limited feature set, compared to I2C0/I2C1 peripherals.

### 1.6.1 Configuring RTC_I2C

Before the ULP coprocessor can use the I2C instruction, certain parameters of the RTC_I2C need to be configured. This can be done by the program running on one of the main CPUs, or by the ULP coprocessor itself. Configuration is performed by writing certain timing parameters into the RTC_I2C registers:

1. Set the low and high SCL half-periods by using RTC_I2C_SCL_LOW_PERIOD_REG and RTC_I2C_SCL_HIGH_PERIOD_REG in RTC_FAST_CLK cycles (e.g. RTC_I2C_SCL_LOW_PERIOD=40, RTC_I2C_SCL_HIGH_PERIOD=40 for 100 kHz frequency).
2. Set the number of cycles between the SDA switch and the falling edge of SCL by using RTC_I2C_SDA_DUTY_REG in RTC_FAST_CLK (e.g. RTC_I2C_SDA_DUTY=16).
3. Set the waiting time after the START condition by using RTC_I2C_SCL_START_PERIOD_REG (e.g. RTC_I2C_SCL_START_PERIOD=30).
4. Set the waiting time before the END condition by using RTC_I2C_SCL_STOP_PERIOD_REG (e.g. RTC_I2C_SCL_STOP_PERIOD=44).
5. Set the transaction timeout by using RTC_I2C_TIMEOUT_REG (e.g. RTC_I2C_TIMEOUT=200).
6. Enable the master mode (set the RTC_I2C_MS_MODE bit in RTC_I2C_CTRL_REG).
7. Write the address(es) of external slave(s) to SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7). Up to eight slave addresses can be pre-programmed this way. One of these addresses can then be selected for each transaction as part of the ULP I2C instruction.

Once RTC_I2C is configured, instructions ULP I2C_RD and I2C_WR can be used.

### 1.6.2 Using RTC_I2C

The ULP coprocessor supports two instructions (with a single OpCode) for using RTC_I2C: I2C_RD (read) and I2C_WR (write).

#### 1.6.2.1 I2C_RD - Read a Single Byte

The I2C_RD instruction performs the following I2C transaction (see Figure 1.6-1):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 ("write"). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*, where *n* is given as an argument to the I2C_RD instruction.
3. Slave generates ACK.
4. Master sends slave register address (given as an argument to the I2C_RD instruction).
5. Slave generates ACK.
6. Master generates a repeated START condition.
7. Master sends slave address, with r/w bit set to 1 ("read").
8. Slave sends one byte of data.
9. Master generates NACK.
10. Master generates a STOP condition.

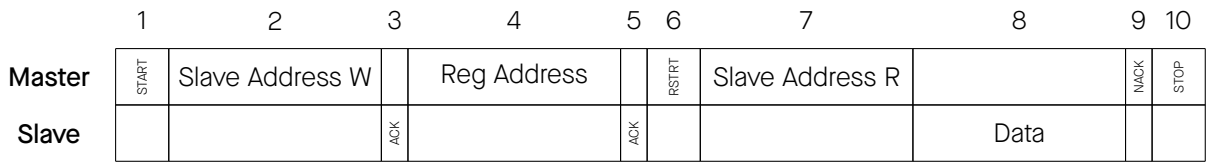



Figure 1.6-1. I2C Read Operation

**Note:**

The RTC_I2C peripheral samples the SDA signals on the falling edge of SCL. If the slave changes SDA in less than 0.38 microseconds, the master will receive incorrect data.

The byte received from the slave is stored into the R0 register.

### 1.6.2.2 I2C_WR - Write a Single Byte

The I2C_WR instruction performs the following I2C transaction (see Figure 1.6-2):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 ("write"). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*, where *n* is given as an argument to the I2C_WR instruction.
3. Slave generates ACK.
4. Master sends slave register address (given as an argument to the I2C_WR instruction).
5. Slave generates ACK.
6. Master generates a repeated START condition.
7. Master sends slave address, with r/w bit set to 0 ("write").
8. Master sends one byte of data.
9. Slave generates ACK.
10. Master generates a STOP condition.

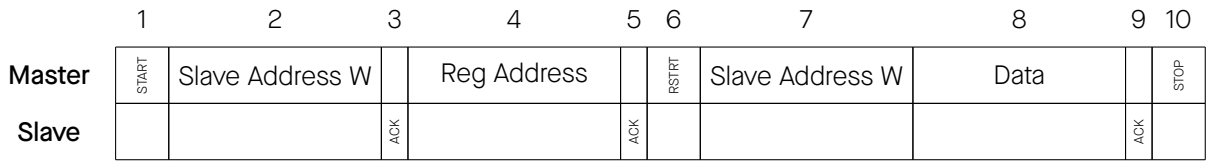



Figure 1.6-2. I2C Write Operation

### 1.6.2.3 Detecting Error Conditions

ULP I2C_RD and I2C_WR instructions will not report error conditions, such as a NACK from a slave, via ULP registers. Instead, applications can query specific bits in the RTC_I2C_INT_ST_REG register to determine if the transaction was successful. To enable checking for specific communication events, their corresponding bits should be set in register RTC_I2C_INT_EN_REG. Note that the bit map is shifted by 1. If a specific communication event is detected and set in register RTC_I2C_INT_ST_REG, it can then be cleared using RTC_I2C_INT_CLR_REG.

#### 1.6.2.4 Connecting I2C Signals

SDA and SCL signals can be mapped onto two out of the four GPIO pins, which are identified in Table *RTC_MUX Pin Summary* in Chapter *IO_MUX and GPIO Matrix*, using the RTCIO_SAR_I2C_IO_REG register.

## 1.7 Register Summary

### 1.7.1 SENS_ULP Address Space

| Name | Description | Address | Access |
|---|---|---|---|
| **ULP Timer cycles select** | | | |
| SENS_ULP_CP_SLEEP_CYC0_REG | Timer cycles setting 0 | 0x3FF48818 | R/W |
| SENS_ULP_CP_SLEEP_CYC1_REG | Timer cycles setting 1 | 0x3FF4881C | R/W |
| SENS_ULP_CP_SLEEP_CYC2_REG | Timer cycles setting 2 | 0x3FF48820 | R/W |
| SENS_ULP_CP_SLEEP_CYC3_REG | Timer cycles setting 3 | 0x3FF48824 | R/W |
| SENS_ULP_CP_SLEEP_CYC4_REG | Timer cycles setting 4 | 0x3FF48828 | R/W |
| **RTC I2C slave address select** | | | |
| SENS_SAR_SLAVE_ADDR1_REG | I2C addresses 0 and 1 | 0x3FF4883C | R/W |
| SENS_SAR_SLAVE_ADDR2_REG | I2C addresses 2 and 3 | 0x3FF48840 | R/W |
| SENS_SAR_SLAVE_ADDR3_REG | I2C addresses 4 and 5 | 0x3FF48844 | R/W |
| SENS_SAR_SLAVE_ADDR4_REG | I2C addresses 6 and 7, I2C control | 0x3FF48848 | R/W |
| **RTC I2C control** | | | |
| SENS_SAR_I2C_CTRL_REG | I2C control registers | 0x3FF48850 | R/W |

### 1.7.2 RTC_I2C Address Space

| Name | Description | Address | Access |
|---|---|---|---|
| **RTC I2C control registers** | | | |
| RTC_I2C_CTRL_REG | Transmission setting | 0x3FF48C04 | R/W |
| RTC_I2C_DEBUG_STATUS_REG | Debug status | 0x3FF48C08 | R/W |
| RTC_I2C_TIMEOUT_REG | Timeout setting | 0x3FF48C0C | R/W |
| RTC_I2C_SLAVE_ADDR_REG | Local slave address setting | 0x3FF48C10 | R/W |
| **RTC I2C signal setting registers** | | | |
| RTC_I2C_SDA_DUTY_REG | Configures the SDA hold time after a negative SCL edge | 0x3FF48C30 | R/W |
| RTC_I2C_SCL_LOW_PERIOD_REG | Configures the low level width of SCL | 0x3FF48C00 | R/W |
| RTC_I2C_SCL_HIGH_PERIOD_REG | Configures the high level width of SCL | 0x3FF48C38 | R/W |
| RTC_I2C_SCL_START_PERIOD_REG | Configures the delay between the SDA and SCL negative edge for a start condition | 0x3FF48C40 | R/W |
| RTC_I2C_SCL_STOP_PERIOD_REG | Configures the delay between the SDA and SCL positive edge for a stop condition | 0x3FF48C44 | R/W |
| **RTC I2C interrupt registers - listed only for debugging** | | | |
| RTC_I2C_INT_CLR_REG | Clear status of I2C communication events | 0x3FF48C24 | R/W |
| RTC_I2C_INT_EN_REG | Enable capture of I2C communication status events | 0x3FF48C28 | R/W |
| RTC_I2C_INT_ST_REG | Status of captured I2C communication events | 0x3FF48C2C | R/O |

**Note:**

Interrupts from RTC_I2C are not connected. The interrupt registers above are listed only for debugging purposes.

# 1.8 Registers

## 1.8.1 SENS_ULP Address Space

The addresses in parenthesis besides register names are the register addresses relative to (the RTC base address + 0x0800). The RTC base address is provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 1.7.1 *SENS_ULP Address Space*.

**Register 1.1. SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) (0x18+0x4**n*)**

| 31 | 0 | |
|---|---|---|
| 20 | | Reset |

**SENS_ULP_CP_SLEEP_CYC*n*_REG** ULP timer cycles setting *n*; the ULP coprocessor can select one of such registers by using the SLEEP instruction. (R/W)

**Register 1.2. SENS_SAR_START_FORCE_REG (0x002c)**

| (reserved) | SENS_PC_INIT | (reserved) | SENS_ULP_CP_START_TOP | SENS_ULP_CP_FORCE_START_TOP | (reserved) | |
|---|---|---|---|---|---|---|
| 31 22 | 21 11 | 10 | 9 | 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 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 |

**SENS_PC_INIT** ULP PC entry address. (R/W)

**SENS_ULP_CP_START_TOP** Set this bit to start the ULP coprocessor; it is active only when SENS_ULP_CP_FORCE_START_TOP = 1. (R/W)

**SENS_ULP_CP_FORCE_START_TOP** 1: ULP coprocessor is started by SENS_ULP_CP_START_TOP; 0: ULP coprocessor is started by timer. (R/W)

**Register 1.3. SENS_SAR_SLAVE_ADDR1_REG (0x003c)**

| (reserved) | SENS_I2C_SLAVE_ADDR0 | SENS_I2C_SLAVE_ADDR1 | |
|---|---|---|---|
| 31 22 | 21 11 | 10 0 | |
| 0 0 0 0 0 0 0 0 0 0 | 0x000 | 0x000 | Reset |

**SENS_I2C_SLAVE_ADDR0** I2C slave address 0. (R/W)

**SENS_I2C_SLAVE_ADDR1** I2C slave address 1. (R/W)

**Register 1.4. SENS_SAR_SLAVE_ADDR2_REG (0x0040)**

| (reserved) | SENS_I2C_SLAVE_ADDR2 | SENS_I2C_SLAVE_ADDR3 | |
|---|---|---|---|
| 31 22 | 21 11 | 10 0 | |
| 0 0 0 0 0 0 0 0 0 0 | 0x000 | 0x000 | Reset |

**SENS_I2C_SLAVE_ADDR2** I2C slave address 2. (R/W)

**SENS_I2C_SLAVE_ADDR3** I2C slave address 3. (R/W)

**Register 1.5. SENS_SAR_SLAVE_ADDR3_REG (0x0044)**

| (reserved) | SENS_I2C_SLAVE_ADDR4 | SENS_I2C_SLAVE_ADDR5 | |
|---|---|---|---|
| 31 22 | 21 11 | 10 0 | |
| 0 0 0 0 0 0 0 0 0 0 | 0x000 | 0x000 | Reset |

**SENS_I2C_SLAVE_ADDR4** I2C slave address 4. (R/W)

**SENS_I2C_SLAVE_ADDR5** I2C slave address 5. (R/W)

Register 1.6. SENS_SAR_SLAVE_ADDR4_REG (0x0048)

| (reserved) | SENS_I2C_DONE | SENS_I2C_RDATA | SENS_I2C_SLAVE_ADDR6 | SENS_I2C_SLAVE_ADDR7 | |
|---|---|---|---|---|---|
| 31 | 30 | 29 … 22 | 21 … 11 | 10 … 0 | |
| 0 | 0 | 0x000 | 0x000 | 0x000 | Reset |

**SENS_I2C_DONE** Indicate I2C done. (RO)

**SENS_I2C_RDATA** I2C read data. (RO)

**SENS_I2C_SLAVE_ADDR6** I2C slave address 6. (R/W)

**SENS_I2C_SLAVE_ADDR7** I2C slave address 7. (R/W)

Register 1.7. SENS_SAR_I2C_CTRL_REG (0x0050)

| (reserved) | SENS_SAR_I2C_START_FORCE | SENS_SAR_I2C_START | SENS_SAR_I2C_CTRL | |
|---|---|---|---|---|
| 31 … 30 | 29 | 28 | 27 … 0 | |
| 0 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**SENS_SAR_I2C_START_FORCE** 1: I2C started by SW, 0: I2C started by FSM. (R/W)

**SENS_SAR_I2C_START** Start I2C; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

**SENS_SAR_I2C_CTRL** I2C control data; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

## 1.8.2 RTC_I2C Address Space

The addresses in parenthesis besides register names are the register addresses relative to (the RTC base address + 0x0C00). The RTC base address is provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 1.7.2 *RTC_I2C Address Space*.

**Register 1.8. RTC_I2C_SCL_LOW_PERIOD_REG (0x000)**

| (reserved) | RTC_I2C_SCL_LOW_PERIOD | |
|---|---|---|
| 31 … 19 | 18 … 0 | |
| 0 0 0 0 0 0 0 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 |

**RTC_I2C_SCL_LOW_PERIOD** Number of RTC_FAST_CLK cycles when SCL == 0. (R/W)

**Register 1.9. RTC_I2C_CTRL_REG (0x004)**

| (reserved) | RTC_I2C_RX_LSB_FIRST | RTC_I2C_TX_LSB_FIRST | RTC_I2C_TRANS_START | RTC_I2C_MS_MODE | (reserved) | RTC_I2C_SCL_FORCE_OUT | RTC_I2C_SDA_FORCE_OUT | |
|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**RTC_I2C_RX_LSB_FIRST** Receive LSB first. (R/W)

**RTC_I2C_TX_LSB_FIRST** Send LSB first. (R/W)

**RTC_I2C_TRANS_START** Force to generate a start condition. (R/W)

**RTC_I2C_MS_MODE** Master (1), or slave (0). (R/W)

**RTC_I2C_SCL_FORCE_OUT** SCL is push-pull (1) or open-drain (0). (R/W)

**RTC_I2C_SDA_FORCE_OUT** SDA is push-pull (1) or open-drain (0). (R/W)

**Register 1.10. RTC_I2C_DEBUG_STATUS_REG (0x008)**

| (reserved) | RTC_I2C_SCL_STATE | RTC_I2C_MAIN_STATE | (reserved) | RTC_I2C_BYTE_TRANS | RTC_I2C_SLAVE_ADDR_MATCH | RTC_I2C_BUS_BUSY | RTC_I2C_ARB_LOST | RTC_I2C_TIMED_OUT | RTC_I2C_SLAVE_RW | RTC_I2C_ACK_VAL | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 28 | 27 25 | 24 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 | 0 | 0 | Reset |

**RTC_I2C_SCL_STATE** State of SCL machine. (R/W)

**RTC_I2C_MAIN_STATE** State of the main machine. (R/W)

**RTC_I2C_BYTE_TRANS** 8-bit transmit done. (R/W)

**RTC_I2C_SLAVE_ADDR_MATCH** Indicates whether the addresses are matched, when in slave mode. (R/W)

**RTC_I2C_BUS_BUSY** Operation is in progress. (R/W)

**RTC_I2C_ARB_LOST** Indicates the loss of I2C bus control, when in master mode. (R/W)

**RTC_I2C_TIMED_OUT** Transfer has timed out. (R/W)

**RTC_I2C_SLAVE_RW** Indicates the value of the received R/W bit, when in slave mode. (R/W)

**RTC_I2C_ACK_VAL** The value of ACK signal on the bus. (R/W)

**Register 1.11. RTC_I2C_TIMEOUT_REG (0x00c)**

| (reserved) | RTC_I2C_TIMEOUT | |
|---|---|---|
| 31 20 | 19 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_TIMEOUT** Maximum number of RTC_FAST_CLK cycles that the transmission can take. (R/W)

**Register 1.12. RTC_I2C_SLAVE_ADDR_REG (0x010)**

| RTC_I2C_SLAVE_ADDR_10BIT | (reserved) | RTC_I2C_SLAVE_ADDR | |
|---|---|---|---|
| 31 | 30 … 15 | 14 … 0 | |
| 0 | 0 0 0 0 0 0 0 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 |

**RTC_I2C_SLAVE_ADDR_10BIT** Set if local slave address is 10-bit. (R/W)

**RTC_I2C_SLAVE_ADDR** Local slave address. (R/W)

**Register 1.13. RTC_I2C_INT_CLR_REG (0x024)**

| (reserved) | RTC_I2C_TIME_OUT_INT_CLR | RTC_I2C_TRANS_COMPLETE_INT_CLR | RTC_I2C_MASTER_TRANS_COMPLETE_INT_CLR | RTC_I2C_ARBITRATION_LOST_INT_CLR | RTC_I2C_SLAVE_TRANS_COMPLETE_INT_CLR | (reserved) | |
|---|---|---|---|---|---|---|---|
| 31 … 9 | 8 | 7 | 6 | 5 | 4 | 3 … 0 | |
| 0 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_TIME_OUT_INT_CLR** Clear interrupt upon timeout. (R/W)

**RTC_I2C_TRANS_COMPLETE_INT_CLR** Clear interrupt upon detecting a stop pattern. (R/W)

**RTC_I2C_MASTER_TRANS_COMPLETE_INT_CLR** Clear interrupt upon completion of transaction, when in master mode. (R/W)

**RTC_I2C_ARBITRATION_LOST_INT_CLR** Clear interrupt upon losing control of the bus, when in master mode. (R/W)

**RTC_I2C_SLAVE_TRANS_COMPLETE_INT_CLR** Clear interrupt upon completion of transaction, when in slave mode. (R/W)

**Register 1.14. RTC_I2C_INT_EN_REG (0x028)**

| (reserved) | RTC_I2C_TIME_OUT_INT_ENA | RTC_I2C_TRANS_COMPLETE_INT_ENA | RTC_I2C_MASTER_TRAN_COMP_INT_ENA | RTC_I2C_ARBITRATION_LOST_INT_ENA | (reserved) | |
|---|---|---|---|---|---|---|
| 31 … 9 | 8 | 7 | 6 | 5 | 4 … 0 | |
| 0 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_TIME_OUT_INT_ENA** Enable interrupt upon timeout. (R/W)

**RTC_I2C_TRANS_COMPLETE_INT_ENA** Enable interrupt upon detecting a stop pattern. (R/W)

**RTC_I2C_MASTER_TRAN_COMP_INT_ENA** Enable interrupt upon completion of transaction, when in master mode. (R/W)

**RTC_I2C_ARBITRATION_LOST_INT_ENA** Enable interrupt upon losing control of the bus, when in master mode. (R/W)

**Register 1.15. RTC_I2C_INT_ST_REG (0x02c)**

| (reserved) | RTC_I2C_TIME_OUT_INT_ST | RTC_I2C_TRANS_COMPLETE_INT_ST | RTC_I2C_MASTER_TRAN_COMP_INT_ST | RTC_I2C_ARBITRATION_LOST_INT_ST | (reserved) | |
|---|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 | 3 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_TIME_OUT_INT_ST** Detected timeout. (R/O)

**RTC_I2C_TRANS_COMPLETE_INT_ST** Detected stop pattern on I2C bus. (R/O)

**RTC_I2C_MASTER_TRAN_COMP_INT_ST** Transaction completed, when in master mode. (R/O)

**RTC_I2C_ARBITRATION_LOST_INT_ST** Bus control lost, when in master mode. (R/O)

**Register 1.16. RTC_I2C_SDA_DUTY_REG (0x030)**

| (reserved) | RTC_I2C_SDA_DUTY | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_SDA_DUTY** Number of RTC_FAST_CLK cycles between the SDA switch and the falling edge of SCL. (R/W)

**Register 1.17. RTC_I2C_SCL_HIGH_PERIOD_REG (0x038)**

| (reserved) | RTC_I2C_SCL_HIGH_PERIOD | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_SCL_HIGH_PERIOD** Number of RTC_FAST_CLK cycles when SCL == 1. (R/W)

**Register 1.18. RTC_I2C_SCL_START_PERIOD_REG (0x040)**

| (reserved) | RTC_I2C_SCL_START_PERIOD | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_SCL_START_PERIOD** Number of RTC_FAST_CLK cycles to wait before generating a start condition. (R/W)

**Register 1.19. RTC_I2C_SCL_STOP_PERIOD_REG (0x044)**

| (reserved) | RTC_I2C_SCL_STOP_PERIOD | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**RTC_I2C_SCL_STOP_PERIOD** Number of RTC_FAST_CLK cycles to wait before generating a stop condition. (R/W)

# Chapter 2

# DMA Controller (DMA)

## 2.1 Overview

Direct Memory Access (DMA) is used for high-speed data transfer between peripherals and memory, as well as from memory to memory. Data can be quickly moved with DMA without any CPU intervention, thus allowing for more efficient use of the cores when processing data.

In ESP32, the following peripherals support DMA:

- Three UART interfaces, namely UART0, UART1, and UART2, which use UART DMA (UDMA).
- Three SPI interfaces, namely SPI1, SPI2, and SPI3, which use SPI DMA Interface.
- Two I2S interfaces, namely I2S0 and I2S1, which use I2S DMA Interface.

> **Note:**
> The ADC controller can use DMA through the I2S interface. See Section ADC/DAC mode for details.

- The SDIO slave interface, which uses the dedicated DMA described in SDIO Slave Controller (SDIO).
- The SD/MMC host interface, which uses the dedicated DMA described in SD/MMC Host Controller (SDHOST).
- The Ethernet MAC (EMAC) interface, which uses the dedicated DMA described in Ethernet Media Access Controller (EMAC).
- Bluetooth (BT).
- Wi-Fi.

In the ESP32, 13 peripherals are capable of using DMA for data transfer, namely, UART0, UART1, UART2, SPI1, SPI2, SPI3, I2S0, I2S1, SDIO slave, SD/MMC host, EMAC, BT, and Wi-Fi.

## 2.2 Features

The DMA controllers in the ESP32 feature:

- AHB bus architecture
- Support for full-duplex and half-duplex data transfers
- Programmable data transfer length in bytes
- Support for 4-beat burst transfer
- 328 KB DMA address space

- All high-speed communication modules powered by DMA

## 2.3 Functional Description

All modules that require high-speed data transfer in bulk contain a DMA controller. DMA addressing uses the same data bus as the CPU to read/write to the internal RAM.

Each DMA controller features different functions. However, the architecture of the DMA engine (DMA_ENGINE) is the same in all DMA controllers.

### 2.3.1 DMA Engine Architecture

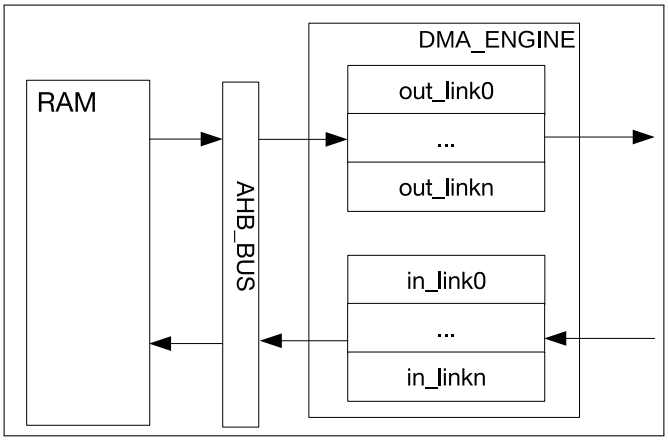



Figure 2.3-1. DMA Engine Architecture

The DMA Engine accesses SRAM over the AHB BUS. In Figure 2.3-1, the RAM represents the internal SRAM banks available on ESP32. Further details on the SRAM addressing range can be found in Chapter 3 *System and Memory*. Software can use a DMA Engine by assigning a linked list to define the DMA operational parameters.

The DMA Engine transmits the data from the RAM to a peripheral, according to the contents of the out_link descriptor. Also, the DMA Engine stores the data received from a peripheral into a specified RAM location, according to the contents of the in_link descriptor.

### 2.3.2 Linked List

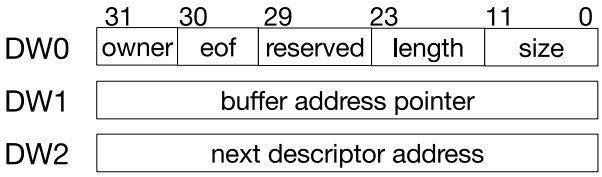



Figure 2.3-2. Linked List Structure

The DMA descriptor's linked lists (out_link and in_link) have the same structure. As shown in Figure 2.3-2, a linked-list descriptor consists of three words. The meaning of each field is as follows:

- owner (DW0) [31]: The allowed operator of the buffer corresponding to the current linked list.
  1'b0: the allowed operator is the CPU;
  1'b1: the allowed operator is the DMA controller.
- eof (DW0) [30]: End-Of-File character.
  1'b0: the linked-list item does not mark the end of the linked list;
  1'b1: the linked-list item is at the end of the linked list.
- reserved (DW0) [29:24]: Reserved bits.
  Software should not write 1's in this space.
- length (DW0) [23:12]: The number of valid bytes in the buffer corresponding to the current linked list. The field value indicates the number of bytes to be transferred to/from the buffer denoted by word DW1.
- size (DW0) [11:0]: The size of the buffer corresponding to the current linked list.
  **NOTE**: The size must be word-aligned.
- buffer address pointer (DW1): Buffer address pointer. This is the address of the data buffer.
  **NOTE**: The buffer address must be word-aligned.
- next descriptor address (DW2): The address pointer of the next linked-list item. The value is 0, if the current linked-list item is the last on the list (eof=1).

When receiving data, if the data transfer length is smaller than the specified buffer size, DMA will not use the remaining space. This enables the DMA engine to be used for transferring an arbitrary number of data bytes.

## 2.4 UART DMA (UDMA)

The ESP32 has three UART interfaces that share two UDMA (UART DMA) controllers. The UHCI_UART*x*_CE (*x* is 0, 1, or 2) is used for selecting the UART controller to use the UDMA.

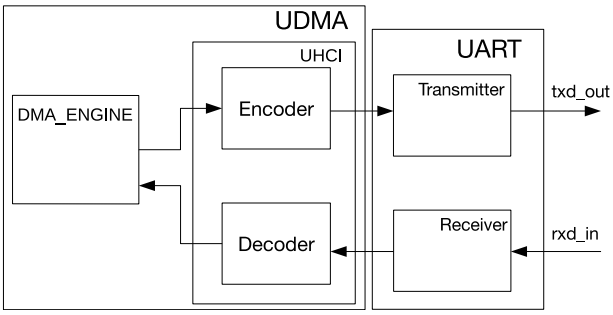



**Figure 2.4-1. Data Transfer in UDMA Mode**

Figure 2.4-1 shows the data transfer in UDMA mode. Before the DMA Engine receives data, software must initialize the receive-linked-list. UHCI_INLINK_ADDR is used to point to the first in_link descriptor. The register must be programmed with the lower 20 bits of the address of the initial linked-list item. After UHCI_INLINK_START is set, the Universal Host Controller Interface (UHCI) will transmit the data received by UART to the Decoder. After being parsed, the data will be stored in the RAM as specified by the receive-linked-list descriptor.

Before DMA transmits data, software must initialize the transmit-linked-list and the data to be transferred. UHCI_
OUTLINK_ADDR is used to point to the first out_link descriptor. The register must be programmed with the lower 20 bits of the address of the initial transmit-linked-list item. After UHCI_OUTLINK_START is set, the DMA Engine will read data from the RAM location specified by the linked-list descriptor and then transfer the data through the Encoder. The DMA Engine will then shift the data out serially through the UART transmitter.

The UART DMA follows a format of (separator + data + separator). The Encoder is used for adding separators before and after data, as well as using special-character sequences to replace data that are the same as separators. The Decoder is used for removing separators before and after data, as well as replacing the special-character sequences with separators. There can be multiple consecutive separators marking the beginning or end of data. These separators can be configured through UHCI_SEPER_CH, with the default values being 0xC0. Data that are the same as separators can be replaced with UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). After the transmission process is complete, a UHCI_OUT_TOTAL_EOF_INT interrupt will be generated. After the reception procedure is complete, a UHCI_IN_
SUC_EOF_INT interrupt will be generated.

**Note:**
Please note that the buffer address pointer field in in_link descriptors should be word-aligned, and the size field in the last in_link descriptor should be at least 4 bytes larger than the length of received data.

## 2.5 SPI DMA Interface

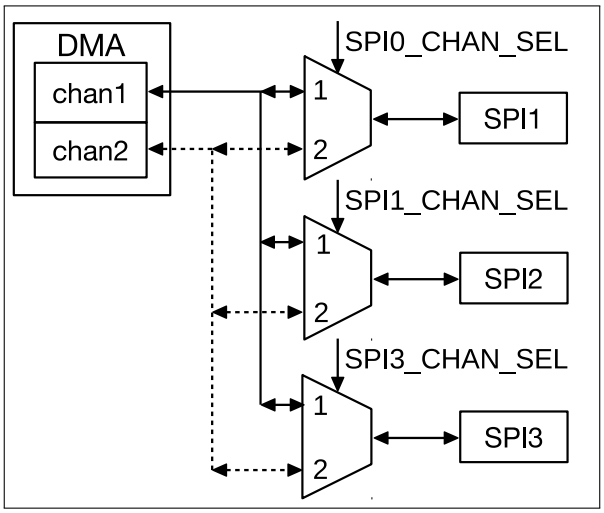



Figure 2.5-1. SPI DMA

ESP32 SPI modules can use DMA as well as the CPU for data exchange with peripherals. As can be seen from Figure 2.5-1, two DMA channels are shared by SPI1, SPI2 and SPI3 controllers. Each DMA channel can be used by any one SPI controller at any given time.

The ESP32 SPI DMA Engine also uses a linked list to receive/transmit data. Burst transmission is supported. The data size for a single transfer must be four bytes aligned. Consecutive data transfer is also

supported.

SPI1_DMA_CHAN_SEL[1:0], SPI2_DMA_CHAN_SEL[1:0] and SPI3_DMA_CHAN_SEL[1:0] in DPORT_SPI_DMA_
CHAN_SEL_REG must be configured to enable the SPI DMA interface for a specific SPI controller. Each SPI controller corresponds to one domain which has two bits with values 0, 1 and 2. Value 3 is reserved and must not be configured for operation.

Considering SPI1 as an example,
if SPI SPI1_DMA_CHAN_SEL[1:0] = 0, then SPI1 does not use any DMA channel;
if SPI1_DMA_CHAN_SEL[1:0] = 1, then SPI1 enables DMA channel1;
if SPI1_DMA_CHAN_SEL[1:0] = 2, then SPI1 enables DMA channel2.

The SPI_OUTLINK_START bit in SPI_DMA_OUT_LINK_REG and the SPI_INLINK_START bit in SPI_DMA_IN_LINK_REG are used for enabling the DMA Engine. The two bits are self-cleared by hardware. When SPI_OUTLINK_START is set to 1, the DMA Engine starts processing the outbound linked list descriptor and prepares to transmit data. When SPI_INLINK_START is set to 1, then the DMA Engine starts processing the inbound linked-list descriptor and gets prepared to receive data.

Software should configure the SPI DMA as follows:

1. Reset the DMA state machine and FIFO parameters;
2. Configure the DMA-related registers for operation;
3. Configure the SPI-controller-related registers accordingly;
4. Set SPI_USR to enable DMA operation.

## 2.6 I2S DMA Interface

The ESP32 integrates two I2S modules, I2S0 and I2S1, each of which is powered by a DMA channel. The REG_I2S_DSCR_EN bit in I2S_FIFO_CONF_REG is used for enabling the DMA operation. ESP32 I2S DMA uses the standard linked-list descriptor to configure DMA operations for data transfer. Burst transfer is supported. However, unlike the SPI DMA channels, the data size for a single transfer is one word, or four bytes. REG_I2S_RX_EOF_NUM[31:0] bit in I2S_RXEOF_NUM_REG is used for configuring the data size of a single transfer operation, in multiples of one word.

I2S_OUTLINK_START bit in I2S_OUT_LINK_REG and I2S_INLINK_START bit in I2S_IN_LINK_REG are used for enabling the DMA Engine and are self-cleared by hardware. When I2S_OUTLINK_START is set to 1, the DMA Engine starts processing the outbound linked-list descriptor and gets prepared to send data. When I2S_INLINK_START is set to 1, the DMA Engine starts processing the inbound linked-list descriptor and gets prepared to receive data.

Software should configure the I2S DMA as follows:

1. Configure I2S-controller-related registers;
2. Reset the DMA state machine and FIFO parameters;
3. Configure DMA-related registers for operation;
4. In I2S master mode, set I2S_TX_START bit or I2S_RX_START bit to initiate an I2S operation;

In I2S slave mode, set I2S_TX_START bit or I2S_RX_START bit and wait for data transfer to be initiated by the host device.

For more information on I2S DMA interrupts, please see Section DMA Interrupts, in Chapter *22 I2S Controller (I2S)*.

# Part II

# Memory Organization

This part provides insights into the system's memory structure, discussing the organization and mapping of RAM, ROM, eFuse, and external memories, offering a framework for understanding memory-related subsystems.

# Chapter 3

# System and Memory

## 3.1 Introduction

The ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. All embedded memory, external memory and peripherals are located on the data bus and/or the instruction bus of these CPUs.

With some minor exceptions (see below), the address mapping of two CPUs is symmetric, meaning that they use the same addresses to access the same memory. Multiple peripherals in the system can access embedded memory via DMA.

The two CPUs are named "PRO_CPU" and "APP_CPU" (for "protocol" and "application"), however, for most purposes the two CPUs are interchangeable.

## 3.2 Features

- Address Space
  - Symmetric address mapping
  - 4 GB (32-bit) address space for both data bus and instruction bus
  - 1296 KB embedded memory address space
  - 19704 KB external memory address space
  - 512 KB peripheral address space
  - Some embedded and external memory regions can be accessed by either data bus or instruction bus
  - 328 KB DMA address space
- Embedded Memory
  - 448 KB Internal ROM
  - 520 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- External Memory
  Off-chip SPI memory can be mapped into the available address space as external memory. Parts of the embedded memory can be used as transparent cache for this external memory.
  - Supports up to 16 MB off-Chip SPI Flash.

  - Supports up to 8 MB off-Chip SPI SRAM.
- Peripherals
  - 41 peripherals
- DMA
  - 13 modules are capable of DMA operation

The block diagram in Figure 3.2-1 illustrates the system structure, and the block diagram in Figure 3.2-2 illustrates the address map structure.

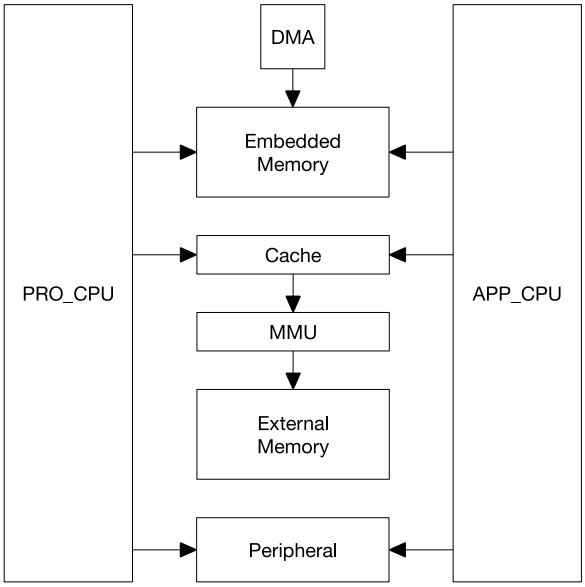



Figure 3.2-1. System Structure

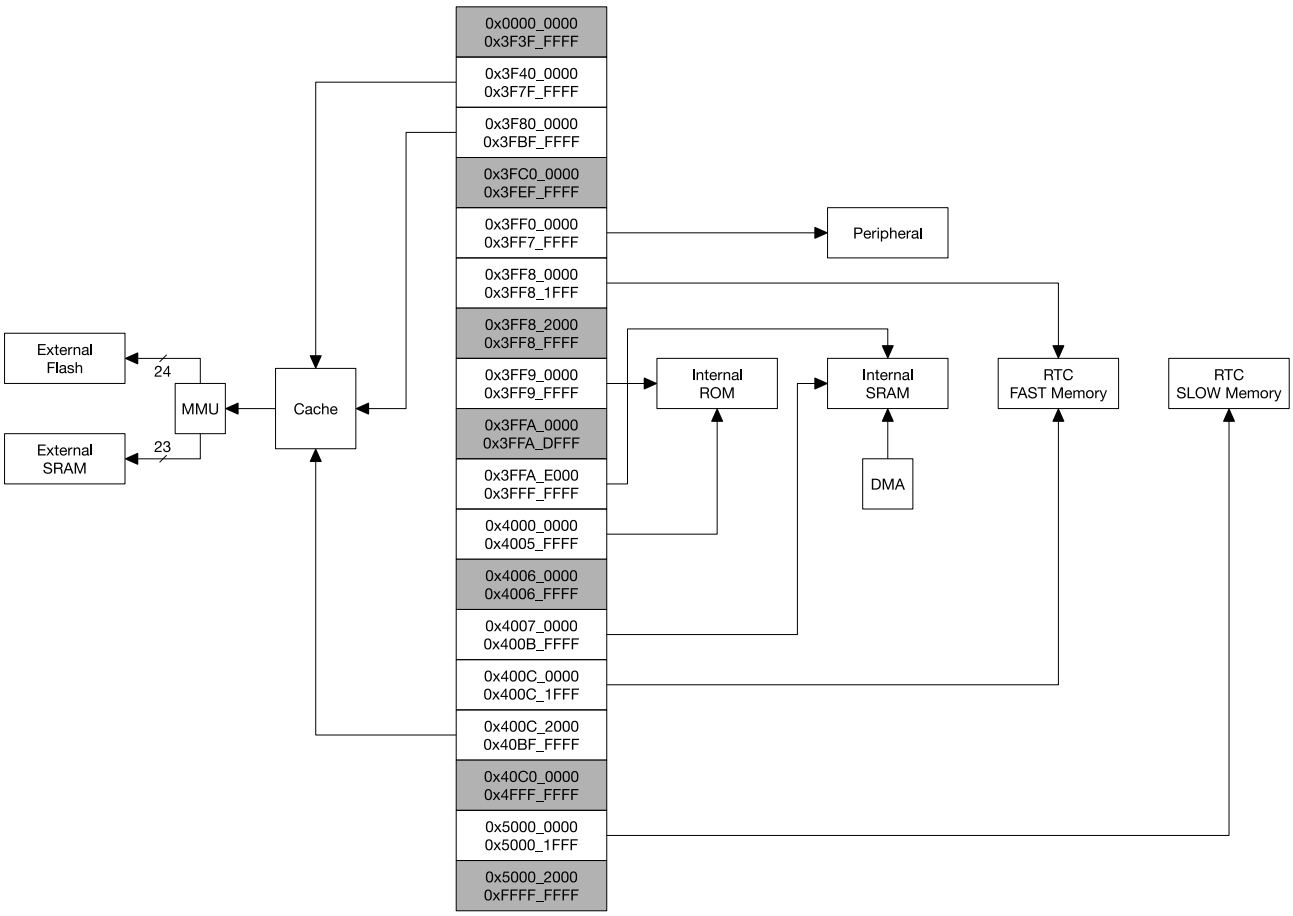



Figure 3.2-2. System Address Mapping

## 3.3 Functional Description

### 3.3.1 Address Mapping

Each of the two Harvard Architecture Xtensa LX6 CPUs has 4 GB (32-bit) address space. Address spaces are symmetric between the two CPUs.

Addresses below 0x4000_0000 are serviced using the data bus. Addresses in the range 0x4000_0000 ~ 0x4FFF_FFFF are serviced using the instruction bus. Finally, addresses over and including 0x5000_0000 are shared by the data and instruction bus.

The data bus and instruction bus are both little-endian: for example, byte addresses 0x0, 0x1, 0x2, 0x3 access the least significant, second least significant, second most significant, and the most significant bytes of the 32-bit word stored at the 0x0 address, respectively. The CPU can access data bus addresses via aligned or non-aligned byte, half-word and word read-and-write operations. The CPU can read and write data through the instruction bus, but only in a **word aligned manner**; non-word-aligned access will cause a CPU exception.

Each CPU can directly access embedded memory through both the data bus and the instruction bus, external memory which is mapped into the address space (via transparent caching & MMU), and peripherals. Table 3.3-1 illustrates address ranges that can be accessed by each CPU's data bus and instruction bus.

Some embedded memories and some external memories can be accessed via the data bus or the instruction bus. In these cases, the same memory is available to either of the CPUs at two address ranges.

**Table 3.3-1. Address Mapping**

| Bus Type | Boundary Address | | Size | Target |
|---|---|---|---|---|
| | Low Address | High Address | | |
| | 0x0000_0000 | 0x3F3F_FFFF | | Reserved |
| Data | 0x3F40_0000 | 0x3F7F_FFFF | 4 MB | External Memory |
| Data | 0x3F80_0000 | 0x3FBF_FFFF | 4 MB | External Memory |
| | 0x3FC0_0000 | 0x3FEF_FFFF | 3 MB | Reserved |
| Data | 0x3FF0_0000 | 0x3FF7_FFFF | 512 KB | Peripheral |
| Data | 0x3FF8_0000 | 0x3FFF_FFFF | 512 KB | Embedded Memory |
| Instruction | 0x4000_0000 | 0x400C_1FFF | 776 KB | Embedded Memory |
| Instruction | 0x400C_2000 | 0x40BF_FFFF | 11512 KB | External Memory |
| | 0x40C0_0000 | 0x4FFF_FFFF | 244 MB | Reserved |
| Data / Instruction | 0x5000_0000 | 0x5000_1FFF | 8 KB | Embedded Memory |
| | 0x5000_2000 | 0xFFFF_FFFF | | Reserved |

### 3.3.2 Embedded Memory

The Embedded Memory consists of four segments: internal ROM (448 KB), internal SRAM (520 KB), RTC FAST memory (8 KB) and RTC SLOW memory (8 KB).

The 448 KB internal ROM is divided into two parts: Internal ROM 0 (384 KB) and Internal ROM 1 (64 KB). The 520 KB internal SRAM is divided into three parts: Internal SRAM 0 (192 KB), Internal SRAM 1 (128 KB), and Internal SRAM 2 (200 KB). RTC FAST Memory and RTC SLOW Memory are both implemented as SRAM.

Table 3.3-2 lists all embedded memories and their address ranges on the data and instruction buses.

Table 3.3-2. Embedded Memory Address Mapping

| Bus Type | Boundary Address | | Size | Target | Comment |
|---|---|---|---|---|---|
| | Low Address | High Address | | | |
| Data | 0x3FF8_0000 | 0x3FF8_1FFF | 8 KB | RTC FAST Memory | PRO_CPU Only |
| | 0x3FF8_2000 | 0x3FF8_FFFF | 56 KB | Reserved | - |
| Data | 0x3FF9_0000 | 0x3FF9_FFFF | 64 KB | Internal ROM 1 | - |
| | 0x3FFA_0000 | 0x3FFA_DFFF | 56 KB | Reserved | - |
| Data | 0x3FFA_E000 | 0x3FFD_FFFF | 200 KB | Internal SRAM 2 | DMA |
| Data | 0x3FFE_0000 | 0x3FFF_FFFF | 128 KB | Internal SRAM 1 | DMA |
| Bus Type | Boundary Address | | Size | Target | Comment |
| | Low Address | High Address | | | |
| Instruction | 0x4000_0000 | 0x4000_7FFF | 32 KB | Internal ROM 0 | Remap |
| Instruction | 0x4000_8000 | 0x4005_FFFF | 352 KB | Internal ROM 0 | - |
| | 0x4006_0000 | 0x4006_FFFF | 64 KB | Reserved | - |
| Instruction | 0x4007_0000 | 0x4007_FFFF | 64 KB | Internal SRAM 0 | Cache |
| Instruction | 0x4008_0000 | 0x4009_FFFF | 128 KB | Internal SRAM 0 | - |
| Instruction | 0x400A_0000 | 0x400A_FFFF | 64 KB | Internal SRAM 1 | - |
| Instruction | 0x400B_0000 | 0x400B_7FFF | 32 KB | Internal SRAM 1 | Remap |
| Instruction | 0x400B_8000 | 0x400B_FFFF | 32 KB | Internal SRAM 1 | - |
| Instruction | 0x400C_0000 | 0x400C_1FFF | 8 KB | RTC FAST Memory | PRO_CPU Only |
| Bus Type | Boundary Address | | Size | Target | Comment |
| | Low Address | High Address | | | |
| Data Instruction | 0x5000_0000 | 0x5000_1FFF | 8 KB | RTC SLOW Memory | - |

#### 3.3.2.1 Internal ROM 0

The capacity of Internal ROM 0 is 384 KB. It is accessible by both CPUs through the address range 0x4000_0000 ~ 0x4005_FFFF, which is on the instruction bus.

The address range of the first 32 KB of the ROM 0 (0x4000_0000 ~ 0x4000_7FFF) can be remapped in order to access a part of Internal SRAM 1 that normally resides in a memory range of 0x400B_0000 ~ 0x400B_7FFF. While remapping, the 32 KB SRAM cannot be accessed by an address range of 0x400B_0000 ~ 0x400B_7FFF any more, but it can still be accessible through the data bus (0x3FFE_8000 ~ 0x3FFE_FFFF). This can be done on a per-CPU basis: setting bit 0 of register DPORT_PRO_BOOT_REMAP_CTRL_REG or DPORT_APP_BOOT_REMAP_CTRL_REG will remap SRAM for the PRO_CPU and APP_CPU, respectively.

#### 3.3.2.2 Internal ROM 1

The capacity of Internal ROM 1 is 64 KB. It can be read by either CPU at an address range 0x3FF9_0000 ~ 0x3FF9_FFFF of the data bus.

#### 3.3.2.3 Internal SRAM 0

The capacity of Internal SRAM 0 is 192 KB. Hardware can be configured to use the first 64 KB to cache external memory access. When not used as cache, the first 64 KB can be read and written by either CPU at addresses

0x4007_0000 ~ 0x4007_FFFF of the instruction bus. The remaining 128 KB can always be read and written by either CPU at addresses 0x4008_0000 ~ 0x4009_FFFF of instruction bus.

#### 3.3.2.4 Internal SRAM 1

The capacity of Internal SRAM 1 is 128 KB. Either CPU can read and write this memory at addresses 0x3FFE_0000 ~ 0x3FFF_FFFF of the data bus, and also at addresses 0x400A_0000 ~ 0x400B_FFFF of the instruction bus.

The address range accessed via the instruction bus is in reverse order (word-wise) compared to access via the data bus. That is to say, address
0x3FFE_0000 and 0x400B_FFFC access the same word
0x3FFE_0004 and 0x400B_FFF8 access the same word
0x3FFE_0008 and 0x400B_FFF4 access the same word
……
0x3FFF_FFF4 and 0x400A_0008 access the same word
0x3FFF_FFF8 and 0x400A_0004 access the same word
0x3FFF_FFFC and 0x400A_0000 access the same word

The data bus and instruction bus of the CPU are still both little-endian, so the byte order of individual words is not reversed between address spaces. For example, address
0x3FFE_0000 accesses the least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0001 accesses the second least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0002 accesses the second most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0003 accesses the most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0004 accesses the least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0005 accesses the second least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0006 accesses the second most significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0007 accesses the most significant byte in the word accessed by 0x400B_FFF8.
……
0x3FFF_FFF8 accesses the least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFF9 accesses the second least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFA accesses the second most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFB accesses the most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFC accesses the least significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFD accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFE accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFF accesses the most significant byte in the word accessed by 0x400A_0000.

Part of this memory can be remapped onto the ROM 0 address space. See Internal Rom 0 for more information.

#### 3.3.2.5 Internal SRAM 2

The capacity of Internal SRAM 2 is 200 KB. It can be read and written by either CPU at addresses 0x3FFA_E000 ~ 0x3FFD_FFFF on the data bus.

#### 3.3.2.6 DMA

DMA uses the same addressing as the CPU data bus to read and write Internal SRAM 1 and Internal SRAM 2. This means DMA uses an address range of 0x3FFE_0000 ~ 0x3FFF_FFFF to read and write Internal SRAM 1 and an address range of 0x3FFA_E000 ~ 0x3FFD_FFFF to read and write Internal SRAM 2.

In the ESP32, 13 peripherals are equipped with DMA. Table 3.3-3 lists these peripherals.

**Table 3.3-3. Module with DMA**

| | | |
|---|---|---|
| UART0 | UART1 | UART2 |
| SPI1 | SPI2 | SPI3 |
| I2S0 | I2S1 | |
| SDIO Slave | SDMMC | |
| EMAC | | |
| BT | WIFI | |

#### 3.3.2.7 RTC FAST Memory

RTC FAST Memory is 8 KB of SRAM. It can be read and written by PRO_CPU only at an address range of 0x3FF8_0000 ~ 0x3FF8_1FFF on the data bus or at an address range of 0x400C_0000 ~ 0x400C_1FFF on the instruction bus. Unlike most other memory regions, RTC FAST memory cannot be accessed by the APP_CPU.

The two address ranges of PRO_CPU access RTC FAST Memory in the same order, so, for example, addresses 0x3FF8_0000 and 0x400C_0000 access the same word. **On the APP_CPU, these address ranges do not provide access to RTC FAST Memory or any other memory location.**

#### 3.3.2.8 RTC SLOW Memory

RTC SLOW Memory is 8 KB of SRAM which can be read and written by either CPU at an address range of 0x5000_0000 ~ 0x5000_1FFF. This address range is shared by both the data bus and the instruction bus.

### 3.3.3 External Memory

The ESP32 can access external SPI flash and SPI SRAM as external memory. Table 3.3-4 provides a list of external memories that can be accessed by either CPU at a range of addresses on the data and instruction buses. When a CPU accesses external memory through the Cache and MMU, the cache will map the CPU's address to an external physical memory address (in the external memory's address space), according to the MMU settings. Due to this address mapping, the ESP32 can address up to 16 MB External Flash and 8 MB External SRAM.

**Table 3.3-4. External Memory Address Mapping**

| Bus Type | Boundary Address | | Size | Target | Comment |
|---|---|---|---|---|---|
| | Low Address | High Address | | | |
| Data | 0x3F40_0000 | 0x3F7F_FFFF | 4 MB | External Flash | Read |

| Data | 0x3F80_0000 | 0x3FBF_FFFF | 4 MB | External SRAM | Read and Write |
|---|---|---|---|---|---|
| Bus Type | Boundary Address | | Size | Target | Comment |
| | Low Address | High Address | | | |
| Instruction | 0x400C_2000 | 0x40BF_FFFF | 11512 KB | External Flash | Read |

## 3.3.4 Cache

As shown in Figure 3.3-1, each of the two CPUs in ESP32 has 32 KB of cache featuring a block size of 32 bytes for accessing external storage. PRO CPU uses bit PRO_CACHE_ENABLE in register DPORT_PRO_CACHE_CTRL_REG to enable the Cache, while APP CPU uses bit APP_CACHE_ENABLE in register DPORT_APP_CACHE_CTRL_REG to enable the same function.

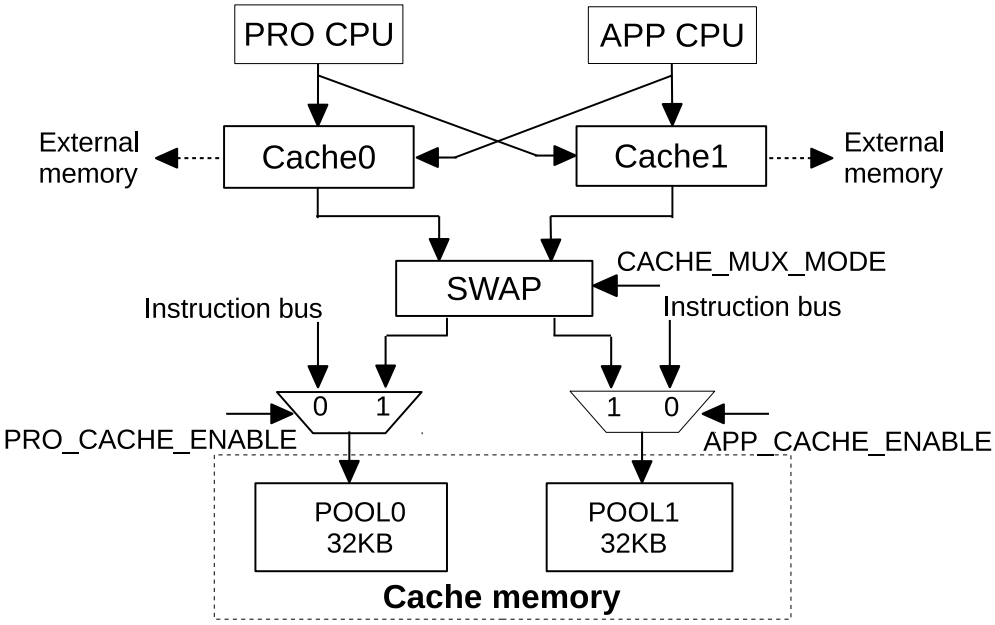



Figure 3.3-1. Cache Block Diagram

ESP32 uses a two-way set-associative cache. When the Cache function is to be used either by PRO CPU or APP CPU, bit CACHE_MUX_MODE[1:0] in register DPORT_CACHE_MUX_MODE_REG can be set to select POOL0 or POOL1 in the Internal SRAM0 as the cache memory. When both PRO CPU and APP CPU use the Cache function, POOL0 and POOL1 in the Internal SRAM0 will be used simultaneously as the cache memory, while they can also be used by the instruction bus. This is depicted in table 3.3-5 below.

Table 3.3-5. Cache memory mode

| CACHE_MUX_MODE | POOL0 | POOL1 |
|---|---|---|
| 0 | PRO CPU | APP CPU |
| 1 | PRO CPU/APP CPU | - |
| 2 | - | PRO CPU/APP CPU |
| 3 | APP CPU | PRO CPU |

As described in table 3.3-5, when bit CACHE_MUX_MODE is set to 1 or 2, PRO CPU and APP CPU cannot enable the Cache function at the same time. When the Cache function is enabled, POOL0 or POOL1 can only

be used as the cache memory, and cannot be used by the instruction bus as well.

ESP32 Cache supports the Flush function. It is worth noting that when the Flush function is used, the data written in the cache will be disposed rather than being rewritten into the External SRAM. To enable the Flush function, first clear bit *x*_CACHE_FLUSH_ENA in register DPORT_*x*_CACHE_CTRL_REG, then set this bit to 1. Afterwards, the system hardware will set bit *x*_CACHE_FLUSH_DONE to 1, where *x* can be "PRO" or "APP", indicating that the cache flush operation has been completed.

For more information about the address mapping of ESP32 Cache, please refer to Embedded Memory and External Memory.

### 3.3.5 Peripherals

The ESP32 has 41 peripherals. Table 3.3-6 specifically describes the peripherals and their respective address ranges. Nearly all peripheral modules can be accessed by either CPU at the same address with just a single exception; this being the PID Controller.

**Table 3.3-6. Peripheral Address Mapping**

| Bus Type | Boundary Address | | Size | Target | Comment |
|---|---|---|---|---|---|
| | Low Address | High Address | | | |
| Data | 0x3FF0_0000 | 0x3FF0_0FFF | 4 KB | DPort Register | |
| Data | 0x3FF0_1000 | 0x3FF0_1FFF | 4 KB | AES Accelerator | |
| Data | 0x3FF0_2000 | 0x3FF0_2FFF | 4 KB | RSA Accelerator | |
| Data | 0x3FF0_3000 | 0x3FF0_3FFF | 4 KB | SHA Accelerator | |
| Data | 0x3FF0_4000 | 0x3FF0_4FFF | 4 KB | Secure Boot | |
| | 0x3FF0_5000 | 0x3FF0_FFFF | 44 KB | Reserved | |
| Data | 0x3FF1_0000 | 0x3FF1_3FFF | 16 KB | Cache MMU Table | |
| | 0x3FF1_4000 | 0x3FF1_EFFF | 44 KB | Reserved | |
| Data | 0x3FF1_F000 | 0x3FF1_FFFF | 4 KB | PID Controller | Per-CPU peripheral |
| | 0x3FF2_0000 | 0x3FF3_FFFF | 128 KB | Reserved | |
| Data | 0x3FF4_0000 | 0x3FF4_0FFF | 4 KB | UART0 | |
| | 0x3FF4_1000 | 0x3FF4_1FFF | 4 KB | Reserved | |
| Data | 0x3FF4_2000 | 0x3FF4_2FFF | 4 KB | SPI1 | |
| Data | 0x3FF4_3000 | 0x3FF4_3FFF | 4 KB | SPI0 | |
| Data | 0x3FF4_4000 | 0x3FF4_4FFF | 4 KB | GPIO | |
| | 0x3FF4_5000 | 0x3FF4_7FFF | 12 KB | Reserved | |
| Data | 0x3FF4_8000 | 0x3FF4_8FFF | 4 KB | RTC | |
| Data | 0x3FF4_9000 | 0x3FF4_9FFF | 4 KB | IO MUX | |
| | 0x3FF4_A000 | 0x3FF4_AFFF | 4 KB | Reserved | |
| Data | 0x3FF4_B000 | 0x3FF4_BFFF | 4 KB | SDIO Slave | One of three parts |
| Data | 0x3FF4_C000 | 0x3FF4_CFFF | 4 KB | UDMA1 | |
| | 0x3FF4_D000 | 0x3FF4_EFFF | 8 KB | Reserved | |
| Data | 0x3FF4_F000 | 0x3FF4_FFFF | 4 KB | I2S0 | |
| Data | 0x3FF5_0000 | 0x3FF5_0FFF | 4 KB | UART1 | |
| | 0x3FF5_1000 | 0x3FF5_2FFF | 8 KB | Reserved | |
| Data | 0x3FF5_3000 | 0x3FF5_3FFF | 4 KB | I2C0 | |

| Bus Type | Boundary Address | | Size | Target | Comment |
|---|---|---|---|---|---|
| | Low Address | High Address | | | |
| Data | 0x3FF5_4000 | 0x3FF5_4FFF | 4 KB | UDMA0 | |
| Data | 0x3FF5_5000 | 0x3FF5_5FFF | 4 KB | SDIO Slave | One of three parts |
| Data | 0x3FF5_6000 | 0x3FF5_6FFF | 4 KB | RMT | |
| Data | 0x3FF5_7000 | 0x3FF5_7FFF | 4 KB | PCNT | |
| Data | 0x3FF5_8000 | 0x3FF5_8FFF | 4 KB | SDIO Slave | One of three parts |
| Data | 0x3FF5_9000 | 0x3FF5_9FFF | 4 KB | LED PWM | |
| Data | 0x3FF5_A000 | 0x3FF5_AFFF | 4 KB | eFuse Controller | |
| Data | 0x3FF5_B000 | 0x3FF5_BFFF | 4 KB | Flash Encryption | |
| | 0x3FF5_C000 | 0x3FF5_DFFF | 8 KB | Reserved | |
| Data | 0x3FF5_E000 | 0x3FF5_EFFF | 4 KB | MCPWM0 | |
| Data | 0x3FF5_F000 | 0x3FF5_FFFF | 4 KB | TIMG0 | |
| Data | 0x3FF6_0000 | 0x3FF6_0FFF | 4 KB | TIMG1 | |
| | 0x3FF6_1000 | 0x3FF6_3FFF | 12 KB | Reserved | |
| Data | 0x3FF6_4000 | 0x3FF6_4FFF | 4 KB | SPI2 | |
| Data | 0x3FF6_5000 | 0x3FF6_5FFF | 4 KB | SPI3 | |
| Data | 0x3FF6_6000 | 0x3FF6_6FFF | 4 KB | SYSCON | |
| Data | 0x3FF6_7000 | 0x3FF6_7FFF | 4 KB | I2C1 | |
| Data | 0x3FF6_8000 | 0x3FF6_8FFF | 4 KB | SDMMC | |
| Data | 0x3FF6_9000 | 0x3FF6_AFFF | 8 KB | EMAC | |
| Data | 0x3FF6_B000 | 0x3FF6_BFFF | 4KB | TWAI | |
| Data | 0x3FF6_C000 | 0x3FF6_CFFF | 4 KB | MCPWM1 | |
| Data | 0x3FF6_D000 | 0x3FF6_DFFF | 4 KB | I2S1 | |
| Data | 0x3FF6_E000 | 0x3FF6_EFFF | 4 KB | UART2 | |
| Data | 0x3FF6_F000 | 0x3FF6_FFFF | 4 KB | Reserved | |
| Data | 0x3FF7_0000 | 0x3FF7_0FFF | 4 KB | Reserved | |
| | 0x3FF7_1000 | 0x3FF7_4FFF | 16 KB | Reserved | |
| Data | 0x3FF7_5000 | 0x3FF7_5FFF | 4 KB | RNG | |
| | 0x3FF7_6000 | 0x3FF7_FFFF | 40 KB | Reserved | |

**Notice:**

- Peripherals accessed by the CPU via 0x3FF40000 ~ 0x3FF7FFFF address space (DPORT address) can also be accessed via 0x60000000 ~ 0x6003FFFF (AHB address). (0x3FF40000 + n) address and (0x60000000 + n) address access the same content, where n = 0 ~ 0x3FFFF.
- The CPU can access peripherals via DPORT address more efficiently than via AHB address. However, DPORT address is characterized by speculative reads, which means it cannot guarantee that each read is valid. In addition, DPORT address will upset the order of r/w operations on the bus to improve performance, which may cause programs that have strict requirements on the r/w order to crash. On the other hand, using AHB address to read FIFO registers will cause unpredictable errors. To address above issues please strictly follow the instructions documented in *ESP32 ECO and Workarounds for Bugs*, specifically sections 3.3, 3.10, 3.16, and 3.17.

#### 3.3.5.1 Asymmetric PID Controller Peripheral

There are two PID Controllers in the system. They serve the PRO_CPU and the APP_CPU, respectively. **The PRO_CPU and the APP_CPU can only access their own PID Controller and not that of their counterpart.** Each CPU uses the same memory range 0x3FF1_F000 ~ 3FF1_FFFF to access its own PID Controller.

#### 3.3.5.2 Non-Contiguous Peripheral Memory Ranges

The SDIO Slave peripheral consists of three parts and the two CPUs use non-contiguous addresses to access these. The three parts are accessed at the address ranges 0x3FF4_B000 ~ 3FF4_BFFF, 0x3FF5_5000 ~ 3FF5_5FFF and 0x3FF5_8000 ~ 3FF5_8FFF of each CPU's data bus. Similarly to other peripherals, access to this peripheral is identical for both CPUs.

#### 3.3.5.3 Memory Speed

The ROM as well as the SRAM are both clocked from CPU_CLK and can be accessed by the CPU in a single cycle. The RTC FAST memory is clocked from the APB_CLOCK and the RTC SLOW memory from the FAST_CLOCK, so access to these memories may be slower. DMA uses the APB_CLK to access memory.

Internally, the SRAM is organized in 32K-sized banks. Each CPU and DMA channel can simultaneously access the SRAM at full speed, provided they access addresses in different memory banks.

# Chapter 4

# Memory Management and Protection Units (MMU, MPU)

## 4.1 Introduction

Every peripheral and memory section in the ESP32 is accessed through either an MMU (Memory Management Unit) or an MPU (Memory Protection Unit). An MPU can allow or disallow the access of an application to a memory range or peripheral, depending on what kind of permission the OS has given to that particular application. An MMU can perform the same operation, as well as a virtual-to-physical memory address translation. This can be used to map an internal or external memory range to a certain virtual memory area. These mappings can be application-specific. Therefore, each application can be adjusted and have the memory configuration that is necessary for it to run properly. To differentiate between the OS and applications, there are eight Process Identifiers (or PIDs) that each application, or OS, can run. Furthermore, each application, or OS, is equipped with their own sets of mappings and rights.

## 4.2 Features

- Eight processes in each of the PRO_CPU and APP_CPU
- MPU/MMU management of on-chip memories, off-chip memories, and peripherals, based on process ID
- On-chip memory management by MPU/MMU
- Off-chip memory management by MMU
- Peripheral management by MPU

## 4.3 Functional Description

### 4.3.1 PID Controller

In the ESP32, a PID controller acts as an indicator that signals the MMU/MPU the owner PID of the code that is currently running. The intention is that the OS updates the PID in the PID controller every time it switches context to another application. The PID controller can detect interrupts and automatically switch PIDs to that of the OS, if so configured.

There are two peripheral PID controllers in the system, one for each of the two CPUs in the ESP32. Having a PID controller per CPU allows running different processes on different CPUs, if so desired.

## 4.3.2 MPU/MMU

The MPU and MMU manage on-chip memories, off-chip memories, and peripherals. To do this they are based on the process of accessing the peripheral or memory region. More specifically, when a code tries to access a MMU/MPU-protected memory region or peripheral, the MMU or MPU will receive the PID from the PID generator that is associated with the CPU on which the process is running.

For on-chip memory and peripherals, the decisions the MMU and MPU make are only based on this PID, whereas the specific CPU the code is running on is not taken into account. Subsequently, the MMU/MPU configuration for the internal memory and peripherals allows entries only for the eight different PIDs. In contrast, the MMU moderating access to the external memory takes not only the PID into account, but also the CPU the request is coming from. This means that MMUs have configuration options for every PID when running on the APP_CPU, as well as every PID when running on the PRO_CPU. While, in practice, accesses from both CPUs will be configured to have the same result for a specific process, doing so is not a hardware requirement.

The decision an MPU can make, based on this information, is to allow or deny a process to access the memory region or peripheral. An MMU has the same function, but additionally it redirects the virtual memory access, which the process acquired, into a physical memory access that can possibly reach out an entirely different physical memory region. This way, MMU-governed memory can be remapped on a process-by-process basis.

### 4.3.2.1 Embedded Memory

The on-chip memory is governed by fixed-function MPUs, configurable MPUs, and MMUs:

Table 4.3-1. MPU and MMU Structure for Internal Memory

| Name | Size | Address range | | Governed by |
|---|---|---|---|---|
| | | From | To | |
| ROM0 | 384 KB | 0x4000_0000 | 0x4005_FFFF | Static MPU |
| ROM1 | 64 KB | 0x3FF9_0000 | 0x3FF9_FFFF | Static MPU |
| SRAM0 | 64 KB | 0x4007_0000 | 0x4007_FFFF | Static MPU |
| | 128 KB | 0x4008_0000 | 0x4009_FFFF | SRAM0 MMU |
| SRAM1 (aliases) | 128 KB | 0x3FFE_0000 | 0x3FFF_FFFF | Static MPU |
| | 128 KB | 0x400A_0000 | 0x400B_FFFF | Static MPU |
| | 32 KB | 0x4000_0000 | 0x4000_7FFF | Static MPU |
| SRAM2 | 72 KB | 0x3FFA_E000 | 0x3FFB_FFFF | Static MPU |
| | 128 KB | 0x3FFC_0000 | 0x3FFD_FFFF | SRAM2 MMU |
| RTC FAST (aliases) | 8 KB | 0x3FF8_0000 | 0x3FF8_1FFF | RTC FAST MPU |
| | 8 KB | 0x400C_0000 | 0x400C_1FFF | RTC FAST MPU |
| RTC SLOW | 8 KB | 0x5000_0000 | 0x5000_1FFF | RTC SLOW MPU |

#### Static MPUs

ROM0, ROM1, the lower 64 KB of SRAM0, SRAM1 and the lower 72 KB of SRAM2 are governed by a static MPU. The behaviour of these MPUs are hardwired and cannot be configured by software. They moderate access to the memory region solely through the PID of the current process. When the PID of the process is 0

or 1, the memory can be read (and written when it is RAM) using the addresses specified in Table 4.3-1. When it is 2 ~ 7, the memory cannot be accessed.

#### RTC FAST & RTC SLOW MPU

The 8 KB RTC FAST Memory as well as the 8 KB of RTC SLOW Memory are governed by two configurable MPUs. The MPUs can be configured to allow or deny access to each individual PID, using the RTC_CNTL_RTC_PID_
CONFIG_REG and DPORT_AHBLITE_MPU_TABLE_RTC_REG registers. Setting a bit in these registers will allow the corresponding PID to read or write from the memory; clearing the bit disallows access. Access for PID 0 and 1 to RTC SLOW memory cannot be configured and is always enabled. Table 4.3-2 and 4.3-3 define the bit-to-PID mappings of the registers.

Table 4.3-2. MPU for RTC FAST Memory

| Size | Boundary address | | Authority |
|---|---|---|---|
| | Low | High | PID<br>RTC_CNTL_RTC_PID_CONFIG bit |
| 8 KB | 0x3FF8_0000 | 0x3FF8_1FFF | 0 1 2 3 4 5 6 7 |
| 8 KB | 0x400C_0000 | 0x400C_1FFF | 0 1 2 3 4 5 6 7 |

Table 4.3-3. MPU for RTC SLOW Memory

| Size | Boundary address | | Authority | |
|---|---|---|---|---|
| | Low | High | PID = 0/1 | PID<br>DPORT_AHBLITE_MPU_TABLE_RTC_REG bit |
| 8 KB | 0x5000_0000 | 0x5000_1FFF | Read/Write | 2 3 4 5 6 7<br>0 1 2 3 4 5 |

Register RTC_CNTL_RTC_PID_CONFIG_REG is part of the RTC peripheral and can only be modified by processes with a PID of 0; register DPORT_AHBLITE_MPU_TABLE_RTC_REG is a Dport register and can be changed by processes with a PID of 0 or 1.

#### SRAM0 and SRAM2 upper 128 KB MMUs

Both the upper 128 KB of SRAM0 and the upper 128 KB of SRAM2 are governed by an MMU. Not only can these MMUs allow or deny access to the memory they govern (just like the MPUs do), but they are also capable of translating the address a CPU reads from or writes to (which is a virtual address) to a possibly different address in memory (the physical address).

In order to accomplish this, the internal RAM MMUs divide the memory range they govern into 16 pages. The page size is configurable as 8 KB, 4 KB and 2 KB. When the page size is 8 KB, the 16 pages span the entire 128 KB memory region; when the page size is 4 KB or 2 KB, a non-MMU-covered region of 64 or 96 KB, respectively, will exist at the end of the memory space. Similar to the virtual and physical addresses, it is also possible to imagine the pages as having a virtual and physical component. The MMU can convert an address within a virtual page to an address within a physical page.

For PID 0 and 1, this mapping is 1-to-1, meaning that a read from or write to a certain virtual page will always be

converted to a read from or write to the exact same physical page. This allows an operating system, running under PID 0 and/or 1, to always have access to the entire physical memory range.

For PID 2 to 7, however, every virtual page can be reconfigured, on a per-PID basis, to map to a different physical page. This way, reads and writes to an offset within a virtual page get translated into reads and writes to the same offset within a different physical page. This is illustrated in Figure 4.3-1: the CPU (running a process with a PID between 2 to 7) tries to access memory address 0x3FFC_2345. This address is within the virtual Page 1 memory region, at offset 0x0345. The MMU is instructed that for this particular PID, it should translate an access to virtual page 1 into physical Page 2. This causes the memory access to be redirected to the same offset as the virtual memory access, yet in Page 2, which results in the effective access of physical memory address 0x3FFC_**4**345. The page size in this example is 8 KB.

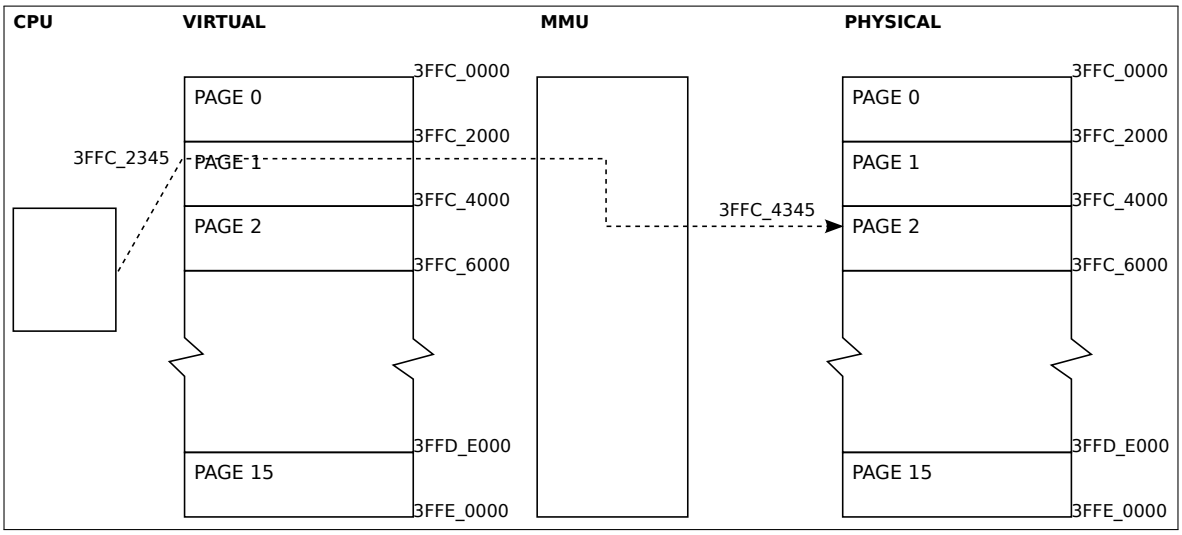



Figure 4.3-1. MMU Access Example

Table 4.3-4. Page Mode of MMU for the Remaining 128 KB of Internal SRAM0 and SRAM2

| DPORT_IMMU_PAGE_MODE | DPORT_DMMU_PAGE_MODE | Page size |
|---|---|---|
| 0 | 0 | 8 KB |
| 1 | 1 | 4 KB |
| 2 | 2 | 2 KB |

### Non-MMU Governed Memory

For the MMU-managed region of SRAM0 and SRAM2, the page size is configurable as 8 KB, 4 KB and 2 KB. The configuration is done by setting the DPORT_IMMU_PAGE_MODE (for SRAM0) and DPORT_DMMU_PAGE_MODE (for SRAM2) bits in registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, as detailed in Table 4.3-4. Because the number of pages for either region is fixed at 16, the total amount of memory covered by these pages is 128 KB when 8 KB pages are selected, 64 KB when 4 KB pages are selected, and 32 KB when 2 KB pages are selected. This implies that for 8 KB pages, the entire MMU-managed range is used, but for the other page sizes there will be a part of the 128 KB memory that will not be governed by the MMU settings. Concretely, for a page size of 4 KB, these regions are 0x4009_0000 to 0x4009_FFFF and 0x3FFD_0000 to 0x3FFD_FFFF; for a page size of 2 KB, the regions are 0x4008_8000 to 0x4009_FFFF and 0x3FFC_8000 to 0x3FFD_FFFF. These ranges are readable and writable by processes with a PID of 0 or 1; processes with other PIDs cannot access this memory.

The layout of the pages in memory space is linear, namely, an SRAM0 MMU page *n* covers address space $0x40080000 + (pagesize \times n)$ to $0x40080000 + (pagesize \times (n+1) - 1)$; similarly, an SRAM2 MMU page *n* covers $0x3FFC0000 + (pagesize \times n)$ to $0x3FFC0000 + (pagesize \times (n+1) - 1)$. Tables 4.3-5 and 4.3-6 show the resulting addresses in full.

**Table 4.3-5. Page Boundaries for SRAM0 MMU**

| Page | 8 KB Pages | | 4 KB Pages | | 2 KB Pages | |
|---|---|---|---|---|---|---|
| | Bottom | Top | Bottom | Top | Bottom | Top |
| 0 | 40080000 | 40081FFF | 40080000 | 40080FFF | 40080000 | 400807FF |
| 1 | 40082000 | 40083FFF | 40081000 | 40081FFF | 40080800 | 40080FFF |
| 2 | 40084000 | 40085FFF | 40082000 | 40082FFF | 40081000 | 400817FF |
| 3 | 40086000 | 40087FFF | 40083000 | 40083FFF | 40081800 | 40081FFF |
| 4 | 40088000 | 40089FFF | 40084000 | 40084FFF | 40082000 | 400827FF |
| 5 | 4008A000 | 4008BFFF | 40085000 | 40085FFF | 40082800 | 40082FFF |
| 6 | 4008C000 | 4008DFFF | 40086000 | 40086FFF | 40083000 | 400837FF |
| 7 | 4008E000 | 4008FFFF | 40087000 | 40087FFF | 40083800 | 40083FFF |
| 8 | 40090000 | 40091FFF | 40088000 | 40088FFF | 40084000 | 400847FF |
| 9 | 40092000 | 40093FFF | 40089000 | 40089FFF | 40084800 | 40084FFF |
| 10 | 40094000 | 40095FFF | 4008A000 | 4008AFFF | 40085000 | 400857FF |
| 11 | 40096000 | 40097FFF | 4008B000 | 4008BFFF | 40085800 | 40085FFF |
| 12 | 40098000 | 40099FFF | 4008C000 | 4008CFFF | 40086000 | 400867FF |
| 13 | 4009A000 | 4009BFFF | 4008D000 | 4008DFFF | 40086800 | 40086FFF |
| 14 | 4009C000 | 4009DFFF | 4008E000 | 4008EFFF | 40087000 | 400877FF |
| 15 | 4009E000 | 4009FFFF | 4008F000 | 4008FFFF | 40087800 | 40087FFF |
| Rest | - | - | 40090000 | 4009FFFF | 4008800 | 4009FFFF |

Table 4.3-6. Page Boundaries for SRAM2 MMU

| Page | 8 KB Pages | | 4 KB Pages | | 2 KB Pages | |
|---|---|---|---|---|---|---|
| | Bottom | Top | Bottom | Top | Bottom | Top |
| 0 | 3FFC0000 | 3FFC1FFF | 3FFC0000 | 3FFC0FFF | 3FFC0000 | 3FFC07FF |
| 1 | 3FFC2000 | 3FFC3FFF | 3FFC1000 | 3FFC1FFF | 3FFC0800 | 3FFC0FFF |
| 2 | 3FFC4000 | 3FFC5FFF | 3FFC2000 | 3FFC2FFF | 3FFC1000 | 3FFC17FF |
| 3 | 3FFC6000 | 3FFC7FFF | 3FFC3000 | 3FFC3FFF | 3FFC1800 | 3FFC1FFF |
| 4 | 3FFC8000 | 3FFC9FFF | 3FFC4000 | 3FFC4FFF | 3FFC2000 | 3FFC27FF |
| 5 | 3FFCA000 | 3FFCBFFF | 3FFC5000 | 3FFC5FFF | 3FFC2800 | 3FFC2FFF |
| 6 | 3FFCC000 | 3FFCDFFF | 3FFC6000 | 3FFC6FFF | 3FFC3000 | 3FFC37FF |
| 7 | 3FFCE000 | 3FFCFFFF | 3FFC7000 | 3FFC7FFF | 3FFC3800 | 3FFC3FFF |
| 8 | 3FFD0000 | 3FFD1FFF | 3FFC8000 | 3FFC8FFF | 3FFC4000 | 3FFC47FF |
| 9 | 3FFD2000 | 3FFD3FFF | 3FFC9000 | 3FFC9FFF | 3FFC4800 | 3FFC4FFF |
| 10 | 3FFD4000 | 3FFD5FFF | 3FFCA000 | 3FFCAFFF | 3FFC5000 | 3FFC57FF |
| 11 | 3FFD6000 | 3FFD7FFF | 3FFCB000 | 3FFCBFFF | 3FFC5800 | 3FFC5FFF |
| 12 | 3FFD8000 | 3FFD9FFF | 3FFCC000 | 3FFCCFFF | 3FFC6000 | 3FFC67FF |
| 13 | 3FFDA000 | 3FFDBFFF | 3FFCD000 | 3FFCDFFF | 3FFC6800 | 3FFC6FFF |
| 14 | 3FFDC000 | 3FFDDFFF | 3FFCE000 | 3FFCEFFF | 3FFC7000 | 3FFC77FF |
| 15 | 3FFDE000 | 3FFDFFFF | 3FFCF000 | 3FFCFFFF | 3FFC7800 | 3FFC7FFF |
| Rest | - | - | 3FFD0000 | 3FFDFFFF | 3FFC8000 | 3FFDFFFF |

### MMU Mapping

For each of the SRAM0 and SRAM2 MMUs, access rights and virtual to physical page mapping are done by a set of 16 registers. In contrast to most of the other MMUs, each register controls a **physical** page, not a virtual one. These registers control which of the PIDs have access to the physical memory, as well as which virtual page maps to this physical page. The bits in the register are described in Table 4.3-7. Keep in mind that these registers only govern accesses from processes with PID 2 to 7; PID 0 and 1 always have full read and write access to all pages and no virtual-to-physical mapping is done. In other words, if a process with a PID of 0 or 1 accesses virtual page *x*, the access will always go to physical page *x*, regardless of these register settings. These registers, as well as the page size selection registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, are only writable from a process with PID 0 or 1.

Table 4.3-7. DPORT_DMMU_TABLE*n*_REG & DPORT_IMMU_TABLE*n*_REG

| [6:4] | Access rights for PID 2 ~ 7 |
|---|---|
| 0 | None of PIDs 2 ~ 7 have access. |
| 1 | All of PIDs 2 ~ 7 have access. |
| 2 | Only PID 2 has access. |
| 3 | Only PID 3 has access. |
| 4 | Only PID 4 has access. |
| 5 | Only PID 5 has access. |
| 6 | Only PID 6 has access. |
| 7 | Only PID 7 has access. |

| [3:0] | Address authority |
|---|---|
| 0x00 | Virtual page 0 accesses this physical page. |
| 0x01 | Virtual page 1 accesses this physical page. |
| 0x02 | Virtual page 2 accesses this physical page. |
| 0x03 | Virtual page 3 accesses this physical page. |
| 0x04 | Virtual page 4 accesses this physical page. |
| 0x05 | Virtual page 5 accesses this physical page. |
| 0x06 | Virtual page 6 accesses this physical page. |
| 0x07 | Virtual page 7 accesses this physical page. |
| 0x08 | Virtual page 8 accesses this physical page. |
| 0x09 | Virtual page 9 accesses this physical page. |
| 0x10 | Virtual page 10 accesses this physical page. |
| 0x11 | Virtual page 11 accesses this physical page. |
| 0x12 | Virtual page 12 accesses this physical page. |
| 0x13 | Virtual page 13 accesses this physical page. |
| 0x14 | Virtual page 14 accesses this physical page. |
| 0x15 | Virtual page 15 accesses this physical page. |

#### Differences Between SRAM0 and SRAM2 MMU

The memory governed by the SRAM0 MMU is accessed through the processors I-bus, while the processor accesses the memory governed by the SRAM2 MMU through the D-bus. Thus, the normal envisioned use is for the code to be stored in the SRAM0 MMU pages and data in the MMU pages of SRAM2. In general, applications running under a PID of 2 to 7 are not expected to modify their own code, because for these PIDs access to the MMU pages of SRAM0 is read-only. These applications must, however, be able to modify their data section, so that they are allowed to read as well as write MMU pages located in SRAM2. As stated before, processes running under PID 0 or 1 always have full read-and-write access to both memory ranges.

#### DMA MPU

Applications may want to configure the DMA to send data straight from or to the peripherals they can control. With access to DMA, a malicious process may also be able to copy data from or to a region it cannot normally access. In order to be secure against that scenario, there is a DMA MPU which can be used to disallow DMA transfers from memory regions with sensitive data in them.

For each 8 KB region in the SRAM1 and SRAM2 regions, there is a bit in the DPORT_AHB_MPU_TABLE_*n*_REG registers which tells the MPU to either allow or disallow DMA access to this region. The DMA MPU uses only these bits to decide if a DMA transfer can be started; the PID of the process is not a factor. This means that when the OS wants to restrict its processes in a heterogeneous fashion, it will need to re-load these registers with the values applicable to the process to be run on every context switch.

The register bits that govern access to the 8 KB regions are detailed in Table 4.3-8. When a register bit is set, DMA can read/write the corresponding 8 KB memory range. When the bit is cleared, access to that memory range is denied.

Table 4.3-8. MPU for DMA

| Size | Boundary address | | Authority | |
|---|---|---|---|---|
| | Low | High | Register | Bit |
| Internal SRAM 2 | | | | |
| 8 KB | 0x3FFA_E000 | 0x3FFA_FFFF | DPORT_AHB_MPU_TABLE_0_REG | 0 |
| 8 KB | 0x3FFB_0000 | 0x3FFB_1FFF | DPORT_AHB_MPU_TABLE_0_REG | 1 |
| 8 KB | 0x3FFB_2000 | 0x3FFB_3FFF | DPORT_AHB_MPU_TABLE_0_REG | 2 |
| 8 KB | 0x3FFB_4000 | 0x3FFB_5FFF | DPORT_AHB_MPU_TABLE_0_REG | 3 |
| 8 KB | 0x3FFB_6000 | 0x3FFB_7FFF | DPORT_AHB_MPU_TABLE_0_REG | 4 |
| 8 KB | 0x3FFB_8000 | 0x3FFB_9FFF | DPORT_AHB_MPU_TABLE_0_REG | 5 |
| 8 KB | 0x3FFB_A000 | 0x3FFB_BFFF | DPORT_AHB_MPU_TABLE_0_REG | 6 |
| 8 KB | 0x3FFB_C000 | 0x3FFB_DFFF | DPORT_AHB_MPU_TABLE_0_REG | 7 |
| 8 KB | 0x3FFB_E000 | 0x3FFB_FFFF | DPORT_AHB_MPU_TABLE_0_REG | 8 |
| 8 KB | 0x3FFC_0000 | 0x3FFC_1FFF | DPORT_AHB_MPU_TABLE_0_REG | 9 |
| 8 KB | 0x3FFC_2000 | 0x3FFC_3FFF | DPORT_AHB_MPU_TABLE_0_REG | 10 |
| 8 KB | 0x3FFC_4000 | 0x3FFC_5FFF | DPORT_AHB_MPU_TABLE_0_REG | 11 |
| 8 KB | 0x3FFC_6000 | 0x3FFC_7FFF | DPORT_AHB_MPU_TABLE_0_REG | 12 |
| 8 KB | 0x3FFC_8000 | 0x3FFC_9FFF | DPORT_AHB_MPU_TABLE_0_REG | 13 |
| 8 KB | 0x3FFC_A000 | 0x3FFC_BFFF | DPORT_AHB_MPU_TABLE_0_REG | 14 |
| 8 KB | 0x3FFC_C000 | 0x3FFC_DFFF | DPORT_AHB_MPU_TABLE_0_REG | 15 |
| 8 KB | 0x3FFC_E000 | 0x3FFC_FFFF | DPORT_AHB_MPU_TABLE_0_REG | 16 |
| 8 KB | 0x3FFD_0000 | 0x3FFD_1FFF | DPORT_AHB_MPU_TABLE_0_REG | 17 |
| 8 KB | 0x3FFD_2000 | 0x3FFD_3FFF | DPORT_AHB_MPU_TABLE_0_REG | 18 |
| 8 KB | 0x3FFD_4000 | 0x3FFD_5FFF | DPORT_AHB_MPU_TABLE_0_REG | 19 |
| 8 KB | 0x3FFD_6000 | 0x3FFD_7FFF | DPORT_AHB_MPU_TABLE_0_REG | 20 |
| 8 KB | 0x3FFD_8000 | 0x3FFD_9FFF | DPORT_AHB_MPU_TABLE_0_REG | 21 |
| 8 KB | 0x3FFD_A000 | 0x3FFD_BFFF | DPORT_AHB_MPU_TABLE_0_REG | 22 |
| 8 KB | 0x3FFD_C000 | 0x3FFD_DFFF | DPORT_AHB_MPU_TABLE_0_REG | 23 |
| 8 KB | 0x3FFD_E000 | 0x3FFD_FFFF | DPORT_AHB_MPU_TABLE_0_REG | 24 |
| Internal SRAM 1 | | | | |
| 8 KB | 0x3FFE_0000 | 0x3FFE_1FFF | DPORT_AHB_MPU_TABLE_0_REG | 25 |
| 8 KB | 0x3FFE_2000 | 0x3FFE_3FFF | DPORT_AHB_MPU_TABLE_0_REG | 26 |
| 8 KB | 0x3FFE_4000 | 0x3FFE_5FFF | DPORT_AHB_MPU_TABLE_0_REG | 27 |
| 8 KB | 0x3FFE_6000 | 0x3FFE_7FFF | DPORT_AHB_MPU_TABLE_0_REG | 28 |
| 8 KB | 0x3FFE_8000 | 0x3FFE_9FFF | DPORT_AHB_MPU_TABLE_0_REG | 29 |
| 8 KB | 0x3FFE_A000 | 0x3FFE_BFFF | DPORT_AHB_MPU_TABLE_0_REG | 30 |
| 8 KB | 0x3FFE_C000 | 0x3FFE_DFFF | DPORT_AHB_MPU_TABLE_0_REG | 31 |
| 8 KB | 0x3FFE_E000 | 0x3FFE_FFFF | DPORT_AHB_MPU_TABLE_1_REG | 0 |
| 8 KB | 0x3FFF_0000 | 0x3FFF_1FFF | DPORT_AHB_MPU_TABLE_1_REG | 1 |
| 8 KB | 0x3FFF_2000 | 0x3FFF_3FFF | DPORT_AHB_MPU_TABLE_1_REG | 2 |
| 8 KB | 0x3FFF_4000 | 0x3FFF_5FFF | DPORT_AHB_MPU_TABLE_1_REG | 3 |
| 8 KB | 0x3FFF_6000 | 0x3FFF_7FFF | DPORT_AHB_MPU_TABLE_1_REG | 4 |
| 8 KB | 0x3FFF_8000 | 0x3FFF_9FFF | DPORT_AHB_MPU_TABLE_1_REG | 5 |
| 8 KB | 0x3FFF_A000 | 0x3FFF_BFFF | DPORT_AHB_MPU_TABLE_1_REG | 6 |

| Size | Boundary address | | Authority | |
|---|---|---|---|---|
| | Low | High | Register | Bit |
| 8 KB | 0x3FFF_C000 | 0x3FFF_DFFF | DPORT_AHB_MPU_TABLE_1_REG | 7 |
| 8 KB | 0x3FFF_E000 | 0x3FFF_FFFF | DPORT_AHB_MPU_TABLE_1_REG | 8 |

Registers DPROT_AHB_MPU_TABLE_0_REG and DPROT_AHB_MPU_TABLE_1_REG are located in the DPort address space. Only processes with a PID of 0 or 1 can modify these two registers.

**Note:**
In hardware, there are three instruction buses corresponding to $VAddr_1$, $VAddr_2$ and $VAddr_3$, respectively. These three buses can initiate load or fetch accesses simultaneously, but only one access is true. If more than one unmasked instruction buses are present, then bit8 of all MMU entries should be set to zero. Otherwise, when an invalid MMU entry is used by an access, the cache will be stalled even if there is no program at this access.

#### 4.3.2.2 External Memory

Accesses to the external flash and external SPI RAM are done through a cache and are also handled by an MMU. This Cache MMU can apply different mappings, depending on the PID of the process as well as the CPU the process is running on. The MMU does this in a way that is similar to the internal memory MMU, that is, for every page of virtual memory, it has a register detailing which physical page this virtual page should map to. There are differences between the MMUs governing the internal memory and the Cache MMU, though. First of all, the Cache MMU has a fixed page size (which is 64 KB for external flash and 32 KB for external RAM) and secondly, instead of specifying access rights in the MMU entries, the Cache MMU has explicit mapping tables for each PID and processor core. The MMU mapping configuration registers will be referred to as 'entries' in the rest of this chapter. These registers are only accessible from processes with a PID of 0 or 1; processes with a PID of 2 to 7 will have to delegate to one of the above-mentioned processes to change their MMU settings.

The MMU entries, as stated before, are used for mapping a virtual memory page access to a physical memory page access. The MMU controls five regions of virtual address space, detailed in Table 4.3-9. $VAddr_1$ to $VAddr_4$ are used for accessing external flash, whereas $VAddr_{\text{RAM}}$ is used for accessing external RAM. Note that $VAddr_4$ is a subset of $VAddr_0$.

Table 4.3-9. Virtual Address for External Memory

| Name | Size | Boundary address | | Page quantity |
|---|---|---|---|---|
| | | Low | High | |
| $VAddr_0$ | 4 MB | 0x3F40_0000 | 0x3F7F_FFFF | 64 |
| $VAddr_1$ | 4 MB | 0x4000_0000 | 0x403F_FFFF | 64* |
| $VAddr_2$ | 4 MB | 0x4040_0000 | 0x407F_FFFF | 64 |
| $VAddr_3$ | 4 MB | 0x4080_0000 | 0x40BF_FFFF | 64 |
| $VAddr_4$ | 1 MB | 0x3F40_0000 | 0x3F4F_FFFF | 16 |
| $VAddr_{\text{RAM}}$ | 4 MB | 0x3F80_0000 | 0x3FBF_FFFF | 128 |

* The configuration entries for address range 0x4000_0000 ~ 0x403F_FFFF are implemented and documented as if it were a full 4 MB address range, but it is not accessible as such. Instead, the address range 0x4000_0000 ~ 0x400C_1FFF accesses on-chip memory. This means that some of the configuration entries for $VAddr_1$ will not be used.

### External Flash

For flash, the relationships among entry numbers, virtual memory ranges, and PIDs are detailed in Tables 4.3-10 and 4.3-11, which for every memory region and PID combination specify the first MMU entry governing the mapping. This number refers to the MMU entry governing the very first page; the entire region is described by the amount of pages specified in the 'count' column.

These two tables are essentially the same, with the sole difference being that the APP_CPU entry numbers are 2048 higher than the corresponding PRO_CPU numbers. Note that memory regions $VAddr_0$ and $VAddr_1$ are only accessible using PID 0 and 1, while $VAddr_1$ can only be accessed by PID 2 ~ 7.

Table 4.3-10. MMU Entry Numbers for PRO_CPU

| VAddr | Count | First MMU entry for PID | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0/1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $VAddr_0$ | 64 | 0 | - | - | - | - | - | - |
| $VAddr_1$ | 64 | 64 | - | - | - | - | - | - |
| $VAddr_2$ | 64 | 128 | 256 | 384 | 512 | 640 | 768 | 896 |
| $VAddr_3$ | 64 | 192 | 320 | 448 | 576 | 704 | 832 | 960 |
| $VAddr_4$ | 16 | - | 1056 | 1072 | 1088 | 1104 | 1120 | 1136 |

Table 4.3-11. MMU Entry Numbers for APP_CPU

| VAddr | Count | First MMU entry for PID | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0/1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $VAddr_0$ | 64 | 2048 | - | - | - | - | - | - |
| $VAddr_1$ | 64 | 2112 | - | - | - | - | - | - |
| $VAddr_2$ | 64 | 2176 | 2304 | 2432 | 2560 | 2688 | 2816 | 2944 |
| $VAddr_3$ | 64 | 2240 | 2368 | 2496 | 2624 | 2752 | 2880 | 3008 |
| $VAddr_4$ | 16 | - | 3104 | 3120 | 3136 | 3152 | 3168 | 3184 |

As these tables show, virtual address $VAddr_1$ can only be used by processes with a PID of 0 or 1. There is a

special mode to allow processes with a PID of 2 to 7 to read the External Flash via address $VAddr_1$. When the DPORT_PRO_SINGLE_IRAM_ENA bit of register DPORT_PRO_CACHE_CTRL_REG is 1, the MMU enters this special mode for PRO_CPU memory accesses. Similarly, when the DPORT_APP_SINGLE_IRAM_ENA bit of register DPORT_APP_CACHE_CTRL_REG is 1, the APP_CPU accesses memory using this special mode. In this mode, the process and virtual address page supported by each configuration entry of MMU are different. For details please see Table 4.3-12 and 4.3-13. As shown in these tables, in this special mode $VAddr_2$ and $VAddr_3$ cannot be used to access External Flash.

Table 4.3-12. MMU Entry Numbers for PRO_CPU (Special Mode)

| VAddr | Count | First MMU entry for PID | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0/1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $VAddr_0$ | 64 | 0 | - | - | - | - | - | - |
| $VAddr_1$ | 64 | 64 | 256 | 384 | 512 | 640 | 768 | 896 |
| $VAddr_2$ | 64 | - | - | - | - | - | - | - |
| $VAddr_3$ | 64 | - | - | - | - | - | - | - |
| $VAddr_4$ | 16 | - | 1056 | 1072 | 1088 | 1104 | 1120 | 1136 |

Table 4.3-13. MMU Entry Numbers for APP_CPU (Special Mode)

| VAddr | Count | First MMU entry for PID | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0/1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $VAddr_0$ | 64 | 2048 | - | - | - | - | - | - |
| $VAddr_1$ | 64 | 2112 | 2304 | 2432 | 2560 | 2688 | 2816 | 2944 |
| $VAddr_2$ | 64 | - | - | - | - | - | - | - |
| $VAddr_3$ | 64 | - | - | - | - | - | - | - |
| $VAddr_4$ | 16 | - | 3104 | 3120 | 3136 | 3152 | 3168 | 3184 |

Every configuration entry of MMU maps a virtual address page of a CPU process to a physical address page. An entry is 32 bits wide. Of these, bits 0~7 indicate the physical page the virtual page is mapped to. Bit 8 should be cleared to indicate that the MMU entry is valid; entries with this bit set will not map any physical address to the virtual address. Bits 10 to 32 are unused and should be written as zero. Because there are eight address bits in an MMU entry, and the page size for external flash is 64 KB, a maximum of 256 × 64 KB = 16 MB of external flash is supported.

### Examples

Example 1. A PRO_CPU process, with a PID of 1, needs to read external flash address 0x07_2375 via virtual address 0x3F70_2375. The MMU is not in the special mode.

- According to Table 4.3-9, virtual address 0x3F70_2375 resides in the 0x30'th page of .$VAddr_0$
- According to Table 4.3-10, the MMU entry for $VAddr_0$ for PID 0/1 for the PRO_CPU starts at 0.
- The modified MMU entry is 0 + 0x30 = 0x30.
- Address 0x07_2375 resides in the 7'th 64 KB-sized page.
- MMU entry 0x30 needs to be set to 7 and marked as valid by setting the 8'th bit to 0. Thus, 0x007 is written to MMU entry 0x30.

Example 2. An APP_CPU process, with a PID of 4, needs to read external flash address 0x44_048C via virtual address 0x4044_048C. The MMU is not in special mode.

- According to Table 4.3-9, virtual address 0x4044_048C resides in the 0x4'th page of $VAddr_2$.
- According to Table 4.3-11, the MMU entry for $VAddr_2$ for PID 4 for the APP_CPU starts at 2560.
- The modified MMU entry is 2560 + 0x4 = 2564.
- Address 0x44_048C resides in the 0x44'th 64 KB-sized page.
- MMU entry 2564 needs to be set to 0x44 and marked as valid by setting the 8'th bit to 0. Thus, 0x044 is written to MMU entry 2564.

### External RAM

Processes running on PRO_CPU and APP_CPU can read and write External SRAM via the Cache at virtual address range $VAddr_{\text{RAM}}$, which is 0x3F80_0000 ~ 0x3FBF_FFFF. As with the flash MMU, the address space and the physical memory are divided into pages. For the External RAM MMU, the page size is 32 KB and the MMU is able to map 256 physical pages into the virtual address space, allowing for 32 KB × 256 = 8 MB of physical external RAM to be mapped.

The mapping of virtual pages into this memory range depends on the mode this MMU is in: Low-High mode, Even-Odd mode, or Normal mode. In all cases, the DPORT_PRO_DRAM_HL bit and DPORT_PRO_DRAM_SPLIT bit in register DPORT_PRO_CACHE_CTRL_REG, the DPORT_APP_DRAM_HL bit and DPORT_APP_DRAM_SPLIT bit in register DPORT_APP_CACHE_CTRL_REG determine the virtual address mode for External SRAM. For details, please see Table 4.3-14. If a different mapping for the PRO_CPU and APP_CPU is required, the Normal Mode should be selected, as it is the only mode that can provide this. If it is allowable for the PRO_CPU and the APP_CPU to share the same mapping, using either High-Low or Even-Odd mode can give a speed gain when both CPUs access memory frequently.

In case the APP_CPU cache is disabled, which renders the region of 0x4007_8000 to 0x4007_FFFF usable as normal internal RAM, the usability of the various cache modes changes. Normal mode will allow PRO_CPU access to external RAM to keep functioning, but the APP_CPU will be unable to access the external RAM. High-Low mode allows both CPUs to use external RAM, but only for the 2 MB virtual memory addresses from 0x3F80_0000 to 0x3F9F_FFFF. It is not advised to use Even-Odd mode with the APP_CPU cache region disabled.

**Table 4.3-14. Virtual Address Mode for External SRAM**

| Mode | DPORT_PRO_DRAM_HL<br>DPORT_APP_DRAM_HL | DPORT_PRO_DRAM_SPLIT<br>DPORT_APP_DRAM_SPLIT |
|---|---|---|
| Low-High | 1 | 0 |
| Even-Odd | 0 | 1 |
| Normal | 0 | 0 |

In normal mode, the virtual-to-physical page mapping can be different for both CPUs. Page mappings for PRO_CPU are set using the MMU entries for $^{L}VAddr_{\text{RAM}}$, and page mappings for the APP_CPU can be configured using the MMU entries for $^{R}VAddr_{\text{RAM}}$. In this mode, all 128 pages of both $^{L}VAddr$ and $^{R}VAddr$ are fully used, allowing a maximum of 8 MB of memory to be mapped; 4 MB into PRO_CPU address space and a possibly different 4 MB into the APP_CPU address space, as can be seen in Table 4.3-15.

Table 4.3-15. Virtual Address for External SRAM ( Normal Mode )

| Virtual address | Size | PRO_CPU address | |
|---|---|---|---|
| | | Low | High |
| $^{L}VAddr_{RAM}$ | 4 MB | 0x3F80_0000 | 0x3FBF_FFFF |
| Virtual address | Size | APP_CPU address | |
| | | Low | High |
| $^{R}VAddr_{RAM}$ | 4 MB | 0x3F80_0000 | 0x3FBF_FFFF |

In Low-High mode, both the PRO_CPU and the APP_CPU use the same mapping entries. In this mode $^{L}VAddr_{RAM}$ is used for the lower 2 MB of the virtual address space, while $^{R}VAddr_{RAM}$ is used for the upper 2 MB. This also means that the upper 64 MMU entries for $^{L}VAddr_{RAM}$, as well as the lower 64 entries for $^{R}VAddr_{RAM}$, are unused. Table 4.3-16 details these address ranges.

Table 4.3-16. Virtual Address for External SRAM ( Low-High Mode )

| Virtual address | Size | PRO_CPU/APP_CPU address | |
|---|---|---|---|
| | | Low | High |
| $^{L}VAddr_{RAM}$ | 2 MB | 0x3F80_0000 | 0x3F9F_FFFF |
| $^{R}VAddr_{RAM}$ | 2 MB | 0x3FA0_0000 | 0x3FBF_FFFF |

In Even-Odd memory, the VRAM is split into 32-byte chunks. The even chunks are resolved through the MMU entries for $^{L}VAddr_{RAM}$, the odd chunks through the entries for $^{R}VAddr_{RAM}$. Generally, the MMU entries for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ are set to the same values, so that the virtual pages map to a contiguous region of physical memory. Table 4.3-17 details this mode.

Table 4.3-17. Virtual Address for External SRAM (Even–Odd Mode)

| Virtual address | Size | PRO_CPU/APP_CPU address | |
|---|---|---|---|
| | | Low | High |
| $^{L}VAddr_{RAM}$ | 32 Bytes | 0x3F80_0000 | 0x3F80_001F |
| $^{R}VAddr_{RAM}$ | 32 Bytes | 0x3F80_0020 | 0x3F80_003F |
| $^{L}VAddr_{RAM}$ | 32 Bytes | 0x3F80_0040 | 0x3F80_005F |
| $^{R}VAddr_{RAM}$ | 32 Bytes | 0x3F80_0060 | 0x3F80_007F |
| ... | | | |
| $^{L}VAddr_{RAM}$ | 32 Bytes | 0x3FBF_FFC0 | 0x3FBF_FFDF |
| $^{R}VAddr_{RAM}$ | 32 Bytes | 0x3FBF_FFE0 | 0x3FBF_FFFF |

The bit configuration of the External RAM MMU entries is the same as for the flash memory: the entries are 32-bit registers, with the lower nine bits being used. Bits 0~7 contain the physical page the entry should map its associate virtual page address to, while bit 8 is cleared when the entry is valid and set when it is not. Table 4.3-18 details the first MMU entry number for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ for all PIDs.

**Table 4.3-18. MMU Entry Numbers for External RAM**

| VAddr | Count | First MMU entry for PID | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0/1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $^{L}VAddr_{RAM}$ | 128 | 1152 | 1280 | 1408 | 1536 | 1664 | 1792 | 1920 |
| $^{R}VAddr_{RAM}$ | 128 | 3200 | 3328 | 3456 | 3584 | 3712 | 3840 | 3968 |

### Examples

Example 1. A PRO_CPU process, with a PID of 7, needs to read or write external RAM address 0x7F_A375 via virtual address 0x3FA7_2375. The MMU is in Low-High mode.

- According to Table 4.3-9, virtual address 0x3FA7_2375 resides in the 0x4E'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 4.3-16, virtual address 0x3FA7_2375 is governed by $^{R}VAddr_{RAM}$.
- According to Table 4.3-18, the MMU entry for $^{R}VAddr_{RAM}$ for PID 7 for the PRO_CPU starts at 3968.
- The modified MMU entry is 3968 + 0x4E = 4046.
- Address 0x7F_A375 resides in the 255'th 32 KB-sized page.
- MMU entry 4046 needs to be set to 255 and marked as valid by clearing the 8'th bit. Thus, 0x0FF is written to MMU entry 4046.

Example 2. An APP_CPU process, with a PID of 5, needs to read or write external RAM address 0x55_5805 up to 0x55_5823 starting at virtual address 0x3F85_5805. The MMU is in Even-Odd mode.

- According to Table 4.3-9, virtual address 0x3F85_5805 resides in the 0x0A'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 4.3-17, the range to be read/written spans both a 32-byte region in $^{R}VAddr_{RAM}$ and $^{L}VAddr_{RAM}$.
- According to Table 4.3-18, the MMU entry for $^{L}VAddr_{RAM}$ for PID 5 starts at 1664.
- According to Table 4.3-18, the MMU entry for $^{R}VAddr_{RAM}$ for PID 5 starts at 3712.
- The modified MMU entries are 1664 + 0x0A = 1674 and 3712 + 0x0A = 3722.
- The addresses 0x55_5805 to 0x55_5823 reside in the 0xAA'th 32 KB-sized page.
- MMU entries 1674 and 3722 need to be set to 0xAA and marked as valid by setting the 8'th bit to 0. Thus, 0x0AA is written to MMU entries 1674 and 3722. This mapping applies to both the PRO_CPU and the APP_CPU.

Example 3. A PRO_CPU process, with a PID of 1, and an APP_CPU process whose PID is also 1, need to read or write external RAM using virtual address 0x3F80_0876. The PRO_CPU needs this region to access physical address 0x10_0876, while the APP_CPU wants to access physical address 0x20_0876 through this virtual address. The MMU is in Normal mode.

- According to Table 4.3-9, virtual address 0x3F80_0876 resides in the 0'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 4.3-18, the MMU entry for PID 1 for the PRO_CPU starts at 1152.
- According to Table 4.3-18, the MMU entry for PID 1 for the APP_CPU starts at 3200.
- The MMU entries that are modified are 1152 + 0 = 1152 for the PRO_CPU and 3200 + 0 = 3200 for the APP_CPU.
- Address 0x10_0876 resides in the 0x20'th 32 KB-sized page.
- Address 0x20_0876 resides in the 0x40'th 32 KB-sized page.
- For the PRO_CPU, MMU entry 1152 needs to be set to 0x20 and marked as valid by clearing the 8'th bit. Thus, 0x020 is written to MMU entry 1152.

- For the APP_CPU, MMU entry 3200 needs to be set to 0x40 and marked as valid by clearing the 8'th bit. Thus, 0x040 is written to MMU entry 3200.
- Now, the PRO_CPU and the APP_CPU can access different physical memory regions through the same virtual address.

#### 4.3.2.3 Peripheral

The Peripheral MPU manages the 39 peripheral modules. This MMU can be configured per peripheral to only allow access from a process with a certain PID. The registers to configure this are detailed in Table 4.3-19.

**Table 4.3-19. MPU for Peripheral**

| Peripheral | Authority | |
|---|---|---|
| | PID = 0/1 | PID = 2 ~ 7 |
| DPort Register | Access | Forbidden |
| AES Accelerator | Access | Forbidden |
| RSA Accelerator | Access | Forbidden |
| SHA Accelerator | Access | Forbidden |
| Secure Boot | Access | Forbidden |
| Cache MMU Table | Access | Forbidden |
| PID Controller | Access | Forbidden |
| UART0 | Access | DPORT_AHBLITE_MPU_TABLE_UART_REG |
| SPI1 | Access | DPORT_AHBLITE_MPU_TABLE_SPI1_REG |
| SPI0 | Access | DPORT_AHBLITE_MPU_TABLE_SPI0_REG |
| GPIO | Access | DPORT_AHBLITE_MPU_TABLE_GPIO_REG |
| RTC | Access | DPORT_AHBLITE_MPU_TABLE_RTC_REG |
| IO MUX | Access | DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG |
| SDIO Slave | Access | DPORT_AHBLITE_MPU_TABLE_HINF_REG |
| UDMA1 | Access | DPORT_AHBLITE_MPU_TABLE_UHCI1_REG |
| I2S0 | Access | DPORT_AHBLITE_MPU_TABLE_I2S0_REG |
| UART1 | Access | DPORT_AHBLITE_MPU_TABLE_UART1_REG |
| I2C0 | Access | DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG |
| UDMA0 | Access | DPORT_AHBLITE_MPU_TABLE_UHCI0_REG |
| SDIO Slave | Access | DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG |
| RMT | Access | DPORT_AHBLITE_MPU_TABLE_RMT_REG |
| PCNT | Access | DPORT_AHBLITE_MPU_TABLE_PCNT_REG |
| SDIO Slave | Access | DPORT_AHBLITE_MPU_TABLE_SLC_REG |
| LED PWM | Access | DPORT_AHBLITE_MPU_TABLE_LEDC_REG |
| Efuse Controller | Access | DPORT_AHBLITE_MPU_TABLE_EFUSE_REG |
| Flash Encryption | Access | DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG |
| PWM0 | Access | DPORT_AHBLITE_MPU_TABLE_PWM0_REG |
| TIMG0 | Access | DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG |
| TIMG1 | Access | DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG |
| SPI2 | Access | DPORT_AHBLITE_MPU_TABLE_SPI2_REG |
| SPI3 | Access | DPORT_AHBLITE_MPU_TABLE_SPI3_REG |

| Peripheral | Authority | |
|---|---|---|
| | PID = 0/1 | PID = 2 ~ 7 |
| SYSCON | Access | DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG |
| I2C1 | Access | DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG |
| SDMMC | Access | DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG |
| EMAC | Access | DPORT_AHBLITE_MPU_TABLE_EMAC_REG |
| PWM1 | Access | DPORT_AHBLITE_MPU_TABLE_PWM1_REG |
| I2S1 | Access | DPORT_AHBLITE_MPU_TABLE_I2S1_REG |
| UART2 | Access | DPORT_AHBLITE_MPU_TABLE_UART2_REG |
| RNG | Access | DPORT_AHBLITE_MPU_TABLE_PWR_REG |

Each bit of register DPORT_AHBLITE_MPU_TABLE_*X*_REG determines whether each process can access the peripherals managed by the register. For details please see Table 4.3-20. When a bit of register DPORT_AHBLITE_
MPU_TABLE_*X*_REG is 1, it means that a process with the corresponding PID can access the corresponding peripheral of the register. Otherwise, the process cannot access the corresponding peripheral.

**Table 4.3-20. DPORT_AHBLITE_MPU_TABLE_*X*_REG**

| PID | 2 3 4 5 6 7 |
|---|---|
| DPORT_AHBLITE_MPU_TABLE_*X*_REG bit | 0 1 2 3 4 5 |

All the DPORT_AHBLITE_MPU_TABLE_*X*_REG registers are in peripheral DPort Register. Only processes with PID 0/1 can modify these registers.

# Chapter 5

# eFuse Controller (EFUSE)

## 5.1 Introduction

The ESP32 has a number of eFuses which store system parameters. Fundamentally, an eFuse is a single bit of non-volatile memory with the restriction that once an eFuse bit is programmed to 1, it can never be reverted to 0. Software can instruct the eFuse Controller to program each bit for each system parameter as needed.

Some of these system parameters can be read by software using the eFuse Controller. Some of the system parameters are also directly used by hardware modules.

## 5.2 Features

- Configuration of 33 system parameters
- Optional write-protection
- Optional software-read-protection

## 5.3 Functional Description

### 5.3.1 Structure

Thirty-three system parameters with different bit width are stored in the eFuses. The name of each system parameter and the corresponding bit width are shown in Table 5.3-1. Among those parameters, efuse_wr_disable, efuse_rd_disable, BLK3_part_reserve and coding_scheme are directly used by the eFuse Controller.

Table 5.3-1. System Parameters

| Name | Bit width | Program -Protection by efuse_wr_disable | Software-Read -Protection by efuse_rd_disable | Description |
|---|---|---|---|---|
| **efuse_wr_disable** | 16 | 1 | - | controls the eFuse Controller |
| **efuse_rd_disable** | 4 | 0 | - | controls the eFuse Controller |
| flash_crypt_cnt | 7 | 2 | - | governs the flash encryption/ decryption |
| WIFI_MAC_Address | 56 | 3 | - | Wi-Fi MAC address and CRC |
| SPI_pad_config_hd | 5 | 3 | - | configures the SPI I/O to a certain pad |

| Name | Bit width | Program -Protection by efuse_wr_disable | Software-Read -Protection by efuse_rd_disable | Description |
|---|---|---|---|---|
| XPD_SDIO_REG | 1 | 5 | - | powers up the flash regulator |
| SDIO_TIEH | 1 | 5 | - | configures the flash regulator voltage: set to 1 for 3.3 V and set to 0 for 1.8 V |
| sdio_force | 1 | 5 | - | determines whether XPD_SDIO_REG and SDIO_TIEH can control the flash regulator |
| **BLK3_part_reserve** | 2 | 10 | 3 | controls the eFuse controller |
| SPI_pad_config_clk | 5 | 6 | - | configures the SPI I/O to a certain pad |
| SPI_pad_config_q | 5 | 6 | - | configures the SPI I/O to a certain pad |
| SPI_pad_config_d | 5 | 6 | - | configures the SPI I/O to a certain pad |
| SPI_pad_config_cs0 | 5 | 6 | - | configures the SPI I/O to a certain pad |
| flash_crypt_config | 4 | 10 | 3 | governs flash encryption/ decryption |
| **coding_scheme*** | 2 | 10 | 3 | controls the eFuse Controller |
| console_debug_disable | 1 | 15 | - | disables the ROM BASIC debug console fallback mode when set to 1 |
| abstract_done_0 | 1 | 12 | - | determines the status of Secure Boot |
| abstract_done_1 | 1 | 13 | - | determines the status of Secure Boot |
| JTAG_disable | 1 | 14 | - | disables access to the JTAG controllers so as to effectively disable external use of JTAG |
| download_dis_encrypt | 1 | 15 | - | governs flash encryption/ decryption |
| download_dis_decrypt | 1 | 15 | - | governs flash encryption/ decryption |
| download_dis_cache | 1 | 15 | - | disables cache when boot mode is the Download Mode |
| key_status | 1 | 10 | 3 | determines whether BLOCK3 is deployed for user purposes |
| BLOCK1* | 256/192/128 | 7 | 0 | governs flash encryption/ decryption |

| Name | Bit width | Program -Protection by efuse_wr_disable | Software-Read -Protection by efuse_rd_disable | Description |
|---|---|---|---|---|
| BLOCK2* | 256/192/128 | 8 | 1 | key for Secure Boot |
| BLOCK3* | 256/192/128 | 9 | 2 | key for user purposes |
| disable_app_cpu | 1 | 3 | - | disables APP CPU |
| disable_bt | 1 | 3 | - | disables Bluetooth |
| pkg_version | 4 | 3 | - | packaging version |
| disable_cache | 1 | 3 | - | disables cache |
| CK8M Frequency | 8 | 4 | - | RC_FAST_CLK frequency |
| vol_level_hp_inv | 2 | 3 | - | stores the voltage level for CPU to run at 240 MHz, or for flash/PSRAM to run at 80 MHz |
| dig_vol_l6 | 4 | 11 | - | stores the difference between the digital regulator voltage at level 6 and 1.2 V. |
| uart_download_dis | 1 | 2 | - | permanently disables Download Boot mode when set to 1. Valid only for ESP32 ECO V3. |

#### 5.3.1.1 System Parameter efuse_wr_disable

The system parameter efuse_wr_disable determines whether all of the system parameters are write-protected. Since efuse_wr_disable is a system parameter as well, it also determines whether itself is write-protected.

If a system parameter is not write-protected, its unprogrammed bits can be programmed from 0 to 1. The bits previously programmed to 1 will remain 1. When a system parameter is write-protected, none of its bits can be programmed: The unprogrammed bits will always remain 0 and the programmed bits will always remain 1.

The write-protection status of each system parameter corresponds to a bit in efuse_wr_disable. When the corresponding bit is set to 0, the system parameter is not write-protected. When the corresponding bit is set to 1, the system parameter is write-protected. If a system parameter is already write-protected, it will remain write-protected. The column entitled "Program-Protection by efuse_wr_disable" in Table 5.3-1 lists the corresponding bits that determine the write-protection status of each system parameter.

#### 5.3.1.2 System Parameter efuse_rd_disable

Of the 33 system parameters, 27 are not constrained by software-read-protection. These are marked by "-" in the column entitled "Software-Read-Protection by efuse_rd_disable" in Table 5.3-1. Those system parameters, some of which are used by software and hardware modules at the same time, can be read by software via the eFuse Controller at any time.

When not software-read-protected, the other six system parameters can both be read by software and used by hardware modules. When they are software-read-protected, they can only be used by the hardware modules.

The column "Software-Read-Protection by efuse_rd_disable" in Table 5.3-1 lists the corresponding bits in efuse_rd
_disable that determine the software read-protection status of the six system parameters. If a bit in the system parameter efuse_rd_disable is 0, the system parameter controlled by the bit is not software-read-protected. If a bit in the system parameter efuse_rd_disable is 1, the system parameter controlled by the bit is software-read-protected. If a system parameter is software-read-protected, it will remain in this state.

#### 5.3.1.3 System Parameter coding_scheme

As Table 5.3-1 shows, only three system parameters, BLOCK1, BLOCK2, and BLOCK3, have variable bit widths. Their bit widths are controlled by another system parameter, coding_scheme. Despite their variable bit widths, BLOCK1, BLOCK2, and BLOCK3 are assigned a fixed number of bits in eFuse. There is an encoding mapping between these three system parameters and their corresponding stored values in eFuse. For details please see Table 5.3-2.

Table 5.3-2. BLOCK1/2/3 Encoding

| coding_scheme[1:0] | Width of BLOCK1/2/3 | Coding scheme | Number of bits in eFuse |
|---|---|---|---|
| 00/11 | 256 | None | 256 |
| 01 | 192 | 3/4 | 256 |
| 10 | 128 | Repeat | 256 |

The three coding schemes are explained as follows:

- $BLOCKN$ represents any of the following three system parameters: BLOCK1, BLOCK2 or BLOCK3.
- $BLOCKN[255:0]$, $BLOCKN[191:0]$, and $BLOCKN[127:0]$ represent each bit of the three system parameters in the three encoding schemes.
- $^{e}$BLOCKN$[255{:}0]$ represents each corresponding bit of those system parameters in eFuse after being encoded.

**None**

$$^{e}BLOCKN[255{:}0] = BLOCKN[255{:}0]$$

**3/4**

$$BLOCKN_i^j[7:0] = BLOCKN[48i+8j+7:48i+8j] \qquad i \in \{0,1,2,3\} \qquad j \in \{0,1,2,3,4,5\}$$
$$^{e}BLOCKN_i^j[7:0] = {}^{e}BLOCKN[64i+8j+7:64i+8j] \qquad i \in \{0,1,2,3\} \qquad j \in \{0,1,2,3,4,5,6,7\}$$

For $i \in \{0,1,2,3\}$,

$$BLOCKN_i^j[7:0] = \begin{cases} BLOCKN_i^j[7:0] & j \in \{0,1,2,3,4,5\} \\ BLOCKN_i^0[7:0] \oplus BLOCKN_i^1[7:0] & \\ \quad BLOCKN_i^2[7:0] \oplus BLOCKN_i^3[7:0] & \\ \quad BLOCKN_i^4[7:0] \oplus BLOCKN_i^5[7:0] & j = 6 \\ \sum_{l=0}^{5}(l+1)\sum_{k=0}^{7} BLOCKN_i^l[k], & j = 7 \end{cases}$$

$\oplus$ denotes bitwise XOR

$\sum$ and $+$ denote addition

**Repeat**

$$^{e}BLOCKN[255:128] = {}^{e}BLOCKN[127:0] = BLOCKN[127:0]$$

#### 5.3.1.4 BLK3_part_reserve

System parameters coding_scheme, BLOCK1, BLOCK2, and BLOCK3 are controlled by the parameter BLK3_part_reserve.

When the value of BLK3_part_reserve is 0, coding_scheme, BLOCK1, BLOCK2, and BLOCK3 can be set to any value.

When the value of BLK3_part_reserve is 1, coding_scheme, BLOCK1, BLOCK2 and BLOCK3 are controlled by 3/4 coding scheme. Meanwhile, BLOCK3$[143:96]$, namely, $^{e}$BLOCK3$[191:128]$ is unavailable.

### 5.3.2 Programming of System Parameters

The programming of variable-length system parameters BLOCK1, BLOCK2, and BLOCK3 is different from that of the fixed-length system parameters. **We program the $^{e}$BLOCKN$[255:0]$ value of encoded system parameters BLOCK1, BLOCK2, and BLOCK3 instead of directly programming the system parameters. The bit width of $^{e}$BLOCKN$[255:0]$ is always 256.** Fixed-length system parameters, in contrast, are programmed without encoding them first.

Each bit of the 30 fixed-length system parameters and the three encoded variable-length system parameters corresponds to a program register bit, as shown in Table 5.3-3. The register bits will be used when programming system parameters.

**Table 5.3-3. Program Registers**

| System parameter | | | Register | |
|---|---|---|---|---|
| Name | Width | Bit | Name | Bit |
| efuse_wr_disable | 16 | [15:0] | EFUSE_BLK0_WDATA0_REG | [15:0] |
| efuse_rd_disable | 4 | [3:0] | | [19:16] |
| flash_crypt_cnt | 7 | [6:0] | | [26:20] |
| uart_download_dis | 1 | [0] | | [27] |
| WIFI_MAC_Address | 56 | [31:0] | EFUSE_BLK0_WDATA1_REG | [31:0] |
| | | [55:32] | EFUSE_BLK0_WDATA2_REG | [23:0] |
| disable_app_cpu | 1 | [0] | EFUSE_BLK0_WDATA3_REG | [0] |
| disable_bt | 1 | [0] | | [1] |
| pkg_version | 4 | [3:0] | | [2], [11:9] |
| disable_cache | 1 | [0] | | [3] |
| SPI_pad_config_hd | 5 | [4:0] | | [8:4] |
| BLK3_part_reserve | 1 | [0] | | [14] |
| CK8M Frequency | 8 | [7:0] | EFUSE_BLK0_WDATA4_REG | [7:0] |
| XPD_SDIO_REG | 1 | [0] | | [14] |
| SDIO_TIEH | 1 | [0] | | [15] |
| sdio_force | 1 | [0] | | [16] |
| SPI_pad_config_clk | 5 | [4:0] | EFUSE_BLK0_WDATA5_REG | [4:0] |
| SPI_pad_config_q | 5 | [4:0] | | [9:5] |

| System parameter | | | Register | |
|---|---|---|---|---|
| Name | Width | Bit | Name | Bit |
| SPI_pad_config_d | 5 | [4:0] | | [14:10] |
| SPI_pad_config_cs0 | 5 | [4:0] | | [19:15] |
| vol_level_hp_inv | 2 | [1:0] | | [23:22] |
| dig_vol_l6 | 4 | [3:0] | | [27:24] |
| flash_crypt_config | 4 | [3:0] | | [31:28] |
| coding_scheme | 2 | [1:0] | EFUSE_BLK0_WDATA6_REG | [1:0] |
| console_debug_disable | 1 | [0] | | [2] |
| abstract_done_0 | 1 | [0] | | [4] |
| abstract_done_1 | 1 | [0] | | [5] |
| JTAG_disable | 1 | [0] | | [6] |
| download_dis_encrypt | 1 | [0] | | [7] |
| download_dis_decrypt | 1 | [0] | | [8] |
| download_dis_cache | 1 | [0] | | [9] |
| key_status | 1 | [0] | | [10] |
| BLOCK1 | 256/192/128 | [31:0] | EFUSE_BLK1_WDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK1_WDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK1_WDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK1_WDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK1_WDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK1_WDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK1_WDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK1_WDATA7_REG | [31:0] |
| BLOCK2 | 256/192/128 | [31:0] | EFUSE_BLK2_WDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK2_WDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK2_WDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK2_WDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK2_WDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK2_WDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK2_WDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK2_WDATA7_REG | [31:0] |
| BLOCK3 | 256/192/128 | [31:0] | EFUSE_BLK3_WDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK3_WDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK3_WDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK3_WDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK3_WDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK3_WDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK3_WDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK3_WDATA7_REG | [31:0] |

The process of programming system parameters is as follows:

1. Configure EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF.
2. Set the corresponding register bit of the system parameter bit to be programmed to 1.

3. Write 0x5A5A into register EFUSE_CONF.
4. Write 0x2 into register EFUSE_CMD.
5. Poll register EFUSE_CMD until it is 0x0, or wait for a program-done interrupt.
6. Write 0x5AA5 into register EFUSE_CONF.
7. Write 0x1 into register EFUSE_CMD.
8. Poll register EFUSE_CMD until it is 0x0, or wait for a read-done interrupt.
9. Set the corresponding register bit of the programmed bit to 0.

The configuration values of the EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and the EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF are based on the current APB_CLK frequency, as is shown in Table 5.3-4.

**Table 5.3-4. Timing Configuration**

| Register | Configuration Value | APB_CLK Frequency | | |
|---|---|---|---|---|
| | | 26 MHz | 40 MHz | 80 MHz |
| EFUSE_CLK | EFUSE_CLK_SEL0[7:0] | 250 | 160 | 80 |
| | EFUSE_CLK_SEL1[7:0] | 255 | 255 | 128 |
| EFUSE_DAC_CONF | EFUSE_DAC_CLK_DIV[7:0] | 52 | 80 | 100 |

The two methods to identify the generation of program/read-done interrupts are as follows:

Method One:

1. Poll bit 1/0 in register EFUSE_INT_RAW until bit 1/0 is 1, which represents the generation of an program/read-done interrupt.
2. Set the bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupts.

Method Two:

1. Set bit 1/0 in register EFUSE_INT_ENA to 1 to enable eFuse Controller to post a program/read-done interrupt.
2. Configure Interrupt Matrix to enable the CPU to respond to an EFUSE_INT interrupt.
3. A program/read-done interrupt is generated.
4. Read bit 1/0 in register EFUSE_INT_ST to identify the generation of the program/read-done interrupt.
5. Set bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupt.

The programming of different system parameters and even the programming of different bits of the same system parameter can be completed separately in multiple programmings. It is, however, recommended that users minimize programming cycles, and program all the bits that need to be programmed in a system parameter in one programming action. In addition, after all system parameters controlled by a certain bit of efuse_wr_disable are programmed, that bit should be immediately programmed. The programming of system parameters controlled by a certain bit of efuse_wr_disable, and the programming of that bit can even be completed at the same time. **Repeated programming of programmed bits is strictly forbidden.**

### 5.3.3 Software Reading of System Parameters

Each bit of the 30 fixed-length system parameters and the three variable-length system parameters corresponds to a software-read register bit, as shown in Table 5.3-5. Software can use the value of each system parameter by reading the value in the corresponding register.

The bit width of system parameters BLOCK1, BLOCK2, and BLOCK3 is variable. Although 256 register bits have been assigned to each of the three parameters, as shown in Table 5.3-5, some of the 256 register bits are useless in the 3/4 coding and the Repeat coding scheme. In the None coding scheme, the corresponding register bit of each bit of $BLOCKN[255:0]$ is used. In the 3/4 coding scheme, only the corresponding register bits of $BLOCKN[191:0]$ are useful. In Repeat coding scheme, only the corresponding bits of $BLOCKN[127:0]$ are useful. In different coding schemes, the values of useless register bits read by software are invalid. **The values of useful register bits read by software are the system parameters BLOCK1, BLOCK2, and BLOCK3 themselves instead of their values after being encoded.**

Table 5.3-5. Software Read Registers

| System parameter | | | Register | |
|---|---|---|---|---|
| Name | Bit Width | Bit | Name | Bit |
| efuse_wr_disable | 16 | [15:0] | EFUSE_BLK0_RDATA0_REG | [15:0] |
| efuse_rd_disable | 4 | [3:0] | | [19:16] |
| flash_crypt_cnt | 7 | [6:0] | | [26:20] |
| uart_download_dis | 1 | [0] | | [27] |
| WIFI_MAC_Address | 56 | [31:0] | EFUSE_BLK0_RDATA1_REG | [31:0] |
| | | [55:32] | EFUSE_BLK0_RDATA2_REG | [23:0] |
| disable_app_cpu | 1 | [0] | EFUSE_BLK0_RDATA3_REG | [0] |
| disable_bt | 1 | [0] | | [1] |
| pkg_version | 4 | [3:0] | | [2], [11:9] |
| disable_cache | 1 | [0] | | [3] |
| SPI_pad_config_hd | 5 | [4:0] | | [8:4] |
| BLK3_part_reserve | 1 | [0] | | [14] |
| CK8M Frequency | 8 | [7:0] | EFUSE_BLK0_RDATA4_REG | [7:0] |
| XPD_SDIO_REG | 1 | [0] | | [14] |
| SDIO_TIEH | 1 | [0] | | [15] |
| sdio_force | 1 | [0] | | [16] |
| SPI_pad_config_clk | 5 | [4:0] | EFUSE_BLK0_RDATA5_REG | [4:0] |
| SPI_pad_config_q | 5 | [4:0] | | [9:5] |
| SPI_pad_config_d | 5 | [4:0] | | [14:10] |
| SPI_pad_config_cs0 | 5 | [4:0] | | [19:15] |
| vol_level_hp_inv | 2 | [1:0] | | [23:22] |
| dig_vol_l6 | 4 | [3:0] | | [27:24] |
| flash_crypt_config | 4 | [3:0] | | [31:28] |
| coding_scheme | 2 | [1:0] | EFUSE_BLK0_RDATA6_REG | [1:0] |
| console_debug_disable | 1 | [0] | | [2] |
| abstract_done_0 | 1 | [0] | | [4] |
| abstract_done_1 | 1 | [0] | | [5] |

| System parameter | | | Register | |
|---|---|---|---|---|
| Name | Bit Width | Bit | Name | Bit |
| JTAG_disable | 1 | [0] | | [6] |
| download_dis_encrypt | 1 | [0] | | [7] |
| download_dis_decrypt | 1 | [0] | | [8] |
| download_dis_cache | 1 | [0] | | [9] |
| key_status | 1 | [0] | | [10] |

| System parameter | | | Register | |
|---|---|---|---|---|
| Name | Bit Width | Bit | Name | Bit |
| BLOCK1 | 256/192/128 | [31:0] | EFUSE_BLK1_RDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK1_RDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK1_RDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK1_RDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK1_RDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK1_RDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK1_RDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK1_RDATA7_REG | [31:0] |
| BLOCK2 | 256/192/128 | [31:0] | EFUSE_BLK2_RDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK2_RDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK2_RDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK2_RDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK2_RDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK2_RDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK2_RDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK2_RDATA7_REG | [31:0] |
| BLOCK3 | 256/192/128 | [31:0] | EFUSE_BLK3_RDATA0_REG | [31:0] |
| | | [63:32] | EFUSE_BLK3_RDATA1_REG | [31:0] |
| | | [95:64] | EFUSE_BLK3_RDATA2_REG | [31:0] |
| | | [127:96] | EFUSE_BLK3_RDATA3_REG | [31:0] |
| | | [159:128] | EFUSE_BLK3_RDATA4_REG | [31:0] |
| | | [191:160] | EFUSE_BLK3_RDATA5_REG | [31:0] |
| | | [223:192] | EFUSE_BLK3_RDATA6_REG | [31:0] |
| | | [255:224] | EFUSE_BLK3_RDATA7_REG | [31:0] |

### 5.3.4 The Use of System Parameters by Hardware Modules

Hardware modules are directly hardwired to the ESP32 in order to use the system parameters. Software cannot change this behaviour. **Hardware modules use the decoded values of system parameters BLOCK1, BLOCK2, and BLOCK3, not their encoded values.**

### 5.3.5 Interrupts

- EFUSE_PGM_DONE_INT: Triggered when eFuse programming has finished.
- EFUSE_READ_DONE_INT: Triggered when eFuse reading has finished.

## 5.4 Register Summary

The addresses in this section are relative to the eFuse Controller base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*.

| Name | Description | Address | Access |
|---|---|---|---|
| **eFuse data read registers** | | | |
| EFUSE_BLK0_RDATA0_REG | Returns data word 0 in eFuse BLOCK 0 | 0x3FF5A000 | RO |
| EFUSE_BLK0_RDATA1_REG | Returns data word 1 in eFuse BLOCK 0 | 0x3FF5A004 | RO |
| EFUSE_BLK0_RDATA2_REG | Returns data word 2 in eFuse BLOCK 0 | 0x3FF5A008 | RO |
| EFUSE_BLK0_RDATA3_REG | Returns data word 3 in eFuse BLOCK 0 | 0x3FF5A00C | RO |
| EFUSE_BLK0_RDATA4_REG | Returns data word 4 in eFuse BLOCK 0 | 0x3FF5A010 | RO |
| EFUSE_BLK0_RDATA5_REG | Returns data word 5 in eFuse BLOCK 0 | 0x3FF5A014 | RO |
| EFUSE_BLK0_RDATA6_REG | Returns data word 6 in eFuse BLOCK 0 | 0x3FF5A018 | RO |
| EFUSE_BLK1_RDATA0_REG | Returns data word 0 in eFuse BLOCK 1 | 0x3FF5A038 | RO |
| EFUSE_BLK1_RDATA1_REG | Returns data word 1 in eFuse BLOCK 1 | 0x3FF5A03C | RO |
| EFUSE_BLK1_RDATA2_REG | Returns data word 2 in eFuse BLOCK 1 | 0x3FF5A040 | RO |
| EFUSE_BLK1_RDATA3_REG | Returns data word 3 in eFuse BLOCK 1 | 0x3FF5A044 | RO |
| EFUSE_BLK1_RDATA4_REG | Returns data word 4 in eFuse BLOCK 1 | 0x3FF5A048 | RO |
| EFUSE_BLK1_RDATA5_REG | Returns data word 5 in eFuse BLOCK 1 | 0x3FF5A04C | RO |
| EFUSE_BLK1_RDATA6_REG | Returns data word 6 in eFuse BLOCK 1 | 0x3FF5A050 | RO |
| EFUSE_BLK1_RDATA7_REG | Returns data word 7 in eFuse BLOCK 1 | 0x3FF5A054 | RO |
| EFUSE_BLK2_RDATA0_REG | Returns data word 0 in eFuse BLOCK 2 | 0x3FF5A058 | RO |
| EFUSE_BLK2_RDATA1_REG | Returns data word 1 in eFuse BLOCK 2 | 0x3FF5A05C | RO |
| EFUSE_BLK2_RDATA2_REG | Returns data word 2 in eFuse BLOCK 2 | 0x3FF5A060 | RO |
| EFUSE_BLK2_RDATA3_REG | Returns data word 3 in eFuse BLOCK 2 | 0x3FF5A064 | RO |
| EFUSE_BLK2_RDATA4_REG | Returns data word 4 in eFuse BLOCK 2 | 0x3FF5A068 | RO |
| EFUSE_BLK2_RDATA5_REG | Returns data word 5 in eFuse BLOCK 2 | 0x3FF5A06C | RO |
| EFUSE_BLK2_RDATA6_REG | Returns data word 6 in eFuse BLOCK 2 | 0x3FF5A070 | RO |
| EFUSE_BLK2_RDATA7_REG | Returns data word 7 in eFuse BLOCK 2 | 0x3FF5A074 | RO |
| EFUSE_BLK3_RDATA0_REG | Returns data word 0 in eFuse BLOCK 3 | 0x3FF5A078 | RO |
| EFUSE_BLK3_RDATA1_REG | Returns data word 1 in eFuse BLOCK 3 | 0x3FF5A07C | RO |
| EFUSE_BLK3_RDATA2_REG | Returns data word 2 in eFuse BLOCK 3 | 0x3FF5A080 | RO |
| EFUSE_BLK3_RDATA3_REG | Returns data word 3 in eFuse BLOCK 3 | 0x3FF5A084 | RO |
| EFUSE_BLK3_RDATA4_REG | Returns data word 4 in eFuse BLOCK 3 | 0x3FF5A088 | RO |
| EFUSE_BLK3_RDATA5_REG | Returns data word 5 in eFuse BLOCK 3 | 0x3FF5A08C | RO |
| EFUSE_BLK3_RDATA6_REG | Returns data word 6 in eFuse BLOCK 3 | 0x3FF5A090 | RO |
| EFUSE_BLK3_RDATA7_REG | Returns data word 7 in eFuse BLOCK 3 | 0x3FF5A094 | RO |
| **eFuse data write registers** | | | |
| EFUSE_BLK0_WDATA0_REG | Writes data to word 0 in eFuse BLOCK 0 | 0x3FF5A01c | R/W |
| EFUSE_BLK0_WDATA1_REG | Writes data to word 1 in eFuse BLOCK 0 | 0x3FF5A020 | R/W |
| EFUSE_BLK0_WDATA2_REG | Writes data to word 2 in eFuse BLOCK 0 | 0x3FF5A024 | R/W |
| EFUSE_BLK0_WDATA3_REG | Writes data to word 3 in eFuse BLOCK 0 | 0x3FF5A028 | R/W |
| EFUSE_BLK0_WDATA4_REG | Writes data to word 4 in eFuse BLOCK 0 | 0x3FF5A02c | R/W |
| EFUSE_BLK0_WDATA5_REG | Writes data to word 5 in eFuse BLOCK 0 | 0x3FF5A030 | R/W |
| EFUSE_BLK0_WDATA6_REG | Writes data to word 6 in eFuse BLOCK 0 | 0x3FF5A034 | R/W |
| EFUSE_BLK1_WDATA0_REG | Writes data to word 0 in eFuse BLOCK 1 | 0x3FF5A098 | R/W |
| EFUSE_BLK1_WDATA1_REG | Writes data to word 1 in eFuse BLOCK 1 | 0x3FF5A09c | R/W |
| EFUSE_BLK1_WDATA2_REG | Writes data to word 2 in eFuse BLOCK 1 | 0x3FF5A0a0 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| EFUSE_BLK1_WDATA3_REG | Writes data to word 3 in eFuse BLOCK 1 | 0x3FF5A0a4 | R/W |
| EFUSE_BLK1_WDATA4_REG | Writes data to word 4 in eFuse BLOCK 1 | 0x3FF5A0a8 | R/W |
| EFUSE_BLK1_WDATA5_REG | Writes data to word 5 in eFuse BLOCK 1 | 0x3FF5A0ac | R/W |
| EFUSE_BLK1_WDATA6_REG | Writes data to word 6 in eFuse BLOCK 1 | 0x3FF5A0b0 | R/W |
| EFUSE_BLK1_WDATA7_REG | Writes data to word 7 in eFuse BLOCK 1 | 0x3FF5A0b4 | R/W |
| EFUSE_BLK2_WDATA0_REG | Writes data to word 0 in eFuse BLOCK 2 | 0x3FF5A0b8 | R/W |
| EFUSE_BLK2_WDATA1_REG | Writes data to word 1 in eFuse BLOCK 2 | 0x3FF5A0bc | R/W |
| EFUSE_BLK2_WDATA2_REG | Writes data to word 2 in eFuse BLOCK 2 | 0x3FF5A0c0 | R/W |
| EFUSE_BLK2_WDATA3_REG | Writes data to word 3 in eFuse BLOCK 2 | 0x3FF5A0c4 | R/W |
| EFUSE_BLK2_WDATA4_REG | Writes data to word 4 in eFuse BLOCK 2 | 0x3FF5A0c8 | R/W |
| EFUSE_BLK2_WDATA5_REG | Writes data to word 5 in eFuse BLOCK 2 | 0x3FF5A0cc | R/W |
| EFUSE_BLK2_WDATA6_REG | Writes data to word 6 in eFuse BLOCK 2 | 0x3FF5A0d0 | R/W |
| EFUSE_BLK2_WDATA7_REG | Writes data to word 7 in eFuse BLOCK 2 | 0x3FF5A0d4 | R/W |
| EFUSE_BLK3_WDATA0_REG | Writes data to word 0 in eFuse BLOCK 3 | 0x3FF5A0d8 | R/W |
| EFUSE_BLK3_WDATA1_REG | Writes data to word 1 in eFuse BLOCK 3 | 0x3FF5A0dc | R/W |
| EFUSE_BLK3_WDATA2_REG | Writes data to word 2 in eFuse BLOCK 3 | 0x3FF5A0e0 | R/W |
| EFUSE_BLK3_WDATA3_REG | Writes data to word 3 in eFuse BLOCK 3 | 0x3FF5A0e4 | R/W |
| EFUSE_BLK3_WDATA4_REG | Writes data to word 4 in eFuse BLOCK 3 | 0x3FF5A0e8 | R/W |
| EFUSE_BLK3_WDATA5_REG | Writes data to word 5 in eFuse BLOCK 3 | 0x3FF5A0ec | R/W |
| EFUSE_BLK3_WDATA6_REG | Writes data to word 6 in eFuse BLOCK 3 | 0x3FF5A0f0 | R/W |
| EFUSE_BLK3_WDATA7_REG | Writes data to word 7 in eFuse BLOCK 3 | 0x3FF5A0f4 | R/W |
| **Control registers** | | | |
| EFUSE_CLK_REG | Timing configuration register | 0x3FF5A0F8 | R/W |
| EFUSE_CONF_REG | Opcode register | 0x3FF5A0FC | R/W |
| EFUSE_CMD_REG | Read/write command register | 0x3FF5A104 | R/W |
| **Interrupt registers** | | | |
| EFUSE_INT_RAW_REG | Raw interrupt status | 0x3FF5A108 | RO |
| EFUSE_INT_ST_REG | Masked interrupt status | 0x3FF5A10C | RO |
| EFUSE_INT_ENA_REG | Interrupt enable bits | 0x3FF5A110 | R/W |
| EFUSE_INT_CLR_REG | Interrupt clear bits | 0x3FF5A114 | WO |
| **Misc registers** | | | |
| EFUSE_DAC_CONF_REG | Efuse timing configuration | 0x3FF5A118 | R/W |
| EFUSE_DEC_STATUS_REG | Status of 3/4 coding scheme | 0x3FF5A11C | RO |

## 5.5 Registers

The addresses in this section are relative to the eFuse Controller base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*.

**Register 5.1. EFUSE_BLK0_RDATA0_REG (0x000)**

| (reserved) | EFUSE_RD_UART_DOWNLOAD_DIS | EFUSE_RD_FLASH_CRYPT_CNT | EFUSE_RD_EFUSE_RD_DIS | EFUSE_RD_EFUSE_WR_DIS | |
|---|---|---|---|---|---|
| 31 28 | 27 | 26 20 | 19 16 | 15 0 | |
| 0 0 0 0 | 0 | 0 0 0 0 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 |

**EFUSE_RD_UART_DOWNLOAD_DIS** This bit returns the value of uart_download_dis. Valid only for ESP32 . (RO)

**EFUSE_RD_FLASH_CRYPT_CNT** This field returns the value of flash_crypt_cnt. (RO)

**EFUSE_RD_EFUSE_RD_DIS** This field returns the value of efuse_rd_disable. (RO)

**EFUSE_RD_EFUSE_WR_DIS** This field returns the value of efuse_wr_disable. (RO)

**Register 5.2. EFUSE_BLK0_RDATA1_REG (0x004)**

| 31 0 | |
|---|---|
| 0 0 0 0 0 0 0 0 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 |

**EFUSE_BLK0_RDATA1_REG** This field returns the value of the lower 32 bits of WIFI_MAC_Address. (RO)

**Register 5.3. EFUSE_BLK0_RDATA2_REG (0x008)**

| (reserved) | EFUSE_RD_WIFI_MAC_CRC_HIGH | |
|---|---|---|
| 31 24 | 23 0 | |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 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 |

**EFUSE_RD_WIFI_MAC_CRC_HIGH** This field returns the value of the higher 24 bits of WIFI_MAC_Address. (RO)

**Register 5.4. EFUSE_BLK0_RDATA3_REG (0x00c)**

| (reserved) | EFUSE_RD_CHIP_VER_PKG | EFUSE_RD_SPI_PAD_CONFIG_HD | EFUSE_RD_CHIP_VER_DIS_CACHE | EFUSE_RD_CHIP_VER_PKG | EFUSE_RD_CHIP_VER_DIS_BT | EFUSE_RD_CHIP_VER_DIS_APP_CPU | |
|---|---|---|---|---|---|---|---|
| 31 – 12 | 11 – 9 | 8 – 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 | 0 | 0 | Reset |

**EFUSE_RD_CHIP_VER_PKG** These are the first three identification bits of chip packaging version among the four identification bits. (RO)

**EFUSE_RD_SPI_PAD_CONFIG_HD** This field returns the value of SPI_pad_config_hd. (RO)

**EFUSE_RD_CHIP_VER_DIS_CACHE** Disables cache. (RO)

**EFUSE_RD_CHIP_VER_PKG** This is the fourth identification bit of chip packaging version among the four identification bits. (RO)

**EFUSE_RD_CHIP_VER_DIS_BT** Disables Bluetooth. (RO)

**EFUSE_RD_CHIP_VER_DIS_APP_CPU** Disables APP CPU. (RO)

**Register 5.5. EFUSE_BLK0_RDATA4_REG (0x010)**

| (reserved) | EFUSE_RD_SDIO_FORCE | EFUSE_RD_SDIO_TIEH | EFUSE_RD_XPD_SDIO | (reserved) | ESFUSE_RD_CK8M_FREQ | |
|---|---|---|---|---|---|---|
| 31 – 17 | 16 | 15 | 14 | 13 – 8 | 7 – 0 | |
| 0 0 0 0 0 0 0 0 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 |

**EFUSE_RD_SDIO_FORCE** This field returns the value of sdio_force. (RO)

**EFUSE_RD_SDIO_TIEH** This field returns the value of SDIO_TIEH. (RO)

**EFUSE_RD_XPD_SDIO** This field returns the value of XPD_SDIO_REG. (RO)

**ESFUSE_RD_CK8M_FREQ** RC_FAST_CLK frequency. (RO)

**Register 5.6. EFUSE_BLK0_RDATA5_REG (0x014)**

| EFUSE_RD_FLASH_CRYPT_CONFIG | EFUSE_RD_DIG_VOL_L6 | EFUSE_RD_VOL_LEVEL_HP_INV | (reserved) | EFUSE_RD_SPI_PAD_CONFIG_CS0 | EFUSE_RD_SPI_PAD_CONFIG_D | EFUSE_RD_SPI_PAD_CONFIG_Q | EFUSE_RD_SPI_PAD_CONFIG_CLK | |
|---|---|---|---|---|---|---|---|---|
| 31 28 | 27 24 | 23 22 | 21 20 | 19 15 | 14 10 | 9 5 | 4 0 | |
| 0 0 0 0 | 0 0 0 0 | 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 |

**EFUSE_RD_FLASH_CRYPT_CONFIG** This field returns the value of flash_crypt_config. (RO)

**EFUSE_RD_DIG_VOL_L6** This field stores the difference between the digital regulator voltage at level 6 and 1.2 V. (RO)

**EFUSE_RD_VOL_LEVEL_HP_INV** This field stores the voltage level for CPU to run at 240 MHz, or for flash/PSRAM to run at 80 MHz. 0x0: level 7; 0x1: level 6; 0x2: level 5; 0x3: level 4. (RO)

**EFUSE_RD_SPI_PAD_CONFIG_CS0** This field returns the value of SPI_pad_config_cs0. (RO)

**EFUSE_RD_SPI_PAD_CONFIG_D** This field returns the value of SPI_pad_config_d. (RO)

**EFUSE_RD_SPI_PAD_CONFIG_Q** This field returns the value of SPI_pad_config_q. (RO)

**EFUSE_RD_SPI_PAD_CONFIG_CLK** This field returns the value of SPI_pad_config_clk. (RO)

### Register 5.7. EFUSE_BLK0_RDATA6_REG (0x018)

| (reserved) | EFUSE_RD_KEY_STATUS | EFUSE_RD_DISABLE_DL_CACHE | EFUSE_RD_DISABLE_DL_DECRYPT | EFUSE_RD_DISABLE_DL_ENCRYPT | EFUSE_RD_DISABLE_JTAG | EFUSE_RD_ABS_DONE_1 | EFUSE_RD_ABS_DONE_0 | (reserved) | EFUSE_RD_CONSOLE_DEBUG_DISABLE | EFUSE_RD_CODING_SCHEME | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 ... 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 |

**EFUSE_RD_KEY_STATUS** This field returns the value of key_status. (RO)

**EFUSE_RD_DISABLE_DL_CACHE** This field returns the value of download_dis_cache. (RO)

**EFUSE_RD_DISABLE_DL_DECRYPT** This field returns the value of download_dis_decrypt. (RO)

**EFUSE_RD_DISABLE_DL_ENCRYPT** This field returns the value of download_dis_encrypt. (RO)

**EFUSE_RD_DISABLE_JTAG** This field returns the value of JTAG_disable. (RO)

**EFUSE_RD_ABS_DONE_1** This field returns the value of abstract_done_1. (RO)

**EFUSE_RD_ABS_DONE_0** This field returns the value of abstract_done_0. (RO)

**EFUSE_RD_CONSOLE_DEBUG_DISABLE** This field returns the value of console_debug_disable. (RO)

**EFUSE_RD_CODING_SCHEME** This field returns the value of coding_scheme. (RO)

### Register 5.8. EFUSE_BLK0_WDATA0_REG (0x01c)

| (reserved) | EFUSE_UART_DOWNLOAD_DIS | EFUSE_FLASH_CRYPT_CNT | EFUSE_RD_DIS | EFUSE_WR_DIS | |
|---|---|---|---|---|---|
| 31 ... 28 | 27 | 26 ... 20 | 19 ... 16 | 15 ... 0 | |
| 0 0 0 0 | 0 | 0 0 0 0 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 |

**EFUSE_UART_DOWNLOAD_DIS** This bit programs the value of uart_download_dis. Valid only for ESP32 ECO V3. (R/W)

**EFUSE_FLASH_CRYPT_CNT** This field programs the value of flash_crypt_cnt. (R/W)

**EFUSE_RD_DIS** This field programs the value of efuse_rd_disable. (R/W)

**EFUSE_WR_DIS** This field programs the value of efuse_wr_disable. (R/W)

### Register 5.9. EFUSE_BLK0_WDATA1_REG (0x020)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**EFUSE_BLK0_WDATA1_REG** This field programs the value of lower 32 bits of WIFI_MAC_Address. (R/W)

### Register 5.10. EFUSE_BLK0_WDATA2_REG (0x024)

| (reserved) | EFUSE_WIFI_MAC_CRC_HIGH | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 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 |

**EFUSE_WIFI_MAC_CRC_HIGH** This field programs the value of higher 24 bits of WIFI_MAC_Address. (R/W)

### Register 5.11. EFUSE_BLK0_WDATA3_REG (0x028)

| (reserved) | EFUSE_CHIP_VER_PKG | EFUSE_SPI_PAD_CONFIG_HD | EFUSE_CHIP_VER_DIS_CACHE | EFUSE_CHIP_VER_PKG | EFUSE_CHIP_VER_DIS_BT | EFUSE_CHIP_VER_DIS_APP_CPU | |
|---|---|---|---|---|---|---|---|
| 31 … 12 | 11 … 9 | 8 … 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 | 0 | 0 | Reset |

**EFUSE_CHIP_VER_PKG** These are the first three bits among the four bits to program chip packaging version. (R/W)

**EFUSE_SPI_PAD_CONFIG_HD** This field programs the value of SPI_pad_config_hd. (R/W)

**EFUSE_CHIP_VER_DIS_CACHE** This field is programmed to disable cache. (R/W)

**EFUSE_CHIP_VER_PKG** This is the fourth bit among the four bits to program chip packaging version. (R/W)

**EFUSE_CHIP_VER_DIS_BT** This field is programmed to disable Bluetooth. (R/W)

**EFUSE_CHIP_VER_DIS_APP_CPU** This field is programmed to disable APP CPU. (R/W)

**Register 5.12. EFUSE_BLK0_WDATA4_REG (0x02c)**

| (reserved) | EFUSE_SDIO_FORCE | EFUSE_SDIO_TIEH | EFUSE_XPD_SDIO | (reserved) | ESFUSE_CK8M_FREQ | |
|---|---|---|---|---|---|---|
| 31 … 17 | 16 | 15 | 14 | 13 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**EFUSE_SDIO_FORCE** This field programs the value of SDIO_TIEH. (R/W)

**EFUSE_SDIO_TIEH** This field programs the value of SDIO_TIEH. (R/W)

**EFUSE_XPD_SDIO** This field programs the value of XPD_SDIO_REG. (R/W)

**ESFUSE_CK8M_FREQ** This field programs the frequency of RC_FAST_CLK. (R/W)

**Register 5.13. EFUSE_BLK0_WDATA5_REG (0x030)**

| EFUSE_FLASH_CRYPT_CONFIG | EFUSE_DIG_VOL_L6 | EFUSE_VOL_LEVEL_HP_INV | (reserved) | EFUSE_SPI_PAD_CONFIG_CS0 | EFUSE_SPI_PAD_CONFIG_D | EFUSE_SPI_PAD_CONFIG_Q | EFUSE_SPI_PAD_CONFIG_CLK | |
|---|---|---|---|---|---|---|---|---|
| 31 … 28 | 27 … 24 | 23 22 | 21 20 | 19 … 15 | 14 … 10 | 9 … 5 | 4 … 0 | |
| 0 0 0 0 | 0 0 0 0 | 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 |

**EFUSE_FLASH_CRYPT_CONFIG** This field programs the value of flash_crypt_config. (R/W)

**EFUSE_DIG_VOL_L6** This field stores the difference between the digital regulator voltage at level 6 and 1.2 V. (R/W)

**EFUSE_VOL_LEVEL_HP_INV** These bits store the voltage level for CPU to run at 240 MHz, or for flash/PSRAM to run at 80 MHz. 0x0: level 7; 0x1: level 6; 0x2: level 5; 0x3: level 4. (R/W)

**EFUSE_SPI_PAD_CONFIG_CS0** This field programs the value of SPI_pad_config_cs0. (R/W)

**EFUSE_SPI_PAD_CONFIG_D** This field programs the value of SPI_pad_config_d. (R/W)

**EFUSE_SPI_PAD_CONFIG_Q** This field programs the value of SPI_pad_config_q. (R/W)

**EFUSE_SPI_PAD_CONFIG_CLK** This field programs the value of SPI_pad_config_clk. (R/W)

**Register 5.14. EFUSE_BLK0_WDATA6_REG (0x034)**

| 31 | | | | | | | | | | | | | | | | | | | | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | EFUSE_KEY_STATUS | EFUSE_DISABLE_DL_CACHE | EFUSE_DISABLE_DL_DECRYPT | EFUSE_DISABLE_DL_ENCRYPT | EFUSE_DISABLE_JTAG | EFUSE_ABS_DONE_1 | EFUSE_ABS_DONE_0 | (reserved) | EFUSE_CONSOLE_DEBUG_DISABLE | EFUSE_CODING_SCHEME | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**EFUSE_KEY_STATUS** This field programs the value of key_status. (R/W)

**EFUSE_DISABLE_DL_CACHE** This field programs the value of download_dis_cache. (R/W)

**EFUSE_DISABLE_DL_DECRYPT** This field programs the value of download_dis_decrypt. (R/W)

**EFUSE_DISABLE_DL_ENCRYPT** This field programs the value of download_dis_encrypt. (R/W)

**EFUSE_DISABLE_JTAG** This field programs the value of JTAG_disable. (R/W)

**EFUSE_ABS_DONE_1** This field programs the value of abstract_done_1. (R/W)

**EFUSE_ABS_DONE_0** This field programs the value of abstract_done_0. (R/W)

**EFUSE_CONSOLE_DEBUG_DISABLE** This field programs the value of console_debug_disable. (R/W)

**EFUSE_CODING_SCHEME** This field programs the value of coding_scheme. (R/W)

**Register 5.15. EFUSE_BLK1_RDATA*n*_REG (*n*: 0-7) (0x38+4**n*)**

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**EFUSE_BLK1_RDATA*n*_REG** This field returns the value of word *n* in BLOCK1. (RO)

**Register 5.16. EFUSE_BLK2_RDATA*n*_REG (*n*: 0-7) (0x58+4**n*)**

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**EFUSE_BLK2_RDATA*n*_REG** This field returns the value of word *n* in BLOCK2. (RO)

### Register 5.17. EFUSE_BLK3_RDATA*n*_REG (*n*: 0-7) (0x78+4***n*)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**EFUSE_BLK3_RDATA*n*_REG** This field returns the value of word *n* in BLOCK3. (RO)

### Register 5.18. EFUSE_BLK1_WDATA*n*_REG (*n*: 0-7) (0x98+4***n*)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**EFUSE_BLK1_WDATA*n*_REG** This field programs the value of word *n* in of BLOCK1. (R/W)

### Register 5.19. EFUSE_BLK2_WDATA*n*_REG (*n*: 0-7) (0xB8+4***n*)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**EFUSE_BLK2_WDATA*n*_REG** This field programs the value of word *n* in of BLOCK2. (R/W)

### Register 5.20. EFUSE_BLK3_WDATA*n*_REG (*n*: 0-7) (0xD8+4***n*)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**EFUSE_BLK3_WDATA*n*_REG** This field programs the value of word *n* in of BLOCK3. (R/W)

### Register 5.21. EFUSE_CLK_REG (0x0f8)

| (reserved) | EFUSE_CLK_SEL1 | EFUSE_CLK_SEL0 |
|---|---|---|
| 31 … 16 | 15 … 8 | 7 … 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x040 | 0x052 |

Reset

**EFUSE_CLK_SEL1** eFuse clock configuration field. (R/W)

**EFUSE_CLK_SEL0** eFuse clock configuration field. (R/W)

### Register 5.22. EFUSE_CONF_REG (0x0fc)

| (reserved) | EFUSE_OP_CODE | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x00000 | Reset |

**EFUSE_OP_CODE** eFuse operation code register. (R/W)

### Register 5.23. EFUSE_CMD_REG (0x104)

| (reserved) | EFUSE_PGM_CMD | EFUSE_READ_CMD | |
|---|---|---|---|
| 31 … 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 |

**EFUSE_PGM_CMD** Set this to 1 to start a program operation. Reverts to 0 when the program operation is done. (R/W)

**EFUSE_READ_CMD** Set this to 1 to start a read operation. Reverts to 0 when the read operation is done. (R/W)

### Register 5.24. EFUSE_INT_RAW_REG (0x108)

| (reserved) | EFUSE_PGM_DONE_INT_RAW | EFUSE_READ_DONE_INT_RAW | |
|---|---|---|---|
| 31 … 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 |

**EFUSE_PGM_DONE_INT_RAW** The raw interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

**EFUSE_READ_DONE_INT_RAW** The raw interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

Register 5.25. EFUSE_INT_ST_REG (0x10c)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EFUSE_PGM_DONE_INT_ST | EFUSE_READ_DONE_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**EFUSE_PGM_DONE_INT_ST** The masked interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

**EFUSE_READ_DONE_INT_ST** The masked interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

Register 5.26. EFUSE_INT_ENA_REG (0x110)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EFUSE_PGM_DONE_INT_ENA | EFUSE_READ_DONE_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**EFUSE_PGM_DONE_INT_ENA** The interrupt enable bit for the EFUSE_PGM_DONE_INT interrupt. (R/W)

**EFUSE_READ_DONE_INT_ENA** The interrupt enable bit for the EFUSE_READ_DONE_INT interrupt. (R/W)

Register 5.27. EFUSE_INT_CLR_REG (0x114)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EFUSE_PGM_DONE_INT_CLR | EFUSE_READ_DONE_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**EFUSE_PGM_DONE_INT_CLR** Set this bit to clear the EFUSE_PGM_DONE_INT interrupt. (WO)

**EFUSE_READ_DONE_INT_CLR** Set this bit to clear the EFUSE_READ_DONE_INT interrupt. (WO)

**Register 5.28. EFUSE_DAC_CONF_REG (0x118)**

| (reserved) | EFUSE_DAC_CLK_DIV | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 40 | Reset |

**EFUSE_DAC_CLK_DIV** eFuse timing configuration register. (R/W)

**Register 5.29. EFUSE_DEC_STATUS_REG (0x11c)**

| (reserved) | EFUSE_DEC_WARNINGS | |
|---|---|---|
| 31 … 12 | 11 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**EFUSE_DEC_WARNINGS** If a bit is set in this register, it means some errors were corrected while decoding the 3/4 encoding scheme. (RO)

# Part III

# System Component

Encompassing a range of system-level functionalities, this part describes components related to clocks, GPIO, timers, watchdogs, interrupt handling, low-power management, and various system registers.

# Chapter 6

# IO MUX and GPIO Matrix (GPIO, IO MUX)

## 6.1 Overview

The ESP32 chip features 34 physical GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. The IO MUX [1], RTC IO MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pins. Together these systems provide highly configurable I/O.

**Note that the I/O GPIO pins are 0-19, 21-23, 25-27, 32-39, while the output GPIOs are 0-19, 21-23, 25-27, 32-33. GPIO pins 34-39 are input-only.**

**GPIO20 serves as a valid input and output only on ESP32-PICO-V3 and ESP32-PICO-V3-02. Please refer to ESP32-PICO Series Datasheet for more information.**

This chapter describes the signal selection and connection between the digital pins (FUN_SEL, IE, OE, WPU, WDU, etc.), 162 peripheral input and 176 output signals (control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.), fast peripheral input/output signals (control signals: IE, OE, etc.), and RTC IO MUX.

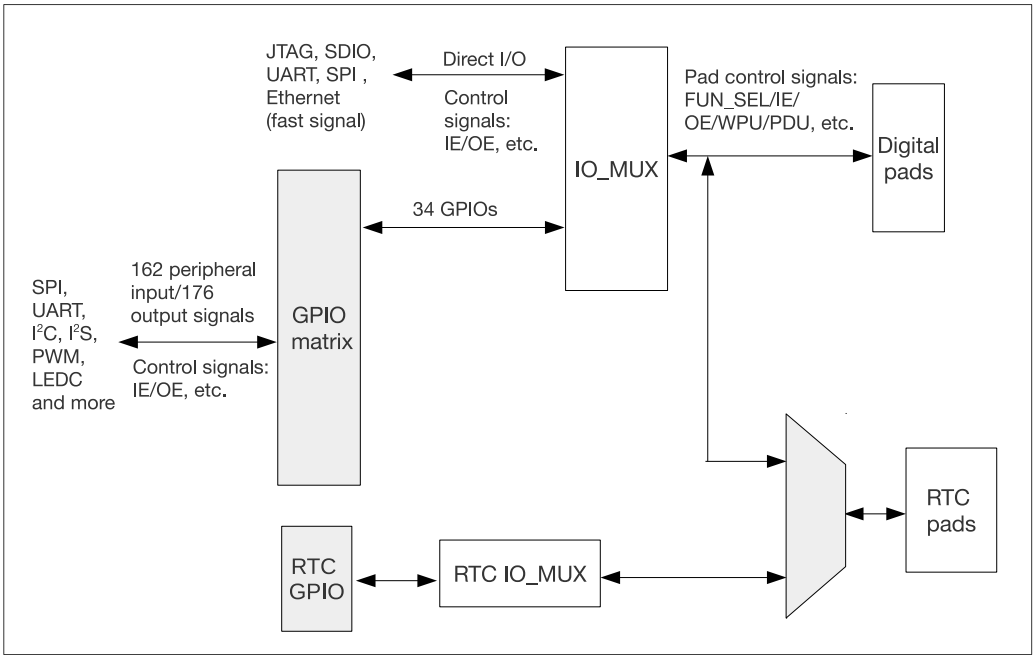



Figure 6.1-1. IO MUX, RTC IO MUX and GPIO Matrix Overview

1. The IO MUX contains one register per GPIO pin. Each pin can be configured to perform a "GPIO" function

[1] MUX: Multiplexer. The ESP32 chip integrates multiple peripherals that require communication with the outside world. To keep the chip package size reasonably small, the number of available pins has to be limited. So the only way to route all the incoming and outgoing signals is through pin multiplexing. Pin muxing is controlled via software programmable registers such as IO_MUX_x_REG.

(when connected to the GPIO Matrix) or a direct function (bypassing the GPIO Matrix). Some high-speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO MUX is used to connect these pins directly to the peripheral.)

See Section 6.10 for a list of IO MUX functions for each I/O pin.

2. The GPIO Matrix is a full-switching matrix between the peripheral input/output signals and the pins.
   - For input to the chip: Each of the 162 internal peripheral inputs can select any GPIO pin as the input source.
   - For output from the chip: The output signal of each of the 34 GPIO pins can be from one of the 176 peripheral output signals.

   See Section 6.9 for a list of GPIO Matrix peripheral signals.

3. RTC IO MUX is used to connect GPIO pins to their low-power and analog functions. Only a subset of GPIO pins have these optional "RTC" functions.

   See Section 6.11 for a list of RTC IO MUX functions.

Figure 6.1-2 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 31 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals.

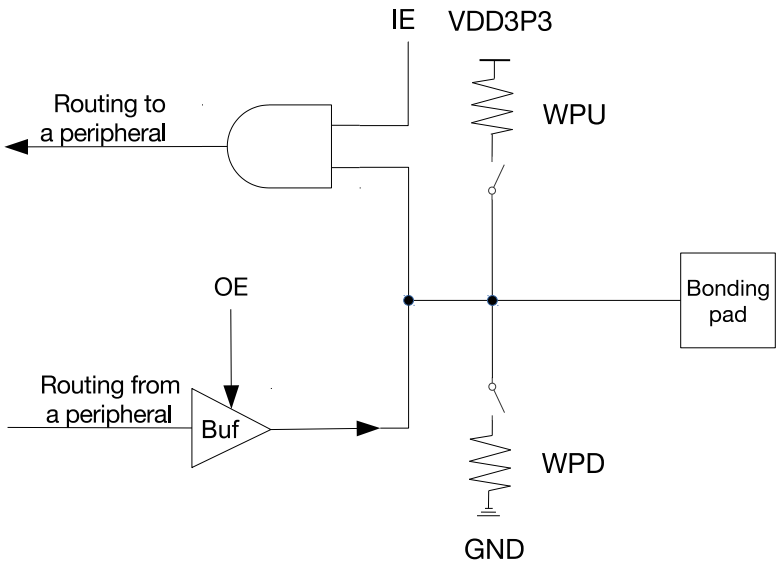



**Figure 6.1-2. Internal Structure of a Pad**

- IE: input enable
- OE: output enable
- WPU: internal weak pull-up resistor
- WPD: internal weak pull-down resistor
- Bonding pad: a terminal point of the chip logic used to make a physical connection from the chip die to GPIO pin in the chip package

## 6.2 Peripheral Input via GPIO Matrix

### 6.2.1 Summary

To receive a peripheral input signal via the GPIO Matrix, the GPIO Matrix is configured to source the peripheral signal's input index (0-18, 23-36, 39-58, 61-90, 95-124, 140-155, 164-181, 190-195, 198-206) from one of the 34 GPIOs (0-19, 21-23, 25-27, 32-39).

The input signal is read from the GPIO pin through the IO MUX. The IO MUX must be configured to set the chosen pin to "GPIO" function. This causes the GPIO pin input signal to be routed into the GPIO Matrix, which in turn routes it to the selected peripheral input.

### 6.2.2 Functional Description

Figure 6.2-1 shows the logic for input selection via GPIO Matrix.

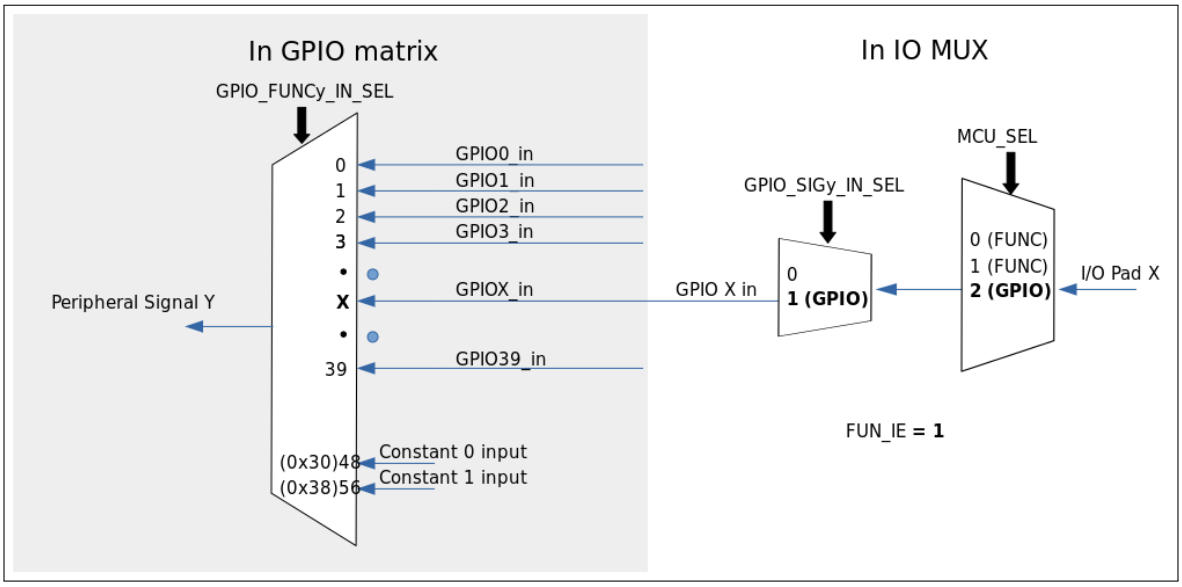



Figure 6.2-1. Peripheral Input via IO MUX, GPIO Matrix

To read GPIO pin *X* into peripheral signal *Y*, follow the steps below:

1. Configure the GPIO_FUNC*y*_IN_SEL_CFG register corresponding to peripheral signal *Y* in the GPIO Matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via GPIO matrix.
   - Set the GPIO_FUNC*y*_IN_SEL field in this register, corresponding to the GPIO pin *X* to read from.
2. Configure the GPIO_FUNC*x*_OUT_SEL_CFG register and clear the GPIO_ENABLE_DATA[*x*] field corresponding to GPIO pin *X* in the GPIO Matrix:
   - Set the GPIO_FUNC*x*_OEN_SEL bit in the GPIO_FUNC*x*_OUT_SEL_CFG register to force the pin's output state to be determined always by the GPIO_ENABLE_DATA[*x*] field.
   - The GPIO_ENABLE_DATA[*x*] field is a bit in either GPIO_ENABLE_REG (GPIOs 0-31) or GPIO_ENABLE1_REG (GPIOs 32-39). Clear this bit to disable the output driver for the GPIO pin.

3. Configure the IO MUX to select the GPIO Matrix. Set the IO_MUX_*x*_REG register corresponding to GPIO pin *X* as follows:
   - Set the function field (MCU_SEL) to the IO MUX function corresponding to GPIO *X* (this is Function 2—numeric value 2—for all pins).
   - Enable the input by setting the FUN_IE bit.
   - Set or clear the FUN_WPU and FUN_WPD bits, as desired, to enable/disable internal pull-up/pull-down resistors.

**Notes:**

- One input pin can be connected to multiple input_signals.
- The input signal can be inverted with GPIO_FUNC*y*_IN_INV_SEL.
- It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
  - When GPIO_FUNC*y*_IN_SEL is 0x30, input_signal_*x* is always 0.
  - When GPIO_FUNC*y*_IN_SEL is 0x38, input_signal_*x* is always 1.

For example, to connect RMT peripheral channel 0 input signal (RMT_SIG_IN0_IDX, signal index 83) to GPIO 15, please follow the steps below. Note that GPIO 15 is also named the MTDO pin:

1. Set the GPIO_FUNC83_IN_SEL_CFG register field GPIO_FUNC83_IN_SEL value to 15.
2. As this is an input-only signal, set GPIO_FUNC15_OEN_SEL bit in GPIO_FUNC15_OUT_SEL_CFG_REG.
3. Clear bit 15 of GPIO_ENABLE_REG (field GPIO_ENABLE_DATA[15]).
4. Set the IO_MUX_GPIO15 register MCU_SEL field to 2 (GPIO function) and also set the FUN_IE bit (input mode).

### 6.2.3 Simple GPIO Input

The GPIO_IN_REG/GPIO_IN1_REG register holds the input values of each GPIO pin.

The input value of any GPIO pin can be read at any time without configuring the GPIO Matrix for a particular peripheral signal. However, it is necessary to enable the input in the IO MUX by setting the FUN_IE bit in the IO_MUX_*x*_REG register corresponding to pin *X*, as mentioned in Section 6.2.2.

## 6.3 Peripheral Output via GPIO Matrix

### 6.3.1 Summary

To output a signal from a peripheral via the GPIO Matrix, the GPIO Matrix is configured to route the peripheral output signal (0-18, 23-37, 61-121, 140-125, 224-228) to one of the 28 GPIOs (0-19, 21-23, 25-27, 32-33).

The output signal is routed from the peripheral into the GPIO Matrix. It is then routed into the IO MUX, which is configured to set the chosen pin to "GPIO" function. This causes the output GPIO signal to be connected to the pin.

**Note:**
The peripheral output signals 224 to 228 can be configured to be routed in from one GPIO and output directly from another GPIO.

## 6.3.2 Functional Description

One of the 176 output signals can be selected to go through the GPIO matrix into the IO MUX and then to a pin. Figure 6.3-1 illustrates the configuration.

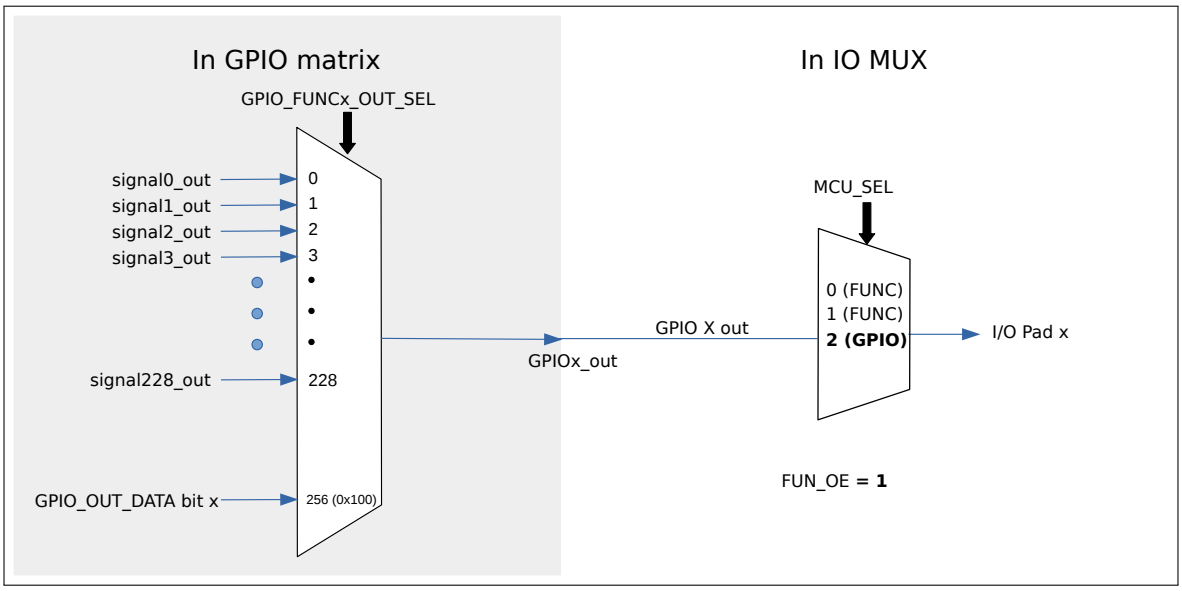



Figure 6.3-1. Output via GPIO Matrix

To output peripheral signal *Y* to particular GPIO pin *X*, follow these steps:

1. Configure the GPIO_FUNC*x*_OUT_SEL_CFG register and GPIO_ENABLE_DATA[*x*] field corresponding to GPIO *X* in the GPIO Matrix:
   - Set the GPIO_FUNC*x*_OUT_SEL field in GPIO_FUNC*x*_OUT_SEL_CFG to the numeric index (*Y*) of desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OEN_SEL bit in the GPIO_FUNC*x*_OUT_SEL_CFG register and the GPIO_ENABLE_DATA[*x*] field in the GPIO_ENABLE_REG register corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic, clear the GPIO_FUNC*x*_OEN_SEL bit instead.
   - The GPIO_ENABLE_DATA[*x*] field is a bit in either GPIO_ENABLE_REG (GPIOs 0-31) or GPIO_ENABLE1_REG (GPIOs 32-39). Clear this bit to disable the output driver for the GPIO pin.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in the GPIO_PIN*x* register corresponding to GPIO pin *X*. For push/pull mode (default), clear this bit.
3. Configure the IO MUX to select the GPIO Matrix. Set the IO_MUX_*x*_REG register corresponding to GPIO pin *X* as follows:
   - Set the function field (MCU_SEL) to the IO MUX function corresponding to GPIO *X* (this is Function 2—numeric value 2—for all pins).

- Set the 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 FUN_WPU and FUN_WPD bits to enable/disable the internal pull-up/down resistors.

**Notes:**

- The output signal from a single peripheral can be sent to multiple pins simultaneously.
- Only the 28 GPIOs can be used as outputs.
- The output signal can be inverted by setting the GPIO_FUNC*x*_OUT_INV_SEL bit.

### 6.3.3 Simple GPIO Output

The GPIO Matrix can also be used for simple GPIO output – setting a bit in the GPIO_OUT_DATA register will write to the corresponding GPIO pin.

To configure a pin as simple GPIO output, the GPIO Matrix GPIO_FUNC*x*_OUT_SEL register is configured with a special peripheral index value (0x100).

## 6.4 Direct I/O via IO MUX

### 6.4.1 Summary

Some high speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO MUX is used to connect these pins directly to the peripheral.

Selecting this option is less flexible than using the GPIO Matrix, as the IO MUX register for each GPIO pin can only select from a limited number of functions. However, better high-frequency digital performance will be maintained.

### 6.4.2 Functional Description

Two registers must be configured in order to bypass the GPIO Matrix for peripheral I/O:

1. IO MUX for the GPIO pin must be set to the required pin function. (Please refer to section 6.10 for a list of pin functions.)
2. For inputs, the SIG_IN_SEL register must be cleared to route the input directly to the peripheral.

## 6.5 RTC IO MUX for Low Power and Analog I/O

### 6.5.1 Summary

18 GPIO pins have low power capabilities (RTC domain) and analog functions which are handled by the RTC subsystem of ESP32. The IO MUX and GPIO Matrix are not used for these functions; rather, the RTC_MUX is used to redirect the I/O to the RTC subsystem.

When configured as RTC GPIOs, the output pins can still retain the output level value when the chip is in Deep-sleep mode, and the input pins can wake up the chip from Deep-sleep.

Section 6.11 has a list of RTC_MUX pins and their functions.

### 6.5.2 Analog Function Description

The RTC function and analog function of RTC_GPIOs can only be selected one at a time. For the RTC_GPIO8 to RTC_GPIO17 pins, their analog outputs can be directed to the IO MUX, controlled by the RTC_IO_TOUCH_PAD*n*/*m*_TO_GPIO bit. If the bit is set to 1, the analog output is enabled, allowing the signal to be routed to IO MUX through analog function. On the other hand, if the bit is set to 0, the input signal from the pin is output to IO MUX through digital function.

## 6.6 Light-sleep Mode Pin Functions

Pins can have different functions when the ESP32 is in Light-sleep mode. If the SLP_SEL bit in the IO MUX register for a GPIO pin is set to 1, a different set of registers is used to control the pin when the ESP32 is in Light-sleep mode:

**Table 6.6-1. IO_MUX Light-sleep Pin Function Registers**

| IO_MUX Function | Normal Execution<br>OR SLP_SEL = 0 | Light-sleep Mode<br>AND SLP_SEL = 1 |
|---|---|---|
| Output Drive Strength | FUN_DRV | MCU_DRV |
| Pull-up Resistor | FUN_WPU | MCU_WPU |
| Pull-down Resistor | FUN_WPD | MCU_WPD |
| Output Enable | (From GPIO Matrix _OEN field) | MCU_OE |

If SLP_SEL is set to 0, the pin functions remain the same in both normal execution and Light-sleep mode.

## 6.7 pin Hold Feature

Each IO pin (including the RTC pins) has an individual hold function controlled by a RTC register. When the pin is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO MUX configuration or GPIO configuration is modified. Users can use the hold function for the pins to retain the pin state through a core reset triggered by watchdog time-out or Deep-sleep events.

The Hold state of each pin is controlled by the result of OR operation of the pin's Hold enable signal and the global Hold enable signal.

- **Digital Pins** (GPIO18 ~ GPIO19, GPIO21 ~ GPIO23, GPIO25 ~ GPIO27, GPIO32 ~ GPIO39)
  - RTCIO_DIG_PAD_HOLD_REG[*n*], controls the Hold enable signal of each digital pin. See Table 6.13-1 for the bit mapping for the pins.
  - RTC_CNTL_DG_PAD_FORCE_HOLD, 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 RTCIO_DIG_PAD_HOLD_REG[*n*] (*n* = 0 ~ 31) before power off. See Table 6.13-1 for the bit mapping for the pins. To disable the Hold function of each pin after the chip is woken up, clear the bits above.
- Alternatively, set RTC_CNTL_DG_PAD_FORCE_HOLD to hold the values of all digital pins, or set RTC_CNTL_DG_PAD_FORCE_UNHOLD to disable the hold function of all digital pins.

- **RTC Pins** (GPIO0 ~ GPIO17)
  - RTC_CNTL_HOLD_FORCE_REG[*n*](*n* = 0 ~ 17), controls the Hold enable signal of each RTC pins (GPIO0 ~ GPIO17).
  - RTC_CNTL_DG_PAD_FORCE_HOLD, controls the global Hold signal of all RTC pins.

To use this feature, follow the steps below:

- To maintain the pin's input/output status in Deep-sleep, set RTC_CNTL_HOLD_FORCE_REG[*n*]. *n* ranges from 0 to 17, corresponding to GPIO0 ~ GPIO17, respectively. To disable the Hold function of each pin after the chip is woken up, clear the bits above.
- Alternatively, set RTC_CNTL_DG_PAD_FORCE_HOLD to hold the values of all RTC pins, or set RTC_CNTL_DG_PAD_FORCE_UNHOLD to disable the hold function of all RTC pins.

## 6.8 I/O Pin Power Supplies

Figure 6.8-1 and 6.8-2 show the IO pin power supplies.

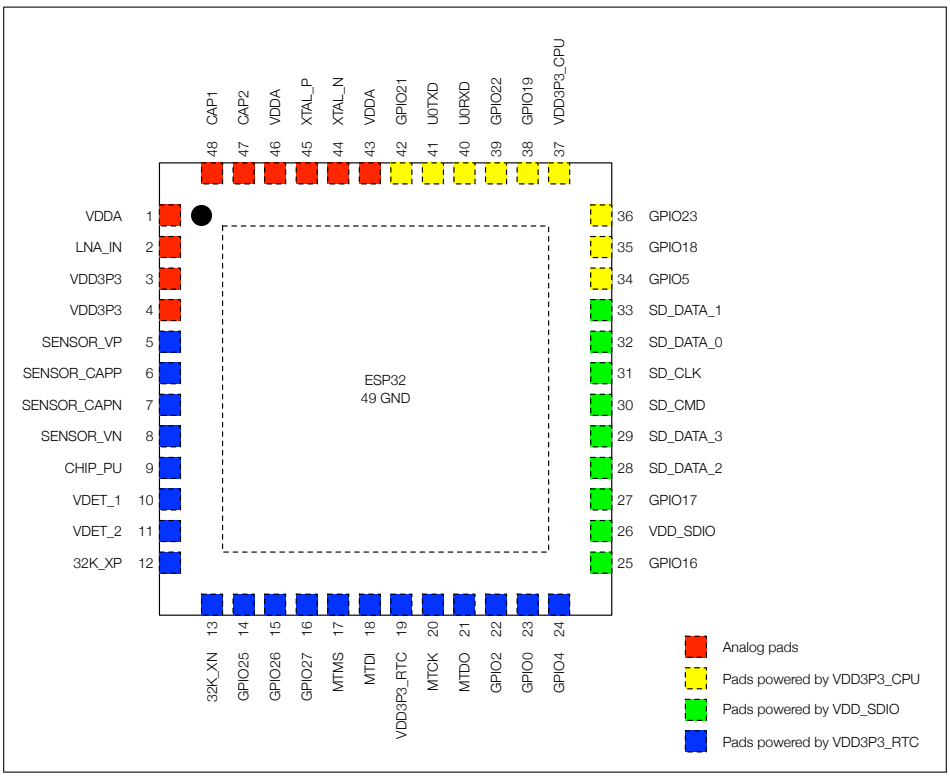



Figure 6.8-1. ESP32 I/O Pin Power Sources (QFN 6*6, Top View)

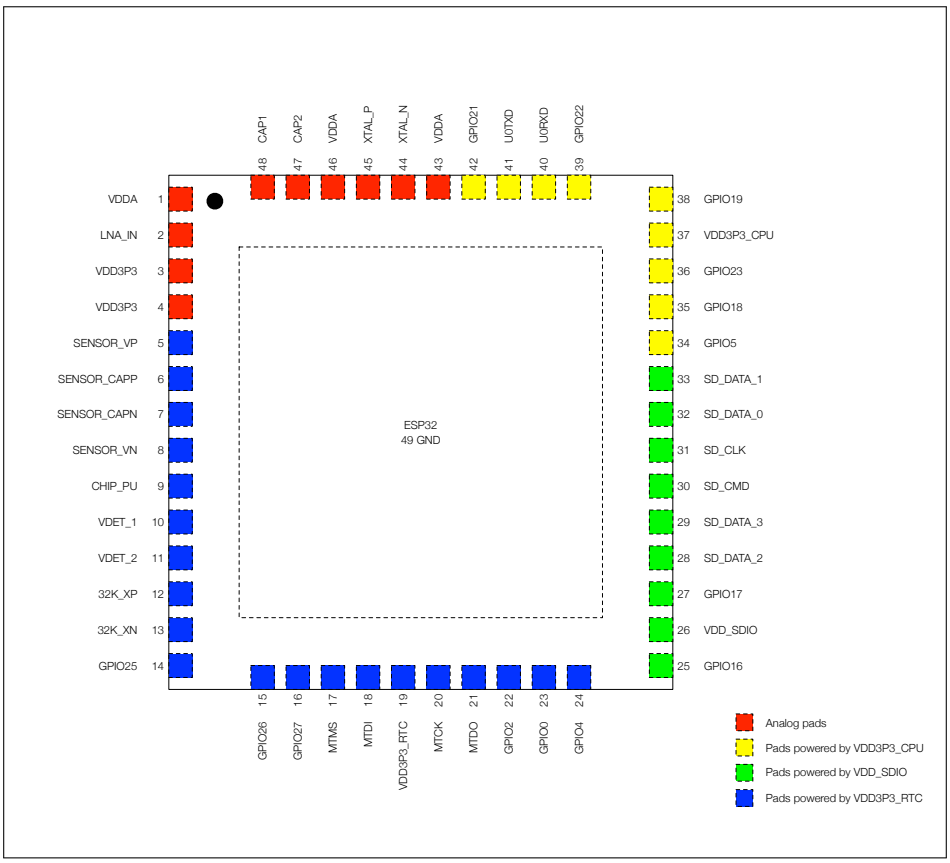



Figure 6.8-2. ESP32 I/O Pin Power Sources (QFN 5*5, Top View)

- Pins marked blue are RTC pins that have their individual analog function and can also act as normal digital IO pins. For details, please see Section 6.11.
- Pins marked yellow and green have digital functions only.
- Pins marked green can be powered externally or internally via VDD_SDIO (see below).

### 6.8.1 VDD_SDIO Power Domain

VDD_SDIO can source or sink current, allowing this power domain to be powered externally or internally. To power VDD_SDIO externally, apply the same power supply of VDD3P3_RTC to the VDD_SDIO pin.

Without an external power supply, the internal regulator will supply VDD_SDIO. The VDD_SDIO voltage can be configured to be either 1.8V or the same as VDD3P3_RTC, depending on the state of the MTDI pin at reset – a high level configures 1.8V and a low level configures the voltage to be the same as VDD3P3_RTC. Setting the efuse bit determines the default voltage of the VDD_SDIO. In addition, software can change the voltage of the VDD_SDIO by configuring register bits.

## 6.9 Peripheral Signal List

Table 6.9-1 shows the peripheral input/output signals via GPIO matrix.

Please pay attention to the configuration of the bit GPIO_FUNC*n*_OEN_SEL:

- GPIO_FUNC*n*_OEN_SEL = 1: the output enable is controlled by the corresponding bit *n* of GPIO_ENABLE_REG:
  - GPIO_ENABLE_REG = 0: output is disabled.
  - GPIO_ENABLE_REG = 1: output is enabled.
- GPIO_FUNC*n*_OEN_SEL = 0: use the output enable signal from peripheral, for example SPIQ_oe in the column "Output Enable of Output Signals" of Table 6.9-1. Note that the signals such as SPIQ_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in column "Output Enable of Output Signals", it indicates that once GPIO_FUNC*n*_OEN_SEL is cleared, the output signal is always enabled by default.

**Note:**

Signals are numbered consecutively, but not all signals are valid.

- Only the signals with a name assigned in the column "Input signal" in Table 6.9-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 6.9-1 are valid output signals.

Table 6.9-1. GPIO_Matrix

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 0 | SPICLK_in | 0 | yes | SPICLK_out | SPICLK_oe |
| 1 | SPIQ_in | 0 | yes | SPIQ_out | SPIQ_oe |
| 2 | SPID_in | 0 | yes | SPID_out | SPID_oe |
| 3 | SPIHD_in | 0 | yes | SPIHD_out | SPIHD_oe |
| 4 | SPIWP_in | 0 | yes | SPIWP_out | SPIWP_oe |
| 5 | SPICS0_in | 0 | yes | SPICS0_out | SPICS0_oe |







GoBack

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 6 | SPICS1_in | 0 | no | SPICS1_out | SPICS1_oe |
| 7 | SPICS2_in | 0 | no | SPICS2_out | SPICS2_oe |
| 8 | HSPICLK_in | 0 | yes | HSPICLK_out | HSPICLK_oe |
| 9 | HSPIQ_in | 0 | yes | HSPIQ_out | HSPIQ_oe |
| 10 | HSPID_in | 0 | yes | HSPID_out | HSPID_oe |
| 11 | HSPICS0_in | 0 | yes | HSPICS0_out | HSPICS0_oe |
| 12 | HSPIHD_in | 0 | yes | HSPIHD_out | HSPIHD_oe |
| 13 | HSPIWP_in | 0 | yes | HSPIWP_out | HSPIWP_oe |
| 14 | U0RXD_in | 0 | yes | U0TXD_out | 1'd1 |
| 15 | U0CTS_in | 0 | yes | U0RTS_out | 1'd1 |
| 16 | U0DSR_in | 0 | no | U0DTR_out | 1'd1 |
| 17 | U1RXD_in | 0 | yes | U1TXD_out | 1'd1 |
| 18 | U1CTS_in | 0 | yes | U1RTS_out | 1'd1 |
| 23 | I2S0O_BCK_in | 0 | no | I2S0O_BCK_out | 1'd1 |
| 24 | I2S1O_BCK_in | 0 | no | I2S1O_BCK_out | 1'd1 |
| 25 | I2S0O_WS_in | 0 | no | I2S0O_WS_out | 1'd1 |
| 26 | I2S1O_WS_in | 0 | no | I2S1O_WS_out | 1'd1 |
| 27 | I2S0I_BCK_in | 0 | no | I2S0I_BCK_out | 1'd1 |
| 28 | I2S0I_WS_in | 0 | no | I2S0I_WS_out | 1'd1 |
| 29 | I2CEXT0_SCL_in | 1 | no | I2CEXT0_SCL_out | 1'd1 |
| 30 | I2CEXT0_SDA_in | 1 | no | I2CEXT0_SDA_out | 1'd1 |
| 31 | pwm0_sync0_in | 0 | no | sdio_tohost_int_out | 1'd1 |
| 32 | pwm0_sync1_in | 0 | no | pwm0_out0a | 1'd1 |
| 33 | pwm0_sync2_in | 0 | no | pwm0_out0b | 1'd1 |
| 34 | pwm0_f0_in | 0 | no | pwm0_out1a | 1'd1 |
| 35 | pwm0_f1_in | 0 | no | pwm0_out1b | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 36 | pwm0_f2_in | 0 | no | pwm0_out2a | 1'd1 |
| 37 | — | 0 | no | pwm0_out2b | 1'd1 |
| 39 | pcnt_sig_ch0_in0 | 0 | no | — | 1'd1 |
| 40 | pcnt_sig_ch1_in0 | 0 | no | — | 1'd1 |
| 41 | pcnt_ctrl_ch0_in0 | 0 | no | — | 1'd1 |
| 42 | pcnt_ctrl_ch1_in0 | 0 | no | — | 1'd1 |
| 43 | pcnt_sig_ch0_in1 | 0 | no | — | 1'd1 |
| 44 | pcnt_sig_ch1_in1 | 0 | no | — | 1'd1 |
| 45 | pcnt_ctrl_ch0_in1 | 0 | no | — | 1'd1 |
| 46 | pcnt_ctrl_ch1_in1 | 0 | no | — | 1'd1 |
| 47 | pcnt_sig_ch0_in2 | 0 | no | — | 1'd1 |
| 48 | pcnt_sig_ch1_in2 | 0 | no | — | 1'd1 |
| 49 | pcnt_ctrl_ch0_in2 | 0 | no | — | 1'd1 |
| 50 | pcnt_ctrl_ch1_in2 | 0 | no | — | 1'd1 |
| 51 | pcnt_sig_ch0_in3 | 0 | no | — | 1'd1 |
| 52 | pcnt_sig_ch1_in3 | 0 | no | — | 1'd1 |
| 53 | pcnt_ctrl_ch0_in3 | 0 | no | — | 1'd1 |
| 54 | pcnt_ctrl_ch1_in3 | 0 | no | — | 1'd1 |
| 55 | pcnt_sig_ch0_in4 | 0 | no | — | 1'd1 |
| 56 | pcnt_sig_ch1_in4 | 0 | no | — | 1'd1 |
| 57 | pcnt_ctrl_ch0_in4 | 0 | no | — | 1'd1 |
| 58 | pcnt_ctrl_ch1_in4 | 0 | no | — | 1'd1 |
| 61 | HSPICS1_in | 0 | no | HSPICS1_out | HSPICS1_oe |
| 62 | HSPICS2_in | 0 | no | HSPICS2_out | HSPICS2_oe |
| 63 | VSPICLK_in | 0 | yes | VSPICLK_out_mux | VSPICLK_oe |
| 64 | VSPIQ_in | 0 | yes | VSPIQ_out | VSPIQ_oe |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 65 | VSPID_in | 0 | yes | VSPID_out | VSPID_oe |
| 66 | VSPIHD_in | 0 | yes | VSPIHD_out | VSPIHD_oe |
| 67 | VSPIWP_in | 0 | yes | VSPIWP_out | VSPIWP_oe |
| 68 | VSPICS0_in | 0 | yes | VSPICS0_out | VSPICS0_oe |
| 69 | VSPICS1_in | 0 | no | VSPICS1_out | VSPICS1_oe |
| 70 | VSPICS2_in | 0 | no | VSPICS2_out | VSPICS2_oe |
| 71 | pcnt_sig_ch0_in5 | 0 | no | ledc_hs_sig_out0 | 1'd1 |
| 72 | pcnt_sig_ch1_in5 | 0 | no | ledc_hs_sig_out1 | 1'd1 |
| 73 | pcnt_ctrl_ch0_in5 | 0 | no | ledc_hs_sig_out2 | 1'd1 |
| 74 | pcnt_ctrl_ch1_in5 | 0 | no | ledc_hs_sig_out3 | 1'd1 |
| 75 | pcnt_sig_ch0_in6 | 0 | no | ledc_hs_sig_out4 | 1'd1 |
| 76 | pcnt_sig_ch1_in6 | 0 | no | ledc_hs_sig_out5 | 1'd1 |
| 77 | pcnt_ctrl_ch0_in6 | 0 | no | ledc_hs_sig_out6 | 1'd1 |
| 78 | pcnt_ctrl_ch1_in6 | 0 | no | ledc_hs_sig_out7 | 1'd1 |
| 79 | pcnt_sig_ch0_in7 | 0 | no | ledc_ls_sig_out0 | 1'd1 |
| 80 | pcnt_sig_ch1_in7 | 0 | no | ledc_ls_sig_out1 | 1'd1 |
| 81 | pcnt_ctrl_ch0_in7 | 0 | no | ledc_ls_sig_out2 | 1'd1 |
| 82 | pcnt_ctrl_ch1_in7 | 0 | no | ledc_ls_sig_out3 | 1'd1 |
| 83 | rmt_sig_in0 | 0 | no | ledc_ls_sig_out4 | 1'd1 |
| 84 | rmt_sig_in1 | 0 | no | ledc_ls_sig_out5 | 1'd1 |
| 85 | rmt_sig_in2 | 0 | no | ledc_ls_sig_out6 | 1'd1 |
| 86 | rmt_sig_in3 | 0 | no | ledc_ls_sig_out7 | 1'd1 |
| 87 | rmt_sig_in4 | 0 | no | rmt_sig_out0 | 1'd1 |
| 88 | rmt_sig_in5 | 0 | no | rmt_sig_out1 | 1'd1 |
| 89 | rmt_sig_in6 | 0 | no | rmt_sig_out2 | 1'd1 |
| 90 | rmt_sig_in7 | 0 | no | rmt_sig_out3 | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 91 | — | — | — | rmt_sig_out4 | 1'd1 |
| 92 | — | — | — | rmt_sig_out6 | 1'd1 |
| 94 | twai_rx | 1 | no | rmt_sig_out7 | 1'd1 |
| 95 | I2CEXT1_SCL_in | 1 | no | I2CEXT1_SCL_out | 1'd1 |
| 96 | I2CEXT1_SDA_in | 1 | no | I2CEXT1_SDA_out | 1'd1 |
| 97 | host_card_detect_n_1 | 0 | no | host_ccmd_od_pullup_en_n | 1'd1 |
| 98 | host_card_detect_n_2 | 0 | no | host_rst_n_1 | 1'd1 |
| 99 | host_card_write_prt_1 | 0 | no | host_rst_n_2 | 1'd1 |
| 100 | host_card_write_prt_2 | 0 | no | gpio_sd0_out | 1'd1 |
| 101 | host_card_int_n_1 | 0 | no | gpio_sd1_out | 1'd1 |
| 102 | host_card_int_n_2 | 0 | no | gpio_sd2_out | 1'd1 |
| 103 | pwm1_sync0_in | 0 | no | gpio_sd3_out | 1'd1 |
| 104 | pwm1_sync1_in | 0 | no | gpio_sd4_out | 1'd1 |
| 105 | pwm1_sync2_in | 0 | no | gpio_sd5_out | 1'd1 |
| 106 | pwm1_f0_in | 0 | no | gpio_sd6_out | 1'd1 |
| 107 | pwm1_f1_in | 0 | no | gpio_sd7_out | 1'd1 |
| 108 | pwm1_f2_in | 0 | no | pwm1_out0a | 1'd1 |
| 109 | pwm0_cap0_in | 0 | no | pwm1_out0b | 1'd1 |
| 110 | pwm0_cap1_in | 0 | no | pwm1_out1a | 1'd1 |
| 111 | pwm0_cap2_in | 0 | no | pwm1_out1b | 1'd1 |
| 112 | pwm1_cap0_in | 0 | no | pwm1_out2a | 1'd1 |
| 113 | pwm1_cap1_in | 0 | no | pwm1_out2b | 1'd1 |
| 114 | pwm1_cap2_in | 0 | no | pwm2_out1h | 1'd1 |
| 115 | pwm2_flta | 1 | no | pwm2_out1l | 1'd1 |
| 116 | pwm2_fltb | 1 | no | pwm2_out2h | 1'd1 |
| 117 | pwm2_cap1_in | 0 | no | pwm2_out2l | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 118 | pwm2_cap2_in | 0 | no | pwm2_out3h | 1'd1 |
| 119 | pwm2_cap3_in | 0 | no | pwm2_out3l | 1'd1 |
| 120 | pwm3_flta | 1 | no | pwm2_out4h | 1'd1 |
| 121 | pwm3_fltb | 1 | no | pwm2_out4l | 1'd1 |
| 122 | pwm3_cap1_in | 0 | no | — | 1'd1 |
| 123 | pwm3_cap2_in | 0 | no | twai_tx | 1'd1 |
| 124 | pwm3_cap3_in | 0 | no | twai_bus_off_on | 1'd1 |
| 125 | — | — | — | twai_clkout | 1'd1 |
| 140 | I2S0I_DATA_in0 | 0 | no | I2S0O_DATA_out0 | 1'd1 |
| 141 | I2S0I_DATA_in1 | 0 | no | I2S0O_DATA_out1 | 1'd1 |
| 142 | I2S0I_DATA_in2 | 0 | no | I2S0O_DATA_out2 | 1'd1 |
| 143 | I2S0I_DATA_in3 | 0 | no | I2S0O_DATA_out3 | 1'd1 |
| 144 | I2S0I_DATA_in4 | 0 | no | I2S0O_DATA_out4 | 1'd1 |
| 145 | I2S0I_DATA_in5 | 0 | no | I2S0O_DATA_out5 | 1'd1 |
| 146 | I2S0I_DATA_in6 | 0 | no | I2S0O_DATA_out6 | 1'd1 |
| 147 | I2S0I_DATA_in7 | 0 | no | I2S0O_DATA_out7 | 1'd1 |
| 148 | I2S0I_DATA_in8 | 0 | no | I2S0O_DATA_out8 | 1'd1 |
| 149 | I2S0I_DATA_in9 | 0 | no | I2S0O_DATA_out9 | 1'd1 |
| 150 | I2S0I_DATA_in10 | 0 | no | I2S0O_DATA_out10 | 1'd1 |
| 151 | I2S0I_DATA_in11 | 0 | no | I2S0O_DATA_out11 | 1'd1 |
| 152 | I2S0I_DATA_in12 | 0 | no | I2S0O_DATA_out12 | 1'd1 |
| 153 | I2S0I_DATA_in13 | 0 | no | I2S0O_DATA_out13 | 1'd1 |
| 154 | I2S0I_DATA_in14 | 0 | no | I2S0O_DATA_out14 | 1'd1 |
| 155 | I2S0I_DATA_in15 | 0 | no | I2S0O_DATA_out15 | 1'd1 |
| 156 | — | — | — | I2S0O_DATA_out16 | 1'd1 |
| 157 | — | — | — | I2S0O_DATA_out17 | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 158 | — | — | — | I2S0O_DATA_out18 | 1'd1 |
| 159 | — | — | — | I2S0O_DATA_out19 | 1'd1 |
| 160 | — | — | — | I2S0O_DATA_out20 | 1'd1 |
| 161 | — | — | — | I2S0O_DATA_out21 | 1'd1 |
| 162 | — | — | — | I2S0O_DATA_out22 | 1'd1 |
| 163 | — | — | — | I2S0O_DATA_out23 | 1'd1 |
| 164 | I2S1I_BCK_in | 0 | no | I2S1I_BCK_out | 1'd1 |
| 165 | I2S1I_WS_in | 0 | no | I2S1I_WS_out | 1'd1 |
| 166 | I2S1I_DATA_in0 | 0 | no | I2S1O_DATA_out0 | 1'd1 |
| 167 | I2S1I_DATA_in1 | 0 | no | I2S1O_DATA_out1 | 1'd1 |
| 168 | I2S1I_DATA_in2 | 0 | no | I2S1O_DATA_out2 | 1'd1 |
| 169 | I2S1I_DATA_in3 | 0 | no | I2S1O_DATA_out3 | 1'd1 |
| 170 | I2S1I_DATA_in4 | 0 | no | I2S1O_DATA_out4 | 1'd1 |
| 171 | I2S1I_DATA_in5 | 0 | no | I2S1O_DATA_out5 | 1'd1 |
| 172 | I2S1I_DATA_in6 | 0 | no | I2S1O_DATA_out6 | 1'd1 |
| 173 | I2S1I_DATA_in7 | 0 | no | I2S1O_DATA_out7 | 1'd1 |
| 174 | I2S1I_DATA_in8 | 0 | no | I2S1O_DATA_out8 | 1'd1 |
| 175 | I2S1I_DATA_in9 | 0 | no | I2S1O_DATA_out9 | 1'd1 |
| 176 | I2S1I_DATA_in10 | 0 | no | I2S1O_DATA_out10 | 1'd1 |
| 177 | I2S1I_DATA_in11 | 0 | no | I2S1O_DATA_out11 | 1'd1 |
| 178 | I2S1I_DATA_in12 | 0 | no | I2S1O_DATA_out12 | 1'd1 |
| 179 | I2S1I_DATA_in13 | 0 | no | I2S1O_DATA_out13 | 1'd1 |
| 180 | I2S1I_DATA_in14 | 0 | no | I2S1O_DATA_out14 | 1'd1 |
| 181 | I2S1I_DATA_in15 | 0 | no | I2S1O_DATA_out15 | 1'd1 |
| 182 | — | — | — | I2S1O_DATA_out16 | 1'd1 |
| 183 | — | — | — | I2S1O_DATA_out17 | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 184 | — | — | — | I2S1O_DATA_out18 | 1'd1 |
| 185 | — | — | — | I2S1O_DATA_out19 | 1'd1 |
| 186 | — | — | — | I2S1O_DATA_out20 | 1'd1 |
| 187 | — | — | — | I2S1O_DATA_out21 | 1'd1 |
| 188 | — | — | — | I2S1O_DATA_out22 | 1'd1 |
| 189 | — | — | — | I2S1O_DATA_out23 | 1'd1 |
| 190 | I2SOI_H_SYNC | 0 | no | pwm3_out1h | 1'd1 |
| 191 | I2SOI_V_SYNC | 0 | no | pwm3_out1l | 1'd1 |
| 192 | I2SOI_H_ENABLE | 0 | no | pwm3_out2h | 1'd1 |
| 193 | I2S1I_H_SYNC | 0 | no | pwm3_out2l | 1'd1 |
| 194 | I2S1I_V_SYNC | 0 | no | pwm3_out3h | 1'd1 |
| 195 | I2S1I_H_ENABLE | 0 | no | pwm3_out3l | 1'd1 |
| 196 | — | — | — | pwm3_out4h | 1'd1 |
| 197 | — | — | — | pwm3_out4l | 1'd1 |
| 198 | U2RXD_in | 0 | yes | U2TXD_out | 1'd1 |
| 199 | U2CTS_in | 0 | yes | U2RTS_out | 1'd1 |
| 200 | emac_mdc_i | 0 | no | emac_mdc_o | emac_mdc_oe |
| 201 | emac_mdi_i | 0 | no | emac_mdo_o | emac_mdo_o_e |
| 202 | emac_crs_i | 0 | no | emac_crs_o | emac_crs_oe |
| 203 | emac_col_i | 0 | no | emac_col_o | emac_col_oe |
| 204 | pcmfsync_in | 0 | no | bt_audio0_irq | 1'd1 |
| 205 | pcmclk_in | 0 | no | bt_audio1_irq | 1'd1 |
| 206 | pcmdin | 0 | no | bt_audio2_irq | 1'd1 |
| 207 | — | — | — | ble_audio0_irq | 1'd1 |
| 208 | — | — | — | ble_audio1_irq | 1'd1 |
| 209 | — | — | — | ble_audio2_irq | 1'd1 |

| Signal No. | Input Signals | Default Value If Unassigned* | Same Input Signal from IO_MUX Core | Output Signals | Output Enable of Output Signals |
|---|---|---|---|---|---|
| 210 | — | — | — | pcmfsync_out | pcmfsync_en |
| 211 | — | — | — | pcmclk_out | pcmclk_en |
| 212 | — | — | — | pcmdout | pcmdout_en |
| 213 | — | — | — | ble_audio_sync0_p | 1'd1 |
| 214 | — | — | — | ble_audio_sync1_p | 1'd1 |
| 215 | — | — | — | ble_audio_sync2_p | 1'd1 |
| 224 | — | — | — | sig_in_func224 | 1'd1 |
| 225 | — | — | — | sig_in_func225 | 1'd1 |
| 226 | — | — | — | sig_in_func226 | 1'd1 |
| 227 | — | — | — | sig_in_func227 | 1'd1 |
| 228 | — | — | — | sig_in_func228 | 1'd1 |

## 6.10 IO MUX Pin List

Table 6.10-1 shows the IO MUX functions for each I/O pin:

**Table 6.10-1. IO_MUX Pin Summary**

| GPIO | Pin Name | Function 0 | Function 1 | Function 2 | Function 3 | Function 4 | Function 5 | Reset | Notes |
|---|---|---|---|---|---|---|---|---|---|
| 0 | GPIO0 | GPIO0 | CLK_OUT1 | GPIO0 | - | - | EMAC_TX_CLK | 3 | R |
| 1 | U0TXD | U0TXD | CLK_OUT3 | GPIO1 | - | - | EMAC_RXD2 | 3 | - |
| 2 | GPIO2 | GPIO2 | HSPIWP | GPIO2 | HS2_DATA0 | SD_DATA0 | - | 2 | R |
| 3 | U0RXD | U0RXD | CLK_OUT2 | GPIO3 | - | - | - | 3 | - |
| 4 | GPIO4 | GPIO4 | HSPIHD | GPIO4 | HS2_DATA1 | SD_DATA1 | EMAC_TX_ER | 2 | R |
| 5 | GPIO5 | GPIO5 | VSPICS0 | GPIO5 | HS1_DATA6 | - | EMAC_RX_CLK | 3 | - |
| 6 | SD_CLK | SD_CLK | SPICLK | GPIO6 | HS1_CLK | U1CTS | - | 3 | - |
| 7 | SD_DATA_0 | SD_DATA0 | SPIQ | GPIO7 | HS1_DATA0 | U2RTS | - | 3 | - |
| 8 | SD_DATA_1 | SD_DATA1 | SPID | GPIO8 | HS1_DATA1 | U2CTS | - | 3 | - |
| 9 | SD_DATA_2 | SD_DATA2 | SPIHD | GPIO9 | HS1_DATA2 | U1RXD | - | 3 | - |
| 10 | SD_DATA_3 | SD_DATA3 | SPIWP | GPIO10 | HS1_DATA3 | U1TXD | - | 3 | - |
| 11 | SD_CMD | SD_CMD | SPICS0 | GPIO11 | HS1_CMD | U1RTS | - | 3 | - |
| 12 | MTDI | MTDI | HSPIQ | GPIO12 | HS2_DATA2 | SD_DATA2 | EMAC_TXD3 | 2 | R |
| 13 | MTCK | MTCK | HSPID | GPIO13 | HS2_DATA3 | SD_DATA3 | EMAC_RX_ER | 2 | R |
| 14 | MTMS | MTMS | HSPICLK | GPIO14 | HS2_CLK | SD_CLK | EMAC_TXD2 | 3 | R |
| 15 | MTDO | MTDO | HSPICS0 | GPIO15 | HS2_CMD | SD_CMD | EMAC_RXD3 | 3 | R |
| 16 | GPIO16 | GPIO16 | - | GPIO16 | HS1_DATA4 | U2RXD | EMAC_CLK_OUT | 1 | - |
| 17 | GPIO17 | GPIO17 | - | GPIO17 | HS1_DATA5 | U2TXD | EMAC_CLK_180 | 1 | - |
| 18 | GPIO18 | GPIO18 | VSPICLK | GPIO18 | HS1_DATA7 | - | - | 1 | - |
| 19 | GPIO19 | GPIO19 | VSPIQ | GPIO19 | U0CTS | - | EMAC_TXD0 | 1 | - |
| 21 | GPIO21 | GPIO21 | VSPIHD | GPIO21 | - | - | EMAC_TX_EN | 1 | - |
| 22 | GPIO22 | GPIO22 | VSPIWP | GPIO22 | U0RTS | - | EMAC_TXD1 | 1 | - |
| 23 | GPIO23 | GPIO23 | VSPID | GPIO23 | HS1_STROBE | - | - | 1 | - |
| 25 | GPIO25 | GPIO25 | - | GPIO25 | - | - | EMAC_RXD0 | 0 | R |
| 26 | GPIO26 | GPIO26 | - | GPIO26 | - | - | EMAC_RXD1 | 0 | R |
| 27 | GPIO27 | GPIO27 | - | GPIO27 | - | - | EMAC_RX_DV | 0 | R |
| 32 | 32K_XP | GPIO32 | - | GPIO32 | - | - | - | 0 | R |
| 33 | 32K_XN | GPIO33 | - | GPIO33 | - | - | - | 0 | R |
| 34 | VDET_1 | GPIO34 | - | GPIO34 | - | - | - | 0 | R, I |
| 35 | VDET_2 | GPIO35 | - | GPIO35 | - | - | - | 0 | R, I |
| 36 | SENSOR_VP | GPIO36 | - | GPIO36 | - | - | - | 0 | R, I |
| 37 | SENSOR_CAPP | GPIO37 | - | GPIO37 | - | - | - | 0 | R, I |
| 38 | SENSOR_CAPN | GPIO38 | - | GPIO38 | - | - | - | 0 | R, I |
| 39 | SENSOR_VN | GPIO39 | - | GPIO39 | - | - | - | 0 | R, I |

### Reset Configurations

"Reset" column shows each pin's default configurations after reset:

- **0** - IE=0 (input disabled).
- **1** - IE=1 (input enabled).
- **2** - IE=1, WPD=1 (input enabled, pull-down resistor).
- **3** - IE=1, WPU=1 (input enabled, pull-up resistor).

### Notes

- **R** - Pin has RTC/analog functions via RTC_MUX.

- **I** - Pin can only be configured as input GPIO. These input-only pins do not feature an output driver or internal pull-up/pull-down circuitry.

Please refer to the ESP32 Pin Lists in *ESP32 Series Datasheet*or more details.

## 6.11 RTC_MUX Pin List

Table 6.11-1 shows the RTC pins and how they correspond to GPIO pins:

**Table 6.11-1. RTC IO MUX Pin Summary**

| RTC GPIO Num | GPIO Num | Pin Name | Analog Function | | | RTC Function | |
|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | Function 0 (FUN_SEL = 0) | Function 1 (FUN_SEL = 3) |
| 0 | 36 | SENSOR_VP | ADC_H | ADC1_CH0 | - | RTC_GPIO0 | - |
| 1 | 37 | SENSOR_CAPP | ADC_H | ADC1_CH1 | - | RTC_GPIO1 | - |
| 2 | 38 | SENSOR_CAPN | ADC_H | ADC1_CH2 | - | RTC_GPIO2 | - |
| 3 | 39 | SENSOR_VN | ADC_H | ADC1_CH3 | - | RTC_GPIO3 | - |
| 4 | 34 | VDET_1 | - | ADC1_CH6 | - | RTC_GPIO4 | - |
| 5 | 35 | VDET_2 | - | ADC1_CH7 | - | RTC_GPIO5 | - |
| 6 | 25 | GPIO25 | DAC_1 | ADC2_CH8 | - | RTC_GPIO6 | - |
| 7 | 26 | GPIO26 | DAC_2 | ADC2_CH9 | - | RTC_GPIO7 | - |
| 8 | 33 | 32K_XN | XTAL_32K_N | ADC1_CH5 | TOUCH8 | RTC_GPIO8 | - |
| 9 | 32 | 32K_XP | XTAL_32K_P | ADC1_CH4 | TOUCH9 | RTC_GPIO9 | - |
| 10 | 4 | GPIO4 | - | ADC2_CH0 | TOUCH0 | RTC_GPIO10 | I2C_SCL* |
| 11 | 0 | GPIO0 | - | ADC2_CH1 | TOUCH1 | RTC_GPIO11 | I2C_SDA* |
| 12 | 2 | GPIO2 | - | ADC2_CH2 | TOUCH2 | RTC_GPIO12 | I2C_SCL* |
| 13 | 15 | MTDO | - | ADC2_CH3 | TOUCH3 | RTC_GPIO13 | I2C_SDA* |
| 14 | 13 | MTCK | - | ADC2_CH4 | TOUCH4 | RTC_GPIO14 | - |
| 15 | 12 | MTDI | - | ADC2_CH5 | TOUCH5 | RTC_GPIO15 | - |
| 16 | 14 | MTMS | - | ADC2_CH6 | TOUCH6 | RTC_GPIO16 | - |
| 17 | 27 | GPIO27 | - | ADC2_CH7 | TOUCH7 | RTC_GPIO17 | - |

**Note:**
For more information on the configuration of sar_i2c_xx, see Section RTC I2C Controller in Chapter 1 *ULP Coprocessor (ULP)*.

## 6.12 Register Summary

### 6.12.1 GPIO Matrix Register Summary

**Table 6.12-1. GPIO Matrix Register Summary**

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_OUT_REG | GPIO 0-31 output register | 0x3FF44004 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_OUT_W1TS_REG | GPIO 0-31 output register_W1TS | 0x3FF44008 | WO |
| GPIO_OUT_W1TC_REG | GPIO 0-31 output register_W1TC | 0x3FF4400C | WO |
| GPIO_OUT1_REG | GPIO 32-39 output register | 0x3FF44010 | R/W |
| GPIO_OUT1_W1TS_REG | GPIO 32-39 output bit set register | 0x3FF44014 | WO |
| GPIO_OUT1_W1TC_REG | GPIO 32-39 output bit clear register | 0x3FF44018 | WO |
| GPIO_ENABLE_REG | GPIO 0-31 output enable register | 0x3FF44020 | R/W |
| GPIO_ENABLE_W1TS_REG | GPIO 0-31 output enable register_W1TS | 0x3FF44024 | WO |
| GPIO_ENABLE_W1TC_REG | GPIO 0-31 output enable register_W1TC | 0x3FF44028 | WO |
| GPIO_ENABLE1_REG | GPIO 32-39 output enable register | 0x3FF4402C | R/W |
| GPIO_ENABLE1_W1TS_REG | GPIO 32-39 output enable bit set register | 0x3FF44030 | WO |
| GPIO_ENABLE1_W1TC_REG | GPIO 32-39 output enable bit clear register | 0x3FF44034 | WO |
| GPIO_STRAP_REG | Bootstrap pin value register | 0x3FF44038 | RO |
| GPIO_IN_REG | GPIO 0-31 input register | 0x3FF4403C | RO |
| GPIO_IN1_REG | GPIO 32-39 input register | 0x3FF44040 | RO |
| GPIO_STATUS_REG | GPIO 0-31 interrupt status register | 0x3FF44044 | R/W |
| GPIO_STATUS_W1TS_REG | GPIO 0-31 interrupt status register_W1TS | 0x3FF44048 | WO |
| GPIO_STATUS_W1TC_REG | GPIO 0-31 interrupt status register_W1TC | 0x3FF4404C | WO |
| GPIO_STATUS1_REG | GPIO 32-39 interrupt status register1 | 0x3FF44050 | R/W |
| GPIO_STATUS1_W1TS_REG | GPIO 32-39 interrupt status bit set register | 0x3FF44054 | WO |
| GPIO_STATUS1_W1TC_REG | GPIO 32-39 interrupt status bit clear register | 0x3FF44058 | WO |
| GPIO_ACPU_INT_REG | GPIO 0-31 APP_CPU interrupt status | 0x3FF44060 | RO |
| GPIO_ACPU_NMI_INT_REG | GPIO 0-31 APP_CPU non-maskable interrupt status | 0x3FF44064 | RO |
| GPIO_PCPU_INT_REG | GPIO 0-31 PRO_CPU interrupt status | 0x3FF44068 | RO |
| GPIO_PCPU_NMI_INT_REG | GPIO 0-31 PRO_CPU non-maskable interrupt status | 0x3FF4406C | RO |
| GPIO_ACPU_INT1_REG | GPIO 32-39 APP_CPU interrupt status | 0x3FF44074 | RO |
| GPIO_ACPU_NMI_INT1_REG | GPIO 32-39 APP_CPU non-maskable interrupt status | 0x3FF44078 | RO |
| GPIO_PCPU_INT1_REG | GPIO 32-39 PRO_CPU interrupt status | 0x3FF4407C | RO |
| GPIO_PCPU_NMI_INT1_REG | GPIO 32-39 PRO_CPU non-maskable interrupt status | 0x3FF44080 | RO |
| GPIO_PIN0_REG | Configuration for GPIO pin 0 | 0x3FF44088 | R/W |
| GPIO_PIN1_REG | Configuration for GPIO pin 1 | 0x3FF4408C | R/W |
| GPIO_PIN2_REG | Configuration for GPIO pin 2 | 0x3FF44090 | R/W |
| ... | ... | | |
| GPIO_PIN38_REG | Configuration for GPIO pin 38 | 0x3FF44120 | R/W |
| GPIO_PIN39_REG | Configuration for GPIO pin 39 | 0x3FF44124 | R/W |
| GPIO_FUNC0_IN_SEL_CFG_REG | Peripheral function 0 input selection register | 0x3FF44130 | R/W |
| GPIO_FUNC1_IN_SEL_CFG_REG | Peripheral function 1 input selection register | 0x3FF44134 | R/W |
| ... | ... | | |
| GPIO_FUNC254_IN_SEL_CFG_REG | Peripheral function 254 input selection register | 0x3FF44528 | R/W |
| GPIO_FUNC255_IN_SEL_CFG_REG | Peripheral function 255 input selection register | 0x3FF4452C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| GPIO_FUNC0_OUT_SEL_CFG_REG | Peripheral output selection for GPIO 0 | 0x3FF44530 | R/W |
| GPIO_FUNC1_OUT_SEL_CFG_REG | Peripheral output selection for GPIO 1 | 0x3FF44534 | R/W |
| ... | ... | | |

## 6.12.2 IO MUX Register Summary

Table 6.12-2. IO MUX Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| IO_MUX_PIN_CTRL | Clock output configuration register | 0x3FF49000 | R/W |
| IO_MUX_GPIO36_REG | Configuration register for GPIO36 | 0x3FF49004 | R/W |
| IO_MUX_GPIO37_REG | Configuration register for GPIO37 | 0x3FF49008 | R/W |
| IO_MUX_GPIO38_REG | Configuration register for GPIO38 | 0x3FF4900C | R/W |
| IO_MUX_GPIO39_REG | Configuration register for GPIO39 | 0x3FF49010 | R/W |
| IO_MUX_GPIO34_REG | Configuration register for GPIO34 | 0x3FF49014 | R/W |
| IO_MUX_GPIO35_REG | Configuration register for GPIO35 | 0x3FF49018 | R/W |
| IO_MUX_GPIO32_REG | Configuration register for GPIO32 | 0x3FF4901C | R/W |
| IO_MUX_GPIO33_REG | Configuration register for GPIO33 | 0x3FF49020 | R/W |
| IO_MUX_GPIO25_REG | Configuration register for GPIO25 | 0x3FF49024 | R/W |
| IO_MUX_GPIO26_REG | Configuration register for GPIO26 | 0x3FF49028 | R/W |
| IO_MUX_GPIO27_REG | Configuration register for GPIO27 | 0x3FF4902C | R/W |
| IO_MUX_MTMS_REG | Configuration register for MTMS | 0x3FF49030 | R/W |
| IO_MUX_MTDI_REG | Configuration register for MTDI | 0x3FF49034 | R/W |
| IO_MUX_MTCK_REG | Configuration register for MTCK | 0x3FF49038 | R/W |
| IO_MUX_MTDO_REG | Configuration register for MTDO | 0x3FF4903C | R/W |
| IO_MUX_GPIO2_REG | Configuration register for GPIO2 | 0x3FF49040 | R/W |
| IO_MUX_GPIO0_REG | Configuration register for GPIO0 | 0x3FF49044 | R/W |
| IO_MUX_GPIO4_REG | Configuration register for GPIO4 | 0x3FF49048 | R/W |
| IO_MUX_GPIO16_REG | Configuration register for GPIO16 | 0x3FF4904C | R/W |
| IO_MUX_GPIO17_REG | Configuration register for GPIO17 | 0x3FF49050 | R/W |
| IO_MUX_SD_DATA2_REG | Configuration register for SD_DATA2 | 0x3FF49054 | R/W |
| IO_MUX_SD_DATA3_REG | Configuration register for SD_DATA3 | 0x3FF49058 | R/W |
| IO_MUX_SD_CMD_REG | Configuration register for SD_CMD | 0x3FF4905C | R/W |
| IO_MUX_SD_CLK_REG | Configuration register for SD_CLK | 0x3FF49060 | R/W |
| IO_MUX_SD_DATA0_REG | Configuration register for SD_DATA0 | 0x3FF49064 | R/W |
| IO_MUX_SD_DATA1_REG | Configuration register for SD_DATA1 | 0x3FF49068 | R/W |
| IO_MUX_GPIO5_REG | Configuration register for GPIO5 | 0x3FF4906C | R/W |
| IO_MUX_GPIO18_REG | Configuration register for GPIO18 | 0x3FF49070 | R/W |
| IO_MUX_GPIO19_REG | Configuration register for GPIO19 | 0x3FF49074 | R/W |
| IO_MUX_GPIO20_REG | Configuration register for GPIO20[1] | 0x3FF49078 | R/W |
| IO_MUX_GPIO21_REG | Configuration register for GPIO21 | 0x3FF4907C | R/W |
| IO_MUX_GPIO22_REG | Configuration register for GPIO22 | 0x3FF49080 | R/W |
| IO_MUX_U0RXD_REG | Configuration register for U0RXD | 0x3FF49084 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| IO_MUX_U0TXD_REG | Configuration register for U0TXD | 0x3FF49088 | R/W |
| IO_MUX_GPIO23_REG | Configuration register for GPIO23 | 0x3FF4908C | R/W |

1. GPIO20 is only available for ESP32-PICO-V3 and ESP32-PICO-V3-02. Please refer to *ESP32-PICO Series Datasheet* for more information.

### 6.12.3 RTC IO MUX Register Summary

Table 6.12-3. RTC IO MUX Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| **GPIO configuration / data registers** | | | |
| RTCIO_RTC_GPIO_OUT_REG | RTC GPIO output register | 0x3FF48400 | R/W |
| RTCIO_RTC_GPIO_OUT_W1TS_REG | RTC GPIO output bit set register | 0x3FF48404 | WO |
| RTCIO_RTC_GPIO_OUT_W1TC_REG | RTC GPIO output bit clear register | 0x3FF48408 | WO |
| RTCIO_RTC_GPIO_ENABLE_REG | RTC GPIO output enable register | 0x3FF4840C | R/W |
| RTCIO_RTC_GPIO_ENABLE_W1TS_REG | RTC GPIO output enable bit set register | 0x3FF48410 | WO |
| RTCIO_RTC_GPIO_ENABLE_W1TC_REG | RTC GPIO output enable bit clear register | 0x3FF48414 | WO |
| RTCIO_RTC_GPIO_STATUS_REG | RTC GPIO interrupt status register | 0x3FF48418 | WO |
| RTCIO_RTC_GPIO_STATUS_W1TS_REG | RTC GPIO interrupt status bit set register | 0x3FF4841C | WO |
| RTCIO_RTC_GPIO_STATUS_W1TC_REG | RTC GPIO interrupt status bit clear register | 0x3FF48420 | WO |
| RTCIO_RTC_GPIO_IN_REG | RTC GPIO input register | 0x3FF48424 | RO |
| RTCIO_RTC_GPIO_PIN0_REG | RTC configuration for pin 0 | 0x3FF48428 | R/W |
| RTCIO_RTC_GPIO_PIN1_REG | RTC configuration for pin 1 | 0x3FF4842C | R/W |
| RTCIO_RTC_GPIO_PIN2_REG | RTC configuration for pin 2 | 0x3FF48430 | R/W |
| RTCIO_RTC_GPIO_PIN3_REG | RTC configuration for pin 3 | 0x3FF48434 | R/W |
| RTCIO_RTC_GPIO_PIN4_REG | RTC configuration for pin 4 | 0x3FF48438 | R/W |
| RTCIO_RTC_GPIO_PIN5_REG | RTC configuration for pin 5 | 0x3FF4843C | R/W |
| RTCIO_RTC_GPIO_PIN6_REG | RTC configuration for pin 6 | 0x3FF48440 | R/W |
| RTCIO_RTC_GPIO_PIN7_REG | RTC configuration for pin 7 | 0x3FF48444 | R/W |
| RTCIO_RTC_GPIO_PIN8_REG | RTC configuration for pin 8 | 0x3FF48448 | R/W |
| RTCIO_RTC_GPIO_PIN9_REG | RTC configuration for pin 9 | 0x3FF4844C | R/W |
| RTCIO_RTC_GPIO_PIN10_REG | RTC configuration for pin 10 | 0x3FF48450 | R/W |
| RTCIO_RTC_GPIO_PIN11_REG | RTC configuration for pin 11 | 0x3FF48454 | R/W |
| RTCIO_RTC_GPIO_PIN12_REG | RTC configuration for pin 12 | 0x3FF48458 | R/W |
| RTCIO_RTC_GPIO_PIN13_REG | RTC configuration for pin 13 | 0x3FF4845C | R/W |
| RTCIO_RTC_GPIO_PIN14_REG | RTC configuration for pin 14 | 0x3FF48460 | R/W |
| RTCIO_RTC_GPIO_PIN15_REG | RTC configuration for pin 15 | 0x3FF48464 | R/W |
| RTCIO_RTC_GPIO_PIN16_REG | RTC configuration for pin 16 | 0x3FF48468 | R/W |
| RTCIO_RTC_GPIO_PIN17_REG | RTC configuration for pin 17 | 0x3FF4846C | R/W |
| RTCIO_DIG_PAD_HOLD_REG | RTC GPIO hold register | 0x3FF48474 | R/W |
| **GPIO RTC function configuration registers** | | | |
| RTCIO_SENSOR_PADS_REG | Sensor pins configuration register | 0x3FF4847C | R/W |
| RTCIO_ADC_PAD_REG | ADC configuration register | 0x3FF48480 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| RTCIO_PAD_DAC1_REG | DAC1 configuration register | 0x3FF48484 | R/W |
| RTCIO_PAD_DAC2_REG | DAC2 configuration register | 0x3FF48488 | R/W |
| RTCIO_XTAL_32K_PAD_REG | 32KHz crystal pins configuration register | 0x3FF4848C | R/W |
| RTCIO_TOUCH_CFG_REG | Touch sensor configuration register | 0x3FF48490 | R/W |
| RTCIO_TOUCH_PAD0_REG | Touch pin configuration register | 0x3FF48494 | R/W |
| ... | ... | | |
| RTCIO_TOUCH_PAD9_REG | Touch pin configuration register | 0x3FF484B8 | R/W |
| RTCIO_EXT_WAKEUP0_REG | External wake up configuration register | 0x3FF484BC | R/W |
| RTCIO_XTL_EXT_CTR_REG | Crystal power down enable GPIO source | 0x3FF484C0 | R/W |
| RTCIO_SAR_I2C_IO_REG | RTC I2C pin selection | 0x3FF484C4 | R/W |

# 6.13 Registers

## 6.13.1 GPIO Matrix Registers

The addresses in parenthesis besides register names are the register addresses relative to the GPIO base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 6.12.1 *GPIO Matrix Register Summary*.

**Register 6.1. GPIO_OUT_REG (0x0004)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_OUT_REG** GPIO0-31 output value. (R/W)

**Register 6.2. GPIO_OUT_W1TS_REG (0x0008)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_OUT_W1TS_REG** GPIO0-31 output set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be set. (WO)

**Register 6.3. GPIO_OUT_W1TC_REG (0x000c)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_OUT_W1TC_REG** GPIO0-31 output clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be cleared. (WO)

**Register 6.4. GPIO_OUT1_REG (0x0010)**

| (reserved) | GPIO_OUT_DATA | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_OUT_DATA** GPIO32-39 output value. (R/W)

**Register 6.5. GPIO_OUT1_W1TS_REG (0x0014)**

| (reserved) | GPIO_OUT_DATA | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_OUT_DATA** GPIO32-39 output value set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be set. (WO)

**Register 6.6. GPIO_OUT1_W1TC_REG (0x0018)**

| (reserved) | GPIO_OUT_DATA | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_OUT_DATA** GPIO32-39 output value clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be cleared. (WO)

**Register 6.7. GPIO_ENABLE_REG (0x0020)**

| 31 … 0 | |
|---|---|
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_ENABLE_REG** GPIO0-31 output enable. (R/W)

### Register 6.8. GPIO_ENABLE_W1TS_REG (0x0024)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_ENABLE_W1TS_REG** GPIO0-31 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be set. (WO)

### Register 6.9. GPIO_ENABLE_W1TC_REG (0x0028)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_ENABLE_W1TC_REG** GPIO0-31 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be cleared. (WO)

### Register 6.10. GPIO_ENABLE1_REG (0x002c)

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | GPIO_ENABLE_DATA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | | | | | | | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | Reset |

**GPIO_ENABLE_DATA** GPIO32-39 output enable. (R/W)

### Register 6.11. GPIO_ENABLE1_W1TS_REG (0x0030)

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | GPIO_ENABLE_DATA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | | | | | | | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | Reset |

**GPIO_ENABLE_DATA** GPIO32-39 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be set. (WO)

**Register 6.12. GPIO_ENABLE1_W1TC_REG (0x0034)**

| (reserved) | GPIO_ENABLE_DATA | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_ENABLE_DATA** GPIO32-39 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be cleared. (WO)

**Register 6.13. GPIO_STRAP_REG (0x0038)**

| (reserved) | GPIO_STRAPPING | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x x x x x x x x x | Reset |

**GPIO_STRAPPING** GPIO strapping results: Bit5-bit0 of boot_sel_chip[5:0] correspond to MTDI, GPIO0, GPIO2, GPIO4, MTDO, GPIO5, respectively.

**Register 6.14. GPIO_IN_REG (0x003c)**

| | |
|---|---|
| 31 … 0 | |
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_IN_REG** GPIO0-31 input value. Each bit represents a pin input value, 1 for high level and 0 for low level. (RO)

**Register 6.15. GPIO_IN1_REG (0x0040)**

| (reserved) | GPIO_IN_DATA_NEXT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_IN_DATA_NEXT** GPIO32-39 input value. Each bit represents a pin input value. (RO)

**Register 6.16. GPIO_STATUS_REG (0x0044)**

| GPIO_STATUS_INT | |
|---|---|
| 31 … 0 | |
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_STATUS_INT** GPIO0-31 interrupt status register. Each bit can be either of the two interrupt sources for the two CPUs. The enable bits in GPIO_PIN*n*_INT_ENA, corresponding to the 13-16 bits in GPIO_PIN*n*_REG should be set to 1. (R/W)

**Register 6.17. GPIO_STATUS_W1TS_REG (0x0048)**

| GPIO_STATUS_INT_W1TS | |
|---|---|
| 31 … 0 | |
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_STATUS_INT_W1TS** GPIO0-31 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INT will be set. (WO)

**Register 6.18. GPIO_STATUS_W1TC_REG (0x004c)**

| GPIO_STATUS_INT_W1TC | |
|---|---|
| 31 … 0 | |
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_STATUS_INT_W1TC** GPIO0-31 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INT will be cleared. (WO)

### Register 6.19. GPIO_STATUS1_REG (0x0050)

| (reserved) | GPIO_STATUS1_INT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_STATUS1_INT** GPIO32-39 interrupt status register. Each bit can be either of the two interrupt sources for the two CPUs. The enable bits in GPIO_PIN*n*_INT_ENA, corresponding to the 13-16 bits in GPIO_PIN*n*_REG should be set to 1. (R/W)

### Register 6.20. GPIO_STATUS1_W1TS_REG (0x0054)

| (reserved) | GPIO_STATUS1_INT_W1TS | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_STATUS1_INT_W1TS** GPIO32-39 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS1_INT will be set. (WO)

### Register 6.21. GPIO_STATUS1_W1TC_REG (0x0058)

| (reserved) | GPIO_STATUS1_INT_W1TC | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_STATUS1_INT_W1TC** GPIO32-39 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS1_INT will be cleared. (WO)

### Register 6.22. GPIO_ACPU_INT_REG (0x0060)

| 31 … 0 | |
|---|---|
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**GPIO_ACPU_INT_REG** GPIO0-31 APP CPU interrupt status. (RO)

### Register 6.23. GPIO_ACPU_NMI_INT_REG (0x0064)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_ACPU_NMI_INT_REG** GPIO0-31 APP CPU non-maskable interrupt status. (RO)

### Register 6.24. GPIO_PCPU_INT_REG (0x0068)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_PCPU_INT_REG** GPIO0-31 PRO CPU interrupt status. (RO)

### Register 6.25. GPIO_PCPU_NMI_INT_REG (0x006c)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**GPIO_PCPU_NMI_INT_REG** GPIO0-31 PRO CPU non-maskable interrupt status. (RO)

### Register 6.26. GPIO_ACPU_INT1_REG (0x0074)

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | GPIO_APPCPU_INT | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | | | | | | | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | Reset |

**GPIO_APPCPU_INT** GPIO32-39 APP CPU interrupt status. (RO)

### Register 6.27. GPIO_ACPU_NMI_INT1_REG (0x0078)

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | GPIO_APPCPU_NMI_INT | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | | | | | | | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | Reset |

**GPIO_APPCPU_NMI_INT** GPIO32-39 APP CPU non-maskable interrupt status. (RO)

**Register 6.28. GPIO_PCPU_INT1_REG (0x007c)**

| (reserved) | GPIO_PROCPU_INT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_PROCPU_INT** GPIO32-39 PRO CPU interrupt status. (RO)

**Register 6.29. GPIO_PCPU_NMI_INT1_REG (0x0080)**

| (reserved) | GPIO_PROCPU_NMI_INT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x x x x | Reset |

**GPIO_PROCPU_NMI_INT** GPIO32-39 PRO CPU non-maskable interrupt status. (RO)

**Register 6.30. GPIO_PINn_REG (n: 0-19, 21-23, 25-27, 32-39) (0x88+0x4*n)**

| (reserved) | GPIO_PINn_INT_ENA | (reserved) | GPIO_PINn_WAKEUP_ENABLE | GPIO_PINn_INT_TYPE | (reserved) | GPIO_PINn_PAD_DRIVER | (reserved) | |
|---|---|---|---|---|---|---|---|---|
| 31 18 | 17 13 | 12 11 | 10 | 9 7 | 6 3 | 2 | 1 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x x x x x | 0 0 | x | x x x | 0 0 0 0 | x | 0 0 | Reset |

**GPIO_PINn_INT_ENA** Interrupt enable bits for pin n: (R/W)

bit0: APP CPU interrupt enable;

bit1: APP CPU non-maskable interrupt enable;

bit2: PRO CPU interrupt enable;

bit3: PRO CPU non-maskable interrupt enable.

**GPIO_PINn_WAKEUP_ENABLE** GPIO wake-up enable will only wake up the CPU from Light-sleep. (R/W)

**GPIO_PINn_INT_TYPE** Interrupt type selection: (R/W)

0: GPIO interrupt disable;

1: rising edge trigger;

2: falling edge trigger;

3: any edge trigger;

4: low level trigger;

5: high level trigger.

**GPIO_PINn_PAD_DRIVER** 0: normal output; 1: open drain output. (R/W)

**Register 6.31. GPIO_FUNCy_IN_SEL_CFG_REG (y: 0-255) (0x130+0x4*y)**

| (reserved) | GPIO_SIGy_IN_SEL | GPIO_FUNCy_IN_INV_SEL | GPIO_FUNCy_IN_SEL | |
|---|---|---|---|---|
| 31 8 | 7 | 6 | 5 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x | x | x x x x x x | Reset |

**GPIO_SIGy_IN_SEL** Bypass the GPIO Matrix. 1: route through GPIO Matrix, 0: connect signal directly to peripheral configured in the IO_MUX. (R/W)

**GPIO_FUNCy_IN_INV_SEL** Invert the input value. 1: invert; 0: do not invert. (R/W)

**GPIO_FUNCy_IN_SEL** Selection control for peripheral input y. A value of 0-39 selects which of the 40 GPIO Matrix input pins this signal is connected to, or 0x38 for a constantly high input or 0x30 for a constantly low input. (R/W)

**Register 6.32. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-19, 21-23, 25-27, 32-33) (0x530+0x4**n*)**

| (reserved) | GPIO_FUNC*n*_OEN_INV_SEL | GPIO_FUNC*n*_OEN_SEL | GPIO_FUNC*n*_OUT_INV_SEL | GPIO_FUNC*n*_OUT_SEL | |
|---|---|---|---|---|---|
| 31 … 12 | 11 | 10 | 9 | 8 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | x | x | x | x x x x x x x x x | Reset |

**GPIO_FUNC*n*_OEN_INV_SEL** 1: Invert the output enable signal; 0: do not invert the output enable signal. (R/W)

[GPIO_FUNC*n*_OEN_SEL]1: Force the output enable signal to be sourced from bit *n* of GPIO_ENABLE_REG; 0: use output enable signal from peripheral. (R/W)

**GPIO_FUNC*n*_OUT_INV_SEL** 1: Invert the output value; 0: do not invert the output value. (R/W)

**GPIO_FUNC*n*_OUT_SEL** Selection control for GPIO output *n*. A value of *s* (0<=*s*<256) connects peripheral output *s* to GPIO output *n*. A value of 256 selects bit *n* of GPIO_OUT_REG/GPIO_OUT1_REG and GPIO_ENABLE_REG/GPIO_ENABLE1_REG as the output value and output enable. (R/W)

## 6.13.2 IO MUX Registers

**Register 6.33. IO_MUX_PIN_CTRL (0x00)**

| (reserved) | PIN_CTRL_CLK3 | PIN_CTRL_CLK2 | PIN_CTRL_CLK1 | |
|---|---|---|---|---|
| 31 … 12 | 11 … 8 | 7 … 4 | 3 … 0 | |
| 0x0 | 0x0 | 0x0 | 0x0 | Reset |

**If you want to output clock for I2S*0* (I2S*0*_CLK) to:**

CLK_OUT1, then set PIN_CTRL[3:0] = 0x0;

CLK_OUT2, then set PIN_CTRL[3:0] = 0x0 and PIN_CTRL[7:4] = 0x0;

CLK_OUT3, then set PIN_CTRL[3:0] = 0x0 and PIN_CTRL[11:8] = 0x0.

**If you want to output clock for I2S*1* (I2S*1*_CLK) to:**

CLK_OUT1, then set PIN_CTRL[3:0] = 0xF;

CLK_OUT2, then set PIN_CTRL[3:0] = 0xF and PIN_CTRL[7:4] = 0x0;

CLK_OUT3, then set PIN_CTRL[3:0] = 0xF and PIN_CTRL[11:8] = 0x0.

**If you want to output clock for APLL to**

CLK_OUT1, then set PIN_CTRL[3:0] = 0x6;

CLK_OUT2, then set PIN_CTRL[3:0] = 0x6 and PIN_CTRL[7:4] = 0x6;

CLK_OUT3, then set PIN_CTRL[3:0] = 0x6 and PIN_CTRL[11:8] = 0x6. (R/W)

**Note:**

- Only the above mentioned combinations of clock source (i.e. I2S*0/1*_CLK, APLL clock) and clock output pins (i.e. CLK_OUT1 ~ 3) are possible.
- The CLK_OUT1 ~ 3 can be found in the IO MUX Pin Summary.

**Register 6.34. IO_MUX_*x*_REG (*x*: 0-19, 20[1], 21-23, 25-27, 32-39) (See Table 6.12-2 for the addresses)**

| 31–15 | 14–12 | 11–10 | 9 | 8 | 7 | 6–5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | MCU_SEL | FUN_DRV | FUN_IE | FUN_WPU | FUN_WPD | MCU_DRV | MCU_IE | MCU_WPU | MCU_WPD | SLP_SEL | MCU_OE | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0 | 0x2 | 0 | 0 | 0 | 0x0 | 0 | 0 | 0 | 0 | 0 | Reset |

**MCU_SEL** Select the IO_MUX function for this signal. 0 selects Function 0, 1 selects Function 1, etc. (R/W)

**FUN_DRV** Select the drive strength of the pin. A higher value corresponds with a higher strength. For GPIO34-39, FUN_DRV is always 0. For detailed drive strength, please see note 8 in Table "Notes on ESP32 Pin Lists", in *ESP32 Datasheet*. (R/W)

**FUN_IE** Input enable of the pin. 1: input enabled; 0: input disabled. (R/W)

**FUN_WPU** Pull-up enable of the pin. 1: internal pull-up enabled; 0: internal pull-up disabled. GPIO pins 34-39 are input-only. These pins do not feature an output driver or internal pull- up/pull-down circuitry, therefore, their FUN_WPU is always 0. (R/W)

**FUN_WPD** Pull-down enable of the pin. 1: internal pull-down enabled, 0: internal pull-down disabled. GPIO pins 34-39 are input-only. These pins do not feature an output driver or internal pull- up/pull-down circuitry, therefore, their FUN_WPD is always 0. (R/W)

**MCU_DRV** Select the drive strength of the pin during sleep mode. A higher value corresponds with a higher strength. (R/W)

**MCU_IE** Input enable of the pin during sleep mode. 1: input enabled; 0: input disabled. (R/W)

**MCU_WPU** Pull-up enable of the pin during sleep mode. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

**MCU_WPD** Pull-down enable of the pin during sleep mode. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

**SLP_SEL** Sleep mode selection of this pin. Set to 1 to put the pin in sleep mode. (R/W)

**MCU_OE** Output enable of the pin in sleep mode. 1: enable output; 0: disable output. (R/W)

**Note:**

1. GPIO20 is available only on ESP32-PICO-V3 and ESP32-PICO-V3-02.

## 6.13.3 RTC IO MUX Registers

The addresses in parenthesis besides register names are the register addresses relative to (the RTC base address + 0x0400 = 0x3FF4_8400). The RTC base address is provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 6.12.3 *RTC IO MUX Register Summary*.

Register 6.35. RTCIO_RTC_GPIO_OUT_REG (0x0000)

| RTCIO_RTC_GPIO_OUT_DATA | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_OUT_DATA** GPIO0-17 output register. Bit14 is GPIO[0], bit15 is GPIO[1], etc. (R/W)

Register 6.36. RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0004)

| RTCIO_RTC_GPIO_OUT_DATA_W1TS | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_OUT_DATA_W1TS** GPIO0-17 output set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be set. (WO)

Register 6.37. RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0008)

| RTCIO_RTC_GPIO_OUT_DATA_W1TC | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_OUT_DATA_W1TC** GPIO0-17 output clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be cleared. (WO)

**Register 6.38. RTCIO_RTC_GPIO_ENABLE_REG (0x000C)**

| RTCIO_RTC_GPIO_ENABLE | | | | | | | | | | | | | | | | (reserved) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | 14 | 13 | | | | | | | | | | | | | | 0 |
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**RTCIO_RTC_GPIO_ENABLE** GPIO0-17 output enable. Bit14 is GPIO[0], bit15 is GPIO[1], etc. 1 means this GPIO pin is output. (R/W)

**Register 6.39. RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0010)**

| RTCIO_RTC_GPIO_ENABLE_W1TS | | | | | | | | | | | | | | | | (reserved) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | 14 | 13 | | | | | | | | | | | | | | 0 |
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**RTCIO_RTC_GPIO_ENABLE_W1TS** GPIO0-17 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be set. (WO)

**Register 6.40. RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0014)**

| RTCIO_RTC_GPIO_ENABLE_W1TC | | | | | | | | | | | | | | | | (reserved) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | 14 | 13 | | | | | | | | | | | | | | 0 |
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**RTCIO_RTC_GPIO_ENABLE_W1TC** GPIO0-17 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be cleared. (WO)

**Register 6.41. RTCIO_RTC_GPIO_STATUS_REG (0x0018)**

| RTCIO_RTC_GPIO_STATUS_INT | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_STATUS_INT** GPIO0-17 interrupt status. Bit14 is GPIO[0], bit15 is GPIO[1], etc. This register should be used together with RTCIO_RTC_GPIO_PIN*n*_INT_TYPE in RTCIO_RTC_GPIO_PIN*n*_REG. 1: corresponding interrupt; 0: no interrupt. (R/W)

**Register 6.42. RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x001C)**

| RTCIO_RTC_GPIO_STATUS_INT_W1TS | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_STATUS_INT_W1TS** GPIO0-17 interrupt set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be set. (WO)

**Register 6.43. RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0020)**

| RTCIO_RTC_GPIO_STATUS_INT_W1TC | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_STATUS_INT_W1TC** GPIO0-17 interrupt clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be cleared. (WO)

**Register 6.44. RTCIO_RTC_GPIO_IN_REG (0x0024)**

| RTCIO_RTC_GPIO_IN_NEXT | (reserved) | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| x x x x x x x x x x x x x x x x | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_RTC_GPIO_IN_NEXT** GPIO0-17 input value. Bit14 is GPIO[0], bit15 is GPIO[1], etc. Each bit represents a pin input value, 1 for high level, and 0 for low level. (RO)

**Register 6.45. RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-17) (0x28+4**n*)**

| (reserved) | RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE | RTCIO_RTC_GPIO_PIN*n*_INT_TYPE | (reserved) | RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER | (reserved) | |
|---|---|---|---|---|---|---|
| 31 … 11 | 10 | 9 … 7 | 6 … 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 | x | x x x | 0 0 0 0 | x | 0 0 | Reset |

**RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE** GPIO wake-up enable. This will only wake up the ESP32 from Light-sleep. (R/W)

**RTCIO_RTC_GPIO_PIN*n*_INT_TYPE** GPIO interrupt type selection. (R/W)

0: GPIO interrupt disable;
1: rising edge trigger;
2: falling edge trigger;
3: any edge trigger;
4: low level trigger;
5: high level trigger.

**RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER** Pin driver selection. 0: normal output; 1: open drain. (R/W)

Register 6.46. RTCIO_DIG_PAD_HOLD_REG (0x0074)

| 31 | 0 | |
|---|---|---|
| 0 | | Reset |

**RTCIO_DIG_PAD_HOLD_REG** Selects the digital pins which should be put on hold. While 0 allows normal operation, 1 puts the pin on hold. (R/W)

Table 6.13-1. Mapping of Bits to Pins

| Name | Description |
|---|---|
| Bit[0] | Set to 1 to enable the Hold function of pin U0RXD |
| Bit[1] | Set to 1 to enable the Hold function of pin U0TXD |
| Bit[2] | Set to 1 to enable the Hold function of pin SD_CLK |
| Bit[3] | Set to 1 to enable the Hold function of pin SD_DATA0 |
| Bit[4] | Set to 1 to enable the Hold function of pin SD_DATA1 |
| Bit[5] | Set to 1 to enable the Hold function of pin SD_DATA2 |
| Bit[6] | Set to 1 to enable the Hold function of pin SD_DATA3 |
| Bit[7] | Set to 1 to enable the Hold function of pin SD_CMD |
| Bit[8] | Set to 1 to enable the Hold function of pin GPIO5 |
| Bit[9] | Set to 1 to enable the Hold function of pin GPIO16 |
| Bit[10] | Set to 1 to enable the Hold function of pin GPIO17 |
| Bit[11] | Set to 1 to enable the Hold function of pin GPIO18 |
| Bit[12] | Set to 1 to enable the Hold function of pin GPIO19 |
| Bit[13] | Set to 1 to enable the Hold function of pin GPIO20[1] |
| Bit[14] | Set to 1 to enable the Hold function of pin GPIO21 |
| Bit[15] | Set to 1 to enable the Hold function of pin GPIO22 |
| Bit[16] | Set to 1 to enable the Hold function of pin GPIO23 |

1. GPIO20 is only available for ESP32-PICO-V3 and ESP32-PICO-V3-02. Please refer to *ESP32-PICO Series Datasheet* for more information.

**Register 6.47. RTCIO_SENSOR_PADS_REG (0x007C)**

| 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 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTCIO_SENSOR_SENSE1_HOLD | RTCIO_SENSOR_SENSE2_HOLD | RTCIO_SENSOR_SENSE3_HOLD | RTCIO_SENSOR_SENSE4_HOLD | RTCIO_SENSOR_SENSE1_MUX_SEL | RTCIO_SENSOR_SENSE2_MUX_SEL | RTCIO_SENSOR_SENSE3_MUX_SEL | RTCIO_SENSOR_SENSE4_MUX_SEL | RTCIO_SENSOR_SENSE1_FUN_SEL | RTCIO_SENSOR_SENSE1_SLP_SEL | RTCIO_SENSOR_SENSE1_SLP_IE | RTCIO_SENSOR_SENSE1_FUN_IE | RTCIO_SENSOR_SENSE2_FUN_SEL | RTCIO_SENSOR_SENSE2_SLP_SEL | RTCIO_SENSOR_SENSE2_SLP_IE | RTCIO_SENSOR_SENSE2_FUN_IE | RTCIO_SENSOR_SENSE3_FUN_SEL | RTCIO_SENSOR_SENSE3_SLP_SEL | RTCIO_SENSOR_SENSE3_SLP_IE | RTCIO_SENSOR_SENSE3_FUN_IE | RTCIO_SENSOR_SENSE4_FUN_SEL | RTCIO_SENSOR_SENSE4_SLP_SEL | RTCIO_SENSOR_SENSE4_SLP_IE | RTCIO_SENSOR_SENSE4_FUN_IE | (reserved) | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 0 0 | Reset |

**RTCIO_SENSOR_SENSE*n*_HOLD** Set to 1 to hold the output value on sense*n*; 0 is for normal operation. (R/W)

**RTCIO_SENSOR_SENSE*n*_MUX_SEL** 1: route sense*n* to the RTC block; 0: route sense*n* to the digital IO_MUX. (R/W)

**RTCIO_SENSOR_SENSE*n*_FUN_SEL** Select the RTC IO_MUX function for this pin. 0: select Function 0. (R/W)

**RTCIO_SENSOR_SENSE*n*_SLP_SEL** Selection of sleep mode for the pin: set to 1 to put the pin in sleep mode. (R/W)

**RTCIO_SENSOR_SENSE*n*_SLP_IE** Input enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

**RTCIO_SENSOR_SENSE*n*_FUN_IE** Input enable of the pin. 1: enabled; 0: disabled. (R/W)

**Register 6.48. RTCIO_ADC_PAD_REG (0x0080)**

| RTCIO_ADC_ADC1_HOLD | RTCIO_ADC_ADC2_HOLD | RTCIO_ADC_ADC1_MUX_SEL | RTCIO_ADC_ADC2_MUX_SEL | RTCIO_ADC_ADC1_FUN_SEL | RTCIO_ADC_ADC1_SLP_SEL | RTCIO_ADC_ADC1_SLP_IE | RTCIO_ADC_ADC1_FUN_IE | RTCIO_ADC_ADC2_FUN_SEL | RTCIO_ADC_ADC2_SLP_SEL | RTCIO_ADC_ADC2_SLP_IE | RTCIO_ADC_ADC2_FUN_IE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 26 | 25 | 24 | 23 | 22 21 | 20 | 19 | 18 | 17 0 | |
| 0 | 0 | 0 | 0 | 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 |

**RTCIO_ADC_ADC*n*_HOLD** Set to 1 to hold the output value on the pin; 0 is for normal operation. (R/W)

**RTCIO_ADC_ADC*n*_MUX_SEL** 0: route pin to the digital IO_MUX; (R/W)
1: route pin to the RTC block.

**RTCIO_ADC_ADC*n*_FUN_SEL** Select the RTC function for this pin. 0: select Function 0; 3: select Function 1. (R/W)

**RTCIO_ADC_ADC*n*_SLP_SEL** Signal selection of pin's sleep mode. Set this bit to 1 to put the pin to sleep. (R/W)

**RTCIO_ADC_ADC*n*_SLP_IE** Input enable of the pin in sleep mode. 1 enabled; 0 disabled. (R/W)

**RTCIO_ADC_ADC*n*_FUN_IE** Input enable of the pin. 1 enabled; 0 disabled. (R/W)

**Register 6.49. RTCIO_PAD_DAC1_REG (0x0084)**

| RTCIO_PAD_PDAC1_DRV | RTCIO_PAD_PDAC1_HOLD | RTCIO_PAD_PDAC1_RDE | RTCIO_PAD_PDAC1_RUE | RTCIO_PAD_PDAC1_DAC | RTCIO_PAD_PDAC1_XPD_DAC | RTCIO_PAD_PDAC1_MUX_SEL | RTCIO_PAD_PDAC1_FUN_SEL | RTCIO_PAD_PDAC1_SLP_SEL | RTCIO_PAD_PDAC1_SLP_IE | RTCIO_PAD_PDAC1_SLP_OE | RTCIO_PAD_PDAC1_FUN_IE | RTCIO_PAD_PDAC1_DAC_XPD_FORCE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 30 | 29 | 28 | 27 | 26 19 | 18 | 17 | 16 15 | 14 | 13 | 12 | 11 | 10 | 9 0 | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_PAD_PDAC1_DRV** Select the drive strength of the pin. (R/W)

**RTCIO_PAD_PDAC1_HOLD** Set to 1 to hold the output value on the pin; set to 0 for normal operation. (R/W)

**RTCIO_PAD_PDAC1_RDE** 1: Pull-down on pin enabled; 0: Pull-down disabled. (R/W)

**RTCIO_PAD_PDAC1_RUE** 1: Pull-up on pin enabled; 0: Pull-up disabled. (R/W)

**RTCIO_PAD_PDAC1_DAC** Pin DAC1 output value. (R/W)

**RTCIO_PAD_PDAC1_XPD_DAC** Power on DAC1. Usually, PDAC1 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

**RTCIO_PAD_PDAC1_MUX_SEL** 0: route pin to the digital IO_MUX; (R/W)
1: route to the RTC block.

**RTCIO_PAD_PDAC1_FUN_SEL** the functional selection signal of the pin. (R/W)

**RTCIO_PAD_PDAC1_SLP_SEL** Sleep mode selection signal of the pin. Set this bit to 1 to put the pin to sleep. (R/W)

**RTCIO_PAD_PDAC1_SLP_IE** Input enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

**RTCIO_PAD_PDAC1_SLP_OE** Output enable of the pin. 1: enabled ; 0: disabled. (R/W)

**RTCIO_PAD_PDAC1_FUN_IE** Input enable of the pin. 1: enabled it; 0: disabled. (R/W)

**RTCIO_PAD_PDAC1_DAC_XPD_FORCE** Power on DAC1. Usually, we need to tristate PDAC1 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

**Register 6.50. RTCIO_PAD_DAC2_REG (0x0088)**

| 31 30 | 29 | 28 | 27 | 26 19 | 18 | 17 | 16 15 | 14 | 13 | 12 | 11 | 10 | 9 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTCIO_PAD_PDAC2_DRV | RTCIO_PAD_PDAC2_HOLD | RTCIO_PAD_PDAC2_RDE | RTCIO_PAD_PDAC2_RUE | RTCIO_PAD_PDAC2_DAC | RTCIO_PAD_PDAC2_XPD_DAC | RTCIO_PAD_PDAC2_MUX_SEL | RTCIO_PAD_PDAC2_FUN_SEL | RTCIO_PAD_PDAC2_SLP_SEL | RTCIO_PAD_PDAC2_SLP_IE | RTCIO_PAD_PDAC2_SLP_OE | RTCIO_PAD_PDAC2_FUN_IE | RTCIO_PAD_PDAC2_DAC_XPD_FORCE | (reserved) | |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_PAD_PDAC2_DRV** Select the drive strength of the pin. (R/W)

**RTCIO_PAD_PDAC2_HOLD** Set to 1 to hold the output value on the pin; 0 is for normal operation. (R/W)

**RTCIO_PAD_PDAC2_RDE** 1: Pull-down on pin enabled; 0: Pull-down disabled. (R/W)

**RTCIO_PAD_PDAC2_RUE** 1: Pull-up on pin enabled; 0: Pull-up disabled. (R/W)

**RTCIO_PAD_PDAC2_DAC** Pin DAC2 output value. (R/W)

**RTCIO_PAD_PDAC2_XPD_DAC** Power on DAC2. PDAC2 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

**RTCIO_PAD_PDAC2_MUX_SEL** 0: route pin to the digital IO_MUX; (R/W)
1: route to the RTC block.

**RTCIO_PAD_PDAC2_FUN_SEL** Select the RTC function for this pin. 0: select Function 0. (R/W)

**RTCIO_PAD_PDAC2_SLP_SEL** Sleep mode selection signal of the pin. Set this bit to 1 to put the pin to sleep. (R/W)

**RTCIO_PAD_PDAC2_SLP_IE** Input enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

**RTCIO_PAD_PDAC2_SLP_OE** Output enable of the pin. 1: enabled; 0: disabled. (R/W)

**RTCIO_PAD_PDAC2_FUN_IE** Input enable of the pin. 1: enabled; 0: disabled. (R/W)

**RTCIO_PAD_PDAC2_DAC_XPD_FORCE** Power on DAC2. Usually, we need to tristate PDAC2 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

**Register 6.51. RTCIO_XTAL_32K_PAD_REG (0x008C)**

| RTCIO_XTAL_X32N_DRV | RTCIO_XTAL_X32N_HOLD | RTCIO_XTAL_X32N_RDE | RTCIO_XTAL_X32N_RUE | RTCIO_XTAL_X32P_DRV | RTCIO_XTAL_X32P_HOLD | RTCIO_XTAL_X32P_RDE | RTCIO_XTAL_X32P_RUE | RTCIO_XTAL_DAC_XTAL_32K | RTCIO_XTAL_XPD_XTAL_32K | RTCIO_XTAL_X32N_MUX_SEL | RTCIO_XTAL_X32P_MUX_SEL | RTCIO_XTAL_X32N_FUN_SEL | RTCIO_XTAL_X32N_SLP_SEL | RTCIO_XTAL_X32N_SLP_IE | RTCIO_XTAL_X32N_SLP_OE | RTCIO_XTAL_X32N_FUN_IE | RTCIO_XTAL_X32P_FUN_SEL | RTCIO_XTAL_X32P_SLP_SEL | RTCIO_XTAL_X32P_SLP_IE | RTCIO_XTAL_X32P_SLP_OE | RTCIO_XTAL_X32P_FUN_IE | RTCIO_XTAL_DRES_XTAL_32K | RTCIO_XTAL_DBIAS_XTAL_32K | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | |
| 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 0 | 0 0 | 0 | Reset |

**RTCIO_XTAL_X32N_DRV** Select the drive strength of the pin. (R/W)

**RTCIO_XTAL_X32N_HOLD** Set to 1 to hold the output value on the pin; 0 is for normal operation. (R/W)

**RTCIO_XTAL_X32N_RDE** 1: Pull-down on pin enabled; 0: Pull-down disabled. (R/W)

**RTCIO_XTAL_X32N_RUE** 1: Pull-up on pin enabled; 0: Pull-up disabled. (R/W)

**RTCIO_XTAL_X32P_DRV** Select the drive strength of the pin. (R/W)

**RTCIO_XTAL_X32P_HOLD** Set to 1 to hold the output value on the pin, 0 is for normal operation. (R/W)

**RTCIO_XTAL_X32P_RDE** 1: Pull-down on pin enabled; 0: Pull-down disabled. (R/W)

**RTCIO_XTAL_X32P_RUE** 1: Pull-up on pin enabled; 0: Pull-up disabled. (R/W)

**RTCIO_XTAL_DAC_XTAL_32K** 32K XTAL bias current DAC value. (R/W)

**RTCIO_XTAL_XPD_XTAL_32K** Power up 32 KHz crystal oscillator. (R/W)

**RTCIO_XTAL_X32N_MUX_SEL** 0: route X32N pin to the digital IO_MUX; 1: route to RTC block. (R/W)

**RTCIO_XTAL_X32P_MUX_SEL** 0: route X32P pin to the digital IO_MUX; 1: route to RTC block. (R/W)

**RTCIO_XTAL_X32N_FUN_SEL** Select the RTC function. 0: select function 0. (R/W)

**RTCIO_XTAL_X32N_SLP_SEL** Sleep mode selection. Set this bit to 1 to put the pin to sleep. (R/W)

**RTCIO_XTAL_X32N_SLP_IE** Input enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

**RTCIO_XTAL_X32N_SLP_OE** Output enable of the pin. 1: enabled; 0; disabled. (R/W)

**RTCIO_XTAL_X32N_FUN_IE** Input enable of the pin. 1: enabled; 0: disabled. (R/W)

**RTCIO_XTAL_X32P_FUN_SEL** Select the RTC function. 0: select function 0; 1: select function 1. (R/W)

**RTCIO_XTAL_X32P_SLP_SEL** Sleep mode selection. Set this bit to 1 to put the pin to sleep. (R/W)

**RTCIO_XTAL_X32P_SLP_IE** Input enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

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

**Register 6.51. RTCIO_XTAL_32K_PAD_REG (0x008C)**

**Continued from the previous page...**

**RTCIO_XTAL_X32P_SLP_OE** Output enable of the pin in sleep mode. 1: enabled; 0: disabled. (R/W)

**RTCIO_XTAL_X32P_FUN_IE** Input enable of the pin. 1: enabled; 0: disabled. (R/W)

**RTCIO_XTAL_DRES_XTAL_32K** 32K XTAL resistor bias control. (R/W)

**RTCIO_XTAL_DBIAS_XTAL_32K** 32K XTAL self-bias reference control. (R/W)

**Register 6.52. RTCIO_TOUCH_CFG_REG (0x0090)**

| RTCIO_TOUCH_XPD_BIAS | RTCIO_TOUCH_DREFH | RTCIO_TOUCH_DREFL | RTCIO_TOUCH_DRANGE | RTCIO_TOUCH_DCUR | (reserved) | |
|---|---|---|---|---|---|---|
| 31 | 30 29 | 28 27 | 26 25 | 24 23 | 22 0 | |
| 0 | 1 1 | 0 0 | 1 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 | Reset |

**RTCIO_TOUCH_XPD_BIAS** Touch sensor bias power on bit. 1: power on; 0: disabled. (R/W)

**RTCIO_TOUCH_DREFH** Touch sensor saw wave top voltage. (R/W)

**RTCIO_TOUCH_DREFL** Touch sensor saw wave bottom voltage. (R/W)

**RTCIO_TOUCH_DRANGE** Touch sensor saw wave voltage range. (R/W)

**RTCIO_TOUCH_DCUR** Touch sensor bias current. When BIAS_SLEEP is enabled, this setting is available. (R/W)

**Register 6.53. RTCIO_TOUCH_PADn_REG (n: 0-7) (0x94+4*n)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTCIO_TOUCH_PADn_HOLD | RTCIO_TOUCH_PADn_DRV | | RTCIO_TOUCH_PADn_RDE | RTCIO_TOUCH_PADn_RUE | (reserved) | | | RTCIO_TOUCH_PADn_DAC | | | RTCIO_TOUCH_PADn_START | RTCIO_TOUCH_PADn_TIE_OPT | | RTCIO_TOUCH_PADn_XPD | RTCIO_TOUCH_PADn_MUX_SEL | RTCIO_TOUCH_PADn_FUN_SEL | RTCIO_TOUCH_PADn_SLP_SEL | RTCIO_TOUCH_PADn_SLP_IE | RTCIO_TOUCH_PADn_SLP_OE | RTCIO_TOUCH_PADn_FUN_IE | RTCIO_TOUCH_PADn_TO_GPIO | (reserved) | |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**RTCIO_TOUCH_PADn_HOLD** Write 1 to hold the current value of the output. (R/W)

**RTCIO_TOUCH_PADn_DRV** Selects the drive strength of the pin. A higher value corresponds with a higher strength. For detailed drive strength, please see *ESP32 Datasheet* > Appendix A.1 *Notes on ESP32 Pin Lists* > Note 8. (R/W)

**RTCIO_TOUCH_PADn_RDE** 1: Pull-down on pin enabled; 0: Pull-down disabled. (R/W)

**RTCIO_TOUCH_PADn_RUE** 1: Pull-up on pin enabled; 0: Pull-up disabled. (R/W)

**RTCIO_TOUCH_PADn_DAC** Touch sensor slope control. 3-bit for each touch pin. Default is b'100. (R/W)

**RTCIO_TOUCH_PADn_START** Write 1 to start touch sensor. (R/W)

**RTCIO_TOUCH_PADn_TIE_OPT** Default touch sensor tie option.
0: Tied to 0 V
1: Tied to VDD_RTC voltage
(R/W)

**RTCIO_TOUCH_PADn_XPD** Write 1 to power on the touch sensor. (R/W)

**RTCIO_TOUCH_PADn_MUX_SEL** Selects RTC IO_MUX or IO_MUX to control the IE/OE/RUE/RDE statues of RTC pin.
1: Selects RTC IO_MUX
0: Selects IO_MUX
(R/W)

**RTCIO_TOUCH_PADn_FUN_SEL** Selects the function of the RTC.
0: RTC Function 0
1: Reserved
2: Reserved
3: RTC Function 1
(R/W)

**RTCIO_TOUCH_PADn_SLP_SEL** Sleep mode selection signal of the pin. Set this bit to 1 to put the pin to sleep. (R/W)

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

**Register 6.53. RTCIO_TOUCH_PADn_REG (n: 0-7) (0x94+4*n)**

**Continued from the previous page...**

**RTCIO_TOUCH_PADn_SLP_IE** Input enable of the pin in sleep mode (SLP_SEL = 1).

1: Enabled

0: Disabled

(R/W)

**RTCIO_TOUCH_PADn_SLP_OE** Output enable of the pin in sleep mode (SLP_SEL = 1).

1: Enabled

0: Disabled

(R/W)

**RTCIO_TOUCH_PADn_FUN_IE** Input enable of the pin in normal working mode (SLP_SEL = 0).

1: Enabled

0: Disabled

(R/W)

**RTCIO_TOUCH_PADn_TO_GPIO** Controls the routing of touch pin input signals to IO_MUX.

1: The input signal from the touch pin is routed to IO_MUX through analog function.

0: The input signal from the touch pin is routed to IO_MUX through digital function.

(R/W)

**Register 6.54. RTCIO_TOUCH_PADm_REG (m = 8, 9) (0x00B4, 0x00B8)**

| (reserved) | RTCIO_TOUCH_PADm_DAC | RTCIO_TOUCH_PADm_START | RTCIO_TOUCH_PADm_TIE_OPT | RTCIO_TOUCH_PADm_XPD | RTCIO_TOUCH_PADm_TO_GPIO | (reserved) | |
|---|---|---|---|---|---|---|---|
| 31 26 | 25 23 | 22 | 21 | 20 | 19 | 16 0 | |
| 0 0 0 0 0 0 | 1 0 0 | 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_TOUCH_PADm_DAC** Touch sensor slope control. 3-bit for each touch pin. Default b'100. (R/W)

**RTCIO_TOUCH_PADm_START** Write 1 to start touch sensor. (R/W)

**RTCIO_TOUCH_PADm_TIE_OPT** Default touch sensor tie option.

0: Tied to 0 V

1: Tied to VDD_RTC voltage

(R/W)

**RTCIO_TOUCH_PADm_XPD** Write 1 to power on the touch sensor. (R/W)

**RTCIO_TOUCH_PADm_TO_GPIO** Controls the routing of touch pin input signals to IO_MUX.

1: The input signal from the touch pin is routed to IO_MUX through analog function.

0: The input signal from the touch pin is routed to IO_MUX through digital function.

(R/W)

**Register 6.55. RTCIO_EXT_WAKEUP0_REG (0x00BC)**

| RTCIO_EXT_WAKEUP0_SEL | (reserved) | |
|---|---|---|
| 31 27 | 26 0 | |
| 0 | 0 0 0 0 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 |

**RTCIO_EXT_WAKEUP0_SEL** GPIO[0-17] can be used to wake up the chip when the chip is in the sleep mode. This register prompts the pin source to wake up the chip when the latter is in deep/light sleep mode. 0: select GPIO0; 1: select GPIO2, etc. (R/W)

**Register 6.56. RTCIO_XTL_EXT_CTR_REG (0x00C0)**

| RTCIO_XTL_EXT_CTR_SEL | (reserved) | |
|---|---|---|
| 31 27 | 26 0 | |
| 0 | 0 0 0 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 |

**RTCIO_XTL_EXT_CTR_SEL** Select the external crystal power down enable source to get into sleep mode. 0: select GPIO0; 1: select GPIO2, etc. The input value on this pin XOR RTC_CNTL_XTL_EXT_CTR_LV is the crystal power down enable signal. (R/W)

**Register 6.57. RTCIO_SAR_I2C_IO_REG (0x00C4)**

| RTCIO_SAR_I2C_SDA_SEL | RTCIO_SAR_I2C_SCL_SEL | (reserved) | |
|---|---|---|---|
| 31 30 | 29 28 | 27 0 | |
| 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTCIO_SAR_I2C_SDA_SEL** Selects the other pin as the RTC I2C SDA signal. 0: pin TOUCH_PAD[1]; 1: pin TOUCH_PAD[3]. Default value is 0. (R/W)

**RTCIO_SAR_I2C_SCL_SEL** Selects the other pin as the RTC I2C SCL signal. 0: pin TOUCH_PAD[0]; 1: pin TOUCH_PAD[2]. Default value is 0. (R/W)

# Chapter 7

# Reset and Clock

## 7.1 System Reset

### 7.1.1 Introduction

The ESP32 has three reset levels: CPU reset, Core reset, and System reset. None of these reset levels clear the RAM. Figure 7.1-1 shows the subsystems included in each reset level.

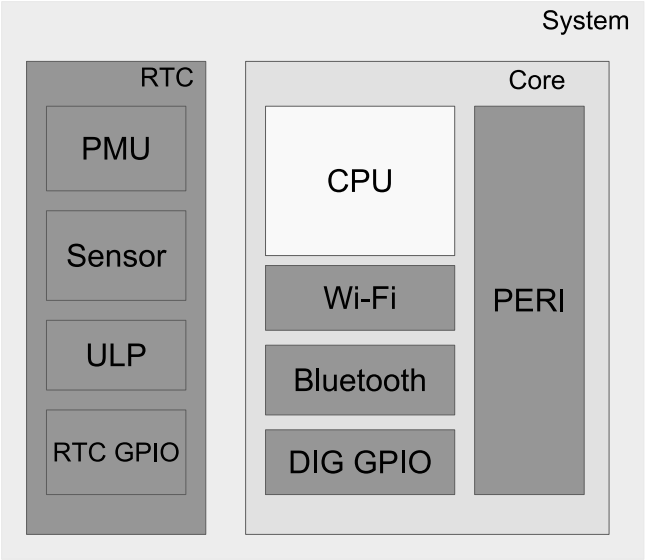



Figure 7.1-1. System Reset

- CPU reset: Only resets the registers of one or both of the CPU cores.
- Core reset: Resets all the digital registers, including CPU cores, external GPIO and digital GPIO. The RTC is not reset.
- System reset: Resets all the registers on the chip, including those of the RTC.

### 7.1.2 Reset Source

While most of the time the APP_CPU and PRO_CPU will be reset simultaneously, some reset sources are able to reset only one of the two cores. The reset reason for each core can be looked up individually: the PRO_CPU reset reason will be stored in RTC_CNTL_RESET_CAUSE_PROCPU, the reset reason for the APP_CPU in RTC_CNTL_RESET_CAUSE_APPCPU. Table 7.1-1 shows the possible reset reason values that can be read from these registers.

Table 7.1-1. PRO_CPU and APP_CPU Reset Reason Values

| PRO | APP | Source | Reset Type | Note |
|---|---|---|---|---|
| 0x01 | 0x01 | Chip Power On Reset | System Reset | - |
| 0x10 | 0x10 | RWDT System Reset | System Reset | See WDT Chapter. |
| 0x0F | 0x0F | Brown Out Reset | System Reset | See Power Management Chapter. |
| 0x03 | 0x03 | Software System Reset | Core Reset | Configure RTC_CNTL_SW_SYS_RST register. |
| 0x05 | 0x05 | Deep Sleep Reset | Core Reset | See Power Management Chapter. |
| 0x06 | 0x06 | SDIO Reset | Core Reset | Reserved |
| 0x07 | 0x07 | MWDT0 Global Reset | Core Reset | See WDT Chapter. |
| 0x08 | 0x08 | MWDT1 Global Reset | Core Reset | See WDT Chapter. |
| 0x09 | 0x09 | RWDT Core Reset | Core Reset | See WDT Chapter. |
| 0x0B | - | MWDT0 CPU Reset | CPU Reset | See WDT Chapter. |
| 0x0C | - | Software CPU Reset | CPU Reset | Configure RTC_CNTL_SW_APPCPU_RST register. |
| - | 0x0B | MWDT1 CPU Reset | CPU Reset | See WDT Chapter. |
| - | 0x0C | Software CPU Reset | CPU Reset | Configure RTC_CNTL_SW_APPCPU_RST register. |
| 0x0D | 0x0D | RWDT CPU Reset | CPU Reset | See WDT Chapter. |
| - | 0xE | PRO CPU Reset | CPU Reset | Indicates that the PRO CPU has independently reset the APP CPU by configuring the DPORT_APPCPU_RESETTING register. |

## 7.2 System Clock

### 7.2.1 Introduction

The ESP32 integrates multiple clock sources for the CPU cores, the peripherals and the RTC. These clocks can be configured to meet different requirements. Figure 7.2-1 shows the system clock structure.

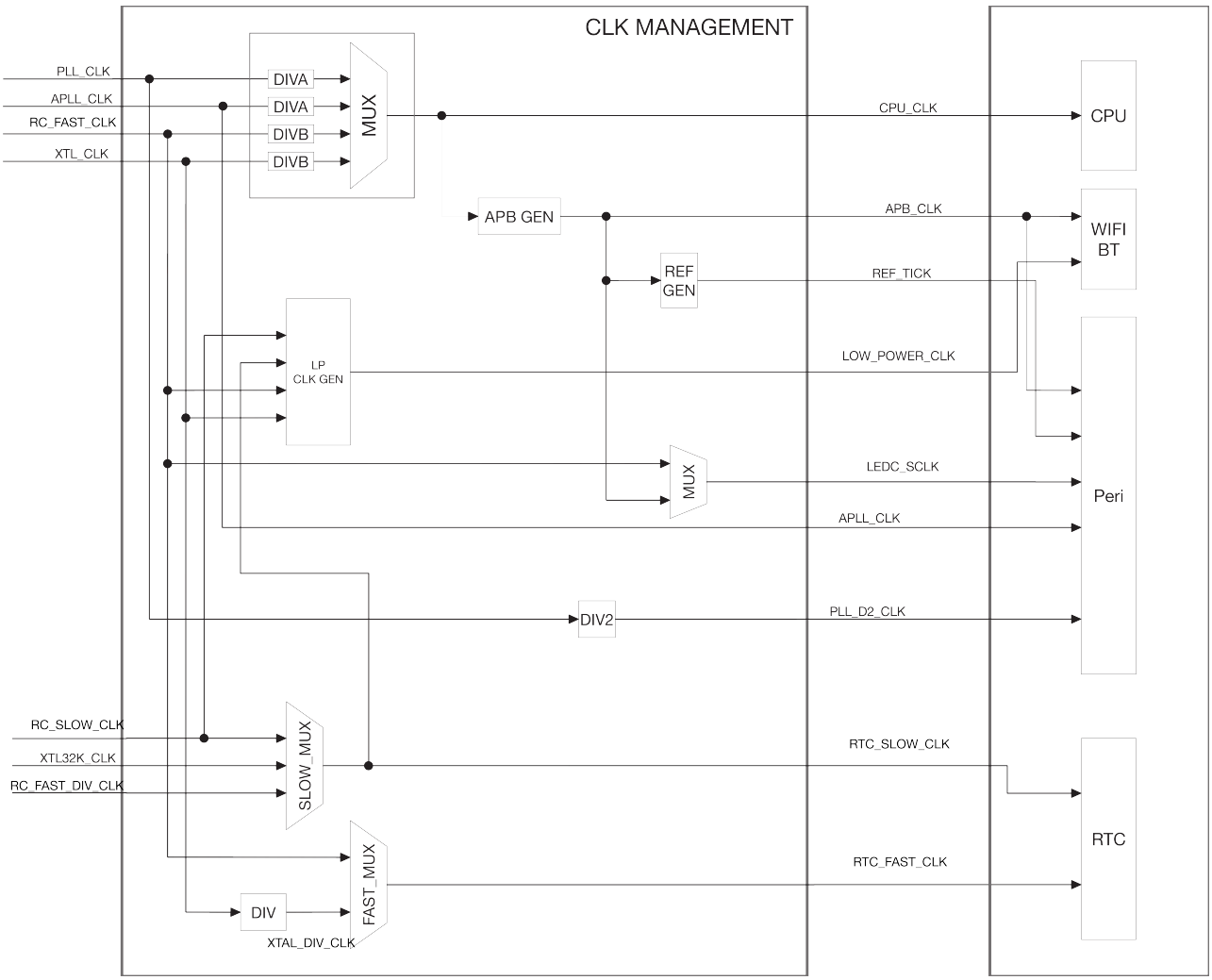



Figure 7.2-1. System Clock

## 7.2.2 Clock Source

The ESP32 can use an external crystal oscillator, an internal PLL or an oscillating circuit as a clock source. Specifically, the clock sources available are:

- High Speed Clocks
  - PLL_CLK is an internal PLL clock with a frequency of 320 MHz or 480 MHz.
  - XTL_CLK is a clock signal generated using an external crystal with a frequency range of 2 ~ 40 MHz.
- Low Power Clocks
  - XTL32K_CLK is a clock generated using an external crystal with a frequency of 32 KHz.
  - RC_FAST_CLK is an internal clock with a default frequency of 8 MHz. This frequency is adjustable.
  - RC_FAST_DIV_CLK is divided from RC_FAST_CLK. Its frequency is (RC_FAST_CLK / 256). With the default RC_FAST_CLK frequency of 8 MHz, this clock runs at 31.250 KHz.
  - RC_SLOW_CLK is an internal low power clock with a default frequency of 150 KHz. This frequency is adjustable.
- Audio Clock

- APLL_CLK is an internal Audio PLL clock with a frequency range of 16 ~ 128 MHz.

### 7.2.3 CPU Clock

As Figure 7.2-1 shows, CPU_CLK is the master clock for both CPU cores. CPU_CLK clock can be as high as 240 MHz when the CPU is in high performance mode. Alternatively, the CPU can run at lower frequencies to reduce power consumption.

The CPU_CLK clock source is determined by the RTC_CNTL_SOC_CLK_SEL register. PLL_CLK, APLL_CLK, RC_FAST_CLK, and XTL_CLK can be set as the CPU_CLK source; see Table 7.2-1 and 7.2-2.

**Table 7.2-1. CPU_CLK Source**

| RTC_CNTL_SOC_CLK_SEL Value | Clock Source |
|---|---|
| 0 | XTL_CLK |
| 1 | PLL_CLK |
| 2 | RC_FAST_CLK |
| 3 | APLL_CLK |

**Table 7.2-2. CPU_CLK Derivation**

| Clock Source | *SEL_0 | *SEL_1 | CPU Clock Frequency |
|---|---|---|---|
| XTL_CLK | 0 | - | CPU_CLK = XTL_CLK / (SYSCON_PRE_DIV_CNT+1) |
| PLL_CLK (320 MHz) | 1 | 0 | CPU_CLK = PLL_CLK / 4<br>CPU_CLK frequency is 80 MHz |
| PLL_CLK (320 MHz) | 1 | 1 | CPU_CLK = PLL_CLK / 2<br>CPU_CLK frequency is 160 MHz |
| PLL_CLK (480 MHz) | 1 | 2 | CPU_CLK = PLL_CLK / 2<br>CPU_CLK frequency is 240 MHz |
| RC_FAST_CLK | 2 | - | CPU_CLK = RC_FAST_CLK / (SYSCON_PRE_DIV_CNT+1) |
| APLL_CLK | 3 | 0 | CPU_CLK = APLL_CLK / 4 |
| APLL_CLK | 3 | 1 | CPU_CLK = APLL_CLK / 2 |

*SEL_0: The value of register RTC_CNTL_SOC_CLK_SEL
*SEL_1: The value of register CPU_CPUPERIOD_SEL

### 7.2.4 Peripheral Clock

Peripheral clocks include APB_CLK, REF_TICK, LEDC_SCLK, APLL_CLK, and PLL_F160M_CLK.
Table 7.2-3 shows which clocks can be used by which peripherals.

**Table 7.2-3. Peripheral Clock Usage**

| Peripherals | APB_CLK | REF_TICK | LEDC_SCLK | APLL_CLK | PLL_F160M_CLK |
|---|---|---|---|---|---|
| EMAC | Y | N | N | Y | N |
| TIMG | Y | N | N | N | N |
| I2S | Y | N | N | Y | Y |
| UART | Y | Y | N | N | N |

| | | | | | |
|---|---|---|---|---|---|
| RMT | Y | Y | N | N | N |
| LED PWM | Y | Y | Y | N | N |
| PWM | Y | N | N | N | Y |
| I2C | Y | N | N | N | N |
| SPI | Y | N | N | N | N |
| PCNT | Y | N | N | N | N |
| eFuse Controller | Y | N | N | N | N |
| SDIO Slave | Y | N | N | N | N |
| SDMMC | Y | N | N | N | N |

#### 7.2.4.1 APB_CLK

The APB_CLK frequency is determined by CPU_CLK source, as detailed in Table 7.2-4.

**Table 7.2-4. APB_CLK**

| CPU_CLK Source | APB_CLK Frequency |
|---|---|
| PLL_CLK | 80 MHz |
| APLL_CLK | CPU_CLK / 2 |
| XTL_CLK | CPU_CLK |
| RC_FAST_CLK | CPU_CLK |

#### 7.2.4.2 REF_TICK

REF_TICK is derived from APB_CLK. The APB_CLK frequency is determined by CPU_CLK source. The REF_TICK frequency should be fixed. When CPU_CLK source changes, users need to make sure the REF_TICK frequency remains unchanged by setting a correct divider value.

Clock divider registers are shown in Table 7.2-5.

**Table 7.2-5. REF_TICK**

| CPU_CLK Source | APB_CLK Frequency | REF_TICK Frequency |
|---|---|---|
| PLL_CLK | 80 MHz | APB_CLK / (SYSCON_PLL_TICK_NUM+1) |
| APLL_CLK | CPU_CLK / 2 | APB_CLK / (SYSCON_APLL_TICK_NUM+1) |
| XTL_CLK | CPU_CLK | APB_CLK / (SYSCON_XTAL_TICK_NUM+1) |
| FOSC_CLK | CPU_CLK | APB_CLK / (SYSCON_CK8M_TICK_NUM+1) |

For example, when CPU_CLK source is PLL_CLK and users need to keep the REF_TICK frequency at 1 MHz, then they should set SYSCON_PLL_TICK_NUM to 79 (0x4F) so that the REF_TICK frequency = 80 MHz / (79+1) = 1 MHz.

#### 7.2.4.3 LEDC_SCLK Source

The LEDC_SCLK clock source is selected by the LEDC_APB_CLK_SEL register, as shown in Table 7.2-6.

Table 7.2-6. LEDC_SCLK Derivation

| LEDC_APB_CLK_SEL Value | LEDC_SCLK Source |
|---|---|
| 0 | RC_FAST_CLK |
| 1 | APB_CLK |

#### 7.2.4.4 APLL_SCLK Source

The APLL_CLK is sourced from PLL_CLK, with its output frequency configured using the APLL configuration registers.

#### 7.2.4.5 PLL_F160M_CLK Source

PLL_F160M_CLK is divided from PLL_CLK by automatically adjusting the clock division and its frequency is always 160 MHz.

#### 7.2.4.6 Clock Source Considerations

Most peripherals will operate using the APB_CLK frequency as a reference. When this frequency changes, the peripherals will need to update their clock configuration to operate at the same frequency after the change. Peripherals accessing REF_TICK can continue operating normally when switching clock sources, without changing clock source. Please see Table 7.2-3 for details.

The LED PWM module can use RC_FAST_CLK as a clock source when APB_CLK is disabled. In other words, when the system is in low-power consumption mode (see Chapter 9 *Low-Power Management (RTC_CNTL)*), normal peripherals will be halted (APB_CLK is turned off), but the LED PWM can work normally via RC_FAST_CLK.

### 7.2.5 Wi-Fi BT Clock

Wi-Fi and BT can only operate if APB_CLK uses PLL_CLK as its clock source. Suspending PLL_CLK requires Wi-Fi and BT to both have entered low-power consumption mode first.

For LOW_POWER_CLK, one of XTL32K_CLK, RTC_SLOW_CLK, RC_FAST_CLK or XTL_CLK can be selected as the low-power consumption mode clock source for Wi-Fi and BT.

### 7.2.6 RTC Clock

The clock sources of RTC_SLOW_CLK and RTC_FAST_CLK are low-frequency clocks. The RTC module can operate when most other clocks are stopped.

RTC_SLOW_CLK is used to clock the Power Management module. It can be sourced from RC_SLOW_CLK, XTL32K_CLK or RC_FAST_DIV_CLK.

RTC_FAST_CLK is used to clock the On-chip Sensor module. It can be sourced from a divided XTL_CLK or from RC_FAST_CLK.

### 7.2.7 Audio PLL

The operation of audio and other time-critical data-transfer applications requires highly-configurable, low-jitter, and accurate clock sources. The clock sources derived from system clocks that serve digital peripherals may

carry jitter and, therefore, they do not support a high-precision clock frequency setting.

Providing an integrated precision clock source can minimize system cost. To this end, ESP32 integrates an audio PLL.

The Audio PLL formula is as follows:

$$f_{\text{out}} = \frac{f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4)}{2(odiv + 2)}$$

The parameters of this formula are defined below:

- $f_{\text{xtal}}$: the frequency of the crystal oscillator, usually 40 MHz;
- sdm0: the value is 0 ~ 255;
- sdm1: the value is 0 ~ 255;
- sdm2: the value is 0 ~ 63;
- odiv: the value is 0 ~ 31;
- The operating frequency range of the numerator is 350 MHz ~ 500 MHz

$$350MHz < f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4) < 500MHz$$

Please note that sdm1 and sdm0 are not available on revision0 of ESP32. Please consult the silicon revision in ESP32 Series SoC Errata for further details.

Audio PLL can be manually enabled or disabled via registers RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA
_FORCE_PD, respectively. Disabling it takes priority over enabling it. When RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD are 0, PLL will follow the state of the system, i.e., when the system enters sleep mode, PLL will be disabled automatically; when the system wakes up, PLL will be enabled automatically.

## 7.3 Register Summary

The addresses in this section are relative to the SYSCON base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*.

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration register** | | | |
| SYSCON_SYSCLK_CONF_REG | Configures system clock frequency | 0x0000 | R/W |
| SYSCON_XTAL_TICK_CONF_REG | Configures the divider value of REF_TICK | 0x0004 | R/W |
| SYSCON_PLL_TICK_CONF_REG | Configures the divider value of REF_TICK | 0x0008 | R/W |
| SYSCON_CK8M_TICK_CONF_REG | Configures the divider value of REF_TICK | 0x000C | R/W |
| SYSCON_APLL_TICK_CONF_REG | Configures the divider value of REF_TICK | 0x003C | R/W |
| **Chip revision register** | | | |
| SYSCON_DATE_REG | Chip revision register | 0x007C | R/W |

## 7.4 Registers

The addresses in this section are relative to the SYSCON base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*.

**Register 7.1. SYSCON_SYSCLK_CONF_REG (0x0000)**

| (reserved) | SYSCON_PRE_DIV_CNT | |
|---|---|---|
| 31 … 10 | 9 … 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 |

**SYSCON_PRE_DIV_CNT** Configures the divider value of CPU_CLK when the source of CPU_CLK is XTL_CLK or RC_FAST_CLK. The value range is 0x0 ~ 0x3FF. CPU_CLK = XTL_CLK ( or RC_FAST_CLK) / (the value of this field +1). (R/W)

**Register 7.2. SYSCON_XTAL_TICK_CONF_REG (0x0004)**

| (reserved) | SYSCON_XTAL_TICK_NUM | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 39 | Reset |

**SYSCON_XTAL_TICK_NUM** Configures the divider value of REF_TICK when the source of APB_CLK is XTL_CLK. The value range is 0x0 ~ 0xFF. REF_TICK = APB_CLK /(the value of this field + 1). (R/W)

**Register 7.3. SYSCON_PLL_TICK_CONF_REG (0x0008)**

| (reserved) | SYSCON_PLL_TICK_NUM | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 79 | Reset |

**SYSCON_PLL_TICK_NUM** Configures the divider value of REF_TICK when the source of APB_CLK is PLL_CLK. The value range is 0x0 ~ 0xFF. REF_TICK = APB_CLK /(the value of this field + 1). (R/W)

Register 7.4. SYSCON_CK8M_TICK_CONF_REG (0x000C)

| (reserved) | SYSCON_CK8M_TICK_NUM | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 11 | Reset |

**SYSCON_CK8M_TICK_NUM** Configures the divider value of REF_TICK when the source of APB_CLK is FOSC_CLK. The value range is 0x0 ~ 0xFF. REF_TICK = APB_CLK /(the value of this field + 1). (R/W)

Register 7.5. SYSCON_APLL_TICK_CONF_REG (0x003C)

| (reserved) | SYSCON_APLL_TICK_NUM | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 99 | Reset |

**SYSCON_APLL_TICK_NUM** Configures the divider value of REF_TICK when the source of APB_CLK is APLL_CLK. The value range is 0x0 ~ 0xFF. REF_TICK = APB_CLK /(the value of this field + 1). (R/W)

Register 7.6. SYSCON_DATE_REG (0x007C)

| SYSCON_DATE | |
|---|---|
| 31 … 0 | |
| 0x16042000 | Reset |

**SYSCON_DATE** Chip revision register. For more information see *ESP32 Series SoC Errata*. (R/W)

# Chapter 8

# Interrupt Matrix (INTERRUPT)

## 8.1 Overview

The Interrupt Matrix embedded in the ESP32 independently allocates peripheral interrupt sources to the two CPUs' peripheral interrupts. This configuration is made to be highly flexible in order to meet many different needs.

## 8.2 Features

- Accepts 71 peripheral interrupt sources as input.
- Generates 26 peripheral interrupt sources per CPU as output (52 total).
- CPU NMI Interrupt Mask.
- Queries current interrupt status of peripheral interrupt sources.

The structure of the Interrupt Matrix is shown in Figure 8.2-1.

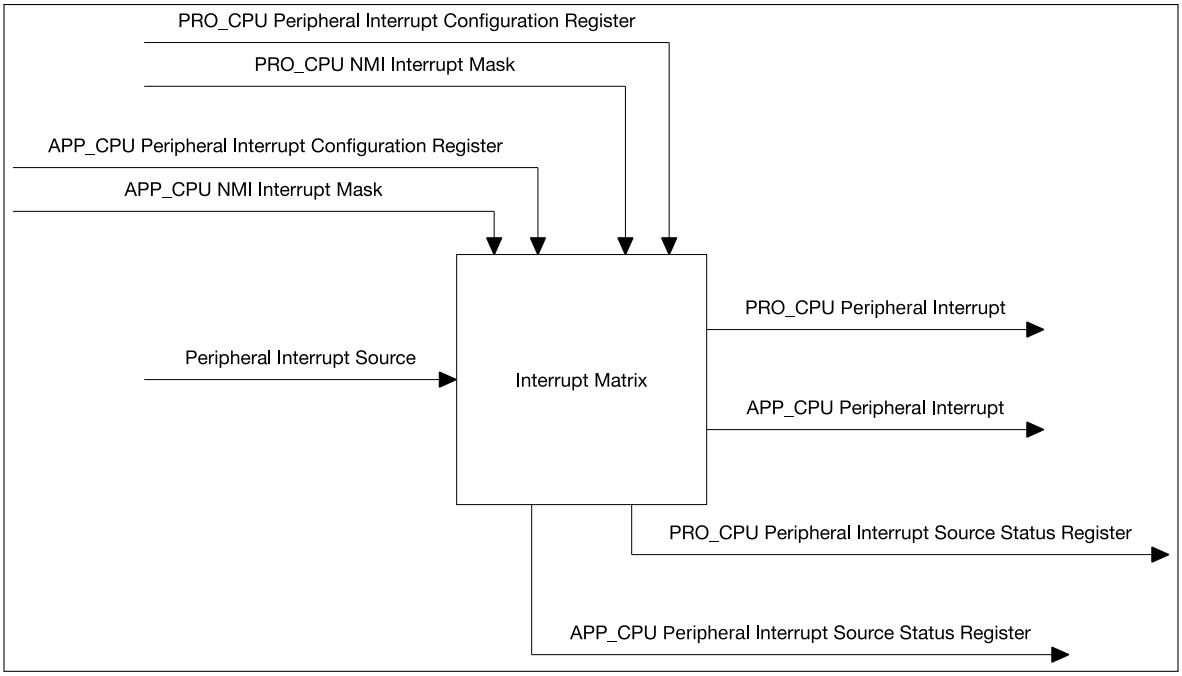



Figure 8.2-1. Interrupt Matrix Structure

## 8.3 Functional Description

### 8.3.1 Peripheral Interrupt Source

ESP32 has 71 peripheral interrupt sources in total. All peripheral interrupt sources are listed in table 8.3-1. 67 of 71 ESP32 peripheral interrupt sources can be allocated to either CPU.

The four remaining peripheral interrupt sources are CPU-specific, two per CPU. GPIO_INTERRUPT_PRO and GPIO_INTERRUPT_PRO_NMI can only be allocated to PRO_CPU. GPIO_INTERRUPT_APP and GPIO_INTERRUPT_APP_NMI can only be allocated to APP_CPU. As a result, PRO_CPU and APP_CPU each have 69 peripheral interrupt sources.

PeripheralInterruptConfigStatusSource.tex

## Table 8.3-1. PRO_CPU, APP_CPU Interrupt Configuration

| PRO_CPU | | | | | | APP_CPU | | |
|---|---|---|---|---|---|---|---|---|
| Peripheral Interrupt Configuration Register | Status Register | | Peripheral Interrupt Source | | | Status Register | | Peripheral Interrupt Configuration Register |
| | Bit | Name | No. | Name | No. | Name | Bit | |
| DPORT_PRO_MAC_INTR_MAP_REG | 0 | DPORT_PRO_INTR_STATUS_REG_0_REG | 0 | MAC_INTR | 0 | DPORT_APP_INTR_STATUS_REG_0_REG | 0 | DPORT_APP_MAC_INTR_MAP_REG |
| DPORT_PRO_MAC_NMI_MAP_REG | 1 | | 1 | MAC_NMI | 1 | | 1 | DPORT_APP_MAC_NMI_MAP_REG |
| DPORT_PRO_BB_INT_MAP_REG | 2 | | 2 | BB_INT | 2 | | 2 | DPORT_APP_BB_INT_MAP_REG |
| DPORT_PRO_BT_MAC_INT_MAP_REG | 3 | | 3 | BT_MAC_INT | 3 | | 3 | DPORT_APP_BT_MAC_INT_MAP_REG |
| DPORT_PRO_BT_BB_INT_MAP_REG | 4 | | 4 | BT_BB_INT | 4 | | 4 | DPORT_APP_BT_BB_INT_MAP_REG |
| DPORT_PRO_BT_BB_NMI_MAP_REG | 5 | | 5 | BT_BB_NMI | 5 | | 5 | DPORT_APP_BT_BB_NMI_MAP_REG |
| DPORT_PRO_RWBT_IRQ_MAP_REG | 6 | | 6 | RWBT_IRQ | 6 | | 6 | DPORT_APP_RWBT_IRQ_MAP_REG |
| DPORT_PRO_RWBLE_IRQ_MAP_REG | 7 | | 7 | RWBLE_IRQ | 7 | | 7 | DPORT_APP_RWBLE_IRQ_MAP_REG |
| DPORT_PRO_RWBT_NMI_MAP_REG | 8 | | 8 | RWBT_NMI | 8 | | 8 | DPORT_APP_RWBT_NMI_MAP_REG |
| DPORT_PRO_RWBLE_NMI_MAP_REG | 9 | | 9 | RWBLE_NMI | 9 | | 9 | DPORT_APP_RWBLE_NMI_MAP_REG |
| DPORT_PRO_SLC0_INTR_MAP_REG | 10 | | 10 | SLC0_INTR | 10 | | 10 | DPORT_APP_SLC0_INTR_MAP_REG |
| DPORT_PRO_SLC1_INTR_MAP_REG | 11 | | 11 | SLC1_INTR | 11 | | 11 | DPORT_APP_SLC1_INTR_MAP_REG |
| DPORT_PRO_UHCI0_INTR_MAP_REG | 12 | | 12 | UHCI0_INTR | 12 | | 12 | DPORT_APP_UHCI0_INTR_MAP_REG |
| DPORT_PRO_UHCI1_INTR_MAP_REG | 13 | | 13 | UHCI1_INTR | 13 | | 13 | DPORT_APP_UHCI1_INTR_MAP_REG |
| DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG | 14 | | 14 | TG_T0_LEVEL_INT | 14 | | 14 | DPORT_APP_TG_T0_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG | 15 | | 15 | TG_T1_LEVEL_INT | 15 | | 15 | DPORT_APP_TG_T1_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG | 16 | | 16 | TG_WDT_LEVEL_INT | 16 | | 16 | DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG | 17 | | 17 | TG_LACT_LEVEL_INT | 17 | | 17 | DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG | 18 | | 18 | TG1_T0_LEVEL_INT | 18 | | 18 | DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG | 19 | | 19 | TG1_T1_LEVEL_INT | 19 | | 19 | DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG | 20 | | 20 | TG1_WDT_LEVEL_INT | 20 | | 20 | DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG |
| DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG | 21 | | 21 | TG1_LACT_LEVEL_INT | 21 | | 21 | DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG |
| DPORT_PRO_GPIO_INTERRUPT_MAP_REG | 22 | | 22 | **GPIO_INTERRUPT_PRO** / **GPIO_INTERRUPT_APP** | 22 | | 22 | DPORT_APP_GPIO_INTERRUPT_MAP_REG |
| DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG | 23 | | 23 | **GPIO_INTERRUPT_PRO_NMI** / **GPIO_INTERRUPT_APP_NMI** | 23 | | 23 | DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG |
| DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG | 24 | | 24 | CPU_INTR_FROM_CPU_0 | 24 | | 24 | DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG |
| DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG | 25 | | 25 | CPU_INTR_FROM_CPU_1 | 25 | | 25 | DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG |
| DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG | 26 | | 26 | CPU_INTR_FROM_CPU_2 | 26 | | 26 | DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG |
| DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG | 27 | | 27 | CPU_INTR_FROM_CPU_3 | 27 | | 27 | DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG |
| DPORT_PRO_SPI_INTR_0_MAP_REG | 28 | | 28 | SPI_INTR_0 | 28 | | 28 | DPORT_APP_SPI_INTR_0_MAP_REG |
| DPORT_PRO_SPI_INTR_1_MAP_REG | 29 | | 29 | SPI_INTR_1 | 29 | | 29 | DPORT_APP_SPI_INTR_1_MAP_REG |
| DPORT_PRO_SPI_INTR_2_MAP_REG | 30 | | 30 | SPI_INTR_2 | 30 | | 30 | DPORT_APP_SPI_INTR_2_MAP_REG |
| DPORT_PRO_SPI_INTR_3_MAP_REG | 31 | | 31 | SPI_INTR_3 | 31 | | 31 | DPORT_APP_SPI_INTR_3_MAP_REG |
| DPORT_PRO_I2S0_INT_MAP_REG | 0 | DPORT_PRO_INTR_STATUS_REG_1_REG | 32 | I2S0_INT | 32 | DPORT_APP_INTR_STATUS_REG_1_REG | 0 | DPORT_APP_I2S0_INT_MAP_REG |
| DPORT_PRO_I2S1_INT_MAP_REG | 1 | | 33 | I2S1_INT | 33 | | 1 | DPORT_APP_I2S1_INT_MAP_REG |
| DPORT_PRO_UART_INTR_MAP_REG | 2 | | 34 | UART_INTR | 34 | | 2 | DPORT_APP_UART_INTR_MAP_REG |
| DPORT_PRO_UART1_INTR_MAP_REG | 3 | | 35 | UART1_INTR | 35 | | 3 | DPORT_APP_UART1_INTR_MAP_REG |
| DPORT_PRO_UART2_INTR_MAP_REG | 4 | | 36 | UART2_INTR | 36 | | 4 | DPORT_APP_UART2_INTR_MAP_REG |
| DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG | 5 | | 37 | SDIO_HOST_INTERRUPT | 37 | | 5 | DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG |
| DPORT_PRO_EMAC_INT_MAP_REG | 6 | | 38 | EMAC_INT | 38 | | 6 | DPORT_APP_EMAC_INT_MAP_REG |
| DPORT_PRO_PWM0_INTR_MAP_REG | 7 | | 39 | PWM0_INTR | 39 | | 7 | DPORT_APP_PWM0_INTR_MAP_REG |
| DPORT_PRO_PWM1_INTR_MAP_REG | 8 | | 40 | PWM1_INTR | 40 | | 8 | DPORT_APP_PWM1_INTR_MAP_REG |
| Reserved | 9 | | 41 | Reserved | 41 | | 9 | Reserved |
| Reserved | 10 | | 42 | Reserved | 42 | | 10 | Reserved |
| DPORT_PRO_LEDC_INT_MAP_REG | 11 | | 43 | LEDC_INT | 43 | | 11 | DPORT_APP_LEDC_INT_MAP_REG |
| DPORT_PRO_EFUSE_INT_MAP_REG | 12 | | 44 | EFUSE_INT | 44 | | 12 | DPORT_APP_EFUSE_INT_MAP_REG |
| DPORT_PRO_TWAI_INT_MAP_REG | 13 | | 45 | TWAI_INT | 45 | | 13 | DPORT_APP_TWAI_INT_MAP_REG |
| DPORT_PRO_RTC_CORE_INTR_MAP_REG | 14 | | 46 | RTC_CORE_INTR | 46 | | 14 | DPORT_APP_RTC_CORE_INTR_MAP_REG |
| DPORT_PRO_RMT_INTR_MAP_REG | 15 | | 47 | RMT_INTR | 47 | | 15 | DPORT_APP_RMT_INTR_MAP_REG |
| DPORT_PRO_PCNT_INTR_MAP_REG | 16 | | 48 | PCNT_INTR | 48 | | 16 | DPORT_APP_PCNT_INTR_MAP_REG |
| DPORT_PRO_I2C_EXT0_INTR_MAP_REG | 17 | | 49 | I2C_EXT0_INTR | 49 | | 17 | DPORT_APP_I2C_EXT0_INTR_MAP_REG |
| DPORT_PRO_I2C_EXT1_INTR_MAP_REG | 18 | | 50 | I2C_EXT1_INTR | 50 | | 18 | DPORT_APP_I2C_EXT1_INTR_MAP_REG |
| DPORT_PRO_RSA_INTR_MAP_REG | 19 | | 51 | RSA_INTR | 51 | | 19 | DPORT_APP_RSA_INTR_MAP_REG |
| DPORT_PRO_SPI1_DMA_INT_MAP_REG | 20 | | 52 | SPI1_DMA_INT | 52 | | 20 | DPORT_APP_SPI1_DMA_INT_MAP_REG |

| PRO_CPU | | | | | APP_CPU | | | |
|---|---|---|---|---|---|---|---|---|
| Peripheral Interrupt Configuration Register | Peripheral Interrupt Source | | | | | | | Peripheral Interrupt Configuration Register |
| | Status Register | | No. | Name | No. | Status Register | | |
| | Bit | Name | | | | Name | Bit | |
| DPORT_PRO_SPI2_DMA_INT_MAP_REG | 21 | DPORT_PRO_INTR_STATUS_REG_1_REG | 53 | SPI2_DMA_INT | 53 | DPORT_APP_INTR_STATUS_REG_1_REG | 21 | DPORT_APP_SPI2_DMA_INT_MAP_REG |
| DPORT_PRO_SPI3_DMA_INT_MAP_REG | 22 | | 54 | SPI3_DMA_INT | 54 | | 22 | DPORT_APP_SPI3_DMA_INT_MAP_REG |
| DPORT_PRO_WDG_INT_MAP_REG | 23 | | 55 | WDG_INT | 55 | | 23 | DPORT_APP_WDG_INT_MAP_REG |
| DPORT_PRO_TIMER_INT1_MAP_REG | 24 | | 56 | TIMER_INT1 | 56 | | 24 | DPORT_APP_TIMER_INT1_MAP_REG |
| DPORT_PRO_TIMER_INT2_MAP_REG | 25 | | 57 | TIMER_INT2 | 57 | | 25 | DPORT_APP_TIMER_INT2_MAP_REG |
| DPORT_PRO_TG_T0_EDGE_INT_MAP_REG | 26 | | 58 | TG_T0_EDGE_INT | 58 | | 26 | DPORT_APP_TG_T0_EDGE_INT_MAP_REG |
| DPORT_PRO_TG_T1_EDGE_INT_MAP_REG | 27 | | 59 | TG_T1_EDGE_INT | 59 | | 27 | DPORT_APP_TG_T1_EDGE_INT_MAP_REG |
| DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG | 28 | | 60 | TG_WDT_EDGE_INT | 60 | | 28 | DPORT_APP_TG_WDT_EDGE_INT_MAP_REG |
| DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG | 29 | | 61 | TG_LACT_EDGE_INT | 61 | | 29 | DPORT_APP_TG_LACT_EDGE_INT_MAP_REG |
| DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG | 30 | | 62 | TG1_T0_EDGE_INT | 62 | | 30 | DPORT_APP_TG1_T0_EDGE_INT_MAP_REG |
| DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG | 31 | | 63 | TG1_T1_EDGE_INT | 63 | | 31 | DPORT_APP_TG1_T1_EDGE_INT_MAP_REG |
| DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG | 0 | DPORT_PRO_INTR_STATUS_REG_2_REG | 64 | TG1_WDT_EDGE_INT | 64 | DPORT_APP_INTR_STATUS_REG_2_REG | 0 | DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG |
| DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG | 1 | | 65 | TG1_LACT_EDGE_INT | 65 | | 1 | DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG |
| DPORT_PRO_MMU_IA_INT_MAP_REG | 2 | | 66 | MMU_IA_INT | 66 | | 2 | DPORT_APP_MMU_IA_INT_MAP_REG |
| DPORT_PRO_MPU_IA_INT_MAP_REG | 3 | | 67 | MPU_IA_INT | 67 | | 3 | DPORT_APP_MPU_IA_INT_MAP_REG |
| DPORT_PRO_CACHE_IA_INT_MAP_REG | 4 | | 68 | CACHE_IA_INT | 68 | | 4 | DPORT_APP_CACHE_IA_INT_MAP_REG |

### 8.3.2 CPU Interrupt

Both of the two CPUs (PRO and APP) have 32 interrupts each, of which 26 are peripheral interrupts. All interrupts in a CPU are listed in Table 8.3-2.

Table 8.3-2. CPU Interrupts

| No. | Category | Type | Priority Level |
|---|---|---|---|
| 0 | Peripheral | Level-Triggered | 1 |
| 1 | Peripheral | Level-Triggered | 1 |
| 2 | Peripheral | Level-Triggered | 1 |
| 3 | Peripheral | Level-Triggered | 1 |
| 4 | Peripheral | Level-Triggered | 1 |
| 5 | Peripheral | Level-Triggered | 1 |
| 6 | Internal | Timer.0 | 1 |
| 7 | Internal | Software | 1 |
| 8 | Peripheral | Level-Triggered | 1 |
| 9 | Peripheral | Level-Triggered | 1 |
| 10 | Peripheral | Edge-Triggered | 1 |
| 11 | Internal | Profiling | 3 |
| 12 | Peripheral | Level-Triggered | 1 |
| 13 | Peripheral | Level-Triggered | 1 |
| 14 | Peripheral | NMI | NMI |
| 15 | Internal | Timer.1 | 3 |
| 16 | Internal | Timer.2 | 5 |
| 17 | Peripheral | Level-Triggered | 1 |
| 18 | Peripheral | Level-Triggered | 1 |
| 19 | Peripheral | Level-Triggered | 2 |
| 20 | Peripheral | Level-Triggered | 2 |
| 21 | Peripheral | Level-Triggered | 2 |
| 22 | Peripheral | Edge-Triggered | 3 |
| 23 | Peripheral | Level-Triggered | 3 |
| 24 | Peripheral | Level-Triggered | 4 |
| 25 | Peripheral | Level-Triggered | 4 |
| 26 | Peripheral | Level-Triggered | 5 |
| 27 | Peripheral | Level-Triggered | 3 |
| 28 | Peripheral | Edge-Triggered | 4 |
| 29 | Internal | Software | 3 |
| 30 | Peripheral | Edge-Triggered | 4 |
| 31 | Peripheral | Level-Triggered | 5 |

### 8.3.3 Allocate Peripheral Interrupt Sources to Peripheral Interrupt on CPU

In this section:

- Source_X stands for any particular peripheral interrupt source.

- PRO_X_MAP_REG (or APP_X_MAP_REG) stands for any particular peripheral interrupt configuration register of the PRO_CPU (or APP_CPU). The peripheral interrupt configuration register corresponds to the peripheral interrupt source Source_X. In Table 8.3-1 the registers listed under "PRO_CPU (APP_CPU) - Peripheral Interrupt Configuration Register" correspond to the peripheral interrupt sources listed in "Peripheral Interrupt Source - Name".
- Interrupt_P stands for CPU peripheral interrupt, numbered as Num_P. Num_P can take the ranges 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28, 30 ~ 31.
- Interrupt_I stands for the CPU internal interrupt numbered as Num_I. Num_I can take values 6, 7, 11, 15, 16, 29.

Using this terminology, the possible operations of the Interrupt Matrix controller can be described as follows:

- **Allocate peripheral interrupt source Source_X to CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG (or APP_X_MAP_REG) to Num_P. Num_P can be any CPU peripheral interrupt number. CPU interrupts can be shared between multiple peripherals (see below).
- **Disable peripheral interrupt source Source_X for CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG (or APP_X _MAP_REG) for peripheral interrupt source to any Num_I. The specific choice of internal interrupt number does not change behaviour, as none of the interrupt numbered as Num_I is connected to either CPU.
- **Allocate multiple peripheral sources Source_X*n* ORed to PRO_CPU (APP_CPU) peripheral interrupt**
  Set multiple PRO_X*n*_MAP_REG (APP_X*n*_MAP_REG) to the same Num_P. Any of these peripheral interrupts will trigger CPU Interrupt_P.

### 8.3.4 CPU NMI Interrupt Mask

The Interrupt Matrix temporarily masks all peripheral interrupt sources allocated to PRO_CPU's ( or APP_CPU's ) NMI interrupt, if it receives the signal PRO_CPU NMI Interrupt Mask ( or APP_CPU NMI Interrupt Mask ) from the peripheral PID Controller, respectively.

### 8.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

The current interrupt status of a peripheral interrupt source can be read via the bit value in PRO_INTR_STATUS_REG_*n* (APP_INTR_STATUS_REG_*n*), as shown in the mapping in Table 8.3-1.

## 8.4 Registers

The interrupt matrix registers are part of the DPORT registers and are described in Section 12.4 in Chapter 12 *DPort Registers*.

# Chapter 9

# Low-Power Management (RTC_CNTL)

## 9.1 Introduction

ESP32 offers efficient and flexible power-management technology to achieve the best balance between power consumption, wakeup latency and available wakeup sources. Users can select out of five predefined power modes of the main processors to suit specific needs of the application. In addition, to save power in power-sensitive applications, control may be executed by the Ultra-Low-Power coprocessor (ULP coprocessor), while the main processors are in Deep-sleep mode.

## 9.2 Features

- Five predefined power modes to support various applications
- Up to 16 KB of retention memory
- 8 x 32 bits of retention registers
- ULP coprocessor enabled in all low-power modes
- RTC boot supported to shorten the wakeup latency

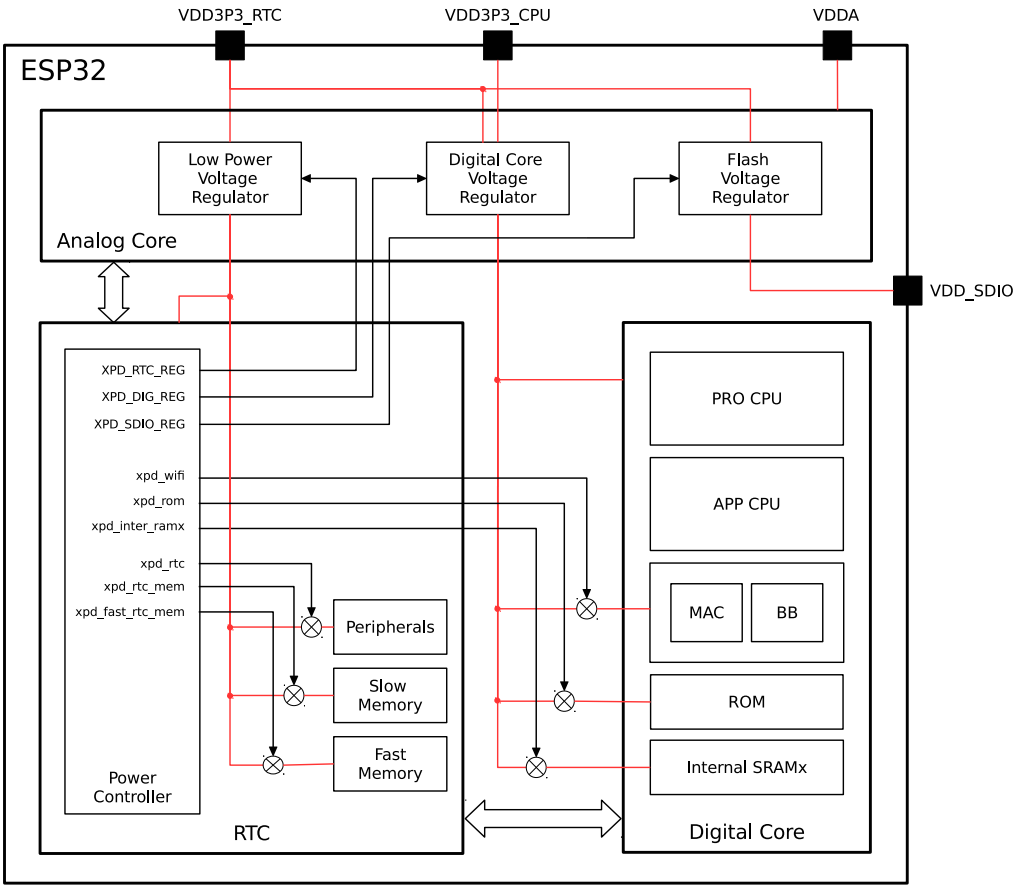



Figure 9.2-1. ESP32 Power Control

# 9.3 Functional Description

## 9.3.1 Overview

The low-power management unit includes voltage regulators, a power controller, power switch cells, power domain isolation cells, etc. Figure 9.2-1 shows the high-level architecture of ESP32's low-power management.

## 9.3.2 Digital Core Voltage Regulator

The built-in voltage regulator can convert the external power supply (typically 3.3 V) to 1.1 V to support the internal digital core. It receives a wide range of external power supplies from 1.8 V to 3.6 V, and provides an output voltage from 0.90 V to 1.25 V.

1. The digital core voltage regulator can be controlled by software via RTC_CNTL_VREG_FORCE_PU and RTC_CNTL_VREG_FORCE_PD.
2. RTC_CNTL_DIG_VREG_DBIAS_WAK and RTC_CNTL_DIG_VREG_DBIAS_SLP tune the supply voltage of the digital core:

$$\text{VDD_DIG} = 0.90 + \text{DBIAS} \cdot 0.05\text{V}$$

3. The current to the digital core comes from pin VDD3P3_CPU and pin VDD3P3_RTC.

Figure 9.3-1 shows the structure of a digital core's voltage regulator.

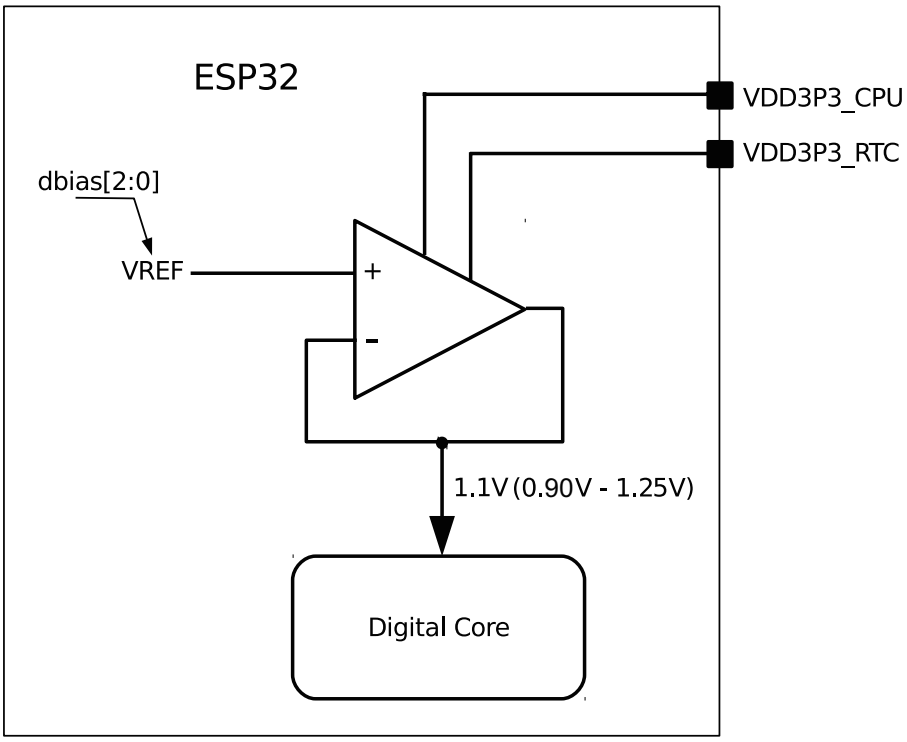



**Figure 9.3-1. Digital Core Voltage Regulator**

### 9.3.3 Low-Power Voltage Regulator

The built-in low-power voltage regulator can convert the external power supply (typically 3.3 V) to 1.1 V to support the internal RTC core. To save power, it receives a wide range of external power supply from 1.8 V to 3.6 V, and supports an adjustable output voltage of 0.90 V to 1.25 V in normal work mode, a fixed output voltage of about 0.75 V both in Deep-sleep mode and Hibernation mode.

1. When the pin CHIP_PU is at a high level, the low-power voltage regulator cannot be turned off. It should be switched only between normal-work mode and Deep-sleep mode.
2. In normal work mode, RTC_CNTL_DBIAS_WAK can be used to tune the output voltage:

$$\text{VDD_RTC} = 0.90 + \text{DBIAS} \cdot 0.05\text{V}$$

3. In Deep-sleep mode, the regulator's output voltage is fixed at about 0.75 V.
4. The current to the RTC core comes from pin VDD3P3_RTC.

Figure 9.3-2 shows the structure of a low-power voltage regulator.

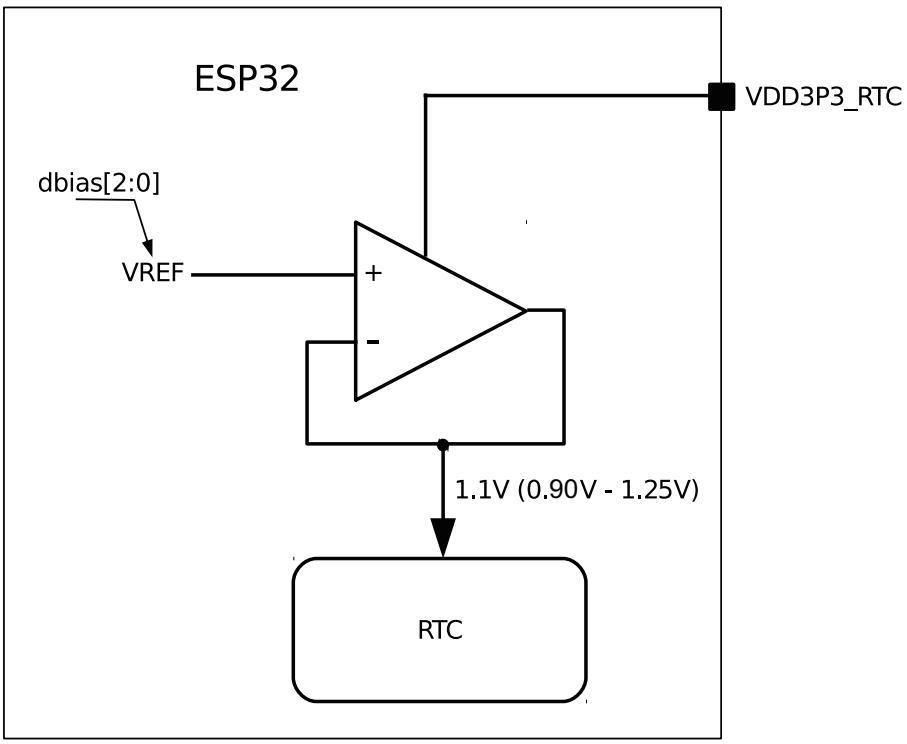



Figure 9.3-2. Low-Power Voltage Regulator

### 9.3.4 Flash Voltage Regulator

The built-in flash voltage regulator can supply a voltage of 3.3 V or 1.8 V to other devices (flash, for example) in the system, with a maximum output current of 40 mA.

1. When XPD_SDIO_VREG == 1, the regulator outputs a voltage of 3.3 V or 1.8 V; when XPD_SDIO_VREG == 0, the output is high-impedance and, in this case, the voltage is provided by the external power supply.
2. When SDIO_TIEH == 1, the regulator shorts pin VDD_SDIO to pin VDD3P3_RTC. The regulator then outputs a voltage of 3.3 V which is the voltage of pin VDD3P3_RTC. When SDIO_TIEH == 0, the inner loop ties the regulator output to the voltage of VREF, which is typically 1.8 V.
3. DREFH_SDIO, DREFM_SDIO and DREFL_SDIO could be used to tune the reference voltage VREF slightly. However, it is recommended that users do not change the value of these registers, since it may affect the stability of the inner loop.
4. When the regulator output is 3.3 V or 1.8 V, the output current comes from the pin VDD3P3_RTC.

Figure 9.3-3 shows the structure of a flash voltage regulator.

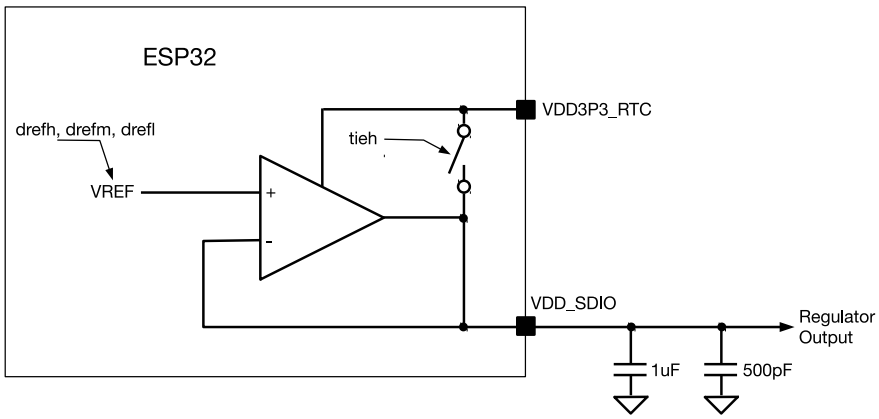



Figure 9.3-3. Flash Voltage Regulator

## 9.3.5 Brownout Detector

The brownout detector checks the voltage of pin VDD3P3_RTC. If the voltage drops rapidly and becomes too low, the detector would trigger a signal to shut down some power-consuming blocks (such as LNA, PA, etc.) to allow extra time for the digital block to save and transfer important data. The power consumption of the detector is ultra low. It remains enabled whenever the chip is powered on, with an adjustable trigger level calibrated around 2.5V.

1. As the output of the brownout detector, RTC_CNTL_BROWN_OUT_DET goes high when the voltage of pin VDD3P3_RTC is lower than the threshold value.
2. RTC_CNTL_DBROWN_OUT_THRES[2:0] is used to tune the threshold voltage, which is usually calibrated around 2.5V.

Figure 9.3-4 shows the structure of a brownout detector.

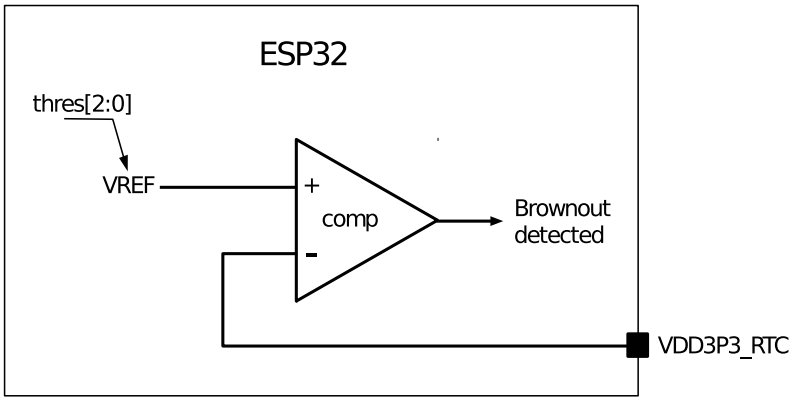



Figure 9.3-4. Brownout Detector

## 9.3.6 RTC Module

The RTC module is designed to handle the entry into, and exit from, the low-power mode, and control the clock sources, PLL, power switch and isolation cells to generate power-gating, clock-gating, and reset signals. As for the low-power management, RTC is composed of the following modules (see Figure 9.3-5):

- RTC main state machine: records the power state.
- Digital & analog power controller: generates actual power-gating/clock-gating signals for digital parts and analog parts.
- Sleep & wakeup controller: handles the entry into & exit from the low-power mode.
- Timers: include RTC main timer, ULP coprocessor timer and touch timer.
- Low-Power processor and sensor controllers: include ULP coprocessor, touch controller, SAR ADC controller, etc.
- Retention memory:
  - RTC slow memory: an 8 KB SRAM, mostly used as retention memory or instruction & data memory for the ULP coprocessor. The CPU accesses it through the APB, starting from address 0x50000000.
  - RTC fast memory: an 8 KB SRAM, mostly used as retention memory. The CPU accesses it through IRAM0/DRAM0. Fast RTC memory is about 10 times faster than the RTC slow memory.
- Retention registers: always-on registers of 8 x 32 bits, serving as data storage.
- RTC IO pads: 18 always-on analog pads, usually functioning as wake-up sources.

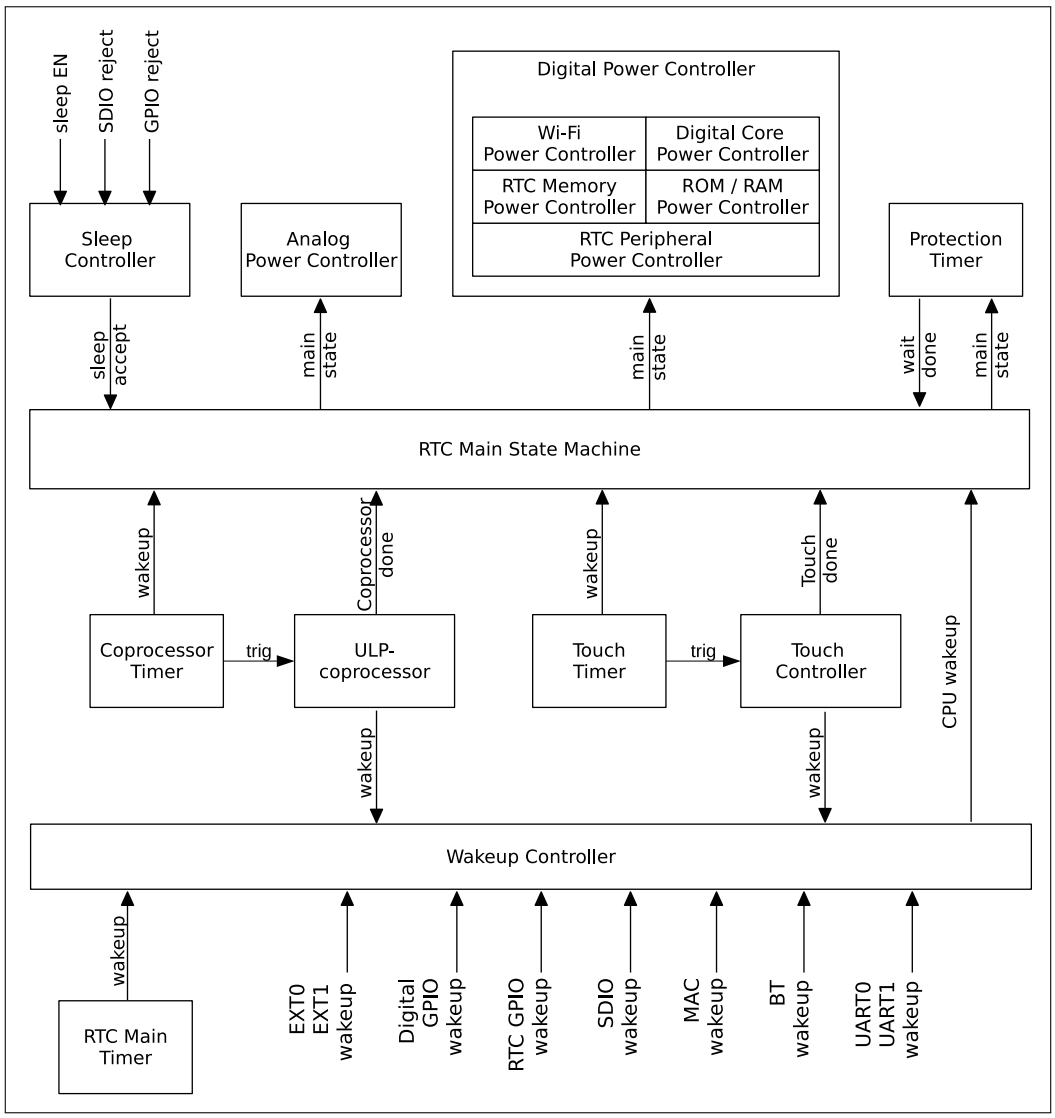



Figure 9.3-5. RTC Structure

## 9.3.7 Low-Power Clocks

In the low-power mode, the 40 MHz crystal and PLL are usually powered down to save power. But clocks are needed for the chip to remain active in the low-power mode.

For the RTC core, there are five possible clock sources:

- external low-speed (32.768 kHz) crystal clock XTL32K_CLK,
- external high-speed (2 MHz ~ 40 MHz) crystal clock XTAL_DIV_CLK,
- internal RC oscillator RC_SLOW_CLK (typically about 150 kHz and adjustable),
- internal 8-MHz oscillator RC_FAST_CLK, and
- internal 31.25-kHz clock RC_FAST_DIV_CLK (derived from the internal 8-MHz oscillator divided by 256).

With these clocks, RTC_FAST_CLK and RTC_SLOW_CLK are derived. Configure RTC_CNTL_RTC_FAST_CLK_SEL to select the clock source of RTC_FAST_CLK and RTC_CNTL_ANA_CLK_RTC_SEL to select the clock source of RTC_SLOW_CLK. Normally, choose

RC_FAST_CLK as the clock source of RTC_FAST_CLK, and RC_SLOW_CLK as the clock source of RTC_SLOW_CLK. For details, please see Figure 9.3-6.

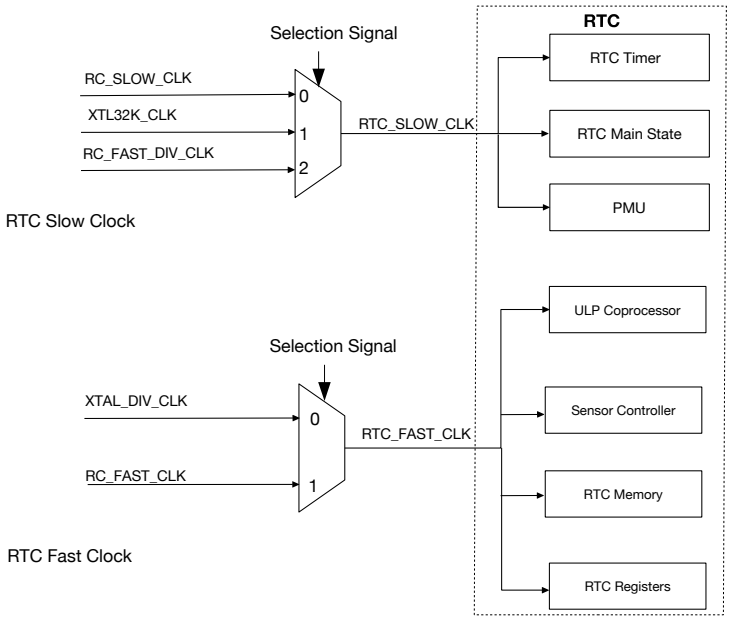



Figure 9.3-6. RTC Low-Power Clocks

For the digital core, LOW_POWER_CLK is switched among four sources (XTL32K_CLK, RTC_SLOW_CLK, RC_FAST_CLK, and XTL_CLK). Configure DPORT_BT_LPCK_DIV_FRAC_REG to select the clock source of LOW_POWER_CLK.

## 9.3.8 Power-Gating Implementation

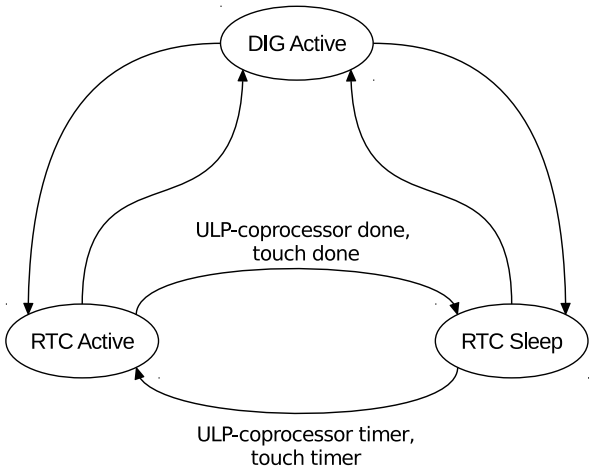



Figure 9.3-7. RTC States

The switch among power-gating states can be seen in Figure 9.3-7. The actual power-control signals could also be set by software as force-power-up (FPU) or force-power-down (FPD) via RTC_CNTL_PWC_REG and RTC_CNTL_DIG_PWC_REG. Since the power domains can be power-gated independently, there are many

combinations for different applications. Table 9.3-1 shows how the power domains in ESP32 are controlled.

**Table 9.3-1. RTC Power Domains**

| Power Domains | | RTC Main State | | | S/W Options | | Notes* |
|---|---|---|---|---|---|---|---|
| | | DIG Active | RTC Active | RTC Sleep | FPU | FPD | |
| RTC | RTC Digital Core | ON | ON | ON | N | N | 1 |
| | RTC Peripherals | ON | ON | OFF | Y | Y | 2 |
| | RTC Slow Memory | ON | OFF | OFF | Y | Y | 3 |
| | RTC Fast Memory | ON | OFF | OFF | Y | Y | 4 |
| Digital | Digital Core | ON | OFF | OFF | Y | Y | 5 |
| | Wi-Fi | ON | OFF | OFF | Y | Y | 6 |
| | ROM | ON | OFF | OFF | Y | Y | - |
| | Internal SRAM | ON | OFF | OFF | Y | Y | 7 |
| Analog | 40 MHz Crystal | ON | OFF | OFF | Y | Y | - |
| | PLL | ON | OFF | OFF | Y | Y | - |
| | 8 MHz OSC | ON | OFF | OFF | Y | Y | - |
| | Radio | - | - | - | Y | Y | - |

Notes*:

1. The power-domain RTC core is the "always-on" power domain, and the FPU/FPD option is not available.
2. The power-domain RTC peripherals include most of the fast logic in RTC, including the ULP co-processor, sensor controllers, etc.
3. The power-domain RTC slow memory should be forced to power on when it is used as retention memory, or when the ULP co-processor is working.
4. The power-domain RTC fast memory should be forced to power on, when it is used as retention memory.
5. When the power-domain digital core is powered down, all included in power domains are powered down.
6. The power-domain Wi-Fi includes the Wi-Fi MAC and BB.
7. Each internal SRAM can be power-gated independently.

### 9.3.9 Predefined Power Modes

In ESP32, we recommend that you always use the predefined power modes first, before trying to tune each power control signal. The predefined power modes should cover most scenarios:

- Active mode
  - The CPU is clocked at XTAL_DIV_N (40 MHz/26 MHz) or PLL (80 MHz/160 MHz/240 MHz).
  - The chip can receive, transmit, or listen.
- Modem-sleep mode
  - The CPU is operational and the clock is configurable.
  - The Wi-Fi/Bluetooth baseband is clock-gated or powered down. The radio is turned off.
  - Current consumption: ~30 mA with 80 MHz PLL.

  - Current consumption: ~3 mA with 2 MHz XTAL.
  - Immediate wake-up.
- Light-sleep mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL, and radio are disabled.
  - The clock in the digital core is gated. The CPUs are stalled.
  - The ULP coprocessor and touch controller can be periodically triggered by monitor sensors.
  - Current consumption: $\sim$ 800 $\mu$A.
  - Wake-up latency: less than 1 ms.
- Deep-sleep mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL and radio are disabled.
  - The digital core is powered down. The CPU context is lost.
  - The supply voltage to the RTC core drops to 0.7V.
  - 8 x 32 bits of data are kept in general-purpose retention registers.
  - The RTC memory and fast RTC memory can be retained.
  - Current consumption: $\sim$ 6.5 $\mu$A.
  - Wake-up latency: less than 1 ms.
  - Recommended for ultra-low-power infrequently-connected Wi-Fi/Bluetooth applications.
- Hibernatation mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL, and radio are disabled.
  - The digital core is powered down. The CPU context is lost.
  - The RTC peripheral domain is powered down.
  - The supply voltage to the RTC core drops to 0.7V.
  - 8 x 32 bits of data are kept in general-purpose retention registers.
  - The RTC memory and fast RTC memory are powered down.
  - Current consumption: $\sim$ 4.5 $\mu$A.
  - Wake-up source: RTC timer only.
  - Wake-up latency: less than 1 ms.
  - Recommended for ultra-low-power infrequently-connected Wi-Fi/Bluetooth applications.

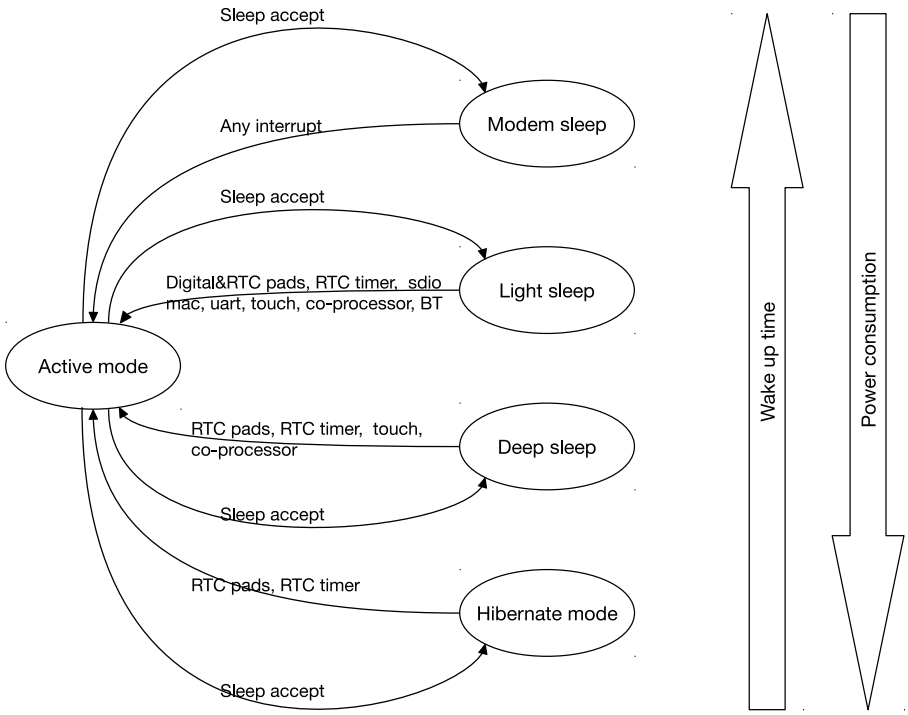



Figure 9.3-8. Power Modes

By default, ESP32 first enters the Modem-sleep mode after a system reset and can be configured to Active mode when transmitting or receiving packets. After the CPU stalls for a while, the chip can enter several low-power modes. It is up to the user to select the mode that best balances power consumption, wake-up latency and available wake-up sources. For details, please see Figure 9.3-8.
Please note that the predefined power mode could be further optimized and adapted to any application.

## 9.3.10 Wakeup Source

The ESP32 supports various wake-up sources, which could wake up the CPU in different sleep modes. The wake-up source is determined by RTC_CNTL_WAKEUP_ENA, as shown in Table 9.3-2.

Table 9.3-2. Wake-up Source

| WAKEUP_ENA | Wake-up Source[1] | Light-sleep | Deep-sleep | Hibernation |
|---|---|---|---|---|
| 0x1 | EXT0[2] | Y | Y | - |
| 0x2 | EXT1[3] | Y | Y | Y |
| 0x4 | GPIO[4] | Y | Y | - |
| 0x8 | RTC timer | Y | Y | Y |
| 0x10 | SDIO[5] | Y | - | - |
| 0x20 | Wi-Fi[6] | Y | - | - |
| 0x40 | UART0 [7] | Y | - | - |
| 0x80 | UART1 [7] | Y | - | - |
| 0x100 | TOUCH | Y | Y | - |
| 0x200 | ULP co-processor | Y | Y | - |
| 0x400 | BT [6] | Y | - | - |

[1] The GPIO and SDIO wakeup sources can also be configured as the causes to reject sleep.

[2] EXT0 can only wake up the chip in light-sleep/deep-sleep mode.
If RTC_CNTL_EXT_WAKEUP0_LV is 1, it is pad high-level triggered; otherwise, it is low-level triggered. Users can set RTCIO_EXT_WAKEUP0_SEL[4:0] to select one of the RTC PADs to be the wake-up source.

[3] EXT1 is especially designed to wake up the chip from any sleep mode, and it also supports multiple pads' combinations.
First, RTC_CNTL_EXT_WAKEUP1_SEL[17:0] should be configured with the bitmap of PADS selected as a wake-up source. Then, if RTC_CNTL_EXT_WAKEUP1_LV is 1, as long as one of the PADs is at high-voltage level, it can trigger a wake-up. However, if RTC_CNTL_EXT_WAKEUP1_LV is 0, it needs all selected PADs to be at low-voltage level to trigger a wake-up.
**Note** that the EXT1 hold time should be longer than three RTC slow clock cycles, otherwise the signal status will not be captured in RTC_CNTL_EXT_WAKEUP1_STATUS.

[4] In Deep-sleep mode, only RTC GPIOs (not DIGITAL GPIOs) can work as wakeup source.

[5] Wake-up is triggered by receiving any SDIO command.

[6] To wake up the chip with a Wi-Fi or BT source, the power mode switches between the Active, Modem- and Light-sleep modes. The CPU, Wi-Fi, Bluetooth, and radio are woken up at predetermined intervals to keep Wi-Fi/BT connections active.

[7] Wake-up is triggered when the number or positive edges of RxD signal is greater than or equal to (UART_ACTIVE_THRESHOLD+2). **Note** that the RxD signal cannot be input through GPIO Matrix but only through IO_MUX.

### 9.3.11 Reject Sleep

ESP32 implements a hardware mechanism that equips the chip with the ability to reject to sleep, which prevents the chip from going to sleep unexpectedly when some peripherals are still working but not detected

by the CPU, thus guaranteeing the proper functioning of the peripherals.

Among the wakeup sources listed in Table 9.3-2, GPIO and SDIO can be configured as the causes to reject sleep.

Users can configure the reject to sleep option via the following registers.

- Configure the RTC_CNTL_SLP_REJECT field to enable or disable the option to reject to sleep:
  - Set RTC_CNTL_LIGHT_SLP_REJECT_EN to enable reject-to-light-sleep.
  - Set RTC_CNTL_DEEP_SLP_REJECT_EN to enable reject-to-deep-sleep.
- Read RTC_CNTL_REJECT_CAUSE to check the reason for rejecting to sleep.

### 9.3.12 RTC Timer

The RTC timer is a 48-bit counter that can be read. The clock is RTC_SLOW_CLK. Any reset/sleep mode, except for the power-up reset, will not stop or reset the RTC timer.

The RTC timer can be used to wake up the CPU at a designated time, and to wake up TOUCH or the ULP coprocessor periodically.

### 9.3.13 RTC Boot

Since the CPU, ROM and RAM are powered down during Deep-sleep and Hibernation mode, the wake-up time is much longer than that in Light sleep/Modem sleep, because of the ROM unpacking and data-copying from the flash (SPI booting). There are two types of SRAM in the RTC, named slow RTC memory and fast RTC memory, which remain powered-on in Deep-sleep mode. For small-scale codes (less than 8 KB), there are two methods of speeding up the wake-up time, i.e. avoiding ROM unpacking and SPI booting.

The first method is to use the RTC slow memory:

1. Set register RTC_CNTL_PROCPU_STAT_VECTOR_SEL for PRO_CPU (or register RTC_CNTL_APPCPU_STAT_VECTOR_SEL for APP_CPU) to 0.
2. Put the chip into sleep.
3. When the CPU is powered up, the reset vector starts from 0x50000000, instead of 0x40000400. ROM unpacking & SPI boot are not needed. The code in RTC memory has to do itself some initialization for the C program environment.

The second method is to use the fast RTC memory:

1. Set register RTC_CNTL_PROCPU_STAT_VECTOR_SEL for PRO_CPU (or register RTC_CNTL_APPCPU_STAT_VECTOR_SEL for APP_CPU) to 1.
2. Calculate CRC for the fast RTC memory, and save the result in register RTC_CNTL_RTC_STORE6_REG[31:0].
3. Input register RTC_CNTL_RTC_STORE7_REG[31:0] with the entry address in the fast RTC memory.
4. Put the chip into sleep.
5. When the CPU is powered up, after ROM unpacking and some necessary initialization, the CRC is calculated again. If the result matches with register RTC_CNTL_RTC_STORE6_REG[31:0], the CPU will jump to the entry address.

The boot flow is shown in Figure 9.3-9.

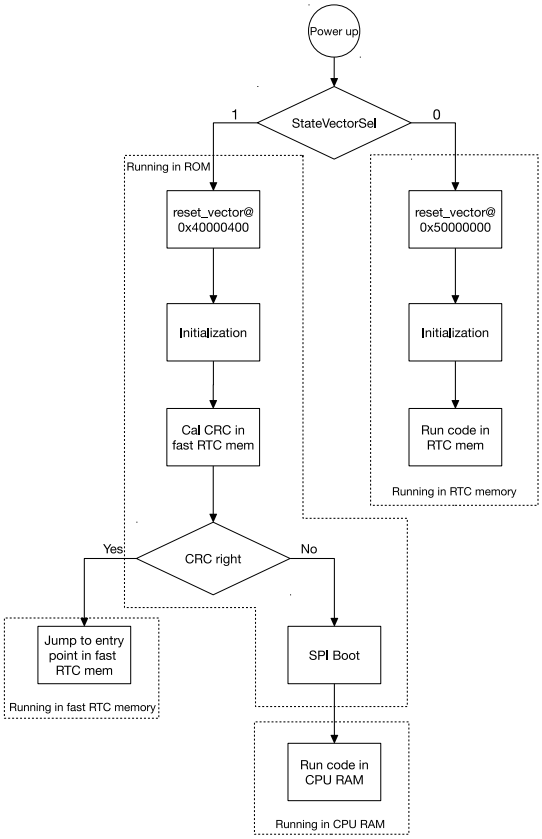



Figure 9.3-9. ESP32 Boot Flow

## 9.4 Register Summary

Notes:

- The registers listed below have been grouped according to their functionality. This particular grouping does not reflect the exact sequential order in which they are stored in memory.
- The base address for registers is 0x60008000 when accessed by AHB, and 0x3FF48000 when accessed by DPORT bus.

| Name | Description | Address | Access |
|---|---|---|---|
| **RTC option register** | | | |
| RTC_CNTL_OPTIONS0_REG | Configure RTC options | 0x3FF48000 | R/W |
| **Control and configuration of RTC timer registers** | | | |
| RTC_CNTL_SLP_TIMER0_REG | RTC sleep timer | 0x3FF48004 | R/W |
| RTC_CNTL_SLP_TIMER1_REG | RTC sleep timer, alarm and control | 0x3FF48008 | R/W |
| RTC_CNTL_TIME_UPDATE_REG | Update control of RTC timer | 0x3FF4800C | RO |
| RTC_CNTL_TIME0_REG | RTC timer low 32 bits | 0x3FF48010 | RO |
| RTC_CNTL_TIME1_REG | RTC timer high 16 bits | 0x3FF48014 | RO |
| RTC_CNTL_STATE0_REG | RTC sleep, SDIO and ULP control | 0x3FF48018 | R/W |
| RTC_CNTL_TIMER1_REG | CPU stall enable | 0x3FF4801C | R/W |
| RTC_CNTL_TIMER2_REG | Slow clock and touch controller configuration | 0x3FF48020 | R/W |
| RTC_CNTL_TIMER5_REG | Minimal sleep cycles in slow clock | 0x3FF4802C | R/W |
| **Reset state and wakeup control registers** | | | |
| RTC_CNTL_RESET_STATE_REG | Reset state control and cause of CPUs | 0x3FF48034 | RO |
| RTC_CNTL_WAKEUP_STATE_REG | Wake-up filter, enable and cause | 0x3FF48038 | RO |
| RTC_CNTL_EXT_WAKEUP_CONF_REG | Configuration of wake-up at low/high level | 0x3FF48060 | R/W |
| RTC_CNTL_EXT_WAKEUP1_REG | Selection of pads for external wake-up and wake-up clear bit | 0x3FF480CC | R/W |
| RTC_CNTL_EXT_WAKEUP1_STATUS_REG | External wake-up status | 0x3FF480D0 | RO |
| **RTC interrupt control and status registers** | | | |
| RTC_CNTL_INT_ENA_REG | Interrupt enable bits | 0x3FF4803C | R/W |
| RTC_CNTL_INT_RAW_REG | Raw interrupt status | 0x3FF48040 | RO |
| RTC_CNTL_INT_ST_REG | Masked interrupt status | 0x3FF48044 | RO |
| RTC_CNTL_INT_CLR_REG | Interrupt clear bits | 0x3FF48048 | WO |
| **RTC general purpose retention registers** | | | |
| RTC_CNTL_STORE0_REG | General purpose retention register 0 | 0x3FF4804C | R/W |
| RTC_CNTL_STORE1_REG | General purpose retention register 1 | 0x3FF48050 | R/W |
| RTC_CNTL_STORE2_REG | General purpose retention register 2 | 0x3FF48054 | R/W |
| RTC_CNTL_STORE3_REG | General purpose retention register 3 | 0x3FF48058 | R/W |
| RTC_CNTL_STORE4_REG | General purpose retention register 4 | 0x3FF480B0 | R/W |
| RTC_CNTL_STORE5_REG | General purpose retention register 5 | 0x3FF480B4 | R/W |
| RTC_CNTL_STORE6_REG | General purpose retention register 6 | 0x3FF480B8 | R/W |
| RTC_CNTL_STORE7_REG | General purpose retention register 7 | 0x3FF480BC | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| **Internal power management registers** | | | |
| RTC_CNTL_ANA_CONF_REG | Power-up/down configuration | 0x3FF48030 | R/W |
| RTC_CNTL_VREG_REG | Internal power distribution and control | 0x3FF4807C | R/W |
| RTC_CNTL_PWC_REG | RTC domain power management | 0x3FF48080 | R/W |
| RTC_CNTL_DIG_PWC_REG | Digital domain power management | 0x3FF48084 | R/W |
| RTC_CNTL_DIG_ISO_REG | Digital domain isolation control | 0x3FF48088 | RO |
| **RTC watchdog configuration and control registers** | | | |
| RTC_CNTL_WDTCONFIG0_REG | WDT Configuration register 0 | 0x3FF4808C | R/W |
| RTC_CNTL_WDTCONFIG1_REG | WDT Configuration register 1 | 0x3FF48090 | R/W |
| RTC_CNTL_WDTCONFIG2_REG | WDT Configuration register 2 | 0x3FF48094 | R/W |
| RTC_CNTL_WDTCONFIG3_REG | WDT Configuration register 3 | 0x3FF48098 | R/W |
| RTC_CNTL_WDTCONFIG4_REG | WDT Configuration register 4 | 0x3FF4809C | R/W |
| RTC_CNTL_WDTFEED_REG | Watchdog feed register | 0x3FF480A0 | WO |
| RTC_CNTL_WDTWPROTECT_REG | Watchdog write protect register | 0x3FF480A4 | R/W |
| **Miscellaneous RTC configuration registers** | | | |
| RTC_CNTL_EXT_XTL_CONF_REG | XTAL control by external pads | 0x3FF4805C | R/W |
| RTC_CNTL_SLP_REJECT_CONF_REG | Reject cause and enable control | 0x3FF48064 | R/W |
| RTC_CNTL_CPU_PERIOD_CONF_REG | CPU period select | 0x3FF48068 | R/W |
| RTC_CNTL_CLK_CONF_REG | Configuration of RTC clocks | 0x3FF48070 | R/W |
| RTC_CNTL_SDIO_CONF_REG | SDIO configuration | 0x3FF48074 | R/W |
| RTC_CNTL_SW_CPU_STALL_REG | Stall of CPUs | 0x3FF480AC | R/W |
| RTC_CNTL_LOW_POWER_ST_REG | RTC state register | 0x3FF480C0 | RO |
| RTC_CNTL_HOLD_FORCE_REG | RTC pad hold register | 0x3FF480C8 | R/W |
| RTC_CNTL_BROWN_OUT_REG | Brownout management | 0x3FF480D4 | R/W |

## 9.5 Registers

The addresses in parenthesis besides register names are the register addresses relative to the Low-power Management (RTC) base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 9.4 *Register Summary*.

**Register 9.1. RTC_CNTL_OPTIONS0_REG (0x0000)**

| 31 | 30 | 29 | 28 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNTL_SW_SYS_RST | RTC_CNTL_DG_WRAP_FORCE_NORST | RTC_CNTL_DG_WRAP_FORCE_RST | (reserved) | | RTC_CNTL_BIAS_CORE_FORCE_PU | RTC_CNTL_BIAS_CORE_FORCE_PD | RTC_CNTL_BIAS_CORE_FOLW_8M | RTC_CNTL_BIAS_I2C_FORCE_PU | RTC_CNTL_BIAS_I2C_FORCE_PD | RTC_CNTL_BIAS_I2C_FOLW_8M | RTC_CNTL_BIAS_FORCE_NOSLEEP | RTC_CNTL_BIAS_FORCE_SLEEP | RTC_CNTL_BIAS_SLEEP_FOLW_8M | RTC_CNTL_XTL_FORCE_PU | RTC_CNTL_XTL_FORCE_PD | RTC_CNTL_BBPLL_FORCE_PU | RTC_CNTL_BBPLL_FORCE_PD | RTC_CNTL_BBPLL_I2C_FORCE_PU | RTC_CNTL_BBPLL_I2C_FORCE_PD | RTC_CNTL_BB_I2C_FORCE_PU | RTC_CNTL_BB_I2C_FORCE_PD | RTC_CNTL_SW_PROCPU_RST | RTC_CNTL_SW_APPCPU_RST | RTC_CNTL_SW_STALL_PROCPU_C0 | RTC_CNTL_SW_STALL_APPCPU_C0 | |
| 0 | 0 | 0 | 0 0 0 0 0 0 | | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 0 | | 0 0 | | Reset |

**RTC_CNTL_SW_SYS_RST** SW system reset. (WO)

**RTC_CNTL_DG_WRAP_FORCE_NORST** The digital core forces no reset in deep sleep. (R/W)

**RTC_CNTL_DG_WRAP_FORCE_RST** The digital core can force a reset in deep sleep. (R/W)

**RTC_CNTL_BIAS_CORE_FORCE_PU** BIAS_CORE force power up. (R/W)

**RTC_CNTL_BIAS_CORE_FORCE_PD** BIAS_CORE force power down. (R/W)

**RTC_CNTL_BIAS_CORE_FOLW_8M** BIAS_CORE follow CK8M. (R/W)

**RTC_CNTL_BIAS_I2C_FORCE_PU** BIAS_I2C force power up. (R/W)

**RTC_CNTL_BIAS_I2C_FORCE_PD** BIAS_I2C force power down. (R/W)

**RTC_CNTL_BIAS_I2C_FOLW_8M** BIAS_I2C follow CK8M. (R/W)

**RTC_CNTL_BIAS_FORCE_NOSLEEP** BIAS_SLEEP force no sleep. (R/W)

**RTC_CNTL_BIAS_FORCE_SLEEP** BIAS_SLEEP force sleep. (R/W)

**RTC_CNTL_BIAS_SLEEP_FOLW_8M** BIAS_SLEEP follow CK8M. (R/W)

**RTC_CNTL_XTL_FORCE_PU** Crystal force power up. (R/W)

**RTC_CNTL_XTL_FORCE_PD** Crystal force power down. (R/W)

**RTC_CNTL_BBPLL_FORCE_PU** BB_PLL force power up. (R/W)

**RTC_CNTL_BBPLL_FORCE_PD** BB_PLL force power down. (R/W)

**RTC_CNTL_BBPLL_I2C_FORCE_PU** BB_PLL_I2C force power up. (R/W)

**RTC_CNTL_BBPLL_I2C_FORCE_PD** BB_PLL _I2C force power down. (R/W)

**RTC_CNTL_BB_I2C_FORCE_PU** BB_I2C force power up. (R/W)

**RTC_CNTL_BB_I2C_FORCE_PD** BB_I2C force power down. (R/W)

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

### Register 9.1. RTC_CNTL_OPTIONS0_REG (0x0000)

Continued from the previous page...

**RTC_CNTL_SW_PROCPU_RST** PRO_CPU SW reset. (WO)

**RTC_CNTL_SW_APPCPU_RST** APP_CPU SW reset. (WO)

**RTC_CNTL_SW_STALL_PROCPU_C0** described under RTC_CNTL_SW_CPU_STALL_REG. (R/W)

**RTC_CNTL_SW_STALL_APPCPU_C0** described under RTC_CNTL_SW_CPU_STALL_REG. (R/W)

### Register 9.2. RTC_CNTL_SLP_TIMER0_REG (0x0004)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RTC_CNTL_SLP_TIMER0_REG** RTC sleep timer low 32 bits. (R/W)

### Register 9.3. RTC_CNTL_SLP_TIMER1_REG (0x0008)

| (reserved) | RTC_CNTL_MAIN_TIMER_ALARM_EN | RTC_CNTL_SLP_VAL_HI | |
|---|---|---|---|
| 31 … 17 | 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0x00000 | Reset |

**RTC_CNTL_MAIN_TIMER_ALARM_EN** Timer alarm enable bit. (R/W)

**RTC_CNTL_SLP_VAL_HI** RTC sleep timer high 16 bits. (R/W)

### Register 9.4. RTC_CNTL_TIME_UPDATE_REG (0x000C)

| RTC_CNTL_TIME_UPDATE | RTC_CNTL_TIME_VALID | (reserved) | |
|---|---|---|---|
| 31 | 30 | 29 … 0 | |
| 0 | 0 | 0 0 0 0 0 0 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 |

**RTC_CNTL_TIME_UPDATE** Set 1: to update register with RTC timer. (WO)

**RTC_CNTL_TIME_VALID** Indicates that the register is updated. (RO)

### Register 9.5. RTC_CNTL_TIME0_REG (0x0010)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RTC_CNTL_TIME0_REG** RTC timer low 32 bits. (RO)

### Register 9.6. RTC_CNTL_TIME1_REG (0x0014)

| (reserved) | RTC_CNTL_TIME_HI | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x00000 | Reset |

**RTC_CNTL_TIME_HI** RTC timer high 16 bits. (RO)

**Register 9.7. RTC_CNTL_STATE0_REG (0x0018)**

| RTC_CNTL_SLEEP_EN | RTC_CNTL_SLP_REJECT | RTC_CNTL_SLP_WAKEUP | RTC_CNTL_SDIO_ACTIVE_IND | (reserved) | RTC_CNTL_ULP_CP_SLP_TIMER_EN | RTC_CNTL_TOUCH_SLP_TIMER_EN | (reserved) | |
|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 – 25 | 24 | 23 | 22 – 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 | 0 | 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 |

**RTC_CNTL_SLEEP_EN** Sleep enable bit. (R/W)

**RTC_CNTL_SLP_REJECT** Sleep reject bit. (R/W)

**RTC_CNTL_SLP_WAKEUP** Sleep wake-up bit. (R/W)

**RTC_CNTL_SDIO_ACTIVE_IND** SDIO active indication. (RO)

**RTC_CNTL_ULP_CP_SLP_TIMER_EN** ULP coprocessor timer enable bit. (R/W)

**RTC_CNTL_TOUCH_SLP_TIMER_EN** Touch timer enable bit. (R/W)

**Register 9.8. RTC_CNTL_TIMER1_REG (0x001C)**

| (reserved) | RTC_CNTL_CPU_STALL_EN | |
|---|---|---|
| 31 – 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 | 1 | Reset |

**RTC_CNTL_CPU_STALL_EN** CPU stall enable bit. (R/W)

**Register 9.9. RTC_CNTL_TIMER2_REG (0x0020)**

| RTC_CNTL_MIN_TIME_CK8M_OFF | RTC_CNTL_ULPCP_TOUCH_START_WAIT | (reserved) | |
|---|---|---|---|
| 31 … 24 | 23 … 15 | 14 … 0 | |
| 0x001 | 0x010 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_MIN_TIME_CK8M_OFF** Minimal amount of cycles in RTC_SLOW_CLK to power down CK8M. (R/W)

**RTC_CNTL_ULPCP_TOUCH_START_WAIT** Awaited cycles in RTC_SLOW_CLK before ULP coprocessor/touch controller starts working. (R/W)

**Register 9.10. RTC_CNTL_TIMER5_REG (0x002C)**

| (reserved) | RTC_CNTL_MIN_SLP_VAL | (reserved) | |
|---|---|---|---|
| 31 … 16 | 15 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x080 | 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_MIN_SLP_VAL** Minimal amount of sleep cycles in RTC_SLOW_CLK. (R/W)

### Register 9.11. RTC_CNTL_ANA_CONF_REG (0x0030)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 … 0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNTL_PLL_I2C_PU | RTC_CNTL_CKGEN_I2C_PU | (reserved) | RTC_CNTL_RFRX_PBUS_PU | RTC_CNTL_TXRF_I2C_PU | RTC_CNTL_PVTMON_PU | (reserved) | RTC_CNTL_PLLA_FORCE_PU | RTC_CNTL_PLLA_FORCE_PD | (reserved) | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_PLL_I2C_PU** 1: PLL_I2C power up, otherwise power down. (R/W)

**RTC_CNTL_CKGEN_I2C_PU** 1: CKGEN_I2C power up, otherwise power down. (R/W)

**RTC_CNTL_RFRX_PBUS_PU** 1: RFRX_PBUS power up, otherwise power down. (R/W)

**RTC_CNTL_TXRF_I2C_PU** 1: TXRF_I2C power up, otherwise power down. (R/W)

**RTC_CNTL_PVTMON_PU** 1: PVTMON power up, otherwise power down. (R/W)

**RTC_CNTL_PLLA_FORCE_PU** PLLA force power up. (R/W)

**RTC_CNTL_PLLA_FORCE_PD** PLLA force power down. (R/W)

### Register 9.12. RTC_CNTL_RESET_STATE_REG (0x0034)

| 31 … 14 | 13 | 12 | 11 … 6 | 5 … 0 | |
|---|---|---|---|---|---|
| (reserved) | RTC_CNTL_PROCPU_STAT_VECTOR_SEL | RTC_CNTL_APPCPU_STAT_VECTOR_SEL | RTC_CNTL_RESET_CAUSE_APPCPU | RTC_CNTL_RESET_CAUSE_PROCPU | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 | 1 | x x x x x x | x x x x x x | Reset |

**RTC_CNTL_PROCPU_STAT_VECTOR_SEL** PRO_CPU state vector selection. (R/W)

**RTC_CNTL_APPCPU_STAT_VECTOR_SEL** APP_CPU state vector selection. (R/W)

**RTC_CNTL_RESET_CAUSE_APPCPU** Reset cause for APP_CPU. (RO)

**RTC_CNTL_RESET_CAUSE_PROCPU** Reset cause for PRO_CPU. (RO)

**Register 9.13. RTC_CNTL_WAKEUP_STATE_REG (0x0038)**

| (reserved) | RTC_CNTL_GPIO_WAKEUP_FILTER | RTC_CNTL_WAKEUP_ENA | RTC_CNTL_WAKEUP_CAUSE | |
|---|---|---|---|---|
| 31 … 23 | 22 | 21 … 11 | 10 … 0 | |
| 0 0 0 0 0 0 0 0 0 | 0 | 0 0 0 0 0 0 0 1 1 0 0 | 0x000 | Reset |

**RTC_CNTL_GPIO_WAKEUP_FILTER** Enable filter for GPIO wake-up event. (R/W)

**RTC_CNTL_WAKEUP_ENA** Wake-up enable bitmap. (R/W)

**RTC_CNTL_WAKEUP_CAUSE** Wake-up cause. (RO)

**Register 9.14. RTC_CNTL_INT_ENA_REG (0x003C)**

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | RTC_CNTL_MAIN_TIMER_INT_ENA | RTC_CNTL_BROWN_OUT_INT_ENA | RTC_CNTL_TOUCH_INT_ENA | RTC_CNTL_ULP_CP_INT_ENA | RTC_CNTL_TIME_VALID_INT_ENA | RTC_CNTL_WDT_INT_ENA | RTC_CNTL_SDIO_IDLE_INT_ENA | RTC_CNTL_SLP_REJECT_INT_ENA | RTC_CNTL_SLP_WAKEUP_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**RTC_CNTL_MAIN_TIMER_INT_ENA** The interrupt enable bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (R/W)

**RTC_CNTL_BROWN_OUT_INT_ENA** The interrupt enable bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (R/W)

**RTC_CNTL_TOUCH_INT_ENA** The interrupt enable bit for the RTC_CNTL_TOUCH_INT interrupt. (R/W)

**RTC_CNTL_ULP_CP_INT_ENA** The interrupt enable bit for the RTC_CNTL_ULP_CP_INT interrupt. (R/W)

**RTC_CNTL_TIME_VALID_INT_ENA** The interrupt enable bit for the RTC_CNTL_TIME_VALID_INT interrupt. (R/W)

**RTC_CNTL_WDT_INT_ENA** The interrupt enable bit for the RTC_CNTL_WDT_INT interrupt. (R/W)

**RTC_CNTL_SDIO_IDLE_INT_ENA** The interrupt enable bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (R/W)

**RTC_CNTL_SLP_REJECT_INT_ENA** The interrupt enable bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (R/W)

**RTC_CNTL_SLP_WAKEUP_INT_ENA** The interrupt enable bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (R/W)

**Register 9.15. RTC_CNTL_INT_RAW_REG (0x0040)**

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | (reserved) | RTC_CNTL_MAIN_TIMER_INT_RAW | RTC_CNTL_BROWN_OUT_INT_RAW | RTC_CNTL_TOUCH_INT_RAW | RTC_CNTL_ULP_CP_INT_RAW | RTC_CNTL_TIME_VALID_INT_RAW | RTC_CNTL_WDT_INT_RAW | RTC_CNTL_SDIO_IDLE_INT_RAW | RTC_CNTL_SLP_REJECT_INT_RAW | RTC_CNTL_SLP_WAKEUP_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RTC_CNTL_MAIN_TIMER_INT_RAW** The raw interrupt status bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (RO)

**RTC_CNTL_BROWN_OUT_INT_RAW** The raw interrupt status bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (RO)

**RTC_CNTL_TOUCH_INT_RAW** The raw interrupt status bit for the RTC_CNTL_TOUCH_INT interrupt. (RO)

**RTC_CNTL_ULP_CP_INT_RAW** The raw interrupt status bit for the RTC_CNTL_ULP_CP_INT interrupt. (RO)

**RTC_CNTL_TIME_VALID_INT_RAW** The raw interrupt status bit for the RTC_CNTL_TIME_VALID_INT interrupt. (RO)

**RTC_CNTL_WDT_INT_RAW** The raw interrupt status bit for the RTC_CNTL_WDT_INT interrupt. (RO)

**RTC_CNTL_SDIO_IDLE_INT_RAW** The raw interrupt status bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (RO)

**RTC_CNTL_SLP_REJECT_INT_RAW** The raw interrupt status bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (RO)

**RTC_CNTL_SLP_WAKEUP_INT_RAW** The raw interrupt status bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (RO)

**Register 9.16. RTC_CNTL_INT_ST_REG (0x0044)**

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | RTC_CNTL_MAIN_TIMER_INT_ST | RTC_CNTL_BROWN_OUT_INT_ST | RTC_CNTL_TOUCH_INT_ST | RTC_CNTL_SAR_INT_ST | RTC_CNTL_TIME_VALID_INT_ST | RTC_CNTL_WDT_INT_ST | RTC_CNTL_SDIO_IDLE_INT_ST | RTC_CNTL_SLP_REJECT_INT_ST | RTC_CNTL_SLP_WAKEUP_INT_ST | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | 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 | 0 | Reset |

**RTC_CNTL_MAIN_TIMER_INT_ST** The masked interrupt status bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (RO)

**RTC_CNTL_BROWN_OUT_INT_ST** The masked interrupt status bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (RO)

**RTC_CNTL_TOUCH_INT_ST** The masked interrupt status bit for the RTC_CNTL_TOUCH_INT interrupt. (RO)

**RTC_CNTL_SAR_INT_ST** The masked interrupt status bit for the RTC_CNTL_SAR_INT interrupt. (RO)

**RTC_CNTL_TIME_VALID_INT_ST** The masked interrupt status bit for the RTC_CNTL_TIME_VALID_INT interrupt. (RO)

**RTC_CNTL_WDT_INT_ST** The masked interrupt status bit for the RTC_CNTL_WDT_INT interrupt. (RO)

**RTC_CNTL_SDIO_IDLE_INT_ST** The masked interrupt status bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (RO)

**RTC_CNTL_SLP_REJECT_INT_ST** The masked interrupt status bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (RO)

**RTC_CNTL_SLP_WAKEUP_INT_ST** The masked interrupt status bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (RO)

### Register 9.17. RTC_CNTL_INT_CLR_REG (0x0048)

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | (reserved) | RTC_CNTL_MAIN_TIMER_INT_CLR | RTC_CNTL_BROWN_OUT_INT_CLR | RTC_CNTL_TOUCH_INT_CLR | RTC_CNTL_SAR_INT_CLR | RTC_CNTL_TIME_VALID_INT_CLR | RTC_CNTL_WDT_INT_CLR | RTC_CNTL_SDIO_IDLE_INT_CLR | RTC_CNTL_SLP_REJECT_INT_CLR | RTC_CNTL_SLP_WAKEUP_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RTC_CNTL_MAIN_TIMER_INT_CLR** Set this bit to clear the RTC_CNTL_MAIN_TIMER_INT interrupt. (WO)

**RTC_CNTL_BROWN_OUT_INT_CLR** Set this bit to clear the RTC_CNTL_BROWN_OUT_INT interrupt. (WO)

**RTC_CNTL_TOUCH_INT_CLR** Set this bit to clear the RTC_CNTL_TOUCH_INT interrupt. (WO)

**RTC_CNTL_SAR_INT_CLR** Set this bit to clear the RTC_CNTL_SAR_INT interrupt. (WO)

**RTC_CNTL_TIME_VALID_INT_CLR** Set this bit to clear the RTC_CNTL_TIME_VALID_INT interrupt. (WO)

**RTC_CNTL_WDT_INT_CLR** Set this bit to clear the RTC_CNTL_WDT_INT interrupt. (WO)

**RTC_CNTL_SDIO_IDLE_INT_CLR** Set this bit to clear the RTC_CNTL_SDIO_IDLE_INT interrupt. (WO)

**RTC_CNTL_SLP_REJECT_INT_CLR** Set this bit to clear the RTC_CNTL_SLP_REJECT_INT interrupt. (WO)

**RTC_CNTL_SLP_WAKEUP_INT_CLR** Set this bit to clear the RTC_CNTL_SLP_WAKEUP_INT interrupt. (WO)

### Register 9.18. RTC_CNTL_STORE*n*_REG (*n*: 0-3) (0x004C+4*n)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | Reset |

**RTC_CNTL_STORE*n*_REG** 32-bit general-purpose retention register. (R/W)

**Register 9.19. RTC_CNTL_LOW_POWER_ST_REG (0x00C0)**

| (reserved) | RTC_CNTL_MAIN_STATE_IN_IDEL | (reserved) | RTC_CNTL_RTC_RDY_FOR_WAKEUP | (reserved) | |
|---|---|---|---|---|---|
| 31 28 | 27 | 26 20 | 19 | 18 0 | |
| 0 0 0 0 | 0 | 0 0 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 |

**RTC_CNTL_RTC_RDY_FOR_WAKEUP** The value 1 indicates the RTC is ready to be triggered by any wakeup source. (RO)

**RTC_CNTL_MAIN_STATE_IN_IDLE** Indicates the RTC state.

- 0: the chip can be either
  - in sleep modes.
  - entering sleep modes. In this case, wait until RTC_CNTL_RTC_RDY_FOR_WAKEUP bit is set, then you can wake up the chip.
  - exiting sleep mode. In this case, RTC_CNTL_MAIN_STATE_IN_IDLE will eventually become 1.
- 1: the chip is not in sleep modes (i.e. running normally).

(RO)

**Register 9.20. RTC_CNTL_EXT_XTL_CONF_REG (0x005C)**

| RTC_CNTL_XTL_EXT_CTR_EN | RTC_CNTL_XTL_EXT_CTR_LV | (reserved) | |
|---|---|---|---|
| 31 | 30 | 29 0 | |
| 0 | 0 | 0 0 0 0 0 0 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 |

**RTC_CNTL_XTL_EXT_CTR_EN** Enable control XTAL with external pads. (R/W)

**RTC_CNTL_XTL_EXT_CTR_LV** 0: power down XTAL at high level, 1: power down XTAL at low level. (R/W)

**Register 9.21. RTC_CNTL_EXT_WAKEUP_CONF_REG (0x0060)**

| RTC_CNTL_EXT_WAKEUP1_LV | RTC_CNTL_EXT_WAKEUP0_LV | (reserved) | |
|---|---|---|---|
| 31 | 30 | 29 … 0 | |
| 0 | 0 | 0 0 0 0 0 0 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 |

**RTC_CNTL_EXT_WAKEUP1_LV** 0: external wake-up at low level, 1: external wake-up at high level. (R/W)

**RTC_CNTL_EXT_WAKEUP0_LV** 0: external wake-up at low level, 1: external wake-up at high level. (R/W)

**Register 9.22. RTC_CNTL_SLP_REJECT_CONF_REG (0x0064)**

| RTC_CNTL_REJECT_CAUSE | RTC_CNTL_DEEP_SLP_REJECT_EN | RTC_CNTL_LIGHT_SLP_REJECT_EN | RTC_CNTL_SDIO_REJECT_EN | RTC_CNTL_GPIO_REJECT_EN | (reserved) | |
|---|---|---|---|---|---|---|
| 31 … 28 | 27 | 26 | 25 | 24 | 23 … 0 | |
| 0 0 0 0 | 0 | 0 | 0 | 0 | 0 0 0 0 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 |

**RTC_CNTL_REJECT_CAUSE** Sleep reject cause. 2: GPIO; 3: SDIO. (RO)

**RTC_CNTL_DEEP_SLP_REJECT_EN** Enable reject for deep sleep. (R/W)

**RTC_CNTL_LIGHT_SLP_REJECT_EN** Enable reject for light sleep. (R/W)

**RTC_CNTL_SDIO_REJECT_EN** Enable SDIO reject. (R/W)

**RTC_CNTL_GPIO_REJECT_EN** Enable GPIO reject. (R/W)

**Register 9.23. RTC_CNTL_CPU_PERIOD_CONF_REG (0x0068)**

| RTC_CNTL_RTC_CPUPERIOD_SEL | RTC_CNTL_CPUSEL_CONF | (reserved) | |
|---|---|---|---|
| 31 30 | 29 | 28 0 | |
| 0 0 | 0 | 0 0 0 0 0 0 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 |

**RTC_CNTL_RTC_CPUPERIOD_SEL** CPU period selection. (R/W)

**RTC_CNTL_CPUSEL_CONF** CPU selection option. (R/W)

**Register 9.24. RTC_CNTL_CLK_CONF_REG (0x0070)**

| RTC_CNTL_ANA_CLK_RTC_SEL | RTC_CNTL_RTC_FAST_CLK_SEL | RTC_CNTL_SOC_CLK_SEL | RTC_CNTL_CK8M_FORCE_PU | RTC_CNTL_CK8M_FORCE_PD | RTC_CNTL_CK8M_DFREQ | (reserved) | RTC_CNTL_CK8M_DIV_SEL | (reserved) | RTC_CNTL_DIG_CLK8M_EN | RTC_CNTL_DIG_CLK8M_D256_EN | RTC_CNTL_DIG_XTAL32K_EN | RTC_CNTL_ENB_CK8M_DIV | RTC_CNTL_ENB_CK8M | RTC_CNTL_CK8M_DIV | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 30 | 29 | 28 27 | 26 | 25 | 24 17 | 16 15 | 14 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 4 | 3 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 1 | 0 0 0 0 | Reset |

**RTC_CNTL_ANA_CLK_RTC_SEL** RTC_SLOW_CLK sel. 0: RC_SLOW_CLK, 1: XTL32K_CLK, 2: RC_FAST_DIV_CLK. (R/W)

**RTC_CNTL_RTC_FAST_CLK_SEL** RTC_FAST_CLK sel. 0: XTAL div 4, 1: CK8M. (R/W)

**RTC_CNTL_SOC_CLK_SEL** SoC clock selection. 0: XTAL, 1: PLL, 2: CK8M, 3: APLL. (R/W)

**RTC_CNTL_CK8M_FORCE_PU** CK8M force power up. (R/W)

**RTC_CNTL_CK8M_FORCE_PD** CK8M force power down. (R/W)

**RTC_CNTL_CK8M_DFREQ** CK8M_DFREQ. (R/W)

**RTC_CNTL_CK8M_DIV_SEL** Divider = reg_rtc_cntl_ck8m_div_sel + 1. (R/W)

**RTC_CNTL_DIG_CLK8M_EN** Enable CK8M for digital core (no relation to RTC core). (R/W)

**RTC_CNTL_DIG_CLK8M_D256_EN** Enable RC_FAST_DIV_CLK for digital core (no relation to RTC core). (R/W)

**RTC_CNTL_DIG_XTAL32K_EN** Enable XTL32K_CLK for digital core (no relation to RTC core). (R/W)

**RTC_CNTL_ENB_CK8M_DIV** 1: RC_FAST_DIV_CLK is actually CK8M, 0: RC_FAST_DIV_CLK is CK8M divided by 256. (R/W)

**RTC_CNTL_ENB_CK8M** Disable CK8M and RC_FAST_DIV_CLK. (R/W)

**RTC_CNTL_CK8M_DIV** RC_FAST_DIV_CLK divider. 00: div128, 01: div256, 10: div512, 11: div1024. (R/W)

**Register 9.25. RTC_CNTL_SDIO_CONF_REG (0x0074)**

| RTC_CNTL_XPD_SDIO_VREG | RTC_CNTL_DREFH_SDIO | RTC_CNTL_DREFM_SDIO | RTC_CNTL_DREFL_SDIO | RTC_CNTL_REG1P8_READY | RTC_CNTL_SDIO_TIEH | RTC_CNTL_SDIO_FORCE | RTC_CNTL_SDIO_VREG_PD_EN | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 29 | 28 27 | 26 25 | 24 | 23 | 22 | 21 | 20 0 | |
| 0 | 0 0 | 0 0 | 0 1 | 0 | 1 | 0 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_XPD_SDIO_VREG** SW option for XPD_SDIO_VREG; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

**RTC_CNTL_DREFH_SDIO** SW option for DREFH_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

**RTC_CNTL_DREFM_SDIO** SW option for DREFM_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

**RTC_CNTL_DREFL_SDIO** SW option for DREFL_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

**RTC_CNTL_REG1P8_READY** Read-only register for REG1P8_READY. (RO)

**RTC_CNTL_SDIO_TIEH** SW option for SDIO_TIEH; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

**RTC_CNTL_SDIO_FORCE** 1: use SW option to control SDIO_VREG; 0: use state machine to control SDIO_VREG. (R/W)

**RTC_CNTL_SDIO_VREG_PD_EN** Power down SDIO_VREG in sleep; active only when reg_rtc_cntl_sdio_force == 0. (R/W)

**Register 9.26. RTC_CNTL_VREG_REG (0x007C)**

| RTC_CNTL_VREG_FORCE_PU | RTC_CNTL_VREG_FORCE_PD | RTC_CNTL_DBOOST_FORCE_PU | RTC_CNTL_DBOOST_FORCE_PD | RTC_CNTL_DBIAS_WAK | RTC_CNTL_DBIAS_SLP | RTC_CNTL_SCK_DCAP | RTC_CNTL_DIG_VREG_DBIAS_WAK | RTC_CNTL_DIG_VREG_DBIAS_SLP | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 25 | 24 22 | 21 14 | 13 11 | 10 8 | 7 0 | |
| 1 | 0 | 1 | 0 | 4 | 4 | 0 | 4 | 4 | 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_VREG_FORCE_PU** RTC voltage regulator - force power up. (R/W)

**RTC_CNTL_VREG_FORCE_PD** RTC voltage regulator - force power down (in this case power down means decreasing the voltage to 0.8 V or lower). (R/W)

**RTC_CNTL_DBOOST_FORCE_PU** RTC_DBOOST force power up. (R/W)

**RTC_CNTL_DBOOST_FORCE_PD** RTC_DBOOST force power down. (R/W)

**RTC_CNTL_DBIAS_WAK** RTC_DBIAS during wake-up. (R/W)

**RTC_CNTL_DBIAS_SLP** RTC_DBIAS during sleep. (R/W)

**RTC_CNTL_SCK_DCAP** Used to adjust the frequency of RTC slow clock. (R/W)

**RTC_CNTL_DIG_VREG_DBIAS_WAK** Digital voltage regulator DBIAS during wake-up. (R/W)

**RTC_CNTL_DIG_VREG_DBIAS_SLP** Digital voltage regulator DBIAS during sleep. (R/W)

**Register 9.27. RTC_CNTL_PWC_REG (0x0080)**

| 31 | | | | | | | | | | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | RTC_CNTL_PD_EN | RTC_CNTL_FORCE_PU | RTC_CNTL_FORCE_PD | RTC_CNTL_SLOWMEM_PD_EN | RTC_CNTL_SLOWMEM_FORCE_PU | RTC_CNTL_SLOWMEM_FORCE_PD | RTC_CNTL_FASTMEM_PD_EN | RTC_CNTL_FASTMEM_FORCE_PU | RTC_CNTL_FASTMEM_FORCE_PD | RTC_CNTL_SLOWMEM_FORCE_LPU | RTC_CNTL_SLOWMEM_FORCE_LPD | RTC_CNTL_SLOWMEM_FOLW_CPU | RTC_CNTL_FASTMEM_FORCE_LPU | RTC_CNTL_FASTMEM_FORCE_LPD | RTC_CNTL_FASTMEM_FOLW_CPU | RTC_CNTL_FORCE_NOISO | RTC_CNTL_FORCE_ISO | RTC_CNTL_SLOWMEM_FORCE_ISO | RTC_CNTL_SLOWMEM_FORCE_NOISO | RTC_CNTL_FASTMEM_FORCE_ISO | RTC_CNTL_FASTMEM_FORCE_NOISO | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | Reset |

**RTC_CNTL_PD_EN** Enable power down rtc_peri in sleep. (R/W)

**RTC_CNTL_FORCE_PU** rtc_peri force power up. (R/W)

**RTC_CNTL_FORCE_PD** rtc_peri force power down. (R/W)

**RTC_CNTL_SLOWMEM_PD_EN** Enable power down RTC memory in sleep. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_PU** RTC memory force power up. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_PD** RTC memory force power down. (R/W)

**RTC_CNTL_FASTMEM_PD_EN** Enable power down fast RTC memory in sleep. (R/W)

**RTC_CNTL_FASTMEM_FORCE_PU** Fast RTC memory force power up. (R/W)

**RTC_CNTL_FASTMEM_FORCE_PD** Fast RTC memory force power down. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_LPU** RTC memory force power up in low-power mode. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_LPD** RTC memory force power down in low-power mode. (R/W)

**RTC_CNTL_SLOWMEM_FOLW_CPU** 1: RTC memory low-power mode PD following CPU; 0: RTC memory low-power mode PD following RTC state machine. (R/W)

**RTC_CNTL_FASTMEM_FORCE_LPU** Fast RTC memory force power up in low-power mode. (R/W)

**RTC_CNTL_FASTMEM_FORCE_LPD** Fast RTC memory force power down in low-power mode. (R/W)

**RTC_CNTL_FASTMEM_FOLW_CPU** 1: Fast RTC memory low-power mode PD following CPU; 0: fast RTC memory low-power mode PD following RTC state machine. (R/W)

**RTC_CNTL_FORCE_NOISO** rtc_peri force no isolation. (R/W)

**RTC_CNTL_FORCE_ISO** rtc_peri force isolation. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_ISO** RTC memory force isolation. (R/W)

**RTC_CNTL_SLOWMEM_FORCE_NOISO** RTC memory force no isolation. (R/W)

**RTC_CNTL_FASTMEM_FORCE_ISO** Fast RTC memory force isolation. (R/W)

**RTC_CNTL_FASTMEM_FORCE_NOISO** Fast RTC memory force no isolation. (R/W)

**Register 9.28. RTC_CNTL_DIG_PWC_REG (0x0084)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNTL_DG_WRAP_PD_EN | RTC_CNTL_WIFI_PD_EN | RTC_CNTL_INTER_RAM4_PD_EN | RTC_CNTL_INTER_RAM3_PD_EN | RTC_CNTL_INTER_RAM2_PD_EN | RTC_CNTL_INTER_RAM1_PD_EN | RTC_CNTL_INTER_RAM0_PD_EN | RTC_CNTL_ROM0_PD_EN | (reserved) | | | RTC_CNTL_DG_WRAP_FORCE_PU | RTC_CNTL_DG_WRAP_FORCE_PD | RTC_CNTL_WIFI_FORCE_PU | RTC_CNTL_WIFI_FORCE_PD | RTC_CNTL_INTER_RAM4_FORCE_PU | RTC_CNTL_INTER_RAM4_FORCE_PD | RTC_CNTL_INTER_RAM3_FORCE_PU | RTC_CNTL_INTER_RAM3_FORCE_PD | RTC_CNTL_INTER_RAM2_FORCE_PU | RTC_CNTL_INTER_RAM2_FORCE_PD | RTC_CNTL_INTER_RAM1_FORCE_PU | RTC_CNTL_INTER_RAM1_FORCE_PD | RTC_CNTL_INTER_RAM0_FORCE_PU | RTC_CNTL_INTER_RAM0_FORCE_PD | RTC_CNTL_ROM0_FORCE_PU | RTC_CNTL_ROM0_FORCE_PD | RTC_CNTL_LSLP_MEM_FORCE_PU | RTC_CNTL_LSLP_MEM_FORCE_PD | (reserved) | | | |
| x | x | x | x | x | x | x | x | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | Reset |

**RTC_CNTL_DG_WRAP_PD_EN** Enable power down digital core in sleep mode. (R/W)

**RTC_CNTL_WIFI_PD_EN** Enable power down Wi-Fi in sleep. (R/W)

**RTC_CNTL_INTER_RAM4_PD_EN** Enable power down internal SRAM 4 in sleep mode. (R/W)

**RTC_CNTL_INTER_RAM3_PD_EN** Enable power down internal SRAM 3 in sleep mode. (R/W)

**RTC_CNTL_INTER_RAM2_PD_EN** Enable power down internal SRAM 2 in sleep mode. (R/W)

**RTC_CNTL_INTER_RAM1_PD_EN** Enable power down internal SRAM 1 in sleep mode. (R/W)

**RTC_CNTL_INTER_RAM0_PD_EN** Enable power down internal SRAM 0 in sleep mode. (R/W)

**RTC_CNTL_ROM0_PD_EN** Enable power down ROM in sleep mode. (R/W)

**RTC_CNTL_DG_WRAP_FORCE_PU** Digital core force power up. (R/W)

**RTC_CNTL_DG_WRAP_FORCE_PD** Digital core force power down. (R/W)

**RTC_CNTL_WIFI_FORCE_PU** Wi-Fi force power up. (R/W)

**RTC_CNTL_WIFI_FORCE_PD** Wi-Fi force power down. (R/W)

**RTC_CNTL_INTER_RAM4_FORCE_PU** Internal SRAM 4 force power up. (R/W)

**RTC_CNTL_INTER_RAM4_FORCE_PD** Internal SRAM 4 force power down. (R/W)

**RTC_CNTL_INTER_RAM3_FORCE_PU** Internal SRAM 3 force power up. (R/W)

**RTC_CNTL_INTER_RAM3_FORCE_PD** Internal SRAM 3 force power down. (R/W)

**RTC_CNTL_INTER_RAM2_FORCE_PU** Internal SRAM 2 force power up. (R/W)

**RTC_CNTL_INTER_RAM2_FORCE_PD** Internal SRAM 2 force power down. (R/W)

**RTC_CNTL_INTER_RAM1_FORCE_PU** Internal SRAM 1 force power up. (R/W)

**RTC_CNTL_INTER_RAM1_FORCE_PD** Internal SRAM 1 force power down. (R/W)

**RTC_CNTL_INTER_RAM0_FORCE_PU** Internal SRAM 0 force power up. (R/W)

**RTC_CNTL_INTER_RAM0_FORCE_PD** Internal SRAM 0 force power down. (R/W)

**RTC_CNTL_ROM0_FORCE_PU** ROM force power up. (R/W)

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

**Register 9.28. RTC_CNTL_DIG_PWC_REG (0x0084)**

**Continued from the previous page...**

**RTC_CNTL_ROM0_FORCE_PD** ROM force power down. (R/W)

**RTC_CNTL_LSLP_MEM_FORCE_PU** Memories in digital core force power up in sleep mode. (R/W)

**RTC_CNTL_LSLP_MEM_FORCE_PD** Memories in digital core force power down in sleep mode. (R/W)

**Register 9.29. RTC_CNTL_DIG_ISO_REG (0x0088)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 … 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNTL_DG_WRAP_FORCE_NOISO | RTC_CNTL_DG_WRAP_FORCE_ISO | RTC_CNTL_WIFI_FORCE_NOISO | RTC_CNTL_WIFI_FORCE_ISO | RTC_CNTL_INTER_RAM4_FORCE_NOISO | RTC_CNTL_INTER_RAM4_FORCE_ISO | RTC_CNTL_INTER_RAM3_FORCE_NOISO | RTC_CNTL_INTER_RAM3_FORCE_ISO | RTC_CNTL_INTER_RAM2_FORCE_NOISO | RTC_CNTL_INTER_RAM2_FORCE_ISO | RTC_CNTL_INTER_RAM1_FORCE_NOISO | RTC_CNTL_INTER_RAM1_FORCE_ISO | RTC_CNTL_INTER_RAM0_FORCE_NOISO | RTC_CNTL_INTER_RAM0_FORCE_ISO | RTC_CNTL_ROM0_FORCE_NOISO | RTC_CNTL_ROM0_FORCE_ISO | RTC_CNTL_DG_PAD_FORCE_HOLD | RTC_CNTL_DG_PAD_FORCE_UNHOLD | RTC_CNTL_DG_PAD_FORCE_ISO | RTC_CNTL_DG_PAD_FORCE_NOISO | RTC_CNTL_REG_RTC_CNTL_DG_PAD_AUTOHOLD_EN | RTC_CNTL_CLR_REG_RTC_CNTL_DG_PAD_AUTOHOLD | RTC_CNTL_DG_PAD_AUTOHOLD | (reserved) | |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_DG_WRAP_FORCE_NOISO** Digital core force no isolation. (R/W)

**RTC_CNTL_DG_WRAP_FORCE_ISO** Digital core force isolation. (R/W)

**RTC_CNTL_WIFI_FORCE_NOISO** Wi-Fi force no isolation. (R/W)

**RTC_CNTL_WIFI_FORCE_ISO** Wi-Fi force isolation. (R/W)

**RTC_CNTL_INTER_RAM4_FORCE_NOISO** Internal SRAM 4 force no isolation. (R/W)

**RTC_CNTL_INTER_RAM4_FORCE_ISO** Internal SRAM 4 force isolation. (R/W)

**RTC_CNTL_INTER_RAM3_FORCE_NOISO** Internal SRAM 3 force no isolation. (R/W)

**RTC_CNTL_INTER_RAM3_FORCE_ISO** Internal SRAM 3 force isolation. (R/W)

**RTC_CNTL_INTER_RAM2_FORCE_NOISO** Internal SRAM 2 force no isolation. (R/W)

**RTC_CNTL_INTER_RAM2_FORCE_ISO** Internal SRAM 2 force isolation. (R/W)

**RTC_CNTL_INTER_RAM1_FORCE_NOISO** Internal SRAM 1 force no isolation. (R/W)

**RTC_CNTL_INTER_RAM1_FORCE_ISO** Internal SRAM 1 force isolation. (R/W)

**RTC_CNTL_INTER_RAM0_FORCE_NOISO** Internal SRAM 0 force no isolation. (R/W)

**RTC_CNTL_INTER_RAM0_FORCE_ISO** Internal SRAM 0 force isolation. (R/W)

**RTC_CNTL_ROM0_FORCE_NOISO** ROM force no isolation. (R/W)

**RTC_CNTL_ROM0_FORCE_ISO** ROM force isolation. (R/W)

**RTC_CNTL_DG_PAD_FORCE_HOLD** Digital pad force hold. (R/W)

**RTC_CNTL_DG_PAD_FORCE_UNHOLD** Digital pad force un-hold. (R/W)

**RTC_CNTL_DG_PAD_FORCE_ISO** Digital pad force isolation. (R/W)

**RTC_CNTL_DG_PAD_FORCE_NOISO** Digital pad force no isolation. (R/W)

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

### Register 9.29. RTC_CNTL_DIG_ISO_REG (0x0088)

Continued from the previous page...

**RTC_CNTL_REG_RTC_CNTL_DG_PAD_AUTOHOLD_EN** Digital pad enable auto-hold. (R/W)

**RTC_CNTL_CLR_REG_RTC_CNTL_DG_PAD_AUTOHOLD** Write-only register clears digital pad auto-hold. (WO)

**RTC_CNTL_DG_PAD_AUTOHOLD** Read-only register indicates digital pad auto-hold status. (RO)

### Register 9.30. RTC_CNTL_WDTCONFIG0_REG (0x008C)

| 31 | 30 28 | 27 25 | 24 22 | 21 19 | 18 | 17 | 16 14 | 13 11 | 10 | 9 | 8 | 7 | 6 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RTC_CNTL_WDT_EN | RTC_CNTL_WDT_STG0 | RTC_CNTL_WDT_STG1 | RTC_CNTL_WDT_STG2 | RTC_CNTL_WDT_STG3 | reserved | reserved | RTC_CNTL_WDT_CPU_RESET_LENGTH | RTC_CNTL_WDT_SYS_RESET_LENGTH | RTC_CNTL_WDT_FLASHBOOT_MOD_EN | RTC_CNTL_WDT_PROCPU_RESET_EN | RTC_CNTL_WDT_APPCPU_RESET_EN | RTC_CNTL_WDT_PAUSE_IN_SLP | reserved | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | Reset |

**RTC_CNTL_WDT_PAUSE_IN_SLP** Pause RTC WDT in sleep. (R/W)

**RTC_CNTL_WDT_APPCPU_RESET_EN** RTC WDT reset APP_CPU enable. (R/W)

**RTC_CNTL_WDT_PROCPU_RESET_EN** RTC WDT reset PRO_CPU enable. (R/W)

**RTC_CNTL_WDT_FLASHBOOT_MOD_EN** Enable RTC WDT in flash boot. (R/W)

**RTC_CNTL_WDT_SYS_RESET_LENGTH** System reset counter length, unit: RTC_SLOW_CLK cycle. The value can be 0 ~ 7. (R/W)

**RTC_CNTL_WDT_CPU_RESET_LENGTH** CPU reset counter length, unit: RTC_SLOW_CLK cycle. The value can be 0 ~ 7. (R/W)

**RTC_CNTL_WDT_STG3** 1: interrupt stage enable, 2: CPU reset stage enable, 3: system reset stage enable, 4: RTC reset stage enable. (R/W)

**RTC_CNTL_WDT_STG2** 1: interrupt stage enable, 2: CPU reset stage enable, 3: system reset stage enable, 4: RTC reset stage enable. (R/W)

**RTC_CNTL_WDT_STG1** 1: interrupt stage enable, 2: CPU reset stage enable, 3: system reset stage enable, 4: RTC reset stage enable. (R/W)

**RTC_CNTL_WDT_STG0** 1: interrupt stage enable, 2: CPU reset stage enable, 3: system reset stage enable, 4: RTC reset stage enable. (R/W)

**RTC_CNTL_WDT_EN** Enable RTC WDT. (R/W)

### Register 9.31. RTC_CNTL_WDTCONFIG*n*_REG (*n*: 1-4) (0x008C+4**n*)

| 31 | 0 | |
|---|---|---|
| 0x000000FFF | | Reset |

**RTC_CNTL_WDTCONFIG*n*_REG** Hold cycles for WDT stage *n*. (R/W)

### Register 9.32. RTC_CNTL_WDTFEED_REG (0x00A0)

| RTC_CNTL_WDT_FEED | (reserved) | |
|---|---|---|
| 31 | 30 … 0 | |
| 0 | 0 0 0 0 0 0 0 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 |

**RTC_CNTL_WDT_FEED** SW feeds WDT. (WO)

### Register 9.33. RTC_CNTL_WDTWPROTECT_REG (0x00A4)

| 31 | 0 | |
|---|---|---|
| 0x050D83AA1 | | Reset |

**RTC_CNTL_WDTWPROTECT_REG** If the register contains a different value than 0x50d83aa1, write protection for the RTC watchdog (RWDT) is enabled. (R/W)

**Register 9.34. RTC_CNTL_SW_CPU_STALL_REG (0x00AC)**

| RTC_CNTL_SW_STALL_PROCPU_C1 | RTC_CNTL_SW_STALL_APPCPU_C1 | (reserved) | |
|---|---|---|---|
| 31 26 | 25 20 | 19 0 | |
| 0 0 0 0 0 0 | 0 0 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 |

**RTC_CNTL_SW_STALL_PROCPU_C1** reg_rtc_cntl_sw_stall_procpu_c1[5:0], reg_rtc_cntl_sw_stall_procpu_c0[1:0] == 0x86 (100001 10) will stall PRO_CPU, see also RTC_CNTL_OPTIONS0_REG. (R/W)

**RTC_CNTL_SW_STALL_APPCPU_C1** reg_rtc_cntl_sw_stall_appcpu_c1[5:0], reg_rtc_cntl_sw_stall_appcpu_c0[1:0] == 0x86 (100001 10) will stall APP_CPU, see also RTC_CNTL_OPTIONS0_REG. (R/W)

**Register 9.35. RTC_CNTL_STORE*n*_REG (*n*: 4-7) (0x00B0+4*(*n*-4))**

| 31 0 | |
|---|---|
| x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x | Reset |

**RTC_CNTL_STORE*n*_REG** 32-bit general-purpose retention register. (R/W)

**Register 9.36. RTC_CNTL_HOLD_FORCE_REG (0x00C8)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | RTC_CNTL_X32N_HOLD_FORCE | RTC_CNTL_X32P_HOLD_FORCE | RTC_CNTL_TOUCH_PAD7_HOLD_FORCE | RTC_CNTL_TOUCH_PAD6_HOLD_FORCE | RTC_CNTL_TOUCH_PAD5_HOLD_FORCE | RTC_CNTL_TOUCH_PAD4_HOLD_FORCE | RTC_CNTL_TOUCH_PAD3_HOLD_FORCE | RTC_CNTL_TOUCH_PAD2_HOLD_FORCE | RTC_CNTL_TOUCH_PAD1_HOLD_FORCE | RTC_CNTL_TOUCH_PAD0_HOLD_FORCE | RTC_CNTL_SENSE4_HOLD_FORCE | RTC_CNTL_SENSE3_HOLD_FORCE | RTC_CNTL_SENSE2_HOLD_FORCE | RTC_CNTL_SENSE1_HOLD_FORCE | RTC_CNTL_PDAC2_HOLD_FORCE | RTC_CNTL_PDAC1_HOLD_FORCE | RTC_CNTL_ADC2_HOLD_FORCE | RTC_CNTL_ADC1_HOLD_FORCE | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RTC_CNTL_X32N_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_X32P_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD7_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD6_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD5_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD4_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD3_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD2_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD1_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_TOUCH_PAD0_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_SENSE4_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_SENSE3_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_SENSE2_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_SENSE1_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_PDAC2_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_PDAC1_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_ADC2_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**RTC_CNTL_ADC1_HOLD_FORCE** Set to preserve pad's state during hibernation. (R/W)

**Register 9.37. RTC_CNTL_EXT_WAKEUP1_REG (0x00CC)**

| (reserved) | RTC_CNTL_EXT_WAKEUP1_STATUS_CLR | RTC_CNTL_EXT_WAKEUP1_SEL | |
|---|---|---|---|
| 31 19 | 18 | 17 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | Reset |

**RTC_CNTL_EXT_WAKEUP1_STATUS_CLR** Clear external wakeup1 status. (WO)

**RTC_CNTL_EXT_WAKEUP1_SEL** Bitmap to select RTC pads for external wakeup1. (R/W)

**Register 9.38. RTC_CNTL_EXT_WAKEUP1_STATUS_REG (0x00D0)**

| (reserved) | RTC_CNTL_EXT_WAKEUP1_STATUS | |
|---|---|---|
| 31 18 | 17 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**RTC_CNTL_EXT_WAKEUP1_STATUS** EXT1 wakeup source status. (RO)

**Register 9.39. RTC_CNTL_BROWN_OUT_REG (0x00D4)**

| RTC_CNTL_BROWN_OUT_DET | RTC_CNTL_BROWN_OUT_ENA | RTC_CNTL_DBROWN_OUT_THRES | RTC_CNTL_BROWN_OUT_RST_ENA | RTC_CNTL_BROWN_OUT_RST_WAIT | RTC_CNTL_BROWN_OUT_PD_RF_ENA | RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA | (reserved) | |
|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 27 | 26 | 25 16 | 15 | 14 | 13 0 | |
| 0 | 0 | 0x2 | 0 | 0x3FF | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**RTC_CNTL_BROWN_OUT_DET** Brownout detect. (RO)

**RTC_CNTL_BROWN_OUT_ENA** Enables brownout. (R/W)

**RTC_CNTL_DBROWN_OUT_THRES** Brownout threshold. The brownout detector will reset the chip when the supply voltage is approximately below this level. Note that there may be some variation of brownout voltage level between each ESP32 chip. 0: 2.43 V ± 0.05; 1: 2.48 V ± 0.05; 2: 2.58 V ± 0.05; 3: 2.62 V ± 0.05; 4: 2.67 V ± 0.05; 5: 2.70 V ± 0.05; 6: 2.77 V ± 0.05; 7: 2.80 V ± 0.05. (R/W)

**RTC_CNTL_BROWN_OUT_RST_ENA** Enables brownout reset. (R/W)

**RTC_CNTL_BROWN_OUT_RST_WAIT** Brownout reset wait cycles. (R/W)

**RTC_CNTL_BROWN_OUT_PD_RF_ENA** Enables power down RF when brownout happens. (R/W)

**RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA** Sends suspend command to flash when brownout happens. (R/W)

# Chapter 10

# Timer Group (TIMG)

## 10.1 Introduction

There are four general-purpose timers embedded in the ESP32. They are all 64-bit generic timers based on 16-bit prescalers and 64-bit auto-reload-capable up/downcounters.

The ESP32 contains two timer modules, each containing two timers. The two timers in a block are indicated by an *x* in TIMG*n*_T*x*; the blocks themselves are indicated by an *n*.

The timers feature:

- A 16-bit clock prescaler, from 2 to 65536
- A 64-bit time-base counter
- Configurable up/down time-base counter: incrementing or decrementing
- Halt and resume of time-base counter
- Auto-reload at alarm
- Software-controlled instant reload
- Level and edge interrupt generation

## 10.2 Functional Description

### 10.2.1 16-bit Prescaler

Each timer uses the APB clock (APB_CLK, normally 80 MHz) as the basic clock. This clock is then divided down by a 16-bit precaler which generates the time-base counter clock (TB_clk). Every cycle of TB_clk causes the time-base counter to increment or decrement by one. The timer must be disabled (TIMG*n*_T*x*_EN is cleared) before changing the prescaler divisor which is configured by TIMG*n*_T*x*_DIVIDER register; changing it on an enabled timer can lead to unpredictable results. The prescaler can divide the APB clock by a factor from 2 to 65536. Specifically, when TIMG*n*_T*x*_DIVIDER is either 1 or 2, the clock divisor is 2; when TIMG*n*_T*x*_DIVIDER is 0, the clock divisor is 65536. Any other value will cause the clock to be divided by exactly that value.

### 10.2.2 64-bit Time-base Counter

The 64-bit time-base counter can be configured to count either up or down, depending on whether TIMG*n*_T*x*_INCREASE is set or cleared, respectively. It supports both auto-reload and software instant reload. An alarm event can be set when the counter reaches a value specified by the software.

Counting can be enabled and disabled by setting and clearing TIMG*n*_T*x*_EN. Clearing this bit essentially freezes the counter, causing it to neither count up nor count down; instead, it retains its value until TIMG*n*_T*x*_EN is set again. Reloading the counter when TIMG*n*_T*x*_EN is cleared will change its value, but counting will not be resumed until TIMG*n*_T*x*_EN is set.

Software can set a new counter value by setting registers TIMG*n*_T*x*_LOAD_LO and TIMG*n*_T*x*_LOAD_HI to the intended new value. The hardware will ignore these register settings until a reload; a reload will cause the contents of these registers to be copied to the counter itself. A reload event can be triggered by an alarm (auto-reload at alarm) or by software (software instant reload). To enable auto-reload at alarm, the register TIMG*n*_T*x*_AUTORELOAD should be set. If auto-reload at alarm is not enabled, the time-base counter will continue incrementing or decrementing after the alarm. To trigger a software instant reload, any value can be written to the register TIMG*n*_T*x*_LOAD_REG; this will cause the counter value to change instantly. Software can also change the direction of the time-base counter instantly by changing the value of TIMG*n*_T*x*_INCREASE.

The time-base counter can also be read by software, but because the counter is 64-bit, the CPU can only get the value as two 32-bit values, the counter value needs to be latched onto TIMG*n*_T*x*LO_REG and TIMG*n*_T*x*HI_REG first. This is done by writing any value to TIMG*n*_T*x*UPDATE_REG; this will instantly latch the 64-bit timer value onto the two registers. Software can then read them at any point in time. This approach stops the timer value being read erroneously when a carry-over happens between reading the low and high word of the timer value.

### 10.2.3 Alarm Generation

The timer can trigger an alarm, which can cause a reload and/or an interrupt to occur. The alarm is triggered when the alarm registers TIMG*n*_T*x*_ALARMLO_REG and TIMG*n*_T*x*_ALARMHI_REG match the current timer value. In order to simplify the scenario where these registers are set 'too late' and the counter has already passed these values, the alarm also triggers when the current timer value is higher (for an up-counting timer) or lower (for a down-counting timer) than the current alarm value: if this is the case, the alarm will be triggered immediately upon loading the alarm registers. The timer alarm enable bit is automatically cleared once an alarm occurs.

### 10.2.4 MWDT

Each timer module also contains a Main System Watchdog Timer and its associated registers. While these registers are described here, their functional description can be found in the chapter entitled Watchdog Timer.

### 10.2.5 Interrupts

- TIMG*n*_INT_WDT_INT: Generated when a watchdog timer interrupt stage times out.
- TIMG*n*_INT_T1_INT: An alarm event on timer 1 generates this interrupt.
- TIMG*n*_INT_T0_INT: An alarm event on timer 0 generates this interrupt.

## 10.3 Register Summary

| Name | Description | TIMG0 | TIMG1 | Acc |
|---|---|---|---|---|
| **Timer 0 configuration and control registers** | | | | |
| TIMG*n*_T0CONFIG_REG | Timer 0 configuration register | 0x3FF5F000 | 0x3FF60000 | R/W |
| TIMG*n*_T0LO_REG | Timer 0 current value, low 32 bits | 0x3FF5F004 | 0x3FF60004 | RO |
| TIMG*n*_T0HI_REG | Timer 0 current value, high 32 bits | 0x3FF5F008 | 0x3FF60008 | RO |
| TIMG*n*_T0UPDATE_REG | Write to copy current timer value to TIMG*n*_T0_(LO/HI)_REG | 0x3FF5F00C | 0x3FF6000C | WO |
| TIMG*n*_T0ALARMLO_REG | Timer 0 alarm value, low 32 bits | 0x3FF5F010 | 0x3FF60010 | R/W |
| TIMG*n*_T0ALARMHI_REG | Timer 0 alarm value, high bits | 0x3FF5F014 | 0x3FF60014 | R/W |
| TIMG*n*_T0LOADLO_REG | Timer 0 reload value, low 32 bits | 0x3FF5F018 | 0x3FF60018 | R/W |
| TIMG*n*_T0LOADHI_REG | Timer 0 reload value, high 32 bits | 0x3FF5F01C | 0x3FF6001C | R/W |
| TIMG*n*_T0LOAD_REG | Write to reload timer from TIMG*n*_T0_(LOADLOLOADHI)_REG | 0x3FF5F020 | 0x3FF60020 | WO |
| **Timer 1 configuration and control registers** | | | | |
| TIMG*n*_T1CONFIG_REG | Timer 1 configuration register | 0x3FF5F024 | 0x3FF60024 | R/W |
| TIMG*n*_T1LO_REG | Timer 1 current value, low 32 bits | 0x3FF5F028 | 0x3FF60028 | RO |
| TIMG*n*_T1HI_REG | Timer 1 current value, high 32 bits | 0x3FF5F02C | 0x3FF6002C | RO |
| TIMG*n*_T1UPDATE_REG | Write to copy current timer value to TIMG*n*_T1_(LO/HI)_REG | 0x3FF5F030 | 0x3FF60030 | WO |
| TIMG*n*_T1ALARMLO_REG | Timer 1 alarm value, low 32 bits | 0x3FF5F034 | 0x3FF60034 | R/W |
| TIMG*n*_T1ALARMHI_REG | Timer 1 alarm value, high 32 bits | 0x3FF5F038 | 0x3FF60038 | R/W |
| TIMG*n*_T1LOADLO_REG | Timer 1 reload value, low 32 bits | 0x3FF5F03C | 0x3FF6003C | R/W |
| TIMG*n*_T1LOADHI_REG | Timer 1 reload value, high 32 bits | 0x3FF5F040 | 0x3FF60040 | R/W |
| TIMG*n*_T1LOAD_REG | Write to reload timer from TIMG*n*_T1_(LOADLOLOADHI)_REG | 0x3FF5F044 | 0x3FF60044 | WO |
| **System watchdog timer configuration and control registers** | | | | |
| TIMG*n*_WDTCONFIG0_REG | Watchdog timer configuration register | 0x3FF5F048 | 0x3FF60048 | R/W |
| TIMG*n*_WDTCONFIG1_REG | Watchdog timer prescaler register | 0x3FF5F04C | 0x3FF6004C | R/W |
| TIMG*n*_WDTCONFIG2_REG | Watchdog timer stage 0 timeout value | 0x3FF5F050 | 0x3FF60050 | R/W |
| TIMG*n*_WDTCONFIG3_REG | Watchdog timer stage 1 timeout value | 0x3FF5F054 | 0x3FF60054 | R/W |
| TIMG*n*_WDTCONFIG4_REG | Watchdog timer stage 2 timeout value | 0x3FF5F058 | 0x3FF60058 | R/W |
| TIMG*n*_WDTCONFIG5_REG | Watchdog timer stage 3 timeout value | 0x3FF5F05C | 0x3FF6005C | R/W |
| TIMG*n*_WDTFEED_REG | Write to feed the watchdog timer | 0x3FF5F060 | 0x3FF60060 | WO |
| TIMG*n*_WDTWPROTECT_REG | Watchdog write protect register | 0x3FF5F064 | 0x3FF60064 | R/W |
| **Configuration and Control Register for RTC CALI** | | | | |
| TIMG*n*_RTCCALICFG_REG | RTC calibration configuration register | 0x3FF5F068 | 0x3FF60068 | varies |
| TIMG*n*_RTCCALICFG1_REG | RTC calibration configuration register 1 | 0x3FF5F06C | 0x3FF6006C | RO |
| **Interrupt registers** | | | | |
| TIMG*n*_INT_ENA_REG | Interrupt enable bits | 0x3FF5F098 | 0x3FF60098 | R/W |

| Name | Description | TIMG0 | TIMG1 | Acc |
|---|---|---|---|---|
| TIMG*n*_INT_RAW_REG | Raw interrupt status | 0x3FF5F09C | 0x3FF6009C | RO |
| TIMG*n*_INT_ST_REG | Masked interrupt status | 0x3FF5F0A0 | 0x3FF600A0 | RO |
| TIMG*n*_INT_CLR_REG | Interrupt clear bits | 0x3FF5F0A4 | 0x3FF600A4 | WO |

## 10.4 Registers

The addresses in parenthesis besides register names are the register addresses relative to the TIMG base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 10.3 *Register Summary*.

**Register 10.1. TIMG*n*_T*x*CONFIG_REG (*x*: 0-1) (0x0+0x24**x*)**

| TIMGn_Tx_EN | TIMGn_Tx_INCREASE | TIMGn_Tx_AUTORELOAD | TIMGn_Tx_DIVIDER | TIMGn_Tx_EDGE_INT_EN | TIMGn_Tx_LEVEL_INT_EN | TIMGn_Tx_ALARM_EN | (reserved) | |
|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 … 13 | 12 | 11 | 10 | 9 … 0 | |
| 0 | 1 | 1 | 0x00001 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**TIMG*n*_T*x*_EN** When set, the timer *x* time-base counter is enabled. (R/W)

**TIMG*n*_T*x*_INCREASE** When set, the timer *x* time-base counter will increment every clock tick. When cleared, the timer *x* time-base counter will decrement. (R/W)

**TIMG*n*_T*x*_AUTORELOAD** When set, timer *x* auto-reload at alarm is enabled. (R/W)

**TIMG*n*_T*x*_DIVIDER** Timer *x* clock (T*x*_clk) prescale value. (R/W)

**TIMG*n*_T*x*_EDGE_INT_EN** When set, an alarm will generate an edge type interrupt. (R/W)

**TIMG*n*_T*x*_LEVEL_INT_EN** When set, an alarm will generate a level type interrupt. (R/W)

**TIMG*n*_T*x*_ALARM_EN** When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W)

**Register 10.2. TIMG*n*_T*x*LO_REG (*x*: 0-1) (0x4+0x24**x*)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*LO_REG** After writing to TIMG*n*_T*x*UPDATE_REG, the low 32 bits of the time-base counter of timer *x* can be read here. (RO)

**Register 10.3. TIMG*n*_T*x*HI_REG (*x*: 0-1) (0x8+0x24**x*)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*HI_REG** After writing to TIMG*n*_T*x*UPDATE_REG, the high 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 10.4. TIMG*n*_T*x*UPDATE_REG (*x*: 0-1) (0xC+0x24*x)

| 31 | 0 |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*UPDATE_REG** Write any value to trigger a timer *x* time-base counter value update (timer *x* current value will be stored in registers above). (WO)

Register 10.5. TIMG*n*_T*x*ALARMLO_REG (*x*: 0-1) (0x10+0x24*x)

| 31 | 0 |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*ALARMLO_REG** Timer *x* alarm trigger time-base counter value, low 32 bits. (R/W)

Register 10.6. TIMG*n*_T*x*ALARMHI_REG (*x*: 0-1) (0x14+0x24*x)

| 31 | 0 |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*ALARMHI_REG** Timer *x* alarm trigger time-base counter value, high 32 bits. (R/W)

Register 10.7. TIMG*n*_T*x*LOADLO_REG (*x*: 0-1) (0x18+0x24*x)

| 31 | 0 |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*LOADLO_REG** Low 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 10.8. TIMG*n*_T*x*LOADHI_REG (*x*: 0-1) (0x1C+0x24*x)

| 31 | 0 |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*LOADHI_REG** High 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

**Register 10.9. TIMG*n*_T*x*LOAD_REG (*x*: 0-1) (0x20+0x24*x)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_T*x*LOAD_REG** Write any value to trigger a timer *x* time-base counter reload. (WO)

**Register 10.10. TIMG*n*_WDTCONFIG0_REG (0x0048)**

| TIMGn_WDT_EN | TIMGn_WDT_STG0 | TIMGn_WDT_STG1 | TIMGn_WDT_STG2 | TIMGn_WDT_STG3 | TIMGn_WDT_EDGE_INT_EN | TIMGn_WDT_LEVEL_INT_EN | TIMGn_WDT_CPU_RESET_LENGTH | TIMGn_WDT_SYS_RESET_LENGTH | TIMGn_WDT_FLASHBOOT_MOD_EN | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 29 | 28 27 | 26 25 | 24 23 | 22 | 21 | 20 18 | 17 15 | 14 | 13 … 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x1 | 0x1 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**TIMG*n*_WDT_EN** When set, MWDT is enabled. (R/W)

**TIMG*n*_WDT_STG0** Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG*n*_WDT_STG1** Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG*n*_WDT_STG2** Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG*n*_WDT_STG3** Stage 3 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG*n*_WDT_EDGE_INT_EN** When set, an edge type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

**TIMG*n*_WDT_LEVEL_INT_EN** When set, a level type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

**TIMG*n*_WDT_CPU_RESET_LENGTH** CPU reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 $\mu$s, 7: 3.2 $\mu$s. (R/W)

**TIMG*n*_WDT_SYS_RESET_LENGTH** System reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 $\mu$s, 7: 3.2 $\mu$s. (R/W)

**TIMG*n*_WDT_FLASHBOOT_MOD_EN** When set, Flash boot protection is enabled. (R/W)

### Register 10.11. TIMG*n*_WDTCONFIG1_REG (0x004c)

| TIMG*n*_WDT_CLK_PRESCALE | (reserved) | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00001 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**TIMG*n*_WDT_CLK_PRESCALE** MWDT clock prescale value. MWDT clock period = MWDT's clock source period * TIMG*n*_WDT_CLK_PRESCALE. (R/W)

### Register 10.12. TIMG*n*_WDTCONFIG2_REG (0x0050)

| 31 … 0 | |
|---|---|
| 26000000 | Reset |

**TIMG*n*_WDTCONFIG2_REG** Stage 0 timeout value, in MWDT clock cycles. (R/W)

### Register 10.13. TIMG*n*_WDTCONFIG3_REG (0x0054)

| 31 … 0 | |
|---|---|
| 0x007FFFFFF | Reset |

**TIMG*n*_WDTCONFIG3_REG** Stage 1 timeout value, in MWDT clock cycles. (R/W)

### Register 10.14. TIMG*n*_WDTCONFIG4_REG (0x0058)

| 31 … 0 | |
|---|---|
| 0x0000FFFFF | Reset |

**TIMG*n*_WDTCONFIG4_REG** Stage 2 timeout value, in MWDT clock cycles. (R/W)

### Register 10.15. TIMG*n*_WDTCONFIG5_REG (0x005c)

| 31 … 0 | |
|---|---|
| 0x0000FFFFF | Reset |

**TIMG*n*_WDTCONFIG5_REG** Stage 3 timeout value, in MWDT clock cycles. (R/W)

### Register 10.16. TIMG*n*_WDTFEED_REG (0x0060)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TIMG*n*_WDTFEED_REG** Write any value to feed the MWDT. (WO)

### Register 10.17. TIMG*n*_WDTWPROTECT_REG (0x0064)

| 31 … 0 | |
|---|---|
| 0x050D83AA1 | Reset |

**TIMG*n*_WDTWPROTECT_REG** If the register contains a different value than its reset value, write protection is enabled. (R/W)

### Register 10.18. TIMG*n*_RTCCALICFG_REG (0x0068)

| TIMG*n*_RTC_CALI_START | TIMG*n*_RTC_CALI_MAX | TIMG*n*_RTC_CALI_RDY | TIMG*n*_RTC_CALI_CLK_SEL | TIMG*n*_RTC_CALI_START_CYCLING | (reserved) | |
|---|---|---|---|---|---|---|
| 31 | 30 … 16 | 15 | 14 … 13 | 12 | 11 … 0 | |
| 0 | 0x01 | 0 | 0x1 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**TIMG*n*_RTC_CALI_START_CYCLING** Reserved. (R/W)

**TIMG*n*_RTC_CALI_CLK_SEL** Used to select the clock to be calibrated. 0: RC_SLOW_CLK. 1: RC_FAST_DIV_CLK. 2: XTAL32K_CLK. (R/W)

**TIMG*n*_RTC_CALI_RDY** Set this bit to mark the completion of calibration. (RO)

**TIMG*n*_RTC_CALI_MAX** Calibration time, in cycles of the clock to be calibrated. (R/W)

**TIMG*n*_RTC_CALI_START** Set this bit to starts calibration. (R/W)

### Register 10.19. TIMGn_RTCCALICFG1_REG (0x006C)

| TIMGn_RTC_CALI_VALUE | (reserved) | |
|---|---|---|
| 31 … 7 | 6 … 0 | |
| 0x00000 | 0 0 0 0 0 0 0 | Reset |

**TIMGn_RTC_CALI_VALUE** Calibration value when cycles of clock to be calibrated reach TIMGn_RTC_CALI_MAX, in unit of XTAL_CLK clock cycles. (RO)

### Register 10.20. TIMGn_INT_ENA_REG (0x0098)

| (reserved) | TIMGn_INT_WDT_INT_ENA | TIMGn_INT_T1_INT_ENA | TIMGn_INT_T0_INT_ENA | |
|---|---|---|---|---|
| 31 … 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 | 0 | 0 | Reset |

**TIMGn_INT_WDT_INT_ENA** The interrupt enable bit for the TIMGn_INT_WDT_INT interrupt. (R/W) (R/W)

**TIMGn_INT_T1_INT_ENA** The interrupt enable bit for the TIMGn_INT_T1_INT interrupt. (R/W) (R/W)

**TIMGn_INT_T0_INT_ENA** The interrupt enable bit for the TIMGn_INT_T0_INT interrupt. (R/W) (R/W)

### Register 10.21. TIMGn_INT_RAW_REG (0x009c)

| (reserved) | TIMGn_INT_WDT_INT_RAW | TIMGn_INT_T1_INT_RAW | TIMGn_INT_T0_INT_RAW | |
|---|---|---|---|---|
| 31 … 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 | 0 | 0 | Reset |

**TIMGn_INT_WDT_INT_RAW** The raw interrupt status bit for the TIMGn_INT_WDT_INT interrupt. (RO)

**TIMGn_INT_T1_INT_RAW** The raw interrupt status bit for the TIMGn_INT_T1_INT interrupt. (RO)

**TIMGn_INT_T0_INT_RAW** The raw interrupt status bit for the TIMGn_INT_T0_INT interrupt. (RO)

**Register 10.22. TIMG*n*_INT_ST_REG (0x00a0)**

| (reserved) | TIMG*n*_INT_WDT_INT_ST | TIMG*n*_INT_T1_INT_ST | TIMG*n*_INT_T0_INT_ST | |
|---|---|---|---|---|
| 31 … 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 | 0 | 0 | Reset |

**TIMG*n*_INT_WDT_INT_ST** The masked interrupt status bit for the TIMG*n*_INT_WDT_INT interrupt. (RO)

**TIMG*n*_INT_T1_INT_ST** The masked interrupt status bit for the TIMG*n*_INT_T1_INT interrupt. (RO)

**TIMG*n*_INT_T0_INT_ST** The masked interrupt status bit for the TIMG*n*_INT_T0_INT interrupt. (RO)

**Register 10.23. TIMG*n*_INT_CLR_REG (0x00a4)**

| (reserved) | TIMG*n*_INT_WDT_INT_CLR | TIMG*n*_INT_T1_INT_CLR | TIMG*n*_INT_T0_INT_CLR | |
|---|---|---|---|---|
| 31 … 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 | 0 | 0 | Reset |

**TIMG*n*_INT_WDT_INT_CLR** Set this bit to clear the TIMG*n*_INT_WDT_INT interrupt. (WO)

**TIMG*n*_INT_T1_INT_CLR** Set this bit to clear the TIMG*n*_INT_T1_INT interrupt. (WO)

**TIMG*n*_INT_T0_INT_CLR** Set this bit to clear the TIMG*n*_INT_T0_INT interrupt. (WO)

# Chapter 11

# Watchdog Timers (WDT)

## 11.1 Introduction

The ESP32 has three watchdog timers: one in each of the two timer modules (called Main System Watchdog Timer, or MWDT) and one in the RTC module (which is called the RTC Watchdog Timer, or RWDT). These watchdog timers are intended to recover from an unforeseen fault, causing the application program to abandon its normal sequence. A watchdog timer has four stages. Each stage may take one out of three or four actions upon the expiry of a programmed period of time for this stage, unless the watchdog is fed or disabled. The actions are: interrupt, CPU reset, core reset and system reset. Only the RWDT can trigger the system reset, and is able to reset the entire chip and the main system including the RTC itself. A timeout value can be set for each stage individually.

During flash boot, the RWDT and the first MWDT start automatically in order to detect and recover from booting problems.

## 11.2 Features

- Four stages, each of which can be configured or disabled separately
- Programmable time period for each stage
- One out of three or four possible actions (interrupt, CPU reset, core reset and system reset) upon the expiry of each stage
- 32-bit expiry counter
- Write protection, to prevent the RWDT and MWDT configuration from being inadvertently altered.
- Flash boot protection
  If the boot process from an SPI flash does not complete within a predetermined period of time, the watchdog will reboot the entire main system.

## 11.3 Functional Description

### 11.3.1 Clock

The RWDT is clocked from the RTC slow clock RTC_SLOW_CLK. The MWDT clock source is derived from the APB clock APB_CLK via a pre-MWDT 16-bit configurable prescaler. For either watchdog, the clock source is fed into the 32-bit expiry counter. When this counter reaches the timeout value of the current stage, the action configured for the stage will execute, the expiry counter will be reset and the next stage will become active.

### 11.3.2 Operating Procedure

When a watchdog timer is enabled, it will proceed in loops from stage 0 to stage 3, then back to stage 0 and start again. The expiry action and time period for each stage can be configured individually.

Every stage can be configured for one of the following actions when the expiry timer reaches the stage's timeout value:

- Trigger an interrupt
  When the stage expires an interrupt is triggered.
- Reset a CPU core
  When the stage expires the designated CPU core will be reset. MWDT0 CPU reset only resets the PRO CPU. MWDT1 CPU reset only resets the APP CPU. The RWDT CPU reset can reset either of them, or both, or none, depending on configuration.
- Reset the main system
  When the stage expires, the main system, including the MWDTs, will be reset. In this article, the main system includes the CPU and all peripherals. The RTC is an exception to this, and it will not be reset.
- Reset the main system and RTC
  When the stage expires the main system and the RTC will both be reset. This action is only available in the RWDT.
- Disabled
  This stage will have no effects on the system.

When software feeds the watchdog timer, it returns to stage 0 and its expiry counter restarts from 0.

### 11.3.3 Write Protection

Both the MWDTs, as well as the RWDT, can be protected from accidental writing. To accomplish this, they have a write-key register (TIMG*n*_WDTWPROTECT_REG for the MWDT, RTC_CNTL_WDTWPROTECT_REG for the RWDT.) On reset, these registers are initialized to the value 0x50D83AA1. When the value in this register is changed from 0x50D83AA1, write protection is enabled. Writes to any WDT register, including the feeding register (but excluding the write-key register itself), are ignored. The recommended procedure for accessing a WDT is:

1. Disable the write protection
2. Make the required modification or feed the watchdog
3. Re-enable the write protection

### 11.3.4 Flash Boot Protection

During flash booting, the MWDT in timer group 0 (TIMG0), as well as the RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon expiry; stage 0 for the RWDT resets the RTC when it expires. After booting, the register TIMERS_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for the MWDT, and RTC_CNTL_WDT_FLASHBOOT_MOD_EN should be cleared to do the same for the RWDT. After this, the MWDT and RWDT can be configured by software.

## 11.4 Registers

The MWDT registers are part of the timer submodule and are described in the Timer Registers section. The RWDT registers are part of the RTC submodule and are described in the RTC Registers section.

# Chapter 12

# DPort Registers

## 12.1 Introduction

The ESP32 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. The DPort registers control clock management (clock gating), power management, and the configuration of peripherals and core-system modules. The system arranges each module with configuration registers contained in the DPort Register.

## 12.2 Features

DPort registers correspond to different peripheral blocks and core modules:

- System and memory
- Reset and clock
- Interrupt matrix
- DMA
- MPU/MMU
- APP_CPU controller
- Peripheral clock gating and reset

## 12.3 Functional Description

### 12.3.1 System and Memory Register

System and memory registers are used for system and memory configuration, such as cache configuration and memory remapping. They are listed in Section 12.4, categorized as "System and memory registers". For a detailed description of these registers, please refer to Chapter *System and Memory*.

### 12.3.2 Reset and Clock Registers

Reset and clock registers are listed in Section 12.4, categorized as "Reset and clock registers". For a detailed description of these registers, please refer to Chapter *Reset and Clock*.

### 12.3.3 Interrupt Matrix Register

The interrupt matrix registers are used for configuring and mapping interrupts through the interrupt matrix. They are listed in Section 12.4, categorized as "Interrupt matrix registers". For a detailed description of these registers, please refer to Chapter *Interrupt Matrix (INTERRUPT)*.

### 12.3.4 DMA Registers

DMA registers are used for the SPI DMA configuration. They are listed in Section 12.4, categorized as "DMA registers". For a detailed description of these registers, please refer to Chapter *DMA Controller (DMA)*.

### 12.3.5 MPU/MMU Registers

MPU/MMU registers are used for MPU/MMU configuration and operation control. They are listed in Section 12.4, categorized as "MPU/MMU registers". For a detailed description of these registers, please refer to Chapter *Memory Management and Protection Units (MMU, MPU)*.

### 12.3.6 APP_CPU Controller Registers

APP_CPU controller registers are used for some basic configuration of the APP_CPU, such as performing a stalling execution, and for configuring the ROM boot jump address. The registers are listed in Section 12.4, categorized as "APP_CPU controller registers". A detailed description of these registers is provided in section 12.5. Note that reset bits are not self-clearing.

### 12.3.7 Peripheral Clock Gating and Reset

The following registers are used for controlling the clock gating and reset of different peripherals. A detailed description of these registers is provided in section 12.5.

- DPORT_PERI_CLK_EN_REG
- DPORT_PERI_RST_EN_REG
- DPORT_PERIP_CLK_EN_REG
- DPORT_PERIP_RST_EN_REG
- DPORT_WIFI_CLK_EN_REG
- DPORT_WIFI_RST_EN_REG

Notice:

- Clock gating and reset registers are active high.
- Reset registers cannot be cleared by hardware. Therefore, SW reset clear is required after setting the reset registers.
- ESP32 features low power consumption. This is why some peripheral clocks are gated (disabled) by default. Before using any of these peripherals, it is mandatory to enable the clock for the given peripheral by setting the corresponding CLK_EN bit to 1, and release the peripheral from reset state to make it operational by setting the RST_EN bit to 0.

## 12.4 Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| System and memory registers | | | |
| DPORT_PRO_BOOT_REMAP_CTRL_REG | remap mode for PRO_CPU | 0x3FF00000 | R/W |
| DPORT_APP_BOOT_REMAP_CTRL_REG | remap mode for APP_CPU | 0x3FF00004 | R/W |
| DPORT_CACHE_MUX_MODE_REG | the mode of the two caches sharing the memory | 0x3FF0007C | R/W |
| Reset and clock registers | | | |
| DPORT_CPU_PER_CONF_REG | Selects CPU clock | 0x3FF0003C | R/W |
| Interrupt matrix registers | | | |
| DPORT_CPU_INTR_FROM_CPU_0_REG | interrupt 0 in both CPUs | 0x3FF000DC | R/W |
| DPORT_CPU_INTR_FROM_CPU_1_REG | interrupt 1 in both CPUs | 0x3FF000E0 | R/W |
| DPORT_CPU_INTR_FROM_CPU_2_REG | interrupt 2 in both CPUs | 0x3FF000E4 | R/W |
| DPORT_CPU_INTR_FROM_CPU_3_REG | interrupt 3 in both CPUs | 0x3FF000E8 | R/W |
| DPORT_PRO_INTR_STATUS_REG_0_REG | PRO_CPU interrupt status 0 | 0x3FF000EC | RO |
| DPORT_PRO_INTR_STATUS_REG_1_REG | PRO_CPU interrupt status 1 | 0x3FF000F0 | RO |
| DPORT_PRO_INTR_STATUS_REG_2_REG | PRO_CPU interrupt status 2 | 0x3FF000F4 | RO |
| DPORT_APP_INTR_STATUS_REG_0_REG | APP_CPU interrupt status 0 | 0x3FF000F8 | RO |
| DPORT_APP_INTR_STATUS_REG_1_REG | APP_CPU interrupt status 1 | 0x3FF000FC | RO |
| DPORT_APP_INTR_STATUS_REG_2_REG | APP_CPU interrupt status 2 | 0x3FF00100 | RO |
| DPORT_PRO_MAC_INTR_MAP_REG | interrupt map | 0x3FF00104 | R/W |
| DPORT_PRO_MAC_NMI_MAP_REG | interrupt map | 0x3FF00108 | R/W |
| DPORT_PRO_BB_INT_MAP_REG | interrupt map | 0x3FF0010C | R/W |
| DPORT_PRO_BT_MAC_INT_MAP_REG | interrupt map | 0x3FF00110 | R/W |
| DPORT_PRO_BT_BB_INT_MAP_REG | interrupt map | 0x3FF00114 | R/W |
| DPORT_PRO_BT_BB_NMI_MAP_REG | interrupt map | 0x3FF00118 | R/W |
| DPORT_PRO_RWBT_IRQ_MAP_REG | interrupt map | 0x3FF0011C | R/W |
| DPORT_PRO_RWBLE_IRQ_MAP_REG | interrupt map | 0x3FF00120 | R/W |
| DPORT_PRO_RWBT_NMI_MAP_REG | interrupt map | 0x3FF00124 | R/W |
| DPORT_PRO_RWBLE_NMI_MAP_REG | interrupt map | 0x3FF00128 | R/W |
| DPORT_PRO_SLC0_INTR_MAP_REG | interrupt map | 0x3FF0012C | R/W |
| DPORT_PRO_SLC1_INTR_MAP_REG | interrupt map | 0x3FF00130 | R/W |
| DPORT_PRO_UHCI0_INTR_MAP_REG | interrupt map | 0x3FF00134 | R/W |
| DPORT_PRO_UHCI1_INTR_MAP_REG | interrupt map | 0x3FF00138 | R/W |
| DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG | interrupt map | 0x3FF0013C | R/W |
| DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00140 | R/W |
| DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00144 | R/W |
| DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00148 | R/W |
| DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG | interrupt map | 0x3FF0014C | R/W |
| DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00150 | R/W |
| DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00154 | R/W |
| DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00158 | R/W |
| DPORT_PRO_GPIO_INTERRUPT_MAP_REG | interrupt map | 0x3FF0015C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG | interrupt map | 0x3FF00160 | R/W |
| DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG | interrupt map | 0x3FF00164 | R/W |
| DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG | interrupt map | 0x3FF00168 | R/W |
| DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG | Interrupt map | 0x3FF0016C | R/W |
| DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG | interrupt map | 0x3FF00170 | R/W |
| DPORT_PRO_SPI_INTR_0_MAP_REG | interrupt map | 0x3FF00174 | R/W |
| DPORT_PRO_SPI_INTR_1_MAP_REG | interrupt map | 0x3FF00178 | R/W |
| DPORT_PRO_SPI_INTR_2_MAP_REG | interrupt map | 0x3FF0017C | R/W |
| DPORT_PRO_SPI_INTR_3_MAP_REG | interrupt map | 0x3FF00180 | R/W |
| DPORT_PRO_I2S0_INT_MAP_REG | interrupt map | 0x3FF00184 | R/W |
| DPORT_PRO_I2S1_INT_MAP_REG | interrupt map | 0x3FF00188 | R/W |
| DPORT_PRO_UART_INTR_MAP_REG | interrupt map | 0x3FF0018C | R/W |
| DPORT_PRO_UART1_INTR_MAP_REG | interrupt map | 0x3FF00190 | R/W |
| DPORT_PRO_UART2_INTR_MAP_REG | interrupt map | 0x3FF00194 | R/W |
| DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG | interrupt map | 0x3FF00198 | R/W |
| DPORT_PRO_EMAC_INT_MAP_REG | interrupt map | 0x3FF0019C | R/W |
| DPORT_PRO_PWM0_INTR_MAP_REG | interrupt map | 0x3FF001A0 | R/W |
| DPORT_PRO_PWM1_INTR_MAP_REG | interrupt map | 0x3FF001A4 | R/W |
| DPORT_PRO_LEDC_INT_MAP_REG | interrupt map | 0x3FF001B0 | R/W |
| DPORT_PRO_EFUSE_INT_MAP_REG | interrupt map | 0x3FF001B4 | R/W |
| DPORT_PRO_TWAI_INT_MAP_REG | interrupt map | 0x3FF001B8 | R/W |
| DPORT_PRO_RTC_CORE_INTR_MAP_REG | interrupt map | 0x3FF001BC | R/W |
| DPORT_PRO_RMT_INTR_MAP_REG | interrupt map | 0x3FF001C0 | R/W |
| DPORT_PRO_PCNT_INTR_MAP_REG | interrupt map | 0x3FF001C4 | R/W |
| DPORT_PRO_I2C_EXT0_INTR_MAP_REG | interrupt map | 0x3FF001C8 | R/W |
| DPORT_PRO_I2C_EXT1_INTR_MAP_REG | interrupt map | 0x3FF001CC | R/W |
| DPORT_PRO_RSA_INTR_MAP_REG | interrupt map | 0x3FF001D0 | R/W |
| DPORT_PRO_SPI1_DMA_INT_MAP_REG | interrupt map | 0x3FF001D4 | R/W |
| DPORT_PRO_SPI2_DMA_INT_MAP_REG | interrupt map | 0x3FF001D8 | R/W |
| DPORT_PRO_SPI3_DMA_INT_MAP_REG | interrupt map | 0x3FF001DC | R/W |
| DPORT_PRO_WDG_INT_MAP_REG | interrupt map | 0x3FF001E0 | R/W |
| DPORT_PRO_TIMER_INT1_MAP_REG | interrupt map | 0x3FF001E4 | R/W |
| DPORT_PRO_TIMER_INT2_MAP_REG | interrupt map | 0x3FF001E8 | R/W |
| DPORT_PRO_TG_T0_EDGE_INT_MAP_REG | interrupt map | 0x3FF001EC | R/W |
| DPORT_PRO_TG_T1_EDGE_INT_MAP_REG | interrupt map | 0x3FF001F0 | R/W |
| DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG | interrupt map | 0x3FF001F4 | R/W |
| DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG | interrupt map | 0x3FF001F8 | R/W |
| DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG | interrupt map | 0x3FF001FC | R/W |
| DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG | interrupt map | 0x3FF00200 | R/W |
| DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG | interrupt map | 0x3FF00204 | R/W |
| DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG | interrupt map | 0x3FF00208 | R/W |
| DPORT_PRO_MMU_IA_INT_MAP_REG | interrupt map | 0x3FF0020C | R/W |
| DPORT_PRO_MPU_IA_INT_MAP_REG | interrupt map | 0x3FF00210 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_PRO_CACHE_IA_INT_MAP_REG | interrupt map | 0x3FF00214 | R/W |
| DPORT_APP_MAC_INTR_MAP_REG | interrupt map | 0x3FF00218 | R/W |
| DPORT_APP_MAC_NMI_MAP_REG | interrupt map | 0x3FF0021C | R/W |
| DPORT_APP_BB_INT_MAP_REG | interrupt map | 0x3FF00220 | R/W |
| DPORT_APP_BT_MAC_INT_MAP_REG | interrupt map | 0x3FF00224 | R/W |
| DPORT_APP_BT_BB_INT_MAP_REG | interrupt map | 0x3FF00228 | R/W |
| DPORT_APP_BT_BB_NMI_MAP_REG | interrupt map | 0x3FF0022C | R/W |
| DPORT_APP_RWBT_IRQ_MAP_REG | interrupt map | 0x3FF00230 | R/W |
| DPORT_APP_RWBLE_IRQ_MAP_REG | interrupt map | 0x3FF00234 | R/W |
| DPORT_APP_RWBT_NMI_MAP_REG | interrupt map | 0x3FF00238 | R/W |
| DPORT_APP_RWBLE_NMI_MAP_REG | interrupt map | 0x3FF0023C | R/W |
| DPORT_APP_SLC0_INTR_MAP_REG | interrupt map | 0x3FF00240 | R/W |
| DPORT_APP_SLC1_INTR_MAP_REG | interrupt map | 0x3FF00244 | R/W |
| DPORT_APP_UHCI0_INTR_MAP_REG | interrupt map | 0x3FF00248 | R/W |
| DPORT_APP_UHCI1_INTR_MAP_REG | interrupt map | 0x3FF0024C | R/W |
| DPORT_APP_TG_T0_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00250 | R/W |
| DPORT_APP_TG_T1_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00254 | R/W |
| DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00258 | R/W |
| DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF0025C | R/W |
| DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00260 | R/W |
| DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00264 | R/W |
| DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF00268 | R/W |
| DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG | interrupt map | 0x3FF0026C | R/W |
| DPORT_APP_GPIO_INTERRUPT_MAP_REG | interrupt map | 0x3FF00270 | R/W |
| DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG | interrupt map | 0x3FF00274 | R/W |
| DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG | interrupt map | 0x3FF00278 | R/W |
| DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG | interrupt map | 0x3FF0027C | R/W |
| DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG | interrupt map | 0x3FF00280 | R/W |
| DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG | interrupt map | 0x3FF00284 | R/W |
| DPORT_APP_SPI_INTR_0_MAP_REG | interrupt map | 0x3FF00288 | R/W |
| DPORT_APP_SPI_INTR_1_MAP_REG | interrupt map | 0x3FF0028C | R/W |
| DPORT_APP_SPI_INTR_2_MAP_REG | interrupt map | 0x3FF00290 | R/W |
| DPORT_APP_SPI_INTR_3_MAP_REG | interrupt map | 0x3FF00294 | R/W |
| DPORT_APP_I2S0_INT_MAP_REG | interrupt map | 0x3FF00298 | R/W |
| DPORT_APP_I2S1_INT_MAP_REG | interrupt map | 0x3FF0029C | R/W |
| DPORT_APP_UART_INTR_MAP_REG | interrupt map | 0x3FF002A0 | R/W |
| DPORT_APP_UART1_INTR_MAP_REG | interrupt map | 0x3FF002A4 | R/W |
| DPORT_APP_UART2_INTR_MAP_REG | interrupt map | 0x3FF002A8 | R/W |
| DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG | interrupt map | 0x3FF002AC | R/W |
| DPORT_APP_EMAC_INT_MAP_REG | interrupt map | 0x3FF002B0 | R/W |
| DPORT_APP_PWM0_INTR_MAP_REG | interrupt map | 0x3FF002B4 | R/W |
| DPORT_APP_PWM1_INTR_MAP_REG | interrupt map | 0x3FF002B8 | R/W |
| DPORT_APP_LEDC_INT_MAP_REG | interrupt map | 0x3FF002C4 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_APP_EFUSE_INT_MAP_REG | interrupt map | 0x3FF002C8 | R/W |
| DPORT_APP_TWAI_INT_MAP_REG | interrupt map | 0x3FF002CC | R/W |
| DPORT_APP_RTC_CORE_INTR_MAP_REG | interrupt map | 0x3FF002D0 | R/W |
| DPORT_APP_RMT_INTR_MAP_REG | interrupt map | 0x3FF002D4 | R/W |
| DPORT_APP_PCNT_INTR_MAP_REG | interrupt map | 0x3FF002D8 | R/W |
| DPORT_APP_I2C_EXT0_INTR_MAP_REG | interrupt map | 0x3FF002DC | R/W |
| DPORT_APP_I2C_EXT1_INTR_MAP_REG | interrupt map | 0x3FF002E0 | R/W |
| DPORT_APP_RSA_INTR_MAP_REG | interrupt map | 0x3FF002E4 | R/W |
| DPORT_APP_SPI1_DMA_INT_MAP_REG | interrupt map | 0x3FF002E8 | R/W |
| DPORT_APP_SPI2_DMA_INT_MAP_REG | interrupt map | 0x3FF002EC | R/W |
| DPORT_APP_SPI3_DMA_INT_MAP_REG | interrupt map | 0x3FF002F0 | R/W |
| DPORT_APP_WDG_INT_MAP_REG | interrupt map | 0x3FF002F4 | R/W |
| DPORT_APP_TIMER_INT1_MAP_REG | interrupt map | 0x3FF002F8 | R/W |
| DPORT_APP_TIMER_INT2_MAP_REG | interrupt map | 0x3FF002FC | R/W |
| DPORT_APP_TG_T0_EDGE_INT_MAP_REG | interrupt map | 0x3FF00300 | R/W |
| DPORT_APP_TG_T1_EDGE_INT_MAP_REG | interrupt map | 0x3FF00304 | R/W |
| DPORT_APP_TG_WDT_EDGE_INT_MAP_REG | interrupt map | 0x3FF00308 | R/W |
| DPORT_APP_TG_LACT_EDGE_INT_MAP_REG | interrupt map | 0x3FF0030C | R/W |
| DPORT_APP_TG1_T0_EDGE_INT_MAP_REG | interrupt map | 0x3FF00310 | R/W |
| DPORT_APP_TG1_T1_EDGE_INT_MAP_REG | interrupt map | 0x3FF00314 | R/W |
| DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG | interrupt map | 0x3FF00318 | R/W |
| DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG | interrupt map | 0x3FF0031C | R/W |
| DPORT_APP_MMU_IA_INT_MAP_REG | interrupt map | 0x3FF00320 | R/W |
| DPORT_APP_MPU_IA_INT_MAP_REG | interrupt map | 0x3FF00324 | R/W |
| DPORT_APP_CACHE_IA_INT_MAP_REG | interrupt map | 0x3FF00328 | R/W |
| **DMA registers** | | | |
| DPORT_SPI_DMA_CHAN_SEL_REG | selects DMA channel for SPI1, SPI2, and SPI3 | 0x3FF005A8 | R/W |
| **MPU/MMU registers** | | | |
| DPORT_PRO_CACHE_CTRL_REG | determines the virtual address mode of the external SRAM | 0x3FF00040 | R/W |
| DPORT_PRO_CACHE_CTRL1_REG | PRO cache MMU configuration | 0x3FF00044 | R/W |
| DPORT_APP_CACHE_CTRL_REG | determines the virtual address mode of the external SRAM | 0x3FF00058 | R/W |
| DPORT_APP_CACHE_CTRL1_REG | APP cache MMU configuration | 0x3FF0005C | R/W |
| DPORT_IMMU_PAGE_MODE_REG | page size in the MMU for the internal SRAM 0 | 0x3FF00080 | R/W |
| DPORT_DMMU_PAGE_MODE_REG | page size in the MMU for the internal SRAM 2 | 0x3FF00084 | R/W |
| DPORT_AHB_MPU_TABLE_0_REG | MPU for configuring DMA | 0x3FF000B4 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_AHB_MPU_TABLE_1_REG | MPU for configuring DMA | 0x3FF000B8 | R/W |
| DPORT_AHBLITE_MPU_TABLE_UART_REG | MPU for peripherals | 0x3FF0032C | R/W |
| DPORT_AHBLITE_MPU_TABLE_SPI1_REG | MPU for peripherals | 0x3FF00330 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SPI0_REG | MPU for peripherals | 0x3FF00334 | R/W |
| DPORT_AHBLITE_MPU_TABLE_GPIO_REG | MPU for peripherals | 0x3FF00338 | R/W |
| DPORT_AHBLITE_MPU_TABLE_RTC_REG | MPU for peripherals | 0x3FF00348 | R/W |
| DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG | MPU for peripherals | 0x3FF0034C | R/W |
| DPORT_AHBLITE_MPU_TABLE_HINF_REG | MPU for peripherals | 0x3FF00354 | R/W |
| DPORT_AHBLITE_MPU_TABLE_UHCI1_REG | MPU for peripherals | 0x3FF00358 | R/W |
| DPORT_AHBLITE_MPU_TABLE_I2S0_REG | MPU for peripherals | 0x3FF00364 | R/W |
| DPORT_AHBLITE_MPU_TABLE_UART1_REG | MPU for peripherals | 0x3FF00368 | R/W |
| DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG | MPU for peripherals | 0x3FF00374 | R/W |
| DPORT_AHBLITE_MPU_TABLE_UHCI0_REG | MPU for peripherals | 0x3FF00378 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG | MPU for peripherals | 0x3FF0037C | R/W |
| DPORT_AHBLITE_MPU_TABLE_RMT_REG | MPU for peripherals | 0x3FF00380 | R/W |
| DPORT_AHBLITE_MPU_TABLE_PCNT_REG | MPU for peripherals | 0x3FF00384 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SLC_REG | MPU for peripherals | 0x3FF00388 | R/W |
| DPORT_AHBLITE_MPU_TABLE_LEDC_REG | MPU for peripherals | 0x3FF0038C | R/W |
| DPORT_AHBLITE_MPU_TABLE_EFUSE_REG | MPU for peripherals | 0x3FF00390 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG | MPU for peripherals | 0x3FF00394 | R/W |
| DPORT_AHBLITE_MPU_TABLE_PWM0_REG | MPU for peripherals | 0x3FF0039C | R/W |
| DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG | MPU for peripherals | 0x3FF003A0 | R/W |
| DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG | MPU for peripherals | 0x3FF003A4 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SPI2_REG | MPU for peripherals | 0x3FF003A8 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SPI3_REG | MPU for peripherals | 0x3FF003AC | R/W |
| DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG | MPU for peripherals | 0x3FF003B0 | R/W |
| DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG | MPU for peripherals | 0x3FF003B4 | R/W |
| DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG | MPU for peripherals | 0x3FF003B8 | R/W |
| DPORT_AHBLITE_MPU_TABLE_EMAC_REG | MPU for peripherals | 0x3FF003BC | R/W |
| DPORT_AHBLITE_MPU_TABLE_PWM1_REG | MPU for peripherals | 0x3FF003C4 | R/W |
| DPORT_AHBLITE_MPU_TABLE_I2S1_REG | MPU for peripherals | 0x3FF003C8 | R/W |
| DPORT_AHBLITE_MPU_TABLE_UART2_REG | MPU for peripherals | 0x3FF003CC | R/W |
| DPORT_AHBLITE_MPU_TABLE_PWR_REG | MPU for peripherals | 0x3FF003E4 | R/W |
| DPORT_IMMU_TABLE0_REG | MMU register 1 for internal SRAM 0 | 0x3FF00504 | R/W |
| DPORT_IMMU_TABLE1_REG | MMU register 1 for internal SRAM 0 | 0x3FF00508 | R/W |
| DPORT_IMMU_TABLE2_REG | MMU register 1 for Internal SRAM 0 | 0x3FF0050C | R/W |
| DPORT_IMMU_TABLE3_REG | MMU register 1 for internal SRAM 0 | 0x3FF00510 | R/W |
| DPORT_IMMU_TABLE4_REG | MMU register 1 for internal SRAM 0 | 0x3FF00514 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_IMMU_TABLE5_REG | MMU register 1 for internal SRAM 0 | 0x3FF00518 | R/W |
| DPORT_IMMU_TABLE6_REG | MMU register 1 for internal SRAM 0 | 0x3FF0051C | R/W |
| DPORT_IMMU_TABLE7_REG | MMU register 1 for internal SRAM 0 | 0x3FF00520 | R/W |
| DPORT_IMMU_TABLE8_REG | MMU register 1 for internal SRAM 0 | 0x3FF00524 | R/W |
| DPORT_IMMU_TABLE9_REG | MMU register 1 for internal SRAM 0 | 0x3FF00528 | R/W |
| DPORT_IMMU_TABLE10_REG | MMU register 1 for internal SRAM 0 | 0x3FF0052C | R/W |
| DPORT_IMMU_TABLE11_REG | MMU register 1 for internal SRAM 0 | 0x3FF00530 | R/W |
| DPORT_IMMU_TABLE12_REG | MMU register 1 for Internal SRAM 0 | 0x3FF00534 | R/W |
| DPORT_IMMU_TABLE13_REG | MMU register 1 for internal SRAM 0 | 0x3FF00538 | R/W |
| DPORT_IMMU_TABLE14_REG | MMU register 1 for internal SRAM 0 | 0x3FF0053C | R/W |
| DPORT_IMMU_TABLE15_REG | MMU register 1 for internal SRAM 0 | 0x3FF00540 | R/W |
| DPORT_DMMU_TABLE0_REG | MMU register 1 for Internal SRAM 2 | 0x3FF00544 | R/W |
| DPORT_DMMU_TABLE1_REG | MMU register 1 for internal SRAM 2 | 0x3FF00548 | R/W |
| DPORT_DMMU_TABLE2_REG | MMU register 1 for internal SRAM 2 | 0x3FF0054C | R/W |
| DPORT_DMMU_TABLE3_REG | MMU register 1 for internal SRAM 2 | 0x3FF00550 | R/W |
| DPORT_DMMU_TABLE4_REG | MMU register 1 for internal SRAM 2 | 0x3FF00554 | R/W |
| DPORT_DMMU_TABLE5_REG | MMU register 1 for internal SRAM 2 | 0x3FF00558 | R/W |
| DPORT_DMMU_TABLE6_REG | MMU register 1 for internal SRAM 2 | 0x3FF0055C | R/W |
| DPORT_DMMU_TABLE7_REG | MMU register 1 for internal SRAM 2 | 0x3FF00560 | R/W |
| DPORT_DMMU_TABLE8_REG | MMU register 1 for internal SRAM 2 | 0x3FF00564 | R/W |
| DPORT_DMMU_TABLE9_REG | MMU register 1 for internal SRAM 2 | 0x3FF00568 | R/W |
| DPORT_DMMU_TABLE10_REG | MMU register 1 for internal SRAM 2 | 0x3FF0056C | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| DPORT_DMMU_TABLE11_REG | MMU register 1 for internal SRAM 2 | 0x3FF00570 | R/W |
| DPORT_DMMU_TABLE12_REG | MMU register 1 for internal SRAM 2 | 0x3FF00574 | R/W |
| DPORT_DMMU_TABLE13_REG | MMU register 1 for internal SRAM 2 | 0x3FF00578 | R/W |
| DPORT_DMMU_TABLE14_REG | MMU register 1 for internal SRAM 2 | 0x3FF0057C | R/W |
| DPORT_DMMU_TABLE15_REG | MMU register 1 for internal SRAM 2 | 0x3FF00580 | R/W |
| **APP_CPU controller registers** | | | |
| DPORT_APPCPU_CTRL_REG_A_REG | reset for APP_CPU | 0x3FF0002C | R/W |
| DPORT_APPCPU_CTRL_REG_B_REG | clock gate for APP_CPU | 0x3FF00030 | R/W |
| DPORT_APPCPU_CTRL_REG_C_REG | stall for APP_CPU | 0x3FF00034 | R/W |
| DPORT_APPCPU_CTRL_REG_D_REG | boot address for APP_CPU | 0x3FF00038 | R/W |
| **Peripheral clock gating and reset registers** | | | |
| DPORT_PERI_CLK_EN_REG | clock gate for peripherals | 0x3FF0001C | R/W |
| DPORT_PERI_RST_EN_REG | reset for peripherals | 0x3FF00020 | R/W |
| DPORT_PERIP_CLK_EN_REG | clock gate for peripherals | 0x3FF000C0 | R/W |
| DPORT_PERIP_RST_EN_REG | reset for peripherals | 0x3FF000C4 | R/W |
| DPORT_WIFI_CLK_EN_REG | clock gate for Wi-Fi | 0x3FF000CC | R/W |
| DPORT_WIFI_RST_EN_REG | reset for Wi-Fi | 0x3FF000D0 | R/W |
| DPORT_BT_LPCK_DIV_FRAC_REG | select LOW_POWER_CLK clock | 0x3FF000D8 | R/W |
| **MMU/MPU Access Exception Registers** | | | |
| DPORT_MEM_ACCESS_DBUG0_REG | SRAM MMU exception flag | 0x3FF003E8 | RO |
| DPORT_MEM_ACCESS_DBUG1_REG | SRAM MPU exception flag | 0x3FF003EC | RO |
| DPORT_PRO_CACHE_DBUG0_REG | PRO CACHE MMU exception flag | 0x3FF003F0 | RO |
| DPORT_APP_CACHE_DBUG0_REG | APP CACHE MMU exception flag | 0x3FF00418 | RO |
| DPORT_MMU_ACCESS_ILLEGAL_INT_EN_REG | SRAM MMU exception interrupt enable register | 0x3FF00598 | R/W |
| DPORT_MPU_ACCESS_ILLEGAL_INT_EN_REG | SRAM MPU exception interrupt enable register | 0x3FF0059C | R/W |
| DPORT_CACHE_ACCESS_ILLEGAL_INT_EN_REG | CACHE MMU exception interrupt enable register | 0x3FF003A0 | R/W |

## 12.5 Registers

The addresses in parenthesis besides register names are the register addresses relative to the DPORT base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 12.4 *Register Summary*.

### Register 12.1. DPORT_PRO_BOOT_REMAP_CTRL_REG (0x000)

| (reserved) | DPORT_PRO_BOOT_REMAP | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**DPORT_PRO_BOOT_REMAP** Remap mode for PRO_CPU. (R/W)

### Register 12.2. DPORT_APP_BOOT_REMAP_CTRL_REG (0x004)

| (reserved) | DPORT_APP_BOOT_REMAP | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**DPORT_APP_BOOT_REMAP** Remap mode for APP_CPU. (R/W)

### Register 12.3. DPORT_PERI_CLK_EN_REG (0x01C)

| (reserved) | DPORT_PERI_EN_RSA | DPORT_PERI_EN_SHA | DPORT_PERI_EN_AES | |
|---|---|---|---|---|
| 31 … 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 | 0 | 0 | 0 | Reset |

**DPORT_PERI_EN_RSA** Set the bit to enable the clock of RSA module. Clear the bit to disable the clock of RSA module. (R/W)

**DPORT_PERI_EN_SHA** Set the bit to enable the clock of SHA module. Clear the bit to disable the clock of SHA module. (R/W)

**DPORT_PERI_EN_AES** Set the bit to enable the clock of AES module. Clear the bit to disable the clock of AES module. (R/W)

### Register 12.4. DPORT_PERI_RST_EN_REG (0x020)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_PERI_RST_RSA | DPORT_PERI_RST_SHA | DPORT_PERI_RST_AES | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**DPORT_PERI_RST_RSA** Set the bit to reset RSA module. Clear the bit to release RSA module. (R/W)

**DPORT_PERI_RST_SHA** Set the bit to reset SHA module. Clear the bit to release SHA module. (R/W)

**DPORT_PERI_RST_AES** Set the bit to reset AES module. Clear the bit to release AES module. (R/W)

### Register 12.5. DPORT_APPCPU_CTRL_REG_A_REG (0x02C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_APPCPU_RESETTING | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | Reset |

**DPORT_APPCPU_RESETTING** Set to 1 to reset APP_CPU. Clear the bit to release APP_CPU. (R/W)

### Register 12.6. DPORT_APPCPU_CTRL_REG_B_REG (0x030)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_APPCPU_CLKGATE_EN | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**DPORT_APPCPU_CLKGATE_EN** Set to 1 to enable the clock of APP_CPU. Clear the bit to disable the clock of APP_CPU. (R/W)

### Register 12.7. DPORT_APPCPU_CTRL_REG_C_REG (0x034)

| 31 | | |

**Register 12.10. DPORT_PRO_CACHE_CTRL_REG (0x040)**

| (reserved) | DPORT_PRO_DRAM_HL | (reserved) | DPORT_PRO_DRAM_SPLIT | DPORT_PRO_SINGLE_IRAM_ENA | (reserved) | DPORT_PRO_CACHE_FLUSH_DONE | DPORT_PRO_CACHE_FLUSH_ENA | DPORT_PRO_CACHE_ENABLE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 17 | 16 | 15 12 | 11 | 10 | 9 6 | 5 | 4 | 3 | 2 0 | |
| 0 0 0 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 0 | Reset |

**DPORT_PRO_DRAM_HL** Determines the virtual address mode of the external SRAM. (R/W)

**DPORT_PRO_DRAM_SPLIT** Determines the virtual address mode of the external SRAM. (R/W)

**DPORT_PRO_SINGLE_IRAM_ENA** Determines a special mode for PRO_CPU access to the external flash. (R/W)

**DPORT_PRO_CACHE_FLUSH_DONE** PRO_CPU cache-flush done. (RO)

**DPORT_PRO_CACHE_FLUSH_ENA** Flushes the PRO_CPU cache. (R/W)

**DPORT_PRO_CACHE_ENABLE** Enables the PRO_CPU cache. (R/W)

**Register 12.11. DPORT_PRO_CACHE_CTRL1_REG (0x044)**

| 31–14 | 13 | 12 | 11–6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | DPORT_PRO_CACHE_MMU_IA_CLR | DPORT_PRO_CMMU_PD | (reserved) | DPORT_PRO_CACHE_MASK_OPSDRAM | DPORT_PRO_CACHE_MASK_DROM0 | DPORT_PRO_CACHE_MASK_DRAM1 | DPORT_PRO_CACHE_MASK_IROM0 | DPORT_PRO_CACHE_MASK_IRAM1 | DPORT_PRO_CACHE_MASK_IRAM0 | |
| 0 0 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 | 1 | 1 | 1 | 1 | 1 | Reset |

**DPORT_PRO_CACHE_MMU_IA_CLR** Clears PRO cache MMU error flag. (R/W)

**DPORT_PRO_CMMU_PD** Disables PRO cache MMU. (R/W)

**DPORT_PRO_CACHE_MASK_OPSDRAM** Disables access from APP_CPU DRAM1 to PRO cache.

1: Disable

0: Enable

(R/W)

**DPORT_PRO_CACHE_MASK_DROM0** Disables access from PRO_CPU DROM0 to PRO cache.

1: Disable

0: Enable

(R/W)

**DPORT_PRO_CACHE_MASK_DRAM1** Disables access from PRO_CPU DRAM1 to PRO cache.

1: Disable

0: Enable

(R/W)

**DPORT_PRO_CACHE_MASK_IROM0** Disables access from PRO_CPU IROM0 to PRO cache.

1: Disable

0: Enable

(R/W)

**DPORT_PRO_CACHE_MASK_IRAM1** Disables access from PRO_CPU IRAM1 to PRO cache.

1: Disable

0: Enable

(R/W)

**DPORT_PRO_CACHE_MASK_IRAM0** Disables access from PRO_CPU IRAM0 to PRO cache.

1: Disable

0: Enable

(R/W)

**Register 12.12. DPORT_APP_CACHE_CTRL_REG (0x058)**

| (reserved) | DPORT_APP_DRAM_HL | (reserved) | DPORT_APP_DRAM_SPLIT | DPORT_APP_SINGLE_IRAM_ENA | (reserved) | DPORT_APP_CACHE_FLUSH_DONE | DPORT_APP_CACHE_FLUSH_ENA | DPORT_APP_CACHE_ENABLE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 15 | 14 | 13 … 12 | 11 | 10 | 9 … 6 | 5 | 4 | 3 | 2 … 0 | |
| 0 0 0 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 0 | Reset |

**DPORT_APP_DRAM_HL** Determines the virtual address mode of the External SRAM. (R/W)

**DPORT_APP_DRAM_SPLIT** Determines the virtual address mode of the External SRAM. (R/W)

**DPORT_APP_SINGLE_IRAM_ENA** Determines a special mode for APP_CPU access to the external flash. (R/W)

**DPORT_APP_CACHE_FLUSH_DONE** APP_CPU cache-flush done. (RO)

**DPORT_APP_CACHE_FLUSH_ENA** Flushes the APP_CPU cache. (R/W)

**DPORT_APP_CACHE_ENABLE** Enables the APP_CPU cache. (R/W)

**Register 12.13. DPORT_APP_CACHE_CTRL1_REG (0x05C)**

| (reserved) | DPORT_APP_CACHE_MMU_IA_CLR | DPORT_APP_CMMU_PD | (reserved) | DPORT_APP_CACHE_MASK_OPSDRAM | DPORT_APP_CACHE_MASK_DROM0 | DPORT_APP_CACHE_MASK_DRAM1 | DPORT_APP_CACHE_MASK_IROM0 | DPORT_APP_CACHE_MASK_IRAM1 | DPORT_APP_CACHE_MASK_IRAM0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 14 | 13 | 12 | 11 … 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 | 1 | 1 | 1 | 1 | 1 | 1 | Reset |

**DPORT_APP_CACHE_MMU_IA_CLR** Clears APP cache MMU error flag. (R/W)

**DPORT_APP_CMMU_PD** Disables APP cache MMU. (R/W)

**DPORT_APP_CACHE_MASK_OPSDRAM** Disables access from PRO_CPU DRAM1 to APP cache.

1: Disable

0: Enable

(R/W)

**DPORT_APP_CACHE_MASK_DROM0** Disables access from APP_CPU DROM0 to APP cache.

1: Disable

0: Enable

(R/W)

**DPORT_APP_CACHE_MASK_DRAM1** Disables access from APP_CPU DRAM1 to APP cache.

1: Disable

0: Enable

(R/W)

**DPORT_APP_CACHE_MASK_IROM0** Disables access from APP_CPU IROM0 to APP cache.

1: Disable

0: Enable

(R/W)

**DPORT_APP_CACHE_MASK_IRAM1** Disables access from APP_CPU IRAM1 to APP cache.

1: Disable

0: Enable

(R/W)

**DPORT_APP_CACHE_MASK_IRAM0** Disables access from APP_CPU IRAM0 to APP cache.

1: Disable

0: Enable

(R/W)

### Register 12.14. DPORT_CACHE_MUX_MODE_REG (0x07C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_CACHE_MUX_MODE | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**DPORT_CACHE_MUX_MODE** The mode of the two caches sharing the memory. (R/W)

### Register 12.15. DPORT_IMMU_PAGE_MODE_REG (0x080)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_IMMU_PAGE_MODE | | (reserved) | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**DPORT_IMMU_PAGE_MODE** Page size in the MMU for the internal SRAM 0. (R/W)

### Register 12.16. DPORT_DMMU_PAGE_MODE_REG (0x084)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_DMMU_PAGE_MODE | | (reserved) | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**DPORT_DMMU_PAGE_MODE** Page size in the MMU for the internal SRAM 2. (R/W)

### Register 12.17. DPORT_AHB_MPU_TABLE_0_REG (0x0B4)

| 31 | | 0 | |
|---|---|---|---|
| 0xFFFFFFFF | | | Reset |

**DPORT_AHB_MPU_TABLE_0_REG** MPU for DMA. (R/W)

**Register 12.18. DPORT_AHB_MPU_TABLE_1_REG (0x0B8)**

| (reserved) | DPORT_AHB_ACCESS_GRANT_1 | |
|---|---|---|
| 31 … 9 | 8 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x1FF | Reset |

**DPORT_AHB_ACCESS_GRANT_1** MPU for DMA. (R/W)

**Register 12.19. DPORT_PERIP_CLK_EN_REG (0x0C0)**

| 31–27 | 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (reserved) | (reserved) | DPORT_UART_MEM_CLK_EN | DPORT_UART2_CLK_EN | DPORT_SPI_DMA_CLK_EN | DPORT_I2S1_CLK_EN | DPORT_PWM1_CLK_EN | DPORT_TWAI_CLK_EN | DPORT_I2C_EXT1_CLK_EN | DPORT_PWM0_CLK_EN | DPORT_SPI3_CLK_EN | DPORT_TIMERGROUP1_CLK_EN | DPORT_EFUSE_CLK_EN | DPORT_TIMERGROUP_CLK_EN | DPORT_UHCI1_CLK_EN | DPORT_LEDC_CLK_EN | DPORT_PCNT_CLK_EN | DPORT_RMT_CLK_EN | DPORT_UHCI0_CLK_EN | DPORT_I2C_EXT0_CLK_EN | DPORT_SPI2_CLK_EN | DPORT_UART1_CLK_EN | (reserved) | DPORT_I2S0_CLK_EN | DPORT_UART_CLK_EN | DPORT_SPI01_CLK_EN | (reserved) | |
| 11111 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | Reset |

Set the following bit to enable the clock of the corresponding module. Clear the bit to disable the clock of the corresponding module.

**DPORT_UART_MEM_CLK_EN** Shared memory of UART0 ~ 2. To use any UART peripherals, enable the clock for UART memory. (R/W)

**DPORT_UART2_CLK_EN** UART2 module. (R/W)

**DPORT_SPI_DMA_CLK_EN** SPI_DMA module. (R/W)

**DPORT_I2S1_CLK_EN** I2S1 module. (R/W)

**DPORT_PWM1_CLK_EN** PWM1 module. (R/W)

**DPORT_TWAI_CLK_EN** TWAI module. (R/W)

**DPORT_I2C_EXT1_CLK_EN** I2C1 module. (R/W)

**DPORT_PWM0_CLK_EN** PWM0 module. (R/W)

**DPORT_SPI3_CLK_EN** SPI3 module. (R/W)

**DPORT_TIMERGROUP1_CLK_EN** TIMG1 module. (R/W)

**DPORT_EFUSE_CLK_EN** eFuse module. (R/W)

**DPORT_TIMERGROUP_CLK_EN** TIMG0 module. (R/W)

**DPORT_UHCI1_CLK_EN** UDMA1 module. (R/W)

**DPORT_LEDC_CLK_EN** LEDC module. (R/W)

**DPORT_PCNT_CLK_EN** PCNT module. (R/W)

**DPORT_RMT_CLK_EN** RMT module. (R/W)

**DPORT_UHCI0_CLK_EN** UDMA0 module. (R/W)

**DPORT_I2C_EXT0_CLK_EN** I2C0 module. (R/W)

**DPORT_SPI2_CLK_EN** SPI2 module. (R/W)

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

### Register 12.19. DPORT_PERIP_CLK_EN_REG (0x0C0)

Continued from the previous page...

**DPORT_UART1_CLK_EN** UART1 module. (R/W)

**DPORT_I2S0_CLK_EN** I2S0 module. (R/W)

**DPORT_UART_CLK_EN** UART0 module. (R/W)

**DPORT_SPI01_CLK_EN** SPI0 and SPI1 module. (R/W)

### Register 12.20. DPORT_PERIP_RST_EN_REG (0x0C4)

| 31–27 | 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (reserved) | (reserved) | DPORT_UART_MEM_RST | DPORT_UART2_RST | DPORT_SPI_DMA_RST | DPORT_I2S1_RST | DPORT_PWM1_RST | DPORT_TWAI_RST | DPORT_I2C_EXT1_RST | DPORT_PWM0_RST | DPORT_SPI3_RST | DPORT_TIMERGROUP1_RST | DPORT_EFUSE_RST | DPORT_TIMERGROUP_RST | DPORT_UHCI1_RST | DPORT_LEDC_RST | DPORT_PCNT_RST | DPORT_RMT_RST | DPORT_UHCI0_RST | DPORT_I2C_EXT0_RST | DPORT_SPI2_RST | DPORT_UART1_RST | (reserved) | DPORT_I2S0_RST | (reserved) | DPORT_UART_RST | DPORT_SPI01_RST | (reserved) | |
| 00000 | 0 | 0 | 0 | 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 |

Set each bit to reset the corresponding module. Clear the bit to release the corresponding module. For the list of modules, please refer to register 12.19.

### Register 12.21. DPORT_WIFI_CLK_EN_REG (0x0CC)

| 31–15 | 14 | 13 | 12–5 | 4 | 3–0 | |
|---|---|---|---|---|---|---|
| (reserved) | DPORT_WIFI_CLK_EMAC_EN | DPORT_WIFI_CLK_SDIO_HOST_EN | (reserved) | DPORT_WIFI_CLK_SDIOSLAVE_EN | (reserved) | |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 | 1 | 1 | 0 0 0 0 0 0 0 1 | 1 | 0 0 0 0 | Reset |

**DPORT_WIFI_CLK_EMAC_EN** Set the bit to enable the clock of Ethernet MAC module. Clear the bit to disable the clock of Ethernet MAC module. (R/W)

**DPORT_WIFI_CLK_SDIO_HOST_EN** Set the bit to enable the clock of SD/MMC module. Clear the bit to disable the clock of SD/MMC module. (R/W)

**DPORT_WIFI_CLK_SDIOSLAVE_EN** Set the bit to enable the clock of SDIO module. Clear the bit to disable the clock of SDIO module. (R/W)

### Register 12.22. DPORT_WIFI_RST_EN_REG (0x0D0)

| (reserved) | DPORT_EMAC_RST | DPORT_SDIO_HOST_RST | DPORT_SDIO_RST | (reserved) | |
|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 … 0 | |
| 0 0 0 0 0 0 0 0 0 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 |

**DPORT_EMAC_RST** Set the bit to reset Ethernet MAC module. Clear the bit to release Ethernet MAC module. (R/W)

**DPORT_SDIO_HOST_RST** Set the bit to reset SD/MMC module. Clear the bit to release SD/MMC module. (R/W)

**DPORT_SDIO_RST** Set the bit to reset SDIO module. Clear the bit to release SDIO module. (R/W)

### Register 12.23. DPORT_BT_LPCK_DIV_FRAC_REG (0x0D8)

| (reserved) | DPORT_LPCLK_SEL_XTL32K | DPORT_LPCLK_SEL_XTAL | DPORT_LPCLK_SEL_8M | DPORT_LPCLK_SEL_RTC_SLOW | DPORT_BT_LPCK_DIV_A | DPORT_BT_LPCK_DIV_B | |
|---|---|---|---|---|---|---|---|
| 31 … 28 | 27 | 26 | 25 | 24 | 23 … 12 | 11 … 0 | |
| 0 0 0 0 | 0 | 0 | 1 | 0 | 0x001 | 0x001 | Reset |

**DPORT_LPCLK_SEL_XTL32K** Selects XTL32K_CLK as the BT low-power clock source. (R/W)

**DPORT_LPCLK_SEL_XTAL** Selects XTL_CLK (XTAL clock) as the BT low-power clock source. (R/W)

**DPORT_LPCLK_SEL_8M** Selects RC_FAST_CLK (8 MHz) as the BT low-power clock source. (R/W)

**DPORT_LPCLK_SEL_RTC_SLOW** Selects RTC_SLOW_CLK as the BT low-power clock source. (R/W)

**DPORT_BT_LPCK_DIV_A** BT low-power clock divider parameter A. (R/W)

**DPORT_BT_LPCK_DIV_B** BT low-power clock divider parameter B. (R/W)

Register 12.24. DPORT_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3) (0xDC+4*n)

| (reserved) | DPORT_CPU_INTR_FROM_CPU_*n* | |
|---|---|---|
| 31 … 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 |

**DPORT_CPU_INTR_FROM_CPU_*n*** Interrupt in both CPUs. (R/W)

Register 12.25. DPORT_PRO_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xEC+4*n)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**DPORT_PRO_INTR_STATUS_REG_*n*_REG** PRO_CPU interrupt status. (RO)

Register 12.26. DPORT_APP_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xF8+4*n)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**DPORT_APP_INTR_STATUS_REG_*n*_REG** APP_CPU interrupt status. (RO)

Register 12.27. DPORT_PRO_MAC_INTR_MAP_REG (0x104)

Register 12.28. DPORT_PRO_MAC_NMI_MAP_REG (0x108)

Register 12.29. DPORT_PRO_BB_INT_MAP_REG (0x10C)

Register 12.30. DPORT_PRO_BT_MAC_INT_MAP_REG (0x110)

Register 12.31. DPORT_PRO_BT_BB_INT_MAP_REG (0x114)

Register 12.32. DPORT_PRO_BT_BB_NMI_MAP_REG (0x118)

Register 12.33. DPORT_PRO_RWBT_IRQ_MAP_REG (0x11C)

Register 12.34. DPORT_PRO_RWBLE_IRQ_MAP_REG (0x120)

Register 12.35. DPORT_PRO_RWBT_NMI_MAP_REG (0x124)

Register 12.36. DPORT_PRO_RWBLE_NMI_MAP_REG (0x128)

Register 12.37. DPORT_PRO_SLC0_INTR_MAP_REG (0x12C)

Register 12.38. DPORT_PRO_SLC1_INTR_MAP_REG (0x130)

Register 12.39. DPORT_PRO_UHCI0_INTR_MAP_REG (0x134)

Register 12.40. DPORT_PRO_UHCI1_INTR_MAP_REG (0x138)

Register 12.41. DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG (0x13C)

Register 12.42. DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG (0x140)

Register 12.43. DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG (0x144)

Register 12.44. DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG (0x148)

Register 12.45. DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG (0x14C)

Register 12.46. DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG (0x150)

Register 12.47. DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG (0x154)

Register 12.48. DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG (0x158)

Register 12.49. DPORT_PRO_GPIO_INTERRUPT_MAP_REG (0x15C)

Register 12.50. DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG (0x160)

Register 12.51. DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG (0x164)

Register 12.52. DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG (0x168)

Register 12.53. DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG (0x16C)

Register 12.54. DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG (0x170)

Register 12.55. DPORT_PRO_SPI_INTR_0_MAP_REG (0x174)

Register 12.56. DPORT_PRO_SPI_INTR_1_MAP_REG (0x178)

Register 12.57. DPORT_PRO_SPI_INTR_2_MAP_REG (0x17C)

Register 12.58. DPORT_PRO_SPI_INTR_3_MAP_REG (0x180)

Register 12.59. DPORT_PRO_I2S0_INT_MAP_REG (0x184)

Register 12.60. DPORT_PRO_I2S1_INT_MAP_REG (0x188)

Register 12.61. DPORT_PRO_UART_INTR_MAP_REG (0x18C)

Register 12.62. DPORT_PRO_UART1_INTR_MAP_REG (0x190)

Register 12.63. DPORT_PRO_UART2_INTR_MAP_REG (0x194)

Register 12.64. DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG (0x198)

Register 12.65. DPORT_PRO_EMAC_INT_MAP_REG (0x19C)

Register 12.66. DPORT_PRO_PWM0_INTR_MAP_REG (0x1A0)

Register 12.67. DPORT_PRO_PWM1_INTR_MAP_REG (0x1A4)

Register 12.68. DPORT_PRO_LEDC_INT_MAP_REG (0x1B0)

Register 12.69. DPORT_PRO_EFUSE_INT_MAP_REG (0x1B4)

Register 12.70. DPORT_PRO_TWAI_INT_MAP_REG (0x1B8)

Register 12.71. DPORT_PRO_RTC_CORE_INTR_MAP_REG (0x1BC)

Register 12.72. DPORT_PRO_RMT_INTR_MAP_REG (0x1C0)

Register 12.73. DPORT_PRO_PCNT_INTR_MAP_REG (0x1C4)

Register 12.74. DPORT_PRO_I2C_EXT0_INTR_MAP_REG (0x1C8)

Register 12.75. DPORT_PRO_I2C_EXT1_INTR_MAP_REG (0x1CC)

Register 12.76. DPORT_PRO_RSA_INTR_MAP_REG (0x1D0)

Register 12.77. DPORT_PRO_SPI1_DMA_INT_MAP_REG (0x1D4)

Register 12.78. DPORT_PRO_SPI2_DMA_INT_MAP_REG (0x1D8)

Register 12.79. DPORT_PRO_SPI3_DMA_INT_MAP_REG (0x1DC)

Register 12.80. DPORT_PRO_WDG_INT_MAP_REG (0x1E0)

Register 12.81. DPORT_PRO_TIMER_INT1_MAP_REG (0x1E4)

Register 12.82. DPORT_PRO_TIMER_INT2_MAP_REG (0x1E8)

Register 12.83. DPORT_PRO_TG_T0_EDGE_INT_MAP_REG (0x1EC)

Register 12.84. DPORT_PRO_TG_T1_EDGE_INT_MAP_REG (0x1F0)

Register 12.85. DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG (0x1F4)

Register 12.86. DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG (0x1F8)

Register 12.87. DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG (0x1FC)

Register 12.88. DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG (0x200)

Register 12.89. DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG (0x204)

Register 12.90. DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG (0x208)

Register 12.91. DPORT_PRO_MMU_IA_INT_MAP_REG (0x20C)

Register 12.92. DPORT_PRO_MPU_IA_INT_MAP_REG (0x210)

Register 12.93. DPORT_PRO_CACHE_IA_INT_MAP_REG (0x214)

| (reserved) | DPORT_PRO_*_MAP | |
|---|---|---|
| 31 … 5 | 4 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 10000 | Reset |

**DPORT_PRO_*_MAP** Interrupt map. (R/W)

Register 12.94. DPORT_APP_MAC_INTR_MAP_REG (0x218)

Register 12.95. DPORT_APP_MAC_NMI_MAP_REG (0x21C)

Register 12.96. DPORT_APP_BB_INT_MAP_REG (0x220)

Register 12.97. DPORT_APP_BT_MAC_INT_MAP_REG (0x224)

Register 12.98. DPORT_APP_BT_BB_INT_MAP_REG (0x228)

Register 12.99. DPORT_APP_BT_BB_NMI_MAP_REG (0x22C)

Register 12.100. DPORT_APP_RWBT_IRQ_MAP_REG (0x230)

Register 12.101. DPORT_APP_RWBLE_IRQ_MAP_REG (0x234)
Register 12.102. DPORT_APP_RWBT_NMI_MAP_REG (0x238)
Register 12.103. DPORT_APP_RWBLE_NMI_MAP_REG (0x23C)
Register 12.104. DPORT_APP_SLC0_INTR_MAP_REG (0x240)
Register 12.105. DPORT_APP_SLC1_INTR_MAP_REG (0x244)
Register 12.106. DPORT_APP_UHCI0_INTR_MAP_REG (0x248)
Register 12.107. DPORT_APP_UHCI1_INTR_MAP_REG (0x24C)
Register 12.108. DPORT_APP_TG_T0_LEVEL_INT_MAP_REG (0x250)
Register 12.109. DPORT_APP_TG_T1_LEVEL_INT_MAP_REG (0x254)
Register 12.110. DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG (0x258)
Register 12.111. DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG (0x25C)
Register 12.112. DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG (0x260)
Register 12.113. DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG (0x264)
Register 12.114. DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG (0x268)
Register 12.115. DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG (0x26C)
Register 12.116. DPORT_APP_GPIO_INTERRUPT_MAP_REG (0x270)
Register 12.117. DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG (0x274)
Register 12.118. DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG (0x278)
Register 12.119. DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG (0x27C)
Register 12.120. DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG (0x280)
Register 12.121. DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG (0x284)
Register 12.122. DPORT_APP_SPI_INTR_0_MAP_REG (0x288)
Register 12.123. DPORT_APP_SPI_INTR_1_MAP_REG (0x28C)
Register 12.124. DPORT_APP_SPI_INTR_2_MAP_REG (0x290)
Register 12.125. DPORT_APP_SPI_INTR_3_MAP_REG (0x294)
Register 12.126. DPORT_APP_I2S0_INT_MAP_REG (0x298)
Register 12.127. DPORT_APP_I2S1_INT_MAP_REG (0x29C)
Register 12.128. DPORT_APP_UART_INTR_MAP_REG (0x2A0)
Register 12.129. DPORT_APP_UART1_INTR_MAP_REG (0x2A4)
Register 12.130. DPORT_APP_UART2_INTR_MAP_REG (0x2A8)
Register 12.131. DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG (0x2AC)
Register 12.132. DPORT_APP_EMAC_INT_MAP_REG (0x2B0)
Register 12.133. DPORT_APP_PWM0_INTR_MAP_REG (0x2B4)
Register 12.134. DPORT_APP_PWM1_INTR_MAP_REG (0x2B8)
Register 12.135. DPORT_APP_LEDC_INT_MAP_REG (0x2C4)

Register 12.136. DPORT_APP_EFUSE_INT_MAP_REG (0x2C8)

Register 12.137. DPORT_APP_TWAI_INT_MAP_REG (0x2CC)

Register 12.138. DPORT_APP_RTC_CORE_INTR_MAP_REG (0x2D0)

Register 12.139. DPORT_APP_RMT_INTR_MAP_REG (0x2D4)

Register 12.140. DPORT_APP_PCNT_INTR_MAP_REG (0x2D8)

Register 12.141. DPORT_APP_I2C_EXT0_INTR_MAP_REG (0x2DC)

Register 12.142. DPORT_APP_I2C_EXT1_INTR_MAP_REG (0x2E0)

Register 12.143. DPORT_APP_RSA_INTR_MAP_REG (0x2E4)

Register 12.144. DPORT_APP_SPI1_DMA_INT_MAP_REG (0x2E8)

Register 12.145. DPORT_APP_SPI2_DMA_INT_MAP_REG (0x2EC)

Register 12.146. DPORT_APP_SPI3_DMA_INT_MAP_REG (0x2F0)

Register 12.147. DPORT_APP_WDG_INT_MAP_REG (0x2F4)

Register 12.148. DPORT_APP_TIMER_INT1_MAP_REG (0x2F8)

Register 12.149. DPORT_APP_TIMER_INT2_MAP_REG (0x2FC)

Register 12.150. DPORT_APP_TG_T0_EDGE_INT_MAP_REG (0x300)

Register 12.151. DPORT_APP_TG_T1_EDGE_INT_MAP_REG (0x304)

Register 12.152. DPORT_APP_TG_WDT_EDGE_INT_MAP_REG (0x308)

Register 12.153. DPORT_APP_TG_LACT_EDGE_INT_MAP_REG (0x30C)

Register 12.154. DPORT_APP_TG1_T0_EDGE_INT_MAP_REG (0x310)

Register 12.155. DPORT_APP_TG1_T1_EDGE_INT_MAP_REG (0x314)

Register 12.156. DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG (0x318)

Register 12.157. DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG (0x31C)

Register 12.158. DPORT_APP_MMU_IA_INT_MAP_REG (0x320)

Register 12.159. DPORT_APP_MPU_IA_INT_MAP_REG (0x324)

Register 12.160. DPORT_APP_CACHE_IA_INT_MAP_REG (0x328)

| (reserved) | DPORT_APP_*_MAP | |
|---|---|---|
| 31 … 5 | 4 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 10000 | Reset |

**DPORT_APP_*_MAP** Interrupt map. (R/W)

Register 12.161. DPORT_AHBLITE_MPU_TABLE_UART_REG (0x32C)

Register 12.162. DPORT_AHBLITE_MPU_TABLE_SPI1_REG (0x330)

Register 12.163. DPORT_AHBLITE_MPU_TABLE_SPIO_REG (0x334)
Register 12.164. DPORT_AHBLITE_MPU_TABLE_GPIO_REG (0x338)
Register 12.165. DPORT_AHBLITE_MPU_TABLE_RTC_REG (0x348)
Register 12.166. DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG (0x34C)
Register 12.167. DPORT_AHBLITE_MPU_TABLE_HINF_REG (0x354)
Register 12.168. DPORT_AHBLITE_MPU_TABLE_UHCI1_REG (0x358)
Register 12.169. DPORT_AHBLITE_MPU_TABLE_I2S0_REG (0x364)
Register 12.170. DPORT_AHBLITE_MPU_TABLE_UART1_REG (0x368)
Register 12.171. DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG (0x374)
Register 12.172. DPORT_AHBLITE_MPU_TABLE_UHCI0_REG (0x378)
Register 12.173. DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG (0x37C)
Register 12.174. DPORT_AHBLITE_MPU_TABLE_RMT_REG (0x380)
Register 12.175. DPORT_AHBLITE_MPU_TABLE_PCNT_REG (0x384)
Register 12.176. DPORT_AHBLITE_MPU_TABLE_SLC_REG (0x388)
Register 12.177. DPORT_AHBLITE_MPU_TABLE_LEDC_REG (0x38C)
Register 12.178. DPORT_AHBLITE_MPU_TABLE_EFUSE_REG (0x390)
Register 12.179. DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG (0x394)
Register 12.180. DPORT_AHBLITE_MPU_TABLE_PWM0_REG (0x39C)
Register 12.181. DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG (0x3A0)
Register 12.182. DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG (0x3A4)
Register 12.183. DPORT_AHBLITE_MPU_TABLE_SPI2_REG (0x3A8)
Register 12.184. DPORT_AHBLITE_MPU_TABLE_SPI3_REG (0x3AC)
Register 12.185. DPORT_AHBLITE_MPU_TABLE_SYSCON_REG (0x3B0)
Register 12.186. DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG (0x3B4)
Register 12.187. DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG (0x3B8)
Register 12.188. DPORT_AHBLITE_MPU_TABLE_EMAC_REG (0x3BC)
Register 12.189. DPORT_AHBLITE_MPU_TABLE_PWM1_REG (0x3C4)
Register 12.190. DPORT_AHBLITE_MPU_TABLE_I2S1_REG (0x3C8)
Register 12.191. DPORT_AHBLITE_MPU_TABLE_UART2_REG (0x3CC)
Register 12.192. DPORT_AHBLITE_MPU_TABLE_PWR_REG (0x3E4)

| (reserved) | DPORT_AHBLITE_*_ACCESS_GRANT_CONFIG | |
|---|---|---|
| 31 … 6 | 5 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**DPORT_AHBLITE_*_ACCESS_GRANT_CONFIG** MPU for peripherals. (R/W)

**Register 12.193. DPORT_MEM_ACCESS_DBUG0_REG (0x3E8)**

| (reserved) | DPORT_MEM_MMU_MULTI_HIT | DPORT_MEM_ACCESS_ILLEGAL | DPORT_MEM_ACCESS_DENY | (reserved) | |
|---|---|---|---|---|---|
| 31 30 | 29 26 | 25 14 | 13 10 | 9 0 | |
| 0 | 0 0 | 0 0 | 0 0 0 0 | 0 | Reset |

**DPORT_MEM_MMU_MULTI_HIT** Indicates configuration errors of SRAM MMU.

Bit 0: APP CPU hits multiple entries when accessing SRAM0

Bit 1: PRO CPU hits multiple entries when accessing SRAM0

Bit 2: APP CPU hits multiple entries when accessing SRAM2

Bit 3: PRO CPU hits multiple entries when accessing SRAM2

(RO)

**DPORT_MEM_ACCESS_ILLEGAL** CPU address overflow when accessing SRAM (related to page size).

Bit 0-1: reserved

Bit 2: APP CPU address overflow when accessing SRAM0

Bit 3: PRO CPU address overflow when accessing SRAM0

Bit 4-7: reserved

Bit 8: APP CPU address overflow when accessing SRAM2

Bit 9: PRO CPU address overflow when accessing SRAM3.

Bit 10-11: reserved

(RO)

**DPORT_MEM_ACCESS_DENY** CPU's access to SRAM is denied.

Bit 0: APP CPU's access to SRAM0 is denied

Bit 1: PRO CPU's access to SRAM0 is denied

Bit 2: APP CPU's access to SRAM2 is denied

Bit 3: PRO CPU's access to SRAM2 is denied

(RO)

Register 12.194. DPORT_MEM_ACCESS_DBUG1_REG (0x3EC)

| (reserved) | DPORT_DMA_ACCESS_DENY | DPORT_ACCESS_PID_ILLEGAL | (reserved) | DPORT_MEM_ACCESS_MISS | |
|---|---|---|---|---|---|
| 31 9 | 8 | 7 6 | 5 4 | 3 0 | |
| 0 | 0 | 0 | 0 | 0 | Reset |

**DPORT_DMA_ACCESS_DENY** DMA's access to SRAM is denied. (RO)

**DPORT_ACCESS_PID_ILLEGAL** CPU's access to the PID controller is illegal.

Bit 0: APP CPU's illegal access to the PID controller

Bit 1: PRO CPU's illegal access to the PID controller

(RO)

**DPORT_MEM_ACCESS_MISS** CPU's access to SRAM is denied.

Bit 0: APP CPU's access to SRAM0 is denied

Bit 1: PRO CPU's access to SRAM0 is denied

Bit 2: APP CPU's access to SRAM2 is denied

Bit 3: PRO CPU's access to SRAM2 is denied

(RO)

Register 12.195. DPORT_PRO_CACHE_DBUG0_REG (0x3F0)

| (reserved) | DPORT_PRO_CACHE_ACCESS_ILLEGAL | DPORT_PRO_CACHE_MMU_ILLEGAL | |
|---|---|---|---|
| 30 6 | 6 1 | 0 | |
| 0 | 0 | 0 | Reset |

**DPORT_PRO_CACHE_ACCESS_ILLEGAL** PRO CPU's illegal access to CACHE address region.

Bit 0: APP CPU's illegal access to $^{L}$VAddr$_{RAM}$ (low-high mode) address region

Bit 1: PRO CPU's illegal access to $VAddr_{RAM}$ address region

Bit 2: PRO CPU's illegal access to $VAddr_3$ address region

Bit 3: PRO CPU's illegal access to $VAddr_2$ address region

Bit 4: PRO CPU's illegal access to $VAddr_1$ address region

Bit 5: PRO CPU's illegal access to $VAddr_4$ address region

(RO)

**DPORT_PRO_CACHE_MMU_ILLEGAL** PRO CPU's access to invalid CACHE entry. (RO)

**Register 12.196. DPORT_APP_CACHE_DBUG0_REG (0x418)**

| (reserved) | DPORT_APP_CACHE_ACCESS_ILLEGAL | DPORT_APP_CACHE_MMU_ILLEGAL | |
|---|---|---|---|
| 30 … 6 | 6 … 1 | 0 | |
| 0 | 0 | 0 | Reset |

**DPORT_APP_CACHE_ACCESS_ILLEGAL** APP CPU's illegal access CACHE address region.

Bit 0: PRO CPU's illegal access $^{R}VAddr_{RAM}$ (low-high mode) address region

Bit 1: APP CPU's illegal access $VAddr_{RAM}$ address region

Bit 2: APP CPU's illegal access $VAddr_3$ address region

Bit 3: APP CPU's illegal access $VAddr_2$ address region

Bit 4: APP CPU's illegal access $VAddr_1$ address region

Bit 5: APP CPU's illegal access $VAddr_4$ address region

(RO)

**DPORT_APP_CACHE_MMU_ILLEGAL** APP CPU's access to invalid CACHE entry. (RO)

**Register 12.197. DPORT_IMMU_TABLE*n*_REG (*n*: 0-15) (0x504+4**n*)**

| (reserved) | DPORT_IMMU_TABLE*n* | |
|---|---|---|
| 31 … 7 | 6 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 15 | Reset |

**DPORT_IMMU_TABLE*n*** Configures Internal SRAM MMU. When *n* is 0 ~ 9, reset value is 0. When *n* is 10 ~ 15, reset values are 10, 11, 12, 13, 14, 15, respectively. (R/W)

**Register 12.198. DPORT_DMMU_TABLE*n*_REG (*n*: 0-15) (0x544+4**n*)**

| (reserved) | DPORT_DMMU_TABLE*n* | |
|---|---|---|
| 31 … 7 | 6 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 15 | Reset |

**DPORT_DMMU_TABLE*n*** Configures Internal SRAM MMU. When *n* is 0 ~ 15, reset values are 0 ~ 15, respectively. (R/W)

**Register 12.199. DPORT_MMU_ACCESS_ILLEGAL_INT_EN_REG (0x598)**

| (reserved) | DPORT_DMA_ACCESS_DENY_INT_EN | DPORT_ACCESS_PID_ILLEGAL_INT_EN | (reserved) | |
|---|---|---|---|---|
| 31 … 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 | Reset |

**DPORT_DMA_ACCESS_DENY_INT_EN** Enables the DMA's access to SRAM denied interrupt. (R/W)

**DPORT_ACCESS_PID_ILLEGAL_INT_EN** Enables the CPU's illegal access to the PID controller interrupt.

Bit 0: Enables the APP CPU's illegal access to the PID controller interrupt

Bit 1: Enables the PRO CPU's illegal access to the PID controller interrupt

(R/W)

**Register 12.200. DPORT_MPU_ACCESS_ILLEGAL_INT_EN_REG (0x59C)**

| (reserved) | DPORT_MEM_MMU_MULTI_HIT_INT_EN | DPORT_MEM_ACCESS_ILLEGAL_INT_EN | DPORT_MEM_ACCESS_DENY_INT_EN | (reserved) | |
|---|---|---|---|---|---|
| 31 … 24 | 23 … 20 | 19 … 8 | 7 … 4 | 3 … 0 | |
| 0 | 0 0 | 0 0 | 0 0 0 0 | 0 | Reset |

**DPORT_MEM_MMU_MULTI_HIT_INT_EN** Enables the SRAM MMU configuration error interrupt.

Bit 0: Enables multiple entry hit interrupt when APP CPU accesses SRAM0

Bit 1: Enables multiple entry hit interrupt when PRO CPU accesses SRAM0

Bit 2: Enables multiple entry hit interrupt when APP CPU accesses SRAM2

Bit 3: Enables multiple entry hit interrupt when PRO CPU accesses SRAM2

(R/W)

**DPORT_MEM_ACCESS_ILLEGAL_INT_EN** Enables CPU address overflow interrupt when accessing SRAM (related to page size).

Bit 0-1: reserved

Bit 2: Enables address overflow interrupt when APP CPU accesses SRAM0

Bit 3: Enables address overflow interrupt when PRO CPU accesses SRAM0

Bit 4-7: reserved

Bit 8: Enables address overflow interrupt when APP CPU accesses SRAM2

Bit 9: Enable address overflow interrupt when PRO CPU accesses SRAM3

Bit 10-11: reserved

(R/W)

**DPORT_MEM_ACCESS_DENY_INT_EN** Enables the CPU's access to SRAM denied interrupt.

Bit 0: Enables denied interrupt when APP CPU accesses SRAM0

Bit 1: Enables denied interrupt when PRO CPU accesses SRAM0

Bit 2: Enables denied interrupt when APP CPU accesses SRAM2

Bit 3: Enable denied interrupt when PRO CPU accesses SRAM2

(R/W)

**Register 12.201. DPORT_CACHE_ACCESS_ILLEGAL_INT_EN_REG (0x3A0)**

| (reserved) | DPORT_PRO_CACHE_ACCESS_ILLEGAL_INT_EN | DPORT_PRO_CACHE_MMU_ILLEGAL_INT_EN | (reserved) | DPORT_APP_CACHE_ACCESS_ILLEGAL_INT_EN | DPORT_APP_CACHE_MMU_ILLEGAL_INT_EN | (reserved) | |
|---|---|---|---|---|---|---|---|
| 31 28 | 27 22 | 21 | 20 14 | 13 8 | 7 | 6 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**DPORT_PRO_CACHE_ACCESS_ILLEGAL_INT_EN** Enables the PRO CPU's illegal access to CACHE address region interrupt.

Bit 0: Enables the APP CPU's illegal access to $^{L}$VAddr$_{RAM}$ (low-high mode) address region interrupt

Bit 1: Enables the PRO CPU's illegal access to $VAddr_{RAM}$ address region interrupt

Bit 2: Enables the PRO CPU's illegal access to $VAddr_3$ address region interrupt

Bit 3: Enables the PRO CPU's illegal access to $VAddr_2$ address region interrupt

Bit 4: Enables the PRO CPU's illegal access to $VAddr_1$ address region interrupt

Bit 5: Enables the PRO CPU's illegal access to $VAddr_4$ address region interrupt

(RO)

**DPORT_PRO_CACHE_MMU_ILLEGAL_INT_EN** Enables the PRO CPU's access to invalid CACHE entry. (RO)

**DPORT_APP_CACHE_ACCESS_ILLEGAL_INT_EN** Enables the APP CPU's illegal access to CACHE address region interrupt.

Bit 0: Enables the PRO CPU's illegal access to $^{R}$VAddr$_{RAM}$ (low-high mode) address region interrupt

Bit 1: Enables the APP CPU's illegal access to $VAddr_{RAM}$ address region interrupt

Bit 2: Enables the APP CPU's illegal access to $VAddr_3$ address region interrupt

Bit 3: Enables the APP CPU's illegal access to $VAddr_2$ address region interrupt

Bit 4: Enables the APP CPU's illegal access to $VAddr_1$ address region interrupt

Bit 5: Enables the APP CPU's illegal access to $VAddr_4$ address region interrupt

(RO)

**DPORT_APP_CACHE_MMU_ILLEGAL_INT_EN** Enables the APP CPU's access to invalid CACHE entry. (RO)

**Register 12.202. DPORT_SPI_DMA_CHAN_SEL_REG (0x5A8)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | 6 | 5 | | 4 | 3 | | 2 | 1 | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | DPORT_SPI_SPI3_DMA_CHAN_SEL | | | DPORT_SPI_SPI2_DMA_CHAN_SEL | | | DPORT_SPI_SPI1_DMA_CHAN_SEL | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**DPORT_SPI_SPI3_DMA_CHAN_SEL** Selects the DMA channel for SPI3. (R/W)

**DPORT_SPI_SPI2_DMA_CHAN_SEL** Selects the DMA channel for SPI2. (R/W)

**DPORT_SPI_SPI1_DMA_CHAN_SEL** Selects the DMA channel for SPI1. (R/W)

# Chapter 13

# Process ID Controller (PID)

## 13.1 Overview

The ESP32 is a dual core device and is capable of running and managing multiple processes. The PID Controller supports switching of PID when a process switch occurs. In addition to PID management, the PID Controller also facilitates management of nested interrupts by recording execution status just before an interrupt service routine is executed. This enables the user application to manage process switches and nested interrupts more efficiently.

## 13.2 Features

The PID Controller features:

- Process management and priority
- Process PID switch
- Interrupt information recording
- Nested interrupt management

## 13.3 Functional Description

Eight processes run on the CPU, and are assigned with PID of 0 ~ 7 respectively. Among the eight processes, processes with PID of 0 or 1 are elevated processes with higher authority compared to processes with PID ranging from 2 ~ 7.

A CPU process switch may occur in two cases:

- An interrupt occurs and the CPU fetches an instruction from the interrupt vector. Instruction fetch or execution from interrupt vector is always treated as a process with PID of 0, irrespective of which process was being executed on the CPU when the interrupt occurred.
- A currently active process explicitly performs a process switch. Only elevated processes with PID of 0 or 1 may perform a process switch.

### 13.3.1 Interrupt Identification

Interrupts are classified into seven priority levels: Level 1, Level 2, Level 3, Level 4, Level 5, Level 6 (Debug), and NMI. Each level of interrupt is assigned an interrupt vector entry address. The PID Controller recognizes CPU instruction fetch from an interrupt vector entry address and automatically switches PID to 0. If CPU only accesses the interrupt vector entry address, PID Controller performs no action.

PIDCTRL_INTERRUPT_ENABLE_REG determines whether the PID Controller identifies and registers an interrupt of certain priority. When a bit of register PIDCTRL_INTERRUPT_ENABLE_REG is 1, PID Controller will take action when CPU fetches instruction from the interrupt vector entry address of the corresponding interrupt. Otherwise, PID Controller performs no action. The registers PIDCTRL_INTERRUPT_ADDR_1_REG ~ PIDCTRL_INTERRUPT_ADDR_7_REG define the interrupt vector entry address for all the interrupt priority levels. For details please refer to Table 13.3-1.

**Table 13.3-1. Interrupt Vector Entry Address**

| Priority level | PIDCTRL_INTERRUPT_ENABLE_REG bit controlling interrupt identification | Interrupt vector entry address |
|---|---|---|
| Level 1 | 1 | PIDCTRL_INTERRUPT_ADDR_1_REG |
| Level 2 | 2 | PIDCTRL_INTERRUPT_ADDR_2_REG |
| Level 3 | 3 | PIDCTRL_INTERRUPT_ADDR_3_REG |
| Level 4 | 4 | PIDCTRL_INTERRUPT_ADDR_4_REG |
| Level 5 | 5 | PIDCTRL_INTERRUPT_ADDR_5_REG |
| Level 6 ( Debug ) | 6 | PIDCTRL_INTERRUPT_ADDR_6_REG |
| NMI | 7 | PIDCTRL_INTERRUPT_ADDR_7_REG |

### 13.3.2 Information Recording

When PID Controller identifies an interrupt, it records three items of information in addition to switching PID to 0. The recorded information includes the priority level of current interrupt, previous interrupt status of the system and the previous process running on the CPU.

PID Controller records the priority level of the current interrupt in register PIDCTRL_LEVEL_REG. For details please refer to Table 13.3-2.

**Table 13.3-2. Configuration of PIDCTRL_LEVEL_REG**

| Value | Priority level of the current interrupt |
|---|---|
| 0 | No interrupt |
| 1 | Level 1 |
| 2 | Level 2 |
| 3 | Level 3 |
| 4 | Level 4 |
| 5 | Level 5 |
| 6 | Level 6 |
| 7 | NMI |

PID Controller also records in register PIDCTRL_FROM_*n*_REG the status of the system before the interrupt occurred. The bit width of register PIDCTRL_FROM_*n*_REG is 7. The highest four bits represent the interrupt status of the system before the interrupt indicated by the register occurred. The lowest three bits represent the process running on the CPU before the interrupt indicated by the register occurred. For details please refer to Table 13.3-3.

Table 13.3-3. Configuration of PIDCTRL_FROM_*n*_REG

| [6:3] | Previous interrupt | [2:0] | Previous process |
|---|---|---|---|
| 0 | No interrupt | 0 | Process with PID of 0 |
| 1 | Level 1 Interrupt | 1 | Process with PID of 1 |
| 2 | Level 2 Interrupt | 2 | Process with PID of 2 |
| 3 | Level 3 Interrupt | 3 | Process with PID of 3 |
| 4 | Level 4 Interrupt | 4 | Process with PID of 4 |
| 5 | Level 5 Interrupt | 5 | Process with PID of 5 |
| 6 | Level 6 Interrupt | 6 | Process with PID of 6 |
| 7 | Level 7 Interrupt | 7 | Process with PID of 7 |

PID Controller possesses registers PIDCTRL_FROM_1_REG ~ PIDCTRL_FROM_7_REG, which correspond to the interrupts of Level 1, Level 2, Level 3, Level 4, Level 5, Level 6 (Debug), and NMI respectively. This enables the system to implement interrupt nesting. Please refer to Table 13.3-1 for examples.

If the configuration of register PIDCTRL_INTERRUPT_ENABLE_REG prevents PID Controller from identifying an interrupt, PID Controller will not record any information, and PIDCTRL_LEVEL_REG and PIDCTRL_FROM_*n*_REG will remain unchanged.

### 13.3.3 Proactive Process Switching

As mentioned before, only an elevated process with PID of 0/1 can initiate a process switch. The new process may have any PID from 0 ~ 7 after the process switch. The key for successful proactive process switching is that when the last command of the current process switches to the first command of the new process, PID should switch from 0/1 to that of the new process.

The software procedure for proactive process switching is as follows:

1. Mask all the interrupts except NMI by using software.
2. Set register PIDCTRL_NMI_MASK_ENABLE_REG to 1 to generate a CPU NMI Interrupt Mask signal.
3. Configure registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG.
4. Configure register PIDCTRL_PID_NEW_REG.
5. Configure register PIDCTRL_LEVEL_REG and PIDCTRL_FROM_*n*_REG.
6. Set register PIDCTRL_PID_CONFIRM_REG and register PIDCTRL_NMI_MASK_DISABLE_REG to 1.
7. Revoke the masking of all interrupts but NMI.
8. Switch to the new process and fetch instruction.

Though we can deal with interrupt nesting, an elevated process should not be interrupted during the process switching, and therefore the interrupts have been masked in step 1 and step 2.

In step 3, the configured values of registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG will affect step 6.

In step 4, the configured value of register PIDCTRL_PID_NEW_REG will be the new PID after step 6.

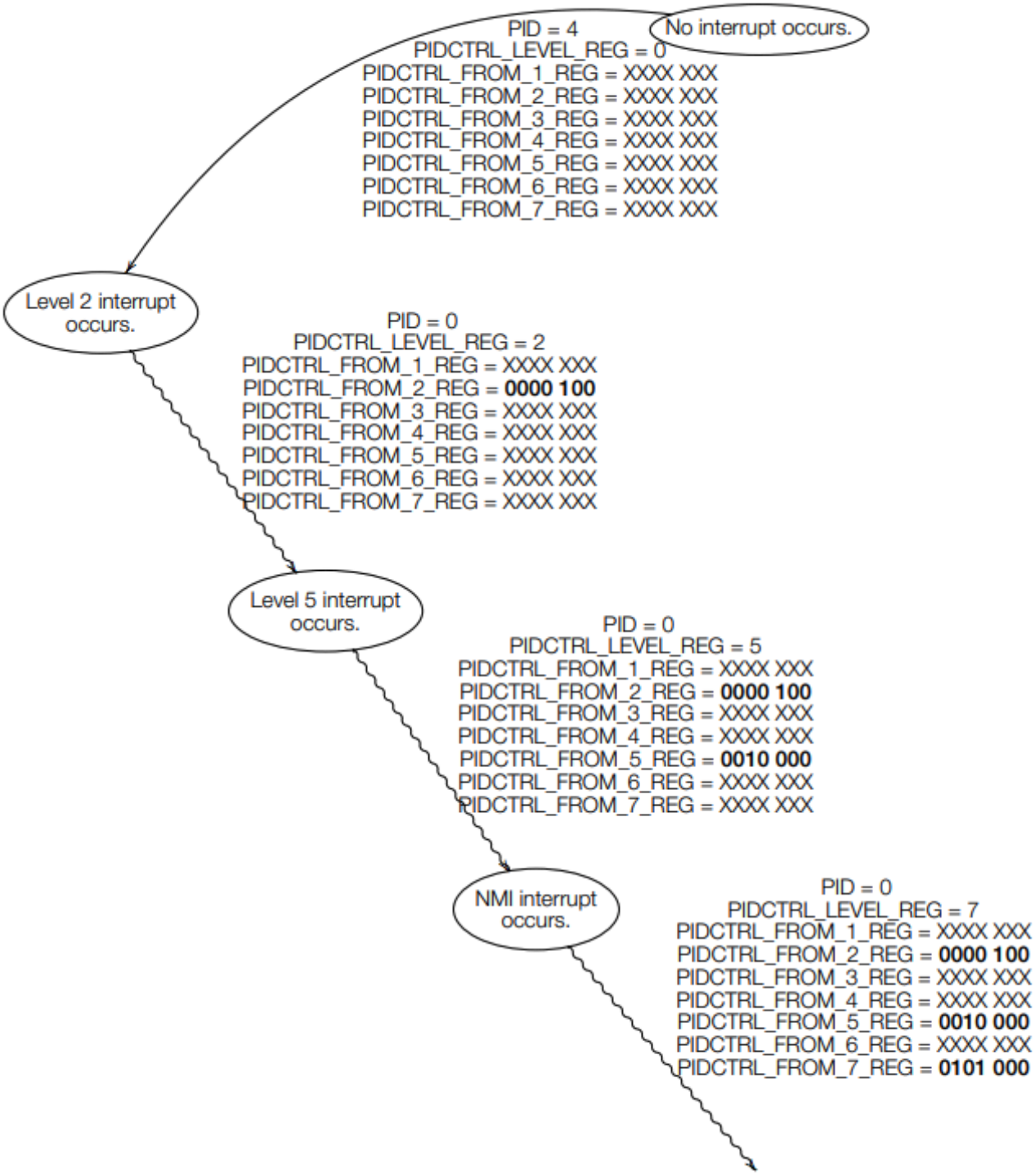



Figure 13.3-1. Interrupt Nesting

If the system is currently in a nested interrupt and needs to revert to the previous interrupt, register PIDCTRL_LEVEL_REG must be restored based on the information recorded in register PIDCTRL_FROM_*n*_REG in step 5.

In step 6, after the values of register PIDCTRL_PID_CONFIRM_REG and register PIDCTRL_NMI_MASK_DISABLE_REG are set to 1, PID Controller will not immediately switch PID to the value of register PIDCTRL_PID_NEW_REG, nor disable CPU NMI Interrupt Mask signal at once. Instead, PID Controller performs each task after a different number of clock cycles. The numbers of clock cycles are the values

specified in register PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG respectively.

In step 7, other tasks can be implemented as well. To do this, the cost of those tasks should be included when configuring registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG in step 3.

## 13.4 Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| PIDCTRL_INTERRUPT_ENABLE_REG | PID interrupt identification enable | 0x3FF1F000 | R/W |
| PIDCTRL_INTERRUPT_ADDR_1_REG | Level 1 interrupt vector address | 0x3FF1F004 | R/W |
| PIDCTRL_INTERRUPT_ADDR_2_REG | Level 2 interrupt vector address | 0x3FF1F008 | R/W |
| PIDCTRL_INTERRUPT_ADDR_3_REG | Level 3 interrupt vector address | 0x3FF1F00C | R/W |
| PIDCTRL_INTERRUPT_ADDR_4_REG | Level 4 interrupt vector address | 0x3FF1F010 | R/W |
| PIDCTRL_INTERRUPT_ADDR_5_REG | Level 5 interrupt vector address | 0x3FF1F014 | R/W |
| PIDCTRL_INTERRUPT_ADDR_6_REG | Level 6 interrupt vector address | 0x3FF1F018 | R/W |
| PIDCTRL_INTERRUPT_ADDR_7_REG | NMI interrupt vector address | 0x3FF1F01C | R/W |
| PIDCTRL_PID_DELAY_REG | New PID valid delay | 0x3FF1F020 | R/W |
| PIDCTRL_NMI_DELAY_REG | NMI mask signal disable delay | 0x3FF1F024 | R/W |
| PIDCTRL_LEVEL_REG | Current interrupt priority | 0x3FF1F028 | R/W |
| PIDCTRL_FROM_1_REG | System status before Level 1 interrupt | 0x3FF1F02C | R/W |
| PIDCTRL_FROM_2_REG | System status before Level 2 interrupt | 0x3FF1F030 | R/W |
| PIDCTRL_FROM_3_REG | System status before Level 3 interrupt | 0x3FF1F034 | R/W |
| PIDCTRL_FROM_4_REG | System status before Level 4 interrupt | 0x3FF1F038 | R/W |
| PIDCTRL_FROM_5_REG | System status before Level 5 interrupt | 0x3FF1F03C | R/W |
| PIDCTRL_FROM_6_REG | System status before Level 6 interrupt | 0x3FF1F040 | R/W |
| PIDCTRL_FROM_7_REG | System status before NMI | 0x3FF1F044 | R/W |
| PIDCTRL_PID_NEW_REG | New PID configuration register | 0x3FF1F048 | R/W |
| PIDCTRL_PID_CONFIRM_REG | New PID confirmation register | 0x3FF1F04C | WO |
| PIDCTRL_NMI_MASK_ENABLE_REG | NMI mask enable register | 0x3FF1F054 | WO |
| PIDCTRL_NMI_MASK_DISABLE_REG | NMI mask disable register | 0x3FF1F058 | WO |

## 13.5 Registers

The addresses in parenthesis besides register names are the register addresses relative to the PID Controller base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 13.4 *Register Summary*.

### Register 13.1. PIDCTRL_INTERRUPT_ENABLE_REG (0x000)

| (reserved) | PIDCTRL_INTERRUPT_ENABLE | (reserved) | |
|---|---|---|---|
| 31 … 8 | 7 … 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 |

**PIDCTRL_INTERRUPT_ENABLE** These bits are used to enable interrupt identification and processing. (R/W)

### Register 13.2. PIDCTRL_INTERRUPT_ADDR_1_REG (0x004)

| 31 … 0 | |
|---|---|
| 0x040000340 | Reset |

**PIDCTRL_INTERRUPT_ADDR_1_REG** Level 1 interrupt vector entry address. (R/W)

### Register 13.3. PIDCTRL_INTERRUPT_ADDR_2_REG (0x008)

| 31 … 0 | |
|---|---|
| 0x040000180 | Reset |

**PIDCTRL_INTERRUPT_ADDR_2_REG** Level 2 interrupt vector entry address. (R/W)

### Register 13.4. PIDCTRL_INTERRUPT_ADDR_3_REG (0x00C)

| 31 … 0 | |
|---|---|
| 0x0400001C0 | Reset |

**PIDCTRL_INTERRUPT_ADDR_3_REG** Level 3 interrupt vector entry address. (R/W)

### Register 13.5. PIDCTRL_INTERRUPT_ADDR_4_REG (0x010)

| 31 … 0 | |
|---|---|
| 0x040000200 | Reset |

**PIDCTRL_INTERRUPT_ADDR_4_REG** Level 4 interrupt vector entry address. (R/W)

### Register 13.6. PIDCTRL_INTERRUPT_ADDR_5_REG (0x014)

| 31 … 0 | |
|---|---|
| 0x040000240 | Reset |

**PIDCTRL_INTERRUPT_ADDR_5_REG** Level 5 interrupt vector entry address. (R/W)

### Register 13.7. PIDCTRL_INTERRUPT_ADDR_6_REG (0x018)

| 31 … 0 | |
|---|---|
| 0x040000280 | Reset |

**PIDCTRL_INTERRUPT_ADDR_6_REG** Level 6 interrupt vector entry address. (R/W)

### Register 13.8. PIDCTRL_INTERRUPT_ADDR_7_REG (0x01C)

| 31 … 0 | |
|---|---|
| 0x0400002C0 | Reset |

**PIDCTRL_INTERRUPT_ADDR_7_REG** NMI interrupt vector entry address. (R/W)

### Register 13.9. PIDCTRL_PID_DELAY_REG (0x020)

| (reserved) 31 … 12 | PIDCTRL_PID_DELAY 11 … 0 | |
|---|---|---|
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 20 | Reset |

**PIDCTRL_PID_DELAY** Delay until newly assigned PID is valid. (R/W)

### Register 13.10. PIDCTRL_NMI_DELAY_REG (0x024)

| (reserved) 31 … 12 | PIDCTRL_NMI_DELAY 11 … 0 | |
|---|---|---|
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 16 | Reset |

**PIDCTRL_NMI_DELAY** Delay for disabling CPU NMI interrupt mask signal. (R/W)

**Register 13.11. PIDCTRL_LEVEL_REG (0x028)**

| (reserved) | | PIDCTRL_CURRENT_STATUS | |
|---|---|---|---|
| 31 | 4 | 3 | 0 |
| 0 0 0 0 0 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 |

**PIDCTRL_CURRENT_STATUS** The current status of the system. (R/W)

**Register 13.12. PIDCTRL_FROM_*n*_REG (*n*: 1-7) (0x28+0x4**n*)**

| (reserved) | | PIDCTRL_PREVIOUS_STATUS_*n* | |
|---|---|---|---|
| 31 | 7 | 6 | 0 |
| 0 0 0 0 0 0 0 0 0 0 0 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 |

**PIDCTRL_PREVIOUS_STATUS_*n*** System status before any of Level 1 to Level 6, NMI interrupts occurs. (R/W)

**Register 13.13. PIDCTRL_PID_NEW_REG (0x048)**

| (reserved) | | PIDCTRL_PID_NEW | |
|---|---|---|---|
| 31 | 3 | 2 | 0 |
| 0 0 0 0 0 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 |

**PIDCTRL_PID_NEW** New PID. (R/W)

Register 13.14. PIDCTRL_PID_CONFIRM_REG (0x04C)

| (reserved) | PIDCTRL_PID_CONFIRM | |
|---|---|---|
| 31 … 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 |

**PIDCTRL_PID_CONFIRM** This bit is used to confirm the switch of PID. (WO)

Register 13.15. PIDCTRL_NMI_MASK_ENABLE_REG (0x054)

| (reserved) | PIDCTRL_NMI_MASK_ENABLE | |
|---|---|---|
| 31 … 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 |

**PIDCTRL_NMI_MASK_ENABLE** This bit is used to enable CPU NMI interrupt mask signal. (WO)

Register 13.16. PIDCTRL_NMI_MASK_DISABLE_REG (0x058)

| (reserved) | PIDCTRL_NMI_MASK_DISABLE | |
|---|---|---|
| 31 … 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 |

**PIDCTRL_NMI_MASK_DISABLE** This bit is used to disable CPU NMI interrupt mask signal. (WO)

# Part IV

# Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like SHA and AES. It also covers random number generation, and encryption/decryption algorithms, showcasing the SoC's capabilities in cryptography and secure data processing.

# Chapter 14

# AES Accelerator (AES)

## 14.1 Introduction

The AES Accelerator speeds up AES operations significantly, compared to AES algorithms implemented solely in software. The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption.

## 14.2 Features

- Supports AES-128 encryption and decryption
- Supports AES-192 encryption and decryption
- Supports AES-256 encryption and decryption
- Supports four variations of key endianness and four variations of text endianness

## 14.3 Functional Description

### 14.3.1 AES Algorithm Operations

The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption. The AES_MODE_REG register can be configured to different values to enable different algorithm operations, as shown in Table 14.3-1.

**Table 14.3-1. Operation Mode**

| AES_MODE_REG[2:0] | Operation |
|---|---|
| 0 | AES-128 Encryption |
| 1 | AES-192 Encryption |
| 2 | AES-256 Encryption |
| 4 | AES-128 Decryption |
| 5 | AES-192 Decryption |
| 6 | AES-256 Decryption |

### 14.3.2 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/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG. For AES-192

encryption/decryption, the 192-bit key is stored in AES_KEY_0_REG ~ AES_KEY_5_REG. For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

Plaintext and ciphertext is stored in the AES_TEXT_*m*_REG registers. There are four 32-bit registers. To enable AES-128/192/256 encryption, initialize the AES_TEXT_*m*_REG registers with plaintext before encryption. When encryption is finished, the AES Accelerator will store back the resulting ciphertext in the AES_TEXT_*m*_REG registers. To enable AES-128/192/256 decryption, initialize the AES_TEXT_*m*_REG registers with ciphertext before decryption. When decryption is finished, the AES Accelerator will store back the resulting plaintext in the AES_TEXT_*m*_REG registers.

## 14.3.3 Endianness

### Key Endianness

Bit 0 and bit 1 in AES_ENDIAN_REG define the key endianness. For detailed information, please see Table 14.3-3, Table 14.3-4 and Table 14.3-5. w[0] ~ w[3] in Table 14.3-3, w[0] ~ w[5] in Table 14.3-4 and w[0] ~ w[7] in Table 14.3-5 are "the first Nk words of the expanded key" as specified in "5.2: Key Expansion" of FIPS PUB 197. "Column Bit" specifies the bytes in the word from w[0] to w[7]. The bytes of AES_KEY_*n*_REG comprise "the first Nk words of the expanded key".

### Text Endianness

Bit 2 and bit 3 in AES_ENDIAN_REG define the endianness of input text, while Bit 4 and Bit 5 define the endianness of output text. The input text refers to the plaintext in AES-128/192/256 encryption and the ciphertext in decryption. The output text refers to the ciphertext in AES-128/192/256 encryption and the plaintext in decryption. For details, please see Table 14.3-2. "State" in Table 14.3-2 is defined as that in "3.4: The State" of FIPS PUB 197: "The AES algorithm operations are performed on a two-dimensional array of bytes called the State". The ciphertext or plaintexts stored in each byte of AES_TEXT_*m*_REG comprise the State.

**Table 14.3-2. AES Text Endianness**

| AES_ENDIAN_REG[3]/[5] | AES_ENDIAN_REG[2]/[4] | Plaintext/Ciphertext | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | State | | c | | | |
| | | | | 0 | 1 | 2 | 3 |
| | | r | 0 | AES_TEXT_3_REG[31:24] | AES_TEXT_2_REG[31:24] | AES_TEXT_1_REG[31:24] | AES_TEXT_0_REG[31:24] |
| | | | 1 | AES_TEXT_3_REG[23:16] | AES_TEXT_2_REG[23:16] | AES_TEXT_1_REG[23:16] | AES_TEXT_0_REG[23:16] |
| | | | 2 | AES_TEXT_3_REG[15:8] | AES_TEXT_2_REG[15:8] | AES_TEXT_1_REG[15:8] | AES_TEXT_0_REG[15:8] |
| | | | 3 | AES_TEXT_3_REG[7:0] | AES_TEXT_2_REG[7:0] | AES_TEXT_1_REG[7:0] | AES_TEXT_0_REG[7:0] |
| 0 | 1 | State | | c | | | |
| | | | | 0 | 1 | 2 | 3 |
| | | r | 0 | AES_TEXT_3_REG[7:0] | AES_TEXT_2_REG[7:0] | AES_TEXT_1_REG[7:0] | AES_TEXT_0_REG[7:0] |
| | | | 1 | AES_TEXT_3_REG[15:8] | AES_TEXT_2_REG[15:8] | AES_TEXT_1_REG[15:8] | AES_TEXT_0_REG[15:8] |
| | | | 2 | AES_TEXT_3_REG[23:16] | AES_TEXT_2_REG[23:16] | AES_TEXT_1_REG[23:16] | AES_TEXT_0_REG[23:16] |
| | | | 3 | AES_TEXT_3_REG[31:24] | AES_TEXT_2_REG[31:24] | AES_TEXT_1_REG[31:24] | AES_TEXT_0_REG[31:24] |
| 1 | 0 | State | | c | | | |
| | | | | 0 | 1 | 2 | 3 |
| | | r | 0 | AES_TEXT_0_REG[31:24] | AES_TEXT_1_REG[31:24] | AES_TEXT_2_REG[31:24] | AES_TEXT_3_REG[31:24] |
| | | | 1 | AES_TEXT_0_REG[23:16] | AES_TEXT_1_REG[23:16] | AES_TEXT_2_REG[23:16] | AES_TEXT_3_REG[23:16] |
| | | | 2 | AES_TEXT_0_REG[15:8] | AES_TEXT_1_REG[15:8] | AES_TEXT_2_REG[15:8] | AES_TEXT_3_REG[15:8] |
| | | | 3 | AES_TEXT_0_REG[7:0] | AES_TEXT_1_REG[7:0] | AES_TEXT_2_REG[7:0] | AES_TEXT_3_REG[7:0] |
| 1 | 1 | State | | c | | | |
| | | | | 0 | 1 | 2 | 3 |
| | | r | 0 | AES_TEXT_0_REG[7:0] | AES_TEXT_1_REG[7:0] | AES_TEXT_2_REG[7:0] | AES_TEXT_3_REG[7:0] |
| | | | 1 | AES_TEXT_0_REG[15:8] | AES_TEXT_1_REG[15:8] | AES_TEXT_2_REG[15:8] | AES_TEXT_3_REG[15:8] |
| | | | 2 | AES_TEXT_0_REG[23:16] | AES_TEXT_1_REG[23:16] | AES_TEXT_2_REG[23:16] | AES_TEXT_3_REG[23:16] |
| | | | 3 | AES_TEXT_0_REG[31:24] | AES_TEXT_1_REG[31:24] | AES_TEXT_2_REG[31:24] | AES_TEXT_3_REG[31:24] |

## Table 14.3-3. AES-128 Key Endianness

| AES_ENDIAN_REG[1] | AES_ENDIAN_REG[0] | Bit | w[0] | w[1] | w[2] | w[3] |
|---|---|---|---|---|---|---|
| 0 | 0 | [31:24] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| | | [23:16] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [15:8] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [7:0] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| 0 | 1 | [31:24] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| | | [23:16] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [15:8] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [7:0] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| 1 | 0 | [31:24] | AES_KEY_0_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_3_REG[31:24] |
| | | [23:16] | AES_KEY_0_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_3_REG[23:16] |
| | | [15:8] | AES_KEY_0_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_3_REG[15:8] |
| | | [7:0] | AES_KEY_0_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_3_REG[7:0] |
| 1 | 1 | [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] |

## Table 14.3-4. AES-192 Key Endianness

| AES_ENDIAN_REG[1] | AES_ENDIAN_REG[0] | Bit | w[0] | w[1] | w[2] | w[3] | w[4] | w[5] |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [31:24] | AES_KEY_5_REG[31:24] | AES_KEY_4_REG[31:24] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| | | [23:16] | AES_KEY_5_REG[23:16] | AES_KEY_4_REG[23:16] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [15:8] | AES_KEY_5_REG[15:8] | AES_KEY_4_REG[15:8] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [7:0] | AES_KEY_5_REG[7:0] | AES_KEY_4_REG[7:0] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| 0 | 1 | [31:24] | AES_KEY_5_REG[7:0] | AES_KEY_4_REG[7:0] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| | | [23:16] | AES_KEY_5_REG[15:8] | AES_KEY_4_REG[15:8] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [15:8] | AES_KEY_5_REG[23:16] | AES_KEY_4_REG[23:16] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [7:0] | AES_KEY_5_REG[31:24] | AES_KEY_4_REG[31:24] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| 1 | 0 | [31:24] | 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] |
| | | [23:16] | 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] |
| | | [15:8] | 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] |
| | | [7:0] | 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] |
| 1 | 1 | [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] |
| | | [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] |
| | | [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] |
| | | [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] |

## Table 14.3-5. AES-256 Key Endianness

| AES_ENDIAN_REG[1] | AES_ENDIAN_REG[0] | Bit | w[0] | w[1] | w[2] | w[3] | w[4] | w[5] | w[6] | w[7] |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [31:24] | AES_KEY_7_REG[31:24] | AES_KEY_6_REG[31:24] | AES_KEY_5_REG[31:24] | AES_KEY_4_REG[31:24] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| | | [23:16] | AES_KEY_7_REG[23:16] | AES_KEY_6_REG[23:16] | AES_KEY_5_REG[23:16] | AES_KEY_4_REG[23:16] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [15:8] | AES_KEY_7_REG[15:8] | AES_KEY_6_REG[15:8] | AES_KEY_5_REG[15:8] | AES_KEY_4_REG[15:8] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [7:0] | AES_KEY_7_REG[7:0] | AES_KEY_6_REG[7:0] | AES_KEY_5_REG[7:0] | AES_KEY_4_REG[7:0] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| 0 | 1 | [31:24] | AES_KEY_7_REG[7:0] | AES_KEY_6_REG[7:0] | AES_KEY_5_REG[7:0] | AES_KEY_4_REG[7:0] | AES_KEY_3_REG[7:0] | AES_KEY_2_REG[7:0] | AES_KEY_1_REG[7:0] | AES_KEY_0_REG[7:0] |
| | | [23:16] | AES_KEY_7_REG[15:8] | AES_KEY_6_REG[15:8] | AES_KEY_5_REG[15:8] | AES_KEY_4_REG[15:8] | AES_KEY_3_REG[15:8] | AES_KEY_2_REG[15:8] | AES_KEY_1_REG[15:8] | AES_KEY_0_REG[15:8] |
| | | [15:8] | AES_KEY_7_REG[23:16] | AES_KEY_6_REG[23:16] | AES_KEY_5_REG[23:16] | AES_KEY_4_REG[23:16] | AES_KEY_3_REG[23:16] | AES_KEY_2_REG[23:16] | AES_KEY_1_REG[23:16] | AES_KEY_0_REG[23:16] |
| | | [7:0] | AES_KEY_7_REG[31:24] | AES_KEY_6_REG[31:24] | AES_KEY_5_REG[31:24] | AES_KEY_4_REG[31:24] | AES_KEY_3_REG[31:24] | AES_KEY_2_REG[31:24] | AES_KEY_1_REG[31:24] | AES_KEY_0_REG[31:24] |
| 1 | 0 | [31:24] | 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] |
| | | [23:16] | 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] |
| | | [15:8] | 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] |
| | | [7:0] | 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] |
| 1 | 1 | [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] |

### 14.3.4 Encryption and Decryption Operations

**Single Operation**

1. Initialize AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_*m*_REG and AES_ENDIAN_REG.
2. Write 1 to AES_START_REG.
3. Wait until AES_IDLE_REG reads 1.
4. Read results from AES_TEXT_*m*_REG.

**Consecutive Operations**

Every time an operation is completed, only AES_TEXT_*m*_REG is modified by the AES Accelerator. Initialization can, therefore, be simplified in a series of consecutive operations.

1. Update contents of AES_MODE_REG, AES_KEY_*n*_REG and AES_ENDIAN_REG, if required.
2. Load AES_TEXT_*m*_REG.
3. Write 1 to AES_START_REG.
4. Wait until AES_IDLE_REG reads 1.
5. Read results from AES_TEXT_*m*_REG.

### 14.3.5 Speed

The AES Accelerator requires 11 to 15 clock cycles to encrypt a message block, and 21 or 22 clock cycles to decrypt a message block.

## 14.4 Register Summary

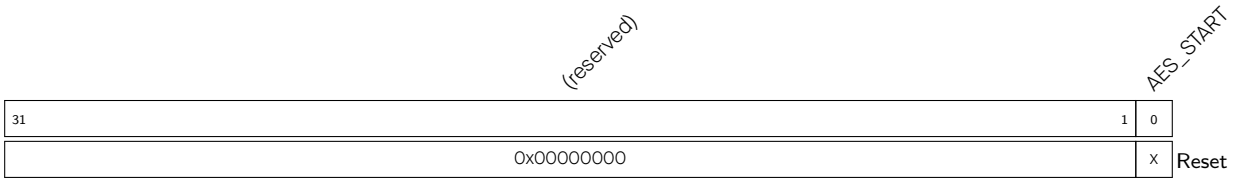
| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| AES_MODE_REG | Mode of operation of the AES Accelerator | 0x3FF01008 | R/W |
| AES_ENDIAN_REG | Endianness configuration register | 0x3FF01040 | R/W |
| **Key registers** | | | |
| AES_KEY_0_REG | AES key material register 0 | 0x3FF01010 | R/W |
| AES_KEY_1_REG | AES key material register 1 | 0x3FF01014 | R/W |
| AES_KEY_2_REG | AES key material register 2 | 0x3FF01018 | R/W |
| AES_KEY_3_REG | AES key material register 3 | 0x3FF0101C | R/W |
| AES_KEY_4_REG | AES key material register 4 | 0x3FF01020 | R/W |
| AES_KEY_5_REG | AES key material register 5 | 0x3FF01024 | R/W |
| AES_KEY_6_REG | AES key material register 6 | 0x3FF01028 | R/W |
| AES_KEY_7_REG | AES key material register 7 | 0x3FF0102C | R/W |
| **Encrypted/decrypted data registers** | | | |
| AES_TEXT_0_REG | AES encrypted/decrypted data register 0 | 0x3FF01030 | R/W |
| AES_TEXT_1_REG | AES encrypted/decrypted data register 1 | 0x3FF01034 | R/W |
| AES_TEXT_2_REG | AES encrypted/decrypted data register 2 | 0x3FF01038 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| AES_TEXT_3_REG | AES encrypted/decrypted data register 3 | 0x3FF0103C | R/W |
| **Control/status registers** | | | |
| AES_START_REG | AES operation start control register | 0x3FF01000 | WO |
| AES_IDLE_REG | AES idle status register | 0x3FF01004 | RO |

## 14.5 Registers

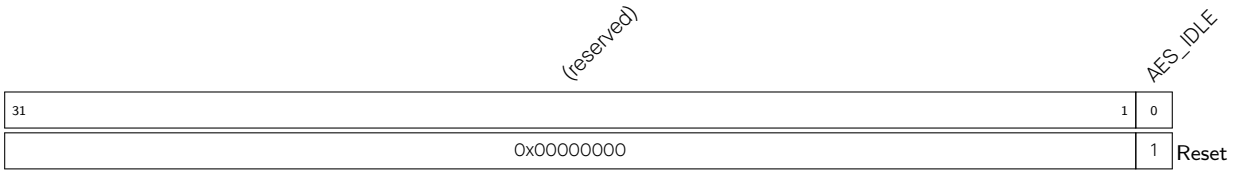
The addresses in parenthesis besides register names are the register addresses relative to the AES base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 14.4 *Register Summary*.

**Register 14.1. AES_START_REG (0x000)**



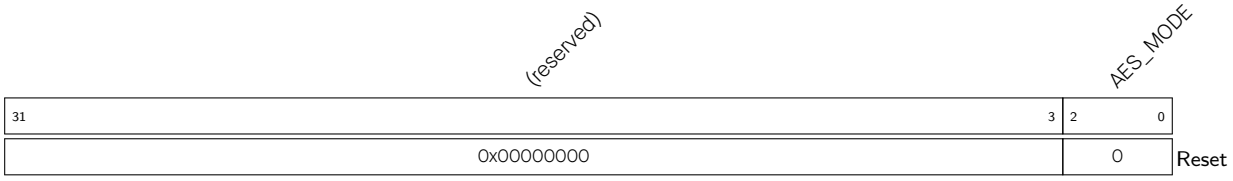

**AES_START** Write 1 to start the AES operation. (WO)

**Register 14.2. AES_IDLE_REG (0x004)**




**AES_IDLE** AES Idle register. Reads 'zero' while the AES Accelerator is busy processing; reads 'one' otherwise. (RO)

**Register 14.3. AES_MODE_REG (0x008)**




**AES_MODE** Selects the AES accelerator mode of operation. See Table 14.3-1 for details. (R/W)

**Register 14.4. AES_KEY_*n*_REG (*n*: 0-7) (0x10+4**n*)**

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**AES_KEY_*n*_REG (*n*: 0-7)** AES key material register. (R/W)

**Register 14.5. AES_TEXT_*m*_REG (*m*: 0-3) (0x30+4**m*)**

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**AES_TEXT_*m*_REG (*m*: 0-3)** Plaintext and ciphertext register. (R/W)

**Register 14.6. AES_ENDIAN_REG (0x040)**

| (reserved) | AES_ENDIAN | | | | | |
|---|---|---|---|---|---|---|
| 31 … 6 | 5 | | | | | 0 |
| 0x0000000 | 1 | 1 | 1 | 1 | 1 | 1 |

Reset

**AES_ENDIAN** Endianness selection register. See Table 14.3-2 for details. (R/W)

# Chapter 15

# RSA Accelerator (RSA)

## 15.1 Introduction

The RSA Accelerator provides hardware support for multiple precision arithmetic operations used in RSA asymmetric cipher algorithms.

Sometimes, multiple precision arithmetic is also called "bignum arithmetic", "bigint arithmetic" or "arbitrary precision arithmetic".

## 15.2 Features

- Support for large-number modular exponentiation
- Support for large-number modular multiplication
- Support for large-number multiplication
- Support for various lengths of operands

## 15.3 Functional Description

### 15.3.1 Initialization

The RSA Accelerator is activated by enabling the corresponding peripheral clock, and by clearing the DPORT_RSA_PD bit in the DPORT_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

When the RSA Accelerator is released from reset, the register RSA_CLEAN_REG reads 0 and an initialization process begins. Hardware initializes the four memory blocks by setting them to 0. After initialization is complete, RSA_CLEAN_REG reads 1. For this reason, software should query RSA_CLEAN_REG after being released from reset, and before writing to any RSA Accelerator memory blocks or registers for the first time.

### 15.3.2 Large Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The operation is based on Montgomery multiplication. Aside from the arguments $X$, $Y$, and $M$, two additional ones are needed — $\overline{r}$ and $M'$. These arguments are calculated in advance by software.

The RSA Accelerator supports operand lengths of $N \in \{512, 1024, 1536, 2048, 2560, 3072, 3584, 4096\}$ bits. The bit length of arguments $Z$, $X$, $Y$, $M$, and $\overline{r}$ can be any one from the N set, but all numbers in a calculation

must be of the same length. The bit length of $M'$ is always 32.

To represent the numbers used as operands, define a base-$b$ positional notation, as follows:

$$b = 2^{32}$$

In this notation, each number is represented by a sequence of base-$b$ digits, where each base-$b$ digit is a 32-bit word. Representing an $N$-bit number requires $n$ base-$b$ digits (all of the possible $N$ lengths are multiples of 32).

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the $n$ values in $Z_{n-1}$ ~ $Z_0$, $X_{n-1}$ ~ $X_0$, $Y_{n-1}$ ~ $Y_0$, $M_{n-1}$ ~ $M_0$, $\overline{r}_{n-1}$ ~ $\overline{r}_0$ 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

$$R = b^n$$

then, we can calculate the additional arguments, as follows:

$$\overline{r} = R^2 \bmod M \tag{15.1}$$

$$\begin{cases} M'' \times M + 1 = R \times R^{-1} \\ M' = M'' \bmod b \end{cases} \tag{15.2}$$

(Equation 15.2 is written in a form suitable for calculations using the extended binary GCD algorithm.)

Software can implement large-number modular exponentiations in the following order:

1. Write $(\frac{N}{512} - 1)$ to RSA_MODEXP_MODE_REG.
2. Write $X_i$, $Y_i$, $M_i$ and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) 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 are ignored.
3. Write $M'$ to RSA_M_PRIME_REG.
4. Write 1 to RSA_MODEXP_START_REG.
5. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.

6. Read the result $Z_i$ ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
7. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MODEXP_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG will not have changed. However, $X_i$ in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM will have been overwritten. In order to perform another operation, refresh the registers and memory blocks, as required.

### 15.3.3 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This operation is based on Montgomery multiplication. The same values $\overline{r}$ and $M'$ are derived by software using the formulas 15.1 and 15.2 shown above.

The RSA Accelerator supports large-number modular multiplication with eight different operand lengths, which are the same as in the large-number modular exponentiation. The operation is performed by a combination of software and hardware. The software performs two hardware operations in sequence.

The software process is as follows:

1. Write $(\frac{N}{512} - 1)$ to RSA_MULT_MODE_REG.
2. Write $X_i$, $M_i$ and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) to registers RSA_X_MEM, RSA_M_MEM and RSA_Z_MEM. Write data to each memory block only according to the length of the number. Data beyond this length are ignored.
3. Write $M'$ to RSA_M_PRIME_REG.
4. Write 1 to RSA_MULT_START_REG.
5. Wait for the first round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.
7. Write $Y_i$ ($i \in [0, n) \cap \mathbb{N}$) to RSA_X_MEM.

   Users need to write to the memory block only according to the length of the number. Data beyond this length are ignored.
8. Write 1 to RSA_MULT_START_REG.
9. Wait for the second round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
10. Read the result $Z_i$ ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
11. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MULT_MODE_REG register, and memory blocks RSA_M_MEM and RSA_M_PRIME_REG remain unchanged. Users do not need to refresh these registers or memory blocks if the values remain the same.

### 15.3.4 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of $Z$ is twice that of $X$ and $Y$. Therefore, the RSA Accelerator supports large-number multiplication with only four operand lengths of

$N \in \{512, 1024, 1536, 2048\}$ bits. The length $\hat{N}$ of the result $Z$ is $2 \times N$ bits.

Operands $X$ and $Y$ need to be extended to form arguments $\hat{X}$ and $\hat{Y}$ which have the same length ($\hat{N}$ bits) as the result $Z$. $X$ is left-extended and $Y$ is right-extended, and defined as follows:

$$n = \frac{N}{32}$$

$$\hat{N} = 2 \times N$$

$$\hat{n} = \frac{\hat{N}}{32} = 2n$$

$$\hat{X} = (\hat{X}_{\hat{n}-1}\hat{X}_{\hat{n}-2}\cdots\hat{X}_0)_b = (\underbrace{00\cdots0}_{n}X)_b = (\underbrace{00\cdots0}_{n}X_{n-1}X_{n-2}\cdots X_0)_b$$

$$\hat{Y} = (\hat{Y}_{\hat{n}-1}\hat{Y}_{\hat{n}-2}\cdots\hat{Y}_0)_b = (Y\underbrace{00\cdots0}_{n})_b = (Y_{n-1}Y_{n-2}\cdots Y_0\underbrace{00\cdots0}_{n})_b$$

Software performs the operation in the following order:

1. Write $(\frac{\hat{N}}{512} - 1 + 8)$ to RSA_MULT_MODE_REG.
2. Write $\hat{X}_i$ and $\hat{Y}_i$ $(i \in [0, \hat{n}) \cap \mathbb{N})$ to RSA_X_MEM and RSA_Z_MEM, respectively.

   Write the valid data into each number's memory block, according to their lengths. Values beyond this length are ignored. Half of the base-$b$ positional notations written to the memory are zero (using the derivations shown above). These zero values are indispensable.
3. Write 1 to RSA_MULT_START_REG.
4. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
5. Read the result $Z_i$ $(i \in [0, \hat{n}) \cap \mathbb{N})$ from RSA_Z_MEM.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, only the RSA_MULT_MODE_REG register remains unmodified.

## 15.4 Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| RSA_M_PRIME_REG | Register to store M' | 0x3FF02800 | R/W |
| **Modular exponentiation registers** | | | |
| RSA_MODEXP_MODE_REG | Modular exponentiation mode | 0x3FF02804 | R/W |
| RSA_MODEXP_START_REG | Start bit | 0x3FF02808 | WO |
| **Modular multiplication registers** | | | |
| RSA_MULT_MODE_REG | Modular multiplication mode | 0x3FF0280C | R/W |
| RSA_MULT_START_REG | Start bit | 0x3FF02810 | WO |
| **Misc registers** | | | |
| RSA_INTERRUPT_REG | RSA interrupt register | 0x3FF02814 | R/W |
| RSA_CLEAN_REG | RSA clean register | 0x3FF02818 | RO |

## 15.5 Registers

The addresses in parenthesis besides register names are the register addresses relative to the RSA base address provided in Table 3.3-6 *Peripheral Address Mapping* in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 15.4 *Register Summary*.

**Register 15.1. RSA_M_PRIME_REG (0x800)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RSA_M_PRIME_REG** This register contains M'. (R/W)

**Register 15.2. RSA_MODEXP_MODE_REG (0x804)**

| (reserved) | RSA_MODEXP_MODE | |
|---|---|---|
| 31 … 3 | 2 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 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 |

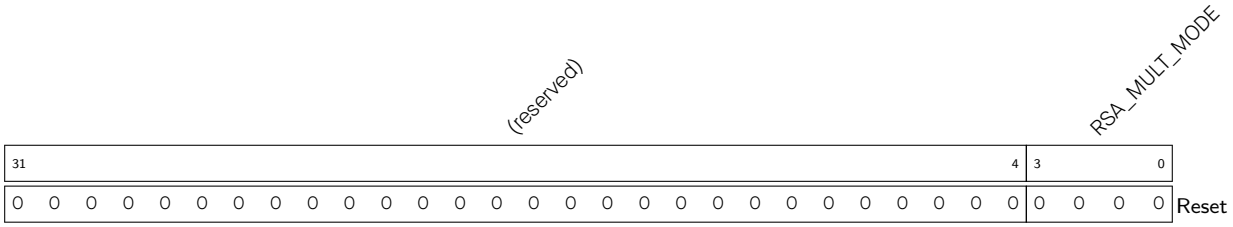
**RSA_MODEXP_MODE** This register contains the mode of modular exponentiation. (R/W)

**Register 15.3. RSA_MODEXP_START_REG (0x808)**

| (reserved) | RSA_MODEXP_START | |
|---|---|---|
| 31 … 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 | 0 | Reset |

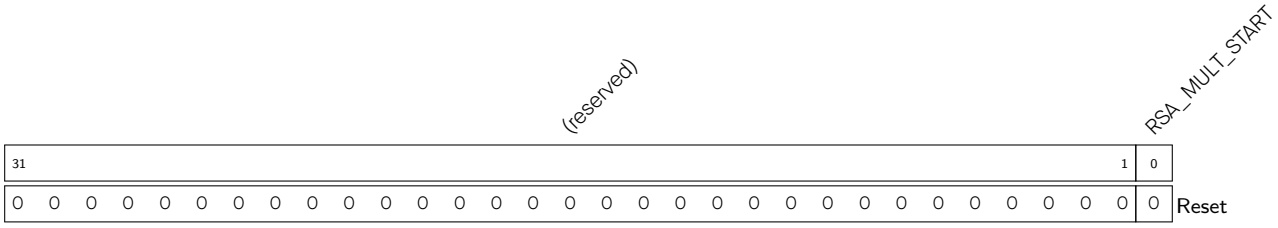
**RSA_MODEXP_START** Write 1 to start modular exponentiation. (WO)

### Register 15.4. RSA_MULT_MODE_REG (0x80C)



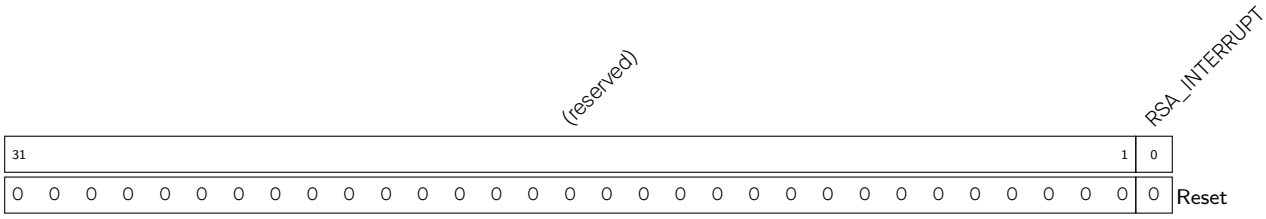

**RSA_MULT_MODE** This register contains the mode of modular multiplication and multiplication. (R/W)

### Register 15.5. RSA_MULT_START_REG (0x810)



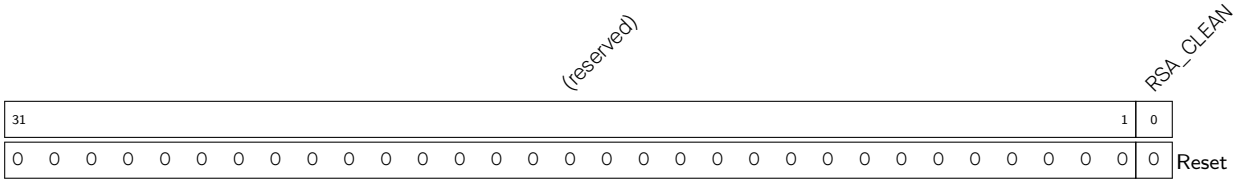

**RSA_MULT_START** Write 1 to start modular multiplication or multiplication. (WO)

### Register 15.6. RSA_INTERRUPT_REG (0x814)




**RSA_INTERRUPT** RSA interrupt status register. Will read 1 once an operation has completed. (R/W)

### Register 15.7. RSA_CLEAN_REG (0x818)




**RSA_CLEAN** This bit will read 1 once the memory initialization is completed. (RO)

# Chapter 16

# SHA Accelerator (SHA)

## 16.1 Introduction

The SHA Accelerator is included to speed up SHA hashing operations significantly, compared to SHA hashing algorithms implemented solely in software. The SHA Accelerator supports four algorithms of FIPS PUB 180-4, specifically SHA-1, SHA-256, SHA-384 and SHA-512.

## 16.2 Features

Hardware support for popular secure hashing algorithms:

- SHA-1
- SHA-256
- SHA-384
- SHA-512

## 16.3 Functional Description

### 16.3.1 Padding and Parsing the Message

The SHA Accelerator can only accept one message block at a time. Software divides the message into blocks according to "5.2 Parsing the Message" in FIPS PUB 180-4 and writes one block to the SHA_TEXT_*n*_REG registers each time. For SHA-1 and SHA-256, software writes a 512-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_15_REG each time. For SHA-384 and SHA-512, software writes a 1024-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_31_REG each time.

The SHA Accelerator is unable to perform the padding operation of "5.1 Padding the Message" in FIPS PUB 180-4; Note that the user software is expected to pad the message before feeding it into the accelerator.

As described in "2.2.1: Parameters" in FIPS PUB 180-4, "$M_0^{(i)}$ is the leftmost word of message block i". $M_0^{(i)}$ is stored in SHA_TEXT_0_REG. In the same fashion, the SHA_TEXT_1_REG register stores the second left-most word of a message block $M_1^{(N)}$, etc.

### 16.3.2 Message Digest

When the hashing operation is finished, the message digest will be refreshed by SHA Accelerator and will be stored in SHA_TEXT_*n*_REG. SHA-1 produces a 160-bit message digest and stores it in SHA_TEXT_0_REG ~

SHA_TEXT_4_REG. SHA-256 produces a 256-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_7_REG. SHA-384 produces a 384-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_11_REG. SHA-512 produces a 512-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_15_REG.

As described in "2.2.1 Parameters" in FIPS PUB 180-4, "$H^{(N)}$ is the final hash value, and is used to determine the message digest", while "$H_0^{(i)}$ is the leftmost word of hash value i", so the leftmost word $H_0^{(N)}$ in the message digest is stored in SHA_TEXT_0_REG. In the same fashion, the second leftmost word $H_1^{(N)}$ in the message digest is stored in SHA_TEXT_1_REG, etc.

### 16.3.3 Hash Operation

There is a set of control registers for SHA-1, SHA-256, SHA-384 and SHA-512, respectively; different hashing algorithms use different control registers.
SHA-1 uses SHA_SHA1_START_REG, SHA_SHA1_CONTINUE_REG, SHA_SHA1_LOAD_REG and SHA_SHA1_BUSY_REG.
SHA-256 uses SHA_SHA256_START_REG, SHA_SHA256_CONTINUE_REG, SHA_SHA256_LOAD_REG and SHA_SHA256_BUSY_REG. SHA-384 uses SHA_SHA384_START_REG, SHA_SHA384_CONTINUE_REG, SHA_SHA384_LOAD_REG and SHA_SHA384_BUSY_REG.
SHA-512 uses SHA_SHA512_START_REG, SHA_SHA512_CONTINUE_REG, SHA_SHA512_LOAD_REG and SHA_SHA512_BUSY_REG. The following steps describe the operation in a detailed manner.

1. Feed the accelerator with the first message block:
   (a) Use the first message block to initialize SHA_TEXT_*n*_REG.
   (b) Write 1 to SHA_*X*_START_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
2. Similarly, feed the accelerator with subsequent message blocks:
   (a) Initialize SHA_TEXT_*n*_REG using the subsequent message block.
   (b) Write 1 to SHA_*X*_CONTINUE_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
3. Get message digest:
   (a) Write 1 to SHA_*X*_LOAD_REG.
   (b) Wait for SHA_*X*_BUSY_REG to read 0, indicating that operation is completed.
   (c) Read message digest from SHA_TEXT_*n*_REG.

### 16.3.4 Speed

The SHA Accelerator requires 60 to 100 clock cycles to process a message block and 8 to 20 clock cycles to calculate the final digest.

## 16.4 Register Summary

The addresses in this section are relative to the SHA Accelerator base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| **Encrypted/decrypted data registers** | | | |
| SHA_TEXT_0_REG | SHA encrypted/decrypted data register 0 | 0x3FF03000 | R/W |
| SHA_TEXT_1_REG | SHA encrypted/decrypted data register 1 | 0x3FF03004 | R/W |
| SHA_TEXT_2_REG | SHA encrypted/decrypted data register 2 | 0x3FF03008 | R/W |
| SHA_TEXT_3_REG | SHA encrypted/decrypted data register 3 | 0x3FF0300C | R/W |
| SHA_TEXT_4_REG | SHA encrypted/decrypted data register 4 | 0x3FF03010 | R/W |
| SHA_TEXT_5_REG | SHA encrypted/decrypted data register 5 | 0x3FF03014 | R/W |
| SHA_TEXT_6_REG | SHA encrypted/decrypted data register 6 | 0x3FF03018 | R/W |
| SHA_TEXT_7_REG | SHA encrypted/decrypted data register 7 | 0x3FF0301C | R/W |
| SHA_TEXT_8_REG | SHA encrypted/decrypted data register 8 | 0x3FF03020 | R/W |
| SHA_TEXT_9_REG | SHA encrypted/decrypted data register 9 | 0x3FF03024 | R/W |
| SHA_TEXT_10_REG | SHA encrypted/decrypted data register 10 | 0x3FF03028 | R/W |
| SHA_TEXT_11_REG | SHA encrypted/decrypted data register 11 | 0x3FF0302C | R/W |
| SHA_TEXT_12_REG | SHA encrypted/decrypted data register 12 | 0x3FF03030 | R/W |
| SHA_TEXT_13_REG | SHA encrypted/decrypted data register 13 | 0x3FF03034 | R/W |
| SHA_TEXT_14_REG | SHA encrypted/decrypted data register 14 | 0x3FF03038 | R/W |
| SHA_TEXT_15_REG | SHA encrypted/decrypted data register 15 | 0x3FF0303C | R/W |
| SHA_TEXT_16_REG | SHA encrypted/decrypted data register 16 | 0x3FF03040 | R/W |
| SHA_TEXT_17_REG | SHA encrypted/decrypted data register 17 | 0x3FF03044 | R/W |
| SHA_TEXT_18_REG | SHA encrypted/decrypted data register 18 | 0x3FF03048 | R/W |
| SHA_TEXT_19_REG | SHA encrypted/decrypted data register 19 | 0x3FF0304C | R/W |
| SHA_TEXT_20_REG | SHA encrypted/decrypted data register 20 | 0x3FF03050 | R/W |
| SHA_TEXT_21_REG | SHA encrypted/decrypted data register 21 | 0x3FF03054 | R/W |
| SHA_TEXT_22_REG | SHA encrypted/decrypted data register 22 | 0x3FF03058 | R/W |
| SHA_TEXT_23_REG | SHA encrypted/decrypted data register 23 | 0x3FF0305C | R/W |
| SHA_TEXT_24_REG | SHA encrypted/decrypted data register 24 | 0x3FF03060 | R/W |
| SHA_TEXT_25_REG | SHA encrypted/decrypted data register 25 | 0x3FF03064 | R/W |
| SHA_TEXT_26_REG | SHA encrypted/decrypted data register 26 | 0x3FF03068 | R/W |
| SHA_TEXT_27_REG | SHA encrypted/decrypted data register 27 | 0x3FF0306C | R/W |
| SHA_TEXT_28_REG | SHA encrypted/decrypted data register 28 | 0x3FF03070 | R/W |
| SHA_TEXT_29_REG | SHA encrypted/decrypted data register 29 | 0x3FF03074 | R/W |
| SHA_TEXT_30_REG | SHA encrypted/decrypted data register 30 | 0x3FF03078 | R/W |
| SHA_TEXT_31_REG | SHA encrypted/decrypted data register 31 | 0x3FF0307C | R/W |
| **Control/status registers** | | | |
| SHA_SHA1_START_REG | Control register to initiate SHA1 operation | 0x3FF03080 | WO |
| SHA_SHA1_CONTINUE_REG | Control register to continue SHA1 operation | 0x3FF03084 | WO |

| Name | Description | Address | Access |
|---|---|---|---|
| SHA_SHA1_LOAD_REG | Control register to calculate the final SHA1 hash | 0x3FF03088 | WO |
| SHA_SHA1_BUSY_REG | Status register for SHA1 operation | 0x3FF0308C | RO |
| SHA_SHA256_START_REG | Control register to initiate SHA256 operation | 0x3FF03090 | WO |
| SHA_SHA256_CONTINUE_REG | Control register to continue SHA256 operation | 0x3FF03094 | WO |
| SHA_SHA256_LOAD_REG | Control register to calculate the final SHA256 hash | 0x3FF03098 | WO |
| SHA_SHA256_BUSY_REG | Status register for SHA256 operation | 0x3FF0309C | RO |
| SHA_SHA384_START_REG | Control register to initiate SHA384 operation | 0x3FF030A0 | WO |
| SHA_SHA384_CONTINUE_REG | Control register to continue SHA384 operation | 0x3FF030A4 | WO |
| SHA_SHA384_LOAD_REG | Control register to calculate the final SHA384 hash | 0x3FF030A8 | WO |
| SHA_SHA384_BUSY_REG | Status register for SHA384 operation | 0x3FF030AC | RO |
| SHA_SHA512_START_REG | Control register to initiate SHA512 operation | 0x3FF030B0 | WO |
| SHA_SHA512_CONTINUE_REG | Control register to continue SHA512 operation | 0x3FF030B4 | WO |
| SHA_SHA512_LOAD_REG | Control register to calculate the final SHA512 hash | 0x3FF030B8 | WO |
| SHA_SHA512_BUSY_REG | Status register for SHA512 operation | 0x3FF030BC | RO |

## 16.5 Registers

The addresses in this section are relative to the SHA Accelerator base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

**Register 16.1. SHA_TEXT_*n*_REG (*n*: 0-31) (0x0+4**n*)**

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**SHA_TEXT_*n*_REG (*n*: 0-31)** SHA Message block and hash result register. (R/W)

**Register 16.2. SHA_SHA1_START_REG (0x080)**

| (reserved) | SHA_SHA1_START | |
|---|---|---|
| 31 1 | 0 | |
| 0x00000000 | 0 | Reset |

**SHA_SHA1_START** Write 1 to start an SHA-1 operation on the first message block. (WO)

**Register 16.3. SHA_SHA1_CONTINUE_REG (0x084)**

| (reserved) | SHA_SHA1_CONTINUE | |
|---|---|---|
| 31 1 | 0 | |
| 0x00000000 | 0 | Reset |

**SHA_SHA1_CONTINUE** Write 1 to continue the SHA-1 operation with subsequent blocks. (WO)

**Register 16.4. SHA_SHA1_LOAD_REG (0x088)**

| (reserved) | SHA_SHA1_LOAD | |
|---|---|---|
| 31 1 | 0 | |
| 0x00000000 | 0 | Reset |

**SHA_SHA1_LOAD** Write 1 to finish the SHA-1 operation to calculate the final message hash. (WO)

**Register 16.5. SHA_SHA1_BUSY_REG (0x08C)**

| (reserved) | SHA_SHA1_BUSY |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA1_BUSY** SHA-1 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

**Register 16.6. SHA_SHA256_START_REG (0x090)**

| (reserved) | SHA_SHA256_START |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA256_START** Write 1 to start an SHA-256 operation on the first message block. (WO)

**Register 16.7. SHA_SHA256_CONTINUE_REG (0x094)**

| (reserved) | SHA_SHA256_CONTINUE |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA256_CONTINUE** Write 1 to continue the SHA-256 operation with subsequent blocks. (WO)

**Register 16.8. SHA_SHA256_LOAD_REG (0x098)**

| (reserved) | SHA_SHA256_LOAD |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA256_LOAD** Write 1 to finish the SHA-256 operation to calculate the final message hash. (WO)

**Register 16.9. SHA_SHA256_BUSY_REG (0x09C)**

| (reserved) | SHA_SHA256_BUSY |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA256_BUSY** SHA-256 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

**Register 16.10. SHA_SHA384_START_REG (0x0A0)**

| (reserved) | SHA_SHA384_START |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA384_START** Write 1 to start an SHA-384 operation on the first message block. (WO)

**Register 16.11. SHA_SHA384_CONTINUE_REG (0x0A4)**

| (reserved) | SHA_SHA384_CONTINUE |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA384_CONTINUE** Write 1 to continue the SHA-384 operation with subsequent blocks. (WO)

**Register 16.12. SHA_SHA384_LOAD_REG (0x0A8)**

| (reserved) | SHA_SHA384_LOAD |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA384_LOAD** Write 1 to finish the SHA-384 operation to calculate the final message hash. (WO)

**Register 16.13. SHA_SHA384_BUSY_REG (0x0AC)**

| (reserved) | SHA_SHA384_BUSY |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA384_BUSY** SHA-384 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 16.14. SHA_SHA512_START_REG (0x0B0)

| (reserved) | SHA_SHA512_START |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA512_START** Write 1 to start an SHA-512 operation on the first message block. (WO)

Register 16.15. SHA_SHA512_CONTINUE_REG (0x0B4)

| (reserved) | SHA_SHA512_CONTINUE |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA512_CONTINUE** Write 1 to continue the SHA-512 operation with subsequent blocks. (WO)

Register 16.16. SHA_SHA512_LOAD_REG (0x0B8)

| (reserved) | SHA_SHA512_LOAD |
|---|---|
| 31 … 1 | 0 |
| 0x00000000 | 0 |

Reset

**SHA_SHA512_LOAD** Write 1 to finish the SHA-512 operation to calculate the final message hash. (WO)

**Register 16.17. SHA_SHA512_BUSY_REG (0x0BC)**

| (reserved) | SHA_SHA512_BUSY | |
|---|---|---|
| 31 … 1 | 0 | |
| 0x00000000 | 0 | Reset |

**SHA_SHA512_BUSY** SHA-512 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

# Chapter 17

# External Memory Encryption and Decryption (FLASH)

## 17.1 Overview

Many variants of the ESP32 must store programs and data in external flash memory. The external flash memory chip is likely to contain proprietary firmware and sensitive user data, such as credentials for gaining access to a private network. The Flash Encryption block can encrypt code and write encrypted code to off-chip flash memory for enhanced hardware security. When the CPU reads off-chip flash through the cache, the Flash Decryption block can automatically decrypt instructions and data read from the off-chip flash, thus providing hardware-based security for application code.

## 17.2 Features

- Various key generation methods
- Software-based encryption
- High-speed, hardware decryption
- Register configuration, system parameters and boot mode jointly determine the flash encryption/decryption function.

## 17.3 Functional Description

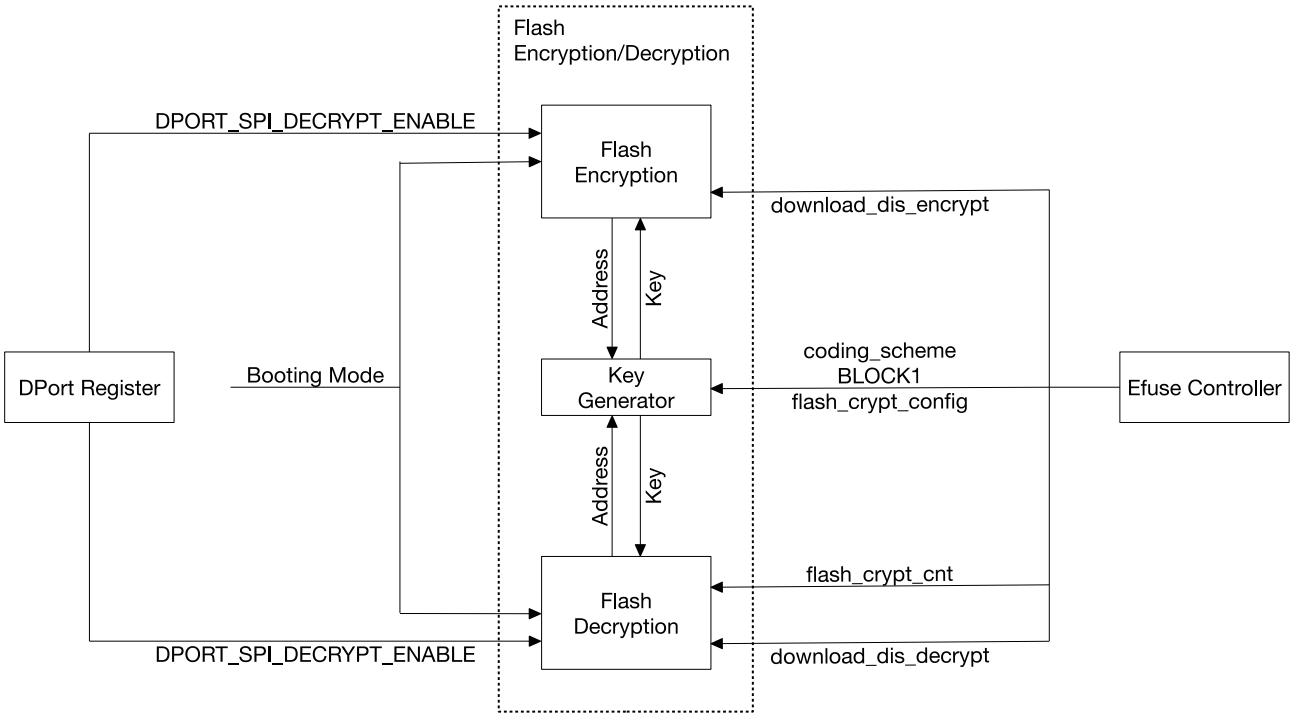



Figure 17.3-1. Flash Encryption/Decryption Module Architecture

The Flash Encryption/Decryption module consists of three parts, namely the Key Generator, Flash Encryption block and Flash Decryption block. The structure of these parts is shown in Figure 17.3-1. The Key Generator is shared by both the Flash Encryption block and the Flash Decryption block, which can function simultaneously.

In the peripheral DPort Register, the register relevant to Flash Encryption/Decryption is DPORT_SPI_ENCRYPT_ENABLE bit and DPORT_SPI_DECRYPT_ENABLE bit in DPORT_SLAVE_SPI_CONFIG_REG. The Flash Encryption/Decryption module will fetch six system parameters from the peripheral eFuse Controller. These parameters are: coding_scheme, BLOCK1, flash_crypt_config, download_dis_encrypt, flash_crypt_cnt, and download_dis_decrypt.

### 17.3.1 Key Generator

According to system parameters coding_scheme and BLOCK1, the Key Generator will first generate $Key_o = f(coding_scheme, BLOCK1)$.

Then, according to system parameter flash_crypt_config, and off-chip flash physical addresses $Addr_e$ and $Addr_d$ accessed by the Flash Encryption block and the Flash Decryption block, the Key Generator will respectively figure out that:
$Key_e = g(Key_o, flash_crypt_config, Addr_e)$,
$Key_d = g(Key_o, flash_crypt_config, Addr_d)$.

When all values of system parameter flash_crypt_config are 0, $Key_e$ and $Key_d$ are not relevant to the physical address of the off-chip flash. When all values of system parameter flash_crypt_config are not 0, every 8-word block on the off-chip flash has a dedicated $Key_e$ and $Key_d$.

### 17.3.2 Flash Encryption Block

The Flash Encryption block is equipped with registers that can be accessed by the CPU directly. Registers embedded in the Flash Encryption block, registers in the peripheral DPort Register, system parameters and Boot Mode jointly configure and control this block.

The Flash Encryption block requires software intervention during operation. The steps are as follows:

1. Set the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG.
2. Write the physical address prepared for the off-chip flash on register FLASH_ENCRYPT_ADDRESS_REG. The address must be 8-word boundary aligned.
3. The Flash Encryption block must encrypt 8-word long code segments. Write the lowest word to register FLASH_ENCRYPT_BUFFER_0_REG, the second-lowest word into FLASH_ENCRYPT_BUFFER_1_REG, and so on, up to FLASH_ENCRYPT_BUFFER_7_REG.
4. Set the FLASH_START bit in FLASH_ENCRYPT_START_REG.
5. Wait for the FLASH_DONE bit to be set in FLASH_ENCRYPT_DONE_REG.
6. Use this function and write any 8-word code to the 8-word aligned address on the off-chip flash via the peripheral SPI0.

In Steps 1 to 5, the Flash Encryption block encrypts 8-word long codes. The key encryption algorithm uses $Key_e$. The encryption result will also be 8-word long. In Step 6, the peripheral SPI0 writes encrypted results of the Flash Encryption block to the off-chip flash. One parameter of the function used in Step 6 will be the physical address of the off-chip flash. The physical address must be 8-word boundary aligned. Also, the value must be the same as the value written into register FLASH_ENCRYPT_ADDRESS_REG during Step 2. Even though the function used in Step 6 still has a parameter with an 8-word long code, the parameter will be meaningless if Steps 1 to 5 are executed. The Peripheral SPI0 will use the encrypted result instead. If the Flash Encryption block is not operating, or has not executed Steps 1 to 5, Step 6 will not use the encrypted result. Instead, the function parameter will be used.

Flash Encryption Operating Conditions:

- During SPI Flash Boot

  If the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG is 1, the Flash Encryption block is operational. Otherwise, it is not.

- During Download Boot

  If the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG is 1, and system parameter download_dis_encrypt is 0, the Flash Encryption block is operational. Otherwise, it is not.

Even though software participates in the whole process, it cannot directly read the encrypted codes. Instead, the encrypted codes are integrated into the off-chip flash. Even though the CPU can skip the cache and get the encrypted code directly by reading the off-chip flash, the software can by no means access $Key_e$.

### 17.3.3 Flash Decryption Block

Flash Decryption is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access the Flash Decryption block. The Peripheral DPort Register, system parameters and Booting Mode jointly control and configure the Flash Decryption block.

When the Flash Decryption block is operating, the CPU will read instructions and data from the off-chip flash via the cache. The Flash Decryption block automatically decrypts the instructions and data in the cache. The entire decryption process does not need software intervention and is transparent to the cache. The decryption algorithm can decrypt the code that has been encrypted by the Flash Encryption block. Software cannot access the key algorithm $Key_d$ used.

When the Flash Decryption block is not operating, it does not have any effect on the contents stored in the off-chip flash, be they encrypted or unencrypted. What the CPU reads via the cache is the original information stored in the off-chip flash.

Flash Encryption Operating Conditions:

- During SPI Flash Boot

  In the efuse system parameter flash_crypt_cnt (7 bits wide), if the number of bits with value 1 is odd, the Flash Decryption block is operational. Otherwise, it is not.

- During Download Boot

  If the DPORT_SPI_DECRYPT_ENABLE bit in DPORT_SLAVE_SPI_CONFIG_REG is 1, and system parameter download_dis_decrypt is 0, the Flash Decryption block is operational. Otherwise, it is not.

## 17.4 Register Summary

The addresses in this section are relative to the External Memory Encryption and Decryption (FLASH) base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| FLASH_ENCRYPTION_BUFFER_0_REG | Flash encryption buffer register 0 | 0x3FF5B000 | WO |
| FLASH_ENCRYPTION_BUFFER_1_REG | Flash encryption buffer register 1 | 0x3FF5B004 | WO |
| FLASH_ENCRYPTION_BUFFER_2_REG | Flash encryption buffer register 2 | 0x3FF5B008 | WO |
| FLASH_ENCRYPTION_BUFFER_3_REG | Flash encryption buffer register 3 | 0x3FF5B00C | WO |
| FLASH_ENCRYPTION_BUFFER_4_REG | Flash encryption buffer register 4 | 0x3FF5B010 | WO |
| FLASH_ENCRYPTION_BUFFER_5_REG | Flash encryption buffer register 5 | 0x3FF5B014 | WO |
| FLASH_ENCRYPTION_BUFFER_6_REG | Flash encryption buffer register 6 | 0x3FF5B018 | WO |
| FLASH_ENCRYPTION_BUFFER_7_REG | Flash encryption buffer register 7 | 0x3FF5B01C | WO |
| FLASH_ENCRYPTION_START_REG | Encrypt operation control register | 0x3FF5B020 | WO |
| FLASH_ENCRYPTION_ADDRESS_REG | External flash address register | 0x3FF5B024 | WO |
| FLASH_ENCRYPTION_DONE_REG | Encrypt operation status register | 0x3FF5B028 | RO |

## 17.5 Register

The addresses in this section are relative to the External Memory Encryption and Decryption (FLASH) base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 17.4 *Register Summary*.

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

**Register 17.1. FLASH_ENCRYPTION_BUFFER_*n*_REG (*n*: 0-7) (0x0+4**n*)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**FLASH_ENCRYPTION_BUFFER_*n*_REG** Data buffers for encryption. (WO)

**Register 17.2. FLASH_ENCRYPTION_START_REG (0x020)**

| (reserved) | FLASH_START | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**FLASH_START** Set this bit to start encryption operation on data buffer. (WO)

**Register 17.3. FLASH_ENCRYPTION_ADDRESS_REG (0x024)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**FLASH_ENCRYPTION_ADDRESS_REG** The physical address on the off-chip flash must be 8-word boundary aligned. (WO)

**Register 17.4. FLASH_ENCRYPTION_DONE_REG (0x028)**

| (reserved) | FLASH_DONE | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**FLASH_DONE** Set this bit when encryption operation is complete. (RO)

# Chapter 18

# Random Number Generator (RNG)

## 18.1 Introduction

The ESP32 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

## 18.2 Feature

The random number generator generates true random numbers, which means random number 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.

## 18.3 Functional Description

Every 32-bit value that the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the thermal noise in the system and the asynchronous clock mismatch.

Thermal noise comes from the high-speed ADC or SAR ADC or both. Whenever the high-speed ADC or SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.

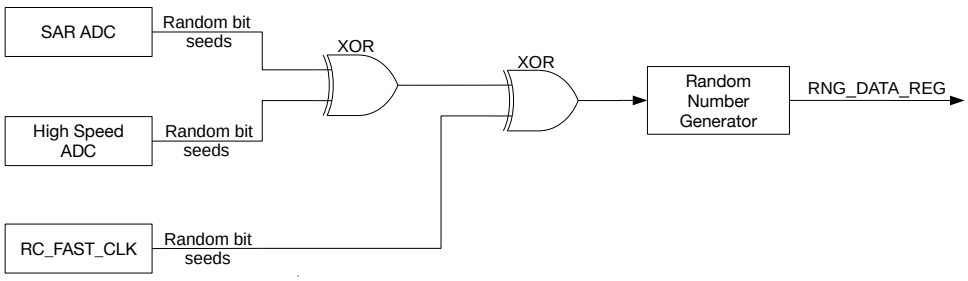



Figure 18.3-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle of RC_FAST_CLK (8 MHz), which is generated from an internal RC oscillator (see Chapter Reset and Clock for details). Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 500 kHz to obtain the maximum entropy.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

## 18.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC or high-speed ADC is enabled. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 31 *On-Chip Sensors and Analog Signal Processing*.
- High-speed ADC is enabled automatically when the Wi-Fi or Bluetooth modules is enabled.

**Note:**
Note that, when the Wi-Fi module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.

When using the random number generator, read the 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 18.3 above.

## 18.5 Register Summary

| Name | Description | Address | Access |
|---|---|---|---|
| RNG_DATA_REG | Random number data | 0x3FF75144 | RO |

## 18.6 Register

**Register 18.1. RNG_DATA_REG (0x3FF75144)**

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**RNG_DATA_REG** Random number source. (RO)

# Part V

# Connectivity Interface

This part addresses the connectivity aspects of the system, describing components related to various communication interfaces like I2C, I2S, SPI, UART, and more. The part also covers interfaces to generate signals used in remote control, motor control, LED control, etc.

# Chapter 19

# UART Controller (UART)

## 19.1 Overview

Embedded applications often require a simple method of exchanging data between devices that need minimal system resources. The Universal Asynchronous Receiver/Transmitter (UART) is one such standard that can realize a flexible full-duplex data exchange among different devices. The three UART controllers available on a chip are compatible with UART-enabled devices from various manufacturers. The UART can also carry out an IrDA (Infrared Data Exchange), or function as an RS-485 modem.

All UART controllers integrated in the ESP32 feature an identical set of registers for ease of programming and flexibility. In this documentation, these controllers are referred to as UART*n*, where *n* = 0, 1, and 2, referring to UART0, UART1, and UART2, respectively.

## 19.2 UART Features

The UART modules have the following main features:

- Programmable baud rate
- 1024 × 8-bit RAM shared by three UART transmit-FIFOs and receive-FIFOs
- Supports input baud rate self-check
- Supports 5/6/7/8 bits of data length
- Supports 1/1.5/2 STOP bits
- Supports parity bit
- Supports RS485 Protocol
- Supports IrDA Protocol
- Supports DMA to communicate data in high speed
- Supports UART wake-up
- Supports both software and hardware flow control

## 19.3 Functional Description

### 19.3.1 Introduction

UART is a character-oriented data link that can be used to achieve communication between two devices. The asynchronous mode of transmission means that it is not necessary to add clocking information to the data

being sent. This, in turn, requires that the data rate, STOP bits, parity, etc., be identical at the transmitting and receiving end for the devices to communicate successfully.

A typical UART frame begins with a START bit, followed by a "character" and an optional parity bit for error detection, and it ends with a STOP condition. The UART controllers available on the ESP32 provide hardware support for multiple lengths of data and STOP bits. In addition, the controllers support both software and hardware flow control, as well as DMA, for seamless high-speed data transfer. This allows the developer to employ multiple UART ports in the system with minimal software overhead.

## 19.3.2 UART Architecture

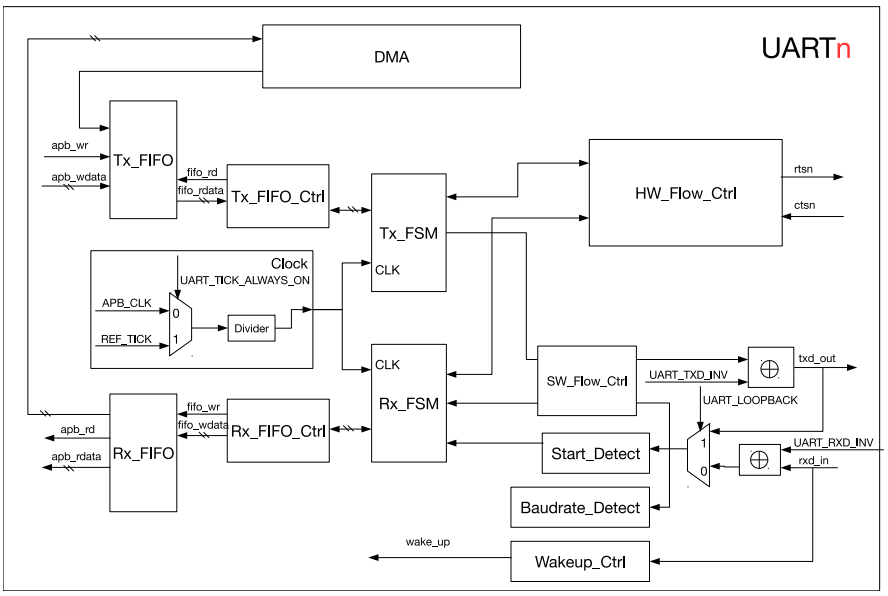



Figure 19.3-1. UART Basic Structure

Figure 19.3-1 shows the basic block diagram of the UART controller. The UART block can derive its clock from two sources: the 80-MHz APB_CLK, or the reference clock REF_TICK (please refer to Chapter Reset and Clock for more details). These two clock sources can be selected by configuring UART_TICK_REF_ALWAYS_ON.

Then, a divider in the clock path divides the selected clock source to generate clock signals that drive the UART module. UART_CLKDIV_REG contains the clock divider value in two parts — UART_CLKDIV (integral part) and UART_CLKDIV_FRAG (decimal part).

The UART controller can be further broken down into two functional blocks — the transmit block and the receive block.

The transmit block contains a transmit-FIFO buffer, which buffers data awaiting to be transmitted. Software can write Tx_FIFO via APB, and transmit data into Tx_FIFO via DMA. Tx_FIFO_Ctrl is used to control read- and write-access to the Tx_FIFO. When Tx_FIFO is not null, Tx_FSM reads data via Tx_FIFO_Ctrl, and transmits data out according to the set frame format. The outgoing bit stream can be inverted by appropriately configuring the register UART_TXD_INV.

The receive-block contains a receive-FIFO buffer, which buffers incoming data awaiting to be processed. The input bit stream, rxd_in, is fed to the UART controller. Negation of the input stream can be controlled by

configuring the UART_RXD_INV register. Baudrate_Detect measures the baud rate of the input signal by measuring the minimum pulse width of the input bit stream. Start_Detect is used to detect a START bit in a frame of incoming data. After detecting the START bit, RX_FSM stores data retrieved from the received frame into Rx_FIFO through Rx_FIFO_Ctrl.

Software can read data in the Rx_FIFO through the APB. In order to free the CPU from engaging in data transfer operations, the DMA can be configured for sending or receiving data.

HW_Flow_Ctrl is able to control the data flow of rxd_in and txd_out through standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls the data flow by inserting special characters in the incoming and outgoing data flow. When UART is in Light-sleep mode (refer to Chapter Low-Power Management), Wakeup_Ctrl will start counting pulses in rxd_in. When the number or positive edges of RxD signal is greater than or equal to (UART_ACTIVE_THRESHOLD+2), a wake_up signal will be generated and sent to RTC. RTC will then wake up the UART controller. **Note** that only UART1 and UART2 support Light-sleep mode and that rxd_in cannot be input through GPIO Matrix but only through IO_MUX.

### 19.3.3 UART RAM

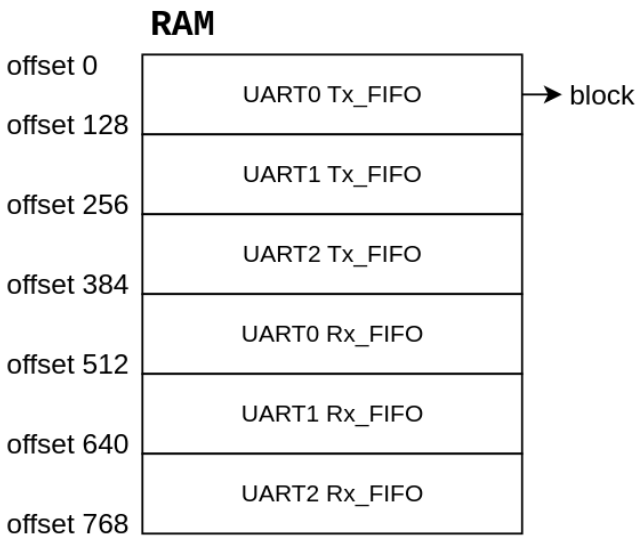



Figure 19.3-2. UART Shared RAM

Three UART controllers share a 1024 × 8-bit RAM space. As illustrated in Figure 19.3-2, RAM is allocated in different blocks. One block holds 128 × 8-bit data. Figure 19.3-2 illustrates the default RAM allocated to Tx_FIFO and Rx_FIFO of the three UART controllers. Tx_FIFO of UART*n* can be extended by setting UART*n*_TX_SIZE, while Rx_FIFO of UART*n* can be extended by setting UART*n*_RX_SIZE.

**NOTICE:** Extending the FIFO space of a UART controller may take up the FIFO space of another UART controller.

If none of the UART controllers is active, setting UART_MEM_PD, UART1_MEM_PD, and UART2_MEM_PD can prompt the RAM to enter low-power mode.

In UART0, bit UART_TXFIFO_RST and bit UART_RXFIFO_RST can be set to reset Tx_FIFO or Rx_FIFO, respectively. In UART1, bit UART1_TXFIFO_RST and bit UART1_RXFIFO_RST can be set to reset Tx_FIFO or Rx_FIFO, respectively.

**Note:**
UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these 2 registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data.

UART*n* can access FIFO via register UART_FIFO_REG.

### 19.3.4 Baud Rate Detection

Setting UART_AUTOBAUD_EN for a UART controller will enable the baud rate detection function. The Baudrate_Detect block shown in Figure 19.3-1 can filter glitches with a pulse width lower than UART_GLITCH_FILT.

In order to use the baud rate detection feature, some random data should be sent to the receiver before starting the UART communication stream. This is required so that the baud rate can be determined based on the pulse width. UART_LOWPULSE_MIN_CNT stores minimum low-pulse width, UART_HIGHPULSE_MIN_CNT stores minimum high-pulse width. By reading these two registers, software can calculate the baud rate of the transmitter.

### 19.3.5 UART Data Frame

Figure 19.3-3 shows the basic data frame structure. A data frame starts with a START condition and ends with a STOP condition. The START condition requires 1 bit and the STOP condition can be realized using 1/1.5/2-bit widths (as set by UART_STOP_BIT_NUM) (in RS485 mode turnaround delay may be added by configuring UART_DLO_EN and UART_DL1_EN). The START is low level, while the STOP is high level.

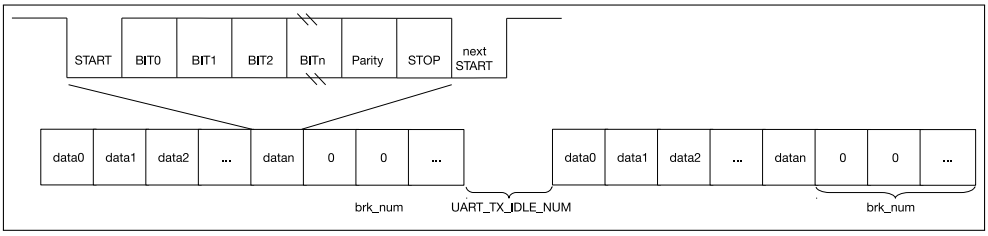



Figure 19.3-3. UART Data Frame Structure

The length of a character (BIT0 to BITn) can comprise 5 to 8 bits and can be configured by UART_BIT_NUM. When UART_PARITY_EN is set, the UART controller hardware will add the appropriate parity bit after the data. UART_PARITY is used to select odd parity or even parity. If the receiver detects an error in the input character, interrupt UART_PARITY_ERR_INT will be generated. If the receiver detects an error in the frame format, interrupt UART_FRM_ERR_INT will be generated.

Interrupt UART_TX_DONE_INT will be generated when all data in Tx_FIFO have been transmitted. When UART_TXD
_BRK is set, the transmitter enter the Break condition and send several NULL characters after the process of sending data is completed. The number of NULL characters can be configured by UART_TX_BRK_NUM. After the transmitter finishes sending all NULL characters, interrupt UART_TX_BRK_DONE_INT will be generated. The minimum interval between data frames can be configured with UART_TX_IDLE_NUM. If the idle time of a data

frame is equal to, or larger than, the configured value of register UART_TX_IDLE_NUM, interrupt UART_TX_BRK_IDLE_DONE_INT will be generated.

The receiver can also detect the Break conditions when the RX data line remains logical low for one NULL character transmission, and a UART_BRK_DET_INT interrupt will be triggered to detect that 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.

### 19.3.6 AT_CMD Character Structure

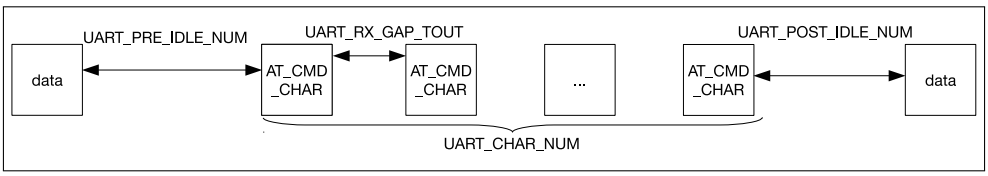



Figure 19.3-4. AT_CMD Character Format

Figure 19.3-4 shows a special AT_CMD character format. If the receiver constantly receives UART_AT_CMD_CHAR characters and these characters satisfy the following conditions, interrupt UART_AT_CMD_CHAR_DET_INT will be generated.

- Between the first UART_AT_CMD_CHAR and the last non-UART_AT_CMD_CHAR, there are at least UART_PER_IDLE_NUM APB clock cycles.
- Between every UART_AT_CMD_CHAR character there must be less than UART_RX_GAP_TOUT APB clock cycles.
- The number of received UART_AT_CMD_CHAR characters must be equal to, or greater than, UART_CHAR_NUM.
- Between the last UART_AT_CMD_CHAR character received and the next non-UART_AT_CMD_CHAR, there are at least UART_POST_IDLE_NUM APB clock cycles.

### 19.3.7 Flow Control

UART controller supports both hardware and software flow control. Hardware flow control regulates data flow through input signal dsrn_in and output signal rtsn_out. Software flow control regulates data flow by inserting special characters in the flow of sent data and by detecting special characters in the flow of received data.

#### 19.3.7.1 Hardware Flow Control

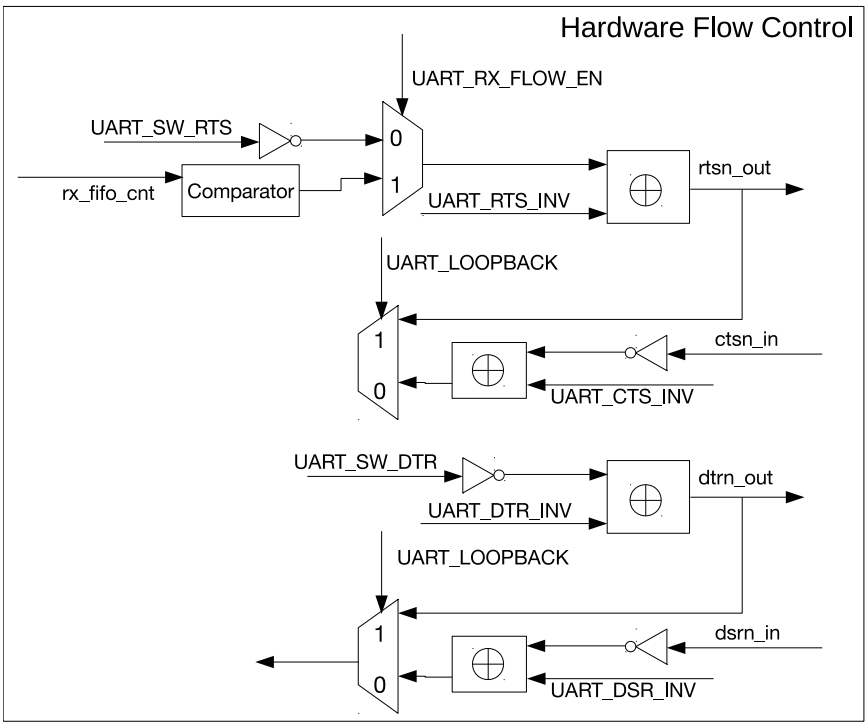



Figure 19.3-5. Hardware Flow Control

Figure 19.3-5 illustrates how the UART hardware flow control works. In hardware flow control, a high state of the output signal rtsn_out signifies that a data transmission is requested, while a low state of the same signal notifies the counterpart to stop data transmission until rtsn_out is pulled high again. There are two ways for a transmitter to realize hardware flow control:

- UART_RX_FLOW_EN is 0: The level of rtsn_out can be changed by configuring UART_SW_RTS.
- UART_RX_FLOW_EN is 1: If data in Rx_FIFO is greater than UART_RX_FLOW_THRHD, the level of rtsn_out will be lowered.

If the UART controller detects an edge on ctsn_in, it will generate interrupt UART_CTS_CHG_INT and will stop transmitting data, once the current data transmission is completed.

The high level of the output signal dtrn_out signifies that the transmitter has finished data preparation. UART controller will generate interrupt UART_DSR_CHG_INT, after it detects an edge on the input signal dsrn_in. After the software detects the above-mentioned interrupt, the input signal level of dsrn_in can be figured out by reading UART_DSRN. The software then decides whether it is able to receive data at that time or not.

Setting UART_LOOPBACK will enable the UART loopback detection function. In this mode, the output signal txd_out of UART 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 transmitted corresponds to the data received, UART is able to transmit and receive data normally.

#### 19.3.7.2 Software Flow Control

Software can force the transmitter to stop transmitting data by setting UART_FORCE_XOFF, as well as force the transmitter to continue sending data by setting UART_FORCE_XON.

UART can also control the software flow by transmitting special characters. Setting UART_SW_FLOW_CON_EN will enable the software flow control function. If the number of data bytes that UART has received exceeds that of the UART_XOFF threshold, the UART controller can send UART_XOFF_CHAR to instruct its counterpart to stop data transmission.

When UART_SW_FLOW_CON_EN is 1, software can send flow control characters at any time. When UART_SEND
_XOFF is set, the transmitter will insert a UART_XOFF_CHAR and send it after the current data transmission is completed. When UART_SEND_XON is set, the transmitter will insert a UART_XON_CHAR and send it after the current data transmission is completed.

### 19.3.8 UART DMA

For information on the UART DMA, please refer to Chapter DMA Controller.

### 19.3.9 UART Interrupts

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects the configured at_cmd char.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between transmitter and receiver in RS-485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when a data frame error is detected in RS-485.
- UART_RS485_PARITY_ERR_INT: Triggered when a parity error is detected in RS-485 mode.
- UART_TX_DONE_INT: Triggered when the transmitter has sent out all FIFO data.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter's idle state has been kept to a minimum after sending the last data.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter completes sending NULL characters, after all data in transmit-FIFO are sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a START bit.
- UART_SW_XOFF_INT: Triggered, if the receiver gets an Xon char when UART_SW_FLOW_CON_EN is set to 1.
- UART_SW_XON_INT: Triggered, if the receiver gets an Xoff char when UART_SW_FLOW_CON_EN is set to 1.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than RX_TOUT_THRHD to receive a 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 the CTSn signal.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of the DSRn signal.
- UART_RXFIFO_OVF_INT: Triggered when the receiver gets more data than the FIFO can store.
- 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 in the data.

- UART_TXFIFO_EMPTY_INT: Triggered when the amount of data in the transmit-FIFO is less than what tx_mem_cnttxfifo_cnt specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver gets more data than what (rx_flow_thrhd_h3, rx_flow_thrhd) specifies.

### 19.3.10 UHCI Interrupts

- UHCI_SEND_A_REG_Q_INT: When using the always_send registers to send a series of short packets, this is triggered when DMA has sent a short packet.
- UHCI_SEND_S_REG_Q_INT: When using the single_send registers to send a series of short packets, this is triggered when DMA has sent a short packet.
- UHCI_OUT_TOTAL_EOF_INT: Triggered when all data have been sent.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when there are some errors in EOF in the outlink descriptor.
- UHCI_IN_DSCR_EMPTY_INT: Triggered when there are not enough inlinks for DMA.
- UHCI_OUT_DSCR_ERR_INT: Triggered when there are some errors in the inlink descriptor.
- UHCI_IN_DSCR_ERR_INT: Triggered when there are some errors in the outlink descriptor.
- UHCI_OUT_EOF_INT: Triggered when the current descriptor's EOF bit is 1.
- UHCI_OUT_DONE_INT: Triggered when an outlink descriptor is completed.
- UHCI_IN_ERR_EOF_INT: Triggered when there are some errors in EOF in the inlink descriptor.
- UHCI_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- UHCI_IN_DONE_INT: Triggered when an inlink descriptor has been completed.
- UHCI_TX_HUNG_INT: Triggered when DMA takes much time to read data from RAM.
- UHCI_RX_HUNG_INT: Triggered when DMA takes much time to receive data .
- UHCI_TX_START_INT: Triggered when DMA detects a separator char.
- UHCI_RX_START_INT: Triggered when a separator char has been sent.

## 19.4 Register Summary

### 19.4.1 UART Register Summary

The addresses in this section are relative to the UART base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | UART0 | UART1 | UART2 | Acc |
|---|---|---|---|---|---|
| **Configuration registers** | | | | | |
| UART_CONF0_REG | Configuration register 0 | 0x3FF40020 | 0x3FF50020 | 0x3FF6E020 | R/W |

| | | | | | |
|---|---|---|---|---|---|
| UART_CONF1_REG | Configuration register 1 | 0x3FF40024 | 0x3FF50024 | 0x3FF6E024 | R/W |
| UART_CLKDIV_REG | Clock divider configuration | 0x3FF40014 | 0x3FF50014 | 0x3FF6E014 | R/W |
| UART_FLOW_CONF_REG | Software flow-control configuration | 0x3FF40034 | 0x3FF50034 | 0x3FF6E034 | R/W |
| UART_SWFC_CONF_REG | Software flow-control character configuration | 0x3FF4003C | 0x3FF5003C | 0x3FF6E03C | R/W |
| UART_SLEEP_CONF_REG | Sleep-mode configuration | 0x3FF40038 | 0x3FF50038 | 0x3FF6E038 | R/W |
| UART_IDLE_CONF_REG | Frame-end idle configuration | 0x3FF40040 | 0x3FF50040 | 0x3FF6E040 | R/W |
| UART_RS485_CONF_REG | RS485 mode configuration | 0x3FF40044 | 0x3FF50044 | 0x3FF6E044 | R/W |
| **Status registers** | | | | | |
| UART_STATUS_REG | UART status register | 0x3FF4001C | 0x3FF5001C | 0x3FF6E01C | RO |
| UART_MEM_TX_STATUS_REG | TX FIFO write and read offset address | 0x3FF4005C | 0x3FF5005C | 0x3FF6E05C | RO |
| UART_MEM_RX_STATUS_REG | RX FIFO write and read offset address | 0x3FF40060 | 0x3FF50060 | 0x3FF6E060 | RO |
| **Autobaud registers** | | | | | |
| UART_AUTOBAUD_REG | Autobaud configuration register | 0x3FF40018 | 0x3FF50018 | 0x3FF6E018 | R/W |
| UART_LOWPULSE_REG | Autobaud minimum low pulse duration register | 0x3FF40028 | 0x3FF50028 | 0x3FF6E028 | RO |
| UART_HIGHPULSE_REG | Autobaud minimum high pulse duration register | 0x3FF4002C | 0x3FF5002C | 0x3FF6E02C | RO |
| UART_POSPULSE_REG | Autobaud high pulse register | 0x3FF40068 | 0x3FF50068 | 0x3FF6E068 | RO |
| UART_NEGPULSE_REG | Autobaud low pulse register | 0x3FF4006C | 0x3FF5006C | 0x3FF6E06C | RO |
| UART_RXD_CNT_REG | Autobaud edge change count register | 0x3FF40030 | 0x3FF50030 | 0x3FF6E030 | RO |
| **AT escape sequence detection configuration** | | | | | |
| UART_AT_CMD_PRECNT_REG | Pre-sequence timing configuration | 0x3FF40048 | 0x3FF50048 | 0x3FF6E048 | R/W |
| UART_AT_CMD_POSTCNT_REG | Post-sequence timing configuration | 0x3FF4004C | 0x3FF5004C | 0x3FF6E04C | R/W |
| UART_AT_CMD_GAPTOUT_REG | Timeout configuration | 0x3FF40050 | 0x3FF50050 | 0x3FF6E050 | R/W |

| | | | | | |
|---|---|---|---|---|---|
| UART_AT_CMD_CHAR_REG | AT escape sequence detection configuration | 0x3FF40054 | 0x3FF50054 | 0x3FF6E054 | R/W |
| **FIFO configuration** | | | | | |
| UART_FIFO_REG | FIFO data register | 0x3FF40000 | 0x3FF50000 | 0x3FF6E000 | R/W |
| UART_MEM_CONF_REG | UART threshold and allocation configuration | 0x3FF40058 | 0x3FF50058 | 0x3FF6E058 | R/W |
| UART_MEM_CNT_STATUS_REG | Receive and transmit memory configuration | 0x3FF40064 | 0x3FF50064 | 0x3FF6E064 | RO |
| **Interrupt registers** | | | | | |
| UART_INT_RAW_REG | Raw interrupt status | 0x3FF40004 | 0x3FF50004 | 0x3FF6E004 | RO |
| UART_INT_ST_REG | Masked interrupt status | 0x3FF40008 | 0x3FF50008 | 0x3FF6E008 | RO |
| UART_INT_ENA_REG | Interrupt enable bits | 0x3FF4000C | 0x3FF5000C | 0x3FF6E00C | R/W |
| UART_INT_CLR_REG | Interrupt clear bits | 0x3FF40010 | 0x3FF50010 | 0x3FF6E010 | WO |

### 19.4.2 UHCI Register Summary

The addresses in this section are relative to the UDMA base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | UDMA0 | UDMA1 | Acc |
|---|---|---|---|---|
| **Configuration registers** | | | | |
| UHCI_CONF0_REG | UART and frame separation config | 0x3FF54000 | 0x3FF4C000 | R/W |
| UHCI_CONF1_REG | UHCI config register | 0x3FF5402C | 0x3FF4C02C | R/W |
| UHCI_ESCAPE_CONF_REG | Escape characters configuration | 0x3FF54064 | 0x3FF4C064 | R/W |
| UHCI_HUNG_CONF_REG | Timeout configuration | 0x3FF54068 | 0x3FF4C068 | R/W |
| UHCI_ESC_CONF0_REG | Escape sequence configuration register 0 | 0x3FF540B0 | 0x3FF4C0B0 | R/W |
| UHCI_ESC_CONF1_REG | Escape sequence configuration register 1 | 0x3FF540B4 | 0x3FF4C0B4 | R/W |
| UHCI_ESC_CONF2_REG | Escape sequence configuration register 2 | 0x3FF540B8 | 0x3FF4C0B8 | R/W |
| UHCI_ESC_CONF3_REG | Escape sequence configuration register 3 | 0x3FF540BC | 0x3FF4C0BC | R/W |
| **DMA configuration** | | | | |
| UHCI_DMA_OUT_LINK_REG | Link descriptor address and control | 0x3FF54024 | 0x3FF4C024 | R/W |
| UHCI_DMA_IN_LINK_REG | Link descriptor address and control | 0x3FF54028 | 0x3FF4C028 | R/W |
| UHCI_DMA_OUT_PUSH_REG | FIFO data push register | 0x3FF54018 | 0x3FF4C018 | R/W |

| | | | | |
|---|---|---|---|---|
| UHCI_DMA_IN_POP_REG | FIFO data pop register | 0x3FF54020 | 0x3FF4C020 | RO |
| **DMA status** | | | | |
| UHCI_DMA_OUT_STATUS_REG | DMA FIFO status | 0x3FF54014 | 0x3FF4C014 | RO |
| UHCI_DMA_OUT_EOF_DES_ADDR_REG | Out EOF link descriptor address on success | 0x3FF54038 | 0x3FF4C038 | RO |
| UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG | Out EOF link descriptor address on error | 0x3FF54044 | 0x3FF4C044 | RO |
| UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG | In EOF link descriptor address on success | 0x3FF5403C | 0x3FF4C03C | RO |
| UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG | In EOF link descriptor address on error | 0x3FF54040 | 0x3FF4C040 | RO |
| UHCI_DMA_IN_DSCR_REG | Current inlink descriptor, first word | 0x3FF5404C | 0x3FF4C04C | RO |
| UHCI_DMA_IN_DSCR_BF0_REG | Current inlink descriptor, second word | 0x3FF54050 | 0x3FF4C050 | RO |
| UHCI_DMA_IN_DSCR_BF1_REG | Current inlink descriptor, third word | 0x3FF54054 | 0x3FF4C054 | RO |
| UHCI_DMA_OUT_DSCR_REG | Current outlink descriptor, first word | 0x3FF54058 | 0x3FF4C058 | RO |
| UHCI_DMA_OUT_DSCR_BF0_REG | Current outlink descriptor, second word | 0x3FF5405C | 0x3FF4C05C | RO |
| UHCI_DMA_OUT_DSCR_BF1_REG | Current outlink descriptor, third word | 0x3FF54060 | 0x3FF4C060 | RO |
| **Interrupt registers** | | | | |
| UHCI_INT_RAW_REG | Raw interrupt status | 0x3FF54004 | 0x3FF4C004 | RO |
| UHCI_INT_ST_REG | Masked interrupt status | 0x3FF54008 | 0x3FF4C008 | RO |
| UHCI_INT_ENA_REG | Interrupt enable bits | 0x3FF5400C | 0x3FF4C00C | R/W |
| UHCI_INT_CLR_REG | Interrupt clear bits | 0x3FF54010 | 0x3FF4C010 | WO |

# 19.5 Registers

## 19.5.1 UART Registers

The addresses in this section are relative to the UART base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 19.4.1 *UART Register Summary*.

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

**Register 19.1. UART_FIFO_REG (0x0)**

| | (reserved) | UART_RXFIFO_RD_BYTE | |
|---|---|---|---|
| | 31 … 8 | 7 … 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 0 0 0 0 | |

**UART_RXFIFO_RD_BYTE** UART*n* accesses FIFO via this register. (R/W)

**Register 19.2. UART_INT_RAW_REG (0x4)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | UART_AT_CMD_CHAR_DET_INT_RAW | UART_RS485_CLASH_INT_RAW | UART_RS485_FRM_ERR_INT_RAW | UART_RS485_PARITY_ERR_INT_RAW | UART_TX_DONE_INT_RAW | UART_TX_BRK_IDLE_DONE_INT_RAW | UART_TX_BRK_DONE_INT_RAW | UART_GLITCH_DET_INT_RAW | UART_SW_XOFF_INT_RAW | UART_SW_XON_INT_RAW | UART_RXFIFO_TOUT_INT_RAW | UART_BRK_DET_INT_RAW | UART_CTS_CHG_INT_RAW | UART_DSR_CHG_INT_RAW | UART_RXFIFO_OVF_INT_RAW | UART_FRM_ERR_INT_RAW | UART_PARITY_ERR_INT_RAW | UART_TXFIFO_EMPTY_INT_RAW | UART_RXFIFO_FULL_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UART_AT_CMD_CHAR_DET_INT_RAW** The raw interrupt status bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (RO)

**UART_RS485_CLASH_INT_RAW** The raw interrupt status bit for the UART_RS485_CLASH_INT interrupt. (RO)

**UART_RS485_FRM_ERR_INT_RAW** The raw interrupt status bit for the UART_RS485_FRM_ERR_INT interrupt. (RO)

**UART_RS485_PARITY_ERR_INT_RAW** The raw interrupt status bit for the UART_RS485_PARITY_ERR_INT interrupt. (RO)

**UART_TX_DONE_INT_RAW** The raw interrupt status bit for the UART_TX_DONE_INT interrupt. (RO)

**UART_TX_BRK_IDLE_DONE_INT_RAW** The raw interrupt status bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (RO)

**UART_TX_BRK_DONE_INT_RAW** The raw interrupt status bit for the UART_TX_BRK_DONE_INT interrupt. (RO)

**UART_GLITCH_DET_INT_RAW** The raw interrupt status bit for the UART_GLITCH_DET_INT interrupt. (RO)

**UART_SW_XOFF_INT_RAW** The raw interrupt status bit for the UART_SW_XOFF_INT interrupt. (RO)

**UART_SW_XON_INT_RAW** The raw interrupt status bit for the UART_SW_XON_INT interrupt. (RO)

**UART_RXFIFO_TOUT_INT_RAW** The raw interrupt status bit for the UART_RXFIFO_TOUT_INT interrupt. (RO)

**UART_BRK_DET_INT_RAW** The raw interrupt status bit for the UART_BRK_DET_INT interrupt. (RO)

**UART_CTS_CHG_INT_RAW** The raw interrupt status bit for the UART_CTS_CHG_INT interrupt. (RO)

**UART_DSR_CHG_INT_RAW** The raw interrupt status bit for the UART_DSR_CHG_INT interrupt. (RO)

**UART_RXFIFO_OVF_INT_RAW** The raw interrupt status bit for the UART_RXFIFO_OVF_INT interrupt. (RO)

**UART_FRM_ERR_INT_RAW** The raw interrupt status bit for the UART_FRM_ERR_INT interrupt. (RO)

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

Register 19.2. UART_INT_RAW_REG (0x4)

Continued from the previous page...

**UART_PARITY_ERR_INT_RAW** The raw interrupt status bit for the UART_PARITY_ERR_INT interrupt. (RO)

**UART_TXFIFO_EMPTY_INT_RAW** The raw interrupt status bit for the UART_TXFIFO_EMPTY_INT interrupt. (RO)

**UART_RXFIFO_FULL_INT_RAW** The raw interrupt status bit for the UART_RXFIFO_FULL_INT interrupt. (RO)

**Register 19.3. UART_INT_ST_REG (0x8)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | UART_AT_CMD_CHAR_DET_INT_ST | UART_RS485_CLASH_INT_ST | UART_RS485_FRM_ERR_INT_ST | UART_RS485_PARITY_ERR_INT_ST | UART_TX_DONE_INT_ST | UART_TX_BRK_IDLE_DONE_INT_ST | UART_TX_BRK_DONE_INT_ST | UART_GLITCH_DET_INT_ST | UART_SW_XOFF_INT_ST | UART_SW_XON_INT_ST | UART_RXFIFO_TOUT_INT_ST | UART_BRK_DET_INT_ST | UART_CTS_CHG_INT_ST | UART_DSR_CHG_INT_ST | UART_RXFIFO_OVF_INT_ST | UART_FRM_ERR_INT_ST | UART_PARITY_ERR_INT_ST | UART_TXFIFO_EMPTY_INT_ST | UART_RXFIFO_FULL_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UART_AT_CMD_CHAR_DET_INT_ST** The masked interrupt status bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (RO)

**UART_RS485_CLASH_INT_ST** The masked interrupt status bit for the UART_RS485_CLASH_INT interrupt. (RO)

**UART_RS485_FRM_ERR_INT_ST** The masked interrupt status bit for the UART_RS485_FRM_ERR_INT interrupt. (RO)

**UART_RS485_PARITY_ERR_INT_ST** The masked interrupt status bit for the UART_RS485_PARITY_ERR_INT interrupt. (RO)

**UART_TX_DONE_INT_ST** The masked interrupt status bit for the UART_TX_DONE_INT interrupt. (RO)

**UART_TX_BRK_IDLE_DONE_INT_ST** The masked interrupt status bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (RO)

**UART_TX_BRK_DONE_INT_ST** The masked interrupt status bit for the UART_TX_BRK_DONE_INT interrupt. (RO)

**UART_GLITCH_DET_INT_ST** The masked interrupt status bit for the UART_GLITCH_DET_INT interrupt. (RO)

**UART_SW_XOFF_INT_ST** The masked interrupt status bit for the UART_SW_XOFF_INT interrupt. (RO)

**UART_SW_XON_INT_ST** The masked interrupt status bit for the UART_SW_XON_INT interrupt. (RO)

**UART_RXFIFO_TOUT_INT_ST** The masked interrupt status bit for the UART_RXFIFO_TOUT_INT interrupt. (RO)

**UART_BRK_DET_INT_ST** The masked interrupt status bit for the UART_BRK_DET_INT interrupt. (RO)

**UART_CTS_CHG_INT_ST** The masked interrupt status bit for the UART_CTS_CHG_INT interrupt. (RO)

**UART_DSR_CHG_INT_ST** The masked interrupt status bit for the UART_DSR_CHG_INT interrupt. (RO)

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

Register 19.3. UART_INT_ST_REG (0x8)

Continued from the previous page...

**UART_RXFIFO_OVF_INT_ST** The masked interrupt status bit for the UART_RXFIFO_OVF_INT interrupt. (RO)

**UART_FRM_ERR_INT_ST** The masked interrupt status bit for the UART_FRM_ERR_INT interrupt. (RO)

**UART_PARITY_ERR_INT_ST** The masked interrupt status bit for the UART_PARITY_ERR_INT interrupt. (RO)

**UART_TXFIFO_EMPTY_INT_ST** The masked interrupt status bit for the UART_TXFIFO_EMPTY_INT interrupt. (RO)

**UART_RXFIFO_FULL_INT_ST** The masked interrupt status bit for UART_RXFIFO_FULL_INT. (RO)

**Register 19.4. UART_INT_ENA_REG (0xC)**

| 31–19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | UART_AT_CMD_CHAR_DET_INT_ENA | UART_RS485_CLASH_INT_ENA | UART_RS485_FRM_ERR_INT_ENA | UART_RS485_PARITY_ERR_INT_ENA | UART_TX_DONE_INT_ENA | UART_TX_BRK_IDLE_DONE_INT_ENA | UART_TX_BRK_DONE_INT_ENA | UART_GLITCH_DET_INT_ENA | UART_SW_XOFF_INT_ENA | UART_SW_XON_INT_ENA | UART_RXFIFO_TOUT_INT_ENA | UART_BRK_DET_INT_ENA | UART_CTS_CHG_INT_ENA | UART_DSR_CHG_INT_ENA | UART_RXFIFO_OVF_INT_ENA | UART_FRM_ERR_INT_ENA | UART_PARITY_ERR_INT_ENA | UART_TXFIFO_EMPTY_INT_ENA | UART_RXFIFO_FULL_INT_ENA | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**UART_AT_CMD_CHAR_DET_INT_ENA** The interrupt enable bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (R/W)

**UART_RS485_CLASH_INT_ENA** The interrupt enable bit for the UART_RS485_CLASH_INT interrupt. (R/W)

**UART_RS485_FRM_ERR_INT_ENA** The interrupt enable bit for the UART_RS485_FRM_ERR_INT interrupt. (R/W)

**UART_RS485_PARITY_ERR_INT_ENA** The interrupt enable bit for the UART_RS485_PARITY_ERR_INT interrupt. (R/W)

**UART_TX_DONE_INT_ENA** The interrupt enable bit for the UART_TX_DONE_INT interrupt. (R/W)

**UART_TX_BRK_IDLE_DONE_INT_ENA** The interrupt enable bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (R/W)

**UART_TX_BRK_DONE_INT_ENA** The interrupt enable bit for the UART_TX_BRK_DONE_INT interrupt. (R/W)

**UART_GLITCH_DET_INT_ENA** The interrupt enable bit for the UART_GLITCH_DET_INT interrupt. (R/W)

**UART_SW_XOFF_INT_ENA** The interrupt enable bit for the UART_SW_XOFF_INT interrupt. (R/W)

**UART_SW_XON_INT_ENA** The interrupt enable bit for the UART_SW_XON_INT interrupt. (R/W)

**UART_RXFIFO_TOUT_INT_ENA** The interrupt enable bit for the UART_RXFIFO_TOUT_INT interrupt. (R/W)

**UART_BRK_DET_INT_ENA** The interrupt enable bit for the UART_BRK_DET_INT interrupt. (R/W)

**UART_CTS_CHG_INT_ENA** The interrupt enable bit for the UART_CTS_CHG_INT interrupt. (R/W)

**UART_DSR_CHG_INT_ENA** The interrupt enable bit for the UART_DSR_CHG_INT interrupt. (R/W)

**UART_RXFIFO_OVF_INT_ENA** The interrupt enable bit for the UART_RXFIFO_OVF_INT interrupt. (R/W)

**UART_FRM_ERR_INT_ENA** The interrupt enable bit for the UART_FRM_ERR_INT interrupt. (R/W)

**UART_PARITY_ERR_INT_ENA** The interrupt enable bit for the UART_PARITY_ERR_INT interrupt. (R/W)

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

Register 19.4. UART_INT_ENA_REG (0xC)

Continued from the previous page...

**UART_TXFIFO_EMPTY_INT_ENA** The interrupt enable bit for the UART_TXFIFO_EMPTY_INT interrupt. (R/W)

**UART_RXFIFO_FULL_INT_ENA** The interrupt enable bit for the UART_RXFIFO_FULL_INT interrupt. (R/W)

**Register 19.5. UART_INT_CLR_REG (0x10)**

| (reserved) | UART_AT_CMD_CHAR_DET_INT_CLR | UART_RS485_CLASH_INT_CLR | UART_RS485_FRM_ERR_INT_CLR | UART_RS485_PARITY_ERR_INT_CLR | UART_TX_DONE_INT_CLR | UART_TX_BRK_IDLE_DONE_INT_CLR | UART_TX_BRK_DONE_INT_CLR | UART_GLITCH_DET_INT_CLR | UART_SW_XOFF_INT_CLR | UART_SW_XON_INT_CLR | UART_RXFIFO_TOUT_INT_CLR | UART_BRK_DET_INT_CLR | UART_CTS_CHG_INT_CLR | UART_DSR_CHG_INT_CLR | UART_RXFIFO_OVF_INT_CLR | UART_FRM_ERR_INT_CLR | UART_PARITY_ERR_INT_CLR | UART_TXFIFO_EMPTY_INT_CLR | UART_RXFIFO_FULL_INT_CLR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**UART_AT_CMD_CHAR_DET_INT_CLR** Set this bit to clear the UART_AT_CMD_CHAR_DET_INT interrupt. (WO)

**UART_RS485_CLASH_INT_CLR** Set this bit to clear the UART_RS485_CLASH_INT interrupt. (WO)

**UART_RS485_FRM_ERR_INT_CLR** Set this bit to clear the UART_RS485_FRM_ERR_INT interrupt. (WO)

**UART_RS485_PARITY_ERR_INT_CLR** Set this bit to clear the UART_RS485_PARITY_ERR_INT interrupt. (WO)

**UART_TX_DONE_INT_CLR** Set this bit to clear the UART_TX_DONE_INT interrupt. (WO)

**UART_TX_BRK_IDLE_DONE_INT_CLR** Set this bit to clear the UART_TX_BRK_IDLE_DONE_INT interrupt. (WO)

**UART_TX_BRK_DONE_INT_CLR** Set this bit to clear the UART_TX_BRK_DONE_INT interrupt. (WO)

**UART_GLITCH_DET_INT_CLR** Set this bit to clear the UART_GLITCH_DET_INT interrupt. (WO)

**UART_SW_XOFF_INT_CLR** Set this bit to clear the UART_SW_XOFF_INT interrupt. (WO)

**UART_SW_XON_INT_CLR** Set this bit to clear the UART_SW_XON_INT interrupt. (WO)

**UART_RXFIFO_TOUT_INT_CLR** Set this bit to clear the UART_RXFIFO_TOUT_INT interrupt. This bit can be set only when both rxfifo_cnt and rx_mem_cnt are 0. (WO)

**UART_BRK_DET_INT_CLR** Set this bit to clear the UART_BRK_DET_INT interrupt. (WO)

**UART_CTS_CHG_INT_CLR** Set this bit to clear the UART_CTS_CHG_INT interrupt. (WO)

**UART_DSR_CHG_INT_CLR** Set this bit to clear the UART_DSR_CHG_INT interrupt. (WO)

**UART_RXFIFO_OVF_INT_CLR** Set this bit to clear the UART_RXFIFO_OVF_INT interrupt. (WO)

**UART_FRM_ERR_INT_CLR** Set this bit to clear the UART_FRM_ERR_INT interrupt. (WO)

**UART_PARITY_ERR_INT_CLR** Set this bit to clear the UART_PARITY_ERR_INT interrupt. (WO)

**UART_TXFIFO_EMPTY_INT_CLR** Set this bit to clear the UART_TXFIFO_EMPTY_INT interrupt. (WO)

**UART_RXFIFO_FULL_INT_CLR** Set this bit to clear the UART_RXFIFO_FULL_INT interrupt. This bit can be set only when data in Rx_FIFO is less than UART_RXFIFO_FULL_THRHD. (WO)

**Register 19.6. UART_CLKDIV_REG (0x14)**

| (reserved) | UART_CLKDIV_FRAG | UART_CLKDIV | |
|---|---|---|---|
| 31 … 24 | 23 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x00 | 0x0002B6 | Reset |

**UART_CLKDIV_FRAG** The decimal part of the frequency divider factor. (R/W)

**UART_CLKDIV** The integral part of the frequency divider factor. (R/W)

**Register 19.7. UART_AUTOBAUD_REG (0x18)**

| (reserved) | UART_GLITCH_FILT | (reserved) | UART_AUTOBAUD_EN | |
|---|---|---|---|---|
| 31 … 16 | 15 … 8 | 7 … 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x010 | 0 0 0 0 0 0 0 | 0 | Reset |

**UART_GLITCH_FILT** When the input pulse width is lower than this value, the pulse is ignored. This register is used in the autobauding process. (R/W)

**UART_AUTOBAUD_EN** This is the enable bit for autobaud. (R/W)

**Register 19.8. UART_STATUS_REG (0x1C)**

| UART_TXD | UART_RTSN | UART_DTRN | (reserved) | UART_ST_UTX_OUT | UART_TXFIFO_CNT | UART_RXD | UART_CTSN | UART_DSRN | (reserved) | UART_ST_URX_OUT | UART_RXFIFO_CNT | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 24 | 23 16 | 15 | 14 | 13 | 12 | 11 8 | 7 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 | 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 |

**UART_TXD** This bit represents the level of the internal UART RxD signal. (RO)

**UART_RTSN** This bit corresponds to the level of the internal UART CTS signal. (RO)

**UART_DTRN** This bit corresponds to the level of the internal UAR DSR signal. (RO)

**UART_ST_UTX_OUT** This register stores the state of the transmitter's finite state machine. 0: TX_IDLE; 1: TX_STRT; 2: TX_DAT0; 3: TX_DAT1; 4: TX_DAT2; 5: TX_DAT3; 6: TX_DAT4; 7: TX_DAT5; 8: TX_DAT6; 9: TX_DAT7; 10: TX_PRTY; 11: TX_STP1; 12: TX_STP2; 13: TX_DL0; 14: TX_DL1. (RO)

**UART_TXFIFO_CNT** (tx_mem_cnt, txfifo_cnt) stores the number of bytes of valid data in transmit-FIFO. tx_mem_cnt stores the three most significant bits, txfifo_cnt stores the eight least significant bits. (RO)

**UART_RXD** This bit corresponds to the level of the internal UART RxD signal. (RO)

**UART_CTSN** This bit corresponds to the level of the internal UART CTS signal. (RO)

**UART_DSRN** This bit corresponds to the level of the internal UAR DSR signal. (RO)

**UART_ST_URX_OUT** This register stores the value of the receiver's finite state machine. 0: RX_IDLE; 1: RX_STRT; 2: RX_DAT0; 3: RX_DAT1; 4: RX_DAT2; 5: RX_DAT3; 6: RX_DAT4; 7: RX_DAT5; 8: RX_DAT6; 9: RX_DAT7; 10: RX_PRTY; 11: RX_STP1; 12:RX_STP2; 13: RX_DL1. (RO)

**UART_RXFIFO_CNT** (rx_mem_cnt, rxfifo_cnt) stores the number of bytes of valid data in the receive-FIFO. rx_mem_cnt register stores the three most significant bits, rxfifo_cnt stores the eight least significant bits. (RO)

**Register 19.9. UART_CONF0_REG (0x20)**

| (reserved) | UART_TICK_REF_ALWAYS_ON | (reserved) | UART_DTR_INV | UART_RTS_INV | UART_TXD_INV | UART_DSR_INV | UART_CTS_INV | UART_RXD_INV | UART_TXFIFO_RST | UART_RXFIFO_RST | UART_IRDA_EN | UART_TX_FLOW_EN | UART_LOOPBACK | UART_IRDA_RX_INV | UART_IRDA_TX_INV | UART_IRDA_WCTL | UART_IRDA_TX_EN | UART_IRDA_DPLX | UART_TXD_BRK | UART_SW_DTR | UART_SW_RTS | UART_STOP_BIT_NUM | UART_BIT_NUM | UART_PARITY_EN | UART_PARITY | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | 1 | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | Reset |

**UART_TICK_REF_ALWAYS_ON** This register is used to select the clock; 1: APB clock; 0: REF_TICK. (R/W)

**UART_DTR_INV** Set this bit to invert the level of the UART DTR signal. (R/W)

**UART_RTS_INV** Set this bit to invert the level of the UART RTS signal. (R/W)

**UART_TXD_INV** Set this bit to invert the level of the UART TxD signal. (R/W)

**UART_DSR_INV** Set this bit to invert the level of the UART DSR signal. (R/W)

**UART_CTS_INV** Set this bit to invert the level of the UART CTS signal. (R/W)

**UART_RXD_INV** Set this bit to invert the level of the UART Rxd signal. (R/W)

**UART_TXFIFO_RST** Set this bit to reset the UART transmit-FIFO. **NOTICE**: UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these two registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data. (R/W)

**UART_RXFIFO_RST** Set this bit to reset the UART receive-FIFO. **NOTICE**: UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these two registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data. (R/W)

**UART_IRDA_EN** Set this bit to enable the IrDA protocol. (R/W)

**UART_TX_FLOW_EN** Set this bit to enable the flow control function for the transmitter. (R/W)

**UART_LOOPBACK** Set this bit to enable the UART loopback test mode. (R/W)

**UART_IRDA_RX_INV** Set this bit to invert the level of the IrDA receiver. (R/W)

**UART_IRDA_TX_INV** Set this bit to invert the level of the IrDA transmitter. (R/W)

**UART_IRDA_WCTL** 1: The IrDA transmitter's 11th bit is the same as its 10th bit; 0: set IrDA transmitter's 11th bit to 0. (R/W)

**UART_IRDA_TX_EN** This is the start enable bit of the IrDA transmitter. (R/W)

**UART_IRDA_DPLX** Set this bit to enable the IrDA loopback mode. (R/W)

**UART_TXD_BRK** Set this bit to enable the transmitter to send NULL, when the process of sending data is completed. (R/W)

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

Register 19.9. UART_CONF0_REG (0x20)

Continued from the previous page...

**UART_SW_DTR** This register is used to configure the software DTR signal used in software flow control. (R/W)

**UART_SW_RTS** This bit is used in hardware flow control when UART_RX_FLOW_EN is 0. Set this bit to drive the RTS (or rtsn_out) signal low, and reset to drive the signal high. (R/W)

**UART_STOP_BIT_NUM** This register is used to set the length of the stop bit.
0: Invalid. No effect
1: 1 bit
2: 1.5 bits
3: 2 bits
(R/W)

**UART_BIT_NUM** This register is used to set the length of data; 0: 5 bits, 1: 6 bits, 2: 7 bits, 3: 8 bits. (R/W)

**UART_PARITY_EN** Set this bit to enable the UART parity check. (R/W)

**UART_PARITY** This register is used to configure the parity check mode; 0: even, 1: odd. (R/W)

**Register 19.10. UART_CONF1_REG (0x24)**

| UART_RX_TOUT_EN | UART_RX_TOUT_THRHD | UART_RX_FLOW_EN | UART_RX_FLOW_THRHD | (reserved) | UART_TXFIFO_EMPTY_THRHD | (reserved) | UART_RXFIFO_FULL_THRHD | |
|---|---|---|---|---|---|---|---|---|
| 31 | 30 24 | 23 | 22 16 | 15 | 14 8 | 7 | 6 0 | |
| 0 | 0 0 0 0 0 0 0 | 0 | 0x00 | 0 | 0x60 | 0 | 0x60 | Reset |

**UART_RX_TOUT_EN** This is the enable bit for the UART receive-timeout function. (R/W)

**UART_RX_TOUT_THRHD** This register is used to configure the UART receiver's timeout value when receiving a byte. When using APB_CLK as the clock source, the register counts by UART baud cycle multiplied by 8. When using REF_TICK as the clock source, the register counts by UART baud cycle * 8 * (REF_TICK frequency)/(APB_CLK frequency). (R/W)

**UART_RX_FLOW_EN** This is the flow enable bit of the UART receiver; 1: choose software flow control by configuring the sw_rts signal; 0: disable software flow control. (R/W)

**UART_RX_FLOW_THRHD** When UART_RX_FLOW_EN is 1 and the receiver gets more data than its threshold value, the receiver produces an rtsn_out signal that tells the transmitter to stop transferring data. The threshold value is (rx_flow_thrhd_h3, rx_flow_thrhd). (R/W)

**UART_TXFIFO_EMPTY_THRHD** When the data amount in transmit-FIFO is less than its threshold value, it will produce a TXFIFO_EMPTY_INT_RAW interrupt. The threshold value is (tx_mem_empty_thrhd, txfifo_empty_thrhd). (R/W)

**UART_RXFIFO_FULL_THRHD** When the receiver gets more data than its threshold value, the receiver will produce an RXFIFO_FULL_INT_RAW interrupt. The threshold value is (rx_flow_thrhd_h3, rxfifo_full_thrhd). (R/W)

**Register 19.11. UART_LOWPULSE_REG (0x28)**

| (reserved) | UART_LOWPULSE_MIN_CNT | |
|---|---|---|
| 31 20 | 19 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0xOFFFFF | Reset |

**UART_LOWPULSE_MIN_CNT** This register stores the value of the minimum duration of the low-level pulse. It is used in the baud rate detection process. (RO)

**Register 19.12. UART_HIGHPULSE_REG (0x2C)**

| (reserved) | UART_HIGHPULSE_MIN_CNT | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0FFFFF | Reset |

**UART_HIGHPULSE_MIN_CNT** This register stores the value of the minimum duration of the high level pulse. It is used in baud rate detection process. (RO)

**Register 19.13. UART_RXD_CNT_REG (0x30)**

| (reserved) | UART_RXD_EDGE_CNT | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x000 | Reset |

**UART_RXD_EDGE_CNT** This register stores the count of the RxD edge change. It is used in the baud rate detection process. (RO)

**Register 19.14. UART_FLOW_CONF_REG (0x34)**

| (reserved) | UART_SEND_XOFF | UART_SEND_XON | UART_FORCE_XOFF | UART_FORCE_XON | UART_XONOFF_DEL | UART_SW_FLOW_CON_EN | |
|---|---|---|---|---|---|---|---|
| 31 … 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 | 0 | Reset |

**UART_SEND_XOFF** Hardware auto-clear; set to 1 to send Xoff char. (R/W)

**UART_SEND_XON** Hardware auto-clear; set to 1 to send Xon char. (R/W)

**UART_FORCE_XOFF** Set this bit to set the internal CTSn and stop the transmitter from sending data. (R/W)

**UART_FORCE_XON** Set this bit to clear the internal CTSn and enable the transmitter to continue sending data. (R/W)

**UART_XONOFF_DEL** Set this bit to remove the flow-control char from the received data. (R/W)

**UART_SW_FLOW_CON_EN** Set this bit to enable software flow control. It is used with register sw_xon or sw_xoff. (R/W)

**Register 19.15. UART_SLEEP_CONF_REG (0x38)**

| (reserved) | UART_ACTIVE_THRESHOLD | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0F0 | Reset |

**UART_ACTIVE_THRESHOLD** When the number of positive edges of RxD signal is larger than or equal to (UART_ACTIVE_THRESHOLD+2), the system emerges from Light-sleep mode and becomes active. (R/W)

**Register 19.16. UART_SWFC_CONF_REG (0x3C)**

| UART_XOFF_CHAR | UART_XON_CHAR | UART_XOFF_THRESHOLD | UART_XON_THRESHOLD | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x013 | 0x011 | 0x0E0 | 0x000 | Reset |

**UART_XOFF_CHAR** This register stores the Xoff flow control char. (R/W)

**UART_XON_CHAR** This register stores the Xon flow control char. (R/W)

**UART_XOFF_THRESHOLD** When the data amount in receive-FIFO is more than what this register indicates, it will send an Xoff char, with uart_sw_flow_con_en set to 1. (R/W)

**UART_XON_THRESHOLD** When the data amount in receive-FIFO is less than what this register indicates, it will send an Xon char, with uart_sw_flow_con_en set to 1. (R/W)

**Register 19.17. UART_IDLE_CONF_REG (0x40)**

| (reserved) | UART_TX_BRK_NUM | UART_TX_IDLE_NUM | UART_RX_IDLE_THRHD | |
|---|---|---|---|---|
| 31 … 28 | 27 … 20 | 19 … 10 | 9 … 0 | |
| 0 0 0 0 | 0x00A | 0x100 | 0x100 | Reset |

**UART_TX_BRK_NUM** This register is used to configure the number of zeros (0) sent, after the process of sending data is completed. It is active when txd_brk is set to 1. (R/W)

**UART_TX_IDLE_NUM** This register is used to configure the duration between transfers. (R/W)

**UART_RX_IDLE_THRHD** When the receiver takes more time to receive Byte data than what this register indicates, it will produce a frame-end signal. (R/W)

**Register 19.18. UART_RS485_CONF_REG (0x44)**

| (reserved) | UART_RS485_TX_DLY_NUM | UART_RS485_RX_DLY_NUM | UART_RS485RXBY_TX_EN | UART_RS485TX_RX_EN | UART_DL1_EN | UART_DL0_EN | UART_RS485_EN | |
|---|---|---|---|---|---|---|---|---|
| 31 … 10 | 9 … 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 |

**UART_RS485_TX_DLY_NUM** This register is used to delay the transmitter's internal data signal. (R/W)

**UART_RS485_RX_DLY_NUM** This register is used to delay the receiver's internal data signal. (R/W)

**UART_RS485RXBY_TX_EN** 1: enable the RS-485 transmitter to send data, when the RS-485 receiver line is busy; 0: the RS-485 transmitter should not send data, when its receiver is busy. (R/W)

**UART_RS485TX_RX_EN** Set this bit to enable the transmitter's output signal loop back to the receiver's input signal. (R/W)

**UART_DL1_EN** Set this bit to delay the STOP bit by 1 bit. (R/W)

**UART_DL0_EN** Set this bit to delay the STOP bit by 1 bit after DL1. (R/W)

**UART_RS485_EN** Set this bit to choose the RS-485 mode. (R/W)

**Register 19.19. UART_AT_CMD_PRECNT_REG (0x48)**

| (reserved) | UART_PRE_IDLE_NUM | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x0186A00 | Reset |

**UART_PRE_IDLE_NUM** This register is used to configure the idle-time duration before the first at_cmd is received by the receiver. When the duration is less than what this register indicates, it will not take the next data received as an at_cmd char. (R/W)

**Register 19.20. UART_AT_CMD_POSTCNT_REG (0x4c)**

| (reserved) | UART_POST_IDLE_NUM | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x0186A00 | Reset |

**UART_POST_IDLE_NUM** This register is used to configure the duration between the last at_cmd and the next data. When the duration is less than what this register indicates, it will not take the previous data as an at_cmd char. (R/W)

**Register 19.21. UART_AT_CMD_GAPTOUT_REG (0x50)**

| (reserved) | UART_RX_GAP_TOUT | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x0001E00 | Reset |

**UART_RX_GAP_TOUT** This register is used to configure the interval between the at_cmd chars. When the interval is greater than the value of this register, it will not take the data as continuous at_cmd chars. The register should be configured to more than half of the baud rate. (R/W)

**Register 19.22. UART_AT_CMD_CHAR_REG (0x54)**

| (reserved) | UART_CHAR_NUM | UART_AT_CMD_CHAR | |
|---|---|---|---|
| 31 … 16 | 15 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x003 | 0x02B | Reset |

**UART_CHAR_NUM** This register is used to configure the number of continuous at_cmd chars received by the receiver. (R/W)

**UART_AT_CMD_CHAR** This register is used to configure the content of an at_cmd char. (R/W)

**Register 19.23. UART_MEM_CONF_REG (0x58)**

| (reserved) | UART_TX_MEM_EMPTY_THRHD | UART_RX_MEM_FULL_THRHD | UART_XOFF_THRESHOLD_H2 | UART_XON_THRESHOLD_H2 | UART_RX_TOUT_THRHD_H3 | UART_RX_FLOW_THRHD_H3 | (reserved) | UART_TX_SIZE | UART_RX_SIZE | (reserved) | UART_MEM_PD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 28 | 27 25 | 24 23 | 22 21 | 20 18 | 17 15 | 14 11 | 10 7 | 6 3 | 2 1 | 0 | |
| 0 | 0x0 | 0x0 | 0x0 | 0x0 | 0x0 | 0x0 | 0 0 0 0 | 0x01 | 0x01 | 0 0 | 0 | Reset |

**UART_TX_MEM_EMPTY_THRHD** Refer to the description of TXFIFO_EMPTY_THRHD. (R/W)

**UART_RX_MEM_FULL_THRHD** Refer to the description of RXFIFO_FULL_THRHD. (R/W)

**UART_XOFF_THRESHOLD_H2** Refer to the description of UART_XOFF_THRESHOLD. (R/W)

**UART_XON_THRESHOLD_H2** Refer to the description of UART_XON_THRESHOLD. (R/W)

**UART_RX_TOUT_THRHD_H3** Refer to the description of RX_TOUT_THRHD. (R/W)

**UART_RX_FLOW_THRHD_H3** Refer to the description of RX_FLOW_THRHD. (R/W)

**UART_TX_SIZE** This register is used to configure the amount of memory allocated to the transmit-FIFO. The default number is 128 bytes. (R/W)

**UART_RX_SIZE** This register is used to configure the amount of memory allocated to the receive-FIFO. The default number is 128 bytes. (R/W)

**UART_MEM_PD** Set this bit to power down the memory. When the reg_mem_pd register is set to 1 for all UART controllers, Memory will enter the low-power mode. (R/W)

**Register 19.24. UART_MEM_TX_STATUS_REG (0x5c)**

| (reserved) | UART_MEM_TX_WR_ADDR | UART_MEM_TX_RD_ADDR | (reserved) | |
|---|---|---|---|---|
| 31 24 | 23 13 | 12 2 | 1 0 | |
| 0 | 0 | 0 | 0 | Reset |

**UART_MEM_TX_WR_ADDR** Represents the offset address to write TX FIFO. (RO)

**UART_MEM_TX_RD_ADDR** Represents the offset address to read TX FIFO. (RO)

**Register 19.25. UART_MEM_RX_STATUS_REG (0x0060)**

| (reserved) | UART_MEM_RX_WR_ADDR | UART_MEM_RX_RD_ADDR | (reserved) | |
|---|---|---|---|---|
| 31 – 24 | 23 – 13 | 12 – 2 | 1 – 0 | |
| 0 | 0 | 0 | 0 | Reset |

**UART_MEM_RX_RD_ADDR** Represents the offset address to read RX FIFO. (RO)

**UART_MEM_RX_WR_ADDR** Represents the offset address to write RX FIFO. (RO)

**Register 19.26. UART_MEM_CNT_STATUS_REG (0x64)**

| (reserved) | UART_TX_MEM_CNT | UART_RX_MEM_CNT | |
|---|---|---|---|
| 31 – 6 | 5 – 3 | 2 – 0 | |
| 0 0 0 0 0 0 0 0 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 |

**UART_TX_MEM_CNT** Refer to the description of TXFIFO_CNT. (RO)

**UART_RX_MEM_CNT** Refer to the description of RXFIFO_CNT. (RO)

**Register 19.27. UART_POSPULSE_REG (0x68)**

| (reserved) | UART_POSEDGE_MIN_CNT | |
|---|---|---|
| 31 – 20 | 19 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0xOFFFFF | Reset |

**UART_POSEDGE_MIN_CNT** This register stores the count of RxD positive edges. It is used in the autobaud detection process. (RO)

Register 19.28. UART_NEGPULSE_REG (0x6c)

| (reserved) | UART_NEGEDGE_MIN_CNT | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0FFFFF | Reset |

**UART_NEGEDGE_MIN_CNT** This register stores the count of RxD negative edges. It is used in the autobaud detection process. (RO)

## 19.5.2 UHCI Registers

The addresses in this section are relative to the UDMA base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 19.4.2 *UHCI Register Summary*.

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

Register 19.29. UHCI_CONF0_REG (0x0)

| (reserved) | UHCI_ENCODE_CRC_EN | UHCI_LEN_EOF_EN | UHCI_UART_IDLE_EOF_EN | UHCI_CRC_REC_EN | UHCI_HEAD_EN | UHCI_SEPER_EN | (reserved) | UHCI_UART2_CE | UHCI_UART1_CE | UHCI_UART0_CE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 … 12 | 11 | 10 | 9 | 8 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 0 0 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 | Reset |

**UHCI_ENCODE_CRC_EN** Reserved. Please initialize it to 0. (R/W)

**UHCI_LEN_EOF_EN** Reserved. Please initialize it to 0. (R/W)

**UHCI_UART_IDLE_EOF_EN** Reserved. Please initialize it to 0. (R/W)

**UHCI_CRC_REC_EN** Reserved. Please initialize it to 0. (R/W)

**UHCI_HEAD_EN** Reserved. Please initialize it to 0. (R/W)

**UHCI_SEPER_EN** Set this bit to use a special char and separate the data frame. (R/W)

**UHCI_UART2_CE** Set this bit to use UART2 and transmit or receive data. (R/W)

**UHCI_UART1_CE** Set this bit to use UART1 and transmit or receive data. (R/W)

**UHCI_UART0_CE** Set this bit to use UART and transmit or receive data. (R/W)

**Register 19.30. UHCI_INT_RAW_REG (0x4)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | UHCI_OUT_TOTAL_EOF_INT_RAW | UHCI_OUTLINK_EOF_ERR_INT_RAW | UHCI_IN_DSCR_EMPTY_INT_RAW | UHCI_OUT_DSCR_ERR_INT_RAW | UHCI_IN_DSCR_ERR_INT_RAW | UHCI_OUT_EOF_INT_RAW | UHCI_OUT_DONE_INT_RAW | UHCI_IN_ERR_EOF_INT_RAW | UHCI_IN_SUC_EOF_INT_RAW | UHCI_IN_DONE_INT_RAW | UHCI_TX_HUNG_INT_RAW | UHCI_RX_HUNG_INT_RAW | UHCI_TX_START_INT_RAW | UHCI_RX_START_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UHCI_OUT_TOTAL_EOF_INT_RAW** The raw interrupt status bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (RO)

**UHCI_OUTLINK_EOF_ERR_INT_RAW** The raw interrupt status bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (RO)

**UHCI_IN_DSCR_EMPTY_INT_RAW** The raw interrupt status bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (RO)

**UHCI_OUT_DSCR_ERR_INT_RAW** The raw interrupt status bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (RO)

**UHCI_IN_DSCR_ERR_INT_RAW** The raw interrupt status bit for the UHCI_IN_DSCR_ERR_INT interrupt. (RO)

**UHCI_OUT_EOF_INT_RAW** The raw interrupt status bit for the UHCI_OUT_EOF_INT interrupt. (RO)

**UHCI_OUT_DONE_INT_RAW** The raw interrupt status bit for the UHCI_OUT_DONE_INT interrupt. (RO)

**UHCI_IN_ERR_EOF_INT_RAW** The raw interrupt status bit for the UHCI_IN_ERR_EOF_INT interrupt. (RO)

**UHCI_IN_SUC_EOF_INT_RAW** The raw interrupt status bit for the UHCI_IN_SUC_EOF_INT interrupt. (RO)

**UHCI_IN_DONE_INT_RAW** The raw interrupt status bit for the UHCI_IN_DONE_INT interrupt. (RO)

**UHCI_TX_HUNG_INT_RAW** The raw interrupt status bit for the UHCI_TX_HUNG_INT interrupt. (RO)

**UHCI_RX_HUNG_INT_RAW** The raw interrupt status bit for the UHCI_RX_HUNG_INT interrupt. (RO)

**UHCI_TX_START_INT_RAW** The raw interrupt status bit for the UHCI_TX_START_INT interrupt. (RO)

**UHCI_RX_START_INT_RAW** The raw interrupt status bit for the UHCI_RX_START_INT interrupt. (RO)

**Register 19.31. UHCI_INT_ST_REG (0x8)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | UHCI_DMA_INFIFO_FULL_WM_INT_ST | UHCI_SEND_A_REG_Q_INT_ST | UHCI_SEND_S_REG_Q_INT_ST | UHCI_OUT_TOTAL_EOF_INT_ST | UHCI_OUTLINK_EOF_ERR_INT_ST | UHCI_IN_DSCR_EMPTY_INT_ST | UHCI_OUT_DSCR_ERR_INT_ST | UHCI_IN_DSCR_ERR_INT_ST | UHCI_OUT_EOF_INT_ST | UHCI_OUT_DONE_INT_ST | UHCI_IN_ERR_EOF_INT_ST | UHCI_IN_SUC_EOF_INT_ST | UHCI_IN_DONE_INT_ST | UHCI_TX_HUNG_INT_ST | UHCI_RX_HUNG_INT_ST | UHCI_TX_START_INT_ST | UHCI_RX_START_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UHCI_SEND_A_REG_Q_INT_ST** The masked interrupt status bit for the UHCI_SEND_A_REG_Q_INT interrupt. (RO)

**UHCI_SEND_S_REG_Q_INT_ST** The masked interrupt status bit for the UHCI_SEND_S_REG_Q_INT interrupt. (RO)

**UHCI_OUT_TOTAL_EOF_INT_ST** The masked interrupt status bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (RO)

**UHCI_OUTLINK_EOF_ERR_INT_ST** The masked interrupt status bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (RO)

**UHCI_IN_DSCR_EMPTY_INT_ST** The masked interrupt status bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (RO)

**UHCI_OUT_DSCR_ERR_INT_ST** The masked interrupt status bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (RO)

**UHCI_IN_DSCR_ERR_INT_ST** The masked interrupt status bit for the UHCI_IN_DSCR_ERR_INT interrupt. (RO)

**UHCI_OUT_EOF_INT_ST** The masked interrupt status bit for the UHCI_OUT_EOF_INT interrupt. (RO)

**UHCI_OUT_DONE_INT_ST** The masked interrupt status bit for the UHCI_OUT_DONE_INT interrupt. (RO)

**UHCI_IN_ERR_EOF_INT_ST** The masked interrupt status bit for the UHCI_IN_ERR_EOF_INT interrupt. (RO)

**UHCI_IN_SUC_EOF_INT_ST** The masked interrupt status bit for the UHCI_IN_SUC_EOF_INT interrupt. (RO)

**UHCI_IN_DONE_INT_ST** The masked interrupt status bit for the UHCI_IN_DONE_INT interrupt. (RO)

**UHCI_TX_HUNG_INT_ST** The masked interrupt status bit for the UHCI_TX_HUNG_INT interrupt. (RO)

**UHCI_RX_HUNG_INT_ST** The masked interrupt status bit for the UHCI_RX_HUNG_INT interrupt. (RO)

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

**Register 19.31. UHCI_INT_ST_REG (0x8)**

**Continued from the previous page...**

**UHCI_TX_START_INT_ST** The masked interrupt status bit for the UHCI_TX_START_INT interrupt. (RO)

**UHCI_RX_START_INT_ST** The masked interrupt status bit for the UHCI_RX_START_INT interrupt. (RO)

**Register 19.32. UHCI_INT_ENA_REG (0xC)**

| 31 | | | | | | | | | | | | | | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | UHCI_DMA_INFIFO_FULL_WM_INT_ENA | UHCI_SEND_A_REG_Q_INT_ENA | UHCI_SEND_S_REG_Q_INT_ENA | UHCI_OUT_TOTAL_EOF_INT_ENA | UHCI_OUTLINK_EOF_ERR_INT_ENA | UHCI_IN_DSCR_EMPTY_INT_ENA | UHCI_OUT_DSCR_ERR_INT_ENA | UHCI_IN_DSCR_ERR_INT_ENA | UHCI_OUT_EOF_INT_ENA | UHCI_OUT_DONE_INT_ENA | UHCI_IN_ERR_EOF_INT_ENA | UHCI_IN_SUC_EOF_INT_ENA | UHCI_IN_DONE_INT_ENA | UHCI_TX_HUNG_INT_ENA | UHCI_RX_HUNG_INT_ENA | UHCI_TX_START_INT_ENA | UHCI_RX_START_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**UHCI_SEND_A_REG_Q_INT_ENA** The interrupt enable bit for the UHCI_SEND_A_REG_Q_INT interrupt. (R/W)

**UHCI_SEND_S_REG_Q_INT_ENA** The interrupt enable bit for the UHCI_SEND_S_REG_Q_INT interrupt. (R/W)

**UHCI_OUT_TOTAL_EOF_INT_ENA** The interrupt enable bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (R/W)

**UHCI_OUTLINK_EOF_ERR_INT_ENA** The interrupt enable bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (R/W)

**UHCI_IN_DSCR_EMPTY_INT_ENA** The interrupt enable bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (R/W)

**UHCI_OUT_DSCR_ERR_INT_ENA** The interrupt enable bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (R/W)

**UHCI_IN_DSCR_ERR_INT_ENA** The interrupt enable bit for the UHCI_IN_DSCR_ERR_INT interrupt. (R/W)

**UHCI_OUT_EOF_INT_ENA** The interrupt enable bit for the UHCI_OUT_EOF_INT interrupt. (R/W)

**UHCI_OUT_DONE_INT_ENA** The interrupt enable bit for the UHCI_OUT_DONE_INT interrupt. (R/W)

**UHCI_IN_ERR_EOF_INT_ENA** The interrupt enable bit for the UHCI_IN_ERR_EOF_INT interrupt. (R/W)

**UHCI_IN_SUC_EOF_INT_ENA** The interrupt enable bit for the UHCI_IN_SUC_EOF_INT interrupt. (R/W)

**UHCI_IN_DONE_INT_ENA** The interrupt enable bit for the UHCI_IN_DONE_INT interrupt. (R/W)

**UHCI_TX_HUNG_INT_ENA** The interrupt enable bit for the UHCI_TX_HUNG_INT interrupt. (R/W)

**UHCI_RX_HUNG_INT_ENA** The interrupt enable bit for the UHCI_RX_HUNG_INT interrupt. (R/W)

**UHCI_TX_START_INT_ENA** The interrupt enable bit for the UHCI_TX_START_INT interrupt. (R/W)

**UHCI_RX_START_INT_ENA** The interrupt enable bit for the UHCI_RX_START_INT interrupt. (R/W)

**Register 19.33. UHCI_INT_CLR_REG (0x10)**

| 31 | | | | | | | | | | | | | | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | UHCI_DMA_INFIFO_FULL_WM_INT_CLR | UHCI_SEND_A_REG_Q_INT_CLR | UHCI_SEND_S_REG_Q_INT_CLR | UHCI_OUT_TOTAL_EOF_INT_CLR | UHCI_OUTLINK_EOF_ERR_INT_CLR | UHCI_IN_DSCR_EMPTY_INT_CLR | UHCI_OUT_DSCR_ERR_INT_CLR | UHCI_IN_DSCR_ERR_INT_CLR | UHCI_OUT_EOF_INT_CLR | UHCI_OUT_DONE_INT_CLR | UHCI_IN_ERR_EOF_INT_CLR | UHCI_IN_SUC_EOF_INT_CLR | UHCI_IN_DONE_INT_CLR | UHCI_TX_HUNG_INT_CLR | UHCI_RX_HUNG_INT_CLR | UHCI_TX_START_INT_CLR | UHCI_RX_START_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UHCI_SEND_A_REG_Q_INT_CLR** Set this bit to clear the UHCI_SEND_A_REG_Q_INT interrupt. (WO)

**UHCI_SEND_S_REG_Q_INT_CLR** Set this bit to clear the UHCI_SEND_S_REG_Q_INT interrupt. (WO)

**UHCI_OUT_TOTAL_EOF_INT_CLR** Set this bit to clear the UHCI_OUT_TOTAL_EOF_INT interrupt. (WO)

**UHCI_OUTLINK_EOF_ERR_INT_CLR** Set this bit to clear the UHCI_OUTLINK_EOF_ERR_INT interrupt. (WO)

**UHCI_IN_DSCR_EMPTY_INT_CLR** Set this bit to clear the UHCI_IN_DSCR_EMPTY_INT interrupt. (WO)

**UHCI_OUT_DSCR_ERR_INT_CLR** Set this bit to clear the UHCI_OUT_DSCR_ERR_INT interrupt. (WO)

**UHCI_IN_DSCR_ERR_INT_CLR** Set this bit to clear the UHCI_IN_DSCR_ERR_INT interrupt. (WO)

**UHCI_OUT_EOF_INT_CLR** Set this bit to clear the UHCI_OUT_EOF_INT interrupt. (WO)

**UHCI_OUT_DONE_INT_CLR** Set this bit to clear the UHCI_OUT_DONE_INT interrupt. (WO)

**UHCI_IN_ERR_EOF_INT_CLR** Set this bit to clear the UHCI_IN_ERR_EOF_INT interrupt. (WO)

**UHCI_IN_SUC_EOF_INT_CLR** Set this bit to clear the UHCI_IN_SUC_EOF_INT interrupt. (WO)

**UHCI_IN_DONE_INT_CLR** Set this bit to clear the UHCI_IN_DONE_INT interrupt. (WO)

**UHCI_TX_HUNG_INT_CLR** Set this bit to clear the UHCI_TX_HUNG_INT interrupt. (WO)

**UHCI_RX_HUNG_INT_CLR** Set this bit to clear the UHCI_RX_HUNG_INT interrupt. (WO)

**UHCI_TX_START_INT_CLR** Set this bit to clear the UHCI_TX_START_INT interrupt. (WO)

**UHCI_RX_START_INT_CLR** Set this bit to clear the UHCI_RX_START_INT interrupt. (WO)

**Register 19.34. UHCI_DMA_OUT_STATUS_REG (0x14)**

| (reserved) | UHCI_OUT_EMPTY | UHCI_OUT_FULL | |
|---|---|---|---|
| 31 … 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 | 1 | 0 | Reset |

**UHCI_OUT_EMPTY** 1: DMA inlink descriptor's FIFO is empty. (RO)

**UHCI_OUT_FULL** 1: DMA outlink descriptor's FIFO is full. (RO)

**Register 19.35. UHCI_DMA_OUT_PUSH_REG (0x18)**

| (reserved) | UHCI_OUTFIFO_PUSH | (reserved) | UHCI_OUTFIFO_WDATA | |
|---|---|---|---|---|
| 31 … 17 | 16 | 15 … 9 | 8 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 0 0 0 0 0 0 | 0x000 | Reset |

**UHCI_OUTFIFO_PUSH** Set this bit to push data into DMA FIFO. (R/W)

**UHCI_OUTFIFO_WDATA** This is the data that need to be pushed into DMA FIFO. (R/W)

**Register 19.36. UHCI_DMA_IN_POP_REG (0x20)**

| (reserved) | UHCI_INFIFO_POP | (reserved) | UHCI_INFIFO_RDATA | |
|---|---|---|---|---|
| 31 … 17 | 16 | 15 … 12 | 11 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 0 0 0 | 0x0000 | Reset |

**UHCI_INFIFO_POP** Set this bit to pop data from DMA FIFO. (R/W)

**UHCI_INFIFO_RDATA** This register stores the data popping from DMA FIFO. (RO)

**Register 19.37. UHCI_DMA_OUT_LINK_REG (0x24)**

| UHCI_OUTLINK_PARK | UHCI_OUTLINK_RESTART | UHCI_OUTLINK_START | UHCI_OUTLINK_STOP | (reserved) | UHCI_OUTLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**UHCI_OUTLINK_PARK** 1: the outlink descriptor's FSM is in idle state; 0: the outlink descriptor's FSM is working. (RO)

**UHCI_OUTLINK_RESTART** Set this bit to restart the outlink descriptor from the last address. (R/W)

**UHCI_OUTLINK_START** Set this bit to start a new outlink descriptor. (R/W)

**UHCI_OUTLINK_STOP** Set this bit to stop dealing with the outlink descriptor. (R/W)

**UHCI_OUTLINK_ADDR** This register stores the least significant 20 bits of the first outlink descriptor's address. (R/W)

**Register 19.38. UHCI_DMA_IN_LINK_REG (0x28)**

| UHCI_INLINK_PARK | UHCI_INLINK_RESTART | UHCI_INLINK_START | UHCI_INLINK_STOP | (reserved) | UHCI_INLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**UHCI_INLINK_PARK** 1: the inlink descriptor's FSM is in idle state; 0: the inlink descriptor's FSM is working. (RO)

**UHCI_INLINK_RESTART** Set this bit to mount new inlink descriptors. (R/W)

**UHCI_INLINK_START** Set this bit to start dealing with the inlink descriptors. (R/W)

**UHCI_INLINK_STOP** Set this bit to stop dealing with the inlink descriptors. (R/W)

**UHCI_INLINK_ADDR** This register stores the 20 least significant bits of the first inlink descriptor's address. (R/W)

### Register 19.39. UHCI_CONF1_REG (0x2C)

| 31 – 6 | 5 | 4 | 3 – 2 | 1 | 0 | |
|---|---|---|---|---|---|---|
| (reserved) | UHCI_TX_ACK_NUM_RE | UHCI_TX_CHECK_SUM_RE | (reserved) | UHCI_CHECK_SEQ_EN | UHCI_CHECK_SUM_EN | |
| 0 0 0 0 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 | 1 | 0 0 | 1 | 1 | Reset |

**UHCI_TX_ACK_NUM_RE** Reserved. Please initialize to 0. (R/W)

**UHCI_TX_CHECK_SUM_RE** Reserved. Please initialize to 0. (R/W)

**UHCI_CHECK_SEQ_EN** Reserved. Please initialize to 0. (R/W)

**UHCI_CHECK_SUM_EN** Reserved. Please initialize to 0. (R/W)

### Register 19.40. UHCI_DMA_OUT_EOF_DES_ADDR_REG (0x38)

| 31 – 0 | |
|---|---|
| 0x000000000 | Reset |

**UHCI_DMA_OUT_EOF_DES_ADDR_REG** This register stores the address of the outlink descriptor when the EOF bit in this descriptor is 1. (RO)

### Register 19.41. UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG (0x3C)

| 31 – 0 | |
|---|---|
| 0x000000000 | Reset |

**UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG** This register stores the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

### Register 19.42. UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG (0x40)

| 31 – 0 | |
|---|---|
| 0x000000000 | Reset |

**UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG** This register stores the address of the inlink descriptor when there are some errors in this descriptor. (RO)

### Register 19.43. UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x44)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG** This register stores the address of the outlink descriptor when there are some errors in this descriptor. (RO)

### Register 19.44. UHCI_DMA_IN_DSCR_REG (0x4C)

| 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 |

**UHCI_DMA_IN_DSCR_REG** The address of the current inlink descriptor *x*. (RO)

### Register 19.45. UHCI_DMA_IN_DSCR_BF0_REG (0x50)

| 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 |

**UHCI_DMA_IN_DSCR_BF0_REG** The address of the last inlink descriptor *x-1*. (RO)

### Register 19.46. UHCI_DMA_IN_DSCR_BF1_REG (0x54)

| 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 |

**UHCI_DMA_IN_DSCR_BF1_REG** The address of the second-to-last inlink descriptor *x-2*. (RO)

### Register 19.47. UHCI_DMA_OUT_DSCR_REG (0x58)

| 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 |

**UHCI_DMA_OUT_DSCR_REG** The address of the current outlink descriptor *y*. (RO)

### Register 19.48. UHCI_DMA_OUT_DSCR_BF0_REG (0x5C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UHCI_DMA_OUT_DSCR_BF0_REG** The address of the last outlink descriptor *y-1*. (RO)

### Register 19.49. UHCI_DMA_OUT_DSCR_BF1_REG (0x60)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**UHCI_DMA_OUT_DSCR_BF1_REG** The address of the second-to-last outlink descriptor *y-2*. (RO)

### Register 19.50. UHCI_ESCAPE_CONF_REG (0x64)

| | (reserved) | | | | | | | | | | | | | | | | | | | | | | | | UHCI_RX_13_ESC_EN | UHCI_RX_11_ESC_EN | UHCI_RX_DB_ESC_EN | UHCI_RX_C0_ESC_EN | UHCI_TX_13_ESC_EN | UHCI_TX_11_ESC_EN | UHCI_TX_DB_ESC_EN | UHCI_TX_C0_ESC_EN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | | 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 | 1 | 1 | 0 | 0 | 1 | 1 | Reset |

**UHCI_RX_13_ESC_EN** Set this bit to enable replacing flow control char 0x13, when DMA sends data. (R/W)

**UHCI_RX_11_ESC_EN** Set this bit to enable replacing flow control char 0x11, when DMA sends data. (R/W)

**UHCI_RX_DB_ESC_EN** Set this bit to enable replacing 0xdb char, when DMA sends data. (R/W)

**UHCI_RX_C0_ESC_EN** Set this bit to enable replacing 0xc0 char, when DMA sends data. (R/W)

**UHCI_TX_13_ESC_EN** Set this bit to enable decoding flow control char 0x13, when DMA receives data. (R/W)

**UHCI_TX_11_ESC_EN** Set this bit to enable decoding flow control char 0x11, when DMA receives data. (R/W)

**UHCI_TX_DB_ESC_EN** Set this bit to enable decoding 0xdb char, when DMA receives data. (R/W)

**UHCI_TX_C0_ESC_EN** Set this bit to enable decoding 0xc0 char, when DMA receives data. (R/W)

**Register 19.51. UHCI_HUNG_CONF_REG (0x68)**

| (reserved) | UHCI_RXFIFO_TIMEOUT_ENA | UHCI_RXFIFO_TIMEOUT_SHIFT | UHCI_RXFIFO_TIMEOUT | UHCI_TXFIFO_TIMEOUT_ENA | UHCI_TXFIFO_TIMEOUT_SHIFT | UHCI_TXFIFO_TIMEOUT | |
|---|---|---|---|---|---|---|---|
| 31 24 | 23 | 22 20 | 19 12 | 11 | 10 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 | 1 | 0 0 0 | 0x010 | 1 | 0 0 0 | 0x010 | Reset |

**UHCI_RXFIFO_TIMEOUT_ENA** This is the enable bit for DMA send-data timeout. (R/W)

**UHCI_RXFIFO_TIMEOUT_SHIFT** The tick count is cleared when its value is equal to or greater than (17'd8000»reg_rxfifo_timeout_shift). (R/W)

**UHCI_RXFIFO_TIMEOUT** This register stores the timeout value. When DMA takes more time to read data from RAM than what this register indicates, it will produce the UHCI_RX_HUNG_INT interrupt. (R/W)

**UHCI_TXFIFO_TIMEOUT_ENA** The enable bit for Tx FIFO receive-data timeout (R/W)

**UHCI_TXFIFO_TIMEOUT_SHIFT** The tick count is cleared when its value is equal to or greater than (17'd8000»reg_txfifo_timeout_shift). (R/W)

**UHCI_TXFIFO_TIMEOUT** This register stores the timeout value. When DMA takes more time to receive data than what this register indicates, it will produce the UHCI_TX_HUNG_INT interrupt. (R/W)

**Register 19.52. UHCI_ESC_CONF*n*_REG (*n*: 0-3) (0xB0+4**n*)**

| (reserved) | UHCI_ESC_SEQ2_CHAR1 | UHCI_ESC_SEQ2_CHAR0 | UHCI_ESC_SEQ2 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 | 0x0DF | 0x0DB | 0x013 | Reset |

**UHCI_ESC_SEQ2_CHAR1** This register stores the second char used to replace the reg_esc_seq2 in data. (R/W)

**UHCI_ESC_SEQ2_CHAR0** This register stores the first char used to replace the reg_esc_seq2 in data. (R/W)

**UHCI_ESC_SEQ2** This register stores the flow_control char to turn off the flow_control. (R/W)

# Chapter 20

# SPI Controller (SPI)

## 20.1 Overview

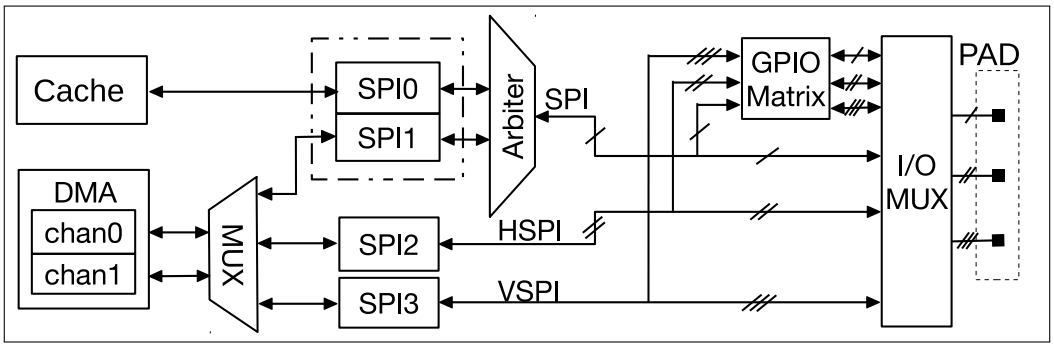



Figure 20.1-1. SPI Architecture

As Figure 20.1-1 shows, ESP32 integrates four SPI controllers which can be used to communicate with external devices that use the SPI protocol. Controller SPI0 is used as a buffer for accessing external memory. Controller SPI1 can be used as a master. Controllers SPI2 and SPI3 can be configured as either a master or a slave. When used as a master, each SPI controller can drive multiple CS signals (CS0~CS2) to activate multiple slaves. Controllers SPI1~SPI3 share two DMA channels.

The SPI signal buses consist of D, Q, CS0-CS2, CLK, WP, and HD signals, as Table 20.1-1 shows. Controllers SPI0 and SPI1 share one signal bus through an arbiter; the signals of the shared bus start with "SPI". Controllers SPI2 and SPI3 use signal buses starting with "HSPI" and "VSPI" respectively. The I/O lines included in the above-mentioned signal buses can be mapped to pins via either the IO_MUX module or the GPIO matrix. (Please refer to Chapter IO_MUX for details.)

The SPI controller supports four-line full-duplex/half-duplex communication (MOSI, MISO, CS, and CLK lines) and three-line half-duplex-only communication (DATA, CS, and CLK lines) in GP-SPI mode. In QSPI mode, an SPI controller accesses the flash or SRAM by using signal buses D, Q, CS0~CS2, CLK, WP, and HD as a four-bit parallel SPI bus. The mapping between SPI bus signals and pin function signals under different communication modes is shown in Table 20.1-1.

Table 20.1-1. Mapping Between SPI Bus Signals and Pin Function Signals

| Four-line GP-SPI Full-duplex/half-duplex signal bus | Three-line GP-SPI Half-duplex signal bus | QSPI Signal bus | Pin function signals | | |
|---|---|---|---|---|---|
| | | | SPI signal bus | HSPI signal bus | VSPI signal bus |
| MOSI | DATA | D | SPID | HSPID | VSPID |
| MISO | - | Q | SPIQ | HSPIQ | VSPIQ |

| | | | | | |
|---|---|---|---|---|---|
| CS | CS | CS | SPICS0 | HSPICS0 | VSPICS0 |
| CLK | CLK | CLK | SPICLK | HSPICLK | VSPICLK |
| - | - | WP | SPIWP | HSPIWP | VSPIWP |
| - | - | HD | SPIHD | HSPIHD | VSPIHD |

## 20.2 SPI Features

**General Purpose SPI (GP-SPI)**

- Programmable data transfer length, in multiples of 1 byte
- Four-line full-duplex/half-duplex communication and three-line half-duplex communication support
- Master mode and slave mode
- Programmable CPOL and CPHA
- Programmable clock

**Parallel QSPI**

- Communication format support for specific slave devices such as flash
- Programmable communication format
- Six variations of flash-read operations available
- Automatic shift between flash and SRAM access
- Automatic wait states for flash access

**SPI DMA Support**

- Support for sending and receiving data using linked lists

**SPI Interrupt Hardware**

- SPI interrupts
- SPI DMA interrupts

## 20.3 GP-SPI

The SPI master mode supports four-line full-duplex/half-duplex communication and three-line half-duplex communication. Figure 20.3-1 outlines the connections needed for four-line full-duplex/half-duplex communications.

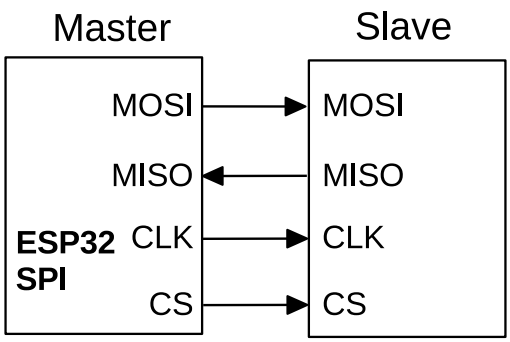



Figure 20.3-1. SPI Master and Slave Full-duplex/Half-duplex Communication

The SPI1~SPI3 controllers can communicate with other slaves as a standard SPI master. SPI2 and SPI3 can be configured as either a master or a slave. Every SPI master can be connected to three slaves at most by default. When not using DMA, the maximum length of data received/sent in one burst is 64 bytes. The data length is in multiples of one byte.

### 20.3.1 GP-SPI Four-line Full-duplex Communication

When configured to four-line full-duplex mode, the ESP32 SPI can act as either a master or a slave. The length of received and sent data needs to be set by configuring the SPI_MISO_DLEN_REG, SPI_MOSI_DLEN_REG registers for master mode as well as SPI_SLV_RDBUF_DLEN_REG, SPI_SLV_WRBUF_DLEN_REG registers for slave mode. The SPI_DOUTDIN bit and SPI_USR_MOSI bit in register SPI_USER_REG should be configured to enable this communication mode. The SPI_USR bit in register SPI_CMD_REG needs to be configured to initialize a data transfer.

### 20.3.2 GP-SPI Four-line Half-duplex Communication

When configured to four-line half-duplex mode, the ESP32 SPI can act as either a master or a slave. In this mode, the SPI communication supports flexible communication format as: command + address + dummy phase + received and/or sent data. The format is specified as follows:

1. command: length of 0~16 bits; Master Out Slave In (MOSI).
2. address: length of 0~32/64 bits; Master Out Slave In (MOSI).
3. dummy phase: length of 0~256 SPI clocks.
4. received and/or sent data: length of 0~512 bits (64 bytes); Master Out Slave In (MOSI) or Master In Slave Out (MISO).

The address length is up to 32 bits in GP-SPI master mode and 64 bits in QSPI master mode. The command phase, address phase, dummy phase and received/sent data phase are controlled by bits SPI_USR_COMMAND, SPI_USR_ADDR, SPI_USR_DUMMY, and SPI_USR_MISO/SPI_USR_MOSI respectively in register SPI_USER_REG. A certain phase is enabled only when its corresponding control bit is set to 1. Details can be found in register description. When SPI works as a master, the register can be configured by software as required to determine whether or not to enable a certain phase.

When SPI works as a slave, the communication format must contain command, address, received and/or sent data, among which the command has several options listed in Table 20.3-1. During data transmission or

Table 20.3-1. Command Definitions Supported by GP-SPI Slave in Half-duplex Mode

| Command | Description |
|---|---|
| 0x1 | Received by slave; writes data sent by the master into the slave status register via MOSI. |
| 0x2 | Received by slave; writes data sent by the master into the slave data buffer via MOSI. |
| 0x3 | Sent by slave; sends data in the slave buffer to master via MISO. |
| 0x4 | Sent by slave; sends data in the slave status register to master via MISO. |
| 0x6 | Writes master data on MOSI into data buffer and then sends the date in the slave data buffer to MISO. |

reception, the CS signal should keep logic level low. If the CS signal is pulled up during transmission, the internal state of the slave will be reset.

The master can write the slave status register SPI_SLV_WR_STATUS_REG, and decide whether to read data from register SPI_SLV_WR_STATUS_REG or register SPI_RD_STATUS_REG via the SPI_SLV_STATUS_READBACK bit in register SPI_SLAVE1_REG. The SPI master can maintain communication with the slave by reading and writing slave status register, thus realizing complex communication with ease.

The length of received and sent data is controlled by SPI_MISO_DLEN_REG and SPI_MOSI_DLEN_REG in master mode, as well as SPI_SLV_RDBUF_DLEN_REG and SPI_SLV_WRBUF_DLEN_REG in slave mode. A reception or transmission of data is controlled by bit SPI_USR_MOSI or SPI_USR_MISO in SPI_USER_REG. The SPI_USR bit in register SPI_CMD_REG needs to be configured to initialize a data transfer.

### 20.3.3 GP-SPI Three-line Half-duplex Communication

The three-line half-duplex communication differs from four-line half-duplex communication in that the reception and transmission shares one signal bus and that the communication format must contain command, address, received and/or sent data. Software can enable three-line half-duplex communication by configuring SPI_SIO bit in SPI_USER_REG register.

**Note:**

- In half-duplex communication, the order of command, address, received and/or sent data in the communication format should be followed strictly.
- In half-duplex communication, communication formats "command + address + received data + sent data" and "received data + sent data" are not applicable to DMA.
- When ESP32 SPI acts as a slave, the master CS should be active at least one SPI clock period before a read/write process is initiated, and should be inactive at least one SPI clock period after the read/write process is completed.

### 20.3.4 GP-SPI Data Buffer

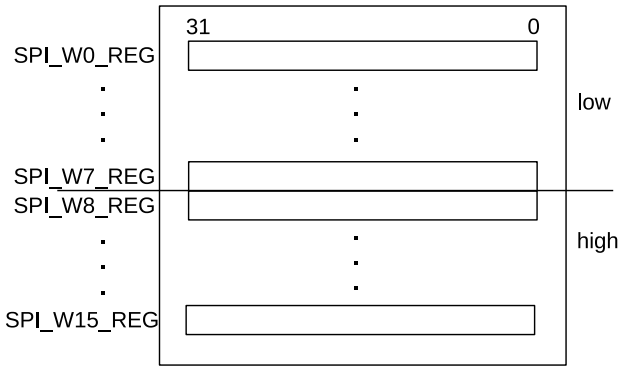



Figure 20.3-2. SPI Data Buffer

ESP32 SPI has 16 × 32 bits of data buffer to buffer data-send and data-receive operations. As is shown in Figure 20.3-2, received data is written from the low byte of SPI_W0_REG by default and the writing ends with SPI_W15_REG. If the data length is over 64 bytes, the extra part will be written from SPI_W0_REG.

Data buffer blocks SPI_W0_REG ~ SPI_W7_REG and SPI_W8_REG ~ SPI_W15_REG data correspond to the lower part and the higher part respectively. They can be used separately, and are controlled by the SPI_USR_MOSI
_HIGHPART bit and the SPI_USR_MISO_HIGHPART bit in register SPI_USER_REG. For example, if SPI is configured as a master, when SPI_USR_MOSI_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for sending data; when SPI_USR_MISO_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for receiving data. If SPI acts as a slave, when SPI_USR_MOSI_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for receiving data; when SPI_USR_MISO_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for sending data.

## 20.4 GP-SPI Clock Control

The maximum output clock frequency of ESP32 GP-SPI master is $f_{\text{apb}}/2$, and the maximum input clock frequency of the ESP32 GP-SPI slave is $f_{\text{apb}}/8$. The master can derive other clock frequencies via frequency division.

$$f_{\text{spi}} = \frac{f_{\text{apb}}}{(\text{SPI_CLKCNT_N+1})(\text{SPI_CLKDIV_PRE+1})}$$

SPI_CLKCNT_N and SPI_CLKDIV_PRE are two bits of register SPI_CLOCK_REG (Please refer to 20.7 Register Description for details). SPI_CLKCNT_H $= \lfloor \frac{\text{SPI_CLKCNT_N+1}}{2} - 1 \rfloor$, SPI_CLKCNT_N=SPI_CLKCNT_L. When the SPI_CLK_EQU_SYSCLK bit in register SPI_CLOCK_REG is set to 1, and the other bits are set to 0, SPI output clock frequency is $f_{\text{apb}}$. For other clock frequencies, SPI_CLK_EQU_SYSCLK needs to be 0. In slave mode, SPI_CLKCNT_N, SPI_CLKCNT_L, SPI_CLKCNT_H and SPI_CLKDIV_PRE should all be 0.

### 20.4.1 GP-SPI Clock Polarity (CPOL) and Clock Phase (CPHA)

The clock polarity and clock phase of ESP32 SPI are controlled by SPI_CK_IDLE_EDGE bit in register SPI_PIN_REG, SPI_CK_OUT_EDGE bit and SPI_CK_I_EDGE bit in register SPI_USER_REG, as well as

Table 20.4-1. Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Master

| Registers | mode0 | mode1 | mode2 | mode3 |
|---|---|---|---|---|
| SPI_CK_IDLE_EDGE | 0 | 0 | 1 | 1 |
| SPI_CK_OUT_EDGE | 0 | 1 | 1 | 0 |
| SPI_MISO_DELAY_MODE | 2(0) | 1(0) | 1(0) | 2(0) |
| SPI_MISO_DELAY_NUM | 0 | 0 | 0 | 0 |
| SPI_MOSI_DELAY_MODE | 0 | 0 | 0 | 0 |
| SPI_MOSI_DELAY_NUM | 0 | 0 | 0 | 0 |

SPI_MISO_DELAY_MODE[1:0] bit, SPI_MISO_DELAY_NUM[2:0] bit, SPI_MOSI_DELAY_MODE[1:0] bit, SPI_MOSI_DELAY_MUM[2:0] bit in register SPI_CTRL2_REG. Table 20.4-1 and Table 20.4-2 show the clock polarity and phase as well as the corresponding register values for ESP32 SPI master and slave, respectively. Note that for mode0 and mode2 in Table 20.4-2, the registers are configured differently in non-DMA mode and DMA mode, and that the SPI slave data is output in advance in DMA mode.

Table 20.4-2. Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Slave

| Registers | mode0 | | mode1 | mode2 | | mode3 |
|---|---|---|---|---|---|---|
| | Non-DMA | DMA | | Non-DMA | DMA | |
| SPI_CK_IDLE_EDGE | 1 | 0 | 1 | 0 | 1 | 0 |
| SPI_CK_I_EDGE | 0 | 1 | 1 | 1 | 0 | 0 |
| SPI_MISO_DELAY_MODE | 0 | 0 | 2 | 0 | 0 | 1 |
| SPI_MISO_DELAY_NUM | 0 | 2 | 0 | 0 | 2 | 0 |
| SPI_MOSI_DELAY_MODE | 2 | 0 | 0 | 1 | 0 | 0 |
| SPI_MOSI_DELAY_NUM | 2 | 3 | 0 | 2 | 3 | 0 |

1. mode0 means CPOL=0, CPHA=0. When SPI is idle, the clock output is logic low; data changes on the falling edge of the SPI clock and is sampled on the rising edge;
2. mode1 means CPOL=0, CPHA=1. When SPI is idle, the clock output is logic low; data changes on the rising edge of the SPI clock and is sampled on the falling edge;

3. mode2 means when CPOL=1, CPHA=0. When SPI is idle, the clock output is logic high; data changes on the rising edge of the SPI clock and is sampled on the falling edge;
4. mode3 means when CPOL=1, CPHA=1. When SPI is idle, the clock output is logic high; data changes on the falling edge of the SPI clock and is sampled on the rising edge.

### 20.4.2 GP-SPI Timing

The data signals of ESP32 GP-SPI can be mapped to physical pins either via IO_MUX or via IO_MUX and GPIO matrix. Input signals will be delayed by two $clk_{\text{apb}}$ clock cycles when they pass through the matrix. Output signals will not be delayed.

When GP-SPI is used as master and the data signals are not received by the SPI controller via GPIO matrix, if GP-SPI output clock frequency is $clk_{\text{apb}}/2$, register SPI_MISO_DELAY_MODE should be set to 0 when configuring the clock polarity. If GP-SPI output clock frequency is not higher than $clk_{\text{apb}}/4$, register SPI_MISO_DELAY_MODE can be set to the corresponding value in Table 20.4-1 when configuring the clock polarity.

When GP-SPI is used in master mode and the data signals enter the SPI controller via the GPIO matrix:

1. If GP-SPI output clock frequency is $clk_{\text{apb}}/2$, register SPI_MISO_DELAY_MODE should be set to 0 and the dummy phase should be enabled (SPI_USR_DUMMY = 1) for one $clk_{\text{spi}}$ clock cycle (SPI_USR_DUMMY_CYC
LELEN = 0) when configuring the clock polarity;
2. If GP-SPI output clock frequency is $clk_{\text{apb}}/4$, register SPI_MISO_DELAY_MODE should be set to 0 when configuring the clock polarity;
3. If GP-SPI output clock frequency is not higher than $clk_{\text{apb}}/8$, register SPI_MISO_DELAY_MODE can be set to the corresponding value in Table 20.4-1 when configuring the clock polarity.

When GP-SPI is used in slave mode, the clock signal and the data signals should be routed to the SPI controller via the same path, i.e., neither the clock signal nor the data signals passes through GPIO matrix, or both of them pass through GPIO matrix. This is important in ensuring that the signals are not delayed by different time periods before they reach the SPI hardware.

Assume that $t_{\text{spi}}$, $t_{\text{pre}}$ and $t_{\text{v}}$ in Figure 20.4-1 denote SPI clock period, how far ahead data output is, and data output delay time, respectively. Assume the SPI slave's main clock period is $t_{\text{apb}}$. For non-DMA mode0, SPI slave data output is delayed by $t_{\text{v}}$:

- $t_{\text{v}} < 3.5 * t_{\text{apb}}$, if CLK does not pass through GPIO matrix;
- $t_{\text{v}} < 5.5 * t_{\text{apb}}$, if CLK passes through GPIO matrix.

In DMA mode1 and mode3, SPI slave data output is delayed by the same period of time as in non-DMA mode. However, for mode0 and mode2, SPI slave data is output earlier by $t_{\text{pre}}$:

- $t_{\text{pre}} < (t_{\text{spi}}/2 - 5.5 * t_{\text{apb}})$, if CLK does not pass through GPIO matrix;
- $t_{\text{pre}} < (t_{\text{spi}}/2 - 7.5 * t_{\text{apb}})$, if CLK passes through GPIO matrix.

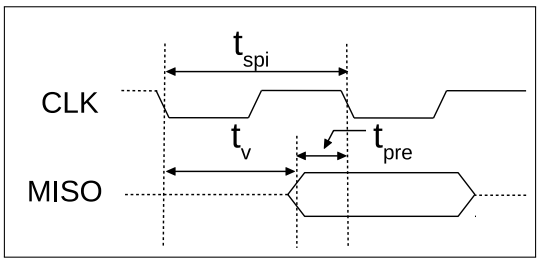



Figure 20.4-1. GP-SPI Slave Data Output

To conclude, if signals do not pass through GPIO matrix, the SPI slave clock frequency is up to $f_{\text{apb}}/8$; if signals pass through GPIO matrix, the SPI slave clock frequency is up to $f_{\text{apb}}/12$. Note that $(t_{\text{spi}}/2-t_{\text{pre}})$ represents data output hold time for SPI slave in mode0 and mode2.

## 20.5 Parallel QSPI

ESP32 SPI controllers support SPI bus memory devices (such as flash and SRAM). The hardware connection between the SPI pins and the memories is shown by Figure 20.5-1.

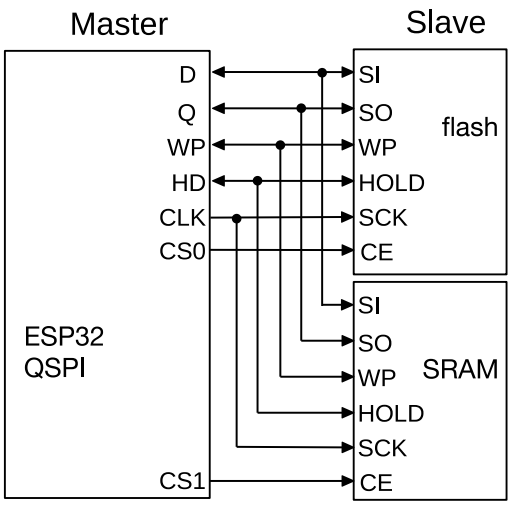



Figure 20.5-1. Parallel QSPI

SPI1, SPI2 and SPI3 controllers can also be configured as QSPI master to connect to external memory. The maximum output clock frequency of the SPI memory interface is $f_{\text{apb}}$, with the same clock configuration as that of the GP-SPI master.

### 20.5.1 Communication Format of Parallel QSPI

To support communication with special slave devices, ESP32 QSPI implements a specifically designed communication protocol. The communication format of ESP32 QSPI master is the same as that of GP-SPI four-line half-duplex communication, except that in address phase and data phase, software can configure registers to enable two-line or four-line transmission. Figure 20.5-2 shows a QSPI communication mode with four-line transmission in address phase and data phase.

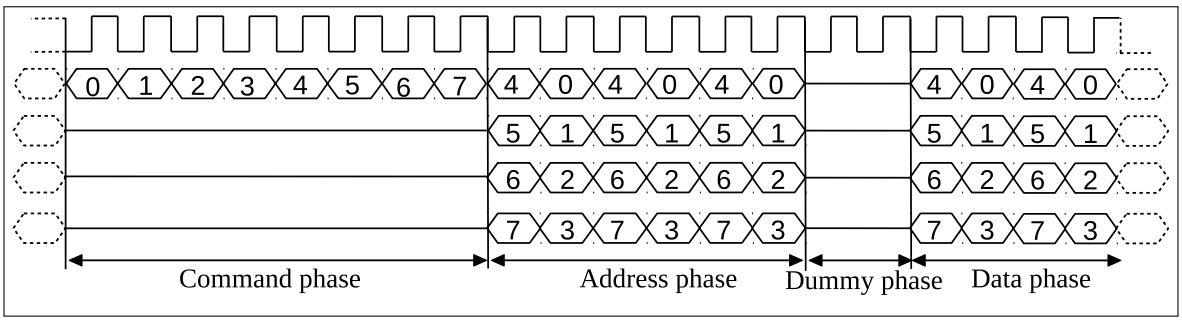



Figure 20.5-2. Communication Format of Parallel QSPI

ESP32 QSPI supports flash-read operation in one-line, two-line, and four-line modes. When working as a QSPI master, the command phase, address phase, dummy phase and data phase can be configured as needed, as flexible as in GP-SPI mode.

**Note** that GPI-SPI full-duplex mode does not support dummy phase.

## 20.6 GP-SPI Interrupt Hardware

ESP32 SPI generates two types of interrupts. One is the SPI interrupt and the other is the SPI DMA interrupt.

ESP32 SPI reckons the completion of send- and/or receive-operations as the completion of one operation from the controller and generates one interrupt. When ESP32 SPI is configured to slave mode, the slave will generate read/write status registers and read/write buffer data interrupts according to different operations.

### 20.6.1 SPI Interrupts

The SPI_*_INTEN bits in the SPI_SLAVE_REG register can be set to enable SPI interrupts. When an SPI interrupt happens, the interrupt flag in the corresponding SPI_*_DONE register will get set. This flag is writable, and an interrupt can be cleared by setting the bit to zero.

- SPI_TRANS_DONE_INT: Triggered when an SPI operation is done.
- SPI_SLV_WR_STA_INT: Triggered when an SPI slave status write is done.
- SPI_SLV_RD_STA_INT: Triggered when an SPI slave status read is done.
- SPI_SLV_WR_BUF_INT: Triggered when an SPI slave buffer write is done.
- SPI_SLV_RD_BUD_INT: Triggered when an SPI slave buffer read is done.

### 20.6.2 DMA Interrupts

- SPI_OUT_TOTAL_EOF_INT: Triggered when all linked lists are sent.
- SPI_OUT_EOF_INT: Triggered when one linked list is sent.
- SPI_OUT_DONE_INT: Triggered when the last linked list item has zero length.
- SPI_IN_SUC_EOF_INT: Triggered when all linked lists are received.

- SPI_IN_ERR_EOF_INT: Triggered when there is an error receiving linked lists.
- SPI_IN_DONE_INT: Triggered when the last received linked list had a length of 0.
- SPI_INLINK_DSCR_ERROR_INT: Triggered when the received linked list is invalid.
- SPI_OUTLINK_DSCR_ERROR_INT: Triggered when the linked list to be sent is invalid.
- SPI_INLINK_DSCR_EMPTY_INT: Triggered when no valid linked list is available.

## 20.7 Register Summary

The addresses in this section are relative to the SPI base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | SPI0 | SPI1 | SPI2 | SPI3 | Acc |
|---|---|---|---|---|---|---|
| **Control and configuration registers** | | | | | | |
| SPI_CTRL_REG | Bit order and QIO/DIO/QOUT/DOUT mode settings | 3FF43008 | 3FF42008 | 3FF64008 | 3FF65008 | R/W |
| SPI_CTRL2_REG | Timing configuration | 3FF43014 | 3FF42014 | 3FF64014 | 3FF65014 | R/W |
| SPI_CLOCK_REG | Clock configuration | 3FF43018 | 3FF42018 | 3FF64018 | 3FF65018 | R/W |
| SPI_PIN_REG | Polarity and CS configuration | 3FF43034 | 3FF42034 | 3FF64034 | 3FF65034 | R/W |
| **Slave mode configuration registers** | | | | | | |
| SPI_SLAVE_REG | Slave mode configuration and interrupt status | 3FF43038 | 3FF42038 | 3FF64038 | 3FF65038 | R/W |
| SPI_SLAVE1_REG | Slave data bit lengths | 3FF4303C | 3FF4203C | 3FF6403C | 3FF6503C | R/W |
| SPI_SLAVE2_REG | Dummy cycle length configuration | 3FF43040 | 3FF42040 | 3FF64040 | 3FF65040 | R/W |
| SPI_SLV_WR_STATUS_REG | Slave status/Part of lower master address | 3FF43030 | 3FF42030 | 3FF64030 | 3FF65030 | R/W |
| SPI_SLV_WRBUF_DLEN_REG | Write-buffer operation length | 3FF43048 | 3FF42048 | 3FF64048 | 3FF65048 | R/W |
| SPI_SLV_RDBUF_DLEN_REG | Read-buffer operation length | 3FF4304C | 3FF4204C | 3FF6404C | 3FF6504C | R/W |
| SPI_SLV_RD_BIT_REG | Read data operation length | 3FF43064 | 3FF42064 | 3FF64064 | 3FF65064 | R/W |
| **User-defined command mode registers** | | | | | | |
| SPI_CMD_REG | Start user-defined command | 3FF43000 | 3FF42000 | 3FF64000 | 3FF65000 | R/W |
| SPI_ADDR_REG | Address data | 3FF43004 | 3FF42004 | 3FF64004 | 3FF65004 | R/W |

| Name | Description | SPI0 | SPI1 | SPI2 | SPI3 | Acc |
|---|---|---|---|---|---|---|
| SPI_USER_REG | User defined command configuration | 3FF4301C | 3FF4201C | 3FF6401C | 3FF6501C | R/W |
| SPI_USER1_REG | Address and dummy cycle configuration | 3FF43020 | 3FF42020 | 3FF64020 | 3FF65020 | R/W |
| SPI_USER2_REG | Command length and value configuration | 3FF43024 | 3FF42024 | 3FF64024 | 3FF65024 | R/W |
| SPI_MOSI_DLEN_REG | MOSI length | 3FF43028 | 3FF42028 | 3FF64028 | 3FF65028 | R/W |
| SPI_W0_REG | SPI data register 0 | 3FF43080 | 3FF42080 | 3FF64080 | 3FF65080 | R/W |
| SPI_W1_REG | SPI data register 1 | 3FF43084 | 3FF42084 | 3FF64084 | 3FF65084 | R/W |
| SPI_W2_REG | SPI data register 2 | 3FF43088 | 3FF42088 | 3FF64088 | 3FF65088 | R/W |
| SPI_W3_REG | SPI data register 3 | 3FF4308C | 3FF4208C | 3FF6408C | 3FF6508C | R/W |
| SPI_W4_REG | SPI data register 4 | 3FF43090 | 3FF42090 | 3FF64090 | 3FF65090 | R/W |
| SPI_W5_REG | SPI data register 5 | 3FF43094 | 3FF42094 | 3FF64094 | 3FF65094 | R/W |
| SPI_W6_REG | SPI data register 6 | 3FF43098 | 3FF42098 | 3FF64098 | 3FF65098 | R/W |
| SPI_W7_REG | SPI data register 7 | 3FF4309C | 3FF4209C | 3FF6409C | 3FF6509C | R/W |
| SPI_W8_REG | SPI data register 8 | 3FF430A0 | 3FF420A0 | 3FF640A0 | 3FF650A0 | R/W |
| SPI_W9_REG | SPI data register 9 | 3FF430A4 | 3FF420A4 | 3FF640A4 | 3FF650A4 | R/W |
| SPI_W10_REG | SPI data register 10 | 3FF430A8 | 3FF420A8 | 3FF640A8 | 3FF650A8 | R/W |
| SPI_W11_REG | SPI data register 11 | 3FF430AC | 3FF420AC | 3FF640AC | 3FF650AC | R/W |
| SPI_W12_REG | SPI data register 12 | 3FF430B0 | 3FF420B0 | 3FF640B0 | 3FF650B0 | R/W |
| SPI_W13_REG | SPI data register 13 | 3FF430B4 | 3FF420B4 | 3FF640B4 | 3FF650B4 | R/W |
| SPI_W14_REG | SPI data register 14 | 3FF430B8 | 3FF420B8 | 3FF640B8 | 3FF650B8 | R/W |
| SPI_W15_REG | SPI data register 15 | 3FF430BC | 3FF420BC | 3FF640BC | 3FF650BC | R/W |
| **DMA configuration registers** | | | | | | |
| SPI_DMA_CONF_REG | DMA configuration register | 3FF43100 | 3FF42100 | 3FF64100 | 3FF65100 | R/W |
| SPI_DMA_OUT_LINK_REG | DMA outlink address and configuration | 3FF43104 | 3FF42104 | 3FF64104 | 3FF65104 | R/W |
| SPI_DMA_IN_LINK_REG | DMA inlink address and configuration | 3FF43108 | 3FF42108 | 3FF64108 | 3FF65108 | R/W |
| SPI_DMA_STATUS_REG | DMA status | 3FF4310C | 3FF4210C | 3FF6410C | 3FF6510C | RO |
| SPI_IN_ERR_EOF_DES_ADDR_REG | Descriptor address where an error occurs | 3FF43120 | 3FF42120 | 3FF64120 | 3FF65120 | RO |
| SPI_IN_SUC_EOF_DES_ADDR_REG | Descriptor address where EOF occurs | 3FF43124 | 3FF42124 | 3FF64124 | 3FF65124 | RO |
| SPI_INLINK_DSCR_REG | Current descriptor pointer | 3FF43128 | 3FF42128 | 3FF64128 | 3FF65128 | RO |
| SPI_INLINK_DSCR_BF0_REG | Next descriptor data pointer | 3FF4312C | 3FF4212C | 3FF6412C | 3FF6512C | RO |
| SPI_INLINK_DSCR_BF1_REG | Current descriptor data pointer | 3FF43130 | 3FF42130 | 3FF64130 | 3FF65130 | RO |

| Name | Description | SPI0 | SPI1 | SPI2 | SPI3 | Acc |
|---|---|---|---|---|---|---|
| SPI_OUT_EOF_BFR_DES_ADDR_REG | Relative buffer address where EOF occurs | 3FF43134 | 3FF42134 | 3FF64134 | 3FF65134 | RO |
| SPI_OUT_EOF_DES_ADDR_REG | Descriptor address where EOF occurs | 3FF43138 | 3FF42138 | 3FF64138 | 3FF65138 | RO |
| SPI_OUTLINK_DSCR_REG | Current descriptor pointer | 3FF4313C | 3FF4213C | 3FF6413C | 3FF6513C | RO |
| SPI_OUTLINK_DSCR_BF0_REG | Next descriptor data pointer | 3FF43140 | 3FF42140 | 3FF64140 | 3FF65140 | RO |
| SPI_OUTLINK_DSCR_BF1_REG | Current descriptor data pointer | 3FF43144 | 3FF42144 | 3FF64144 | 3FF65144 | RO |
| SPI_DMA_RSTATUS_REG | DMA memory read status | 3FF43148 | 3FF42148 | 3FF64148 | 3FF65148 | RO |
| SPI_DMA_TSTATUS_REG | DMA memory write status | 3FF4314C | 3FF4214C | 3FF6414C | 3FF6514C | RO |
| **DMA interrupt registers** | | | | | | |
| SPI_DMA_INT_RAW_REG | Raw interrupt status | 3FF43114 | 3FF42114 | 3FF64114 | 3FF65114 | RO |
| SPI_DMA_INT_ST_REG | Masked interrupt status | 3FF43118 | 3FF42118 | 3FF64118 | 3FF65118 | RO |
| SPI_DMA_INT_ENA_REG | Interrupt enable bits | 3FF43110 | 3FF42110 | 3FF64110 | 3FF65110 | R/W |
| SPI_DMA_INT_CLR_REG | Interrupt clear bits | 3FF4311C | 3FF4211C | 3FF6411C | 3FF6511C | R/W |

## 20.8 Registers

The addresses in this section are relative to the SPI base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 20.7 *Register Summary*.

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

**Register 20.1. SPI_CMD_REG (0x0)**

| (reserved) | SPI_USR | (reserved) | |
|---|---|---|---|
| 31 … 19 | 18 | 17 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**SPI_USR** An SPI operation will be triggered when this bit is set. The bit will be cleared once the operation is done. (R/W)

**Register 20.2. SPI_ADDR_REG (0x4)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**SPI_ADDR_REG** It stores the transmitting address when master is in half-duplex mode or QSPI mode. If the address length is bigger than 32 bits, this register stores the higher 32 bits of address value, SPI_SLV_WR_STATUS_REG stores the rest lower part of address value. If the address length is smaller than 33 bits, this register stores all the address value. The register is in valid only when SPI_USR_ADDR bit is set to 1. (R/W)

**Register 20.3. SPI_CTRL_REG (0x8)**

| (reserved) | SPI_WR_BIT_ORDER | SPI_RD_BIT_ORDER | SPI_FREAD_QIO | SPI_FREAD_DIO | (reserved) | SPI_WP | SPI_FREAD_QUAD | (reserved) | SPI_FREAD_DUAL | SPI_FASTRD_MODE | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 15 | 14 | 13 | 12 0 | |
| 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 0 0 0 0 | 0 | 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**SPI_WR_BIT_ORDER** This bit determines the bit order for command, address and data in transmitted signal. 1: sends LSB first; 0: sends MSB first. (R/W)

**SPI_RD_BIT_ORDER** This bit determines the bit order for received data in received signal. 1: receives LSB first; 0: receives MSB first. (R/W)

**SPI_FREAD_QIO** This bit is used to enable four-line address writes and data reads in QSPI mode. (R/W)

**SPI_FREAD_DIO** This bit is used to enable two-line address writes and data reads in QSPI mode. (R/W)

**SPI_WP** This bit determines the write-protection signal output when SPI is idle in QSPI mode. 1: output high; 0: output low. (R/W)

**SPI_FREAD_QUAD** This bit is used to enable four-line data reads in QSPI mode. (R/W)

**SPI_FREAD_DUAL** This bit is used to enable two-line data reads in QSPI mode. (R/W)

**SPI_FASTRD_MODE** Reserved.

**Register 20.4. SPI_CTRL1_REG (0xC)**

| SPI_CS_HOLD_DELAY | (reserved) | |
|---|---|---|
| 31 28 | 27 0 | |
| 0x05 | 0 0 0 0 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 |

**SPI_CS_HOLD_DELAY** Reserved.

**Register 20.5. SPI_RD_STATUS_REG (0x10)**

| SPI_STATUS_EXT | | SPI_STATUS | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 24 | | 23 … 16 | | 15 | | | | | | | | | | | | | 0 |
| 0x000 | | 0x000 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Reset

**SPI_STATUS_EXT** Reserved.

**SPI_STATUS** Reserved.

**Register 20.6. SPI_CTRL2_REG (0x14)**

| SPI_CS_DELAY_NUM | SPI_CS_DELAY_MODE | SPI_MOSI_DELAY_NUM | SPI_MOSI_DELAY_MODE | SPI_MISO_DELAY_NUM | SPI_MISO_DELAY_MODE | reserved | reserved | SPI_HOLD_TIME | SPI_SETUP_TIME | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 28 | 27 26 | 25 23 | 22 21 | 20 18 | 17 16 | 15 12 | 11 8 | 7 4 | 3 0 | |
| 0x00 | 0x0 | 0x0 | 0x0 | 0x0 | 0x0 | 0x00 | 0x00 | 0x01 | 0x01 | Reset |

**SPI_CS_DELAY_NUM** Reserved.

**SPI_CS_DELAY_MODE** Reserved.

**SPI_MOSI_DELAY_NUM** It is used to configure the number of system clock cycles by which the MOSI signals are delayed. (R/W)

**SPI_MOSI_DELAY_MODE** This register field determines the way the MOSI signals are delayed by SPI clock. (R/W)

After being delayed by SPI_MOSI_DELAY_NUM system clocks, the MOSI signals will then be delayed by the configuration of SPI_MOSI_DELAY_MODE, specifically:

0: no delay.

1: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MOSI signals are delayed by half a cycle, otherwise they are delayed by one cycle.

2: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MOSI signals are delayed by one cycle, otherwise they are delayed by half a cycle.

3: the MOSI signals are delayed one cycle.

**SPI_MISO_DELAY_NUM** It is used to configure the number of system clock cycles by which the MISO signals are delayed. (R/W)

**SPI_MISO_DELAY_MODE** This register field determines the way MISO signals are delayed by SPI clock. (R/W)

After being delayed by SPI_MISO_DELAY_NUM system clock, the MISO signals will then be delayed by the configuration of SPI_MISO_DELAY_MODE, specifically:

0: no delay.

1: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MISO signals are delayed by half a cycle, otherwise they are delayed by one cycle.

2: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MISO signals are delayed by one cycle, otherwise they are delayed by half a cycle.

3: the MISO signals are delayed by one cycle.

**SPI_HOLD_TIME** The number of SPI clock cycles by which CS pin signals are delayed. It is only valid when SPI_CS_HOLD is set to 1. (R/W)

**SPI_SETUP_TIME** It is to configure the time between the CS signal active edge and the first SPI clock edge. It is only valid in half-duplex mode or QSPI mode and when SPI_CS_SETUP is set to 1. (R/W)

**Register 20.7. SPI_CLOCK_REG (0x18)**

| SPI_CLK_EQU_SYSCLK | SPI_CLKDIV_PRE | SPI_CLKCNT_N | SPI_CLKCNT_H | SPI_CLKCNT_L | |
|---|---|---|---|---|---|
| 31 | 30 18 | 17 12 | 11 6 | 5 0 | |
| 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x03 | 0x01 | 0x03 | Reset |

**SPI_CLK_EQU_SYSCLK** In master mode, when this bit is set to 1, SPI output clock is equal to system clock; when set to 0, SPI output clock is divided from system clock. In slave mode, it should be set to 0. (R/W)

**SPI_CLKDIV_PRE** In master mode, it is used to configure the pre-divider value for SPI output clock. It is only valid when SPI_CLK_EQU_SYSCLK is 0. In slave mode, it should be set to 0. (R/W)

**SPI_CLKCNT_N** In master mode, it is used to configure the divider for SPI output clock. It is only valid when SPI_CLK_EQU_SYSCLK is 0. In slave mode, it should be set to 0. (R/W)

**SPI_CLKCNT_H** In master mode, SPI_CLKCNT_H $= \lfloor \frac{\text{SPI_CLKCNT_N}+1}{2} - 1 \rfloor$. It is only valid when SPI_CLK_EQU_SYSCLK is 0. In slave mode, it should be set to 0. (R/W)

**SPI_CLKCNT_L** In master mode, it is equal to SPI_CLKCNT_N. It is only valid when SPI_CLK_EQU_SYSCLK is 0. In slave mode, it should be set to 0. (R/W)

**Register 20.8. SPI_USER_REG (0x1C)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SPI_USR_COMMAND | SPI_USR_ADDR | SPI_USR_DUMMY | SPI_USR_MISO | SPI_USR_MOSI | SPI_USR_DUMMY_IDLE | SPI_USR_MOSI_HIGHPART | SPI_USR_MISO_HIGHPART | (reserved) | SPI_SIO | SPI_FWRITE_QIO | SPI_FWRITE_DIO | SPI_FWRITE_QUAD | SPI_FWRITE_DUAL | SPI_WR_BYTE_ORDER | SPI_RD_BYTE_ORDER | (reserved) | | SPI_CK_OUT_EDGE | SPI_CK_I_EDGE | SPI_CS_SETUP | SPI_CS_HOLD | (reserved) | | SPI_DOUTDIN | |
| 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 | 1 | 0 | 0 | 0 | 0 | 0 | Reset |

**SPI_USR_COMMAND** This bit enables the command phase of an SPI operation in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_ADDR** This bit enables the address phase of an SPI operation in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_DUMMY** This bit enables the dummy phase of an SPI operation in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_MISO** This bit enables the read-data phase of an SPI operation in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_MOSI** This bit enables the write-data phase of an SPI operation in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_DUMMY_IDLE** The SPI clock signal is disabled in the dummy phase when the bit is set in SPI half-duplex mode and QSPI mode. (R/W)

**SPI_USR_MOSI_HIGHPART** If set, MOSI data is stored in SPI_W8 ~ SPI_W15 of the SPI buffer. (R/W)

**SPI_USR_MISO_HIGHPART** If set, MISO data is stored in SPI_W8 ~ SPI_W15 of the SPI buffer. (R/W)

**SPI_SIO** Set this bit to enable three-line half-duplex communication. (R/W)

**SPI_FWRITE_QIO** Reserved.

**SPI_FWRITE_DIO** Reserved.

**SPI_FWRITE_QUAD** Reserved.

**SPI_FWRITE_DUAL** Reserved.

**SPI_WR_BYTE_ORDER** This bit determines the byte order of the command, address and data in transmitted signal. 1: big-endian; 0: little-endian. (R/W)

**SPI_RD_BYTE_ORDER** This bit determines the byte order of received data in transmitted signal. 1: big-endian; 0: little_endian. (R/W)

**SPI_CK_OUT_EDGE** This bit, combined with SPI_MOSI_DELAY_MODE, sets the MOSI signal delay mode. It is only valid in master mode. (R/W)

**SPI_CK_I_EDGE** In slave mode, the bit is the same as SPI_CK_OUT_EDGE in master mode. It is combined with SPI_MISO_DELAY_MODE. It is only valid in slave mode. (R/W)

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

### Register 20.8. SPI_USER_REG (0x1C)

**Continued from the previous page...**

**SPI_CS_SETUP** Setting this bit enables a delay between CS active edge and the first clock edge, in multiples of one SPI clock cycle. In full-duplex mode and QSPI mode, setting this bit results in $(\text{SPI_SETUP_TIME} + 1.5)$ SPI clock cycles delay. In full-duplex mode, there will be 1.5 SPI clock cycles delay for mode0 and mode2, and 1 SPI clock cycle delay for mode1 and mode3. (R/W)

**SPI_CS_HOLD** Setting this bit enables a delay between the end of a transmission and CS being inactive, as specified in SPI_HOLD_TIME. (R/W)

**SPI_DOUTDIN** Set the bit to enable full-duplex communication. (R/W)

### Register 20.9. SPI_USER1_REG (0x20)

| SPI_USR_ADDR_BITLEN | (reserved) | SPI_USR_DUMMY_CYCLELEN | |
|---|---|---|---|
| 31 26 | 25 8 | 7 0 | |
| 23 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 7 | Reset |

**SPI_USR_ADDR_BITLEN** It indicates the bit length of the transmitted address minus one in half-duplex mode and QSPI mode, in multiples of one bit. It is only valid when SPI_USR_ADDR is set to 1. (RO)

**SPI_USR_DUMMY_CYCLELEN** It indicates the number of SPI clock cycles for the dummy phase minus one in SPI half-duplex mode and QSPI mode. It is only valid when SPI_USR_DUMMY is set to 1. (R/W)

Register 20.10. SPI_USER2_REG (0x24)

| SPI_USR_COMMAND_BITLEN | (reserved) | SPI_USR_COMMAND_VALUE | |
|---|---|---|---|
| 31 28 | 27 16 | 15 0 | |
| 7 | 0 0 0 0 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 |

**SPI_USR_COMMAND_BITLEN** It indicates the bit length of the command phase minus one in SPI half-duplex mode and QSPI mode. It is only valid when SPI_USR_COMMAND is set to 1. (R/W)

**SPI_USR_COMMAND_VALUE** It indicates the value of the command to be transmitted in SPI half-duplex mode and QSPI mode. It is only valid when SPI_USR_COMMAND is set to 1. (R/W)

Register 20.11. SPI_MOSI_DLEN_REG (0x28)

| (reserved) | SPI_USR_MOSI_DBITLEN | |
|---|---|---|
| 31 24 | 23 0 | |
| 0 0 0 0 0 0 0 0 | 0x0000000 | Reset |

**SPI_USR_MOSI_DBITLEN** It indicates the length of MOSI data minus one, in multiples of one bit. It is only valid when SPI_USR_MOSI is set to 1 in master mode. (R/W)

Register 20.12. SPI_MISO_DLEN_REG (0x2C)

| (reserved) | SPI_USR_MISO_DBITLEN | |
|---|---|---|
| 31 24 | 23 0 | |
| 0 0 0 0 0 0 0 0 | 0x0000000 | Reset |

**SPI_USR_MISO_DBITLEN** It indicates the length of MISO data minus one, in multiples of one bit. It is only valid when SPI_USR_MISO is set to 1 in master mode. (R/W)

Register 20.13. SPI_SLV_WR_STATUS_REG (0x30)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_SLV_WR_STATUS_REG** In the slave mode this register is the status register for the master to write the slave. In the master mode, if the address length is bigger than 32 bits, SPI_ADDR_REG stores the higher 32 bits of address value, and this register stores the rest lower part of address value. (R/W)

Register 20.14. SPI_PIN_REG (0x34)

| (reserved) | SPI_CS_KEEP_ACTIVE | SPI_CK_IDLE_EDGE | (reserved) | SPI_MASTER_CK_SEL | (reserved) | SPI_MASTER_CS_POL | SPI_CK_DIS | (reserved) | SPI_CS2_DIS | SPI_CS1_DIS | SPI_CS0_DIS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 – 14 | 13 – 11 | 10 – 9 | 8 – 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 | 1 | 1 | 0 | Reset |

**SPI_CS_KEEP_ACTIVE** This bit is only used in master mode where when it is set, the CS signal will keep active. (R/W)

**SPI_CK_IDLE_EDGE** This bit is only used in master mode to configure the logicl level of SPI output clock in idle state. (R/W)
1: the spi_clk line keeps high when idle;
0: the spi_clk line keeps low when idle.

**SPI_MASTER_CK_SEL** Reserved.

**SPI_MASTER_CS_POL** Reserved.

**SPI_CK_DIS** Reserved.

**SPI_CS2_DIS** This bit enables the SPI CS2 signal. 1: disables CS2; 0: enables CS2. (R/W)

**SPI_CS1_DIS** This bit enables the SPI CS1 signal. 1: disables CS1; 0: enables CS1. (R/W)

**SPI_CS0_DIS** This bit enables the SPI CS0 signal. 1: disables CS0; 0: enables CS0. (R/W)

**Register 20.15. SPI_SLAVE_REG (0x38)**

| 31 | 30 | 29 | 28 | 27 | 26 23 | 22 20 | 19 17 | 16 12 | 11 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SPI_SYNC_RESET | SPI_SLAVE_MODE | SPI_SLV_WR_RD_BUF_EN | SPI_SLV_WR_RD_STA_EN | SPI_SLV_CMD_DEFINE | SPI_TRANS_CNT | SPI_SLV_LAST_STATE | SPI_SLV_LAST_COMMAND | (reserved) | SPI_CS_I_MODE | SPI_TRANS_INTEN | SPI_SLV_WR_STA_INTEN | SPI_SLV_RD_STA_INTEN | SPI_SLV_WR_BUF_INTEN | SPI_SLV_RD_BUF_INTEN | SPI_TRANS_DONE | SPI_SLV_WR_STA_DONE | SPI_SLV_RD_STA_DONE | SPI_SLV_WR_BUF_DONE | SPI_SLV_RD_BUF_DONE | |
| 0 | 0 | 0 | 0 | 0 | 0 0 0 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 |

**SPI_SYNC_RESET** When set, it resets the latched values of the SPI clock line, CS line and data line. (R/W)

**SPI_SLAVE_MODE** This bit is used to set the mode of the SPI device. (R/W)

1: slave mode;

0: master mode.

**SPI_SLV_WR_RD_BUF_EN** This bit is only used in slave half-duplex mode, where when it is set, the write and read data commands are enabled. (R/W)

**SPI_SLV_WR_RD_STA_EN** This bit is only used in slave half-duplex mode, where when it is set, the write and read status commands are enabled. (R/W)

**SPI_SLV_CMD_DEFINE** Reserved.

**SPI_TRANS_CNT** The counter for operations in both the master mode and the slave mode. (RO)

**SPI_SLV_LAST_STATE** In slave mode, this contains the state of the SPI state machine. (RO)

**SPI_SLV_LAST_COMMAND** Reserved.

**SPI_CS_I_MODE** Reserved.

**SPI_TRANS_INTEN** The interrupt enable bit for the SPI_TRANS_DONE_INT interrupt. (R/W)

**SPI_SLV_WR_STA_INTEN** The interrupt enable bit for the SPI_SLV_WR_STA_INT interrupt. (R/W)

**SPI_SLV_RD_STA_INTEN** The interrupt enable bit for the SPI_SLV_RD_STA_INT interrupt. (R/W)

**SPI_SLV_WR_BUF_INTEN** The interrupt enable bit for the SPI_SLV_WR_BUF_INT interrupt. (R/W)

**SPI_SLV_RD_BUF_INTEN** The interrupt enable bit for the SPI_SLV_RD_BUF_INT interrupt. (R/W)

**SPI_TRANS_DONE** The raw interrupt status bit for the SPI_TRANS_DONE_INT interrupt. It is set by hardware and cleared by software. (R/W)

**SPI_SLV_WR_STA_DONE** The raw interrupt status bit for the SPI_SLV_WR_STA_INT interrupt. It is set by hardware and cleared by software, and only applicable to slave half-duplex mode. (R/W)

**SPI_SLV_RD_STA_DONE** The raw interrupt status bit for the SPI_SLV_RD_STA_INT interrupt. It is set by hardware and cleared by software, and only applicable to slave half-duplex mode. (R/W)

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

### Register 20.15. SPI_SLAVE_REG (0x38)

Continued from the previous page ...

**SPI_SLV_WR_BUF_DONE** The raw interrupt status bit for the SPI_SLV_WR_BUF_INT interrupt. It is set by hardware and cleared by software, and only applicable to slave half-duplex mode. (R/W)

**SPI_SLV_RD_BUF_DONE** The raw interrupt status bit for the SPI_SLV_RD_BUF_INT interrupt. It is set by hardware and cleared by software, and only applicable to slave half-duplex mode. (R/W)

### Register 20.16. SPI_SLAVE1_REG (0x3C)

| SPI_SLV_STATUS_BITLEN | SPI_SLV_STATUS_FAST_EN | SPI_SLV_STATUS_READBACK | (reserved) | SPI_SLV_RD_ADDR_BITLEN | SPI_SLV_WR_ADDR_BITLEN | SPI_SLV_WRSTA_DUMMY_EN | SPI_SLV_RDSTA_DUMMY_EN | SPI_SLV_WRBUF_DUMMY_EN | SPI_SLV_RDBUF_DUMMY_EN | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 16 | 15 10 | 9 4 | 3 | 2 | 1 | 0 | |
| 0 0 0 0 0 | 0 | 1 | 0 0 0 0 0 0 0 0 0 | 0x00 | 0x00 | 0 | 0 | 0 | 0 | Reset |

**SPI_SLV_STATUS_BITLEN** It is only used in slave half-duplex mode to configure the length of the master writing into the status register. (R/W)

**SPI_SLV_STATUS_FAST_EN** Reserved.

**SPI_SLV_STATUS_READBACK** Reserved.

**SPI_SLV_RD_ADDR_BITLEN** It indicates the address length in bits minus one for a slave-read operation. It is only valid in slave half-duplex mode. (R/W)

**SPI_SLV_WR_ADDR_BITLEN** It indicates the address length in bits minus one for a slave-write operation. It is only valid in slave half-duplex mode. (R/W)

**SPI_SLV_WRSTA_DUMMY_EN** In slave mode, this bit enables the dummy phase for write-status operations. It is only valid in slave half-duplex mode.(R/W)

**SPI_SLV_RDSTA_DUMMY_EN** In slave mode, this bit enables the dummy phase for read-status operations. It is only valid in slave half-duplex mode. (R/W)

**SPI_SLV_WRBUF_DUMMY_EN** In slave mode, this bit enables the dummy phase for write-buffer operations. It is only valid in slave half-duplex mode. (R/W)

**SPI_SLV_RDBUF_DUMMY_EN** In slave mode, this bit enables the dummy phase for read-buffer operations. It is only valid in slave half-duplex mode. (R/W)

Register 20.17. SPI_SLAVE2_REG (0x40)

| SPI_SLV_WRBUF_DUMMY_CYCLELEN | SPI_SLV_RDBUF_DUMMY_CYCLELEN | SPI_SLV_WRSTA_DUMMY_CYCLELEN | SPI_SLV_RDSTA_DUMMY_CYCLELEN | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 | 0x000 | 0x000 | 0x000 | Reset |

**SPI_SLV_WRBUF_DUMMY_CYCLELEN** It indicates the number of SPI clock cycles minus one for the dummy phase for write-data operations. It is only valid when SPI_SLV_WRBUF_DUMMY_EN is set to 1 in slave half-duplex mode. (R/W)

**SPI_SLV_RDBUF_DUMMY_CYCLELEN** It indicates the number of SPI clock cycles minus one for the dummy phase for read-data operations. It is only valid when SPI_SLV_RDBUF_DUMMY_EN is set to 1 in slave half-duplex mode. (R/W)

**SPI_SLV_WRSTA_DUMMY_CYCLELEN** It indicates the number of SPI clock cycles minus one for the dummy phase for write-status register operations. It is only valid when SPI_SLV_WRSTA_DUMMY_EN is set to 1 in slave half-duplex mode. (R/W)

**SPI_SLV_RDSTA_DUMMY_CYCLELEN** It indicates the number of SPI clock cycles minus one for the dummy phase for read-status register operations. It is only valid when SPI_SLV_RDSTA_DUMMY_EN is set to 1 in slave half-duplex mode. (R/W)

Register 20.18. SPI_SLAVE3_REG (0x44)

| SPI_SLV_WRSTA_CMD_VALUE | SPI_SLV_RDSTA_CMD_VALUE | SPI_SLV_WRBUF_CMD_VALUE | SPI_SLV_RDBUF_CMD_VALUE | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 | 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 |

**SPI_SLV_WRSTA_CMD_VALUE** Reserved.

**SPI_SLV_RDSTA_CMD_VALUE** Reserved.

**SPI_SLV_WRBUF_CMD_VALUE** Reserved.

**SPI_SLV_RDBUF_CMD_VALUE** Reserved.

### Register 20.19. SPI_SLV_WRBUF_DLEN_REG (0x48)

| (reserved) | SPI_SLV_WRBUF_DBITLEN | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x0000000 | Reset |

**SPI_SLV_WRBUF_DBITLEN** It indicates the length of written data minus one, in multiples of one bit. It is only valid in slave half-duplex mode. (R/W)

### Register 20.20. SPI_SLV_RDBUF_DLEN_REG (0x4C)

| (reserved) | SPI_SLV_RDBUF_DBITLEN | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 0x0000000 | Reset |

**SPI_SLV_RDBUF_DBITLEN** It indicates the length of read data minus one, in multiples of one bit. It is only valid in slave half-duplex mode. (R/W)

### Register 20.21. SPI_SLV_RD_BIT_REG (0x64)

| (reserved) | SPI_SLV_RDATA_BIT | |
|---|---|---|
| 31 … 24 | 23 … 0 | |
| 0 0 0 0 0 0 0 0 | 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 |

**SPI_SLV_RDATA_BIT** It indicates the bit length of data the master reads from the slave, minus one. It is only valid in slave half-duplex mode. (R/W)

### Register 20.22. SPI_W*n*_REG (*n*: 0-15) (0x80+4**n*)

| 31 … 0 | |
|---|---|
| 0 0 0 0 0 0 0 0 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 |

**SPI_W*n*_REG** Data buffer. (R/W)

### Register 20.23. SPI_TX_CRC_REG (0xC0)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_TX_CRC_REG** Reserved.

### Register 20.24. SPI_EXT2_REG (0xF8)

| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SPI_ST | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | 2 | | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_ST** The current state of the SPI state machine: (RO)

0: idle state
1: preparation state
2: send command state
3: send data state
4: read data state
5: write data state
6: wait state
7: done state

**Register 20.25. SPI_DMA_CONF_REG (0x100)**

| 31–17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8–6 | 5 | 4 | 3 | 2 | 1–0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | SPI_DMA_CONTINUE | SPI_DMA_TX_STOP | SPI_DMA_RX_STOP | (reserved) | SPI_OUT_DATA_BURST_EN | SPI_INDSCR_BURST_EN | SPI_OUTDSCR_BURST_EN | SPI_OUT_EOF_MODE | (reserved) | SPI_AHBM_RST | SPI_AHBM_FIFO_RST | SPI_OUT_RST | SPI_IN_RST | (reserved) | |
| 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 | 0 | 0 | 0 | 0 | 0 0 | Reset |

**SPI_DMA_CONTINUE** This bit enables SPI DMA continuous data TX/RX mode. (R/W)

**SPI_DMA_TX_STOP** When in continuous TX/RX mode, setting this bit stops sending data. (R/W)

**SPI_DMA_RX_STOP** When in continuous TX/RX mode, setting this bit stops receiving data. (R/W)

**SPI_OUT_DATA_BURST_EN** SPI DMA reads data from memory in burst mode. (R/W)

**SPI_INDSCR_BURST_EN** SPI DMA reads inlink descriptor in burst mode. (R/W)

**SPI_OUTDSCR_BURST_EN** SPI DMA reads outlink descriptor in burst mode. (R/W)

**SPI_OUT_EOF_MODE** DMA out-EOF-flag generation mode. (R/W)

1: out-EOF-flag is generated when DMA has popped all data from the FIFO;

0: out-EOF-flag is generated when DMA has pushed all data to the FIFO.

**SPI_AHBM_RST** reset SPI DMA AHB master. (R/W)

**SPI_AHBM_FIFO_RST** This bit is used to reset SPI DMA AHB master FIFO pointer. (R/W)

**SPI_OUT_RST** The bit is used to reset DMA out-FSM and out-data FIFO pointer. (R/W)

**SPI_IN_RST** The bit is used to reset DMA in-DSM and in-data FIFO pointer. (R/W)

**Register 20.26. SPI_DMA_OUT_LINK_REG (0x104)**

| 31 | 30 | 29 | 28 | 27–20 | 19–0 | |
|---|---|---|---|---|---|---|
| (reserved) | SPI_OUTLINK_RESTART | SPI_OUTLINK_START | SPI_OUTLINK_STOP | (reserved) | SPI_OUTLINK_ADDR | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**SPI_OUTLINK_RESTART** Set the bit to add new outlink descriptors. (R/W)

**SPI_OUTLINK_START** Set the bit to start to use outlink descriptor. (R/W)

**SPI_OUTLINK_STOP** Set the bit to stop to use outlink descriptor. (R/W)

**SPI_OUTLINK_ADDR** The address of the first outlink descriptor. (R/W)

**Register 20.27. SPI_DMA_IN_LINK_REG (0x108)**

| (reserved) | SPI_INLINK_RESTART | SPI_INLINK_START | SPI_INLINK_STOP | (reserved) | SPI_INLINK_AUTO_RET | SPI_INLINK_ADDR | |
|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 21 | 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 | 0 | 0x000000 | Reset |

**SPI_INLINK_RESTART** Set the bit to add new inlink descriptors. (R/W)

**SPI_INLINK_START** Set the bit to start to use inlink descriptor. (R/W)

**SPI_INLINK_STOP** Set the bit to stop to use inlink descriptor. (R/W)

**SPI_INLINK_AUTO_RET** when the bit is set, inlink descriptor jumps to the next descriptor when a packet is invalid. (R/W)

**SPI_INLINK_ADDR** The address of the first inlink descriptor. (R/W)

**Register 20.28. SPI_DMA_STATUS_REG (0x10C)**

| (reserved) | SPI_DMA_TX_EN | SPI_DMA_RX_EN | |
|---|---|---|---|
| 31 … 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 |

**SPI_DMA_TX_EN** SPI DMA write-data status bit. (RO)

**SPI_DMA_RX_EN** SPI DMA read-data status bit. (RO)

**Register 20.29. SPI_DMA_INT_ENA_REG (0x110)**

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | (reserved) | SPI_OUT_TOTAL_EOF_INT_ENA | SPI_OUT_EOF_INT_ENA | SPI_OUT_DONE_INT_ENA | SPI_IN_SUC_EOF_INT_ENA | SPI_IN_ERR_EOF_INT_ENA | SPI_IN_DONE_INT_ENA | SPI_INLINK_DSCR_ERROR_INT_ENA | SPI_OUTLINK_DSCR_ERROR_INT_ENA | SPI_INLINK_DSCR_EMPTY_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**SPI_OUT_TOTAL_EOF_INT_ENA** The interrupt enable bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (R/W)

**SPI_OUT_EOF_INT_ENA** The interrupt enable bit for the SPI_OUT_EOF_INT interrupt. (R/W)

**SPI_OUT_DONE_INT_ENA** The interrupt enable bit for the SPI_OUT_DONE_INT interrupt. (R/W)

**SPI_IN_SUC_EOF_INT_ENA** The interrupt enable bit for the SPI_IN_SUC_EOF_INT interrupt. (R/W)

**SPI_IN_ERR_EOF_INT_ENA** The interrupt enable bit for the SPI_IN_ERR_EOF_INT interrupt. (R/W)

**SPI_IN_DONE_INT_ENA** The interrupt enable bit for the SPI_IN_DONE_INT interrupt. (R/W)

**SPI_INLINK_DSCR_ERROR_INT_ENA** The interrupt enable bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (R/W)

**SPI_OUTLINK_DSCR_ERROR_INT_ENA** The interrupt enable bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (R/W)

**SPI_INLINK_DSCR_EMPTY_INT_ENA** The interrupt enable bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (R/W)

**Register 20.30. SPI_DMA_INT_RAW_REG (0x114)**

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | SPI_OUT_TOTAL_EOF_INT_RAW | SPI_OUT_EOF_INT_RAW | SPI_OUT_DONE_INT_RAW | SPI_IN_SUC_EOF_INT_RAW | SPI_IN_ERR_EOF_INT_RAW | SPI_IN_DONE_INT_RAW | SPI_INLINK_DSCR_ERROR_INT_RAW | SPI_OUTLINK_DSCR_ERROR_INT_RAW | SPI_INLINK_DSCR_EMPTY_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUT_TOTAL_EOF_INT_RAW** The raw interrupt status bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (RO)

**SPI_OUT_EOF_INT_RAW** The raw interrupt status bit for the SPI_OUT_EOF_INT interrupt. (RO)

**SPI_OUT_DONE_INT_RAW** The raw interrupt status bit for the SPI_OUT_DONE_INT interrupt. (RO)

**SPI_IN_SUC_EOF_INT_RAW** The raw interrupt status bit for the SPI_IN_SUC_EOF_INT interrupt. (RO)

**SPI_IN_ERR_EOF_INT_RAW** The raw interrupt status bit for the SPI_IN_ERR_EOF_INT interrupt. (RO)

**SPI_IN_DONE_INT_RAW** The raw interrupt status bit for the SPI_IN_DONE_INT interrupt. (RO)

**SPI_INLINK_DSCR_ERROR_INT_RAW** The raw interrupt status bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (RO)

**SPI_OUTLINK_DSCR_ERROR_INT_RAW** The raw interrupt status bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (RO)

**SPI_INLINK_DSCR_EMPTY_INT_RAW** The raw interrupt status bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (RO)

**Register 20.31. SPI_DMA_INT_ST_REG (0x118)**

| 31 | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | (reserved) | SPI_OUT_TOTAL_EOF_INT_ST | SPI_OUT_EOF_INT_ST | SPI_OUT_DONE_INT_ST | SPI_IN_SUC_EOF_INT_ST | SPI_IN_ERR_EOF_INT_ST | SPI_IN_DONE_INT_ST | SPI_INLINK_DSCR_ERROR_INT_ST | SPI_OUTLINK_DSCR_ERROR_INT_ST | SPI_INLINK_DSCR_EMPTY_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUT_TOTAL_EOF_INT_ST** The masked interrupt status bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (RO)

**SPI_OUT_EOF_INT_ST** The masked interrupt status bit for the SPI_OUT_EOF_INT interrupt. (RO)

**SPI_OUT_DONE_INT_ST** The masked interrupt status bit for the SPI_OUT_DONE_INT interrupt. (RO)

**SPI_IN_SUC_EOF_INT_ST** The masked interrupt status bit for the SPI_IN_SUC_EOF_INT interrupt. (RO)

**SPI_IN_ERR_EOF_INT_ST** The masked interrupt status bit for the SPI_IN_ERR_EOF_INT interrupt. (RO)

**SPI_IN_DONE_INT_ST** The masked interrupt status bit for the SPI_IN_DONE_INT interrupt. (RO)

**SPI_INLINK_DSCR_ERROR_INT_ST** The masked interrupt status bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (RO)

**SPI_OUTLINK_DSCR_ERROR_INT_ST** The masked interrupt status bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (RO)

**SPI_INLINK_DSCR_EMPTY_INT_ST** The masked interrupt status bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (RO)

### Register 20.32. SPI_DMA_INT_CLR_REG (0x11C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | (reserved) | SPI_OUT_TOTAL_EOF_INT_CLR | SPI_OUT_EOF_INT_CLR | SPI_OUT_DONE_INT_CLR | SPI_IN_SUC_EOF_INT_CLR | SPI_IN_ERR_EOF_INT_CLR | SPI_IN_DONE_INT_CLR | SPI_INLINK_DSCR_ERROR_INT_CLR | SPI_OUTLINK_DSCR_ERROR_INT_CLR | SPI_INLINK_DSCR_EMPTY_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUT_TOTAL_EOF_INT_CLR** Set this bit to clear the SPI_OUT_TOTAL_EOF_INT interrupt. (R/W)

**SPI_OUT_EOF_INT_CLR** Set this bit to clear the SPI_OUT_EOF_INT interrupt. (R/W)

**SPI_OUT_DONE_INT_CLR** Set this bit to clear the SPI_OUT_DONE_INT interrupt. (R/W)

**SPI_IN_SUC_EOF_INT_CLR** Set this bit to clear the SPI_IN_SUC_EOF_INT interrupt. (R/W)

**SPI_IN_ERR_EOF_INT_CLR** Set this bit to clear the SPI_IN_ERR_EOF_INT interrupt. (R/W)

**SPI_IN_DONE_INT_CLR** Set this bit to clear the SPI_IN_DONE_INT interrupt. (R/W)

**SPI_INLINK_DSCR_ERROR_INT_CLR** Set this bit to clear the SPI_INLINK_DSCR_ERROR_INT interrupt. (R/W)

**SPI_OUTLINK_DSCR_ERROR_INT_CLR** Set this bit to clear the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (R/W)

**SPI_INLINK_DSCR_EMPTY_INT_CLR** Set this bit to clear the SPI_INLINK_DSCR_EMPTY_INT interrupt. (R/W)

### Register 20.33. SPI_IN_ERR_EOF_DES_ADDR_REG (0x120)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_IN_ERR_EOF_DES_ADDR_REG** The inlink descriptor address when SPI DMA encountered an error in receiving data. (RO)

### Register 20.34. SPI_IN_SUC_EOF_DES_ADDR_REG (0x124)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_IN_SUC_EOF_DES_ADDR_REG** The last inlink descriptor address when SPI DMA encountered EOF. (RO)

### Register 20.35. SPI_INLINK_DSCR_REG (0x128)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_INLINK_DSCR_REG** The address of the current inlink descriptor. (RO)

### Register 20.36. SPI_INLINK_DSCR_BF0_REG (0x12C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_INLINK_DSCR_BF0_REG** The address of the next inlink descriptor. (RO)

### Register 20.37. SPI_INLINK_DSCR_BF1_REG (0x130)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_INLINK_DSCR_BF1_REG** The address of the next inlink data buffer. (RO)

### Register 20.38. SPI_OUT_EOF_BFR_DES_ADDR_REG (0x134)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUT_EOF_BFR_DES_ADDR_REG** The buffer address corresponding to the outlink descriptor that produces EOF. (RO)

### Register 20.39. SPI_OUT_EOF_DES_ADDR_REG (0x138)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUT_EOF_DES_ADDR_REG** The last outlink descriptor address when SPI DMA encountered EOF. (RO)

### Register 20.40. SPI_OUTLINK_DSCR_REG (0x13C)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUTLINK_DSCR_REG** The address of the current outlink descriptor. (RO)

### Register 20.41. SPI_OUTLINK_DSCR_BF0_REG (0x140)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUTLINK_DSCR_BF0_REG** The address of the next outlink descriptor. (RO)

### Register 20.42. SPI_OUTLINK_DSCR_BF1_REG (0x144)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**SPI_OUTLINK_DSCR_BF1_REG** The address of the next outlink data buffer. (RO)

### Register 20.43. SPI_DMA_RSTATUS_REG (0x148)

| TX_FIFO_EMPTY | TX_FIFO_FULL | (reserved) | TX_DES_ADDRESS | |
|---|---|---|---|---|
| 31 | 30 | 29 ... 20 | 19 ... 0 | |
| 0 | 0 | 0 0 0 0 0 0 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 |

**TX_FIFO_EMPTY** The SPI DMA TX FIFO is empty. (RO)

**TX_FIFO_FULL** The SPI DMA TX FIFO is full. (RO)

**TX_DES_ADDRESS** The LSB of the SPI DMA outlink descriptor address. (RO)

**Register 20.44. SPI_DMA_TSTATUS_REG (0x14C)**

| RX_FIFO_EMPTY | RX_FIFO_FULL | (reserved) | RX_DES_ADDRESS | |
|---|---|---|---|---|
| 31 | 30 | 29 … 20 | 19 … 0 | |
| 0 | 0 | 0 0 0 0 0 0 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 |

**RX_FIFO_EMPTY** The SPI DMA RX FIFO is empty. (RO)

**RX_FIFO_FULL** The SPI DMA RX FIFO is full. (RO)

**RX_DES_ADDRESS** The LSB of the SPI DMA inlink descriptor address. (RO)

# Chapter 21

# I2C Controller (I2C)

## 21.1 Overview

An I2C (Inter-Integrated Circuit) bus can be used for communication with several external devices connected to the same bus as ESP32. The ESP32 has dedicated hardware to communicate with peripherals on the I2C bus.

## 21.2 Features

The I2C controller has the following features:

- Supports both master mode and slave mode
- Supports multi-master and multi-slave communication
- Supports standard mode (100 kbit/s)
- Supports fast mode (400 kbit/s)
- Supports 7-bit addressing and 10-bit addressing
- Supports continuous data transmission with disabled Serial Clock Line (SCL)
- Supports programmable digital noise filter

## 21.3 Functional Description

### 21.3.1 Introduction

I2C is a two-wire bus, consisting of an SDA and an SCL line. These lines are configured to open the drain output. The lines are shared by two or more devices: usually one or more masters and one or more slaves.

Communication starts when a master sends out a start condition: it will pull the SDA line low, and will then pull the SCL line high. It will send out nine clock pulses over the SCL line. The first eight pulses are used to shift out a byte consisting of a 7-bit address and a read/write bit. If a slave with this address is active on the bus, the slave can answer by pulling the SDA low on the ninth clock pulse. The master can then send out more 9-bit clock pulse clusters and, depending on the read/write bit sent, the device or the master will shift out data on the SDA line, with the other side acknowledging the transfer by pulling the SDA low on the ninth clock pulse. During data transfer, the SDA line changes only when the SCL line is low. When the master has finished the communication, it will send a stop condition on the bus by raising SDA, while SCL will already be high.

The ESP32 I2C peripheral can handle the I2C protocol, freeing up the processor cores for other tasks.

## 21.3.2 Architecture

An I2C controller can operate either in master mode or slave mode. The I2C_MS_MODE register is used to select the mode. Figure 21.3-1 shows the I2C Master architecture, while Figure 21.3-2 shows the I2C Slave architecture.

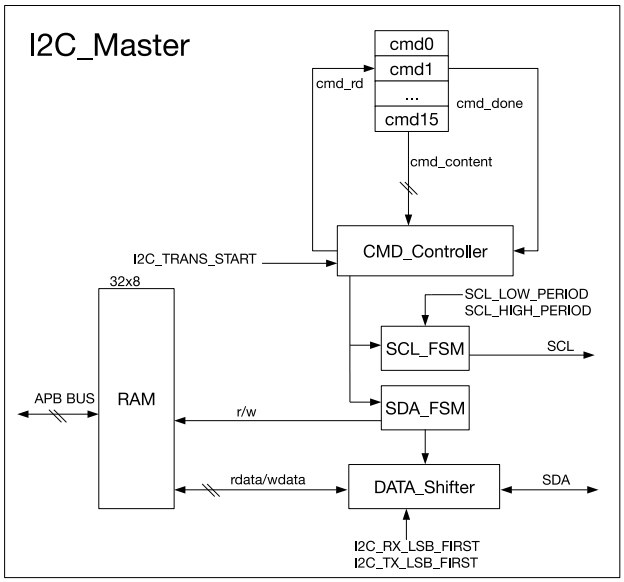



Figure 21.3-1. I2C Master Architecture

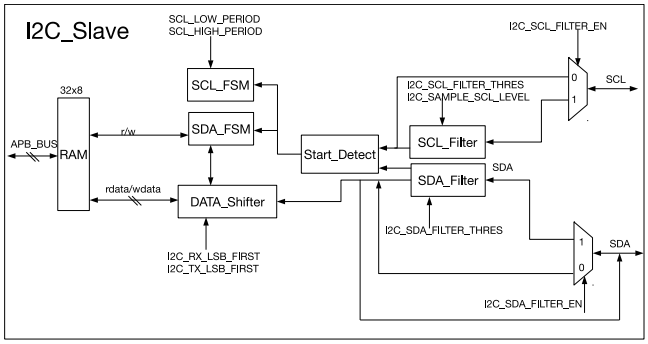



Figure 21.3-2. I2C Slave Architecture

The I2C controller contains the following units:

- RAM, the size of which is 32 x 8 bits, and it is directly mapped onto the address space of the CPU cores, starting at address REG_I2C_BASE+0x100. Each byte of I2C data is stored in a 32-bit word of memory (so, the first byte is at +0x100, the second byte at +0x104, the third byte at +0x108, etc.) Users need to set register I2C_NONFIFO_EN.

  > **Note:**
  > The I2C controller only supports NON-FIFO mode.

- A CMD_Controller and 16 command registers (cmd0 ~ cmd15), which are used by the I2C Master to control data transmission. One command at a time is executed by the I2C controller.
- SCL_FSM: A state machine that controls the SCL clock. The I2C_SCL_HIGH_PERIOD_REG and

I2C_SCL_
LOW_PERIOD_REG registers are used to configure the frequency and duty cycle of the signal on the SCL line.

- SDA_FSM: A state machine that controls the SDA data line.
- DATA_Shifter which converts the byte data to an outgoing bitstream, or converts an incoming bitstream to byte data. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used for configuring whether the LSB or MSB is stored or transmitted first.
- SCL_Filter and SDA_Filter: Input noise filter for the I2C_Slave. The filter can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN. The filter can remove line glitches with pulse width less than I2C_SCL_FILTER_THRES and I2C_SDA_FILTER_THRES ABP clock cycles.

### 21.3.3 I2C Bus Timing

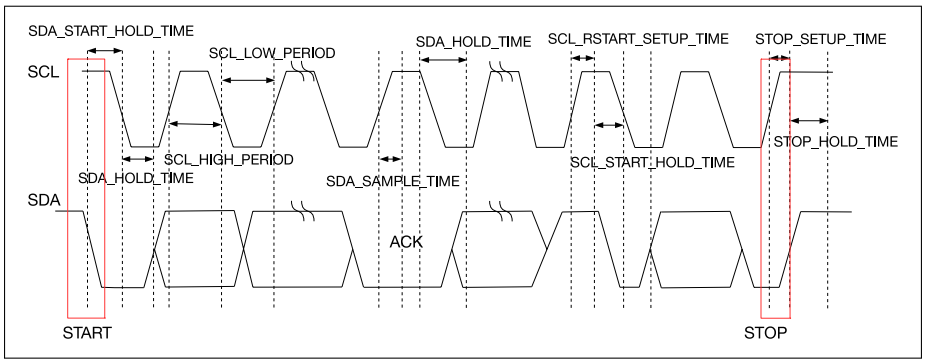



Figure 21.3-3. I2C Sequence Chart

Figure 21.3-3 is an I2C sequence chart. When the I2C controller works in master mode, SCL is an output signal. In contrast, when the I2C controller works in slave mode, the SCL becomes an input signal. The values assigned to I2C_SDA_HOLD_REG and I2C_SDA_SAMPLE_REG are still valid in slave mode. Users need to configure the values of I2C_SDA_HOLD_TIME and I2C_SDA_SAMPLE_TIME, according to the host characteristics, for the I2C slave to receive data properly. Table 21.3-1 shows available settings of SCL low and high level cycles when SCL is configured to direct output mode. The settings determine the SCL output frequency $f_{scl}$.

Table 21.3-1. SCL Frequency Configuration

| I2C_SCL_FILTER_EN | I2C_SCL_FILTER_THRES | SCL_Low_Level_Cycles | SCL_High_Level_Cycles |
|---|---|---|---|
| 0 | Don't care | I2C_SCL_LOW_PERIOD+1 | I2C_SCL_HIGH_PERIOD+7 |
| 1 | [0,2] | | I2C_SCL_HIGH_PERIOD+8 |
| | [3,7] | | I2C_SCL_HIGH_PERIOD+6+I2C_SCL_FILTER_THRES |

$$f_{scl} = \frac{80\ \text{MHz}}{\text{SCL_Low_Level_Cycles} + \text{SCL_High_Level_Cycles}}$$

According to the I2C protocol, each transmission of data begins with a START condition and ends with a STOP condition. Data is transmitted by one byte at a time, and each byte has an ACK bit. The receiver informs the transmitter to continue transmission by pulling down SDA, which indicates an ACK. The receiver can also indicate it wants to stop further transmission by pulling up the SDA line, thereby not indicating an ACK.

Figure 21.3-3 also shows the registers that can configure the START bit, STOP bit, SDA hold time, and SDA sample time.

**Notice:** If the I2C pads are configured in open-drain mode, it will take longer for the signal lines to transition from a low level to a high level. The transition duration is determined together by the pull-up resistor and capacitor. The output frequency of SCL is relatively low in open-drain mode.

### 21.3.4 I2C cmd Structure

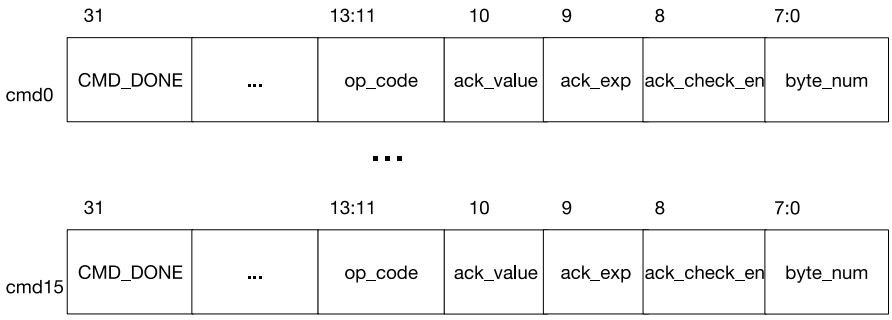



**Figure 21.3-4. Structure of The I2C Command Register**

The Command register is active only in I2C master mode, with its internal structure shown in Figure 21.3-4.

CMD_DONE: The CMD_DONE bit of every command can be read by software to tell if the command has been handled by hardware.

op_code: op_code is used to indicate the command. The I2C controller supports four commands:

- RSTART: op_code = 0 is the RSTART command to control the transmission of a START or RESTART I2C condition.
- WRITE: op_code = 1 is the WRITE command for the I2C Master to transmit data.
- READ: op_code = 2 is the READ command for the I2C Master to receive data.
- STOP: op_code = 3 is the STOP command to control the transmission of a STOP I2C condition.
- END: op_code = 4 is the END command for continuous data transmission. When the END command is given, SCL is temporarily disabled to allow software to reload the command and data registers for subsequent events before resuming. Transmission will then continue seamlessly.

A complete data transmission process begins with an RSTART command, and ends with a STOP command.

ack_value: When receiving data, this bit is used to indicate whether the receiver will send an ACK after this byte has been received.

ack_exp: This bit is to set an expected ACK value for the transmitter.

ack_check_en: When transmitting a byte, this bit enables checking the ACK value received against the ack_exp value. Checking is enabled by 1, while 0 disables it.

byte_num: This register specifies the length of data (in bytes) to be read or written. The maximum length is 255, while the minimum is 1. When the op_code is RSTART, STOP or END, this value is meaningless.

### 21.3.5 I2C Master Writes to Slave

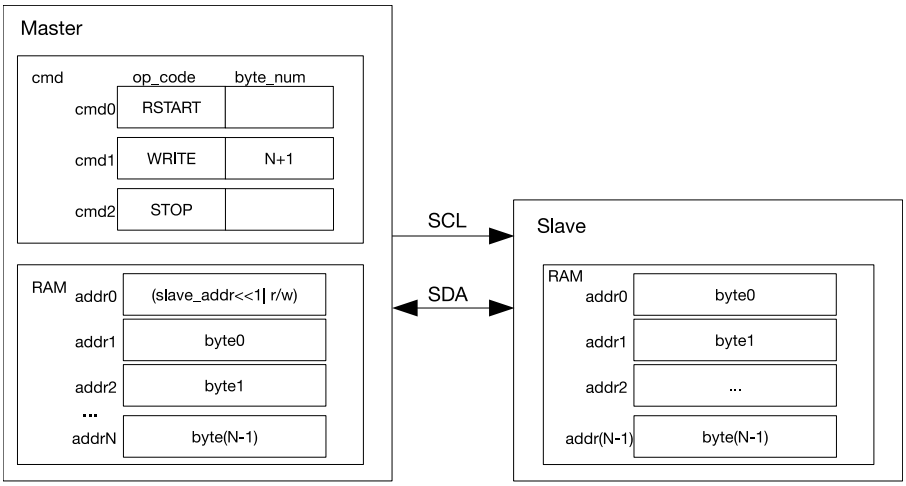



Figure 21.3-5. I2C Master Writes to Slave with 7-bit Address

In all subsequent figures that illustrate I2C transactions and behavior, both the I2C Master and Slave devices are assumed to be ESP32 I2C peripheral controllers for ease of demonstration.

Figure 21.3-5 shows the I2C Master writing N bytes of data to an I2C Slave. According to the I2C protocol, the first byte is the Slave address. As shown in the diagram, the first byte of the RAM unit has been populated with the Slave's 7-bit address plus the 1-bit read/write flag. In this case, the flag is zero, indicating a write operation. The rest of the RAM unit holds N bytes of data ready for transmission. The cmd unit has been populated with the sequence of commands for the operation.

For the I2C master to begin an operation, the bus must not be busy, i.e. the SCL line must not be pulled low by another device on the I2C bus. The I2C operation can only begin when the SCL line is released (made high) to indicate that the I2C bus is free. After the cmd unit and data are prepared, I2C_TRANS_START bit in I2C_CTR_REG must be set to begin the configured I2C Master operation. The I2C Master then initiates a START condition on the bus and progresses to the WRITE command which will fetch N+1 bytes from RAM and send them to the Slave. The first of these bytes is the address byte.

When the transmitted data size exceeds I2C_NONFIFO_TX_THRES, an I2C_TX_SEND_EMPTY_INT interrupt will be generated. After detecting the interrupt, software can read TXFIFO_END_ADDR in register RXFIFO_ST_REG, get the last address of the data in the RAM and refresh the old data in the RAM. TXFIFO_END_ADDR will be refreshed each time interrupt I2C_TX_SEND_EMPTY_INT or I2C_TRANS_COMPLETE_INT occurs.

When ack_check_en is set to 1, the Master will check the ACK value each time it sends a data byte. If the ACK value received does not match ack_exp (the expected ACK value) in the WRITE command, then the Master will generate an I2C_ACK_ERR_INT interrupt and stop the transmission.

During transmission, when the SCL is high, if the input value and output value of SDA do not match, then the Master will generate an I2C_ARBITRATION_LOST_INT interrupt. When the transmission is finished, the Master will generate an I2C_TRANS_COMPLETE_INT interrupt.

After detecting the START bit sent from the Master, the Slave will start receiving the address and comparing it to its own. If the address does not match I2C_SLAVE_ADDR, then the Slave will ignore the rest of the transmission. If they do match, the Slave will store the rest of the data into RAM in the receiving order. When the data size exceeds I2C_NONFIFO_RX_THRES, an I2C_RX_REC_FULL_INT interrupt is generated. After

detecting the interrupt, software will get the starting and ending addresses in the RAM by reading RXFIFO_START_ADDR and RXFIFO_END_ADDR bits in register RXFIFO_ST_REG, and fetch the data for further processing. Register RXFIFO_START_ADDR is refreshed only once during each transmission, while RXFIFO_END_ADDR gets refreshed every time when either I2C_RX_REC_FULL_INT or I2C_TRANS_COMPLETE_INT interrupt is generated.

When the END command is not used, the I2C master can transmit up to (14*255-1) bytes of valid data, and the cmd unit is populated with RSTART + 14 WRITE + 1 STOP.

There are several special cases to be noted:

- If the Master fails to send a STOP bit, because the SDA is pulled low by other devices, then the Master needs to be reset.
- If the Master fails to send a START bit, because the SDA or SCL is pulled low by other devices, then the Master needs to be reset. It is recommended that the software uses a timeout period to implement the reset.
- If the SDA is pulled low by the Slave during transmission, the Master can simply release it by sending it nine SCL clock signals at the most.

It is important to note that the behaviour of another I2C master or slave device on the bus may not always be similar to that of the ESP32 I2C peripheral in the master- or slave-mode operation described above. Please consult the datasheets of the respective I2C devices to ensure proper operation under all bus conditions.

The ESP32 I2C controller uses 7-bit addressing by default. However, 10-bit addressing can also be used. In the master, this is done by sending a second I2C address byte after the first address byte. In the slave, the I2C_SLAVE_ADDR_10BIT_EN bit in I2C_SLAVE_ADDR_REG can be set to activate a 10-bit addressing mode. I2C_SLAVE_ADDR is used to configure the I2C Slave address, as per usual. Figure 21.3-6 shows the equivalent of I2C Master operation writing N-bytes of data to an I2C Slave with a 10-bit address. Since 10-bit Slave addresses require an extra address byte, both the byte_num field of the WRITE command and the number of total bytes in RAM increase by one.

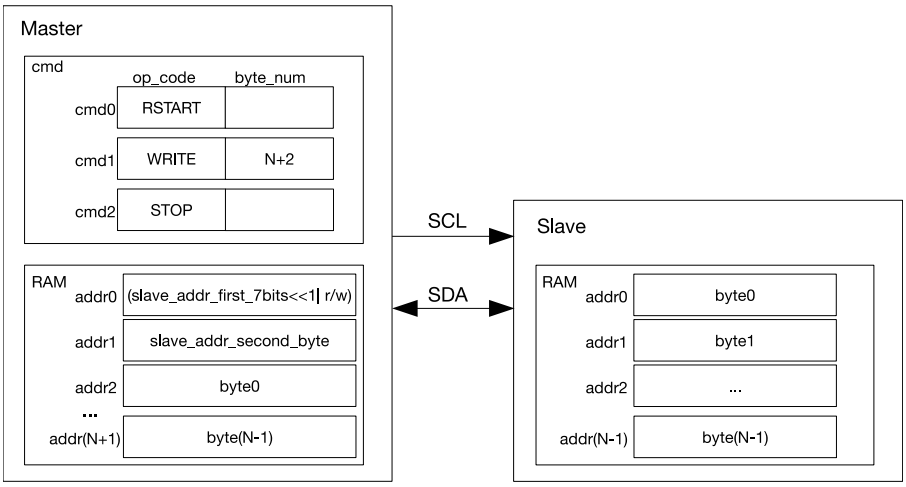



Figure 21.3-6. I2C Master Writes to Slave with 10-bit Address

When the END command is not used, the I2C master can transmit up to (14*255-2) bytes of valid data to Slave with 10-bit address.

One way many I2C Slave devices are designed is by exposing a register block containing various settings. The I2C Master can write one or more of these registers by sending the Slave a register address. The ESP32 I2C Slave controller has hardware support for such a scheme.

Specifically, on the Slave, I2C_FIFO_ADDR_CFG_EN can be set so that the I2C Master can write to a specified register address inside the I2C Slave memory block. Figure 21.3-7 shows the Master writing N-bytes of data byte0 ~ byte(N-1) from the RAM unit to register address M (determined by addrM in RAM unit) with the Slave. In this mode, Slave can receive up to 32 bytes of valid data. When Master needs to transmit extra amount of data, segmented transmission can be enabled.

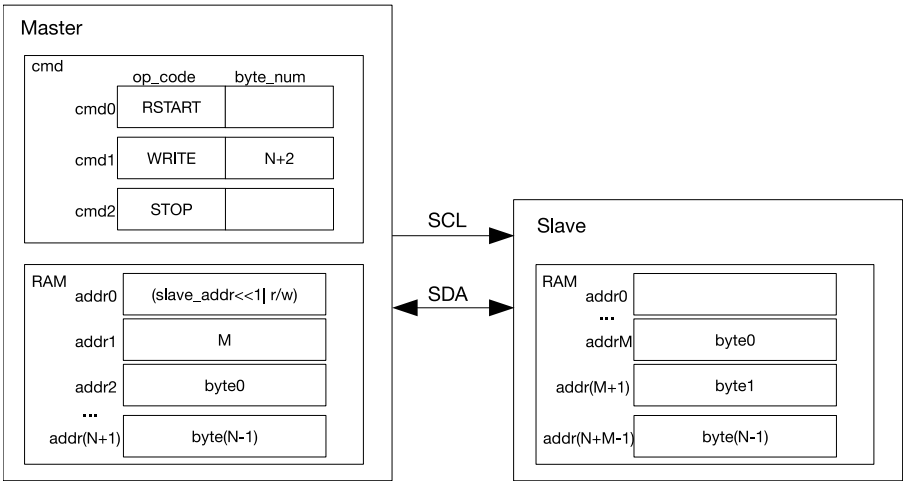



Figure 21.3-7. I2C Master Writes to addrM in RAM of Slave with 7-bit Address

If the data size exceeds the capacity of a 14-byte read/write cmd, the END command can be called to enable segmented transmission. Figure 21.3-8 shows the Master writing data to the Slave, in three segments. The first segment shows the configuration of the Master's commands and the preparation of data in the RAM unit. When the I2C_TRANS_START bit is enabled, the Master starts transmission. After executing the END command, the Master will turn off the SCL clock and pull the SCL low to reserve the bus and prevent any other device from transacting on the bus. The controller will generate an I2C_END_DETECT_INT interrupt to notify the software.

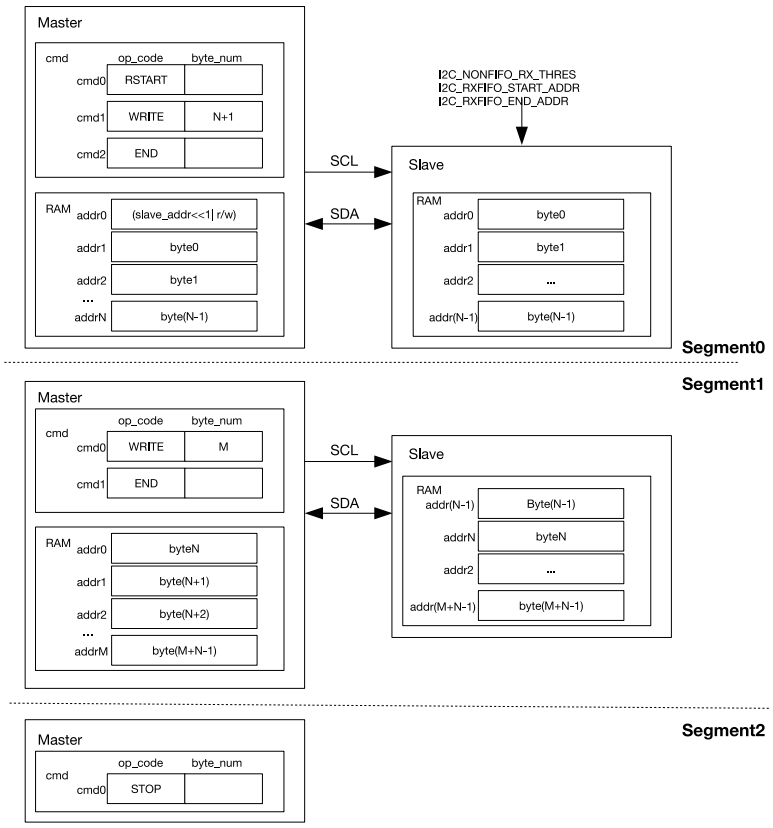



Figure 21.3-8. Master Writes to Slave with 7-bit Address in Three Segments

After detecting an I2C_END_DETECT_INT interrupt, the software can refresh the contents of the cmd and RAM blocks, as shown in the second segment. Subsequently, it should clear the I2C_END_DETECT_INT interrupt and resume the transaction by setting the I2C_TRANS_START bit. To stop the transaction, it should configure the cmd, as the third segment shows, and enable the I2C_TRANS_START bit to generate a STOP bit, after detecting the I2C_END_DETECT_INT interrupt.

Please note that the other masters on the bus will be starved of bus time between two segments. The bus is only released after a STOP signal is sent.

**Note:** When there are more than three segments, the address of an END command in the cmd should not be altered into another command by the next segment.

### 21.3.6 Master Reads from Slave

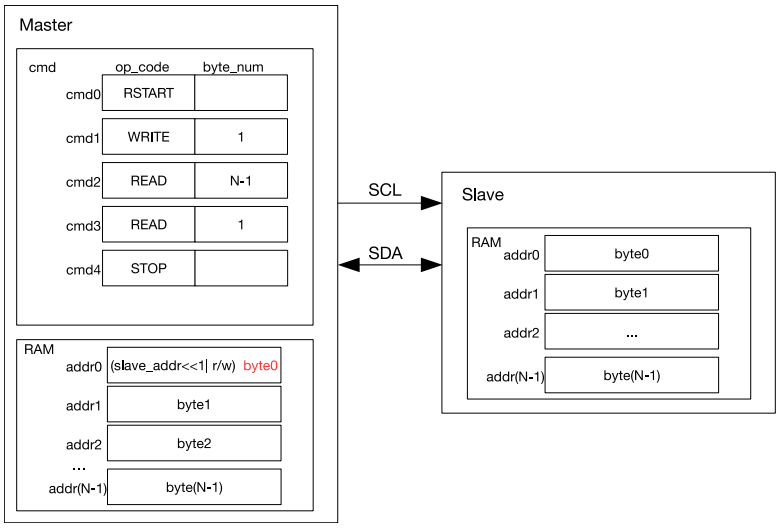



Figure 21.3-9. Master Reads from Slave with 7-bit Address

Figure 21.3-9 shows the Master reading N-bytes of data from an Slave with a 7-bit address. At first, the Master needs to send the address of the Slave, so cmd1 is a WRITE command. The byte that this command sends is the slave address plus the R/W flag, which in this case is 1 and, therefore, indicates that this is going to be a read operation. The Slave starts to send data to the Master if the addresses match. The Master will return ACK, according to the ack_value in the READ command, upon receiving every byte. As can be seen from Figure 21.3-9, READ is divided into two segments. The Master replies ACK to N-1 bytes in cmd2 and does not reply ACK to the single byte READ command in cmd3, i.e., the last transmitted data. Users can configure it as they wish.

When storing the received data, Master will start from the first address in RAM. Byte0 (Slave address + 1-bit R/W marker bit) will be overwritten.

When the END command is not used, the Master can receive up to (13*255) bytes of valid data. The cmd unit is populated with RSTART + 1 WRITE + 13 READ + 1 STOP.

Figure 21.3-10 shows the Master reading data from a slave with a 10-bit address. This mode can be enabled by setting I2C_SLAVE_ADDR_10BIT_EN bit and preparing data to be sent in the slave RAM. In the Master, two bytes of RAM are used for a 10-bit address. Finally, the I2C _TRANS_START bit must be set to enable one transaction.

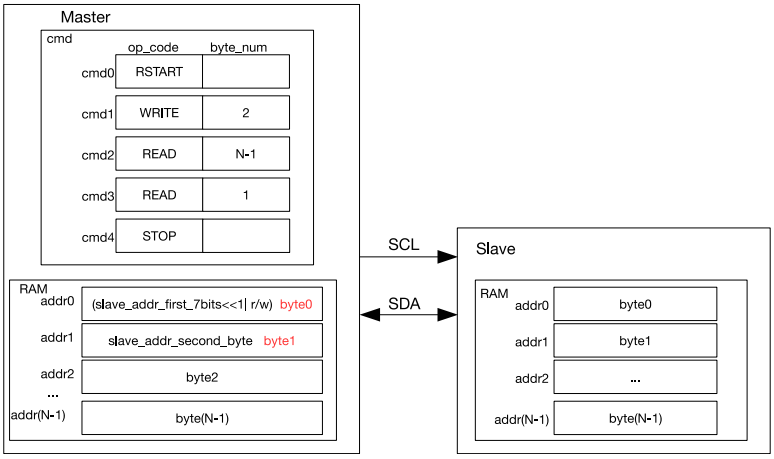



Figure 21.3-10. Master Reads from Slave with 10-bit Address

Figure 21.3-11 shows the Master reading data from a specified address in the Slave. This mode can be enabled by setting I2C_FIFO_ADDR_CFG_EN and preparing the data to be read by the master in the Slave RAM block. Subsequently, the address of the Slave and the address of the specified register (that is, M) have to be determined by the master. Finally, the I2C_TRANS_START bit must be set in the Master to initiate the read operation, following which the Slave will fetch N bytes of data from RAM and send them to the Master.

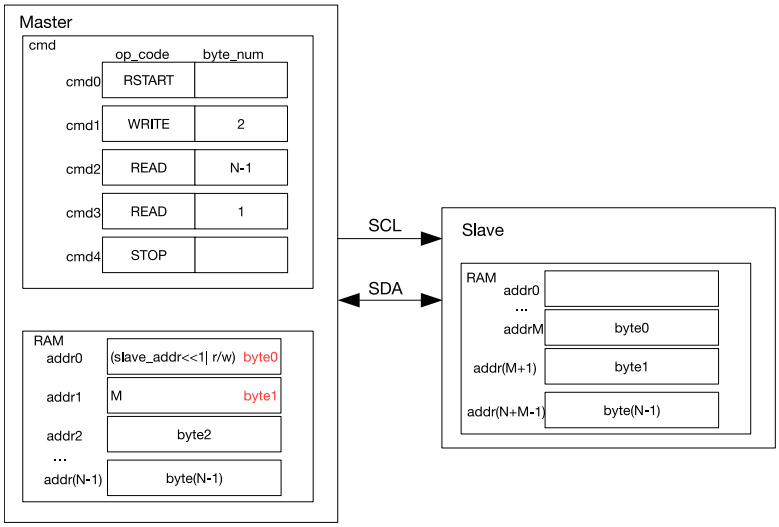



Figure 21.3-11. Master Reads N Bytes of Data from addrM in Slave with 7-bit Address

Figure 21.3-12 shows the Master reading N+M bytes of data in three segments from the Slave. The first segment shows the configuration of the cmd and the preparation of data in the Slave RAM. When the I2C_TRANS_START bit is enabled, the Master starts the operation. The Master will refresh the cmd after executing the END command. It will clear the I2C_END_DETECT_INT interrupt, set the I2C_TRANS_START bit and resume the transaction. To stop the transaction, the Master will configure the cmd, as the third segment shows, after detecting the I2C_END_DETECT_INT interrupt. After setting the I2C_TRANS_START bit, Master will send a STOP bit to stop the transaction.

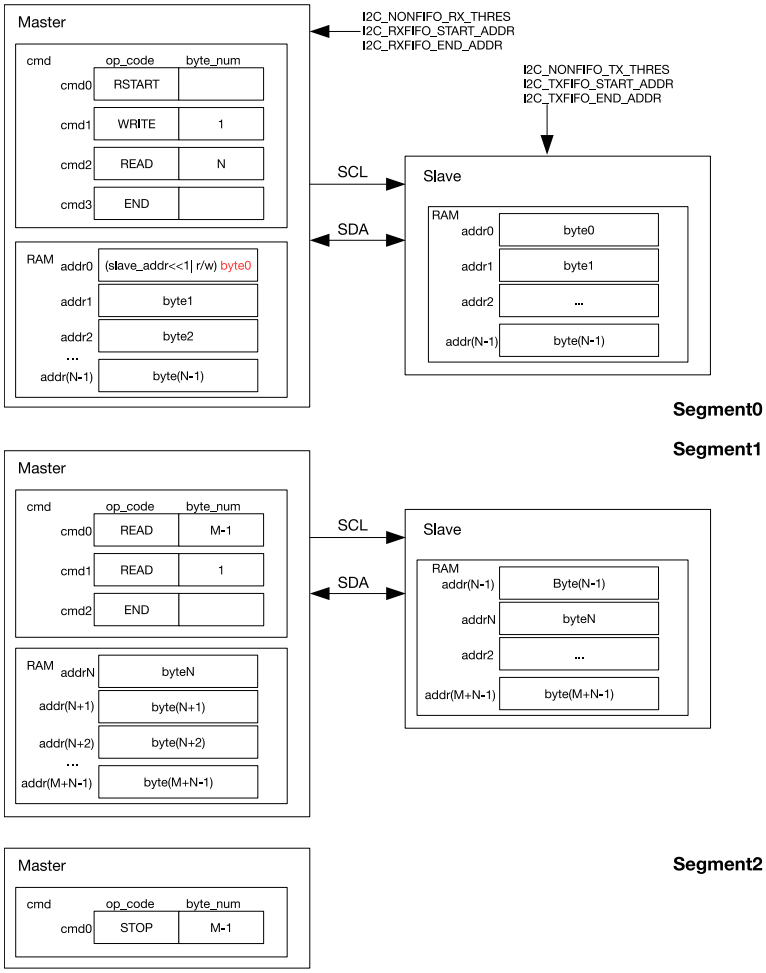



Figure 21.3-12. Master Reads from Slave with 7-bit Address in Three Segments

## 21.3.7 Interrupts

- I2C_TX_SEND_EMPTY_INT: Triggered when the Master or Slave has sent nonfifo_tx_thres bytes of data.
- I2C_RX_REC_FULL_INT: Triggered when the Master or Slave has received nonfifo_rx_thres bytes of data.
- I2C_ACK_ERR_INT: Triggered when the Master receives an ACK that is not as expected, or when the Slave receives an ACK whose value is 1.
- I2C_TRANS_START_INT: Triggered when the Master or Slave sends the START bit.
- I2C_TIME_OUT_INT: Triggered when the SCL stays high or low for more than I2C_TIME_OUT clocks.
- I2C_TRANS_COMPLETE_INT: Triggered when the Master or Slave detects a STOP bit.
- I2C_MASTER_TRAN_COMP_INT: Triggered when the Master sends or receives a byte.
- I2C_ARBITRATION_LOST_INT: Triggered when the Master's SCL is high, while the output value and input value of the SDA do not match.
- I2C_SLAVE_TRAN_COMP_INT: Triggered when the Slave sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when the Master deals with the END command.
- I2C_RXFIFO_OVF_INT: Triggered when RX FIFO of the I2C controller overflows.

- I2C_TXFIFO_EMPTY_INT: I2C TX FIFO interrupt. Triggered when the pointers of TX FIFO are less than I2C_TXFIFO_EMPTY_THRHD[4:0].
- I2C_RXFIFO_FULL_INT: I2C RX FIFO interrupt. Triggered when the pointers of RX FIFO are greater than I2C_RXFIFO_FULL_THRHD[4:0].

## 21.4 Register Summary

The addresses in this section are relative to the I2C base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | I2C0 | I2C1 | Acc |
|---|---|---|---|---|
| **Configuration registers** | | | | |
| I2C_CTR_REG | Transmission configuration register | 0x0004 | 0x0004 | R/W |
| I2C_TO_REG | Timeout control register | 0x000C | 0x000C | R/W |
| I2C_SLAVE_ADDR_REG | Configures the I2C slave address | 0x3FF53010 | 0x3FF67010 | R/W |
| I2C_RXFIFO_ST_REG | FIFO status register | 0x3FF53014 | 0x3FF67014 | RO |
| I2C_FIFO_CONF_REG | FIFO configuration register | 0x3FF53018 | 0x3FF67018 | R/W |
| **Status registers** | | | | |
| I2C_SR_REG | Describes I2C work status | 0x0008 | 0x0008 | RO |
| **Timing registers** | | | | |
| I2C_SDA_HOLD_REG | Configures the hold time after a negative SCL edge | 0x3FF53030 | 0x3FF67030 | R/W |
| I2C_SDA_SAMPLE_REG | Configures the sample time after a positive SCL edge | 0x3FF53034 | 0x3FF67034 | R/W |
| I2C_SCL_LOW_PERIOD_REG | Configures the low level width of the SCL clock | 0x3FF53000 | 0x3FF67000 | R/W |
| I2C_SCL_HIGH_PERIOD_REG | Configures the high level width of the SCL clock | 0x3FF53038 | 0x3FF67038 | R/W |
| I2C_SCL_START_HOLD_REG | Configures the delay between the SDA and SCL negative edge for a start condition | 0x3FF53040 | 0x3FF67040 | R/W |
| I2C_SCL_RSTART_SETUP_REG | Configures the delay between the positive edge of SCL and the negative edge of SDA | 0x3FF53044 | 0x3FF67044 | R/W |
| I2C_SCL_STOP_HOLD_REG | Configures the delay after the SCL clock edge for a stop condition | 0x3FF53048 | 0x3FF67048 | R/W |
| I2C_SCL_STOP_SETUP_REG | Configures the delay between the SDA and SCL positive edge for a stop condition | 0x3FF5304C | 0x3FF6704C | R/W |
| **Filter registers** | | | | |
| I2C_SCL_FILTER_CFG_REG | SCL filter configuration register | 0x3FF53050 | 0x3FF67050 | R/W |
| I2C_SDA_FILTER_CFG_REG | SDA filter configuration register | 0x3FF53054 | 0x3FF67054 | R/W |
| **Interrupt registers** | | | | |
| I2C_INT_RAW_REG | Raw interrupt status | 0x3FF53020 | 0x3FF67020 | RO |
| I2C_INT_CLR_REG | Interrupt clear bits | 0x3FF53024 | 0x3FF67024 | WO |
| I2C_INT_ENA_REG | Interrupt enable bits | 0x3FF53028 | 0x3FF67028 | R/W |
| I2C_INT_STATUS_REG | Interrupt status bits | 0x3FF5302C | 0x3FF6702C | RO |
| **Command registers** | | | | |
| I2C_COMD0_REG | I2C command register 0 | 0x3FF53058 | 0x3FF67058 | R/W |
| I2C_COMD1_REG | I2C command register 1 | 0x3FF5305C | 0x3FF6705C | R/W |
| I2C_COMD2_REG | I2C command register 2 | 0x3FF53060 | 0x3FF67060 | R/W |

| Name | Description | I2C0 | I2C1 | Acc |
|---|---|---|---|---|
| I2C_COMD3_REG | I2C command register 3 | 0x3FF53064 | 0x3FF67064 | R/W |
| I2C_COMD4_REG | I2C command register 4 | 0x3FF53068 | 0x3FF67068 | R/W |
| I2C_COMD5_REG | I2C command register 5 | 0x3FF5306C | 0x3FF6706C | R/W |
| I2C_COMD6_REG | I2C command register 6 | 0x3FF53070 | 0x3FF67070 | R/W |
| I2C_COMD7_REG | I2C command register 7 | 0x3FF53074 | 0x3FF67074 | R/W |
| I2C_COMD8_REG | I2C command register 8 | 0x3FF53078 | 0x3FF67078 | R/W |
| I2C_COMD9_REG | I2C command register 9 | 0x3FF5307C | 0x3FF6707C | R/W |
| I2C_COMD10_REG | I2C command register 10 | 0x3FF53080 | 0x3FF67080 | R/W |
| I2C_COMD11_REG | I2C command register 11 | 0x3FF53084 | 0x3FF67084 | R/W |
| I2C_COMD12_REG | I2C command register 12 | 0x3FF53088 | 0x3FF67088 | R/W |
| I2C_COMD13_REG | I2C command register 13 | 0x3FF5308C | 0x3FF6708C | R/W |
| I2C_COMD14_REG | I2C command register 14 | 0x3FF53090 | 0x3FF67090 | R/W |
| I2C_COMD15_REG | I2C command register 15 | 0x3FF53094 | 0x3FF67094 | R/W |

## 21.5 Registers

The addresses in this section are relative to the I2C base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 21.4 *Register Summary*.

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

**Register 21.1. I2C_SCL_LOW_PERIOD_REG (0x0000)**

| (reserved) | I2C_SCL_LOW_PERIOD | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SCL_LOW_PERIOD** This register is used to configure for how long SCL remains low in master mode, in APB clock cycles. (R/W)

**Register 21.2. I2C_CTR_REG (0x0004)**

| (reserved) | I2C_RX_LSB_FIRST | I2C_TX_LSB_FIRST | I2C_TRANS_START | I2C_MS_MODE | (reserved) | I2C_SAMPLE_SCL_LEVEL | I2C_SCL_FORCE_OUT | I2C_SDA_FORCE_OUT | |
|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 | 1 | 1 | Reset |

**I2C_RX_LSB_FIRST** This bit is used to control the storage mode for received data. (R/W)
1: receive data from the least significant bit;
0: receive data from the most significant bit.

**I2C_TX_LSB_FIRST** This bit is used to control the sending mode for data needing to be sent. (R/W)
1: send data from the least significant bit;
0: send data from the most significant bit.

**I2C_TRANS_START** Set this bit to start sending the data in txfifo. (R/W)

**I2C_MS_MODE** Set this bit to configure the module as an I2C Master. Clear this bit to configure the module as an I2C Slave. (R/W)

**I2C_SAMPLE_SCL_LEVEL** 1: sample SDA data on the SCL low level; 0: sample SDA data on the SCL high level. (R/W)

**I2C_SCL_FORCE_OUT** 0: direct output; 1: open drain output. (R/W)

**I2C_SDA_FORCE_OUT** 0: direct output; 1: open drain output. (R/W)

**Register 21.3. I2C_SR_REG (0x0008)**

| (reserved) | I2C_SCL_STATE_LAST | (reserved) | I2C_SCL_MAIN_STATE_LAST | I2C_TXFIFO_CNT | (reserved) | I2C_RXFIFO_CNT | (reserved) | I2C_BYTE_TRANS | I2C_SLAVE_ADDRESSED | I2C_BUS_BUSY | I2C_ARB_LOST | I2C_TIME_OUT | I2C_SLAVE_RW | I2C_ACK_REC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 28 | 27 | 26 24 | 23 18 | 17 14 | 13 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 |

**I2C_SCL_STATE_LAST** This field indicates the states of the state machine used to produce SCL. (RO)

0: Idle; 1: Start; 2: Negative edge; 3: Low; 4: Positive edge; 5: High; 6: Stop

**I2C_SCL_MAIN_STATE_LAST** This field indicates the states of the I2C module state machine. (RO)

0: Idle; 1: Address shift; 2: ACK address; 3: Rx data; 4: Tx data; 5: Send ACK; 6: Wait ACK

**I2C_TXFIFO_CNT** This field stores the amount of received data in RAM. (RO)

**I2C_RXFIFO_CNT** This field represents the amount of data needed to be sent. (RO)

**I2C_BYTE_TRANS** This field changes to 1 when one byte is transferred. (RO)

**I2C_SLAVE_ADDRESSED** When configured as an I2C Slave, and the address sent by the master is equal to the address of the slave, then this bit will be of high level. (RO)

**I2C_BUS_BUSY** 1: the I2C bus is busy transferring data; 0: the I2C bus is in idle state. (RO)

**I2C_ARB_LOST** When the I2C controller loses control of SCL line, this register changes to 1. (RO)

**I2C_TIME_OUT** When the I2C controller takes more than I2C_TIME_OUT clocks to receive a data bit, this field changes to 1. (RO)

**I2C_SLAVE_RW** When in slave mode, 1: master reads from slave; 0: master writes to slave. (RO)

**I2C_ACK_REC** This register stores the value of the received ACK bit. (RO)

**Register 21.4. I2C_TO_REG (0x000c)**

| (reserved) | I2C_TIME_OUT_REG | |
|---|---|---|
| 31 20 | 19 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_TIME_OUT_REG** This register is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

**Register 21.5. I2C_SLAVE_ADDR_REG (0x0010)**

| I2C_SLAVE_ADDR_10BIT_EN | (reserved) | I2C_SLAVE_ADDR | |
|---|---|---|---|
| 31 | 30 … 15 | 14 … 0 | |
| 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SLAVE_ADDR_10BIT_EN** This field is used to enable the slave 10-bit addressing mode in master mode. (R/W)

**I2C_SLAVE_ADDR** When configured as an I2C Slave, this field is used to configure the slave address. (R/W)

**Register 21.6. I2C_RXFIFO_ST_REG (0x0014)**

| (reserved) | I2C_RXFIFO_TXFIFO_END_ADDR | I2C_RXFIFO_TXFIFO_START_ADDR | I2C_RXFIFO_END_ADDR | I2C_RXFIFO_START_ADDR | |
|---|---|---|---|---|---|
| 31 … 20 | 19 … 15 | 14 … 10 | 9 … 5 | 4 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | Reset |

**I2C_TXFIFO_END_ADDR** This is the offset address of the last sent data, as described in nonfifo_tx_thres register. The value refreshes when I2C_TX_SEND_EMPTY_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

**I2C_TXFIFO_START_ADDR** This is the offset address of the first sent data, as described in nonfifo_tx_thres register. (RO)

**I2C_RXFIFO_END_ADDR** This is the offset address of the last received data, as described in nonfifo_rx_thres_register. This value refreshes when I2C_RX_REC_FULL_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

**I2C_RXFIFO_START_ADDR** This is the offset address of the last received data, as described in nonfifo_rx_thres_register. (RO)

**Register 21.7. I2C_FIFO_CONF_REG (0x0018)**

| (reserved) | I2C_NONFIFO_TX_THRES | I2C_NONFIFO_RX_THRES | (reserved) | I2C_FIFO_ADDR_CFG_EN | I2C_NONFIFO_EN | I2C_TXFIFO_EMPTY_THRHD | I2C_RXFIFO_FULL_THRHD | |
|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 20 | 19 14 | 13 12 | 11 | 10 | 9 5 | 4 0 | |
| 0 0 0 0 0 0 | 0x15 | 0x15 | 0 0 | 0 | 0 | 0 | 0 | Reset |

**I2C_NONFIFO_TX_THRES** When I2C sends more than nonfifo_tx_thres bytes of data, it will generate a tx_send_empty_int_raw interrupt and update the current offset address of the sent data. (R/W)

**I2C_NONFIFO_RX_THRES** When I2C receives more than nonfifo_rx_thres bytes of data, it will generate a rx_send_full_int_raw interrupt and update the current offset address of the received data. (R/W)

**I2C_FIFO_ADDR_CFG_EN** When this bit is set to 1, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM. (R/W)

**I2C_NONFIFO_EN** Set this bit to enable APB nonfifo access. (R/W)

**I2C_TXFIFO_EMPTY_THRHD** Configures the TX FIFO threshold in non-FIFO mode. When the TX FIFO count is greater than I2C_TXFIFO_EMPTY_THRHD [4:0], I2C_TXFIFO_EMPTY_INT_RAW bit is valid. (R/W)

**I2C_RXFIFO_FULL_THRHD** Configures the RX FIFO threshold in non-FIFO mode. When the RX FIFO count is greater than I2C_RXFIFO_FULL_THRHD [4:0], I2C_RXFIFO_FULL_INT_RAW bit is valid. (R/W)

**Register 21.8. I2C_INT_RAW_REG (0x0020)**

| (reserved) | I2C_TX_SEND_EMPTY_INT_RAW | I2C_RX_REC_FULL_INT_RAW | I2C_ACK_ERR_INT_RAW | I2C_TRANS_START_INT_RAW | I2C_TIME_OUT_INT_RAW | I2C_TRANS_COMPLETE_INT_RAW | I2C_MASTER_TRAN_COMP_INT_RAW | I2C_ARBITRATION_LOST_INT_RAW | I2C_SLAVE_TRAN_COMP_INT_RAW | I2C_END_DETECT_INT_RAW | I2C_RXFIFO_OVF_INT_RAW | I2C_TXFIFO_EMPTY_INT_RAW | I2C_RXFIFO_FULL_INT_RAW | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**I2C_TX_SEND_EMPTY_INT_RAW** The raw interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

**I2C_RX_REC_FULL_INT_RAW** The raw interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

**I2C_ACK_ERR_INT_RAW** The raw interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

**I2C_TRANS_START_INT_RAW** The raw interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

**I2C_TIME_OUT_INT_RAW** The raw interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

**I2C_TRANS_COMPLETE_INT_RAW** The raw interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

**I2C_MASTER_TRAN_COMP_INT_RAW** The raw interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

**I2C_ARBITRATION_LOST_INT_RAW** The raw interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

**I2C_SLAVE_TRAN_COMP_INT_RAW** The raw interrupt status bit for the I2C_SLAVE_TRAN_COMP_INT interrupt. (RO)

**I2C_END_DETECT_INT_RAW** The raw interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

**I2C_RXFIFO_OVF_INT_RAW** The raw interrupt status bit for the I2C_RXFIFO_OVF_INT interrupt. (RO)

**I2C_TXFIFO_EMPTY_INT_RAW** The raw interrupt status bit for the I2C_TXFIFO_EMPTY_INT interrupt. (RO)

**I2C_RXFIFO_FULL_INT_RAW** The raw interrupt status bit for the I2C_RXFIFO_FULL_INT interrupt. (RO)

**Register 21.9. I2C_INT_CLR_REG (0x0024)**

| (reserved) | I2C_TX_SEND_EMPTY_INT_CLR | I2C_RX_REC_FULL_INT_CLR | I2C_ACK_ERR_INT_CLR | I2C_TRANS_START_INT_CLR | I2C_TIME_OUT_INT_CLR | I2C_TRANS_COMPLETE_INT_CLR | I2C_MASTER_TRAN_COMP_INT_CLR | I2C_ARBITRATION_LOST_INT_CLR | I2C_SLAVE_TRAN_COMP_INT_CLR | I2C_END_DETECT_INT_CLR | I2C_RXFIFO_OVF_INT_CLR | I2C_TXFIFO_EMPTY_INT_CLR | I2C_RXFIFO_FULL_INT_CLR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**I2C_TX_SEND_EMPTY_INT_CLR** Set this bit to clear the I2C_TX_SEND_EMPTY_INT interrupt. (WO)

**I2C_RX_REC_FULL_INT_CLR** Set this bit to clear the I2C_RX_REC_FULL_INT interrupt. (WO)

**I2C_ACK_ERR_INT_CLR** Set this bit to clear the I2C_ACK_ERR_INT interrupt. (WO)

**I2C_TRANS_START_INT_CLR** Set this bit to clear the I2C_TRANS_START_INT interrupt. (WO)

**I2C_TIME_OUT_INT_CLR** Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WO)

**I2C_TRANS_COMPLETE_INT_CLR** Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WO)

**I2C_MASTER_TRAN_COMP_INT_CLR** Set this bit to clear the I2C_MASTER_TRAN_COMP_INT interrupt. (WO)

**I2C_ARBITRATION_LOST_INT_CLR** Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WO)

**I2C_SLAVE_TRAN_COMP_INT_CLR** Set this bit to clear the I2C_SLAVE_TRAN_COMP_INT interrupt. (WO)

**I2C_END_DETECT_INT_CLR** Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

**I2C_RXFIFO_OVF_INT_CLR** Set this bit to clear the I2C_RXFIFO_OVF_INT interrupt. (WO)

**I2C_TXFIFO_EMPTY_INT_CLR** Set this bit to clear the I2C_TXFIFO_EMPTY_INT interrupt. (WO)

**I2C_RXFIFO_FULL_INT_CLR** Set this bit to clear the I2C_RXFIFO_FULL_INT interrupt. (WO)

**Register 21.10. I2C_INT_ENA_REG (0x0028)**

| (reserved) | I2C_TX_SEND_EMPTY_INT_ENA | I2C_RX_REC_FULL_INT_ENA | I2C_ACK_ERR_INT_ENA | I2C_TRANS_START_INT_ENA | I2C_TIME_OUT_INT_ENA | I2C_TRANS_COMPLETE_INT_ENA | I2C_MASTER_TRAN_COMP_INT_ENA | I2C_ARBITRATION_LOST_INT_ENA | I2C_SLAVE_TRAN_COMP_INT_ENA | I2C_END_DETECT_INT_ENA | I2C_RXFIFO_OVF_INT_ENA | I2C_TXFIFO_EMPTY_INT_ENA | I2C_RXFIFO_FULL_INT_ENA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**I2C_TX_SEND_EMPTY_INT_ENA** The interrupt enable bit for the I2C_TX_SEND_EMPTY_INT interrupt. (R/W)

**I2C_RX_REC_FULL_INT_ENA** The interrupt enable bit for the I2C_RX_REC_FULL_INT interrupt. (R/W)

**I2C_ACK_ERR_INT_ENA** The interrupt enable bit for the I2C_ACK_ERR_INT interrupt. (R/W)

**I2C_TRANS_START_INT_ENA** The interrupt enable bit for the I2C_TRANS_START_INT interrupt. (R/W)

**I2C_TIME_OUT_INT_ENA** The interrupt enable bit for the I2C_TIME_OUT_INT interrupt. (R/W)

**I2C_TRANS_COMPLETE_INT_ENA** The interrupt enable bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

**I2C_MASTER_TRAN_COMP_INT_ENA** The interrupt enable bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (R/W)

**I2C_ARBITRATION_LOST_INT_ENA** The interrupt enable bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

**I2C_SLAVE_TRAN_COMP_INT_ENA** The interrupt enable bit for the I2C_SLAVE_TRAN_COMP_INT interrupt. (R/W)

**I2C_END_DETECT_INT_ENA** The interrupt enable bit for the I2C_END_DETECT_INT interrupt. (R/W)

**I2C_RXFIFO_OVF_INT_ENA** The interrupt enable bit for the I2C_RXFIFO_OVF_INT interrupt. (R/W)

**I2C_TXFIFO_EMPTY_INT_ENA** The interrupt enable bit for the I2C_TXFIFO_EMPTY_INT interrupt. (R/W)

**I2C_RXFIFO_FULL_INT_ENA** The interrupt enable bit for the I2C_RXFIFO_FULL_INT interrupt. (R/W)

**Register 21.11. I2C_INT_STATUS_REG (0x002c)**

| 31 | | | | | | | | | | | | | | | | | | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | I2C_TX_SEND_EMPTY_INT_ST | I2C_RX_REC_FULL_INT_ST | I2C_ACK_ERR_INT_ST | I2C_TRANS_START_INT_ST | I2C_TIME_OUT_INT_ST | I2C_TRANS_COMPLETE_INT_ST | I2C_MASTER_TRAN_COMP_INT_ST | I2C_ARBITRATION_LOST_INT_ST | I2C_SLAVE_TRAN_COMP_INT_ST | I2C_END_DETECT_INT_ST | I2C_RXFIFO_OVF_INT_ST | I2C_TXFIFO_EMPTY_INT_ST | I2C_RXFIFO_FULL_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**I2C_TX_SEND_EMPTY_INT_ST** The masked interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

**I2C_RX_REC_FULL_INT_ST** The masked interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

**I2C_ACK_ERR_INT_ST** The masked interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

**I2C_TRANS_START_INT_ST** The masked interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

**I2C_TIME_OUT_INT_ST** The masked interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

**I2C_TRANS_COMPLETE_INT_ST** The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

**I2C_MASTER_TRAN_COMP_INT_ST** The masked interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

**I2C_ARBITRATION_LOST_INT_ST** The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

**I2C_SLAVE_TRAN_COMP_INT_ST** The masked interrupt status bit for the I2C_SLAVE_TRAN_COMP_INT interrupt. (RO)

**I2C_END_DETECT_INT_ST** The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

**I2C_RXFIFO_OVF_INT_ST** The masked interrupt status bit for the I2C_RXFIFO_OVF_INT interrupt. (RO)

**I2C_TXFIFO_EMPTY_INT_ST** The masked interrupt status bit for the I2C_TXFIFO_EMPTY_INT interrupt. (RO)

**I2C_RXFIFO_FULL_INT_ST** The masked interrupt status bit for the I2C_RXFIFO_FULL_INT interrupt. (RO)

**Register 21.12. I2C_SDA_HOLD_REG (0x0030)**

| (reserved) | I2C_SDA_HOLD_TIME | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SDA_HOLD_TIME** This register is used to configure the time to hold the data after the negative edge of SCL, in APB clock cycles. (R/W)

**Register 21.13. I2C_SDA_SAMPLE_REG (0x0034)**

| (reserved) | I2C_SDA_SAMPLE_TIME | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SDA_SAMPLE_TIME** This register is used to configure for how long SDA is sampled, in APB clock cycles. (R/W)

**Register 21.14. I2C_SCL_HIGH_PERIOD_REG (0x0038)**

| (reserved) | I2C_SCL_HIGH_PERIOD | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SCL_HIGH_PERIOD** This register is used to configure for how long SCL remains high in master mode, in APB clock cycles. (R/W)

**Register 21.15. I2C_SCL_START_HOLD_REG (0x0040)**

| (reserved) | I2C_SCL_START_HOLD_TIME | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 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 |

**I2C_SCL_START_HOLD_TIME** This register is used to configure the time between the negative edge of SDA and the negative edge of SCL for a START condition, in APB clock cycles. (R/W)

**Register 21.16. I2C_SCL_RSTART_SETUP_REG (0x0044)**

| (reserved) | I2C_SCL_RSTART_SETUP_TIME | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 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 |

**I2C_SCL_RSTART_SETUP_TIME** This register is used to configure the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition, in APB clock cycles. (R/W)

**Register 21.17. I2C_SCL_STOP_HOLD_REG (0x0048)**

| (reserved) | I2C_SCL_STOP_HOLD_TIME | |
|---|---|---|
| 31 … 14 | 13 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SCL_STOP_HOLD_TIME** This register is used to configure the delay after the STOP condition, in APB clock cycles. (R/W)

**Register 21.18. I2C_SCL_STOP_SETUP_REG (0x004C)**

| (reserved) | I2C_SCL_STOP_SETUP_TIME | |
|---|---|---|
| 31 … 10 | 9 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_SCL_STOP_SETUP_TIME** This register is used to configure the time between the positive edge of SCL and the positive edge of SDA, in APB clock cycles. (R/W)

**Register 21.19. I2C_SCL_FILTER_CFG_REG (0x0050)**

| (reserved) | I2C_SCL_FILTER_EN | I2C_SCL_FILTER_THRES | |
|---|---|---|---|
| 31 … 4 | 3 | 2 … 0 | |
| 0 0 0 0 0 0 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 |

**I2C_SCL_FILTER_EN** This is the filter enable bit for SCL. (R/W)

**I2C_SCL_FILTER_THRES** When a pulse on the SCL input has smaller width than this register value in APB clock cycles, the I2C controller will ignore that pulse. (R/W)

**Register 21.20. I2C_SDA_FILTER_CFG_REG (0x0054)**

| (reserved) | I2C_SDA_FILTER_EN | I2C_SDA_FILTER_THRES | |
|---|---|---|---|
| 31 … 4 | 3 | 2 … 0 | |
| 0 0 0 0 0 0 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 |

**I2C_SDA_FILTER_EN** This is the filter enable bit for SDA. (R/W)

**I2C_SDA_FILTER_THRES** When a pulse on the SDA input has smaller width than this register value in APB clock cycles, the I2C controller will ignore that pulse. (R/W)

**Register 21.21. I2C_COMD*n*_REG (*n*: 0-15) (0x58+4**n*)**

| I2C_COMMAND*n*_DONE | (reserved) | I2C_COMMAND*n* | |
|---|---|---|---|
| 31 | 30 … 14 | 13 … 0 | |
| 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**I2C_COMMAND*n*_DONE** When command *n* is done in I2C Master mode, this bit changes to high level. (R/W)

**I2C_COMMAND*n*** This is the content of command *n*. It consists of three parts: (R/W)
op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END.
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 for more information.

# Chapter 22

# I2S Controller (I2S)

## 22.1 Overview

The I2S bus provides a flexible communication interface for streaming digital data in multimedia applications, especially digital audio applications. The ESP32 includes two I2S interfaces: I2S0 and I2S1.

The I2S standard bus defines three signals: a clock signal, a channel selection signal, and a serial data signal. A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication. The I2S modules on the ESP32 provide separate transmit and receive channels for high performance.

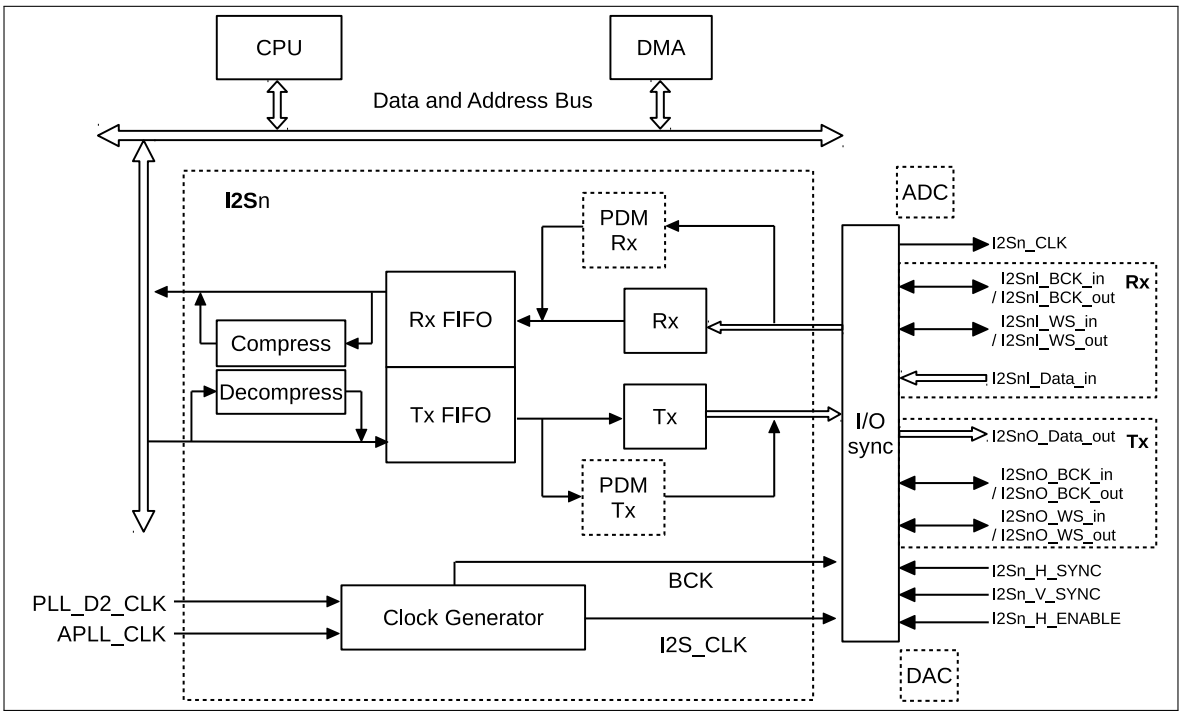



Figure 22.1-1. I2S System Block Diagram

Figure 22.1-1 is the system block diagram of the ESP32 I2S module. In the figure above, the value of "*n*" can be either 0 or 1. There are two independent I2S modules embedded in ESP32, namely I2S0 and I2S1. Each I2S module contains a Tx (transmit) unit and a Rx (receive) unit. Both the Tx unit and the Rx unit have a three-wire interface that includes a clock line, a channel selection line and a serial data line. The serial data line of the Tx unit is fixed as output, and the serial data line of the Rx unit is fixed as input. The clock line and the channel selection line of the Tx and Rx units can be configured to both master transmitting mode and slave receiving mode. In the LCD mode, the serial data line extends to the parallel data bus. Both the Tx unit and the Rx unit have a 32-bit-wide FIFO with a depth of 64. Besides, only I2S0 supports on-chip DAC/ADC modes, as

well as receiving and transmitting PDM signals.

The right side of Figure 22.1-1 shows the signal bus of the I2S module. The signal naming rule of the Rx and Tx units is I2S*n*A_B_C, where "*n*" stands for either I2S0 or I2S1; "A" represents the direction of I2S module's data bus signal, "I" represents input, "O" represents output; "B" represents signal function; "C" represents the signal direction, "in" means that the signal is input into the I2S module, while "out" means that the I2S module outputs the signal. For a detailed description of the I2S signal bus, please refer to Table 22.1-1. Table 22.1-1 describes the signal bus of the I2S module. Except for the I2S*n*_CLK signal, all other signals are mapped to the chip pin via the GPIO matrix and IO MUX. The I2S*n*_CLK signal is mapped to the chip pin via the IO_MUX. For details, please refer to the chapter about IO_MUX and the GPIO Matrix.

**Table 22.1-1. I2S Signal Bus Description**

| Signal Bus | Signal Direction | Data Signal Direction |
|---|---|---|
| I2S*n*I_BCK_in | In slave mode, I2S module inputs signals. | I2S module receives data. |
| I2S*n*I_BCK_out | In master mode, I2S module outputs signals. | I2S module receives data. |
| I2S*n*I_WS_in | In slave mode, I2S module inputs signals. | I2S module receives data. |
| I2S*n*I_WS_out | In master mode, I2S module outputs signals. | I2S module receives data. |
| I2S*n*I_Data_in[1] | I2S module inputs signals. | In I2S mode, I2S*n*I_Data_in[15] is the serial data bus of I2S. In LCD mode, the data bus width can be configured as needed. |
| I2S*n*O_Data_out[1] | I2S module outputs signals. | In I2S mode, I2S*n*O_Data_out[23] is the serial data bus of I2S. In LCD mode, the data bus width can be configured as needed. |
| I2S*n*O_BCK_in | In slave mode, I2S module inputs signals. | I2S module sends data. |
| I2S*n*O_BCK_out | In master mode, I2S module outputs signals. | I2S module sends data. |
| I2S*n*O_WS_in | In slave mode, I2S module inputs signals. | I2S module sends data. |
| I2S*n*O_WS_out | In master mode, I2S module outputs signals. | I2S module sends data. |
| I2S*n*_CLK[2] | I2S module outputs signals. | It is used as a clock source for peripheral chips. |
| I2S*n*_H_SYNC | In Camera mode, I2S module inputs signals. | The signals are sent from the Camera. |
| I2S*n*_V_SYNC | | |
| I2S*n*_H_ENABLE | | |

**Note:**

1. Assume that the bit width of the input/output signal is *N*, the input signal should be configured to I2S*n*I_Data_in[*N*-1:0], and the output signal to I2S*n*O_Data_out[23:23-*N*+1]. Generally, for input signals, *N*=8 or 16; while for output signals, *N*=8, 16 or 24 (note that I2S1 does not support 24-bit width).
2. I2S*n*_CLK can only be mapped to GPIO0, U0RXD (GPIO3) or U0TXD (GPIO1) by selecting GPIO functions CLK_OUT1, CLK_OUT2, or CLK_OUT3. For more information, see Table 6.10-1: *IO_MUX Pin Summary*.

## 22.2 Features

I2S mode

- Configurable high-precision output clock
- Full-duplex and half-duplex data transmit and receive modes
- Supports multiple digital audio standards
- Embedded A-law compression/decompression module
- Configurable clock signal
- Supports PDM signal input and output
- Configurable data transmit and receive modes

LCD mode

- Supports multiple LCD modes, including external LCD
- Supports external Camera
- Supports on-chip DAC/ADC modes

I2S interrupts

- Standard I2S interface interrupts
- I2S DMA interface interrupts

## 22.3 The Clock of I2S Module

As is shown in Figure 22.3-1, I2S*n*_CLK, as the master clock of I2S module, is derived from the 160 MHz clock PLL_F160M_CLK or the configurable analog PLL output clock APLL_CLK. The serial clock (BCK) of the I2S module is derived from I2S*n*_CLK. The I2S_CLKA_ENA bit of register I2S_CLKM_CONF_REG is used to select either PLL_F160M_CLK or APLL_CLK as the clock source for I2S*n*. PLL_F160_CLK is used as the clock source for I2S*n*, by default.

**Notice:**

- When using PLL_F160M_CLK as the clock source, it is not recommended to divide it using decimals. For high performance audio applications, the analog PLL output clock source APLL_CLK must be used to acquire highly accurate I2S*n*_CLK and BCK. For further details, please refer to the chapter entitled Reset and Clock.
- When ESP32 I2S works in slave mode, the master must use I2S*n*_CLK as the master clock and $f_{i2s}$ >= 8 * $f_{BCK}$.

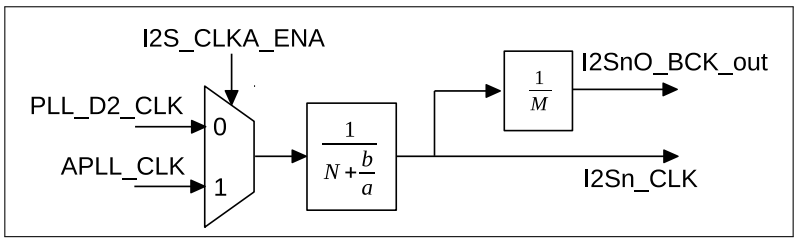



Figure 22.3-1. I2S Clock

The relation between I2S*n*_CLK frequency $f_{i2s}$ and the divider clock source frequency $f_{pll}$ can be seen in the equation below:

$$f_{i2s} = \frac{f_{pll}}{N + \frac{b}{a}}$$

"N", whose value is >=2, corresponds to the REG _CLKM_DIV_NUM [7:0] bits of register I2S_CLKM_CONF_REG , "b" is the I2S_CLKM_DIV_B[5:0] bit and "a" is the I2S_CLKM_DIV_A[5:0] bit.

In master mode, the serial clock BCK in the I2S module is derived from I2S*n*_CLK, that is:

$$f_{BCK} = \frac{f_{i2s}}{M}$$

In master transmitting mode, "M", whose value is >=2, is the I2S_TX_BCK_DIV_NUM[5:0] bit of register I2S_SAMPLE
_RATE_CONF_REG. In master receiving mode, "M" is the I2S_RX_BCK_DIV_NUM[5:0] bit of register I2S_SAMPLE
_RATE_CONF_REG.

## 22.4 I2S Mode

The ESP32 I2S module integrates an A-law compression/decompression module to enable compression/decompression of the received audio data. The RX_PCM_BYPASS bit and the TX_PCM_BYPASS bit of register I2S_CONF1_REG should be cleared when using the A-law compression/decompression module.

### 22.4.1 Supported Audio Standards

In the I2S bus, BCK is the serial clock, WS is the left- /right-channel selection signal (also called word select signal), and SD is the serial data signal for transmitting/receiving digital audio data. WS and SD signals in the I2S module change on the falling edge of BCK, while the SD signal can be sampled on the rising edge of BCK. If the I2S_RX_RIGHT_FIRST bit and the I2S_TX_RIGHT_FIRST bit of register I2S_CONF_REG are set to 1, the I2S module is configured to receive and transmit right-channel data first. Otherwise, the I2S module receives and transmits left-channel data first.

#### 22.4.1.1 Philips Standard

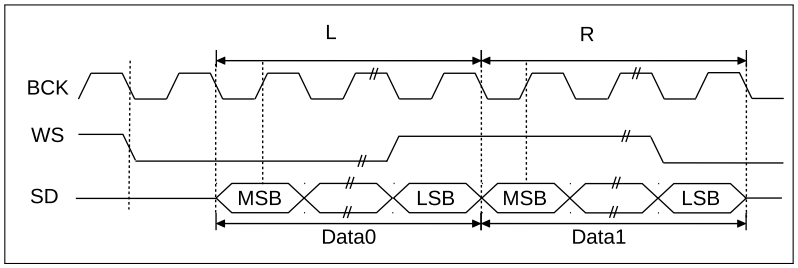



Figure 22.4-1. Philips Standard

As is shown in Figure 22.4-1, the Philips I2S bus specifications require that the WS signal starts to change one BCK clock cycle earlier than the SD signal on BCK falling edge, which means the WS signal becomes valid one clock cycle before the first bit of data transfer on the current channel, and changes one clock cycle earlier than the end of data transfer on the current channel. The SD signal line transmits the most significant bit of audio data first. If the I2S_RX_MSB_SHIFT bit and the I2S_TX_MSB_SHIFT bit of register I2S_CONF_REG are set to 1, respectively, the I2S module will use the Philips standard when receiving and transmitting data.

#### 22.4.1.2 MSB Alignment Standard

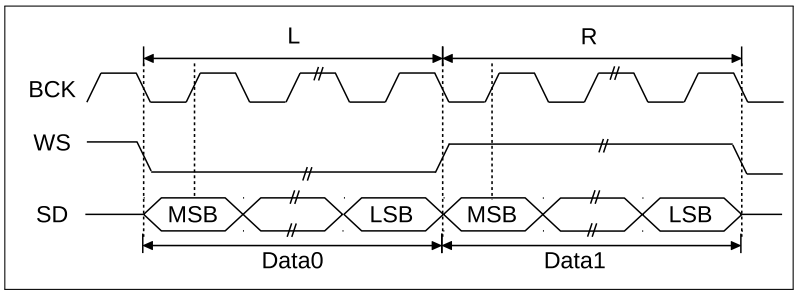



Figure 22.4-2. MSB Alignment Standard

The MSB alignment standard is shown in Figure 22.4-2. WS and SD signals both change simultaneously on the falling edge of BCK under the MSB alignment standard. The WS signal continues until the end of the current channel-data transmission, and the SD signal line transmits the most significant bit of audio data first. If the I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT bits of register I2S_CONF_REG are cleared, the I2S module will use the MSB alignment standard when receiving and transmitting data.

#### 22.4.1.3 PCM Standard

As is shown in Figure 22.4-3, under the short frame synchronization mode of the PCM standard, the WS signal starts to change a BCK clock cycle earlier than the SD signal, which means that the WS signal takes effect a clock cycle earlier than the first bit of the current channel-data transmission and continues for one extra BCK clock cycle. The SD signal line transmits the most significant bit of audio data first. If the I2S_RX_SHORT_SYNC and I2S_TX_SHORT_SYNC bits of register I2S_CONF_REG are set, the I2S module will receive and transmit data in the short frame synchronization mode.

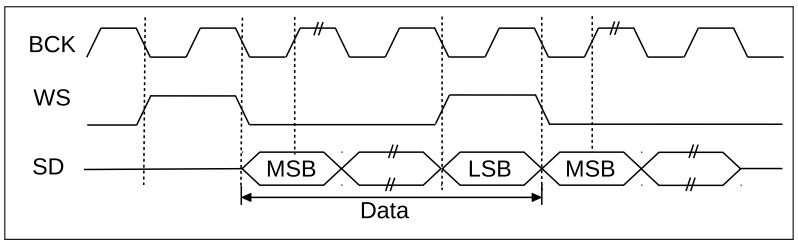



Figure 22.4-3. PCM Standard

## 22.4.2 Module Reset

The four low-order bits in register I2S_CONF_REG, that is, I2S_TX_RESET, I2S_RX_RESET, I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET reset the receive module, the transmit module and the corresponding FIFO buffer, respectively. In order to finish a reset operation, the corresponding bit should be set and then cleared by software.

## 22.4.3 FIFO Operation

The data read/write packet length for a FIFO operation is 32 bits. The data packet format for the FIFO buffer can be configured using configuration registers. As shown in Figure 22.1-1, both sent and received data should be written into FIFO first and then read from FIFO. There are two approaches to accessing the FIFO; one is to directly access the FIFO using a CPU, the other is to access the FIFO using a DMA controller.

Generally, both the I2S_RX_FIFO_MOD_FORCE_EN bit and I2S_TX_FIFO_MOD_FORCE_EN bits of register I2S_
FIFO_CONF_REG should be set to 1. I2S_TX_DATA_NUM[5:0] bit and I2S_RX_DATA_NUM[5:0] are used to control the length of the data that have been sent, received and buffered. Hardware inspects the received-data length RX_LEN and the transmitted-data length TX_LEN. Both the received and the transmitted data are buffered in the FIFO method.

When RX_LEN is greater than I2S_RX_DATA_NUM[5:0], the received data, which is buffered in FIFO, has reached the set threshold and needs to be read out to prevent an overflow. When TX_LEN is less than I2S_TX_DATA_NUM[5:0], the transmitted data, which is buffered in FIFO, has not reached the set threshold and software can continue feeding data into FIFO.

## 22.4.4 Sending Data

The ESP32 I2S module carries out a data-transmit operation in three stages:

- Read data from internal storage and transfer it to FIFO
- Read data to be sent from FIFO
- Clock out data serially, or in parallel, as configured by the user

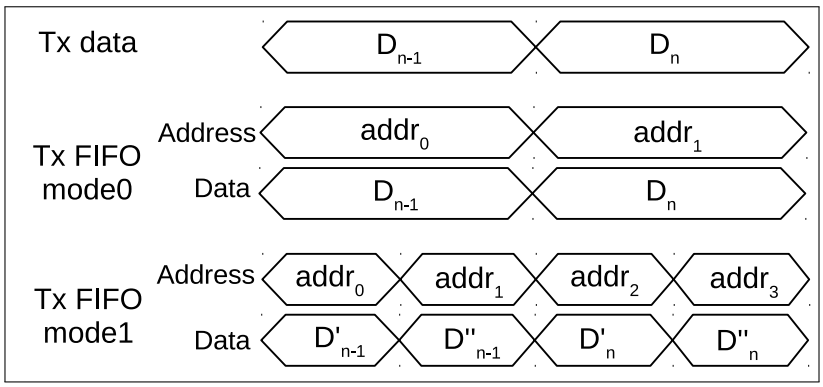



Figure 22.4-4. Tx FIFO Data Mode

Table 22.4-1. Register Configuration

| | I2S_TX_FIFO_MOD[2:0] | Description |
|---|---|---|
| Tx FIFO mode0 | 0 | 16-bit dual channel data |
| | 2 | 32-bit dual channel data |
| | 3 | 32-bit single channel data |
| Tx FIFO mode1 | 1 | 16-bit single channel data |

At the first stage, there are two modes for data to be sent and written into FIFO. In Tx FIFO mode0, the Tx data-to-be-sent are written into FIFO according to the time order. In Tx FIFO mode1, the data-to-be-sent are divided into 16 high- and 16 low-order bits. Then, both the 16 high- and 16 low-order bits are recomposed and written into FIFO. The details are shown in Figure 22.4-4 with the corresponding registers listed in Table 22.4-1. $D_n^{'}$ consists of 16 high-order bits of $D_n$ and 16 zeros. $D_n^{''}$ consists of 16 low-order bits of $D_n$ and 16 zeros. That is to say, $D_n^{'} = \{D_n[31:16], 16'h0\}$, $D_n^{''} = \{D_n[15:0], 16'h0\}$.

At the second stage, the system reads data that will be sent from FIFO, according to the relevant register configuration. The mode in which the system reads data from FIFO is relevant to the configuration of I2S_TX_FIFO_MOD[2.0] and I2S_TX_CHAN_MOD[2:0]. I2S_TX_FIFO_MOD[2.0] determines whether the data are 16-bit or 32-bit, as shown in Table 22.4-1, while I2S_TX_CHAN_MOD[2:0] determines the format of the data-to-be-sent, as shown in Table 22.4-2.

Table 22.4-2. Send Channel Mode

| I2S_TX_CHAN_MOD[2:0] | Description |
|---|---|
| 0 | Dual channel mode |
| 1 | Mono mode<br>When I2S_TX_MSB_RIGHT equals 0, the left-channel data are "holding" their values and the right-channel data change into the left-channel data.<br>When I2S_TX_MSB_RIGHT equals 1, the right-channel data are "holding" their values and the left-channel data change into the right-channel data. |
| 2 | Mono mode<br>When I2S_TX_MSB_RIGHT equals 0, the right-channel data are "holding" their values and the left-channel data change into the right-channel data. |

| I2S_TX_CHAN_MOD[2:0] | Description |
|---|---|
| | When I2S_TX_MSB_RIGHT equals 1, the left-channel data are "holding" their values and the right-channel data change into the left-channel data. |
| 3 | Mono mode<br>When I2S_TX_MSB_RIGHT equals 0, the left-channel data are constants in the range of REG[31:0].<br>When I2S_TX_MSB_RIGHT equals 1, the right-channel data are constants in the range of REG[31:0]. |
| 4 | Mono mode<br>When I2S_TX_MSB_RIGHT equals 0, the right-channel data are constants in the range of REG[31:0].<br>When I2S_TX_MSB_RIGHT equals 1, the left-channel data are constants in the range of REG[31:0]. |

REG[31:0] is the value of register I2S_CONF_SINGLE_DATA_REG[31:0].

The output of the third stage is determined by the mode of the I2S and I2S_TX_BITS_MOD[5:0] bits of register I2S_SAMPLE_RATE_CONF_REG.

### 22.4.5 Receiving Data

The data-receive phase of the ESP32 I2S module consists of another three stages:

- The input serial-bit stream is transformed into a 64-bit parallel-data stream in I2S mode. In LCD mode, the input parallel-data stream will be extended to a 64-bit parallel-data stream.
- Received data are written into FIFO.
- Data are read from FIFO by CPU/DMA and written into the internal memory.

At the first stage of receiving data, the received-data stream is expanded to a zero-padded parallel-data stream with 32 high-order bits and 32 low-order bits, according to the level of the I2S*n*I_WS_out (or I2S*n*I_WS_in) signal. The I2S_RX_MSB_RIGHT bit of register I2S_CONF_REG is used to determine how the data are to be expanded.

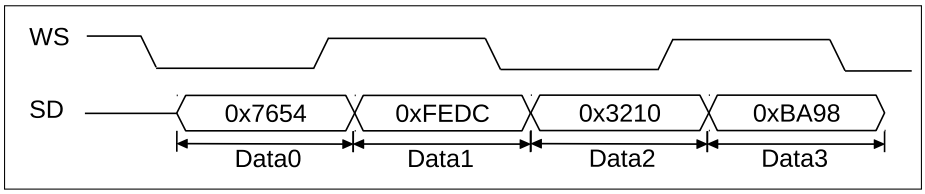



Figure 22.4-5. The First Stage of Receiving Data

For example, as is shown in Figure 22.4-5, if the width of serial data is 16 bits, when I2S_RX_RIGHT_FIRST equals 1, Data0 will be discarded and I2S will start receiving data from Data1. If I2S_RX_MSB_RIGHT equals 1, data of the first stage would be $\{0xFEDC0000, 0x32100000\}$. If I2S_RX_MSB_RIGHT equals 0, data of the first stage would be $\{0x32100000, 0xFEDC0000\}$. When I2S_RX_RIGHT_FIRST equals 0, I2S will start receiving data from Data0. If I2S_RX_MSB_RIGHT equals 1, data of the first stage would be $\{0xFEDC0000, 0x76540000\}$. If I2S_RX_MSB_RIGHT equals 0, data of the first stage would be $\{0x76540000, 0xFEDC0000\}$.

As is shown in Table 22.4-3 and Figure 22.4-6, at the second stage, the received data of the Rx unit is written into FIFO. There are four modes of writing received data into FIFO. Each mode corresponds to a value of I2S_RX_FIFO_MOD[2:0] bit.

Table 22.4-3. Modes of Writing Received Data into FIFO and the Corresponding Register Configuration

| I2S_RX_FIFO_MOD[2:0] | Data format |
|---|---|
| 0 | 16-bit dual channel data |
| 1 | 16-bit single channel data |
| 2 | 32-bit dual channel data |
| 3 | 32-bit single channel data |

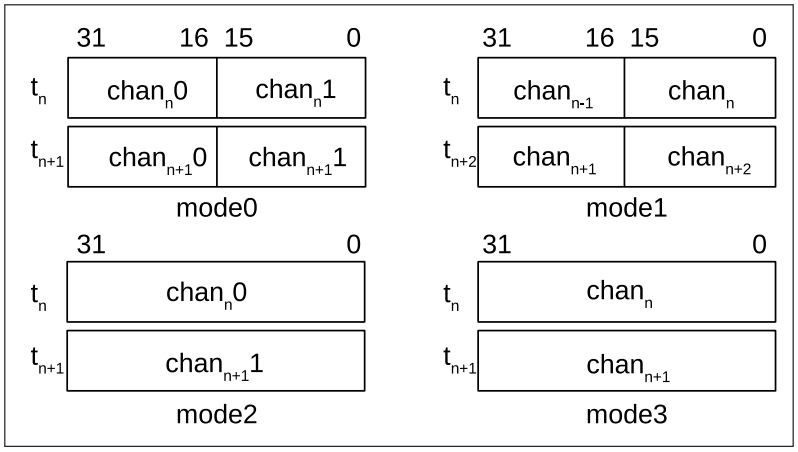



Figure 22.4-6. Modes of Writing Received Data into FIFO

At the third stage, CPU or DMA will read data from FIFO and write them into the internal memory directly. The register configuration that each mode corresponds to is shown in Table 22.4-4.

Table 22.4-4. The Register Configuration to Which the Four Modes Correspond

| I2S_RX_MSB_RIGHT | I2S_RX_CHAN_MOD | mode0 | mode1 | mode2 | mode3 |
|---|---|---|---|---|---|
| 0 | 0 | left channel + right channel | - | left channel + right channel | - |
| | 1 | | left channel + left channel | | left channel + left channel |
| | 2 | | right channel + right channel | | right channel + right channel |
| | 3 | | - | | - |
| 1 | 0 | right channel + left channel | - | right channel + left channel | - |
| | 1 | | right channel + right channel | | right channel + right channel |
| | 2 | | left channel + left channel | | left channel + left channel |
| | 3 | | - | | - |

### 22.4.6 I2S Master/Slave Mode

The ESP32 I2S module can be configured to act as a master or slave device on the I2S bus. The module supports slave transmitter and receiver configurations in addition to master transmitter and receiver configurations. All these modes can support full-duplex and half-duplex communication over the I2S bus.

I2S_RX_SLAVE_MOD bit and I2S_TX_SLAVE_MOD bit of register I2S_CONF_REG can configure I2S to slave receiving mode and slave transmitting mode, respectively.

I2S_TX_START bit of register I2S_CONF_REG is used to enable transmission. When I2S is in master transmitting mode and this bit is set, the module will keep driving the clock signal and data of left and right channels. If FIFO sends out all the buffered data and there are no new data to shift, the last batch of data will be looped on the data line. When this bit is reset, master will stop driving clock and data lines. When I2S is configured to slave transmitting mode and this bit is set, the module will wait for the master BCK clock to enable a transmit operation.

The I2S_RX_START bit of register I2S_CONF_REG is used to enable a receive operation. When I2S is in master receiving mode and this bit is set, the module will keep driving the clock signal and sampling the input data stream until this bit is reset. If I2S is configured to slave receiving mode and this bit is set, the receiving module will wait for the master BCK clock to enable a receiving operation.

### 22.4.7 I2S PDM

As is shown in Figure 22.1-1, ESP32 I2S0 allows for pulse density modulation (PDM), which enables fast conversion between pulse code modulation (PCM) and PDM signals.

The output clock of PDM is mapped to the I2S0*_WS_out signal. Its configuration is identical to I2S's BCK. Please refer to section 22.3, "The Clock of I2S Module", for further details. The bit width for both received and transmitted I2S PCM signals is 16 bits.

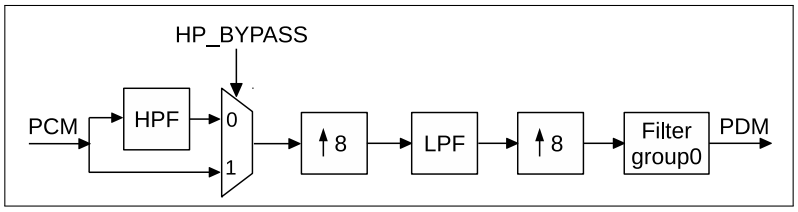



Figure 22.4-7. PDM Transmitting Module

The PDM transmitting module is used to convert PCM signals into PDM signals, as shown in Figure 22.4-7. HPF is a high-speed channel filter, and LPF is a low-speed channel filter. The PDM signal is derived from the PCM signal, after upsampling and filtering. Signal I2S_TX_PDM_HP_BYPASS of register I2S_PDM_CONF_REG can be set to bypass the HPF at the PCM input. Filter module group0 carries out the upsampling. If the frequency of the PDM signal is $f_{\text{pdm}}$ and the frequency of the PCM signal is $f_{\text{pcm}}$, the relation between $f_{\text{pdm}}$ and $f_{\text{pcm}}$ is given by:

$$f_{\text{pdm}} = 64 \times f_{\text{pcm}} \times \frac{I2S_TX_PDM_FP}{I2S_TX_PDM_FS}$$

The upsampling factor of 64 is the result of the two upsampling stages.

Table 22.4-5 lists the configuration rates of the I2S_TX_PDM_FP bit and the I2S_TX_PDM_FS bit of register I2S_PDM

_FREQ_CONF_REG, whose output PDM signal frequency remains 48×128 KHz at different PCM signal frequencies.

Table 22.4-5. Upsampling Rate Configuration

| $f_{pcm}$ (KHz) | I2S_TX_PDM_FP | I2S_TX_PDM_FS | $f_{pdm}$ (KHz) |
|---|---|---|---|
| 48 | 960 | 480 | 48×128 |
| 44.1 | 960 | 441 | |
| 32 | 960 | 320 | |
| 24 | 960 | 240 | |
| 16 | 960 | 160 | |
| 8 | 960 | 80 | |

The I2S_TX_PDM_SINC_OSR2 bit of I2S_PDM_CONF_REG is the upsampling rate of the Filter group0.

$$I2S_TX_PDM_SINC_OSR2 = \left\lfloor \frac{I2S_TX_PDM_FP}{I2S_TX_PDM_FS} \right\rfloor$$

As is shown in Figure 22.4-8, the I2S_TX_PDM_EN bit and the I2S_PCM2PDM_CONV_EN bit of register I2S_PDM_
CONF_REG should be set to 1 to use the PDM sending module. The I2S_TX_PDM_SIGMADELTA_IN_SHIFT bit, I2S_TX_PDM_SINC_IN_SHIFT bit, I2S_TX_PDM_LP_IN_SHIFT bit and I2S_TX_PDM_HP_IN_SHIFT bit of register I2S_PDM_CONF_REG are used to adjust the size of the input signal of each filter module.

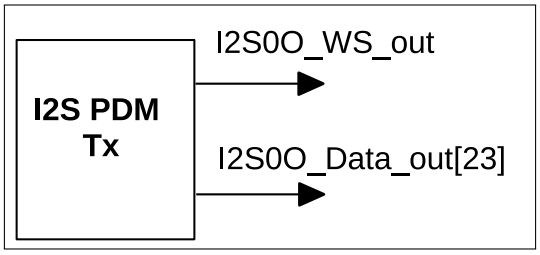



Figure 22.4-8. PDM Sends Signal

As is shown in Figure 22.4-9, the I2S_RX_PDM_EN bit and the I2S_PDM2PCM_CONV_EN bit of register I2S_PDM_
CONF_REG should be set to 1, in order to use the PDM receiving module. As is shown in Figure 22.4-10, the PDM receiving module will convert the received PDM signal into a 16-bit PCM signal. Filter group1 is used to downsample the PDM signal, and the I2S_RX_PDM_SINC_DSR_16_EN bit of register I2S_PDM_CONF_REG is used to adjust the corresponding down-sampling rate.

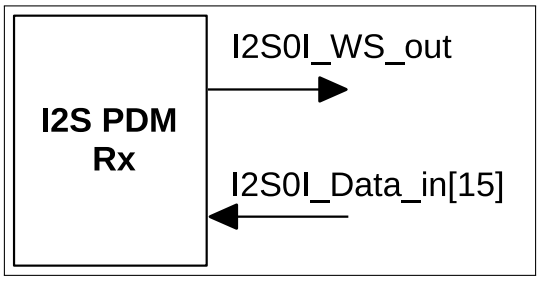



Figure 22.4-9. PDM Receives Signal

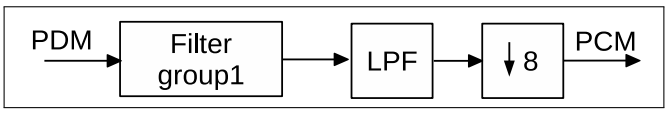



Figure 22.4-10. PDM Receive Module

Table 22.4-6 shows the configuration of the I2S_RX_PDM_SINC_DSR_16_EN bit whose PCM signal frequency remains 48 KHz at different PDM signal frequencies.

Table 22.4-6. Down-sampling Configuration

| PDM freq (KHz) | I2S_RX_PDM_SINC_DSR_16_EN | PCM freq (KHz) |
|---|---|---|
| $f_{pcm}$×128 | 1 | $f_{pcm}$ |
| $f_{pcm}$×64 | 0 | |

## 22.5 Camera-LCD Controller

There are three operational modes in the LCD mode of ESP32 I2S:

- LCD master transmitting mode
- Camera slave receiving mode
- ADC/DAC mode

The clock configuration of the LCD master transmitting mode is identical to I2S' clock configuration. In the LCD mode, the frequency of WS is half of $f_{BCK}$.

In the ADC/DAC mode, use PLL_F160M_CLK as the clock source.

### 22.5.1 LCD Master Transmitting Mode

As is shown in Figure 22.5-1, the WR signal of LCD connects to the WS signal of I2S. The LCD data bus width is 24 bits.

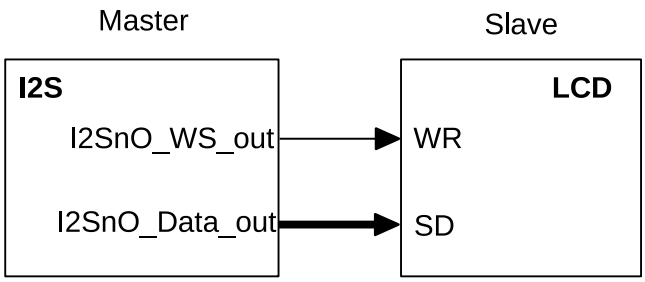



Figure 22.5-1. LCD Master Transmitting Mode

The I2S_LCD_EN bit of register I2S_CONF2_REG needs to be set and the I2S_TX_SLAVE_MOD bit of register I2S_CONF_REG needs to be cleared, in order to configure I2S to the LCD master transmitting mode. Meanwhile, data should be sent under the correct mode, according to the I2S_TX_CHAN_MOD[2:0] bit of register I2S_CONF_CHAN_REG and the I2S_TX_FIFO_MOD[2:0] bit of register I2S_FIFO_CONF_REG. The WS signal needs to be inverted when it is routed through the GPIO Matrix. For details, please refer to the chapter

about IO_MUX and the GPIO Matrix. The I2S_LCD_TX_SDX2_EN bit and the I2S_LCD_TX_WRX2_EN bit of register I2S_CONF2_REG should be set to the LCD master transmitting mode, so that both the data bus and WR signal work in the appropriate mode.

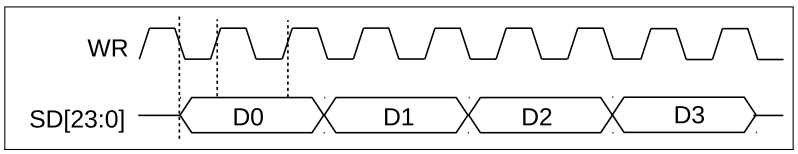



Figure 22.5-2. LCD Master Transmitting Data Frame, Form 1

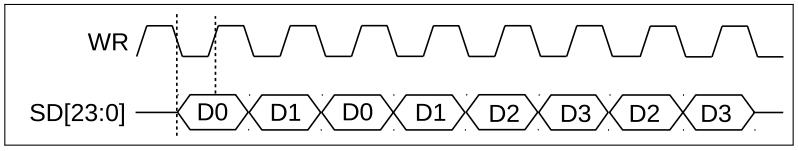



Figure 22.5-3. LCD Master Transmitting Data Frame, Form 2

As is shown in Figure 22.5-2 and Figure 22.5-3, the I2S_LCD_TX_WRX2_EN bit should be set to 1 and the I2S_LCD_TX_SDX2_EN bit should be set to 0 in the data frame, form 1. Both I2S_LCD_TX_SDX2_EN bit and I2S_LCD_TX_WRX2_EN bit are set to 1 in the data frame, form 2.

## 22.5.2 Camera Slave Receiving Mode

ESP32 I2S supports a camera slave mode for high-speed data transfer from external camera modules. As shown in Figure 22.5-4, in this mode, I2S is set to slave receiving mode. Besides the 16-channel data signal bus I2S*n*I_Data_in, there are other signals, such as I2S*n*_H_SYNC, I2S*n*_V_SYNC and I2S*n*_H_ENABLE.

The PCLK in the Camera module connects to I2S*n*I_WS_in in the I2S module, as Figure 22.5-4 shows.

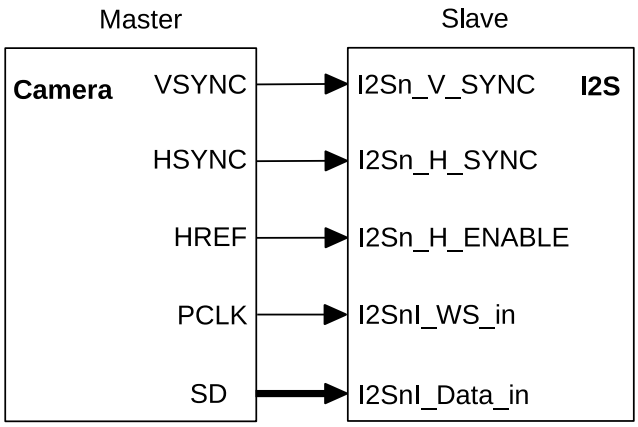



Figure 22.5-4. Camera Slave Receiving Mode

When I2S is in the camera slave receiving mode, and when I2S*n*_H_SYNC, I2S_V_SYNC and I2S_H_REF are held high, the master starts transmitting data, that is,

$$transmission_start = (I2Sn_H_SYNC == 1)\&\&(I2Sn_V_SYNC == 1)\&\&(I2Sn_H_ENABLE == 1)$$

Thus, during data transmission, these three signals should be kept at a high level. For example, if the I2S*n*_V_SYNC signal of a camera is at low level during data transmission, it will be inverted when routed to the

I2S module. ESP32 supports signal inversion through the GPIO matrix. For details, please refer to the chapter about IO_MUX and the GPIO Matrix.

In order to make I2S work in camera mode, the I2S_LCD_EN bit and the I2S_CAMERA_EN bit of register I2S_CONF2
_REG are set to 1, the I2S_RX_SLAVE_MOD bit of register I2S_CONF_REG is set to 1, the I2S_RX_MSB_RIGHT bit and the I2S_RX_RIGHT_FIRST bit of I2S_CONF_REG are set to 0. Thus, I2S works in the LCD slave receiving mode. At the same time, in order to use the correct mode to receive data, both the I2S_RX_CHAN_MOD[2:0] bit of register I2S_CONF_CHAN_REG and the I2S_RX_FIFO_MOD[2:0] bit of register I2S_FIFO_CONF_REG are set to 1.

### 22.5.3 ADC/DAC mode

In LCD mode, ESP32's ADC and DAC can receive data. When the I2S0 module connects to the on-chip ADC, the I2S0 module should be set to master receiving mode. Figure 22.5-5 shows the signal connection between the I2S0 module and the ADC.

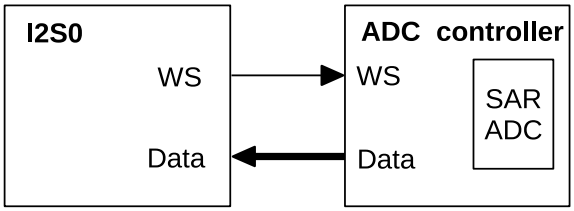



Figure 22.5-5. ADC Interface of I2S0

Firstly, the I2S_LCD_EN bit of register I2S_CONF2_REG is set to 1, and the I2S_RX_SLAVE_MOD bit of register I2S_CONF_REG is set to 0, so that the I2S0 module works in LCD master receiving mode, and the I2S0 module clock is configured such that the WS signal of I2S0 outputs an appropriate frequency. Then, the SYSCON_SAR
ADC_DATA_TO_I2S bit of register SYSCON_APB_SARADC_CTRL_REG is set to 1. Enable I2S to receive data after configuring the relevant registers of SARADC. For details, please refer to Chapter On-Chip Sensors and Analog Signal Processing.

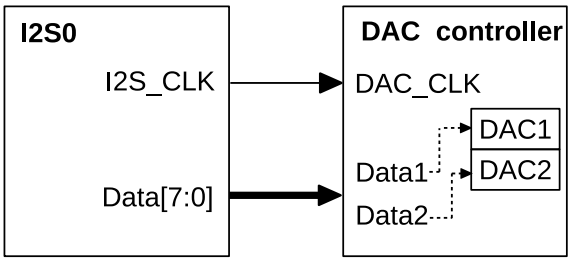



Figure 22.5-6. DAC Interface of I2S

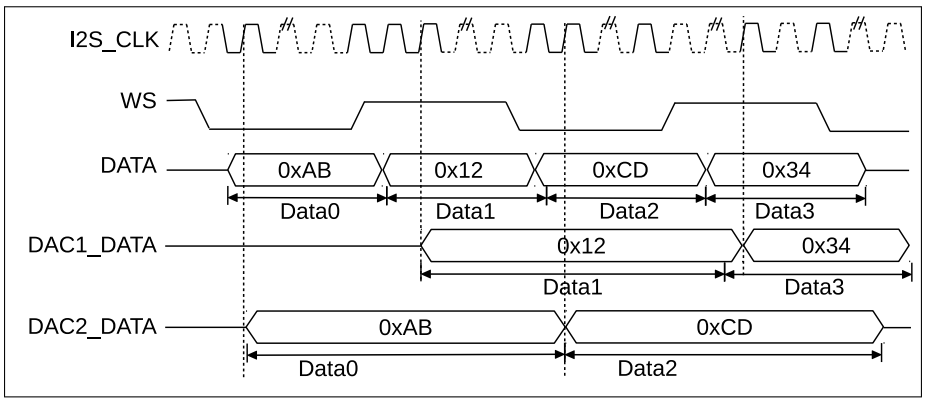



Figure 22.5-7. Data Input by I2S DAC Interface

The I2S0 module should be configured to master transmitting mode when it connects to the on-chip DAC. Figure 22.5-6 shows the signal connection between the I2S0 module and the DAC. The DAC's control module regards I2S_CLK as the clock in this configuration. As shown in Figure 22.5-7, when the data bus inputs data to the DAC's control module, the latter will input right-channel data to DAC1 module and left-channel data to DAC2 module. When using the I2S DMA module, 8 bits of data-to-be-transmitted are shifted to the left by 8 bits of data-to-be-received into the DMA double-byte type of buffer.

The I2S_LCD_EN bit of register I2S_CONF2_REG should be set to 1, while I2S_RX_SHORT_SYNC, I2S_TX_SHORT
_SYNC, I2S_CONF_REG , I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT should all be reset to 0. The I2S_TX_
SLAVE_MOD bit of register I2S_CONF_REG should be set to 0, as well, when using the DAC mode of I2S0. Select a suitable transmit mode according to the standards of transmitting a 16-bit digital data stream. Configure the I2S0 module clock to output a suitable frequency for the I2S_CLK and the WS of I2S. Enable I2S0 to send data after configuring the relevant DAC registers.

## 22.6 I2S Interrupts

### 22.6.1 FIFO Interrupts

- I2S_TX_HUNG_INT: Triggered when transmitting data is timed out.
- I2S_RX_HUNG_INT: Triggered when receiving data is timed out.
- I2S_TX_REMPTY_INT: Triggered when the transmit FIFO is empty.
- I2S_TX_WFULL_INT: Triggered when the transmit FIFO is full.
- I2S_RX_REMPTY_INT: Triggered when the receive FIFO is empty.
- I2S_RX_WFULL_INT: Triggered when the receive FIFO is full.
- I2S_TX_PUT_DATA_INT: Triggered when the transmit FIFO is almost empty.
- I2S_RX_TAKE_DATA_INT: Triggered when the receive FIFO is almost full.

### 22.6.2 DMA Interrupts

- I2S_OUT_TOTAL_EOF_INT: Triggered when all transmitting linked lists are used up.
- I2S_IN_DSCR_EMPTY_INT: Triggered when there are no valid receiving linked lists left.
- I2S_OUT_DSCR_ERR_INT: Triggered when invalid txlink descriptors are encountered.
- I2S_IN_DSCR_ERR_INT: Triggered when invalid rxlink descriptors are encountered.
- I2S_OUT_EOF_INT: Triggered when txlink has finished sending a packet.
- I2S_OUT_DONE_INT: Triggered when all transmitted and buffered data have been read.
- I2S_IN_SUC_EOF_INT: Triggered when all data have been received.
- I2S_IN_DONE_INT: Triggered when the current rxlink descriptor is handled.

## 22.7 Register Summary

The addresses in this section are relative to the I2S base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | I2S0 | I2S1 | Acc |
|---|---|---|---|---|
| **I2S FIFO registers** | | | | |
| I2S_FIFO_WR_REG | Writes the data sent by I2S into FIFO | 0x3FF4F000 | 0x3FF6D000 | WO |
| I2S_FIFO_RD_REG | Stores the data that I2S receives from FIFO | 0x3FF4F004 | 0x3FF6D004 | RO |
| **Configuration registers** | | | | |
| I2S_CONF_REG | Configuration and start/stop bits | 0x3FF4F008 | 0x3FF6D008 | R/W |
| I2S_CONF1_REG | PCM configuration register | 0x3FF4F0A0 | 0x3FF6D0A0 | R/W |
| I2S_CONF2_REG | ADC/LCD/camera configuration register | 0x3FF4F0A8 | 0x3FF6D0A8 | R/W |
| I2S_TIMING_REG | Signal delay and timing parameters | 0x3FF4F01C | 0x3FF6D01C | R/W |
| I2S_FIFO_CONF_REG | FIFO configuration | 0x3FF4F020 | 0x3FF6D020 | R/W |
| I2S_CONF_SINGLE_DATA_REG | Static channel output value | 0x3FF4F028 | 0x3FF6D028 | R/W |
| I2S_CONF_CHAN_REG | Channel configuration | 0x3FF4F02C | 0x3FF6D02C | R/W |
| I2S_LC_HUNG_CONF_REG | Timeout detection configuration | 0x3FF4F074 | 0x3FF6D074 | R/W |
| I2S_CLKM_CONF_REG | Bitclock configuration | 0x3FF4F0AC | 0x3FF6D0AC | R/W |
| I2S_SAMPLE_RATE_CONF_REG | Sample rate configuration | 0x3FF4F0B0 | 0x3FF6D0B0 | R/W |
| I2S_PD_CONF_REG | Power-down register | 0x3FF4F0A4 | 0x3FF6D0A4 | R/W |
| I2S_STATE_REG | I2S status register | 0x3FF4F0BC | 0x3FF6D0BC | RO |
| **DMA registers** | | | | |
| I2S_LC_CONF_REG | DMA configuration register | 0x3FF4F060 | 0x3FF6D060 | R/W |
| I2S_RXEOF_NUM_REG | Receive data count | 0x3FF4F024 | 0x3FF6D024 | R/W |

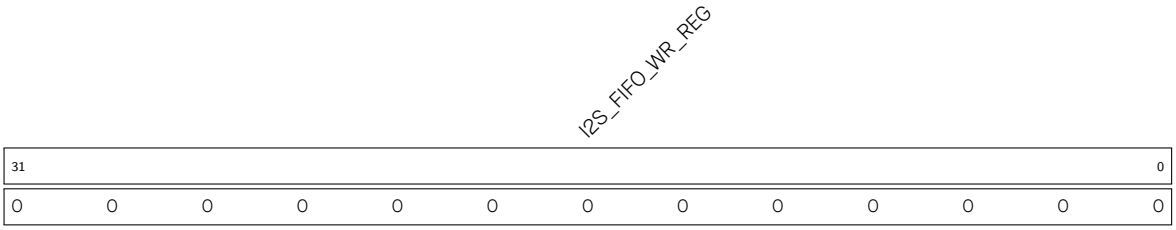
| | | | | |
|---|---|---|---|---|
| I2S_OUT_LINK_REG | DMA transmit linked list configuration and address | 0x3FF4F030 | 0x3FF6D030 | R/W |
| I2S_IN_LINK_REG | DMA receive linked list configuration and address | 0x3FF4F034 | 0x3FF6D034 | R/W |
| I2S_OUT_EOF_DES_ADDR_REG | The address of transmit link descriptor producing EOF | 0x3FF4F038 | 0x3FF6D038 | RO |
| I2S_IN_EOF_DES_ADDR_REG | The address of receive link descriptor producing EOF | 0x3FF4F03C | 0x3FF6D03C | RO |
| I2S_OUT_EOF_BFR_DES_ADDR_REG | The address of transmit buffer producing EOF | 0x3FF4F040 | 0x3FF6D040 | RO |
| I2S_INLINK_DSCR_REG | The address of current inlink descriptor | 0x3FF4F048 | 0x3FF6D048 | RO |
| I2S_INLINK_DSCR_BF0_REG | The address of next inlink descriptor | 0x3FF4F04C | 0x3FF6D04C | RO |
| I2S_INLINK_DSCR_BF1_REG | The address of next inlink data buffer | 0x3FF4F050 | 0x3FF6D050 | RO |
| I2S_OUTLINK_DSCR_REG | The address of current outlink descriptor | 0x3FF4F054 | 0x3FF6D054 | RO |
| I2S_OUTLINK_DSCR_BF0_REG | The address of next outlink descriptor | 0x3FF4F058 | 0x3FF6D058 | RO |
| I2S_OUTLINK_DSCR_BF1_REG | The address of next outlink data buffer | 0x3FF4F05C | 0x3FF6D05C | RO |
| I2S_LC_STATE0_REG | DMA receive status | 0x3FF4F06C | 0x3FF6D06C | RO |
| I2S_LC_STATE1_REG | DMA transmit status | 0x3FF4F070 | 0x3FF6D070 | RO |
| **Pulse density (DE) modulation registers** | | | | |
| I2S_PDM_CONF_REG | PDM configuration | 0x3FF4F0B4 | 0x3FF6D0B4 | R/W |
| I2S_PDM_FREQ_CONF_REG | PDM frequencies | 0x3FF4F0B8 | 0x3FF6D0B8 | R/W |
| **Interrupt registers** | | | | |
| I2S_INT_RAW_REG | Raw interrupt status | 0x3FF4F00C | 0x3FF6D00C | RO |
| I2S_INT_ST_REG | Masked interrupt status | 0x3FF4F010 | 0x3FF6D010 | RO |
| I2S_INT_ENA_REG | Interrupt enable bits | 0x3FF4F014 | 0x3FF6D014 | R/W |
| I2S_INT_CLR_REG | Interrupt clear bits | 0x3FF4F018 | 0x3FF6D018 | WO |

## 22.8 Registers

The addresses in this section are relative to the I2S base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 22.7 *Register Summary*.

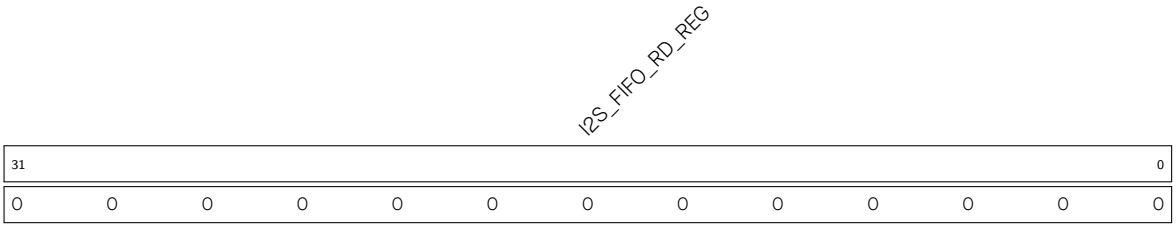
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 22.1. I2S_FIFO_WR_REG (0x0000)**




**I2S_FIFO_WR_REG** Writes the data sent by I2S into FIFO. (WO)

**Register 22.2. I2S_FIFO_RD_REG (0x0004)**




**I2S_FIFO_RD_REG** Stores the data that I2S receives from FIFO. (RO)

**Register 22.3. I2S_CONF_REG (0x0008)**

| 31–19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | I2S_SIG_LOOPBACK | I2S_RX_MSB_RIGHT | I2S_TX_MSB_RIGHT | I2S_RX_MONO | I2S_TX_MONO | I2S_RX_SHORT_SYNC | I2S_TX_SHORT_SYNC | I2S_RX_MSB_SHIFT | I2S_TX_MSB_SHIFT | I2S_RX_RIGHT_FIRST | I2S_TX_RIGHT_FIRST | I2S_RX_SLAVE_MOD | I2S_TX_SLAVE_MOD | I2S_RX_START | I2S_TX_START | I2S_RX_FIFO_RESET | I2S_TX_FIFO_RESET | I2S_RX_RESET | I2S_TX_RESET | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**I2S_SIG_LOOPBACK** Enable signal loopback mode, with transmitter module and receiver module sharing the same WS and BCK signals. (R/W)

**I2S_RX_MSB_RIGHT** Set this to place right-channel data at the MSB in the receive FIFO. (R/W)

**I2S_TX_MSB_RIGHT** Set this bit to place right-channel data at the MSB in the transmit FIFO. (R/W)

**I2S_RX_MONO** Set this bit to enable receiver's mono mode in PCM standard mode. (R/W)

**I2S_TX_MONO** Set this bit to enable transmitter's mono mode in PCM standard mode. (R/W)

**I2S_RX_SHORT_SYNC** Set this bit to enable receiver in PCM standard mode. (R/W)

**I2S_TX_SHORT_SYNC** Set this bit to enable transmitter in PCM standard mode. (R/W)

**I2S_RX_MSB_SHIFT** Set this bit to enable receiver in Philips standard mode. (R/W)

**I2S_TX_MSB_SHIFT** Set this bit to enable transmitter in Philips standard mode. (R/W)

**I2S_RX_RIGHT_FIRST** Set this bit to receive right-channel data first. (R/W)

**I2S_TX_RIGHT_FIRST** Set this bit to transmit right-channel data first. (R/W)

**I2S_RX_SLAVE_MOD** Set this bit to enable slave receiver mode. (R/W)

**I2S_TX_SLAVE_MOD** Set this bit to enable slave transmitter mode. (R/W)

**I2S_RX_START** Set this bit to start receiving data. (R/W)

**I2S_TX_START** Set this bit to start transmitting data. (R/W)

**I2S_RX_FIFO_RESET** Set this bit to reset the receive FIFO. (R/W)

**I2S_TX_FIFO_RESET** Set this bit to reset the transmit FIFO. (R/W)

**I2S_RX_RESET** Set this bit to reset the receiver. (R/W)

**I2S_TX_RESET** Set this bit to reset the transmitter. (R/W)

**Register 22.4. I2S_INT_RAW_REG (0x000c)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | I2S_OUT_TOTAL_EOF_INT_RAW | I2S_IN_DSCR_EMPTY_INT_RAW | I2S_OUT_DSCR_ERR_INT_RAW | I2S_IN_DSCR_ERR_INT_RAW | I2S_OUT_EOF_INT_RAW | I2S_OUT_DONE_INT_RAW | (reserved) | I2S_IN_SUC_EOF_INT_RAW | I2S_IN_DONE_INT_RAW | I2S_TX_HUNG_INT_RAW | I2S_RX_HUNG_INT_RAW | I2S_TX_REMPTY_INT_RAW | I2S_TX_WFULL_INT_RAW | I2S_RX_REMPTY_INT_RAW | I2S_RX_WFULL_INT_RAW | I2S_TX_PUT_DATA_INT_RAW | I2S_RX_TAKE_DATA_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**I2S_OUT_TOTAL_EOF_INT_RAW** The raw interrupt status bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

**I2S_IN_DSCR_EMPTY_INT_RAW** The raw interrupt status bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

**I2S_OUT_DSCR_ERR_INT_RAW** The raw interrupt status bit for the I2S_OUT_DSCR_ERR_INT interrupt. (RO)

**I2S_IN_DSCR_ERR_INT_RAW** The raw interrupt status bit for the I2S_IN_DSCR_ERR_INT interrupt. (RO)

**I2S_OUT_EOF_INT_RAW** The raw interrupt status bit for the I2S_OUT_EOF_INT interrupt. (RO)

**I2S_OUT_DONE_INT_RAW** The raw interrupt status bit for the I2S_OUT_DONE_INT interrupt. (RO)

**I2S_IN_SUC_EOF_INT_RAW** The raw interrupt status bit for the I2S_IN_SUC_EOF_INT interrupt. (RO)

**I2S_IN_DONE_INT_RAW** The raw interrupt status bit for the I2S_IN_DONE_INT interrupt. (RO)

**I2S_TX_HUNG_INT_RAW** The raw interrupt status bit for the I2S_TX_HUNG_INT interrupt. (RO)

**I2S_RX_HUNG_INT_RAW** The raw interrupt status bit for the I2S_RX_HUNG_INT interrupt. (RO)

**I2S_TX_REMPTY_INT_RAW** The raw interrupt status bit for the I2S_TX_REMPTY_INT interrupt. (RO)

**I2S_TX_WFULL_INT_RAW** The raw interrupt status bit for the I2S_TX_WFULL_INT interrupt. (RO)

**I2S_RX_REMPTY_INT_RAW** The raw interrupt status bit for the I2S_RX_REMPTY_INT interrupt. (RO)

**I2S_RX_WFULL_INT_RAW** The raw interrupt status bit for the I2S_RX_WFULL_INT interrupt. (RO)

**I2S_TX_PUT_DATA_INT_RAW** The raw interrupt status bit for the I2S_TX_PUT_DATA_INT interrupt. (RO)

**I2S_RX_TAKE_DATA_INT_RAW** The raw interrupt status bit for the I2S_RX_TAKE_DATA_INT interrupt. (RO)

**Register 22.5. I2S_INT_ST_REG (0x0010)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | I2S_OUT_TOTAL_EOF_INT_ST | I2S_IN_DSCR_EMPTY_INT_ST | I2S_OUT_DSCR_ERR_INT_ST | I2S_IN_DSCR_ERR_INT_ST | I2S_OUT_EOF_INT_ST | I2S_OUT_DONE_INT_ST | (reserved) | I2S_IN_SUC_EOF_INT_ST | I2S_TX_DONE_INT_ST | I2S_TX_HUNG_INT_ST | I2S_RX_HUNG_INT_ST | I2S_TX_REMPTY_INT_ST | I2S_TX_WFULL_INT_ST | I2S_RX_REMPTY_INT_ST | I2S_RX_WFULL_INT_ST | I2S_TX_PUT_DATA_INT_ST | I2S_RX_TAKE_DATA_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**I2S_OUT_TOTAL_EOF_INT_ST** The masked interrupt status bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

**I2S_IN_DSCR_EMPTY_INT_ST** The masked interrupt status bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

**I2S_OUT_DSCR_ERR_INT_ST** The masked interrupt status bit for the I2S_OUT_DSCR_ERR_INT interrupt. (RO)

**I2S_IN_DSCR_ERR_INT_ST** The masked interrupt status bit for the I2S_IN_DSCR_ERR_INT interrupt. (RO)

**I2S_OUT_EOF_INT_ST** The masked interrupt status bit for the I2S_OUT_EOF_INT interrupt. (RO)

**I2S_OUT_DONE_INT_ST** The masked interrupt status bit for the I2S_OUT_DONE_INT interrupt. (RO)

**I2S_IN_SUC_EOF_INT_ST** The masked interrupt status bit for the I2S_IN_SUC_EOF_INT interrupt. (RO)

**I2S_IN_DONE_INT_ST** The masked interrupt status bit for the I2S_IN_DONE_INT interrupt. (RO)

**I2S_TX_HUNG_INT_ST** The masked interrupt status bit for the I2S_TX_HUNG_INT interrupt. (RO)

**I2S_RX_HUNG_INT_ST** The masked interrupt status bit for the I2S_RX_HUNG_INT interrupt. (RO)

**I2S_TX_REMPTY_INT_ST** The masked interrupt status bit for the I2S_TX_REMPTY_INT interrupt. (RO)

**I2S_TX_WFULL_INT_ST** The masked interrupt status bit for the I2S_TX_WFULL_INT interrupt. (RO)

**I2S_RX_REMPTY_INT_ST** The masked interrupt status bit for the I2S_RX_REMPTY_INT interrupt. (RO)

**I2S_RX_WFULL_INT_ST** The masked interrupt status bit for the I2S_RX_WFULL_INT interrupt. (RO)

**I2S_TX_PUT_DATA_INT_ST** The masked interrupt status bit for the I2S_TX_PUT_DATA_INT interrupt. (RO)

**I2S_RX_TAKE_DATA_INT_ST** The masked interrupt status bit for the I2S_RX_TAKE_DATA_INT interrupt. (RO)

**Register 22.6. I2S_INT_ENA_REG (0x0014)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | I2S_OUT_TOTAL_EOF_INT_ENA | I2S_IN_DSCR_EMPTY_INT_ENA | I2S_OUT_DSCR_ERR_INT_ENA | I2S_IN_DSCR_ERR_INT_ENA | I2S_OUT_EOF_INT_ENA | I2S_OUT_DONE_INT_ENA | (reserved) | I2S_IN_SUC_EOF_INT_ENA | I2S_IN_DONE_INT_ENA | I2S_TX_HUNG_INT_ENA | I2S_RX_HUNG_INT_ENA | I2S_TX_REMPTY_INT_ENA | I2S_TX_WFULL_INT_ENA | I2S_RX_REMPTY_INT_ENA | I2S_RX_WFULL_INT_ENA | I2S_TX_PUT_DATA_INT_ENA | I2S_RX_TAKE_DATA_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**I2S_OUT_TOTAL_EOF_INT_ENA** The interrupt enable bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (R/W)

**I2S_IN_DSCR_EMPTY_INT_ENA** The interrupt enable bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (R/W)

**I2S_OUT_DSCR_ERR_INT_ENA** The interrupt enable bit for the I2S_OUT_DSCR_ERR_INT interrupt. (R/W)

**I2S_IN_DSCR_ERR_INT_ENA** The interrupt enable bit for the I2S_IN_DSCR_ERR_INT interrupt. (R/W)

**I2S_OUT_EOF_INT_ENA** The interrupt enable bit for the I2S_OUT_EOF_INT interrupt. (R/W)

**I2S_OUT_DONE_INT_ENA** The interrupt enable bit for the I2S_OUT_DONE_INT interrupt. (R/W)

**I2S_IN_SUC_EOF_INT_ENA** The interrupt enable bit for the I2S_IN_SUC_EOF_INT interrupt. (R/W)

**I2S_IN_DONE_INT_ENA** The interrupt enable bit for the I2S_IN_DONE_INT interrupt. (R/W)

**I2S_TX_HUNG_INT_ENA** The interrupt enable bit for the I2S_TX_HUNG_INT interrupt. (R/W)

**I2S_RX_HUNG_INT_ENA** The interrupt enable bit for the I2S_RX_HUNG_INT interrupt. (R/W)

**I2S_TX_REMPTY_INT_ENA** The interrupt enable bit for the I2S_TX_REMPTY_INT interrupt. (R/W)

**I2S_TX_WFULL_INT_ENA** The interrupt enable bit for the I2S_TX_WFULL_INT interrupt. (R/W)

**I2S_RX_REMPTY_INT_ENA** The interrupt enable bit for the I2S_RX_REMPTY_INT interrupt. (R/W)

**I2S_RX_WFULL_INT_ENA** The interrupt enable bit for the I2S_RX_WFULL_INT interrupt. (R/W)

**I2S_TX_PUT_DATA_INT_ENA** The interrupt enable bit for the I2S_TX_PUT_DATA_INT interrupt. (R/W)

**I2S_RX_TAKE_DATA_INT_ENA** The interrupt enable bit for the I2S_RX_TAKE_DATA_INT interrupt. (R/W)

**Register 22.7. I2S_INT_CLR_REG (0x0018)**

| 31–17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | I2S_OUT_TOTAL_EOF_INT_CLR | I2S_IN_DSCR_EMPTY_INT_CLR | I2S_OUT_DSCR_ERR_INT_CLR | I2S_IN_DSCR_ERR_INT_CLR | I2S_OUT_EOF_INT_CLR | I2S_OUT_DONE_INT_CLR | (reserved) | I2S_IN_SUC_EOF_INT_CLR | I2S_IN_DONE_INT_CLR | I2S_TX_HUNG_INT_CLR | I2S_RX_HUNG_INT_CLR | I2S_TX_REMPTY_INT_CLR | I2S_TX_WFULL_INT_CLR | I2S_RX_REMPTY_INT_CLR | I2S_RX_WFULL_INT_CLR | I2S_TX_PUT_DATA_INT_CLR | I2S_RX_TAKE_DATA_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**I2S_OUT_TOTAL_EOF_INT_CLR** Set this bit to clear the I2S_OUT_TOTAL_EOF_INT interrupt. (WO)

**I2S_IN_DSCR_EMPTY_INT_CLR** Set this bit to clear the I2S_IN_DSCR_EMPTY_INT interrupt. (WO)

**I2S_OUT_DSCR_ERR_INT_CLR** Set this bit to clear the I2S_OUT_DSCR_ERR_INT interrupt. (WO)

**I2S_IN_DSCR_ERR_INT_CLR** Set this bit to clear the I2S_IN_DSCR_ERR_INT interrupt. (WO)

**I2S_OUT_EOF_INT_CLR** Set this bit to clear the I2S_OUT_EOF_INT interrupt. (WO)

**I2S_OUT_DONE_INT_CLR** Set this bit to clear the I2S_OUT_DONE_INT interrupt. (WO)

**I2S_IN_SUC_EOF_INT_CLR** Set this bit to clear the I2S_IN_SUC_EOF_INT interrupt. (WO)

**I2S_IN_DONE_INT_CLR** Set this bit to clear the I2S_IN_DONE_INT interrupt. (WO)

**I2S_TX_HUNG_INT_CLR** Set this bit to clear the I2S_TX_HUNG_INT interrupt. (WO)

**I2S_RX_HUNG_INT_CLR** Set this bit to clear the I2S_RX_HUNG_INT interrupt. (WO)

**I2S_TX_REMPTY_INT_CLR** Set this bit to clear the I2S_TX_REMPTY_INT interrupt. (WO)

**I2S_TX_WFULL_INT_CLR** Set this bit to clear the I2S_TX_WFULL_INT interrupt. (WO)

**I2S_RX_REMPTY_INT_CLR** Set this bit to clear the I2S_RX_REMPTY_INT interrupt. (WO)

**I2S_RX_WFULL_INT_CLR** Set this bit to clear the I2S_RX_WFULL_INT interrupt. (WO)

**I2S_TX_PUT_DATA_INT_CLR** Set this bit to clear the I2S_TX_PUT_DATA_INT interrupt. (WO)

**I2S_RX_TAKE_DATA_INT_CLR** Set this bit to clear the I2S_RX_TAKE_DATA_INT interrupt. (WO)

**Register 22.8. I2S_TIMING_REG (0x001c)**

| (reserved) | I2S_TX_BCK_IN_INV | I2S_DATA_ENABLE_DELAY | I2S_RX_DSYNC_SW | I2S_TX_DSYNC_SW | I2S_RX_BCK_OUT_DELAY | I2S_RX_WS_OUT_DELAY | I2S_TX_SD_OUT_DELAY | I2S_TX_WS_OUT_DELAY | I2S_TX_BCK_OUT_DELAY | I2S_RX_SD_IN_DELAY | I2S_RX_WS_IN_DELAY | I2S_RX_BCK_IN_DELAY | I2S_TX_WS_IN_DELAY | I2S_TX_BCK_IN_DELAY | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 |

**I2S_TX_BCK_IN_INV** Set this bit to invert the BCK signal into the slave transmitter. (R/W)

**I2S_DATA_ENABLE_DELAY** Number of delay cycles for data valid flag. (R/W)

**I2S_RX_DSYNC_SW** Set this bit to synchronize signals into the receiver in double sync method. (R/W)

**I2S_TX_DSYNC_SW** Set this bit to synchronize signals into the transmitter in double sync method. (R/W)

**I2S_RX_BCK_OUT_DELAY** Number of delay cycles for BCK signal out of the receiver. (R/W)

**I2S_RX_WS_OUT_DELAY** Number of delay cycles for WS signal out of the receiver. (R/W)

**I2S_TX_SD_OUT_DELAY** Number of delay cycles for SD signal out of the transmitter. (R/W)

**I2S_TX_WS_OUT_DELAY** Number of delay cycles for WS signal out of the transmitter. (R/W)

**I2S_TX_BCK_OUT_DELAY** Number of delay cycles for BCK signal out of the transmitter. (R/W)

**I2S_RX_SD_IN_DELAY** Number of delay cycles for SD signal into the receiver. (R/W)

**I2S_RX_WS_IN_DELAY** Number of delay cycles for WS signal into the receiver. (R/W)

**I2S_RX_BCK_IN_DELAY** Number of delay cycles for BCK signal into the receiver. (R/W)

**I2S_TX_WS_IN_DELAY** Number of delay cycles for WS signal into the transmitter. (R/W)

**I2S_TX_BCK_IN_DELAY** Number of delay cycles for BCK signal into the transmitter. (R/W)

### Register 22.9. I2S_FIFO_CONF_REG (0x0020)

| (reserved) | I2S_RX_FIFO_MOD_FORCE_EN | I2S_TX_FIFO_MOD_FORCE_EN | I2S_RX_FIFO_MOD | I2S_TX_FIFO_MOD | I2S_DSCR_EN | I2S_TX_DATA_NUM | I2S_RX_DATA_NUM | |
|---|---|---|---|---|---|---|---|---|
| 31 … 21 | 20 | 19 | 18 … 16 | 15 … 13 | 12 | 11 … 6 | 5 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 0 0 | 0 0 0 | 1 | 32 | 32 | Reset |

**I2S_RX_FIFO_MOD_FORCE_EN** The bit should always be set to 1. (R/W)

**I2S_TX_FIFO_MOD_FORCE_EN** The bit should always be set to 1. (R/W)

**I2S_RX_FIFO_MOD** Receive FIFO mode configuration bit. (R/W)

**I2S_TX_FIFO_MOD** Transmit FIFO mode configuration bit. (R/W)

**I2S_DSCR_EN** Set this bit to enable I2S DMA mode. (R/W)

**I2S_TX_DATA_NUM** Threshold of data length in the transmit FIFO. (R/W)

**I2S_RX_DATA_NUM** Threshold of data length in the receive FIFO. (R/W)

### Register 22.10. I2S_RXEOF_NUM_REG (0x0024)

| 31 … 0 | |
|---|---|
| 64 | Reset |

**I2S_RXEOF_NUM_REG** The length of the data to be received. It will trigger I2S_IN_SUC_EOF_INT. (R/W)

### Register 22.11. I2S_CONF_SINGLE_DATA_REG (0x0028)

| 31 … 0 | |
|---|---|
| 0 | Reset |

**I2S_CONF_SINGLE_DATA_REG** The right channel or the left channel outputs constant values stored in this register according to TX_CHAN_MOD and I2S_TX_MSB_RIGHT. (R/W)

### Register 22.12. I2S_CONF_CHAN_REG (0x002c)

| (reserved) | I2S_RX_CHAN_MOD | I2S_TX_CHAN_MOD | |
|---|---|---|---|
| 31 … 5 | 4 … 3 | 2 … 0 | |
| 0 0 0 0 0 0 0 0 0 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 |

**I2S_RX_CHAN_MOD** I2S receiver channel mode configuration bits. Please refer to Section 22.4.5 for further details. (R/W)

**I2S_TX_CHAN_MOD** I2S transmitter channel mode configuration bits. Please refer to Section 22.4.4 for further details. (R/W)

### Register 22.13. I2S_OUT_LINK_REG (0x0030)

| (reserved) | I2S_OUTLINK_RESTART | I2S_OUTLINK_START | I2S_OUTLINK_STOP | (reserved) | I2S_OUTLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**I2S_OUTLINK_RESTART** Set this bit to restart outlink descriptor. (R/W)

**I2S_OUTLINK_START** Set this bit to start outlink descriptor. (R/W)

**I2S_OUTLINK_STOP** Set this bit to stop outlink descriptor. (R/W)

**I2S_OUTLINK_ADDR** The address of first outlink descriptor. (R/W)

### Register 22.14. I2S_IN_LINK_REG (0x0034)

| (reserved) | I2S_INLINK_RESTART | I2S_INLINK_START | I2S_INLINK_STOP | (reserved) | I2S_INLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**I2S_INLINK_RESTART** Set this bit to restart inlink descriptor. (R/W)

**I2S_INLINK_START** Set this bit to start inlink descriptor. (R/W)

**I2S_INLINK_STOP** Set this bit to stop inlink descriptor. (R/W)

**I2S_INLINK_ADDR** The address of first inlink descriptor. (R/W)

### Register 22.15. I2S_OUT_EOF_DES_ADDR_REG (0x0038)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**I2S_OUT_EOF_DES_ADDR_REG** The address of outlink descriptor that produces EOF. (RO)

### Register 22.16. I2S_IN_EOF_DES_ADDR_REG (0x003c)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**I2S_IN_EOF_DES_ADDR_REG** The address of inlink descriptor that produces EOF. (RO)

### Register 22.17. I2S_OUT_EOF_BFR_DES_ADDR_REG (0x0040)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**I2S_OUT_EOF_BFR_DES_ADDR_REG** The address of the buffer corresponding to the outlink descriptor that produces EOF. (RO)

### Register 22.18. I2S_INLINK_DSCR_REG (0x0048)

| 31 … 0 | |
|---|---|
| 0 0 0 0 0 0 0 0 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 |

**I2S_INLINK_DSCR_REG** The address of current inlink descriptor. (RO)

### Register 22.19. I2S_INLINK_DSCR_BF0_REG (0x004c)

| 31 … 0 | |
|---|---|
| 0 0 0 0 0 0 0 0 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 |

**I2S_INLINK_DSCR_BF0_REG** The address of next inlink descriptor. (RO)

### Register 22.20. I2S_INLINK_DSCR_BF1_REG (0x0050)

| 31 … 0 | |
|---|---|
| 0 0 0 0 0 0 0 0 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 |

**I2S_INLINK_DSCR_BF1_REG** The address of next inlink data buffer. (RO)

### Register 22.21. I2S_OUTLINK_DSCR_REG (0x0054)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**I2S_OUTLINK_DSCR_REG** The address of current outlink descriptor. (RO)

### Register 22.22. I2S_OUTLINK_DSCR_BF0_REG (0x0058)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**I2S_OUTLINK_DSCR_BF0_REG** The address of next outlink descriptor. (RO)

### Register 22.23. I2S_OUTLINK_DSCR_BF1_REG (0x005c)

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**I2S_OUTLINK_DSCR_BF1_REG** The address of next outlink data buffer. (RO)

**Register 22.24. I2S_LC_CONF_REG (0x0060)**

| 31–13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | I2S_CHECK_OWNER | I2S_OUT_DATA_BURST_EN | I2S_INDSCR_BURST_EN | I2S_OUTDSCR_BURST_EN | I2S_OUT_EOF_MODE | (reserved) | I2S_OUT_AUTO_WRBACK | I2S_OUT_LOOP_TEST | I2S_IN_LOOP_TEST | I2S_AHBM_RST | I2S_AHBM_FIFO_RST | I2S_OUT_RST | I2S_IN_RST | |
| 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 | 0 | 0 | 0 | 0 | 0 | Reset |

**I2S_CHECK_OWNER** Set this bit to check the owner bit by hardware. (R/W)

**I2S_OUT_DATA_BURST_EN** Transmitter data transfer mode configuration bit. (R/W)

1: Transmit data in burst mode;

0: Transmit data in byte mode.

**I2S_INDSCR_BURST_EN** DMA inlink descriptor transfer mode configuration bit. (R/W)

1: Transfer inlink descriptor in burst mode;

0: Transfer inlink descriptor in byte mode.

**I2S_OUTDSCR_BURST_EN** DMA outlink descriptor transfer mode configuration bit. (R/W)

1: Transfer outlink descriptor in burst mode;

0: Transfer outlink descriptor in byte mode.

**I2S_OUT_EOF_MODE** DMA I2S_OUT_EOF_INT generation mode. (R/W)

1: When DMA has popped all data from the FIFO;

0: When AHB has pushed all data to the FIFO.

**I2S_OUT_AUTO_WRBACK** Set this bit to enable automatic outlink-writeback when all the data in tx buffer has been transmitted. (R/W)

**I2S_OUT_LOOP_TEST** Set this bit to loop test outlink. (R/W)

**I2S_IN_LOOP_TEST** Set this bit to loop test inlink. (R/W)

**I2S_AHBM_RST** Set this bit to reset AHB interface of DMA. (R/W)

**I2S_AHBM_FIFO_RST** Set this bit to reset AHB interface cmdFIFO of DMA. (R/W)

**I2S_OUT_RST** Set this bit to reset out DMA FSM. (R/W)

**I2S_IN_RST** Set this bit to reset in DMA FSM. (R/W)

**Register 22.25. I2S_LC_STATE0_REG (0x006c)**

| 31–0 | |
|---|---|
| 0x000000000 | Reset |

**I2S_LC_STATE0_REG** Receiver DMA channel status register. (RO)

### Register 22.26. I2S_LC_STATE1_REG (0x0070)

| 31 … 0 | Reset |
|---|---|
| 0x000000000 | |

**I2S_LC_STATE1_REG** Transmitter DMA channel status register. (RO)

### Register 22.27. I2S_LC_HUNG_CONF_REG (0x0074)

| (reserved) | I2S_LC_FIFO_TIMEOUT_ENA | I2S_LC_FIFO_TIMEOUT_SHIFT | I2S_LC_FIFO_TIMEOUT | |
|---|---|---|---|---|
| 31 … 12 | 11 | 10 … 8 | 7 … 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 | 0x010 | Reset |

**I2S_LC_FIFO_TIMEOUT_ENA** The enable bit for FIFO timeout. (R/W)

**I2S_LC_FIFO_TIMEOUT_SHIFT** The bits are used to set the tick counter threshold. The tick counter is reset when the counter value >= $88000/2^{i2s_lc_fifo_timeout_shift}$. (R/W)

**I2S_LC_FIFO_TIMEOUT** When the value of FIFO hung counter is equal to this bit value, sending data-timeout interrupt or receiving data-timeout interrupt will be triggered. (R/W)

Register 22.28. I2S_CONF1_REG (0x00a0)

| 31 – 9 | 8 | 7 | 6 – 4 | 3 | 2 – 0 | |
|---|---|---|---|---|---|---|
| (reserved) | I2S_TX_STOP_EN | I2S_RX_PCM_BYPASS | I2S_RX_PCM_CONF | I2S_TX_PCM_BYPASS | I2S_TX_PCM_CONF | |
| 0 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 | 0x0 | 1 | 0x1 | Reset |

**I2S_TX_STOP_EN** Set this bit and the transmitter will stop transmitting BCK signal and WS signal when tx FIFO is empty. (R/W)

**I2S_RX_PCM_BYPASS** Set this bit to bypass the Compress/Decompress module for the received data. (R/W)

**I2S_RX_PCM_CONF** Compress/Decompress module configuration bit. (R/W)

0: Decompress received data;

1: Compress received data.

**I2S_TX_PCM_BYPASS** Set this bit to bypass the Compress/Decompress module for the transmitted data. (R/W)

**I2S_TX_PCM_CONF** Compress/Decompress module configuration bit. (R/W)

0: Decompress transmitted data;

1: Compress transmitted data.

Register 22.29. I2S_PD_CONF_REG (0x00a4)

| 31 – 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|
| (reserved) | (reserved) | (reserved) | I2S_FIFO_FORCE_PU | I2S_FIFO_FORCE_PD | |
| 0 0 0 0 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 | 1 | 0 | Reset |

**I2S_FIFO_FORCE_PU** Force FIFO power-up. (R/W)

**I2S_FIFO_FORCE_PD** Force FIFO power-down. (R/W)

Register 22.30. I2S_CONF2_REG (0x00a8)

| (reserved) | I2S_INTER_VALID_EN | I2S_EXT_ADC_START_EN | I2S_LCD_EN | (reserved) | I2S_LCD_TX_SDX2_EN | I2S_LCD_TX_WRX2_EN | I2S_CAMERA_EN | |
|---|---|---|---|---|---|---|---|---|
| 31 … 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 | 1 | 0 0 | 0 | 0 | 0 | Reset |

**I2S_INTER_VALID_EN** Set this bit to enable camera's internal validation. (R/W)

**I2S_EXT_ADC_START_EN** Set this bit to enable the start of external ADC . (R/W)

**I2S_LCD_EN** Set this bit to enable LCD mode. (R/W)

**I2S_LCD_TX_SDX2_EN** Set this bit to duplicate data pairs (Data Frame, Form 2) in LCD mode. (R/W)

**I2S_LCD_TX_WRX2_EN** One datum will be written twice in LCD mode. (R/W)

**I2S_CAMERA_EN** Set this bit to enable camera mode. (R/W)

Register 22.31. I2S_CLKM_CONF_REG (0x00ac)

| (reserved) | I2S_CLKA_ENA | (reserved) | I2S_CLKM_DIV_A | I2S_CLKM_DIV_B | I2S_CLKM_DIV_NUM | |
|---|---|---|---|---|---|---|
| 31 … 22 | 21 | 20 | 19 … 14 | 13 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0x00 | 0x00 | 4 | Reset |

**I2S_CLKA_ENA** Set this bit to enable APLL_CLK. Default is PLL_F160M_CLK. (R/W)

**I2S_CLKM_DIV_A** Fractional clock divider's denominator value. (R/W)

**I2S_CLKM_DIV_B** Fractional clock divider's numerator value. (R/W)

**I2S_CLKM_DIV_NUM** I2S clock divider's integral value. (R/W)

**Register 22.32. I2S_SAMPLE_RATE_CONF_REG (0x00b0)**

| (reserved) | I2S_RX_BITS_MOD | I2S_TX_BITS_MOD | I2S_RX_BCK_DIV_NUM | I2S_TX_BCK_DIV_NUM | |
|---|---|---|---|---|---|
| 31 … 24 | 23 … 18 | 17 … 12 | 11 … 6 | 5 … 0 | |
| 0 0 0 0 0 0 0 0 | 16 | 16 | 6 | 6 | Reset |

**I2S_RX_BITS_MOD** Set the bits to configure the bit length of I2S receiver channel. (R/W)

**I2S_TX_BITS_MOD** Set the bits to configure the bit length of I2S transmitter channel. (R/W)

**I2S_RX_BCK_DIV_NUM** Bit clock configuration bit in receiver mode. (R/W)

**I2S_TX_BCK_DIV_NUM** Bit clock configuration bit in transmitter mode. (R/W)

**Register 22.33. I2S_PDM_CONF_REG (0x00b4)**

| (reserved) | I2S_TX_PDM_HP_BYPASS | I2S_RX_PDM_SINC_DSR_16_EN | I2S_TX_PDM_SIGMADELTA_IN_SHIFT | I2S_TX_PDM_SINC_IN_SHIFT | I2S_TX_PDM_LP_IN_SHIFT | I2S_TX_PDM_HP_IN_SHIFT | (reserved) | I2S_TX_PDM_SINC_OSR2 | I2S_PDM2PCM_CONV_EN | I2S_PCM2PDM_CONV_EN | I2S_RX_PDM_EN | I2S_TX_PDM_EN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 | 24 | 23 22 | 21 20 | 19 18 | 17 16 | 15 8 | 7 4 | 3 | 2 | 1 | 0 | |
| 0 0 0 0 0 0 | 0 | 1 | 0x1 | 0x1 | 0x1 | 0x1 | 0 0 0 0 0 0 0 0 | 0x02 | 1 | 1 | 0 | 0 | Reset |

**I2S_TX_PDM_HP_BYPASS** Set this bit to bypass the transmitter's PDM HP filter. (R/W)

**I2S_RX_PDM_SINC_DSR_16_EN** PDM downsampling rate for filter group 1 in receiver mode. (R/W)

1: downsampling rate = 128;

0: downsampling rate = 64.

**I2S_TX_PDM_SIGMADELTA_IN_SHIFT** Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

**I2S_TX_PDM_SINC_IN_SHIFT** Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

**I2S_TX_PDM_LP_IN_SHIFT** Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

**I2S_TX_PDM_HP_IN_SHIFT** Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

**I2S_TX_PDM_SINC_OSR2** Upsampling rate = 64×i2s_tx_pdm_sinc_osr2 (R/W)

**I2S_PDM2PCM_CONV_EN** Set this bit to enable PDM-to-PCM converter. (R/W)

**I2S_PCM2PDM_CONV_EN** Set this bit to enable PCM-to-PDM converter. (R/W)

**I2S_RX_PDM_EN** Set this bit to enable receiver's PDM mode. (R/W)

**I2S_TX_PDM_EN** Set this bit to enable transmitter's PDM mode. (R/W)

**Register 22.34. I2S_PDM_FREQ_CONF_REG (0x00b8)**

| (reserved) | I2S_TX_PDM_FP | I2S_TX_PDM_FS | |
|---|---|---|---|
| 31 20 | 19 10 | 9 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 960 | 441 | Reset |

**I2S_TX_PDM_FP** PCM-to-PDM converter's PDM frequency parameter. (R/W)

**I2S_TX_PDM_FS** PCM-to-PDM converter's PCM frequency parameter. (R/W)

**Register 22.35. I2S_STATE_REG (0x00bc)**

| (reserved) | I2S_RX_FIFO_RESET_BACK | I2S_TX_FIFO_RESET_BACK | I2S_TX_IDLE | |
|---|---|---|---|---|
| 31 … 3 | 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 1 | Reset |

**I2S_RX_FIFO_RESET_BACK** This bit is used to confirm if the Rx FIFO reset is done. 1: reset is not ready; 0: reset is ready. (RO)

**I2S_TX_FIFO_RESET_BACK** This bit is used to confirm if the Tx FIFO reset is done. 1: reset is not ready; 0: reset is ready. (RO)

**I2S_TX_IDLE** The status bit of the transmitter. 1: the transmitter is idle; 0: the transmitter is busy. (RO)

# Chapter 23

# Pulse Count Controller (PCNT)

## 23.1 Overview

The pulse counter module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

The pulse counter has eight independent units, referred to as PULSE_CNT_U*n*.

The maximum frequency of pulses supported by ESP32's pulse counter is 40 MHz.

## 23.2 Functional Description

## 23.2.1 Architecture

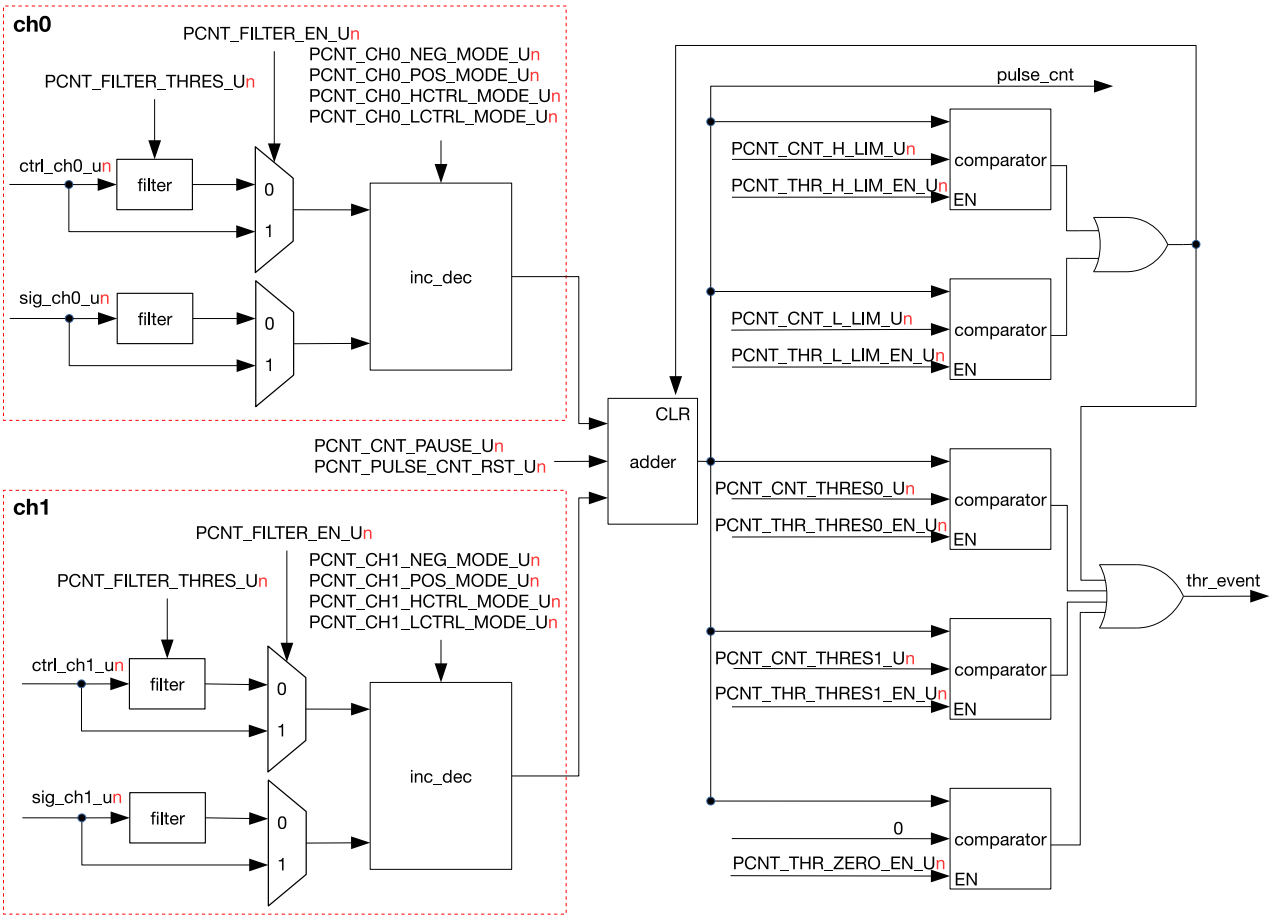



Figure 23.2-1. PULSE_CNT Architecture

The architecture of a pulse counter unit is illustrated in Figure 23.2-1. Each unit has two channels: ch0 and ch1, which are functionally equivalent. Each channel has a signal input, as well as a control input, which can both be connected to I/O pads. The counting behavior on both the positive and negative edge can be configured separately to increase, decrease, or do nothing to the counter value. Separately, for both control signal levels, the hardware can be configured to modify the edge action: invert it, disable it, or do nothing. The counter itself is a 16-bit signed up/down counter. Its value can be read by software directly, but is also monitored by a set of comparators which can trigger an interrupt.

## 23.2.2 Counter Channel Inputs

As stated before, the two inputs of a channel can affect the pulse counter in various ways. The specifics of this behaviour are set by LCTRL_MODE and HCTRL_MODE in this case when the control signal is low or high, respectively, and POS_MODE and NEG_MODE for positive and negative edges of the input signal. Setting POS_MODE and NEG_MODE to 1 will increase the counter when an edge is detected, setting them to 2 will decrease the counter and setting at any other value will neutralize the effect of the edge on the counter. LCTR_MODE and HCTR_MODE modify this behaviour, when the control input has the corresponding low or high value: 0 does not modify the NEG_MODE and POS_MODE behaviour, 1 inverts it (setting POS_MODE/NEG_MODE to increase the counter should now decrease the counter and vice versa) and any other value disables counter effects for that signal level.

To summarize, a few examples have been considered. In this table, the effect on the counter for a rising edge is shown for both a low and a high control signal, as well as various other configuration options. For clarity, a short description in brackets is added after the values. Note: x denotes 'do not care'.

| POS_ MODE | LCTRL_ MODE | HCTRL_ MODE | sig l→h when ctrl=0 | sig l→h when ctrl=1 |
|---|---|---|---|---|
| 1 (inc) | 0 (-) | 0 (-) | Inc ctr | Inc ctr |
| 2 (dec) | 0 (-) | 0 (-) | Dec ctr | Dec ctr |
| 0 (-) | x | x | No action | No action |
| 1 (inc) | 0 (-) | 1 (inv) | Inc ctr | Dec ctr |
| 1 (inc) | 1 (inv) | 0 (-) | Dec ctr | Inc ctr |
| 2 (dec) | 0 (-) | 1 (inv) | Dec ctr | Inc ctr |
| 1 (inc) | 0 (-) | 2 (dis) | Inc ctr | No action |
| 1 (inc) | 2 (dis) | 0 (-) | No action | Inc ctr |

This table is also valid for negative edges (sig h→l) on substituting NEG_MODE for POS_MODE.

Each pulse counter unit also features a filter on each of the four inputs, adding the option to ignore short glitches in the signals. If a PCNT_FILTER_EN_U*n* can be set to filter the four input signals of the unit. If this filter is enabled, any pulses shorter than REG_FILTER_THRES_U*n* number of APB_CLK clock cycles will be filtered out and will have no effect on the counter. With the filter disabled, in theory infinitely small glitches could possibly trigger pulse counter action. However, in practice the signal inputs are sampled on APB_CLK edges and even with the filter disabled, pulse widths lasting shorter than one APB_CLK cycle may be missed.

Apart from the input channels, software also has some control over the counter. In particular, the counter value can be frozen to the current value by configuring PCNT_CNT_PAUSE_U*n*. It can also be reset to 0 by configuring PCNT_PLUS_CNT_RST_U*n*.

### 23.2.3 Watchpoints

The pulse counters have five watchpoints that share one interrupt. Interrupt generation can be enabled or disabled for each individual watchpoint. The watchpoints are:

- Maximum count value: Triggered when PULSE_CNT >= PCNT_CNT_H_LIM_U*n*. Additionally, this will reset the counter to 0. PCNT_CNT_H_LIM_U*n* should be a positive number.
- Minimum count value: Triggered when PULSE_CNT <= PCNT_CNT_L_LIM_U*n*. Additionally, this will reset the counter to 0. PCNT_CNT_L_LIM_U*n* should be a negative number.
- Two threshold values: Triggered when PULSE_CNT = PCNT_THR_THRESO_U*n* or PCNT_THR_THRES1_U*n*.
- Zero: Triggered when PULSE_CNT = 0.

### 23.2.4 Examples

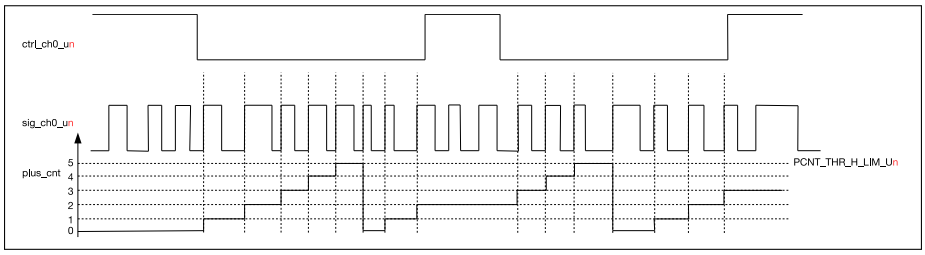



Figure 23.2-2. PULSE_CNT Upcounting Diagram

Figure 23.2-2 shows channel 0 being used as an up-counter. The configuration of channel 0 is shown below.

- CNT_CH0_POS_MODE_U*n* = 1: increase counter on the rising edge of sig_ch0_u*n*.
- PCNT_CH0_NEG_MODE_U*n* = 0: no counting on the falling edge of sig_ch0_u*n*.
- PCNT_CH0_LCTRL_MODE_U*n* = 0: Do not modify counter mode when ctrl_ch0_u*n* is low.
- PCNT_CH0_HCTRL_MODE_U*n* = 2: Do not allow counter increments/decrements when ctrl_ch0_u*n* is high.
- PCNT_CNT_H_LIM_U*n* = 5: PULSE_CNT resets to 0 when the count value increases to 5.

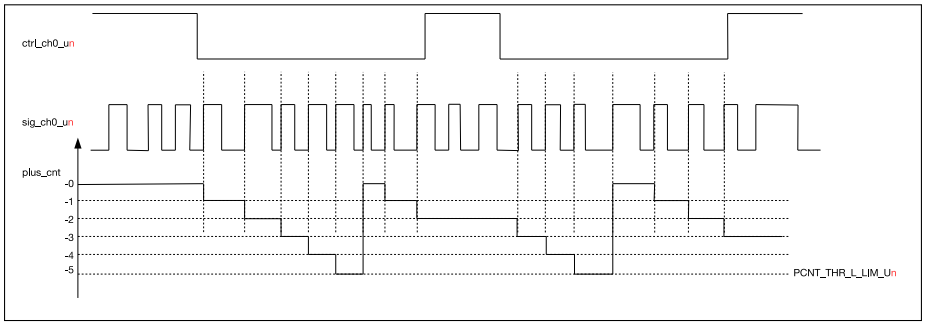



Figure 23.2-3. PULSE_CNT Downcounting Diagram

Figure 23.2-3 shows channel 0 decrementing the counter. The configuration of channel 0 differs from that in Figure 23.2-2 in the following two aspects:

- PCNT_CH0_LCTRL_MODE_U*n* = 1: invert counter mode when ctrl_ch0_u*n* is at low level, so it will decrease, rather than increase, the counter.
- PCNT_CNT_H_LIM_U*n* = −5: PULSE_CNT resets to 0 when the count value decreases to −5.

### 23.2.5 Interrupts

PCNT_CNT_THR_EVENT_U*n*_INT: This interrupt gets triggered when one of the five channel comparators detects a match.

## 23.3 Register Summary

The addresses in this section are relative to the PCNT base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| PCNT_U0_CONF0_REG | Configuration register 0 for unit 0 | 0x3FF57000 | R/W |
| PCNT_U1_CONF0_REG | Configuration register 0 for unit 1 | 0x3FF5700C | R/W |
| PCNT_U2_CONF0_REG | Configuration register 0 for unit 2 | 0x3FF57018 | R/W |
| PCNT_U3_CONF0_REG | Configuration register 0 for unit 3 | 0x3FF57024 | R/W |
| PCNT_U4_CONF0_REG | Configuration register 0 for unit 4 | 0x3FF57030 | R/W |
| PCNT_U5_CONF0_REG | Configuration register 0 for unit 5 | 0x3FF5703C | R/W |
| PCNT_U6_CONF0_REG | Configuration register 0 for unit 6 | 0x3FF57048 | R/W |
| PCNT_U7_CONF0_REG | Configuration register 0 for unit 7 | 0x3FF57054 | R/W |
| PCNT_U0_CONF1_REG | Configuration register 1 for unit 0 | 0x3FF57004 | R/W |
| PCNT_U1_CONF1_REG | Configuration register 1 for unit 1 | 0x3FF57010 | R/W |
| PCNT_U2_CONF1_REG | Configuration register 1 for unit 2 | 0x3FF5701C | R/W |
| PCNT_U3_CONF1_REG | Configuration register 1 for unit 3 | 0x3FF57028 | R/W |
| PCNT_U4_CONF1_REG | Configuration register 1 for unit 4 | 0x3FF57034 | R/W |
| PCNT_U5_CONF1_REG | Configuration register 1 for unit 5 | 0x3FF57040 | R/W |
| PCNT_U6_CONF1_REG | Configuration register 1 for unit 6 | 0x3FF5704C | R/W |
| PCNT_U7_CONF1_REG | Configuration register 1 for unit 7 | 0x3FF57058 | R/W |
| PCNT_U0_CONF2_REG | Configuration register 2 for unit 0 | 0x3FF57008 | R/W |
| PCNT_U1_CONF2_REG | Configuration register 2 for unit 1 | 0x3FF57014 | R/W |
| PCNT_U2_CONF2_REG | Configuration register 2 for unit 2 | 0x3FF57020 | R/W |
| PCNT_U3_CONF2_REG | Configuration register 2 for unit 3 | 0x3FF5702C | R/W |
| PCNT_U4_CONF2_REG | Configuration register 2 for unit 4 | 0x3FF57038 | R/W |
| PCNT_U5_CONF2_REG | Configuration register 2 for unit 5 | 0x3FF57044 | R/W |
| PCNT_U6_CONF2_REG | Configuration register 2 for unit 6 | 0x3FF57050 | R/W |
| PCNT_U7_CONF2_REG | Configuration register 2 for unit 7 | 0x3FF5705C | R/W |
| **Counter values** | | | |
| PCNT_U0_CNT_REG | Counter value for unit 0 | 0x3FF57060 | RO |
| PCNT_U1_CNT_REG | Counter value for unit 1 | 0x3FF57064 | RO |
| PCNT_U2_CNT_REG | Counter value for unit 2 | 0x3FF57068 | RO |
| PCNT_U3_CNT_REG | Counter value for unit 3 | 0x3FF5706C | RO |
| PCNT_U4_CNT_REG | Counter value for unit 4 | 0x3FF57070 | RO |
| PCNT_U5_CNT_REG | Counter value for unit 5 | 0x3FF57074 | RO |
| PCNT_U6_CNT_REG | Counter value for unit 6 | 0x3FF57078 | RO |
| PCNT_U7_CNT_REG | Counter value for unit 7 | 0x3FF5707C | RO |
| **Control registers** | | | |
| PCNT_CTRL_REG | Control register for all counters | 0x3FF570B0 | R/W |
| **Interrupt registers** | | | |

| Name | Description | Address | Access |
|---|---|---|---|
| PCNT_INT_RAW_REG | Raw interrupt status | 0x3FF57080 | RO |
| PCNT_INT_ST_REG | Masked interrupt status | 0x3FF57084 | RO |
| PCNT_INT_ENA_REG | Interrupt enable bits | 0x3FF57088 | R/W |
| PCNT_INT_CLR_REG | Interrupt clear bits | 0x3FF5708C | WO |
| **Status registers** | | | |
| PCNT_U*n*_STATUS_REG | Indicate the status of counter | 0x3FF57090 | RO |

## 23.4 Registers

The addresses in this section are relative to the PCNT base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 23.3 *Register Summary*.

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

**Register 23.1. PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0C**n*)**

| PCNT_CH1_LCTRL_MODE_U*n* | PCNT_CH1_HCTRL_MODE_U*n* | PCNT_CH1_POS_MODE_U*n* | PCNT_CH1_NEG_MODE_U*n* | PCNT_CH0_LCTRL_MODE_U*n* | PCNT_CH0_HCTRL_MODE_U*n* | PCNT_CH0_POS_MODE_U*n* | PCNT_CH0_NEG_MODE_U*n* | PCNT_THR_THRES1_EN_U*n* | PCNT_THR_THRES0_EN_U*n* | PCNT_THR_L_LIM_EN_U*n* | PCNT_THR_H_LIM_EN_U*n* | PCNT_THR_ZERO_EN_U*n* | PCNT_FILTER_EN_U*n* | PCNT_FILTER_THRES_U*n* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 30 | 29 28 | 27 26 | 25 24 | 23 22 | 21 20 | 19 18 | 17 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0x010 | Reset |

**PCNT_CH1_LCTRL_MODE_U*n*** This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase → decrease, decrease → increase); 2, 3: Inhibit counter modification

**PCNT_CH1_HCTRL_MODE_U*n*** This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is high. (R/W) 0: No modification; 1: Invert behaviour (increase → decrease, decrease → increase); 2, 3: Inhibit counter modification

**PCNT_CH1_POS_MODE_U*n*** This register sets the behaviour when the signal input of channel 1 detects a positive edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

**PCNT_CH1_NEG_MODE_U*n*** This register sets the behaviour when the signal input of channel 1 detects a negative edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

**PCNT_CH0_LCTRL_MODE_U*n*** This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase → decrease, decrease → increase); 2, 3: Inhibit counter modification

**PCNT_CH0_HCTRL_MODE_U*n*** This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is high. (R/W) 0: No modification; 1: Invert behaviour (increase → decrease, decrease → increase); 2, 3: Inhibit counter modification

**PCNT_CH0_POS_MODE_U*n*** This register sets the behaviour when the signal input of channel 0 detects a positive edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

**PCNT_CH0_NEG_MODE_U*n*** This register sets the behaviour when the signal input of channel 0 detects a negative edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

**PCNT_THR_THRES1_EN_U*n*** This is the enable bit for unit *n*'s thres1 comparator. (R/W)

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

### Register 23.1. PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0C***n*)

Continued from the previous page...

**PCNT_THR_THRES0_EN_U*n*** This is the enable bit for unit *n*'s thres0 comparator. (R/W)

**PCNT_THR_L_LIM_EN_U*n*** This is the enable bit for unit *n*'s thr_l_lim comparator. (R/W)

**PCNT_THR_H_LIM_EN_U*n*** This is the enable bit for unit *n*'s thr_h_lim comparator. (R/W)

**PCNT_THR_ZERO_EN_U*n*** This is the enable bit for unit *n*'s zero comparator. (R/W)

**PCNT_FILTER_EN_U*n*** This is the enable bit for unit *n*'s input filter. (R/W)

**PCNT_FILTER_THRES_U*n*** This sets the maximum threshold, in APB_CLK cycles, for the filter. Any pulses lasting shorter than this will be ignored when the filter is enabled. (R/W)

### Register 23.2. PCNT_U*n*_CONF1_REG (*n*: 0-7) (0x4+0x0C***n*)

| PCNT_CNT_THRES1_U*n* | PCNT_CNT_THRES0_U*n* | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x000 | 0x000 | Reset |

**PCNT_CNT_THRES1_U*n*** This register is used to configure the thres1 value for unit *n*. (R/W)

**PCNT_CNT_THRES0_U*n*** This register is used to configure the thres0 value for unit *n*. (R/W)

### Register 23.3. PCNT_U*n*_CONF2_REG (*n*: 0-7) (0x8+0x0C***n*)

| PCNT_CNT_L_LIM_U*n* | PCNT_CNT_H_LIM_U*n* | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x000 | 0x000 | Reset |

**PCNT_CNT_L_LIM_U*n*** This register is used to configure the thr_l_lim value for unit *n*. (R/W)

**PCNT_CNT_H_LIM_U*n*** This register is used to configure the thr_h_lim value for unit *n*. (R/W)

**Register 23.4. PCNT_U*n*_CNT_REG (*n*: 0-7) (0x60+0x04**n*)**

| (reserved) | PCNT_PLUS_CNT_U*n* | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x00000 | Reset |

**PCNT_PLUS_CNT_U*n*** This register stores the current pulse count value for unit *n*. (RO)

**Register 23.5. PCNT_INT_RAW_REG (0x0080)**

| (reserved) | PCNT_CNT_THR_EVENT_U7_INT_RAW | PCNT_CNT_THR_EVENT_U6_INT_RAW | PCNT_CNT_THR_EVENT_U5_INT_RAW | PCNT_CNT_THR_EVENT_U4_INT_RAW | PCNT_CNT_THR_EVENT_U3_INT_RAW | PCNT_CNT_THR_EVENT_U2_INT_RAW | PCNT_CNT_THR_EVENT_U1_INT_RAW | PCNT_CNT_THR_EVENT_U0_INT_RAW | |
|---|---|---|---|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PCNT_CNT_THR_EVENT_U*n*_INT_RAW** The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

**Register 23.6. PCNT_INT_ST_REG (0x0084)**

| (reserved) | PCNT_CNT_THR_EVENT_U7_INT_ST | PCNT_CNT_THR_EVENT_U6_INT_ST | PCNT_CNT_THR_EVENT_U5_INT_ST | PCNT_CNT_THR_EVENT_U4_INT_ST | PCNT_CNT_THR_EVENT_U3_INT_ST | PCNT_CNT_THR_EVENT_U2_INT_ST | PCNT_CNT_THR_EVENT_U1_INT_ST | PCNT_CNT_THR_EVENT_U0_INT_ST | |
|---|---|---|---|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PCNT_CNT_THR_EVENT_U*n*_INT_ST** The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

**Register 23.7. PCNT_INT_ENA_REG (0x0088)**

| (reserved) | PCNT_CNT_THR_EVENT_U7_INT_ENA | PCNT_CNT_THR_EVENT_U6_INT_ENA | PCNT_CNT_THR_EVENT_U5_INT_ENA | PCNT_CNT_THR_EVENT_U4_INT_ENA | PCNT_CNT_THR_EVENT_U3_INT_ENA | PCNT_CNT_THR_EVENT_U2_INT_ENA | PCNT_CNT_THR_EVENT_U1_INT_ENA | PCNT_CNT_THR_EVENT_U0_INT_ENA | |
|---|---|---|---|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PCNT_CNT_THR_EVENT_U*n*_INT_ENA** The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

**Register 23.8. PCNT_INT_CLR_REG (0x008c)**

| (reserved) | PCNT_CNT_THR_EVENT_U7_INT_CLR | PCNT_CNT_THR_EVENT_U6_INT_CLR | PCNT_CNT_THR_EVENT_U5_INT_CLR | PCNT_CNT_THR_EVENT_U4_INT_CLR | PCNT_CNT_THR_EVENT_U3_INT_CLR | PCNT_CNT_THR_EVENT_U2_INT_CLR | PCNT_CNT_THR_EVENT_U1_INT_CLR | PCNT_CNT_THR_EVENT_U0_INT_CLR | |
|---|---|---|---|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PCNT_CNT_THR_EVENT_U*n*_INT_CLR** Set this bit to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

### Register 23.9. PCNT_CTRL_REG (0x00b0)

| (reserved) | PCNT_CLK_EN | PCNT_CNT_PAUSE_U7 | PCNT_PLUS_CNT_RST_U7 | PCNT_CNT_PAUSE_U6 | PCNT_PLUS_CNT_RST_U6 | PCNT_CNT_PAUSE_U5 | PCNT_PLUS_CNT_RST_U5 | PCNT_CNT_PAUSE_U4 | PCNT_PLUS_CNT_RST_U4 | PCNT_CNT_PAUSE_U3 | PCNT_PLUS_CNT_RST_U3 | PCNT_CNT_PAUSE_U2 | PCNT_PLUS_CNT_RST_U2 | PCNT_CNT_PAUSE_U1 | PCNT_PLUS_CNT_RST_U1 | PCNT_CNT_PAUSE_U0 | PCNT_PLUS_CNT_RST_U0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 – 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | Reset |

**PCNT_CLK_EN** Configures register clock gating.

0: Support clock only when the application writes registers.

1: Always force the clock on for registers.

(R/W)

**PCNT_CNT_PAUSE_U*n*** Set this bit to freeze unit *n*'s counter. (R/W)

**PCNT_PLUS_CNT_RST_U*n*** Set this bit to clear unit *n*'s counter. (R/W)

### Register 23.10. PCNT_U*n*_STATUS_REG (*n*: 0-7) (0x90+0x0C*n)

| (reserved) | PCNT_THR_ZERO_LAT_U*n* | PCNT_THR_H_LIM_LAT_U*n* | PCNT_THR_L_LIM_LAT_U*n* | PCNT_THR_THRES0_LAT_U*n* | PCNT_THR_THRES1_LAT_U*n* | PCNT_THR_ZERO_MODE_U*n* | |
|---|---|---|---|---|---|---|---|
| 31 – 7 | 6 | 5 | 4 | 3 | 2 | 1 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 00 | Reset |

**PCNT_THR_ZERO_LAT_U*n*** The last interrupt happened on counter for unit *n* reaching 0. (RO)

**PCNT_THR_H_LIM_LAT_U*n*** The last interrupt happened on counter for unit *n* reaching thr_h_lim. (RO)

**PCNT_THR_L_LIM_LAT_U*n*** The last interrupt happened on counter for unit *n* reaching thr_l_lim. (RO)

**PCNT_THR_THRES0_LAT_U*n*** The last interrupt happened on counter for unit *n* reaching thres0. (RO)

**PCNT_THR_THRES1_LAT_U*n*** The last interrupt happened on counter for unit *n* reaching thres1. (RO)

**PCNT_THR_ZERO_MODE_U*n*** This register stores the current status of the counter. 0: counting value is +0 (the counter values are represented by signed binary numbers); 1: counting value is -0; 2: counting value is negative; 3: counting value is positive. (RO)

# Chapter 24

# Ethernet Media Access Controller (EMAC)

## 24.1 Overview

### Features of Ethernet

By using the external Ethernet PHY (physical layer), ESP32 can send and receive data via Ethernet MAC (Media Access Controller) according to the IEEE 802.3 standard, as Figure 24.1-1 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.

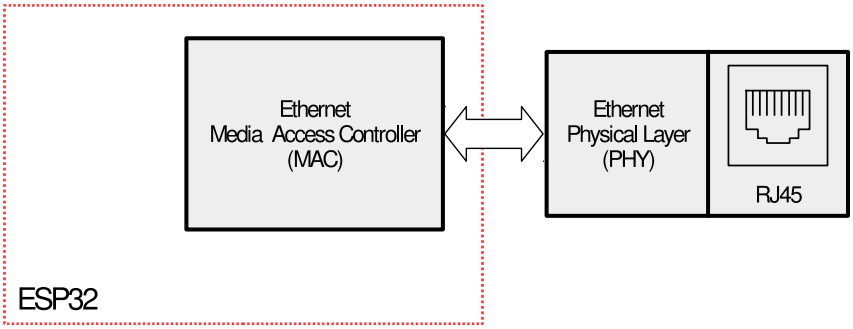



Figure 24.1-1. Ethernet MAC Functionality Overview

ESP32 MAC Ethernet complies with the following criteria:

- IEEE 802.3-2002 for Ethernet MAC
- Two industry-standard interfaces conforming with IEEE 802.3-2002: Media-Independent Interface (MII) and Reduced Media-Independent Interface (RMII).

### Features of MAC Layer

- Support for a data transmission rate of 10 Mbit/s or 100 Mbit/s through an external PHY interface
- Communication with an external Fast Ethernet PHY through IEEE 802.3-compliant MII and RMII interfaces
- Support for:
  - Carrier Sense Multiple Access / Collision Detection (CSMA/CD) protocol in half-duplex mode
  - IEEE 802.3x flow control in full-duplex mode
  - operations in full-duplex mode, forwarding the received pause-control frame to the user application
  - backpressure flow control in half-duplex mode

  - If the flow control input signal disappears during a full-duplex operation, a pause frame with zero pause time value is automatically transmitted.
- The Preamble and the Start Frame Delimiter (SFD) are inserted in the Transmit path, and deleted in the Receive path.
- Cyclic Redundancy Check (CRC) and Pad can be controlled on a per-frame basis.
- The Pad is generated automatically, if data is below the minimum frame length.
- Programmable frame length supporting jumbo frames of up to 16 KB
- Programmable Inter-frame Gap (IFG) (40-96 bit times in steps of 8)
- Support for a variety of flexible address filtering modes:
  - Up to eight 48-bit perfect address filters to mask each byte
  - Up to eight 48-bit SA address comparison checks to mask each byte
  - All multicast address frames can be transmitted
  - All frames in mixed mode can be transmitted without being filtered for network monitoring
  - A status report is attached each time all incoming packets are transmitted and filtered
- Returning a 32-bit status for transmission and reception of packets respectively
- Separate transmission, reception, and control interfaces for the application
- Use of the Management Data Input/Output (MDIO) interface to configure and manage PHY devices
- Support for the offloading of received IPv4 and TCP packets encapsulated by an Ethernet frame in the reception function
- Support for checking IPv4 header checksums, as well as TCP, UDP, or ICMP (Internet Control Message Protocol) checksums encapsulated in IPv4/IPv6 packets in the enhanced reception function
- Two sets of FIFOs: one 2 KB Tx FIFO with programmable threshold and one 2 KB Rx FIFO with configurable threshold (64 bytes by default)
- When Rx FIFO stores multiple frames, the Receive Status Vector is inserted into the Rx FIFO after transmitting an EOF (end of frame), so that the Rx FIFO does not need to store the Receive Status of these frames.
- In store-and-forward mode, all error frames can be filtered during reception, but not forwarded to the application.
- Under-sized good frames can be forwarded.
- Support for data statistics by generating pulses for lost or corrupted frames in the Rx FIFO due to an overflow
- Support for store-and-forward mechanism when transmitting data to the MAC core
- Automatic re-transmission of collided frames during transmission (subject to certain conditions, see section 24.2.1.2)
- Discarding frames in cases of late collisions, excessive collisions, excessive deferrals, and under-run conditions

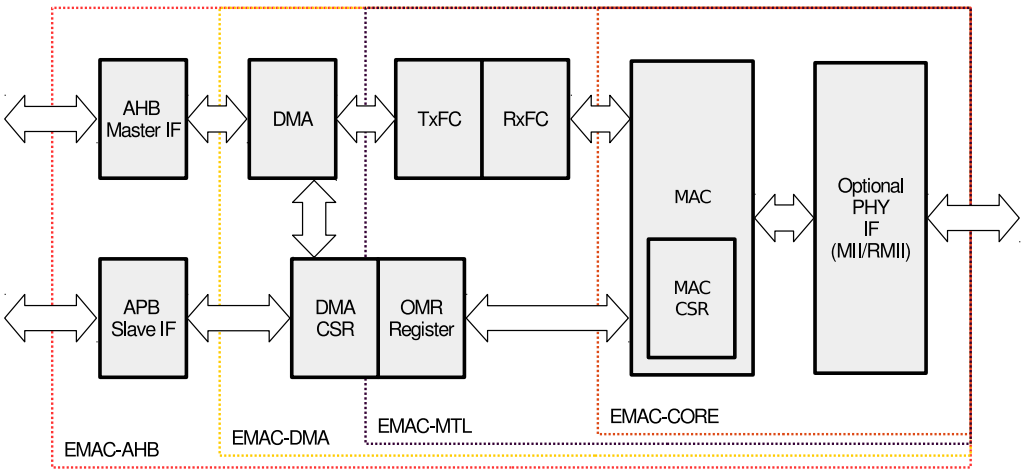



Figure 24.1-2. Ethernet Block Diagram

- The Tx FIFO is flushed by software control.
- Calculating the IPv4 header checksum, as well as the TCP, UDP, or ICMP checksum, and then inserting them into frames transmitted in store-and-forward mode.

**Ethernet Block Diagram**

Figure 24.1-2 shows the block diagram of the Ethernet.

Ethernet MAC consists of the MAC-layer configuration register module and 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.

## 24.2 EMAC_CORE

The MAC supports many interfaces with the PHY chip. The PHY interface can be selected only once after reset. The MAC communicates with the application side (DMA side), using the MAC Transmit Interface (MTI), MAC Receive Interface (MRI) and the MAC Control Interface (MCI).

### 24.2.1 Transmit Operation

A transmit operation is initiated when the MTL Application pushes in data at the time a response signal is asserted. When the SOF (start of frame) 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, varies, depending on delay factors like IFG delay, time to transmit Preamble or SFD (Start Frame Delimiter), and any back-off delays in half-duplex mode. Until then, the MAC does not accept the data received from MTL by de-asserting the ready signal.

After the EOF (end of frame) is transmitted to the MAC, the MAC completes the normal transmission and yields the Transmit Status to the MTL. If a normal collision (in half-duplex mode) occurs during transmission, the MAC makes valid the Transmit Status in the MTL. It then accepts and drops all further data until the next SOF is

received. The MTL block should retransmit the same frame from SOF upon 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 transmission. During the normal transmission of a frame from MTL, if the MAC receives an SOF without getting an EOF for the previous frame, it ignores the SOF and considers the new frame as a continuation of the previous one.

#### 24.2.1.1 Transmit Flow Control

In full-duplex mode, when the Transmit Flow Control Enable bit (TFE bit in the Flow Control Register) is set to 1, the MAC will generate and send a pause frame, as needed. The pause frame is added and transmitted together with the calculated CRC. The generation of pause frames can be initiated in two ways.

When the application sets the Flow Control Busy bit (FCB bit in the Flow Control Register) to 1, or when the Rx FIFO is full, a pause frame is transmitted.

- If an application has requested flow control by setting the FCB bit in the Flow Control Register to 1, the MAC will generate and send a single pause frame. The pause time value in the generated frame is the pause time value programmed in the Flow Control Register. To extend or end the pause time before the time specified in the previously transmitted pause frame, the application program must configure the pause time value in the Flow Control Register to the appropriate value and, then, request another pause frame transmission.
- 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 of the generated frame is the pause time value programmed in the Flow Control Register. If the Rx FIFO remains full during the configurable interval, which is determined by the Pause Low Threshold bit (PLT) in the Flow Control Register before the pause time expires, a second pause frame will be transmitted. As long as the Rx FIFO remains full, the process repeats itself. If the FIFO is no longer full before the sample time, the MAC will send a pause frame with zero pause time, indicating to the remote end that the Rx buffer is ready to receive the new data frame.

#### 24.2.1.2 Retransmission During a Collision

In half-duplex mode, a collision may occur on the MAC line interface when frames are transmitted to the MAC. The MAC may even give a status to indicate a retry before the end of the frame is received. The retransmission is then enabled and the frame is popped out from the FIFO. When more than 96 bytes are transmitted to the MAC core, the FIFO controller frees the space in the FIFO, allowing the DMA to push more data into FIFO. This means that data cannot be retransmitted after the threshold is exceeded or when the MAC core indicates that a late collision has occurred.

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, and late collision. When frame transmission is aborted because of collision, the MAC requests retransmission of the frame.

### 24.2.2 Receive Operation

A receive operation is initiated when the MAC detects an SFD on the RMII or MII. The MAC strips the Preamble and SFD before processing the frame. The header fields are checked for the 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 frame received by the MAC will be pushed into the Rx FIFO. Once the FIFO status exceeds the Receive Threshold, configured by the Receive Threshold Control (RTC) bit in the Operation Mode register, the DMA can initiate a preconfigured burst transmission to the AHB interface.

In the default pass-through mode, when the FIFO receives a complete packet or 64 bytes configured by the RTC bit in the Operation Mode Register, the data pops up and its availability is notified to the DMA. After the DMA initiates the transmission to the AHB interface, the data transmission continues from the FIFO until the complete packet is transmitted. Upon completing transmitting the EOF, the status word will pop up and be transmitted to the DMA controller.

In the Rx FIFO Store-and-Forward mode (configured through the RSF or Receive Store and Forward bit in the Operation Mode Register), only the valid frames are read and forwarded to the application. In the passthrough mode, error frames are not discarded because the error status is received at the end of the frame. The start of frame will have been read from the FIFO at that point.

#### 24.2.2.1 Reception Protocol

After the receive module receives the packets, the Preamble and SFD of the received frames are removed. When the SFD is detected, the MAC starts sending Ethernet frame data to the Rx FIFO, starting at the first byte (destination address) following the SFD.

If the received frame length/type is less than 0x600 and the automatic CRC/Pad removal option is programmed for the MAC, the MAC will send frame data to the Rx FIFO (the amount of data does not exceed the number specified in the length/type field). Then MAC begins discarding the remaining section, including the FCS field. If the frame length/type is greater than, or equal to, 0x600, the MAC will send all received Ethernet frame data to the Rx FIFO, regardless of the programmed value of the automatic CRC removal option. By default, the MAC watchdog timer is enabled, meaning that frames, including DA, SA, LT, data, pad and FCS, which exceed 2048 bytes, are cut off. This function can be disabled by programming the Watchdog Disable (WD) bit in the MAC Configuration Register. However, even if the watchdog timer is disabled, frames longer than 16 KB will be cut off and the watchdog timeout status will be given.

#### 24.2.2.2 Receive Frame Controller

If the RA (Receive All) bit in the MAC Frame Filter Register is reset, the MAC will filter frames based on the destination and source addresses. If the application decides not to receive any bad frames, such as runt frames and CRC error frames, another level of filtering is needed. When a frame fails the filtering, the frame is discarded and is not transmitted to the application. When the filter parameters are changed dynamically, if a frame fails the DA and SA filterings, the remaining part of the frame is discarded and the Receive Status word is updated immediately and, therefore, the zero frame length bit, CRC error bit, and runt frame error bit are set to 1. This indicates that the frame has failed the filtering.

#### 24.2.2.3 Receive Flow Control

The MAC will detect the received pause frame and pause transmission of frames for a specified delay within the received pause frame (in full-duplex mode only). The Pause Frame Detect Function can be enabled or disabled by the RFCE (Receive Flow Control Enable) bit in the Flow Control Register. When receive flow control is enabled, it starts monitoring whether the destination address of the received frame matches 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 will determine whether to transmit the received control frame to the application, according to the PCF (Pass Control Frames) bit in the Frame Filter Register.

The MAC will also decode the type, the opcode, and the pause timer field of the Receive Control Frame. If the value of the status byte counter is 64 bits and there are no CRC errors, the MAC transmitter will halt the transmission of any data frame. The duration of the pause is the decoded pause time value multiplied by the interval (which is 64 bytes for both 10 Mbit/s and 100 Mb/s modes). At the same time, if another pause frame of zero pause time is detected, the MAC will reset the pause time to manage the new pause request.

If the type field (0x8808), the opcode (0x00001), and the byte length (64 bytes) of the received control frame are not 0x8808, 0x00001, and 64 bytes, respectively, or if there is a CRC error, the MAC will not generate a pause.

If a pause frame has a multicast destination address, the MAC filters the frame, according to the address matching.

For pause frames with a unicast destination address, the MAC checks whether the DA matches the content of the EMACADDR0 Register, and whether the Unicast Pause Frame Detect (UPFD) bit in the Flow Control Register is set to 1. The Pass Control Frames (PCF) bits in the Frame Filter Register [7:6] control the filtering of frames and addresses.

#### 24.2.2.4 Reception of Multiple Frames

Since the status is available immediately after the data is received. Frames can be stored there, as long as the FIFO is not full.

#### 24.2.2.5 Error Handling

If the Rx FIFO is full before receiving the EOF data from the MAC, an overflow will be generated and the entire frame will be discarded. In fact, status bit RDES0[11] will indicate that this frame is partial due to an overflow, and that it should be discarded.

If the function that corresponds to the Flush Transmit FIFO (FTF) bit and the Forward Undersized Good Frames (FUGF) bit in the Operation Mode Register is enabled, the Rx FIFO can filter error frames and runt frames. If the receive FIFO is configured to operate in store-and-forward mode, all error frames will be filtered and discarded.

In passthrough mode, if a frame's status and length are available when reading a SOF from the Rx FIFO, the entire error frame can be discarded. DMA can clear the error frame being read from the FIFO by enabling the Receive Frame Clear bit. The data transmission to the application (DMA) will then stop, and the remaining frames will be read internally and discarded. If FIFO is available, the transmission of the next frame will be initiated.

#### 24.2.2.6 Receive Status Word

After receiving the Ethernet frames, the MAC outputs the receive status to the application. The detailed description of the receive status is the same as that which is configured by bit [31:0] in RDES0.

## 24.3 MAC Interrupt Controller

The MAC core can generate interrupts due to various events.

The interrupt register bits only indicate various interrupt events. To clear the interrupts, the corresponding status register and other registers must be read. An Interrupt Status Register describes the events that prompt the MAC core to generate interrupts. Each interrupt event can be prevented by setting the corresponding mask bit in the Interrupt Mask Register to 1. For example, if bit3 of the interrupt register is set high, it indicates that a magic packet or Wake-on-LAN frame has been received in Power-down mode. The PMT Control and Status register must be read to clear this interrupt event.

## 24.4 MAC Address Filtering

Address filtering will check the destination and source addresses of all received frames and report the address filtering status accordingly. For example, filtered frames can be identified either as multicast or broadcast.The address check, then, is based on the parameters selected by the application (Frame Filter Registers).

Physical (MAC) addresses are used for address checking during address filtering.

### 24.4.1 Unicast Destination Address Filtering

The MAC supports up to 8 MAC addresses for perfect filtering of unicast addresses. If a perfect filtering is selected (by resetting bit[1] in the Frame Filter Register), the MAC compares all 48 bits of the received unicast address with the programmed MAC address to determine if there is a match. By default, EMACADDR0 is always enabled, and the other addresses (EMACADDR0 ~ EMACADDR7) are selected by a separate enable bit. When the individual bytes of the other addresses (EMACADDR0 ~ EMACADDR7) are compared with the DA bytes received, the latter can be masked by setting the corresponding Mask Byte Control bit in the register to 1. This facilitates the DA group address filtering.

### 24.4.2 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 will filter multicast addresses, according to Bit[2] in the Frame Filter Register.

In perfect filtering mode, the multicast address is compared with the programmed MAC Destination Address Registers (EMACADDR0 ~ EMACADDR7). Group address filtering is also supported.

### 24.4.3 Broadcast Address Filtering

The MAC does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames, which can happen by setting the Disable Broadcast Frames (DBF) bit in the Frame Filter Register to 1, all broadcast frames will be discarded.

### 24.4.4 Unicast Source Address Filtering

The MAC may also perform a perfect filtering based on the source address field of the received frame. By default, the Address Filtering Module (AFM) compares the Source Address (SA) field with the values

programmed in the SA register. By setting Bit[30] in the SA register to 1, the MAC Address Register (EMACADDR0 - EMACADDR7) can be configured to contain SA, instead of Destination Address (DA), for filtering. Group filtering with SA is also supported. If the Source Address Filter (SAF) enable bit in the Frame Filter Register is set to 1, the MAC discards frames that do not pass the SA filtering. Otherwise, the result of SA filtering is given as a status bit in the Receive Status word (Please refer to Table 24.8-5).

When the SAF enable bit is set to 1, the result of the SA filtering and DA filtering is AND'ed to determine whether or not to forward the frame. Any frame that fails to pass will be discarded. Frames need to pass both filterings in order to be forwarded to the application.

### 24.4.5 Inverse Filtering Operation

For both destination address (DA) and source address (SA) filtering, you can invert the results matched through the filtering at the final output. The inverse filtering of DA and SA are controlled by the DAIF and SAIF bits, respectively, in the Frame Filter Register. The DAIF bit applies to both unicast and multicast DA frames. When DAIF is set to 1, the result of unicast or multicast destination address filtering will be inverted. Similarly, when the SAIF bit is set to 1, the result of unicast SA filtering is reversed.

The following two tables summarize the destination address and source address filtering, based on the type of the frames received.

**Table 24.4-1. Destination Address Filtering**

| Frame Type | PM | PF | DAIF | PAM | DB | DA Filter Result |
|---|---|---|---|---|---|---|
| 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 | All frames pass. |
| | 0 | X | 0 | X | X | Pass when results of perfect/group filtering match. |
| | 0 | X | 1 | X | X | Fail when results of perfect/group filtering match. |
| | 0 | 1 | 0 | X | X | Pass when results of perfect/group filtering match. |
| | 0 | 1 | 1 | X | X | Fail when results of perfect/group filtering match. |
| Multicast | 1 | X | X | X | X | All frames pass. |
| | X | X | X | 1 | X | All frames pass. |
| | 0 | X | 0 | 0 | X | Pass when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x. |
| | 0 | 1 | 0 | 0 | X | Pass when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x. |
| | 0 | X | 1 | 0 | X | Fail when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x. |
| | 0 | 1 | 1 | 0 | X | Fail when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x. |

The filtering parameters in the MAC Frame Filter Register described in Table 24.4-1 are as follows.

**Parameter name:**

PM: Pass All Multicast
PF: Perfect Filter
DAIF: Destination Address Inverse Filtering
PAM: Pass All Multicast
DB: Disable Broadcast Frames

**Parameter setting:**

1: Set
0: Cleared

**Table 24.4-2. Source Address Filtering**

| Frame Type | PM | SAIF | SAF | Source Address Filter Operation |
|---|---|---|---|---|
| Unicast | 1 | X | X | Pass all frames |
| | 0 | 0 | 0 | Pass when results of perfect/group filtering match. Frames not passed are not discarded. |
| | 0 | 1 | 0 | Fail when results of perfect/group filtering match. Frames not passed are not discarded. |
| | 0 | 0 | 1 | Pass when results of perfect/group filtering match. Frames not passed are discarded. |
| | 0 | 1 | 1 | Fail when results of perfect/group filtering match. Frames not passed are discarded. |

The filtering parameters in the MAC Frame Filter Register described in Table 24.4-2 are as follows.

**Parameter name:**

PM: Pass All Multicast
SAF: Source Address Filtering
SAIF: Source Address Inverse Filtering

**Parameter setting:**

1: Set
0: Cleared
X: Don't care

### 24.4.6 Good Transmitted Frames and Received Frames

A frame 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
- No carrier or loss of carrier
- Late collision
- Frame underflow
- Excessive deferral
- Excessive collision

The received frames are considered "good frames", if there are not any of the following errors:

- CRC error
- Runt frames (frames shorter than 64 bytes)
- Alignment error (in 10/100 Mbps modes only)
- Length error (non-type frames only)
- Frame size over the maximum size (for non-type frames over the maximum frame size only)

- MII_RXER input error

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

## 24.5 EMAC_MTL (MAC Transaction Layer)

The MAC Transaction Layer provides FIFO memory to buffer and regulates 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.

## 24.6 PHY Interface

The DMA and the Host driver communicate through two data structures:

- Control and Status Registers (CSR)
- Descriptor lists and data buffers

For details please refer to Register Summary and Linked List Descriptors.

### 24.6.1 MII (Media Independent Interface)

Media Independent Interface (MII) defines the interconnection between MAC sublayers and PHYs at the data transmission rate of 10 Mbit/s and 100 Mbit/s.

#### 24.6.1.1 Interface Signals Between MII and PHY

Interface signals between MII and PHY are shown in Figure 24.6-1.

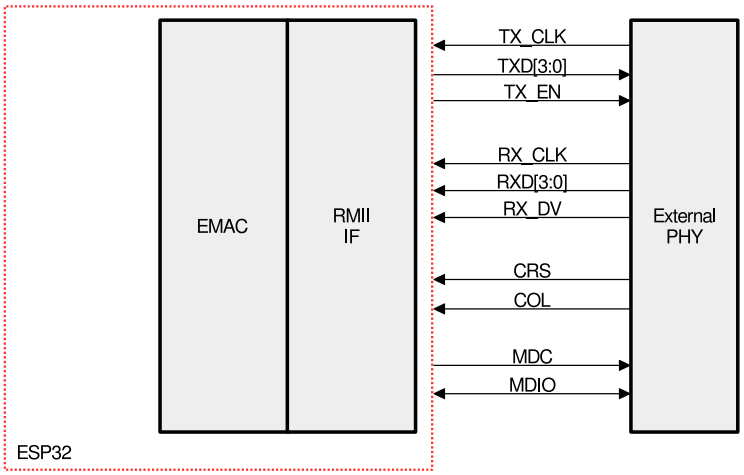



Figure 24.6-1. MII Interface

MII Interface Signal Description:

- MII_TX_CLK: TX clock signal. This signal provides the reference timing for TX data transmission. The frequencies are divided into two types: 2.5 MHz at a data transmission rate of 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_TXD[3:0]: Transmit data signal in groups of four, syn-driven by the MAC sub-layer, and valid only when the MII_TX_EN signal is valid. MII_TXD[0] is the lowest significant bit and MII_TXD[3] is the highest significant bit. When the signal MII_TX_EN is pulled low, sending data does not have any effect on the PHY.
- MII_TX_EN: Transmit data enable signal. This signal indicates that the MAC is currently sending nibbles (4 bits) for the MII. This signal must be synchronized with the first nibble of the header (MII_TX_CLK) and must be synchronized when all nibbles to be transmitted are sent to the MII.
- MII_RX_CLK: RX clock signal. This signal provides the reference timing for RX data transmission. The frequencies are divided into two types: 2.5 MHz at the data transmission rate of 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_RXD[3:0]: Receive data signal in groups of four, syn-driven by the PHY, and valid only when MII_RX_DV signal is valid. MII_RXD[0] is the lowest significant bit and MII_RXD[3] is the highest significant bit. When MII_RX_DV is disabled and MII_RX_ER is enabled, the specific MII_RXD[3:0] value represents specific information from the PHY.
- MII_RX_DV: Receive data valid signal. This signal indicates that the PHY is currently receiving the recovered and decoded nibble that will be transmitted to the MII. This signal must be synchronized with the first nibble of the recovered frame (MII_RX_CLK) and remain synchronized till the last nibble of the recovered frame. This signal must be disabled before the first clock cycle following the last nibble. In order to receive the frame correctly, the MII_RX_DV signal must cover the frame to be received over the time range, starting no later than when the SFD field appears.
- MII_CRS: Carrier sense signal. When the transmitting or receiving medium is in the non-idle state, the signal is enabled by the PHY. When the transmitting or receiving medium is in the idle state, the signal is disabled by the PHY. The PHY must ensure that the MII_CRS signal remains valid under conflicting conditions. This signal does not need to be synchronized with the TX and RX clocks. In full-duplex mode, this signal is insignificant.
- MII_COL: Collision detection signal. After a collision is detected on the medium, the PHY must immediately enable the collision detection signal, and the collision detection signal must remain active as long as a condition for collision exists. This signal does not need to be synchronized with the TX and RX clocks. In full-duplex mode, this signal is meaningless.
- MII_RX_ER: Receive error signal. The signal must remain for one or more cycles (MII_RX_CLK) to indicate to the MAC sublayer that an error has been detected somewhere in the frame.
- MDIO and MDC: Management Data Input/Output and Management Data Clock. 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.

#### 24.6.1.2 MII Clock

In MII mode, there are two directions of clock, Tx and Rx clocks in the interface between MII and the PHY. MII_TX_CLK is used to synchronize the TX data, and MII_RX_CLK is used to synchronize the RX data. The MII_RX_CLK clock is provided by the PHY. The MII_TX_CLK is provided by the chip's internal PLL or external

crystal oscillator. For details regarding Figure 24.6-2, please refer to the clock-related registers in Register Summary.

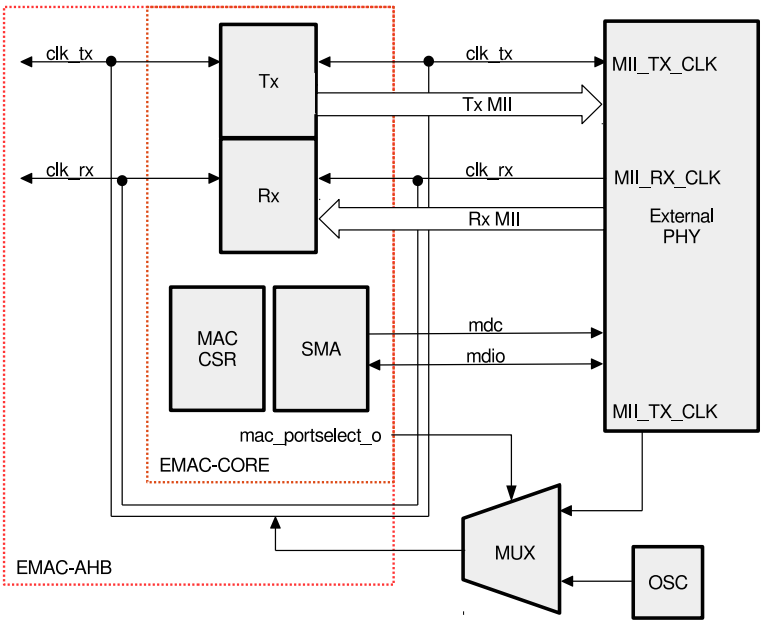



Figure 24.6-2. MII Clock

## 24.6.2 RMII (Reduced Media-Independent Interface)

RMII interface signals are shown in figure 24.6-3.

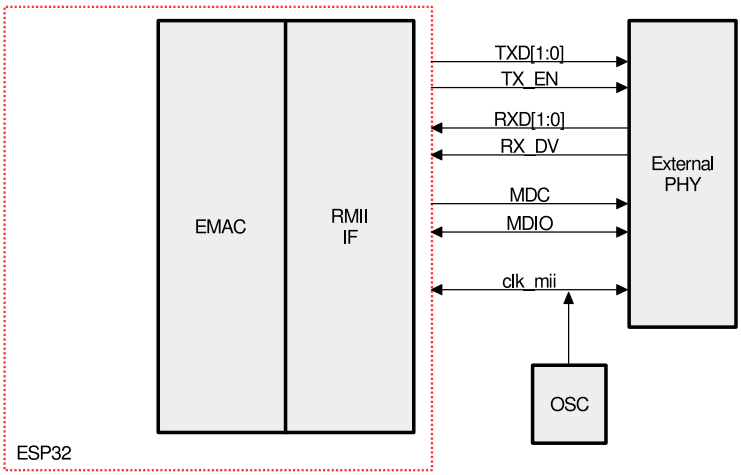



Figure 24.6-3. RMII Interface

### 24.6.2.1 RMII Interface Signal Description

The Reduced Media-Independent Interface (RMII) specification reduces the number of pins between the microcontroller's external peripherals and the external PHY at a data transmission rate of 10 Mbit/s or 100 Mbit/s. According to the IEEE 802.3u standard, MII includes 16 pins that contain data and control signals. The RMII specification reduces 62.5% of the pins to the number of seven.

RMII has the following features:

- Support for an operating rate of 10 Mbit/s or 100 Mbit/s
- The reference clock frequency must be 50 MHz.
- The same reference clock must be provided to the MAC and the external Ethernet PHY. The PHY provides independent 2-bit-wide TX and RX data paths.
- **Note:** If Wi-Fi and Ethernet are used simultaneously, the RMII clock cannot be generated by the internal APLL clock, as it would result in clock instability. In this case, please use an external PHY or external clock source to provide the reference clock.

#### 24.6.2.2 RMII Clock

The configuration of the RMII clock is as figure 24.6-4 shows.

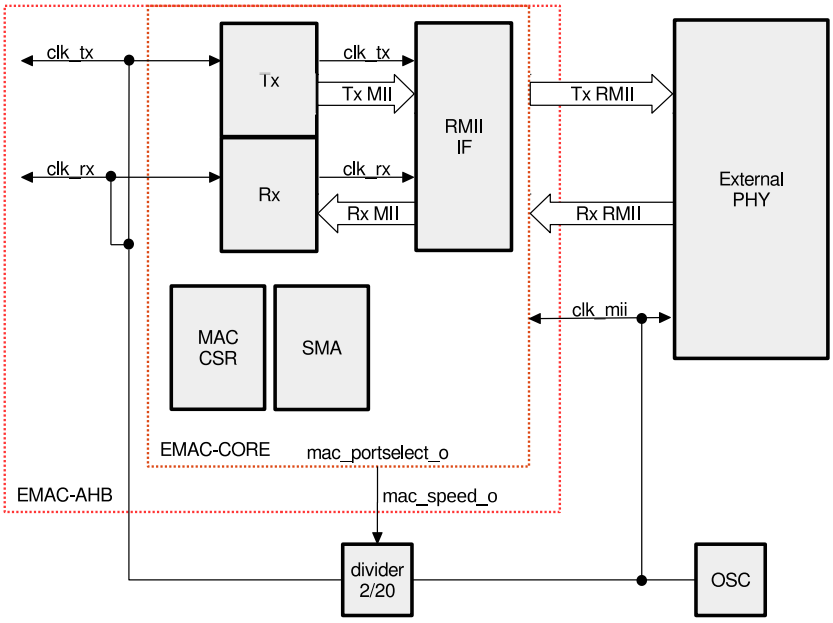



Figure 24.6-4. RMII Clock

### 24.6.3 Station Management Agent (SMA) Interface

As Figure 24.6-2 shows, the MAC uses MDC and MDIO signals to transfer control and data information to the PHY. The maximum clock frequency is 2.5 MHz. The clock is generated from the application clock by a clock divider. The PHY transmits register data during a write/read operation through the MDIO. This signal is driven synchronously to the MDC clock.

Please refer to Register Summary for details about the EMII Address Register and the EMII Data Register.

### 24.6.4 RMII Timing

This section describes the RMII timing specifications.

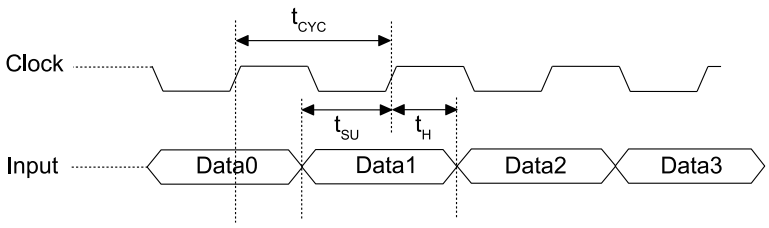



Figure 24.6-5. RMII Timing - Receiving Data

Table 24.6-1. Timing Parameters - Receiving Data

| Timing Parameters | Description | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $t_{CYC}$ | Clock cycle | 20 | 20 | 20 | ns |
| $t_{SU}$ | Setup time | 4 | – | – | ns |
| $t_H$ | Hold time | 1 | – | – | ns |
| $t_{ID}$ | Input delay | 3 | 5 | 8 | ns |

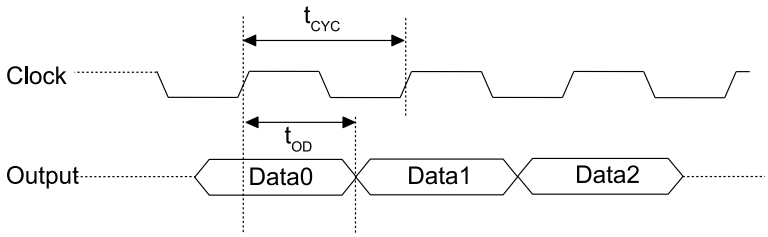



Figure 24.6-6. RMII Timing – Transmitting Data

Table 24.6-2. Timing Parameters – Transmitting Data

| Timing Parameters | Description | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $t_{CYC}$ | Clock cycle | 20 | 20 | 20 | ns |
| $t_{SU}$ | Setup time | 4 | – | – | ns |
| $t_H$ | Hold time | 1 | – | – | ns |
| $t_{OD}$ | Output delay | 6 | 9 | 12 | ns |

## 24.7 Ethernet DMA Features

The DMA has 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 transmission, 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 when errors occur.

# 24.8 Linked List Descriptors

This section shows the structure of the linked lists and the descriptors. Every linked list consists of eight words.

## 24.8.1 Transmit Descriptors

The structure of the transmitter linked lists is shown in Figure 24.8-1. Table 24.8-1 to Table 24.8-4 show the description of the linked lists.

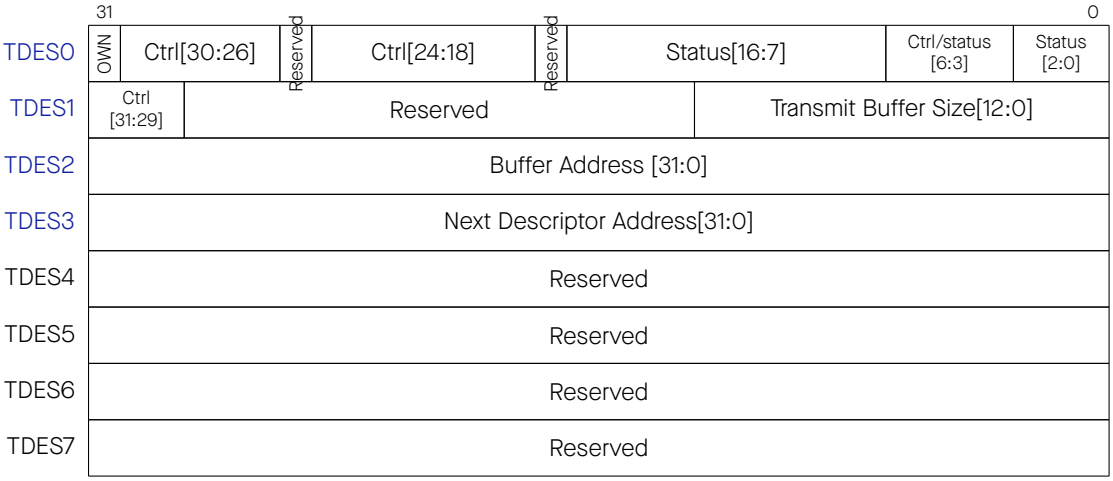



**Figure 24.8-1. Transmit Descriptor**

**Table 24.8-1. 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 this bit is 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 (Register 5[0]) 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. When this bit is set, the TBS1 or TBS2 field in TDES1 should have a non-zero value. |
| [28] | FS: First Segment | When set, this bit indicates that the buffer contains the first segment of a frame. |

| Bits | Name | Description |
|---|---|---|
| [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 Pad | When 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] | Reserved | Reserved |
| [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. The following list describes the bit encoding:<br>• 2'b00: Checksum insertion is disabled.<br>• 2'b01: Only IP header checksum calculation and insertion are enabled.<br>• 2'b10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware.<br>• 2'b11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.<br>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 should 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:<br>• 2'b00: Do not add a VLAN tag.<br>• 2'b01: Remove the VLAN tag from the frames before transmission. This option should be used only with the VLAN frames.<br>• 2'b10: Insert a VLAN tag with the tag value programmed in VLAN Tag Inclusion or Replacement Register.<br>• 2'b1: 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] | Reserved | Reserved |
| [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:<br>• TDES0[14]: Jabber Timeout<br>• TDES0[13]: Frame Flush<br>• TDES0[11]: Loss of Carrier<br>• TDES0[10]: No Carrier<br>• TDES0[9]: Late Collision<br>• TDES0[8]: Excessive Collision<br>• TDES0[2]: Excessive Deferral<br>• TDES0[1]: Underflow Error<br>• TDES0[16]: IP Header Error<br>• TDES0[12]: IP Payload Error |
| [14] | JT: Jabber Timeout | When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit is only set when EMACCONFIG_REG's bit EMACJABBER 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 MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload.<br>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 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 is aborted because of a collision occurring after the collision window (64 byte-times including Preamble in MII mode, 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 bit EMACRETRY 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 is a VLAN-type frame. |
| [6:3] | Ctrl/status | These status bits indicate the number of collisions that occurred before the frame was transmitted. This count is not valid when the Excessive Collisions bit (TDES0[8]) is set. The core 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 bit EMACDEFERRAL 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 Bit[5] in Transmit Underflow Register (Status Register) and Bit[0] in Transmit Interrupt Register (Status Register). |
| [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 24.8-2. 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 the MAC Address 0 register. 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]:<br>• 2'b00: Do not include the source address.<br>• 2'b01: Include or insert the source address. For reliable transmission, the application must provide frames without source addresses.<br>• 2'b10: Replace the source address. For reliable transmission, the application must provide frames with source addresses.<br>• 2'b11: Reserved<br>These bits are valid when the First Segment control bit (TDES0[28]) is set. |
| [28:16] | Reserved | Reserved |
| [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. |

Table 24.8-3. Transmit Descriptor 2 (TDES2)

| Bits | Name | Description |
|---|---|---|
| [31:0] | Buffer 1 Address Pointer | These bits indicate the physical address of Buffer 1. |

Table 24.8-4. Transmit Descriptor 3 (TDES3)

| Bits | Name | Description |
|---|---|---|
| [31:0] | Next Descriptor Address | This address contains the pointer to the physical memory where the Next Descriptor is present. |

## 24.8.2 Receive Descriptors

The structure of the receiver linked lists is shown in Figure 24.8-2. Table 24.8-5 to Table 24.8-9 provide the description of the linked lists.

| | 31 | | | | 0 |
|---|---|---|---|---|---|
| RDES0 | OWN | Status[30:0] | | | |
| RDES1 | Ctrl | Reserved[30:16] | Ctrl [15:14] | Res | Receive Buffer 1 Size[12:0] |
| RDES2 | Buffer1 Address [31:0] | | | | |
| RDES3 | Next Descriptor Address[31:0] | | | | |
| RDES4 | Extended Status[31:0] | | | | |
| RDES5 | Reserved | | | | |
| RDES6 | Reserved | | | | |
| RDES7 | Reserved | | | | |

**Figure 24.8-2. Receive Descriptor**

**Table 24.8-5. Receive Descriptor 0 (RDES0)**

| Bits | Name | Description |
|---|---|---|
| [31] | OWN: Own Bit | When set, this bit indicates that the descriptor is owned by the DMA of the DWC_gmac. When this bit is 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. |
| [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 (RDES0[8]) is set and either the Descriptor Error (RDES0[14]) or Overflow Error bits is 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:<br>• RDES0[1]: CRC Error<br>• RDES0[3]: Receive Error<br>• RDES0[4]: Watchdog Timeout<br>• RDES0[6]: Late Collision<br>• RDES0[7]: Giant Frame<br>• RDES4[4:3]: IP Header or Payload Error<br>• RDES0[11]: Overflow Error<br>• RDES0[14]: Descriptor Error<br>This field is valid only when the Last Descriptor (RDES0[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 the Last Descriptor (RDES0[8]) is set. |

| Bits | Name | Description |
|---|---|---|
| [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 (RDES0[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 to which this descriptor is pointing is a VLAN frame tagged by the MAC. The VLAN tagging depends on checking the VLAN fields of the received frame based on the Register (VLAN Tag Register) settings. |
| [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. |
| [7] | IP Checksum Error (Type1), or Giant Frame | When IP Checksum Engine (Type 1) is selected, this bit, if set, indicates one of the following:<br>• The 16-bit IPv4 header checksum calculated by the core did not match the received checksum bytes.<br>• The header checksum checking is bypassed for non-IPv4 frames.<br>Otherwise, this bit, when set, indicates the Giant Frame Status. Giant frames are larger than 1,518 bytes (or 1,522 bytes for VLAN or 2,000 bytes when Bit[27] of the MAC Configuration register is set), normal frames and larger-than-9,018-byte (9,022-byte for VLAN) frames when Jumbo Frame processing is enabled. |
| [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). When this bit is 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. |

| Bits | Name | Description |
|---|---|---|
| [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. |
| [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.<br>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.<br>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 24.8-6. Receive Descriptor 1 (RDES1)**

| Bits | Name | Description |
|---|---|---|
| [31] | Ctrl | When set, this bit prevents setting the Status Register's RI bit (CSR5[6]) 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] | Reserved | Reserved |
| [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 | Indicates the first data buffer size 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 24.8-7. Receive Descriptor 2 (RDES2)

| Bits | Name | Description |
|---|---|---|
| [31:0] | Buffer 1 Address Pointer | These bits indicate the physical address of Buffer 1. |

Table 24.8-8. Receive Descriptor 3 (RDES3)

| Bits | Name | Description |
|---|---|---|
| [31:0] | Next Descriptor Address | This address contains the pointer to the physical memory where the Next Descriptor is present. |

Table 24.8-9. 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] | Reserved | Reserved |
| [13] | Reserved | Reserved |
| [12] | Reserved | Reserved |
| [11:8] | Message Type | These bits are encoded to give the type of the message received.<br>• 3'b0000: Reserved<br>• 3'b0001: SYNC (all clock types)<br>• 3'b0010: Follow_Up (all clock types)<br>• 3'b0011: Delay_Req (all clock types)<br>• 3'b0100: Delay_Resp (all clock types)<br>• 3'b0101: Pdelay_Req (in peer-to-peer transparent clock)<br>• 3'b0110: Pdelay_Resp (in peer-to-peer transparent clock)<br>• 3'b0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock)<br>• 3'b1000: Announce<br>• 3'b1001: Management<br>• 3'b1010: Signaling<br>• 3'b1011-3'b1110: Reserved<br>• 3'b1111: Reserved |
| [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 Register (MAC Configuration Register) is set. |

| Bits | Name | Description |
|---|---|---|
| [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 Register (MAC Configuration Register) 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 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 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. |
| [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 2'b00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP.<br>• 3'b000: Unknown or did not process IP payload<br>• 3'b001: UDP<br>• 3'b010: TCP<br>• 3'b011: ICMP<br>• 3'b1xx: Reserved<br>This bit is valid when either Bit[7] or Bit[6] is set. |

## 24.9 Register Summary

Note that specific fields or bits of a given register may have different access attributes. Below is the list of all attributes together with the abbreviations used in register descriptions.

- Read Only (RO)
- Write Only (WO)
- Read and Write (R/W)
- Read, Write, and Self Clear (R/W/SC)
- Read, Self Set, and Write Clear (R/SS/WC)
- Read, Write Set, and Self Clear (R/WS/SC)
- Read, Self Set, and Self Clear or Write Clear (R/SS/SC/WC)
- Read Only and Write Trigger (RO/WT)

- Read, Self Set, and Read Clear (R/SS/RC)
- Read, Write, and Self Update (R/W/SU)
- Latched-low (LL)
- Latched-high (LH)

| Name | Description | Address | Access |
|---|---|---|---|
| **DMA configuration and control registers** | | | |
| DMABUSMODE_REG | Bus mode configuration | 0x3FF69000 | R/WS/SC |
| DMATXPOLLDEMAND_REG | Pull demand for data transmit | 0x3FF69004 | RO/WT |
| DMARXPOLLDEMAND_REG | Pull demand for data receive | 0x3FF69008 | RO/WT |
| DMARXBASEADDR_REG | Base address of the first receive descriptor | 0x3FF6900C | R/W |
| DMATXBASEADDR_REG | Base address of the first transmit descriptor | 0x3FF69010 | R/W |
| DMASTATUS_REG | State of interrupts, errors and other events | 0x3FF69014 | R/SS/WC |
| DMAOPERATION_MODE_REG | Receive and Transmit operating modes and command | 0x3FF69018 | R/SS/WC |
| DMAIN_EN_REG | Enable / disable interrupts | 0x3FF6901C | R/W |
| DMAMISSEDFR_REG | Missed Frame and Buffer Overflow Counter Register | 0x3FF69020 | R/W |
| DMARINTWDTIMER_REG | Watchdog timer count on receive | 0x3FF69024 | R/W |
| DMATXCURRDESC_REG | Pointer to current transmit descriptor | 0x3FF69048 | RO |
| DMARXCURRDESC_REG | Pointer to current receive descriptor | 0x3FF6904C | RO |
| DMATXCURRADDR_BUF_REG | Pointer to current transmit buffer | 0x3FF69050 | RO |
| DMARXCURRADDR_BUF_REG | Pointer to current receive buffer | 0x3FF69054 | RO |
| **MAC configuration and control registers** | | | |
| EMACCONFIG_REG | MAC configuration | 0x3FF6A000 | R/W |
| EMACFF_REG | Frame filter settings | 0x3FF6A004 | R/W |
| EMACGMIIADDR_REG | PHY configuration access | 0x3FF6A010 | R/WS/SC |
| EMACMIIDATA_REG | PHY data read write | 0x3FF6A014 | R/W |
| EMACFC_REG | frame flow control | 0x3FF6A018 | R/WS/SC(FCB)<br>R/W(BPA) |
| EMACDEBUG_REG | Status debugging bits | 0x3FF6A024 | RO |
| PMT_RWUFFR_REG | Remote Wake-Up Frame Filter | 0x3FF6A028 | RO |
| PMT_CSR_REG | PMT Control and Status | 0x3FF6A02C | RO |
| EMACLPI_CSR_REG | LPI Control and Status | 0x3FF6A030 | RO |
| EMACLPITIMERSCONTROL_REG | LPI Timers Control | 0x3FF6A034 | RO |
| EMACINTS_REG | Interrupt status | 0x3FF6A038 | RO |
| EMACINTMASK_REG | Interrupt mask | 0x3FF6A03C | R/W |
| EMACADDROHIGH_REG | Upper 16 bits of the first 6-byte MAC address | 0x3FF6A040 | R/W |
| EMACADDROLOW_REG | Lower 32 bits of the first 6-byte MAC address | 0x3FF6A044 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| EMACADDR1HIGH_REG | MAC address filtering and upper 16 bits of the second 6-byte MAC address | 0x3FF6A048 | R/W |
| EMACADDR1LOW_REG | Lower 32 bits of the second 6-byte MAC address | 0x3FF6A04C | R/W |
| EMACADDR2HIGH_REG | MAC address filtering and upper 16 bits of the third 6-byte MAC address | 0x3FF6A050 | R/W |
| EMACADDR2LOW_REG | Lower 32 bits of the third 6-byte MAC address | 0x3FF6A054 | R/W |
| EMACADDR3HIGH_REG | MAC address filtering and upper 16 bits of the fourth 6-byte MAC address | 0x3FF6A058 | R/W |
| EMACADDR3LOW_REG | Lower 32 bits of the fourth 6-byte MAC address | 0x3FF6A05C | R/W |
| EMACADDR4HIGH_REG | MAC address filtering and upper 16 bits of the fifth 6-byte MAC address | 0x3FF6A060 | R/W |
| EMACADDR4LOW_REG | Lower 32 bits of the fifth 6-byte MAC address | 0x3FF6A064 | R/W |
| EMACADDR5HIGH_REG | MAC address filtering and upper 16 bits of the sixth 6-byte MAC address | 0x3FF6A068 | R/W |
| EMACADDR5LOW_REG | Lower 32 bits of the sixth 6-byte MAC address | 0x3FF6A06C | R/W |
| EMACADDR6HIGH_REG | MAC address filtering and upper 16 bits of the seventh 6-byte MAC address | 0x3FF6A070 | R/W |
| EMACADDR6LOW_REG | Lower 32 bits of the seventh 6-byte MAC address | 0x3FF6A074 | R/W |
| EMACADDR7HIGH_REG | MAC address filtering and upper 16 bits of the eighth 6-byte MAC address | 0x3FF6A078 | R/W |
| EMACADDR7LOW_REG | Lower 32 bits of the eighth 6-byte MAC address | 0x3FF6A07C | R/W |
| EMACWDOGTO_REG | Watchdog timeout control | 0x3FF6A0DC | R/W |
| **Clock configuration registers** | | | |
| EMAC_EX_CLKOUT_CONF_REG | RMII clock divider setting | 0x3FF69800 | R/W |
| EMAC_EX_OSCCLK_CONF_REG | RMII clock half and whole divider settings | 0x3FF69804 | R/W |
| EMAC_EX_CLK_CTRL_REG | Clock enable and external / internal clock selection | 0x3FF69808 | R/W |
| **PHY type and SRAM configuration registers** | | | |
| EMAC_EX_PHYINF_CONF_REG | Selection of MII / RMII phy | 0x3FF6980C | R/W |
| EMAC_PD_SEL_REG | Ethernet RAM power-down enable | 0x3FF69810 | R/W |

## 24.10 Registers

The addresses in this section are relative to the EMAC base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 24.9 *Register Summary*.

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

**Note: The value of all reset registers must be set to the reset value.**

**Register 24.1. DMABUSMODE_REG (0x0000)**

| (reserved) | DMAMIXEDBURST | DMAADDRALIBEA | PBLX8_MODE | USE_SEP_PBL | RX_DMA_PBL | FIXED_BURST | PRI_RATIO | PROG_BURST_LEN | ALT_DESC_SIZE | DESC_SKIP_LEN | DMA_ARB_SCH | SW_RST | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 | 23 | 22 17 | 16 | 15 14 | 13 8 | 7 | 6 2 | 1 | 0 | |
| 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0x01 | 0 | 0x0 | 0x01 | 0 | 0x00 | 0 | 1 | Reset |

**DMAMIXEDBURST** When this bit is set high and the FB(FIXES_BURST) bit is low, the AHB master interface starts all bursts of a 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** When this bit is set high and the FB bit is 1, the AHB interface generates all bursts aligned to the start address LS bits. If the FB bit is 0, the first burst (accessing the start address of data buffer) is not aligned, but subsequent bursts are aligned to the address. (R/W)

**PBLX8_MODE** When set high, this bit multiplies the programmed PBL(PROG_BURST_LEN) value (Bits[22:17] and Bits[13:8]) eight times. 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** When set high, this bit configures the Rx DMA to use the value configured in Bits[22:17] as PBL. The PBL value in Bits[13:8] is applicable only to the Tx DMA operations. When reset to low, the PBL value in Bits[13:8] is applicable for both DMA engines. (R/W)

**RX_DMA_PBL** This field indicates the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read or Write.The Rx DMA always attempts to burst as specified in the RPBL(RX_DMA_PBL) bit each time it starts a burst transfer on the host bus. You can program RPBL with values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. This field is valid and applicable only when USP(USE_SEP_PBL) is set high. (R/W)

**FIXED_BURST** This bit controls whether the AHB master interface performs fixed burst transfers or not. When set, the AHB interface uses only SINGLE, INCR4, INCR8, or INCR16 during start of the normal burst transfers. When reset, the AHB interface uses SINGLE and INCR burst transfer operations. (R/W)

**PRI_RATIO** These bits control the priority ratio in the weighted round-robin arbitration between the Rx DMA and Tx DMA. These bits are valid only when Bit 1 (DA) is reset. The priority ratio Rx:Tx represented by each bit: (R/W)

- 2'b00 — 1: 1
- 2'b01 — 2: 0
- 2'b10 — 3: 1
- 2'b11 — 4: 1

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

### Register 24.1. DMABUSMODE_REG (0x0000)

Continued from the previous page...

**PROG_BURST_LEN** These bits indicate 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; 2. Set the PBL. (R/W)

**ALT_DESC_SIZE** When set, the size of the alternate descriptor increases to 32 bytes. (R/W)

**DESC_SKIP_LEN** This bit specifies the number of Word to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When the DSL(DESC_SKIP_LEN) value is equal to zero, the descriptor table is taken as contiguous by the DMA in Ring mode. (R/W)

**DMA_ARB_SCH** This bit specifies the arbitration scheme between the transmit and receive paths. 1'b0: weighted round-robin with RX: TX or TX: RX, priority specified in PR (bit[15:14]); 1'b1 Fixed priority (Rx priority to Tx). (R/W)

**SW_RST** When this bit is set, the MAC DMA Controller resets the logic and all internal registers of the MAC. It is cleared automatically after the reset operation is complete in all of the ETH_MAC clock domains. Before reprogramming any register of the ETH_MAC, you should read a zero (0) value in this bit. (R/WS/SC)

### Register 24.2. DMATXPOLLDEMAND_REG (0x0004)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TRANS_POLL_DEMAND** When these bits are written with any value, the DMA reads the current descriptor to which the Register (Current Host Transmit Descriptor Register) is pointing. If that descriptor is not available (owned by the Host), the transmission returns to the suspend state and Bit[2] (TU) of Status Register is asserted. If the descriptor is available, the transmission resumes. (RO/WT)

### Register 24.3. DMARXPOLLDEMAND_REG (0x0008)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RECV_POLL_DEMAND** When these bits are written with any value, the DMA reads the current descriptor to which the Current Host Receive Descriptor Register is pointing. If that descriptor is not available (owned by the Host), the reception returns to the Suspended state and Bit[7] (RU) of Status Register is asserted. If the descriptor is available, the Rx DMA returns to the active state. (RO/WT)

### Register 24.4. DMARXBASEADDR_REG (0x000C)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**START_RECV_LIST** This field contains the base address of the first descriptor in the Receive Descriptor list. The LSB Bits[1:0] are ignored and internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only. (R/W)

### Register 24.5. DMATXBASEADDR_REG (0x0010)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**START_TRANS_LIST** This field contains the base address of the first descriptor in the Transmit Descriptor list. The LSB Bits[1:0] are ignored and are internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only. (R/W)

### Register 24.6. DMASTATUS_REG (0x0014)

| 31 30 | 29 | 28 | 27 26 | 25 23 | 22 20 | 19 17 | 16 | 15 | 14 | 13 | 12 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (Reserved) | EMAC_PMT_INT | (reserved) | ERROR_BITS | TRANS_PROC_STATE | RECV_PROC_STATE | NORM_INT_SUMM | ABN_INT_SUMM | EARLY_RECV_INT | FATAL_BUS_ERR_INT | (reserved) | EARLY_TRANS_INT | RECV_WDT_TO | RECV_PROC_STOP | RECV_BUF_UNAVAIL | RECV_INT | TRANS_UNDFLOW | RECV_OVFLOW | TRANS_JABBER_TO | TRANS_BUF_UNAVAIL | TRANS_PROC_STOP | TRANS_INT | |
| 0 0 | 0 | 0 | 0 0 | 0x0 | 0x0 | 0x0 | 0 | 0 | 0 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**EMAC_PMT_INT** This bit indicates an interrupt event in the PMT module of the ETH_MAC. The software must read the PMT Control and Status Register in the MAC to get the exact cause of interrupt and clear its source to reset this bit to 1'b0. (RO)

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

**Register 24.6. DMASTATUS_REG (0x0014)**

**Continued from the previous page...**

**ERROR_BITS** This field indicates the type of error that caused a Bus Error, for example, error response on the AHB interface. This field is valid only when Bit[13] (FBI) is set. This field does not generate an interrupt. (RO)

- 3'b000: Error during Rx DMA Write Data Transfer.
- 3'b011: Error during Tx DMA Read Data Transfer.
- 3'b100: Error during Rx DMA Descriptor Write Access.
- 3'b101: Error during Tx DMA Descriptor Write Access.
- 3'b110: Error during Rx DMA Descriptor Read Access.
- 3'b111: Error during Tx DMA Descriptor Read Access.

**TRANS_PROC_STATE** This field indicates the Transmit DMA FSM state. This field does not generate an interrupt. (RO)

- 3'b000: Stopped. Reset or Stop Transmit Command issued.
- 3'b001: Running. Fetching Transmit Transfer Descriptor.
- 3'b010: Reserved for future use.
- 3'b011: Running. Waiting for TX packets.
- 3'b100: Suspended. Receive Descriptor Unavailable.
- 3'b101: Running. Closing Transmit Descriptor.
- 3'b110: Reserved.
- 3'b111: Running. Transferring the TX packets data from transmit buffer to host memory.

**RECV_PROC_STATE** This field indicates the Receive DMA FSM state. This field does not generate an interrupt. (RO)

- 3'b000: Stopped. Reset or Stop Receive Command issued.
- 3'b001: Running. Fetching Receive Transfer Descriptor.
- 3'b010: Reserved for future use.
- 3'b011: Running. Waiting for RX packets.
- 3'b100: Suspended. Receive Descriptor Unavailable.
- 3'b101: Running. Closing Receive Descriptor.
- 3'b110: Reserved.
- 3'b111: Running. Transferring the TX packets data from receive buffer to host memory.

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

**Register 24.6. DMASTATUS_REG (0x0014)**

**Continued from the previous page...**

**NORM_INT_SUMM** Normal Interrupt Summary bit value is the logical OR of the following bits when the corresponding interrupt bits are enabled in Interrupt Enable Register:(R/SS/WC)

- Bit[0]: Transmit Interrupt.
- Bit[2]: Transmit Buffer Unavailable.
- Bit[6]: Receive Interrupt.
- Bit[14]: Early Receive Interrupt. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes NIS to be set, is cleared.

**ABN_INT_SUMM** Abnormal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register: (R/SS/WC)

- 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 bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes AIS to be set, is cleared.

**EARLY_RECV_INT** This bit indicates that the DMA filled the first data buffer of the packet. This bit is cleared when the software writes 1 to this bit or when Bit[6] (RI) of this register is set (whichever occurs earlier). (R/SS/WC)

**FATAL_BUS_ERR_INT** This bit indicates that a bus error occurred, as described in Bits [25:23]. When this bit is set, the corresponding DMA engine disables all of its bus accesses. (R/SS/WC)

**EARLY_TRANS_INT** This bit indicates that the frame to be transmitted is fully transferred to the MTL Transmit FIFO. (R/SS/WC)

**RECV_WDT_TO** When set, this bit indicates that the Receive Watchdog Timer expired while receiving the current frame and the current frame is truncated after the watchdog timeout. (R/SS/WC)

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

**Register 24.6. DMASTATUS_REG (0x0014)**

**Continued from the previous page...**

**RECV_PROC_STOP** This bit is asserted when the Receive Process enters the Stopped state. (R/SS/WC)

**RECV_BUF_UNAVAIL** This bit indicates that 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. This bit is set only when the previous Receive Descriptor is owned by the DMA. (R/SS/WC)

**RECV_INT** This bit indicates that the frame reception is complete. When reception is complete, the Bit[31] of RDES1 (Disable Interrupt on Completion) is reset in the last Descriptor, and the specific frame status information is updated in the descriptor. The reception remains in the Running state. (R/SS/WC)

**TRANS_UNDFLOW** This bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set. (R/SS/WC)

**RECV_OVFLOW** This bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to the application, the overflow status is set in RDES0[11]. (R/SS/WC)

**TRANS_JABBER_TO** This bit indicates that the Transmit Jabber Timer expired, which happens when the frame size exceeds 2,048 (10,240 bytes when the Jumbo frame is enabled). 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** This bit indicates that 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** This bit is set when the transmission is stopped. (R/SS/WC)

**TRANS_INT** This bit indicates that the frame transmission is complete. When transmission is complete, Bit[31] (OWN) of TDES0 is reset, and the specific frame status information is updated in the descriptor. (R/SS/WC)

**Register 24.7. DMAOPERATION_MODE_REG (0x0018)**

| (reserved) | DIS_DROP_TCPIP_ERR_FRAM | RX_STORE_FORWARD | DIS_FLUSH_RECV_FRAMES | (reserved) | (reserved) | TX_STR_FWD | FLUSH_TX_FIFO | (reserved) | TX_THRESH_CTRL | START_STOP_TRANSMISSION_COMMAND | (reserved) | (reserved) | (reserved) | FWD_ERR_FRAME | FWD_UNDER_GF | DROP_GFRM | RX_THRESH_CTRL | OPT_SECOND_FRAME | START_STOP_RX | (reserved) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 17 | 16 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 | Reset |

**DIS_DROP_TCPIP_ERR_FRAM** When this bit is set, the MAC does not drop the frames which only have errors detected by the Receive Checksum engine.When this bit is reset, all error frames are dropped if the Fwd_Err_Frame bit is reset. (R/W)

**RX_STORE_FORWARD** When this bit is set, the MTL reads a frame from the Rx FIFO only after the complete frame has been written to it. (R/W)

**DIS_FLUSH_RECV_FRAMES** When this bit is set, the Rx DMA does not flush any frames because of the unavailability of receive descriptors or buffers. (R/W)

**TX_STR_FWD** When this bit is set, transmission starts when a full frame resides in the MTL Transmit FIFO. When this bit is set, the TX_THRESH_CTRL values specified in TX_THRESH_CTRL are ignored. (R/W)

**FLUSH_TX_FIFO** When this bit is set, the transmit FIFO controller logic is reset to its default values and thus all data in the Tx FIFO is lost or flushed. This bit is cleared internally when the flushing operation is complete. (R/WS/SC)

**TX_THRESH_CTRL** These bits control the threshold level of the MTL Transmit FIFO. Transmission starts when the frame size within the MTL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when TX_STR_FWD is reset. 3'b000: 64, 3'b001: 128, 3'b010: 192, 3'b011: 256, 3'b100: 40, 3'b101: 32, 3'b110: 24, 3'b111: 16. (R/W)

**START_STOP_TRANSMISSION_COMMAND** When this bit is set, transmission is placed in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. (R/W)

**FWD_ERR_FRAME** When this bit is reset, the Rx FIFO drops frames with error status (CRC error, collision error, giant frame, watchdog timeout, or overflow). (R/W)

**FWD_UNDER_GF** When set, the Rx FIFO forwards Undersized frames (that is, frames with no Error and length less than 64 bytes) including pad-bytes and CRC.

**DROP_GFRM** When set, the MAC drops the received giant frames in the Rx FIFO, that is, frames that are larger than the computed giant frame limit. (R/W)

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

Register 24.7. DMAOPERATION_MODE_REG (0x0018)

Continued from the previous page...

**RX_THRESH_CTRL** These two bits control the threshold level of the MTL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MTL Receive FIFO is larger than the threshold. 2'b00: 64; 2'b01: 32; 2'b10: 96; 2'b11: 128. (R/W)

**OPT_SECOND_FRAME** When this bit is set, it instructs the DMA to process the second frame of the Transmit data even before the status for the first frame is obtained. (R/W)

**START_STOP_RX** When this bit is set, the Receive process is placed in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes the incoming frames.When this bit is cleared, the Rx DMA operation is stopped after the transfer of the current frame. (R/W)

**Register 24.8. DMAIN_EN_REG (0x001C)**

| (reserved) | DMAIN_NISE | DMAIN_AISE | DMAIN_ERIE | DMAIN_FBEE | (reserved) | DMAIN_ETIE | DMAIN_RWTE | DMAIN_RSE | DMAIN_RBUE | DMAIN_RIE | DMAIN_UIE | DMAIN_OIE | DMAIN_TJTE | DMAIN_TBUE | DMAIN_TSE | DMAIN_TIE | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**DMAIN_NISE** When this bit is set, normal interrupt summary is enabled. When this bit is reset, normal interrupt summary is disabled. This bit enables the following interrupts in Status Register: (R/W)

- Bit[0]: Transmit Interrupt.
- Bit[2]: Transmit Buffer Unavailable.
- Bit[6]: Receive Interrupt.
- Bit[14]: Early Receive Interrupt.

**DMAIN_AISE** When this bit is set, abnormal interrupt summary is enabled. When this bit is reset, the abnormal interrupt summary is disabled. This bit enables the following interrupts in Status Register:(R/W)

- Bit[1]: Transmit Process Stopped.
- Bit[3]: Transmit Jabber Timeout.
- Bit[4]: Receive 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.

**DMAIN_ERIE** When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Early Receive Interrupt is enabled. When this bit is reset, the Early Receive Interrupt is disabled. (R/W)

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

**Register 24.8. DMAIN_EN_REG (0x001C)**

**Continued from the previous page...**

**DMAIN_FBEE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Fatal Bus Error Interrupt is enabled. When this bit is reset, the Fatal Bus Error Enable Interrupt is disabled. (R/W)

**DMAIN_ETIE** When this bit is set with an Abnormal Interrupt Summary Enable (Bit[15]), the Early Transmit Interrupt is enabled. When this bit is reset, the Early Transmit Interrupt is disabled. (R/W)

**DMAIN_RWTE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Watchdog Timeout Interrupt is enabled. When this bit is reset, the Receive Watchdog Timeout Interrupt is disabled. (R/W)

**DMAIN_RSE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Stopped Interrupt is enabled. When this bit is reset, the Receive Stopped Interrupt is disabled. (R/W)

**DMAIN_RBUE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Buffer Unavailable Interrupt is enabled. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled. (R/W)

**DMAIN_RIE** When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Receive Interrupt is enabled. When this bit is reset, the Receive Interrupt is disabled. (R/W)

**DMAIN_UIE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmit Underflow Interrupt is enabled. When this bit is reset, the Underflow Interrupt is disabled. (R/W)

**DMAIN_OIE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Overflow Interrupt is enabled. When this bit is reset, the Overflow Interrupt is disabled. (R/W)

**DMAIN_TJTE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmit Jabber Timeout Interrupt is enabled. When this bit is reset, the Transmit Jabber Timeout Interrupt is disabled. (R/W)

**DMAIN_TBUE** When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Transmit Buffer Unavailable Interrupt is enabled. When this bit is reset, the Transmit Buffer Unavailable Interrupt is disabled. (R/W)

**DMAIN_TSE** When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmission Stopped Interrupt is enabled. When this bit is reset, the Transmission Stopped Interrupt is disabled. (R/W)

**DMAIN_TIE** When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Transmit Interrupt is enabled. When this bit is reset, the Transmit Interrupt is disabled. (R/W)

**Register 24.9. DMAMISSEDFR_REG (0x0020)**

| (reserved) | Overflow_BFOC | Overflow_FC | Overflow_BMFC | Missed_FC | |
|---|---|---|---|---|---|
| 31 29 | 28 | 27 17 | 16 | 15 0 | |
| 0 | 0x0 | 0x0 | 0x0 | 0x0 | Reset |

**Overflow_BFOC** This bit is set every time the Overflow Frame Counter (Bits[27:17]) 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 and this bit indicates that the rollover happened. (R/SS/RC)

**Overflow_FC** This field indicates the number of frames missed by the application. This counter is incremented each time the MTL FIFO overflows. The counter is cleared when this register is read. (R/SS/RC)

**Overflow_BMFC** This bit is set every time Missed Frame Counter (Bits[15:0]) overflows, that is, the DMA discards an incoming frame because of 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 and this bit indicates that the rollover happened. (R/SS/RC)

**Missed_FC** This field indicates 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. The counter is cleared when this register is read. (R/SS/RC)

**Register 24.10. DMARINTWDTIMER_REG (0x0024)**

| (reserved) | RIWTC | |
|---|---|---|
| 31 8 | 7 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x000 | Reset |

**RIWTC** This bit indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. 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 automatic setting of RI as per RDES1[31] of any received frame. (R/W)

### Register 24.11. DMATXCURRDESC_REG (0x0048)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**TRANS_DSCR_ADDR_PTR** The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

### Register 24.12. DMARXCURRDESC_REG (0x004C)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**RECV_DSCR_ADDR_PTR** The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

### Register 24.13. DMATXCURRADDR_BUF_REG (0x0050)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**TRANS_BUFF_ADDR_PTR** The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

### Register 24.14. DMARXCURRADDR_BUF_REG (0x0054)

| 31 … 0 |
|---|
| 0x000000000 |

Reset

**RECV_BUFF_ADDR_PTR** The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

**Register 24.15. EMACCONFIG_REG (0x1000)**

| (reserved) | SAIRC | ASS2KP | (reserved) | EMACWATCHDOG | EMACJABBER | (reserved) | EMACJUMBOFRAME | EMACINTERFRAMEGAP | EMACDISABLECRS | EMACMII | EMACFESPEED | EMACRXOWN | EMACLOOPBACK | EMACDUPLEX | EMACRXIPCOFFLOAD | EMACRETRY | (reserved) | EMACPADCRCSTRIP | EMACBACKOFFLIMIT | EMACDEFERRALCHECK | EMACTX | EMACRX | PLTF | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 28 | 27 | 26 24 | 23 | 22 | 21 | 20 | 19 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 5 | 4 | 3 | 2 | 1 0 | |
| 0 | 0x0 | 0 | 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x0 | 0 | 0 | 0 | 0x0 | Reset |

**SAIRC** This field controls 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]: (R/W)

- 2'b0x: The input signals mti_sa_ctrl_i and ati_sa_ctrl_i control the SA field generation.
- 2'b10: If Bit[30] is set to 0, the MAC inserts the content of the MAC Address 0 registers in the SA field of all transmitted frames. If Bit[30] is set to 1 the MAC inserts the content of the MAC Address 1 registers in the SA field of all transmitted frames.
- 2'b11: If Bit[30] is set to 0, the MAC replaces the content of the MAC Address 0 registers in the SA field of all transmitted frames. If Bit[30] is set to 1, the MAC replaces the content of the MAC Address 1 registers in the SA field of all transmitted frames.

**ASS2KP** When set, the MAC considers all frames, with up to 2,000 bytes length, as normal packets. When Bit[20] (JE) is not set, the MAC considers all received frames of size more than 2K bytes as Giant frames. When this bit is reset and Bit[20] (JE) is not set, the MAC considers all received frames of size more than 1,518 bytes (1,522 bytes for tagged) as Giant frames. When Bit[20] is set, setting this bit has no effect on Giant Frame status. (R/W)

**EMACWATCHDOG** When this bit is set, the MAC disables the watchdog timer on the receiver. The MAC can receive frames of up to 16,383 bytes. When this bit is reset, the MAC does not allow a receive frame which more than 2,048 bytes (10,240 if JE is set high) or the value programmed in Register (Watchdog Timeout Register). The MAC cuts off any bytes received after the watchdog limit number of bytes. (R/W)

**EMACJABBER** When this bit is set, the MAC disables the jabber timer on the transmitter. The MAC can transfer frames of up to 16,383 bytes. When this bit is reset, 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. (R/W)

**EMACJUMBOFRAME** When this bit is set, 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. (R/W)

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

**Register 24.15. EMACCONFIG_REG (0x1000)**

**Continued from the previous page...**

**EMACINTERFRAMEGAP** These bits control the minimum IFG between frames during transmission. (R/W)

- 3'b000: 96 bit times.
- 3'b001: 88 bit times.
- 3'b010: 80 bit times.
- 3'b111: 40 bit times. In the half-duplex mode, the minimum IFG can be configured only for 64 bit times (IFG = 100). Lower values are not considered.

**EMACDISABLECRS** When set high, this bit makes the MAC transmitter ignore 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. When this bit is low, the MAC transmitter generates such errors because of Carrier Sense and can even abort the transmissions. (R/W)

**EMACMII** This bit selects the Ethernet line speed. It should be set to 1 for 10 or 100 Mbps operations. In 10 or 100 Mbps operations, this bit, along with FES(EMACFESPEED) bit, it selects the exact linespeed. In the 10/100 Mbps-only operations, the bit is always 1. (R/W)

**EMACFESPEED** This bit selects the speed in the MII, RMII interface. 0: 10 Mbps; 1: 100 Mbps. (R/W)

**EMACRXOWN** When this bit is set, the MAC disables the reception of frames when the TX_EN is asserted in the half-duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in the full-duplex mode. (R/W)

**EMACLOOPBACK** When this bit is set, the MAC operates in the loopback mode MII. 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** When this bit is set, the MAC operates in the full-duplex mode where it can transmit and receive simultaneously. This bit is read only with default value of 1'b1 in the full-duplex-mode. (R/W)

**EMACRXIPCOFFLOAD** When this bit is set, 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). When this bit is reset, this function is disabled. (R/W)

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

**Register 24.15. EMACCONFIG_REG (0x1000)**

**Continued from the previous page...**

**EMACRETRY** When this bit is set, the MAC attempts only one transmission. 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. When this bit is reset, the MAC attempts retries based on the settings of the BL field (Bits [6:5]). This bit is applicable only in the half-duplex mode. (R/W)

**EMACPADCRCSTRIP** When this bit is set, 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. When this bit is reset, the MAC passes all incoming frames, without modifying them, to the Host. (R/W)

**EMACBACKOFFLIMIT** The Back-Off limit determines the random integer number (r) of slot time delays (512 bit times for 10/100 Mbps) for which the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only in the half-duplex mode.

- 00: k= min (n, 10).
- 01: k = min (n, 8).
- 10: k = min (n, 4).
- 11: k = min (n, 1), n = retransmission attempt. The random integer r takes the value in the range 0 ~ 2000.

**EMACDEFERRALCHECK** Deferral Check. (R/W)

**EMACTX** When this bit is set, the transmit state machine of the MAC is enabled for transmission on the MII. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and does not transmit any further frames. (R/W)

**EMACRX** When this bit is set, the receiver state machine of the MAC is enabled for receiving frames from the MII. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames from the MII. (R/W)

**PLTF** These bits control the number of preamble bytes that are added to the beginning of every Transmit frame. The preamble reduction occurs only when the MAC is operating in the full-duplex mode. 2'b00: 7 bytes of preamble. 2'b01: 5 bytes of preamble. 2'b10: 3 bytes of preamble. (R/W)

**Register 24.16. EMACFF_REG (0x1004)**

| RECEIVE_ALL | (reserved) | SAFE | SAIF | PCF | DBF | PAM | DAIF | (reserved) | PMODE | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 … 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 | 0x0 | 0 | 0 | 0 | 0 0 | 0 | Reset |

**RECEIVE_ALL** When this bit is set, 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. When this bit is reset, the Receiver module passes only those frames to the Application that pass the SA or DA address filter. (R/W)

**SAFE** When this bit is set, 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. When this bit is reset, the MAC forwards the received frame to the application with updated SAF bit of the Rx Status depending on the SA address comparison. (R/W)

**SAIF** When this bit is set, 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. When this bit is reset, frames whose SA does not match the SA registers are marked as failing the SA Address filter. (R/W)

**PCF** These bits control the forwarding of all control frames (including unicast and multicast Pause frames). (R/W)

- 2'b00: MAC filters all control frames from reaching the application.
- 2'b01: MAC forwards all control frames except Pause frames to application even if they fail the Address filter.
- 2'b10: MAC forwards all control frames to application even if they fail the Address Filter.
- 2'b11: MAC forwards control frames that pass the Address Filter.

The following conditions should be true for the Pause frames processing:

- Condition 1: The MAC is in the full-duplex mode and flow control is enabled by setting Bit 2 (RFE) of Register (Flow Control Register) to 1.
- Condition 2: The destination address (DA) of the received frame matches the special multicast address or the MAC Address 0 when Bit 3 (UP) of the Register(Flow Control Register) is set.
- Condition 3: The Type field of the received frame is 0x8808 and the OPCODE field is 0x0001.

**DBF** When this bit is set, the AFM(Address Filtering Module) module blocks all incoming broadcast frames. In addition, it overrides all other filter settings. When this bit is reset, the AFM module passes all received broadcast frames. (R/W)

**PAM** When set, this bit indicates that all received frames with a multicast destination address (first bit in the destination address field is '1') are passed. (R/W)

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

### Register 24.16. EMACFF_REG (0x1004)

**Continued from the previous page...**

**DAIF** When this bit is set, the Address Check block operates in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When reset, normal filtering of frames is performed. (R/W)

**PMODE** When this bit is set, the Address Filter module passes all incoming frames irrespective of the destination or source address. The SA or DA Filter Fails status bits of the Receive Status Word are always cleared when PR(PRT_RATIO) is set. (R/W)

### Register 24.17. EMACGMIIADDR_REG (0x1010)

| (reserved) | MIIDEV | MIIREG | MIICSRCLK | MIIWRITE | MIIBUSY | |
|---|---|---|---|---|---|---|
| 31 … 16 | 15 … 11 | 10 … 6 | 5 … 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x00 | 0x00 | 0x00 | 0 | 0 | Reset |

**MIIDEV** This field indicates which of the 32 possible PHY devices are being accessed. (R/W)

**MIIREG** This field selects the desired MII register in the selected PHY device. (R/W)

**MIICSRCLK** This field selects the APB clock frequency. It has the following two values. Other values are reserved.

- 4'b0000: The APB clock frequency is 80 MHz. The MDC clock frequency is APB_CLK/42.
- 4'b0011: The APB clock frequency is 40 MHz. The MDC clock frequency is APB_CLK/26. (R/W)

**MIIWRITE** When set, this field indicates to the PHY that this is a Write operation using MII_DATA. If this field is not set, it indicates that this is a Read operation, that is, placing the data in MII_DATA. (R/W)

**MIIBUSY** This field is used in combination with MIIREG and MII_DATA.

Before writing to MIIREG and MII_DATA, this field should read logic 0 (idle state by default).

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 MAC does not receive ACK from PHY during a read or write access to MIIREG and MII_DATA. (R/WS/SC)

**Register 24.18. EMACMIIDATA_REG (0x1014)**

| (reserved) | | | | | | | | | | | | | | | | MII_DATA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | 16 | 15 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x00000 | Reset |

**MII_DATA** This field contains 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 24.19. EMACFC_REG (0x1018)**

| PAUSE_TIME | (reserved) | PLT | UPFD | RFCE | TFCE | FCBBA | |
|---|---|---|---|---|---|---|---|
| 31 … 16 | 15 … 6 | 5 … 4 | 3 | 2 | 1 | 0 | |
| 0x00000 | 0 0 0 0 0 0 0 0 0 0 | 0x0 | 0 | 0 | 0 | 0 | Reset |

**PAUSE_TIME** This field holds 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 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)

**PLT** This field configures the threshold of the Pause timer automatic retransmission of the Pause frame. 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 at 228 (256-28) slot times after the first Pause frame is transmitted. The following list provides the threshold values for different values: (R/W)

- 2'b00: The threshold is Pause time minus 4 slot times (PT-4 slot times).
- 2'b01: The threshold is Pause time minus 28 slot times (PT-28 slot times).
- 2'b10: The threshold is Pause time minus 144 slot times (PT-144 slot times).
- 2'b11: The threshold is Pause time minus 256 slot times (PT-256 slot times). The slot time is defined as the time taken to transmit 512 bits (64 bytes) on the MII interface.

**UPFD** A pause frame is processed when it has the unique multicast address specified in the IEEE Std 802.3. When this bit is set, 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. When this bit is reset, the MAC only detects Pause frames with unique multicast address. (R/W)

**RFCE** When this bit is set, the MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time. When this bit is reset, the decode function of the Pause frame is disabled. (R/W)

**TFCE** In the full-duplex mode, when this bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In the half-duplex mode, when this bit is set, the MAC enables the backpressure operation. When this bit is reset, the backpressure feature is disabled. (R/W)

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

**Register 24.19. EMACFC_REG (0x1018)**

**Continued from the previous page...**

**FCBBA** This bit initiates a Pause frame in the full-duplex mode and activates the backpressure function in the half-duplex mode if the TFCE bit is set. In the full-duplex mode, this bit should be read as 1'b0 before writing to the Flow Control register. To initiate a Pause frame, the Application must set this bit to 1'b1. 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 frame transmission, the MAC resets this bit to 1'b0. The Flow Control register should not be written to until this bit is cleared. 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/WS/SC)(FCB)/(R/W)(BPA(backpressure activate))

**Register 24.20. EMACDEBUG_REG (0x1024)**

| (reserved) | MTLTSFFS | MTLTFNES | (reserved) | MTLTFWCS | MTLTFRCS | MACTP | MACTFCS | MACTPES | (reserved) | MTLRFFLS | (reserved) | MTLRFRCS | MTLRFWCAS | (reserved) | MACRFFCS | MACRPES | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 | 24 | 23 | 22 | 21 20 | 19 | 18 17 | 16 | 15 10 | 9 8 | 7 | 6 5 | 4 | 3 | 2 1 | 0 | |
| 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0x0 | 0 | 0x0 | 0 | 0 0 0 0 0 0 | 0x0 | 0 | 0x0 | 0 | 0 | 0x0 | 0 | Reset |

**MTLTSFFS** When high, this bit indicates that the MTL TxStatus FIFO is full. Therefore, the MTL cannot accept any more frames for transmission. (RO)

**MTLTFNES** When high, this bit indicates that the MTL Tx FIFO is not empty and some data is left for transmission. (RO)

**MTLTFWCS** When high, this bit indicates that the MTL Tx FIFO Write Controller is active and is transferring data to the Tx FIFO. (RO)

**MTLTFRCS** This field indicates the state of the Tx FIFO Read Controller: (RO)

- 2'b00: IDLE state.
- 2'b01: READ state (transferring data to the MAC transmitter).
- 2'b10: Waiting for TxStatus from the MAC transmitter.
- 2'b11: Writing the received TxStatus or flushing the Tx FIFO.

**MACTP** When high, this bit indicates that the MAC transmitter is in the Pause condition (in the full-duplex-mode) and hence does not schedule any frame for transmission. (RO)

**MACTFCS** This field indicates the state of the MAC Transmit Frame Controller module: (RO)

- 2'b00: IDLE state.
- 2'b01: Waiting for status of previous frame or IFG or backoff period to be over.
- 2'b10: Generating and transmitting a Pause frame (in the full-duplex mode).
- 2'b11: Transferring input frame for transmission.

**MACTPES** When high, this bit indicates that the MAC MII transmit protocol engine is actively transmitting data and is not in the IDLE state. (RO)

**MTLRFFLS** This field gives the status of the fill-level of the Rx FIFO: (RO)

- 2'b00: Rx FIFO Empty.
- 2'b01: Rx FIFO fill-level below flow-control deactivate threshold.
- 2'b10: Rx FIFO fill-level above flow-control activate threshold.
- 2'b11: Rx FIFO Full.

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

**Register 24.20. EMACDEBUG_REG (0x1024)**

**Continued from the previous page...**

**MTLRFRCS** This field gives the state of the Rx FIFO read Controller: (RO)

2'b00: IDLE state.

2'b01: Reading frame data.

2'b10: Reserved.

2'b11: Flushing the frame data and status.

**MTLRFWCAS** When high, this bit indicates that the MTL Rx FIFO Write Controller is active and is transferring a received frame to the FIFO. (RO)

**MACRFFCS** When high, this field indicates the active state of the FIFO Read and Write controllers of the MAC Receive Frame Controller Module. MACRFFCS[1] represents the status of FIFO Read controller. MACRFFCS[0] represents the status of small FIFO Write controller. (RO)

**MACRPES** When high, this bit indicates that the MAC MII receive protocol engine is actively receiving data and not in IDLE state. (RO)

**Register 24.21. PMT_RWUFFR_REG (0x1028)**

| PMT_RWUFFR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**WKUPPKTFILTER** The MSB (31st bit) must be zero. Bit j[30:0] is the byte mask. If Bit 1/2/3/4 (byte number) of the byte mask is set, the CRC block processes the filter 0/1/2/3 Offset + j of the incoming packet (RWKPTR is 0/1/2/3). (R/W)

- RWKPTR is 0:Filter 0 Byte Mask;
- RWKPTR is 1: Filter 1 Byte Mask;
- RWKPTR is 2: Filter 2 Byte Mask;
- RWKPTR is 3: Filter 3 Byte Mask;
- RWKPTR is 4: Bit 3/11/19/27 specifies the address type, defining the destination address type of the pattern. When the bit is set, the pattern applies to only multicast packets; when the bit is reset, the pattern applies only to unicast packet for filter 0/1/2/3. Bit 0/8/16/24 is the enable bit for filter 0/1/2/3;
- RWKPTR is 5: This filter 0/1/2/3 offset register defines the offset (within the packet) from which the filter 0/1/2/3 examines the packets;
- RWKPTR is 6: This filter 0 (bit[15:0])/1 (bit[31:16]) CRC16 register contains the CRC16 value calculated from the pattern and also the byte mask programmed to the wake-up filter register block; The polynomial:

  $G(x) = x^{16} + x^{15} + x^{2} + 1$.
- RWKPTR is 7: This filter 2 bit[15:0])/3(bit[31:16]) CRC16 register contains the CRC16 value calculated from the pattern and also the byte mask programmed to the wake-up filter register block. The polynomial:

  $G(x) = x^{16} + x^{15} + x^{2} + 1$.

**Register 24.22. PMT_CSR_REG (0x102C)**

| RWKFILTRST | (reserved) | RWKPTR | (reserved) | GLBLUCAST | (reserved) | RWKPRCVD | MGKPRCVD | (reserved) | RWKPKTEN | MGKPKTEN | PWRDWN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 29 | 28 24 | 23 10 | 9 | 8 7 | 6 | 5 | 4 3 | 2 | 1 | 0 | |
| 0 | 0 0 | 0 0 0 0 0 | 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 |

**RWKFILTRST** When this bit is set, it resets the remote RWKPTR register to 3'b000. (R/WS/SC)

**RWKPTR** The maximum value of the pointer is 7 ,the detail information ,please refer to PMT_RWUFFR. (RO)

**GLBLUCAST** When set, enables any unicast packet filtered by the MAC (DAFilter) address recognition to be a remote wake-up frame. (R/W)

**RWKPRCVD** When set, this bit indicates the power management event is generated because of the reception of a remote wake-up frame. This bit is cleared by a Read into this register. (R/SS/RC)

**MGKPRCVD** When set, this bit indicates that the power management event is generated because of the reception of a magic packet. This bit is cleared by a Read into this register. (R/SS/RC)

**RWKPKTEN** hen set, enables generation of a power management event because of remote wake-up frame reception. (R/W)

**MGKPKTEN** When set, enables generation of a power management event because of magic packet reception. (R/W)

**PWRDWN** hen set, the MAC receiver drops all received frames until it receives the expected magic packet or remote wake-up frame. This bit must only be set when MGKPKTEN, GLBLUCAST, or RWKPKTEN bit is set high. (R/WS/SC)

**Register 24.23. EMACLPI_CSR_REG (0x1030)**

| 31 | | | | | | | | | | 20 | 19 | 18 | 17 | 16 | 15 | | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | LPITXA | (reserved) | PLS | LPIEN | (reserved) | | | | | | | RLPIST | TLPIST | (reserved) | | | | RLPIEX | RLPIEN | TLPIEX | TLPIEN | |
| 0 | 0 | 0 | 0 | 0 | 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 |

**LPITXA** This bit controls the behavior of the MAC when it is entering or coming out of the LPI mode on the transmit side.If the LPITXA and LPIEN bits are set to 1, the MAC enters the LPI mode only after all outstanding frames and pending frames have been transmitted. The MAC comes out of the LPI mode when the application sends any frame.When this bit is 0, the LPIEN bit directly controls behavior of the MAC when it is entering or coming out of the LPI mode. (R/W)

**PLS** This bit indicates the link status of the PHY. When set, the link is considered to be okay (up) and when reset, the link is considered to be down. (R/W)

**LPIEN** When set, this bit instructs the MAC Transmitter to enter the LPI state. When reset, this bit instructs the MAC to exit the LPI state and resume normal transmission.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** When set, this bit indicates that the MAC is receiving the LPI pattern on the MII interface. (R/W)

**TLPIST** When set, this bit indicates that the MAC is receiving the LPI pattern on the MII interface. (R/W)

**RLPIEX** When set, this bit indicates that the MAC Receiver has stopped receiving the LPI pattern on the MII interface, exited the LPI state, and resumed the normal reception. This bit is cleared by a read into this register. (R/SS/RC)

**RLPIEN** When set, this bit indicates that the MAC Receiver has received an LPI pattern and entered the LPI state. This bit is cleared by a read into this register. (R/SS/RC)

**TLPIEX** When set, this bit indicates that the MAC transmitter has exited the LPI state after the user has cleared the LPIEN bit and the LPI_TW_Timer has expired.This bit is cleared by a read into this register. (R/SS/RC)

**TLPIEN** When set, this bit indicates that the MAC Transmitter has entered the LPI state because of the setting of the LPIEN bit. This bit is cleared by a read into this register. (R/SS/RC)

**Register 24.24. EMACLPITIMERSCONTROL_REG (0x1034)**

| (reserved) | LPI_LS_TIMER | LPI_TW_TIMER | |
|---|---|---|---|
| 31 26 | 25 16 | 15 0 | |
| 0 0 0 0 0 0 | 0 x 3 E 8 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**LPI_LS_TIMER** This field specifies the minimum time (in milliseconds) for which the link status from the PHY should be up (OKAY) before the LPI pattern can be transmitted to the PHY. The MAC does not transmit the LPI pattern even when the LPIEN bit is set unless the LPI_LS_Timer reaches the programmed terminal count. The default value of the LPI_LS_Timer is 1000 (1 sec) as defined in the IEEE standard.(R/W)

**LPI_TW_TIMER** This field specifies the minimum time (in microseconds) for which the MAC waits after it stops transmitting the LPI pattern to the PHY and before it resumes the normal transmission. The TLPIEX status bit is set after the expiry of this timer.(R/W)

**Register 24.25. EMACINTS_REG (0x1038)**

| (reserved) | LPIINTS | (reserved) | (reserved) | PMTINTS | (reserved) | |
|---|---|---|---|---|---|---|
| 31 11 | 10 | 9 | 8 4 | 3 | 2 0 | |
| 0 0 0 0 0 0 0 0 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 |

**LPIINTS** When the Energy Efficient Ethernet feature is enabled, this bit is set for any LPI state entry or exit in the MAC Transmitter or Receiver. This bit is cleared on reading Bit[0] of Register (LPI Control and Status Register). (RO)

**PMTINTS** This bit is set when a magic packet or remote wake-up frame is received in the power-down mode (see Bit[5] and Bit[6] in the PMT Control and Status Register). This bit is cleared when both Bits[6:5] are cleared because of a read operation to the PMT Control and Status register. This bit is valid only when you select the optional PMT module during core configuration. (RO)

### Register 24.26. EMACINTMASK_REG (0x103C)

| (reserved) | LPIINTMASK | (reserved) | (reserved) | PMTINTMASK | (reserved) |
|---|---|---|---|---|---|
| 31 … 11 | 10 | 9 | 8 … 4 | 3 | 2 … 0 |
| 0 0 0 0 0 0 0 0 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

**LPIINTMASK** When set, this bit disables the assertion of the interrupt signal because of the setting of the LPI Interrupt Status bit in Register (Interrupt Status Register). (R/W)

**PMTINTMASK** When set, this bit disables the assertion of the interrupt signal because of the setting of PMT Interrupt Status bit in Register (Interrupt Status Register). (R/W)

### Register 24.27. EMACADDR0HIGH_REG (0x1040)

| ADDRESS_ENABLE0 | (reserved) | MAC_ADDRESS0_HI |
|---|---|---|
| 31 | 30 … 16 | 15 … 0 |
| 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0FFFF |

Reset

**ADDRESS_ENABLE0** This bit is always set to 1. (RO)

**MAC_ADDRESS0_HI** This field contains the upper 16 bits (47:32) 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 24.28. EMACADDR0LOW_REG (0x1044)

| 31 … 0 |
|---|
| 0xFFFFFFFF |

Reset

**EMACADDR0LOW_REG** This field contains the lower 32 bits of the first 6-byte MAC address. This is used by the MAC for filtering the received frames and inserting the MAC address in the Transmit Flow Control (Pause) Frames. (R/W)

**Register 24.29. EMACADDR1HIGH_REG (0x1048)**

| ADDRESS_ENABLE1 | SOURCE_ADDRESS | MASK_BYTE_CONTROL | (reserved) | MAC_ADDRESS1_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 24 | 23 16 | 15 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE1** When this bit is set, the address filter module uses the second MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS** When this bit is set, the EMACADDR1[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR1[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL** These bits are mask control bits for comparison of each of the EMACADDR1 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR1 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR1 High [15:8].
- Bit[28]: EMACADDR1 High [7:0].
- Bit[27]: EMACADDR1 Low [31:24].
- Bit[24]: EMACADDR1 Low [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** This field contains the upper 16 bits, Bits[47:32] of the second 6-byte MAC address. (R/W)

**Register 24.30. EMACADDR1LOW_REG (0x104C)**

| 31 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR1LOW_REG** This field contains the lower 32 bits of the second 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.31. EMACADDR2HIGH_REG (0x1050)**

| ADDRESS_ENABLE2 | SOURCE_ADDRESS2 | MASK_BYTE_CONTROL2 | (reserved) | MAC_ADDRESS2_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 24 | 23 16 | 15 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE2** When this bit is set, the address filter module uses the third MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS2** When this bit is set, the EMACADDR2[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR2[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL2** These bits are mask control bits for comparison of each of the EMACADDR2 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR2 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR2 High [15:8].
- Bit[28]: EMACADDR2 High [7:0].
- Bit[27]: EMACADDR2 Low [31:24].
- Bit[24]: EMACADDR2 Low [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** This field contains the upper 16 bits, Bits[47:32] of the third 6-byte MAC address. (R/W)

**Register 24.32. EMACADDR2LOW_REG (0x1054)**

| 31 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR2LOW_REG** This field contains the lower 32 bits of the third 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.33. EMACADDR3HIGH_REG (0x1058)**

| ADDRESS_ENABLE3 | SOURCE_ADDRESS3 | MASK_BYTE_CONTROL3 | (reserved) | MAC_ADDRESS3_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 24 | 23 16 | 15 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE3** When this bit is set, the address filter module uses the fourth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS3** When this bit is set, the EMACADDR3[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR3[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL3** These bits are mask control bits for comparison of each of the EMACADDR3 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR3 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR3 High [15:8].
- Bit[28]: EMACADDR3 High [7:0].
- Bit[27]: EMACADDR3 Low [31:24].
- Bit[24]: EMACADDR3 Low [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** This field contains the upper 16 bits, Bits[47:32] of the fourth 6-byte MAC address. (R/W)

**Register 24.34. EMACADDR3LOW_REG (0x105C)**

| 31 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR3LOW_REG** This field contains the lower 32 bits of the fourth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.35. EMACADDR4HIGH_REG (0x1060)**

| ADDRESS_ENABLE4 | SOURCE_ADDRESS4 | MASK_BYTE_CONTROL4 | (reserved) | MAC_ADDRESS4_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 24 | 23 16 | 15 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE4** When this bit is set, the address filter module uses the fifth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS4** When this bit is set, the EMACADDR4[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR4[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL4** These bits are mask control bits for comparison of each of the EMACADDR4 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR4 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR4 High [15:8].
- Bit[28]: EMACADDR4 High [7:0].
- Bit[27]: EMACADDR4 Low [31:24].
- Bit[24]: EMACADDR4 Low [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** This field contains the upper 16 bits, Bits[47:32] of the fifth 6-byte MAC address. (R/W)

**Register 24.36. EMACADDR4LOW_REG (0x1064)**

| 31 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR4LOW_REG** This field contains the lower 32 bits of the fifth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.37. EMACADDR5HIGH_REG (0x1068)**

| ADDRESS_ENABLE5 | SOURCE_ADDRESS5 | MASK_BYTE_CONTROL5 | (reserved) | MAC_ADDRESS5_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 … 24 | 23 … 16 | 15 … 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE5** When this bit is set, the address filter module uses the sixth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS5** When this bit is set, the EMACADDR5[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR5[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL5** These bits are mask control bits for comparison of each of the EMACADDR5 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR5 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR5 High [15:8].
- Bit[28]: EMACADDR5 High [7:0].
- Bit[27]: EMACADDR5 Low [31:24].
- Bit[24]: EMACADDR5 Low [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** This field contains the upper 16 bits, Bits[47:32] of the sixth 6-byte MAC address. (R/W)

**Register 24.38. EMACADDR5LOW_REG (0x106C)**

| 31 … 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR5LOW_REG** This field contains the lower 32 bits of the sixth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.39. EMACADDR6HIGH_REG (0x1070)**

| ADDRESS_ENABLE6 | SOURCE_ADDRESS6 | MASK_BYTE_CONTROL6 | (reserved) | MAC_ADDRESS6_HI | |
|---|---|---|---|---|---|
| 31 | 30 | 29 … 24 | 23 … 16 | 15 … 0 | |
| 0 | 0 | 0x00 | 0 0 0 0 0 0 0 0 | 0x0FFFF | Reset |

**ADDRESS_ENABLE6** When this bit is set, the address filter module uses the seventh MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS6** When this bit is set, the EMACADDR6[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR6[47:0] is used to compare with the DA fields of the received frame. (R/W)

**MASK_BYTE_CONTROL6** These bits are mask control bits for comparison of each of the EMACADDR6 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR6 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR6 High [15:8].
- Bit[28]: EMACADDR6 High [7:0].
- Bit[27]: EMACADDR6 Low [31:24].
- Bit[24]: EMACADDR6 Low [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)

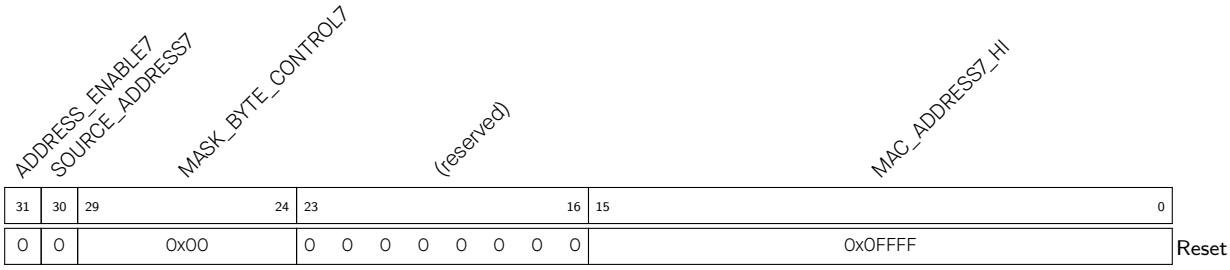
**MAC_ADDRESS6_HI** This field contains the upper 16 bits, Bits[47:32] of the seventh 6-byte MAC address. (R/W)

**Register 24.40. EMACADDR6LOW_REG (0x1074)**

| 31 … 0 | |
|---|---|
| 0x0FFFFFFFF | Reset |

**EMACADDR6LOW_REG** This field contains the lower 32 bits of the seventh 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

**Register 24.41. EMACADDR7HIGH_REG (0x1078)**




**ADDRESS_ENABLE7** When this bit is set, the address filter module uses the eighth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

**SOURCE_ADDRESS7** When this bit is set, the EMACADDR7[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR7[47:0] is used to compare with the DA fields of the received frame. (R/W)

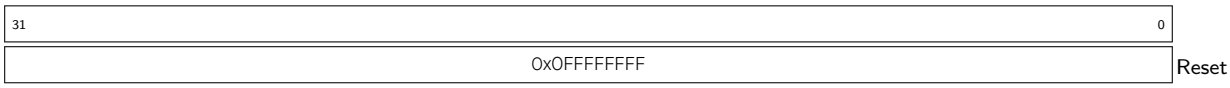
**MASK_BYTE_CONTROL7** These bits are mask control bits for comparison of each of the EMACADDR7 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR7 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR7 High [15:8].
- Bit[28]: EMACADDR7 High [7:0].
- Bit[27]: EMACADDR7 Low [31:24].
- Bit[24]: EMACADDR7 Low [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** This field contains the upper 16 bits, Bits[47:32] of the eighth 6-byte MAC address. (R/W)

**Register 24.42. EMACADDR7LOW_REG (0x107C)**




**EMACADDR7LOW_REG** This field contains the lower 32 bits of the eighth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

### Register 24.43. EMACWDOGTO_REG (0x10DC)

| (reserved) | PWDOGEN | (reserved) | WDOGTO | |
|---|---|---|---|---|
| 31 – 17 | 16 | 15 – 14 | 13 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 0 | 0x0000 | Reset |

**PWDOGEN** When this bit is set and Bit[23] (WD) of EMACCONFIG_REG is reset, the WTO field (Bits[13:0]) is used as watchdog timeout for a received frame. When this bit is cleared, the watchdog timeout for a received frame is controlled by the setting of Bit[23] (WD) and Bit[20] (JE) in EMACCONFIG_REG. (R/W)

**WDOGTO** When Bit[16] (PWE) is set and Bit[23] (WD) of EMACCONFIG_REG is reset, this field is used as watchdog timeout for a received frame. 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 24.44. EMAC_EX_CLKOUT_CONF_REG (0x0800)

| (reserved) | EMAC_CLK_OUT_H_DIV_NUM | EMAC_CLK_OUT_DIV_NUM | |
|---|---|---|---|
| 31 – 8 | 7 – 4 | 3 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x02 | 0x04 | Reset |

**EMAC_CLK_OUT_H_DIV_NUM** RMII CLK using internal APLL CLK, the half divider number, when using RMII PHY. (R/W)

**EMAC_CLK_OUT_DIV_NUM** RMII CLK using internal APLL CLK, the whole divider number, when using RMII PHY. (R/W)

**Register 24.45. EMAC_EX_OSCCLK_CONF_REG (0x0804)**

| (reserved) | EMAC_OSC_CLK_SEL | EMAC_OSC_H_DIV_NUM_100M | EMAC_OSC_DIV_NUM_100M | EMAC_OSC_H_DIV_NUM_10M | EMAC_OSC_DIV_NUM_10M | |
|---|---|---|---|---|---|---|
| 31 – 25 | 24 | 23 – 18 | 17 – 12 | 11 – 6 | 5 – 0 | |
| 0 0 0 0 0 0 0 | 0 | 0 | 1 | 9 | 19 | Reset |

**EMAC_OSC_CLK_SEL** Ethernet work using external PHY output clock or not for RMII CLK, when using RMII PHY. When this bit is set to 1, external PHY CLK is used. When this bit is set to 0, APLL CLK is used. (R/W)

**EMAC_OSC_H_DIV_NUM_100M** RMII/MII half-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 100M. (R/W)

**EMAC_OSC_DIV_NUM_100M** RMII/MII whole-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 100M. (R/W)

**EMAC_OSC_H_DIV_NUM_10M** RMII/MII half-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 10M. (R/W)

**EMAC_OSC_DIV_NUM_10M** RMII/MII whole-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 10M. (R/W)

**Register 24.46. EMAC_EX_CLK_CTRL_REG (0x0808)**

| (reserved) | (reserved) | EMAC_MII_CLK_RX_EN | EMAC_MII_CLK_TX_EN | (reserved) | EMAC_INT_OSC_EN | EMAC_EXT_OSC_EN | |
|---|---|---|---|---|---|---|---|
| 31 – 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 |

**EMAC_MII_CLK_RX_EN** Enable Ethernet RX CLK. (R/W)

**EMAC_MII_CLK_TX_EN** Enable Ethernet TX CLK. (R/W)

**EMAC_INT_OSC_EN** Using internal APLL CLK in RMII PHY mode. (R/W)

**EMAC_EXT_OSC_EN** Using external APLL CLK in RMII PHY mode. (R/W)

### Register 24.47. EMAC_EX_PHYINF_CONF_REG (0x080C)

| (reserved) | EMAC_PHY_INTF_SEL | (reserved) | |
|---|---|---|---|
| 31 … 16 | 15 … 13 | 12 … 0 | |
| 0 0 0 0 0 0 0 0 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 |

**EMAC_PHY_INTF_SEL** The PHY interface selected. 0x0: PHY MII, 0x4: PHY RMII. (R/W)

### Register 24.48. EMAC_PD_SEL_REG (0x0810)

| (reserved) | EMAC_RAM_PD_EN | |
|---|---|---|
| 31 … 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 | Reset |

**EMAC_RAM_PD_EN** Ethernet RAM power-down enable signal. Bit[0]: TX SRAM; Bit[1]: RX SRAM. Setting the bit to 1 powers down the RAM. (R/W)

# Chapter 25

# Two-Wire Automotive Interface (TWAI)

## 25.1 Overview

The Two-wire Automotive Interface (TWAI®) is a multi-master, multi-cast communication protocol with error detection and signaling and inbuilt message priorities and arbitration.The TWAI protocol is suited for automotive and industrial applications (Please see TWAI Protocol Description).

ESP32 contains a TWAI controller that can be connected to a TWAI bus via an external transceiver. The TWAI controller contains numerous advanced features, and can be utilized in a wide range of use cases such as automotive products, industrial automation controls, building automation etc.

## 25.2 Features

ESP32 TWAI controller supports the following features:

- compatible with ISO 11898-1 protocol (CAN Specification 2.0)
- Supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates:
  - from 25 Kbit/s to 1 Mbit/s in chip revision v0.0/v1.0/v1.1
  - from 12.5 Kbit/s to 1 Mbit/s in chip revision v3.0/v3.1
- Multiple modes of operation
  - Normal
  - Listen Only (no influence on bus)
  - Self Test (transmissions do not require acknowledgment)
- 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

## 25.3 Functional Protocol

### 25.3.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows for nodes to exchange messages in a latency bounded manner. A TWAI bus will has a following properties.

**Single Channel and Non-Return-to-Zero:** The bus consists of a single channel to carry bits, thus communication is half-duplex. Synchronization is also derived from this channel, thus 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 a Dominant state will always override a another node transmitting a Recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential wiring).

**Bit-Stuffing:** Certain fields of TWAI messages are bit-stuffed. A Transmitter that transmits five consecutive bits of the same value should automatically insert a complementary bit. Likewise, a Receiver that receives five consecutive bits should treat the next bit as a stuff bit. Bit stuffing is applied to the following fields: SOF, Arbitration Field, Control Field, Data Field, and CRC Sequence (see Section 25.3.2 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 25.3.3).

**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 beginning its own transmission.

**Message-Priorities 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 arbitration.

**Error Detection and Signaling:** Each node will actively monitor the bus for errors, and signal the detection errors by transmitting an Error Frame.

**Fault Confinement:** Each node will maintain a set of error counts that are incremented/decremented according to a set of rules. When the error counts surpass a certain threshold, a 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 within 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 originating a message is a Transmitter. The node remains a Transmitter until the bus is idle or until the node loses arbitration. Note that multiple nodes can be Transmitter if they have yet to lose arbitration.
- All nodes that are not Transmitters are Receivers.

## 25.3.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes. Messages are split into various frame types, and some frame types will have different frame formats. The TWAI protocol has of the following frame types:

- Data Frames
- Remote Frames
- Error Frames
- Overload Frames
- Interframe Space

The TWAI protocol has the following frame formats:

- Standard Frame Format (SFF) that consists of a 11-bit identifier
- Extended Frame Format (EFF) that consists of a 29-bit identifier

### 25.3.2.1 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 to nodes to request a Data Frame with the same Identifier from another node, thus does not contain any data bytes. However, Data Frames and Remote Frames share many common fields. Figure 25.3-1 illustrates the fields and sub fields of the different frames and formats.

**Arbitration Field**

When two or more nodes transmits a Data or Remote Frame simultaneously, the Arbitration Field is used to determine which node will win arbitration of the bus. During the Arbitration Field, if a node transmits a Recessive bit but observes a Dominant bit, this 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 become a Receiver.

The Arbitration Field primarily consists of the Frame Identifier that is transmitted 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 will always win arbitration.
- Given the same ID and format, Data Frames will always prevail over RTR Frames.
- Given the same first 11 bits of ID, a Standard Format Data Frame will prevail over an Extended Format Data Frame due to the SRR being recessive.

**Control Field**

The control field primarily consists of the DLC (Data Length Code) 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 most significant bit first.

**Data Field**

The Data Field contains the actual payload data bytes of a Data Frame. Remote Frames do not contain a Data Field.

**CRC Field**

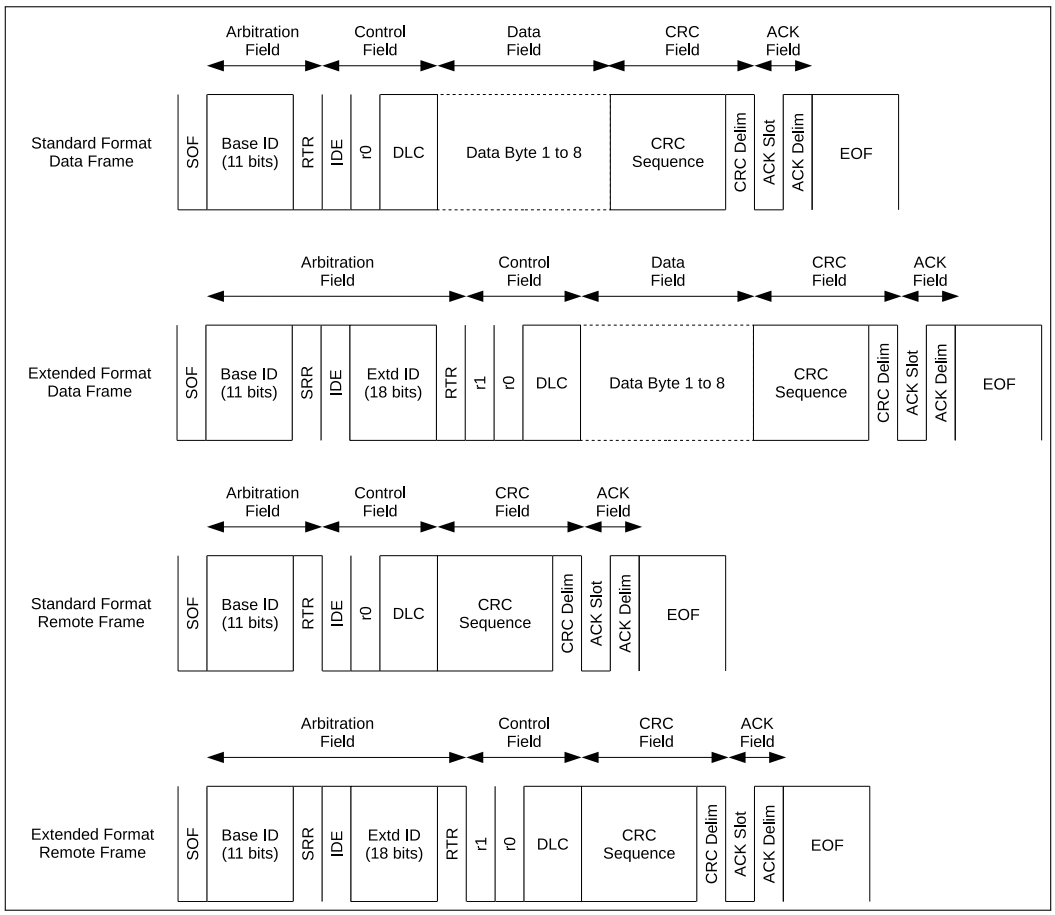



Figure 25.3-1. The bit fields of Data Frames and Remote Frames

The CRC Field primarily consists of a a CRC Sequence. The CRC Sequence is a 15-bit cyclic redundancy code calculated form 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 is mainly intended for the receiver to send a message to a transmitter, indicating it has received an effective message.

Table 25.3-1. 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 given they have the same ID. |
| SRR | The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF. |

| Data/Remote Frames | Description |
|---|---|
| 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 given 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 (reserved bit 1) is always Dominant. |
| r0 | The r0 (reserved bit 0) is always Dominant. |
| DLC | The DLC (Data Length Code) is 4-bits and should have a value from 0 to 8. Data Frames use the DLC to indicate the number 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 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) that intended for Receiver nodes to indicate that the Data or Remote Frame was received without 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. |

#### 25.3.2.2 Error and Overload Frames

**Error Frames**

Error Frames are transmitted when a node detects a Bus Error. Error Frames notably consist of an Error Flag which is made up of 6 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 on the next bit. However, if the type of Bus Error was a CRC Error, then the Error Frame will start at the bit following the ACK Delim (see Section 25.3.3). The following Figure 25.3-2 shows the various fields of an Error Frame:

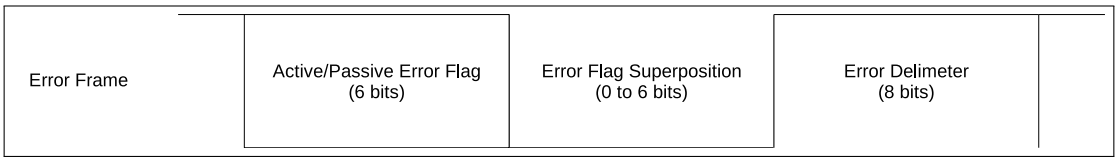



Figure 25.3-2. Various Fields of an Error Frame

Table 25.3-2. Error Frame

| Error Frame | Description |
|---|---|
| Error Flag | The Error Flag has two forms, the Active Error Flag consisting of 6 Dominant bits and the Passive Error Flag consisting of 6 Recessive bits (unless overridden by Dominant bits of other nodes). Active Error Flags are sent by Error Active nodes, whilst Passive Error Flags are sent by Error Passive nodes. |
| Error Flag Superposition | The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first Recessive bit is detected (i.e., the first it of the Delimiter). |
| Error Delimiter | The Delimiter field marks the end of the Error/Overload Frame, and consists of 8 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 conditions that can trigger the transmission of an Overload Frame. Figure 25.3-3 below shows the bit fields of an Overload Frame.

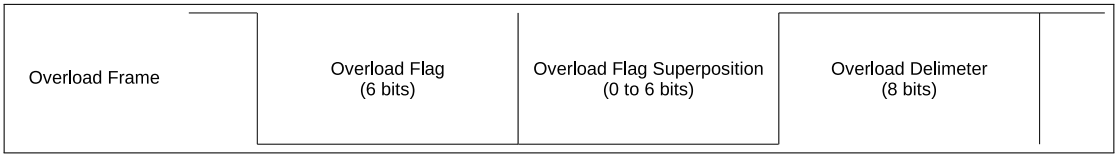



Figure 25.3-3. The Bit Fields of an Overload Frame

Table 25.3-3. Overload Frame

| Overload Frame | Description |
|---|---|
| Overload Flag | Consists of 6 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 8 Recessive. Same as an Error Delimiter. |

Overload Frames will be transmitted under the following conditions:

1. The internal conditions of a Receiver requires a delay of the next Data or Remote Frame.
2. Detection of a Dominant bit at the first and second bit of Intermission.
3. If 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 25.3.3).

Transmitting an overload frame due to one of the conditions must also satisfy the following rules:

- Transmitting an Overload Frame due to condition 1 must only be started at the first bit of Intermission.
- Transmitting an Overload Frame due to condition 2 and 3 must start one bit after the detecting the Dominant bit of the condition.

- A maximum of two Overload frames may be generated in order to delay the next Data or Remote Frame.

#### 25.3.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Data Frames and Remote Frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (Data Frame, Remote Frame, Error Frame, Overload Frame). However, Error Frames and Overload Frames do not need to be separated from preceding frames.

Figure 25.3-4 shows the fields within an Interframe Space:

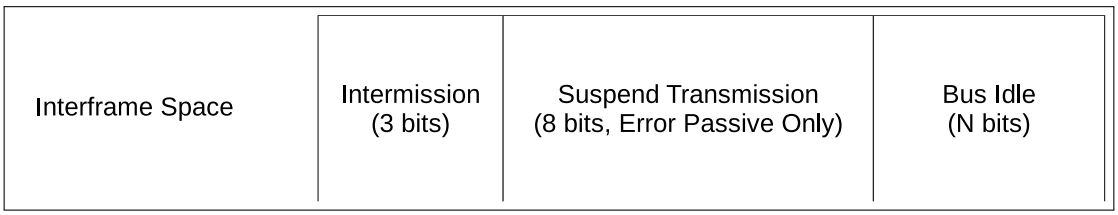



Figure 25.3-4. The Fields within an Interframe Space

Table 25.3-4. Interframe Space

| Interframe Space | Description |
|---|---|
| Intermission | The Intermission consists of 3 Recessive bits. |
| Suspend Transmission | An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of 8 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 Intermission. |

### 25.3.3 TWAI Errors

#### 25.3.3.1 Error Types

Bus Errors in TWAI are categorized into one of the following types:

**Bit Error**

A Bit Error occurs when a node transmits a bit value (i.e., Dominant or Recessive) but the opposite bit is detected (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 or ACK Slot or Passive Error Flag, then detecting a Dominant bit will not be considered a Bit Error.

**Stuff Error**

A stuff error is detected when 6 consecutive bits of the same value are detected (thus violating the bit-stuffing encoding).

**CRC Error**

A Receiver of a Data or Remote Frame will calculate a CRC based on the bits it has received. A CRC error occurs when the CRC calculated by the Receiver does not match the CRC sequence in the received Data or Remote Frame.

**Form Error**

A Form Error is detected when a fixed-form bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be Dominant.

**Acknowledgement Error**

An Acknowledgment Error occurs when a Transmitter does not detect a Dominant bit at the ACK Slot.

#### 25.3.3.2 Error States

TWAI nodes implement fault confinement by each maintaining two error counters, 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, but can only transmit an Passive Error Flag when it detects an error. Error Passive nodes that have transmitted a Data or Remote Frame 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 anything).

#### 25.3.3.3 Error Counters

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply for a given message transfer.**

1. When a Receiver detects an error, the REC will be increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a Receiver detects a Dominant bit as the first bit after sending an Error Flag, the REC will be increased by 8.
3. When a Transmitter sends an Error Flag the TEC is increased by 8. However, the following scenarios are exempt form this rule:
   - If a Transmitter is Error Passive that detects an Acknowledgment Error due to not detecting a Dominant bit in the ACK slot, it should send a Passive Error Flag. If no Dominant bit is detected in that Passive Error Flag, the TEC should not be increased.
   - A Transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the offending bit should have been Recessive but was monitored as Dominant, then the TEC should not be increased.
4. If a Transmitter detects a Bit Error whilst sending an Active Error Flag or Overload Flag, the REC is increased by 8.
5. If a Receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
6. Any node tolerates 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. Each additional eight consecutive Dominant bits will also increase the TEC (for Transmitters) or REC (for Receivers) by 8 as well.

7. When a Transmitter successfully transmits a message (getting ACK and no errors until the EOF is complete), the TEC is decremented by 1, unless the TEC is already at 0.
8. When a Receiver successfully receives a message (no errors before ACK Slot, and successful sending of ACK), the REC is decremented.
   - If the REC was between 1 and 127, the REC is decremented by 1.
   - If the REC was greater than 127, the REC is set to 127.
   - If the REC was 0, the REC remains 0.
9. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. The error condition that causes a node to become Error Passive will cause the node to send an Active Error Flag. Note that once the REC has reached to 128, any further increases to its value are irrelevant until the REC returns to a value less than 128.
10. A node becomes Bus Off when its TEC is greater than or equal to 256.
11. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
12. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive Recessive bits on the bus.

## 25.3.4 TWAI Bit Timing

### 25.3.4.1 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 number of bits transmitted per second from an ideal Transmitter and without any synchronization.
- 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 fixed unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 25.3-5 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 at every Time Quanta. If two consecutive Time Quantas have different bus states (i.e., Recessive to Dominant or vice versa), this will be considered an edge. When the bus is analyzed at the intersection of PBS1 and PBS2, this is considered the Sample Point and the sampled bus value is considered the value of that bit.

**Table 25.3-5. 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. |

| Segment | Description |
|---|---|
| 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. |

#### 25.3.4.2 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 for 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 occurs on Recessive to Dominant edges.

**Hard Synchronization**

Hard Synchronization occurs on the Recessive to Dominant edges during Bus Idle (i.e., the SOF bit). All nodes will restart their internal bit timings such that the Recessive to Dominant edge lies within the SS of the restarted bit timing.

**Resynchronization**

Resynchronization occurs on Recessive to Dominant edges not during Bus 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 a programmable.

- When the magnitude of the Phase Error is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by **e** number of Time Quanta. This has a same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater to 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.

## 25.4 Architectural Overview

The ESP32 contains a TWAI Controller. Figure 25.4-1 shows the major functional blocks of the TWAI Controller.

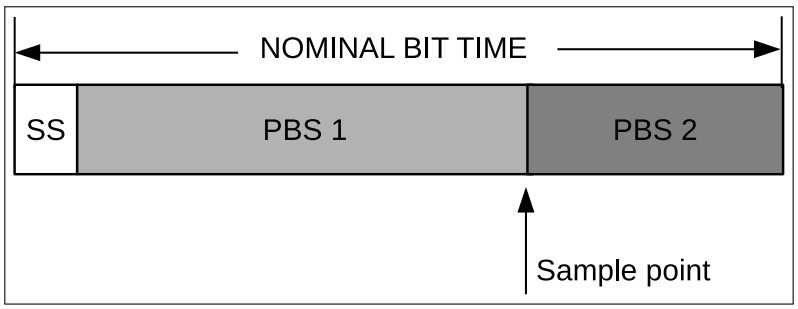



Figure 25.3-5. Layout of a Bit

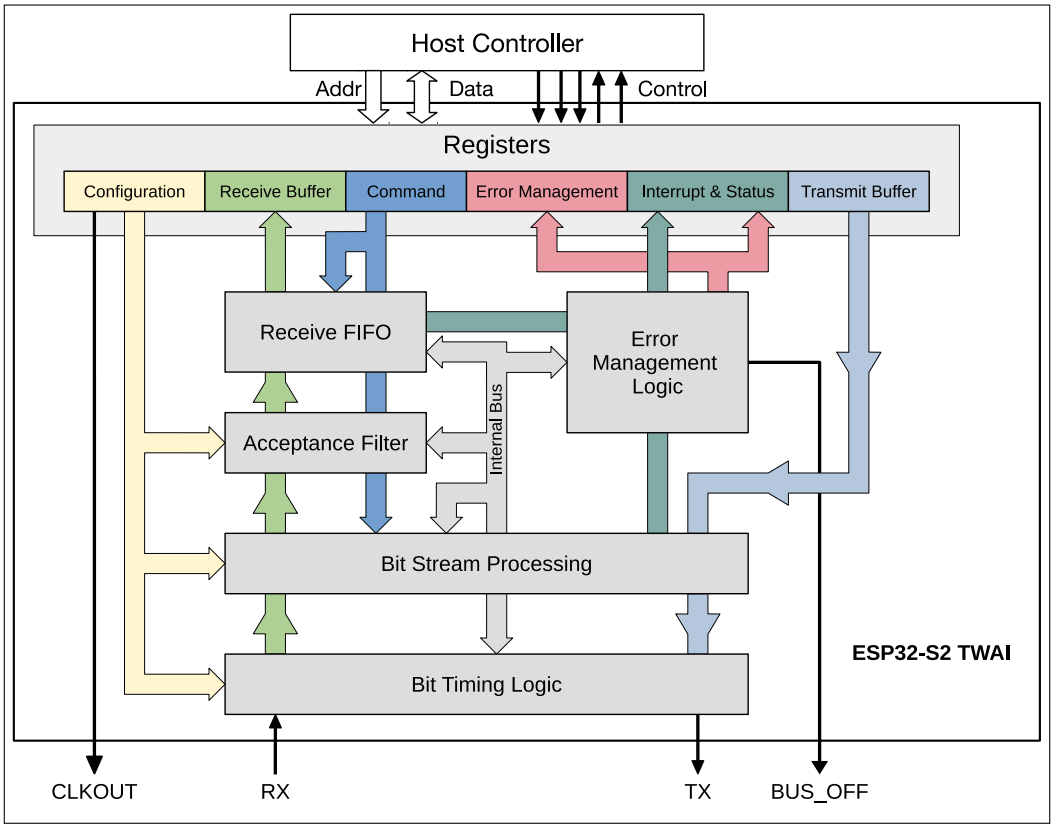



Figure 25.4-1. TWAI Overview Diagram

## 25.4.1 Registers Block

The ESP32 CPU accesses peripherals as 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 options for the TWAI controller such as bit rates, operating mode, Acceptance Filter etc. Configuration registers can only be modified whilst the TWAI controller is in Reset Mode (See Section 25.5.1).

### Command Register

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 25.5.1).

#### 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 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 into Receive FIFO mapping to the first received message in the Receive FIFO to 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, 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.

### 25.4.2 Bit Stream Processor

The Bit Stream Processing (BSP) module is responsible for framing data from the Transmit Buffer (e.g. bit stuffing and additional CRC fields) and generating 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).

### 25.4.3 Error Management Logic

The Error Management Logic (EML) module is responsible for updating the TEC and REC, recording error information like error types and positions, and updating the error state of the TWAI Controller such that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

### 25.4.4 Bit Timing Logic

The Bit Timing Logic (BTL) module is responsible for transmitting and receiving messages at the configured bit rate. The BTL module also handles synchronization of out of phase bits such that communication remains stable. A single bit time consists of multiple programmable segments that allows users to set the length of each segment to account for factors such as propagation delay and controller processing time etc.

### 25.4.5 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter's registers can be programmed to specify a single filter, or specify two separate filters (dual filter mode).

### 25.4.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (internal to 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 next received message in its entirety), the Overrun Interrupt will be triggered, and any subsequent 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 clearing, 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.

## 25.5 Functional Description

### 25.5.1 Modes

The ESP32 TWAI controller has two working modes: Reset Mode and Operation Mode. Reset Mode and Operation Mode are entered by setting the TWAI_RESET_MODE bit to 1 or 0 respectively.

#### 25.5.1.1 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 signaling). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will also not be able to receive any messages.

#### 25.5.1.2 Operation Mode

Entering Operation Mode essentially connects the TWAI controller to the TWAI bus, and write protects the TWAI controller's configuration registers ensuring the configuration stays consistent during operation. When in Operation Mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operating sub-mode the TWAI controller was configured with. The TWAI controller supports the following operating sub-modes:

- **Normal Mode:** The TWAI controller can transmit and receive messages including error signaling (such as Error and Overload Frames).
- **Self Test Mode:** Like Normal Mode, but the TWAI controller will consider the transmission of a Data or RTR Frame successful even if it was not acknowledged. This is commonly used when self testing the 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 monitors.

Note that when exiting Reset Mode (i.e., entering Operation Mode), the TWAI controller must wait for 11 consecutive Recessive bits to occur before being able to fully connect the TWAI bus (i.e., be able to transmit or receive).

### 25.5.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset Mode. The bit rate configuration is located in TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 25.5-1 illustrates the bit fields of TWAI_BUS_TIMING_0_REG.

**Table 25.5-1. Bit Information of TWAI_CLOCK_DIVIDER_REG; TWAI Address 0x18**

| Bit 31-8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Reserved | SJW.1 | SJW.0 | BRP.5 | BRP.4 | BRP.3 | BRP.2 | BRP.1 | BRP.0 |

**Notes:**

- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1).
- BRP: The TWAI Time Quanta clock is derived from a prescaled version of the APB clock that is usually 80 MHz. 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 period and $t_{CLK}$ is APB clock period :
  $t_{Tq} = 2 \times t_{CLK} \times (2^5 \times BRP.5 + 2^4 \times BRP.4 + 2^3 \times BRP.3 + 2^2 \times BRP.2 + 2^1 \times BRP.1 + 2^0 \times BRP.0 + 1)$

The following Table 25.5-2 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

**Table 25.5-2. Bit Information of TWAI_BUS_TIMING_1_REG; TWAI Address 0x1c**

| Bit 31-8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Reserved | SAM | PBS2.2 | PBS2.1 | PBS2.0 | PBS1.3 | PBS1.2 | PBS1.1 | PBS1.0 |

**Notes:**

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined according to the following equation: (8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1).
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined according to the following equation: (4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1).
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses where filtering spikes on the bus line is beneficial.

### 25.5.3 Interrupt Management

The ESP32 TWAI controller provides seven interrupts, each represented by a single bit in the TWAI_INT_RAW_REG. For a particular interrupt to be triggered (i.e., its bit in TWAI_INT_RAW_REG set to 1), the interrupt's corresponding enable bit in TWAI_INT ENA_REG must be set.

The TWAI controller provides the seven following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt
- Arbitration Lost Interrupt
- Bus Error Interrupt

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INT_RAW_REG, and deasserted when all bits in TWAI_INT_RAW_REG are cleared. The majority of interrupt bits in TWAI_INT_RAW_REG are automatically cleared when the register is read. However, the Receive Interrupt is an exception and can only be cleared the Receive FIFO is empty.

#### 25.5.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_CNT_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUF command bit.

#### 25.5.3.2 Transmit Interrupt (TXI)

The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer to be transmitted. The Transmit Buffer becomes free under the following scenarios:

- A message transmission has completed successfully (i.e., Acknowledged without any errors). Any failed messages will automatically be retried.
- A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_TX_COMPLETE bit).
- A message transmission was aborted using the TWAI_ABORT_TX command bit.

#### 25.5.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST and TWAI_BUS_OFF_ST bits of 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_ERR_ST and TWAI_BUS_OFF_ST at the moment the EWI is triggered.

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
  - If the TWAI controller was previously in the Bus Recovery state, it indicates that Bus Recovery has completed successfully.

- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).
- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

#### 25.5.3.4 Data Overrun Interrupt (DOI)

The Data Overrun Interrupt (DOI) is triggered when the message being read is overrun and invalid.

The DOI is only triggered on 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 will only be able to trigger again when all received messages (valid or overrun) have been cleared.

#### 25.5.3.5 Error Passive Interrupt (TXI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller transitions from Error Active to Error Passive, or vice versa.

#### 25.5.3.6 Arbitration Lost Interrupt (ALI)

The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB LOST CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record new bit location until it is cleared (via a read from the CPU).

#### 25.5.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever TWAI controller observes 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).

### 25.5.4 Transmit and Receive Buffers

#### 25.5.4.1 Overview of Buffers

**Table 25.5-3. Buffer Layout for Standard Frame Format and Extended Frame Format**

| Standard Frame Format (SFF) | | Extended Frame Format (EFF) | |
|---|---|---|---|
| TWAI address | Content | TWAI 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 |

| Standard Frame Format (SFF) | | Extended Frame Format (EFF) | |
|---|---|---|---|
| TWAI address | Content | TWAI address | Content |
| 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 25.5-3 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. CPU write operations will access the Transmit Buffer registers, and CPU read operations will access the Receive Buffer registers. However, both buffers share the exact same register layout and fields to represent a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQ bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQ bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQ and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map to the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUF bit in TWAI_CMD_REG so that the next message in the Receive FIFO will be loaded in to the Receive Buffer registers.

### 25.5.4.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 25.5-4.

**Table 25.5-4. 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] | XDLC.3[4] | DLC.2[4] | DLC.1[4] | DLC.0[4] |

**Notes:**

- 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 FF bit is 1, and SFF when FF bit is 0.
- 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.

- 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, thus the DLC should range anywhere from 0 to 8.
- X: Don't care, can be any value.

#### 25.5.4.3 Frame Identifier

The Frame Identifier fields is 2 bytes (11-bits) if the message is SFF, and 4 bytes (29-bits) if the message is EFF.

The Frame Identifier fields for an SFF (11-bits) message is shown in Table 25.5-5-25.5-6.

Table 25.5-5. 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.28 | ID.27 | ID.26 | ID.25 | ID.24 | ID.23 | ID.22 | ID.21 |

Table 25.5-6. 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.20 | ID.19 | ID.18 | X[1] | X[2] | X[2] | X[2] | X[2] |

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 25.5-7-25.5-10.

Table 25.5-7. 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 25.5-8. 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 25.5-9. 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 25.5-10. 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] |

#### 25.5.4.4 Frame Data

The Frame Data fields contains the payload of transmitted or received a 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 8, the number of valid bytes would still be limited to 8. Remote Frames do not have data payloads, thus the Frame Data fields will be unused.

For example, when transmitting a Data Frame with 5 data bytes, the CPU should write a value of 5 to the DLC field, and then fill in data bytes 1 to 5 in the Frame Data fields. Likewise, when receiving a Data Frame with a DLC of 5, only data bytes 1 to 5 will contain valid payload data for the CPU to read.

### 25.5.5 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13-bytes of space in the Receive FIFO, and their byte layout is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO. When the TWAI controller receives a message, it will increment the value of TWAI_RX_MESSAGE_COUNTER up to a maximum of 64. If there is adequate space in the Receive FIFO, the message contents will be written into the Receive FIFO. Once a message has been read from the Receive Buffer, the TWAI_RELEASE_BUF bit should be set. This will decrement TWAI_RX_MESSAGE_COUNTER and free the space occupied by the first message in the Receive FIFO. The Receive Buffer will then map to the next message in the Receive FIFO.

When the TWAI controller receives a message, but the Receive FIFO lacks the 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), the Receive FIFO will internally mark overrun messages as invalid. Subsequent overrun messages will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUF must be called repeatedly called until TWAI_RX_
MESSAGE_COUNTER is 0. This has the effect of freeing all valid messages in the Receive FIFO and clearing all overrun messages.

### 25.5.6 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows for a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) due to the TWAI Controller only needing to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset Mode, due to those registers sharing the same address space as the Transmit Buffer and Receive Buffer registers.

The registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Code value specifies a bit pattern in which each filtered bit of the message must match in order for the message to be accepted. The 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 25.5-1.

The TWAI Controller Acceptance Filter allows the 32-bit 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 whether Single Filter Mode is enabled, and the received message (i.e., SFF or EFF).

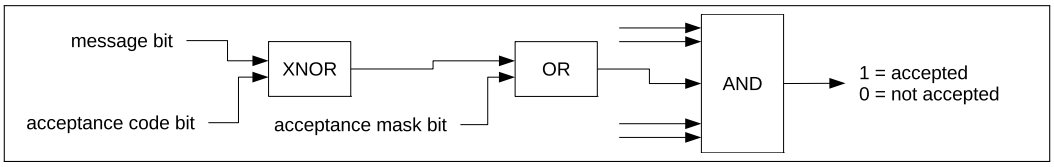



Figure 25.5-1. Acceptance Filter

### 25.5.6.1 Single Filter Mode

Single Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of a Data or Remote Frame:

- 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 25.5-2 illustrates how the 32-bit code and mask values will be interpreted under Single Filter Mode.

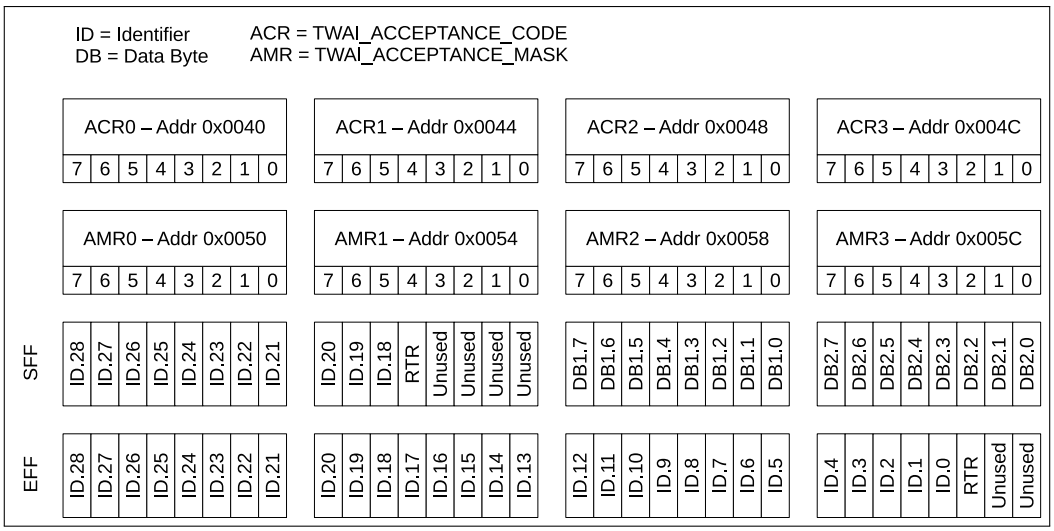



Figure 25.5-2. Single Filter Mode

#### 25.5.6.2 Dual FIlter Mode

Dual Filter Mode is enabled by setting the TWAI_RX_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 two. 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 a Data or Remote Frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)
- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 25.5-3 illustrates how the 32-bit code and mask values will be interpreted under Dual Filter Mode.

### 25.5.7 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Count (TEC) and Receive Error Count (REC). The value of both error counts determine 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 the TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and can be read by the CPU at anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn the user regarding the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_ERR_ST, and TWAI_BUS_OFF_ST. Certain changes to these values and bits will also trigger interrupts, thus allowing the users to be notified of error state transitions (see section 25.5.3). The following figure 25.5-4 shows the relation between the error states, values and bits, and error state related interrupts.

#### 25.5.7.1 Error Warning Limit

The Error Warning Limit (EWL) feature is a configurable threshold value for the TEC and REC, where if exceeded, will trigger an interrupt. 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 the TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset Mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96. When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_ERR_ST bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_ERR_ST bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_ERR_ST bit (or the TWAI_BUS_OFF_ST) changes.

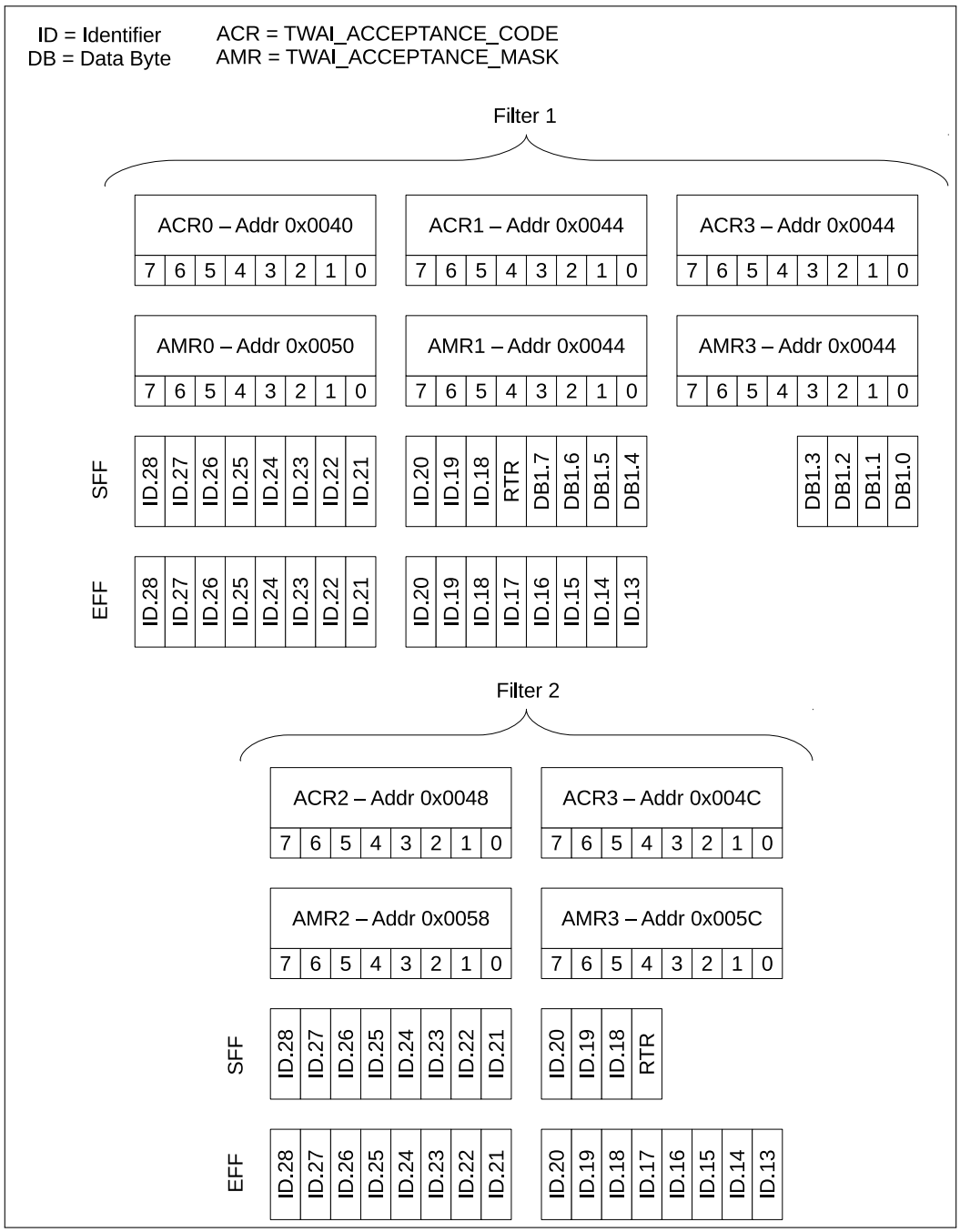



Figure 25.5-3. Dual Filter Mode

### 25.5.7.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

### 25.5.7.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

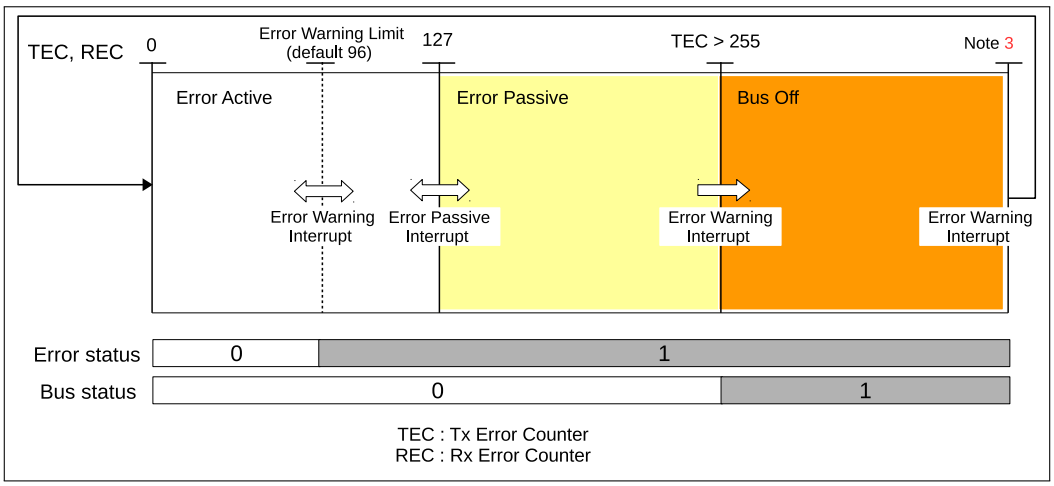



Figure 25.5-4. Error State Transition

- Set REC to 0
- Set TEC to 127
- Set the TWAI_BUS_OFF_ST bit to 1
- Enter Reset Mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_BUS_OFF_ST bit (or the TWAI_ERR_ST bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off recovery. Bus-Off recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive Recessive bits on the bus. To initiate Bus-Off recovery (after entering the Bus-Off state), the TWAI controller should enter Operation Mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off recovery by decrementing the TEC each time the TWAI controller observes 11 consecutive Recessive bits. When Bus-Off recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

## 25.5.8 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in the 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 the TWAI_ERR_CODE_CAP_REG.

The following Table 25.5-11 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 25.5-11. Bit Information of TWAI_ERR_CODE_CAP_REG; TWAI Address 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 form error, 10 for stuff error, 11 for other type of error.

- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error: 0 for Transmitter, 1 for Receiver.
- SEG: The Error Segment (SEG) indicates which segment of the TWAI message (i.e., bit position) the bus error occurred at.

The following Table 25.5-12 shows how to interpret the SEG.0 to SEG.4 bits.

**Table 25.5-12. 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 to ID.21 |
| 0 | 0 | 1 | 1 | 0 | ID.20 to ID.18 |
| 0 | 0 | 1 | 0 | 0 | bit SRTR[1] |
| 0 | 0 | 1 | 0 | 1 | bit IDE[2] |
| 0 | 0 | 1 | 1 | 1 | ID.17 to ID.13 |
| 0 | 1 | 1 | 1 | 1 | ID.12 to ID.5 |
| 0 | 1 | 1 | 1 | 0 | ID.4 to ID.0 |
| 0 | 1 | 1 | 0 | 0 | bit RTR |
| 0 | 1 | 1 | 0 | 1 | reserved bit 1 |
| 0 | 1 | 0 | 0 | 1 | reserved bit 0 |
| 0 | 1 | 0 | 1 | 1 | data length code |
| 0 | 1 | 0 | 1 | 0 | data field |
| 0 | 1 | 0 | 0 | 0 | CRC sequence |
| 1 | 1 | 0 | 0 | 0 | CRC delimiter |
| 1 | 1 | 0 | 0 | 1 | acknowledge slot |
| 1 | 1 | 0 | 1 | 1 | acknowledge 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 RTR: under Standard Frame Format.
- Identifier Extension Bit: 0 for Standard Frame Format.

### 25.5.9 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in the TWAI_ARB LOST CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent loses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in the TWAI_ARB LOST CAP_REG until the current Arbitration Lost Capture is read from the

TWAI_ERR_CODE_CAP_REG.

Table 25.5-13 illustrates the bit fields of the TWAI_ERR_CODE_CAP_REG whilst Figure 25.5-5 illustrates the bit positions of a TWAI message.

**Table 25.5-13. Bit Information of TWAI_ARB LOST CAP_REG; TWAI Address 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.

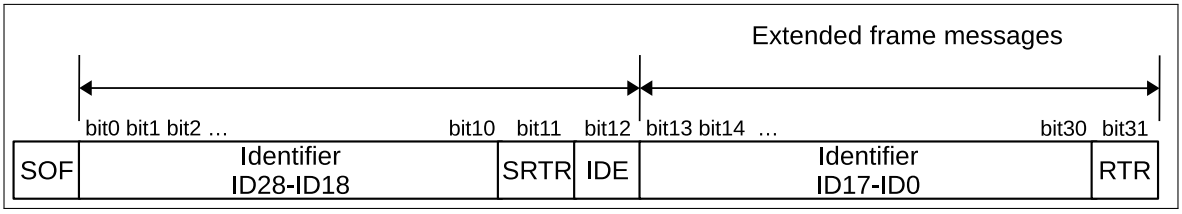



**Figure 25.5-5. Positions of Arbitration Lost Bits**

## 25.6 Register Summary

The addresses in this section are relative to the TWAI base address provided in Table 3.3-6 in Chapter 3 *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 | 0x3FF6B000 | R/W |
| TWAI_BUS_TIMING_0_REG | Bus Timing Register 0 | 0x3FF6B018 | RO \| R/W |
| TWAI_BUS_TIMING_1_REG | Bus Timing Register 1 | 0x3FF6B01C | RO \| R/W |
| TWAI_ERR_WARNING_LIMIT_REG | Error Warning Limit Register | 0x3FF6B034 | RO \| R/W |
| TWAI_DATA_0_REG | Data Register 0 | 0x3FF6B040 | WO \| R/W |
| TWAI_DATA_1_REG | Data Register 1 | 0x3FF6B044 | WO \| R/W |
| TWAI_DATA_2_REG | Data Register 2 | 0x3FF6B048 | WO \| R/W |
| TWAI_DATA_3_REG | Data Register 3 | 0x3FF6B04C | WO \| R/W |
| TWAI_DATA_4_REG | Data Register 4 | 0x3FF6B050 | WO \| R/W |
| TWAI_DATA_5_REG | Data Register 5 | 0x3FF6B054 | WO \| R/W |
| TWAI_DATA_6_REG | Data Register 6 | 0x3FF6B058 | WO \| R/W |
| TWAI_DATA_7_REG | Data Register 7 | 0x3FF6B05C | WO \| R/W |
| TWAI_DATA_8_REG | Data Register 8 | 0x3FF6B060 | WO \| RO |
| TWAI_DATA_9_REG | Data Register 9 | 0x3FF6B064 | WO \| RO |
| TWAI_DATA_10_REG | Data Register 10 | 0x3FF6B068 | WO \| RO |
| TWAI_DATA_11_REG | Data Register 11 | 0x3FF6B06C | WO \| RO |
| TWAI_DATA_12_REG | Data Register 12 | 0x3FF6B070 | WO \| RO |

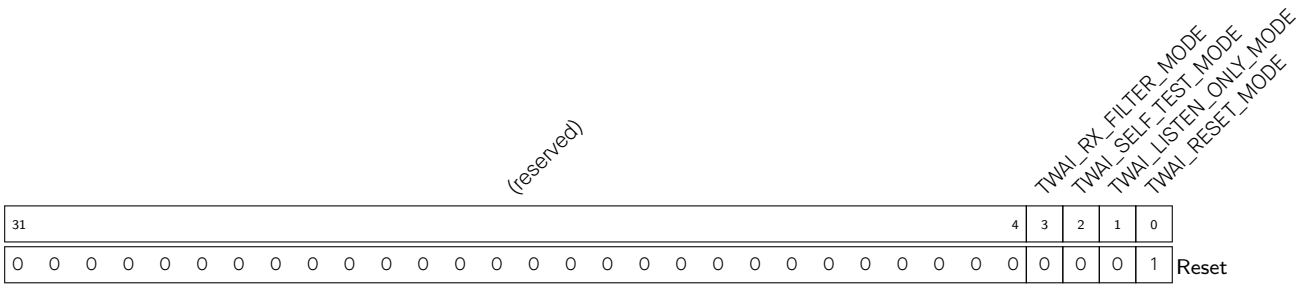
| Name | Description | Address | Access |
|---|---|---|---|
| TWAI_CLOCK_DIVIDER_REG | Clock Divider Register | 0x3FF6B07C | varies |
| **Control Registers** | | | |
| TWAI_CMD_REG | Command Register | 0x3FF6B004 | WO |
| **Status Registers** | | | |
| TWAI_STATUS_REG | Status Register | 0x3FF6B008 | RO |
| TWAI_ARB_LOST_CAP_REG | Arbitration Lost Capture Register | 0x3FF6B02C | RO |
| TWAI_ERR_CODE_CAP_REG | Error Code Capture Register | 0x3FF6B030 | RO |
| TWAI_RX_ERR_CNT_REG | Receive Error Counter Register | 0x3FF6B038 | RO \| R/W |
| TWAI_TX_ERR_CNT_REG | Transmit Error Counter Register | 0x3FF6B03C | RO \| R/W |
| TWAI_RX_MESSAGE_CNT_REG | Receive Message Counter Register | 0x3FF6B074 | RO |
| **Interrupt Registers** | | | |
| TWAI_INT_RAW_REG | Interrupt Register | 0x3FF6B00C | RO |
| TWAI_INT_ENA_REG | Interrupt Enable Register | 0x3FF6B010 | R/W |

## 25.7 Registers

The addresses in this section are relative to the TWAI base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 25.6 *Register Summary*.

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

**Register 25.1. TWAI_MODE_REG (0x0000)**




**TWAI_RESET_MODE** This bit is used to configure the operating mode of the TWAI Controller. 1: Reset mode; 0: Operating mode (R/W)

**TWAI_LISTEN_ONLY_MODE** 1: Listen only mode. In this mode the nodes will only receive messages from the bus, without generating the acknowledge signal nor updating the RX error counter. (R/W)

**TWAI_SELF_TEST_MODE** 1: Self test mode. In this mode the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self reception request command. (R/W)

**TWAI_RX_FILTER_MODE** This bit is used to configure the filter mode. 0: Dual filter mode; 1: Single filter mode (R/W)

**Register 25.2. TWAI_BUS_TIMING_0_REG (0x0018)**

| (reserved) | TWAI_SYNC_JUMP_WIDTH | TWAI_BAUD_PRESC | |
|---|---|---|---|
| 31 … 8 | 7 … 6 | 5 … 0 | |
| 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 | 0x00 | Reset |

**TWAI_BAUD_PRESC** Baud Rate Prescaler, determines the frequency dividing ratio. (RO | R/W)

**TWAI_SYNC_JUMP_WIDTH** Synchronization Jump Width (SJW), 1 ~ 4 Tq wide. (RO | R/W)

**Register 25.3. TWAI_BUS_TIMING_1_REG (0x001C)**

| (reserved) | TWAI_TIME_SAMP | TWAI_TIME_SEG2 | TWAI_TIME_SEG1 | |
|---|---|---|---|---|
| 31 … 8 | 7 | 6 … 4 | 3 … 0 | |
| 0 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 | 0x0 | Reset |

**TWAI_TIME_SEG1** The width of PBS1. (RO | R/W)

**TWAI_TIME_SEG2** The width of PBS2. (RO | R/W)

**TWAI_TIME_SAMP** The number of sample points. 0: the bus is sampled once; 1: the bus is sampled three times (RO | R/W)

**Register 25.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)**

| (reserved) | TWAI_ERR_WARNING_LIMIT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x60 | Reset |

**TWAI_ERR_WARNING_LIMIT** Error warning threshold. In the case when any of a error counter value exceeds the threshold, or all the error counter values are below the threshold, an error warning interrupt will be triggered (given the enable signal is valid). (RO | R/W)

**Register 25.5. TWAI_DATA_0_REG (0x0040)**

| (reserved) | TWAI_TX_BYTE_0 \| TWAI_ACCEPTANCE_CODE_0 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_0** Stored the 0th byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_CODE_0** Stored the 0th byte of the filter code under reset mode. (R/W)

**Register 25.6. TWAI_DATA_1_REG (0x0044)**

| (reserved) | TWAI_TX_BYTE_1 \| TWAI_ACCEPTANCE_CODE_1 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_1** Stored the 1st byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_CODE_1** Stored the 1st byte of the filter code under reset mode. (R/W)

### Register 25.7. TWAI_DATA_2_REG (0x0048)

| (reserved) | TWAI_TX_BYTE_2 \| TWAI_ACCEPTANCE_CODE_2 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_2** Stored the 2nd byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_CODE_2** Stored the 2nd byte of the filter code under reset mode. (R/W)

### Register 25.8. TWAI_DATA_3_REG (0x004C)

| (reserved) | TWAI_TX_BYTE_3 \| TWAI_ACCEPTANCE_CODE_3 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_3** Stored the 3rd byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_CODE_3** Stored the 3rd byte of the filter code under reset mode. (R/W)

### Register 25.9. TWAI_DATA_4_REG (0x0050)

| (reserved) | TWAI_TX_BYTE_4 \| TWAI_ACCEPTANCE_MASK_0 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_4** Stored the 4th byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_MASK_0** Stored the 0th byte of the filter code under reset mode. (R/W)

### Register 25.10. TWAI_DATA_5_REG (0x0054)

| (reserved) | TWAI_TX_BYTE_5 \| TWAI_ACCEPTANCE_MASK_1 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_5** Stored the 5th byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_MASK_1** Stored the 1st byte of the filter code under reset mode. (R/W)

### Register 25.11. TWAI_DATA_6_REG (0x0058)

| (reserved) | TWAI_TX_BYTE_6 \| TWAI_ACCEPTANCE_MASK_2 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_6** Stored the 6th byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_MASK_2** Stored the 2nd byte of the filter code under reset mode. (R/W)

### Register 25.12. TWAI_DATA_7_REG (0x005C)

| (reserved) | TWAI_TX_BYTE_7 \| TWAI_ACCEPTANCE_MASK_3 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_7** Stored the 7th byte information of the data to be transmitted under operating mode. (WO)

**TWAI_ACCEPTANCE_MASK_3** Stored the 3rd byte of the filter code under reset mode. (R/W)

**Register 25.13. TWAI_DATA_8_REG (0x0060)**

| (reserved) | TWAI_TX_BYTE_8 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_8** Stored the 8th byte information of the data to be transmitted under operating mode. (WO)

**Register 25.14. TWAI_DATA_9_REG (0x0064)**

| (reserved) | TWAI_TX_BYTE_9 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_9** Stored the 9th byte information of the data to be transmitted under operating mode. (WO)

**Register 25.15. TWAI_DATA_10_REG (0x0068)**

| (reserved) | TWAI_TX_BYTE_10 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_10** Stored the 10th byte information of the data to be transmitted under operating mode. (WO)

**Register 25.16. TWAI_DATA_11_REG (0x006C)**

| (reserved) | TWAI_TX_BYTE_11 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_11** Stored the 11th byte information of the data to be transmitted under operating mode. (WO)

**Register 25.17. TWAI_DATA_12_REG (0x0070)**

| (reserved) | TWAI_TX_BYTE_12 | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_BYTE_12** Stored the 12th byte information of the data to be transmitted under operating mode. (WO)

**Register 25.18. TWAI_CLOCK_DIVIDER_REG (0x007C)**

| (reserved) | TWAI_EXT_MODE | (reserved) | TWAI_CLOCK_OFF | TWAI_CD | |
|---|---|---|---|---|---|
| 31 … 8 | 7 | 6 … 4 | 3 | 2 … 0 | |
| 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 | 0x0 | Reset |

**TWAI_CD** These bits are used to configure frequency dividing coefficients of the external CLKOUT pin. (R/W)

**TWAI_CLOCK_OFF** This bit can be configured under reset mode. 1: Disable the external CLKOUT pin; 0: Enable the external CLKOUT pin (RO | R/W)

**TWAI_EXT_MODE** This bit can be configured under reset mode. 1: Extended mode, compatible with CAN2.0B; 0: Basic mode (RO | R/W)

**Register 25.19. TWAI_CMD_REG (0x0004)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | TWAI_SELF_RX_REQ | TWAI_CLR_OVERRUN | TWAI_RELEASE_BUF | TWAI_ABORT_TX | TWAI_TX_REQ | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**TWAI_TX_REQ** Set the bit to 1 to allow the driving nodes start transmission. (WO)

**TWAI_ABORT_TX** Set the bit to 1 to cancel a pending transmission request. (WO)

**TWAI_RELEASE_BUF** Set the bit to 1 to release the RX buffer. (WO)

**TWAI_CLR_OVERRUN** Set the bit to 1 to clear the data overrun status bit. (WO)

**TWAI_SELF_RX_REQ** Self reception request command. Set the bit to 1 to allow a message be transmitted and received simultaneously. (WO)

**Register 25.20. TWAI_STATUS_REG (0x0008)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | TWAI_BUS_OFF_ST | TWAI_ERR_ST | TWAI_TX_ST | TWAI_RX_ST | TWAI_TX_COMPLETE | TWAI_TX_BUF_ST | TWAI_OVERRUN_ST | TWAI_RX_BUF_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 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 | 1 | 0 | 0 | Reset |

**TWAI_RX_BUF_ST** 1: The data in the RX buffer is not empty, with at least one received data packet. (RO)

**TWAI_OVERRUN_ST** 1: The RX FIFO is full and data overrun has occurred. (RO)

**TWAI_TX_BUF_ST** 1: The TX buffer is empty, the CPU may write a message into it. (RO)

**TWAI_TX_COMPLETE** 1: The TWAI controller has successfully received a packet from the bus. (RO)

**TWAI_RX_ST** 1: The TWAI Controller is receiving a message from the bus. (RO)

**TWAI_TX_ST** 1: The TWAI Controller is transmitting a message to the bus. (RO)

**TWAI_ERR_ST** 1: At least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

**TWAI_BUS_OFF_ST** 1: In bus-off status, the TWAI Controller is no longer involved in bus activities. (RO)

### Register 25.21. TWAI_ARB_LOST_CAP_REG (0x002C)

| (reserved) | TWAI_ARB_LOST_CAP | |
|---|---|---|
| 31 … 5 | 4 … 0 | |
| 0 0 0 0 0 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 |

**TWAI_ARB_LOST_CAP** This register contains information about the bit position of lost arbitration. (RO)

### Register 25.22. TWAI_ERR_CODE_CAP_REG (0x0030)

| (reserved) | TWAI_ECC_TYPE | TWAI_ECC_DIRECTION | TWAI_ECC_SEGMENT | |
|---|---|---|---|---|
| 31 … 8 | 7 … 6 | 5 | 4 … 0 | |
| 0 0 0 0 0 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 | 0 | 0x0 | Reset |

**TWAI_ECC_SEGMENT** This register contains information about the location of errors, see Table 25.5-11 for details. (RO)

**TWAI_ECC_DIRECTION** This register contains information about transmission direction of the node when error occurs. 1: Error occurs when receiving a message; 0: Error occurs when transmitting a message (RO)

**TWAI_ECC_TYPE** This register contains information about error types: 00: bit error; 01: form error; 10: stuff error; 11: other type of error (RO)

### Register 25.23. TWAI_RX_ERR_CNT_REG (0x0038)

| (reserved) | TWAI_RX_ERR_CNT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 0 0 0 0 0 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 |

**TWAI_RX_ERR_CNT** The RX error counter register, reflects value changes under reception status. (RO | R/W)

### Register 25.24. TWAI_TX_ERR_CNT_REG (0x003C)

| (reserved) | TWAI_TX_ERR_CNT | |
|---|---|---|
| 31 … 8 | 7 … 0 | |
| 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 |

**TWAI_TX_ERR_CNT** The TX error counter register, reflects value changes under transmission status. (RO | R/W)

### Register 25.25. TWAI_RX_MESSAGE_CNT_REG (0x0074)

| (reserved) | TWAI_RX_MESSAGE_COUNTER | |
|---|---|---|
| 31 … 7 | 6 … 0 | |
| 0 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 |

**TWAI_RX_MESSAGE_COUNTER** This register reflects the number of messages available within the RX FIFO. (RO)

**Register 25.26. TWAI_INT_RAW_REG (0x000C)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | TWAI_BUS_ERR_INT_ST | TWAI_ARB_LOST_INT_ST | TWAI_ERR_PASSIVE_INT_ST | (reserved) | TWAI_OVERRUN_INT_ST | TWAI_ERR_WARN_INT_ST | TWAI_TX_INT_ST | TWAI_RX_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**TWAI_RX_INT_ST** Receive interrupt. If this bit is set to 1, it indicates there are messages to be handled in the RX FIFO. (RO)

**TWAI_TX_INT_ST** Transmit interrupt. If this bit is set to 1, it indicates the message transmitting mission is finished and a new transmission is able to execute. (RO)

**TWAI_ERR_WARN_INT_ST** Error warning interrupt. If this bit is set to 1, it indicates the error status signal and the bus-off status signal of Status register have changed (e.g., switched from 0 to 1 or from 1 to 0). (RO)

**TWAI_OVERRUN_INT_ST** Data overrun interrupt. If this bit is set to 1, it indicates the data in the RX FIFO is invalid. (RO)

**TWAI_ERR_PASSIVE_INT_ST** Error passive interrupt. If this bit is set to 1, it indicates the TWAI Controller is switched between error active status and error passive status due to the change of error counters. (RO)

**TWAI_ARB_LOST_INT_ST** Arbitration lost interrupt. If this bit is set to 1, it indicates an arbitration lost interrupt is generated. (RO)

**TWAI_BUS_ERR_INT_ST** Error interrupt. If this bit is set to 1, it indicates an error is detected on the bus. (RO)

**Register 25.27. TWAI_INT ENA_REG (0x0010)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | TWAI_BUS_ERR_INT_ENA | TWAI_ARB_LOST_INT_ENA | TWAI_ERR_PASSIVE_INT_ENA | (reserved) | TWAI_OVERRUN_INT_ENA | TWAI_ERR_WARN_INT_ENA | TWAI_TX_INT_ENA | TWAI_RX_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**TWAI_RX_INT_ENA** Set this bit to 1 to enable receive interrupt. (R/W)

**TWAI_TX_INT_ENA** Set this bit to 1 to enable transmit interrupt. (R/W)

**TWAI_ERR_WARN_INT_ENA** Set this bit to 1 to enable error warning interrupt. (R/W)

**TWAI_OVERRUN_INT_ENA** Set this bit to 1 to enable data overrun interrupt. (R/W)

**TWAI_ERR_PASSIVE_INT_ENA** Set this bit to 1 to enable error passive interrupt. (R/W)

**TWAI_ARB_LOST_INT_ENA** Set this bit to 1 to enable arbitration lost interrupt. (R/W)

**TWAI_BUS_ERR_INT_ENA** Set this bit to 1 to enable error interrupt. (R/W)

# Chapter 26

# SDIO Slave Controller (SDIO)

## 26.1 Overview

The ESP32 features hardware support for the industry-standard Secure Digital (SD) device interface that conforms to the SD Input/Output (SDIO) Specification Version 2.0. This allows a host controller to access the ESP32 via an SDIO bus protocol, enabling high-speed data transfer.

The SDIO interface may be used to read ESP32 SDIO registers directly and access shared memory via Direct Memory Access (DMA), thus reducing processing overhead while maintaining high performance.

## 26.2 Features

- Meets SDIO V2.0 specification
- Supports SDIO SPI, 1-bit, and 4-bit transfer modes
- Full host clock range of 0 ~ 50 MHz
- Configurable sample and drive clock edge
- Integrated, SDIO-accessible registers for information interaction
- Supports SDIO interrupt mechanism
- Automatic data padding
- Block size of up to 512 bytes
- Interrupt vector between Host and Slave for bidirectional interrupt
- Supports DMA for data transfer

## 26.3 Functional Description

### 26.3.1 SDIO Slave Block Diagram

The functional block diagram of the SDIO slave module is shown in Figure 26.3-1.

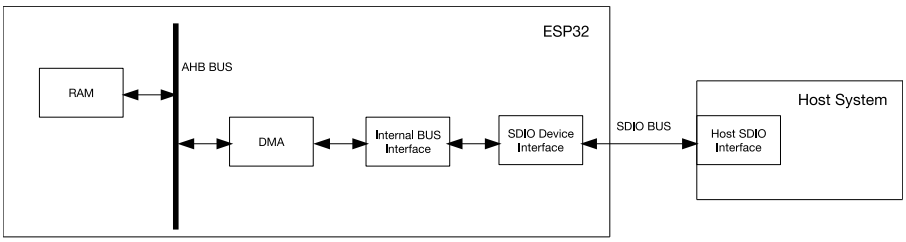



Figure 26.3-1. SDIO Slave Block Diagram

The Host System represents any SDIO specification V2.0-compatible host device. The Host System interacts with the ESP32 (configured as the SDIO slave) via the standard SDIO bus implementation.

The SDIO Device Interface block enables effective communication with the external Host by directly providing SDIO interface registers and enabling DMA operation for high-speed data transfer over the Advanced High-performance Bus (AHB) without engaging the CPU.

## 26.3.2 Sending and Receiving Data on SDIO Bus

Data is transmitted between Host and Slave through the SDIO bus I/O Function1. After the Host enables the I/O Function1 in the Slave, according to the SDIO protocol, data transmission will begin.

ESP32 segregates data into packets sent to/from the Host. To achieve high bus utilization and data transfer rates, we recommend the single block transmission mode. For detailed information on this mode, please refer to the SDIO V2.0 protocol specification. When Host and Slave exchange data as blocks on the SDIO bus, the Slave automatically pads data-when sending data out-and automatically strips padding data from the incoming data block.

Whether the Slave pads or discards the data depends on the data address on the SDIO bus. When the data address is equal to, or greater than, 0x1F800, the Slave will start padding or discarding data. Therefore, the starting data address should be 0x1F800 - Packet_length, where Packet_length is measured in bytes. Data flow on the SDIO bus is shown in Figure 26.3-2.

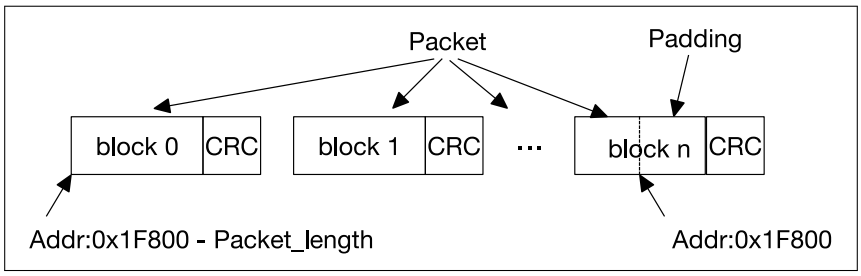



Figure 26.3-2. SDIO Bus Packet Transmission

The standard IO_RW_EXTENDED (CMD53) command is used to initiate a packet transfer of an arbitrary length. The content of the CMD53 command used in data transmission is as illustrated in Figure 26.3-3 below. For detailed information on CMD53, please refer to the SDIO protocol specifications.

| S | D | Command Index 110101b | R/W flag | Function Number 001b | Block Mode 1b | OP Code 1b | Register Address 0X1F800-Packet_length | Byte/Block Count Roundup (Packet_length/Block_size) | CRC7 | E |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 1 | 3 | 1 | 1 | 17 | 9 | 7 | 1 |

Figure 26.3-3. CMD53 Content

## 26.3.3 Register Access

For effective interaction between Host and Slave, the Host can access certain registers in the Slave via the SDIO bus I/O Function1. These registers are in continuous address fields from SLC0HOST_TOKEN_RDATA to SSLC0HOST_INT_ST_REG. The Host device can access these registers by simply setting the register addresses of CMD52 or CMD53 to the low 10 bits of the corresponding register address. The Host can access several consecutive registers at one go with CMD53, thus achieving a higher effective transfer rate.

There are 52 bytes of field between SLCHOST_CONF_W0_REG and SLCHOST_CONF_W15_REG. Host and Slave can access and change these fields, thus facilitating the information interaction between Host and Slave.

## 26.3.4 DMA

The SDIO Slave module uses dedicated DMA to access data residing in the RAM. As shown in Figure 26.3-1, the RAM is accessed over the AHB. DMA accesses RAM through a linked-list descriptor. Every linked list is composed of three words, as shown in Figure 26.3-4.

| 1 | 1 | 6 | 12 | 12 |
|---|---|---|---|---|
| Owner | Eof | Reserved | Length | Size |
| Buffer Address Pointer | | | | |
| Next Descriptor Address | | | | |

Figure 26.3-4. SDIO Slave DMA Linked List Structure

- Owner: The allowed operator of the buffer that corresponds to the current linked list. 0: CPU is the allowed operator; 1: DMA is the allowed operator.
- Eof: End-of-file marker, indicating that this linked-list element is the last element of the data packet.
- Length: The number of valid bytes in the buffer, i.e., the number of bytes that should be accessed from the buffer for reading/writing.
- Size: The maximum number of available buffers.
- Buffer Address Pointer: The address of the data buffer as seen by the CPU (according to the RAM address space).
- Next Descriptor Address: The address of the next linked-list element in the CPU RAM address space. If the current linked list is the last one, the Eof bit should be 1, and the last descriptor address should be 0.

The Slave's linked-list chain is shown in Figure 26.3-5:

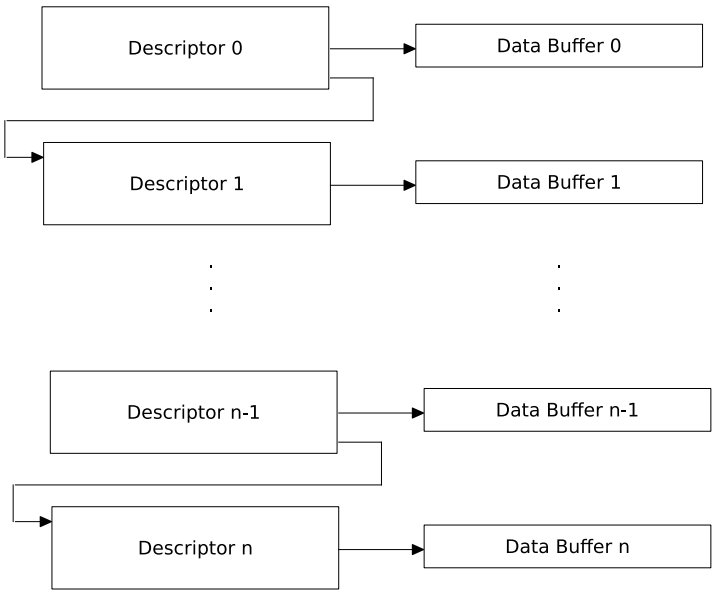



Figure 26.3-5. SDIO Slave Linked List

## 26.3.5 Packet-Sending/-Receiving Procedure

The SDIO Host and Slave devices need to follow specific data transfer procedures to successfully exchange data over the SDIO interface.

### 26.3.5.1 Sending Packets to SDIO Host

The transmission of packets from Slave to Host is initiated by the Slave. The Host will be notified with an interrupt (for detailed information on interrupts, please refer to SDIO protocol). After the Host reads the relevant information from the Slave, it will initiate an SDIO bus transaction accordingly. The whole procedure is illustrated in Figure 26.3-6.

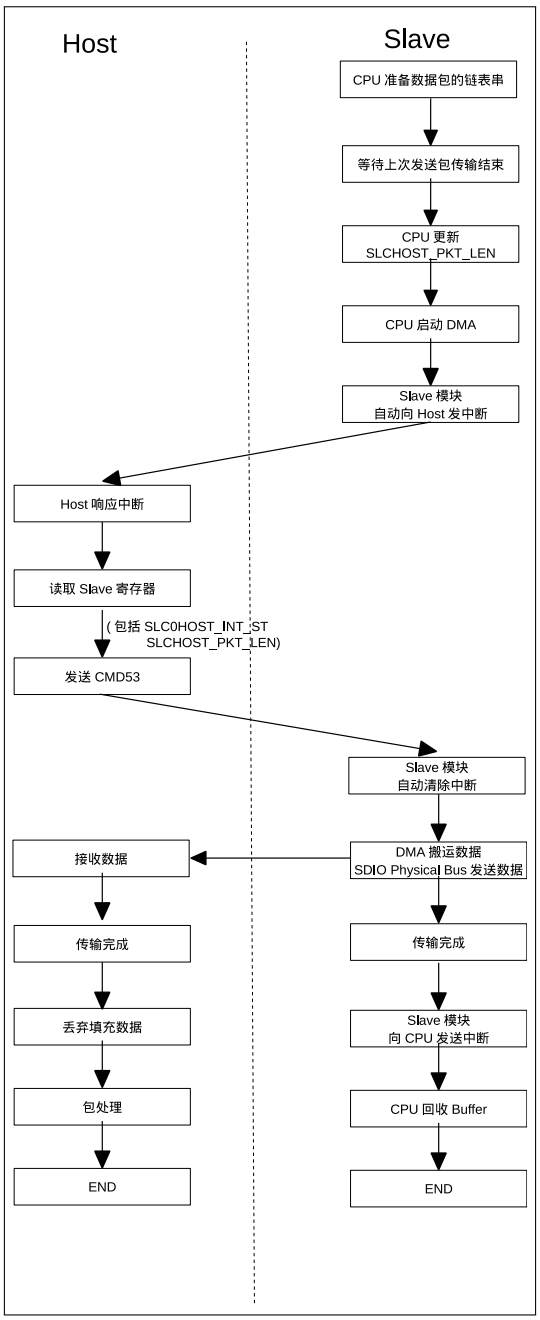



Figure 26.3-6. Packet Sending Procedure (Initiated by Slave)

When the Host is interrupted, it reads relevant information from the Slave by visiting registers SLC0HOST_INT and SLCHOST_PKT_LEN.

- SLC0HOST_INT: Interrupt status register. If the value of SLC0_RX_NEW_PACKET_INT_ST is 1, this indicates that the Slave has a packet to send.
- SLCHOST_PKT_LEN: Packet length accumulator register. The current value minus the value of last time equals the packet length sent this time.

In order to start DMA, the CPU needs to write the low 20 bits of the address of the first linked-list element to the SLC0_RXLINK_ADDR bit of SLC0RX_LINK, then set the SLC0_RXLINK_START bit of SLC0RX_LINK. The DMA will automatically complete the data transfer. Upon completion of the operation, DMA will interrupt the CPU so that the buffer space can be freed or reused.

#### 26.3.5.2 Receiving Packets from SDIO Host

Transmission of packets from Host to Slave is initiated by the Host. The Slave receives data via DMA and stores it in RAM. After transmission is completed, the CPU will be interrupted to process the data. The whole procedure is demonstrated in Figure 26.3-7.

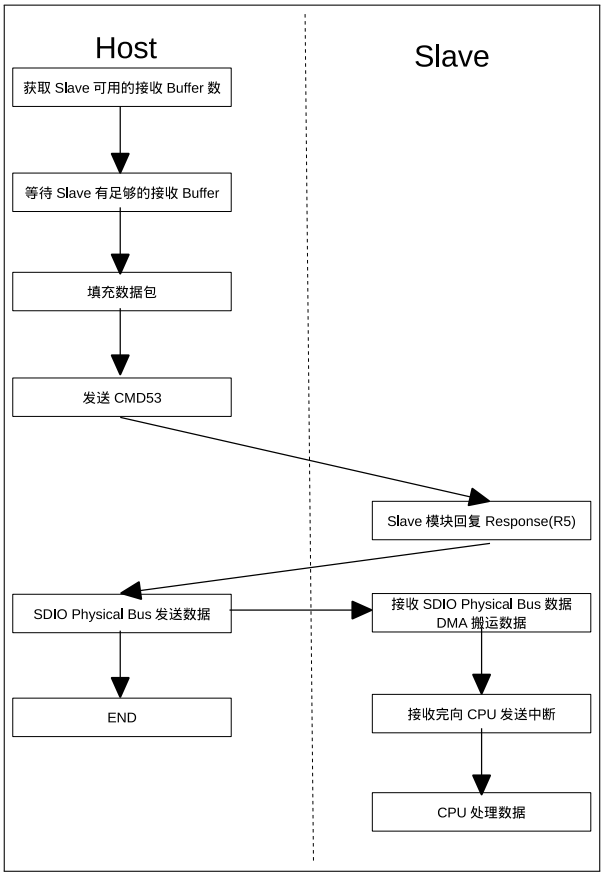



Figure 26.3-7. Packet Receiving Procedure (Initiated by Host)

The Host obtains the number of available receiving buffers from the Slave by accessing register SLCOHOST_TOKEN_RDATA. The Slave CPU should update this value after the receiving DMA linked list is prepared.

HOSTREG_SLCO_TOKEN1 in SLCOHOST_TOKEN_RDATA stores the accumulated number of available buffers.

The Host can figure out the available buffer space, using HOSTREG_SLCO_TOKEN1 minus the number of buffers already used.

If the buffers are not enough, the Host needs to constantly poll the register until there are enough buffers available.

To ensure sufficient receiving buffers, the Slave CPU must constantly load buffers on the receiving linked list. The process is shown in Figure 26.3-8.

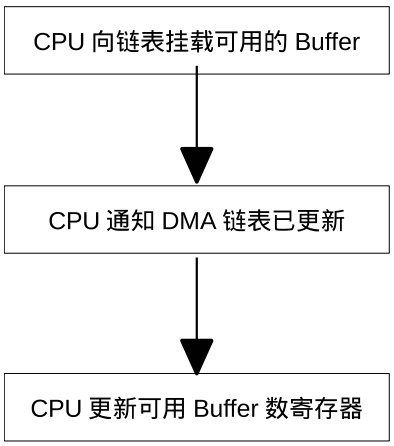



Figure 26.3-8. Loading Receiving Buffer

The CPU first needs to append new buffer segments at the end of the linked list that is being used by DMA and is available for receiving data.

The CPU then needs to notify the DMA that the linked list has been modified. This can be done by setting bit SLCO_TXLINK_RESTART of the SLCOTX_LINK register. Please note that when the CPU initiates DMA to receive packets for the first time, SLCO_TXLINK_RESTART should be set to 1.

Lastly, the CPU refreshes any available buffer information by writing to the SLCOTOKEN1 register.

## 26.3.6 SDIO Bus Timing

The SDIO bus operates at a very high speed and the PCB trace length usually affects signal integrity by introducing latency. To ensure that the timing characteristics conform to the desired bus timing, the SDIO Slave module supports configuration of input sampling clock edge and output driving clock edge.

When the incoming data changes near the rising edge of the clock, the Slave will perform sampling on the falling edge of the clock, or vice versa, as Figure 26.3-9 shows.

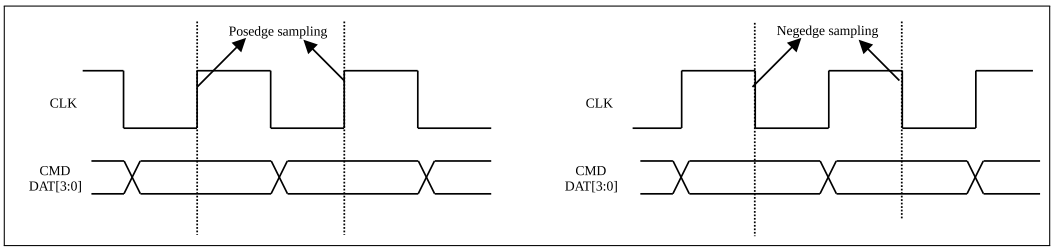



Figure 26.3-9. Sampling Timing Diagram

By default, the MTDO strapping value determines the Slave's sampling edge. However, users can decide the sampling edge by configuring the SLCHOST_CONF_REG register, with priority from high to low: (1) Set SLCHOST_FRC_POS_SAMP to sample the corresponding signal at the rising edge; (2) Set SLCHOST_FRC_NEG_SAMP to sample the corresponding signal at the falling edge.

SLCHOST_FRC_POS_SAMP and SLCHOST_FRC_NEG_SAMP fields are five bits wide. The bits correspond to the CMD line and four DATA lines (0-3). Setting a bit causes the corresponding line to be sampled for input at the rising clock edge or falling clock edge.

The Slave can also select which edge to drive the output lines, in order to accommodate for any latency caused by the physical signal path. The output timing is shown in Figure 26.3-10.

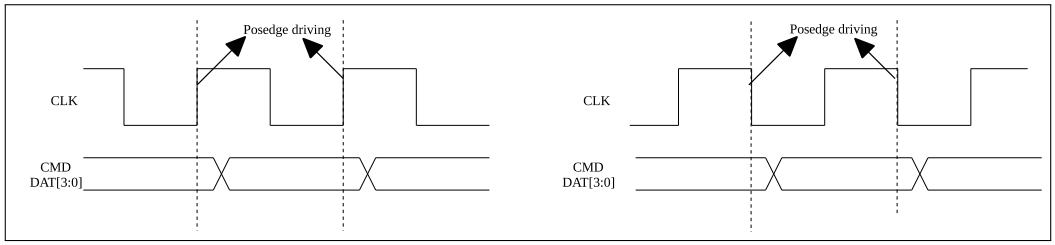



Figure 26.3-10. Output Timing Diagram

By default, the GPIO5 strapping value determines the Slave's output driving edge. However, users can decide the output driving edge by configuring the following registers, with priority from high to low: (1) Set SLCHOST_FRC_SDIO11 in SLCHOST_CONF_REG to output the corresponding signal at the falling clock edge; (2) Set SLCHOST_FRC_SDIO22 in SLCHOST_CONF_REG to output the corresponding signal at the rising clock edge; (3) Set HINF_HIGHSPEED_ENABLE in HINF_CFG_DATA1_REG and SLCHOST_HSPEED_CON_EN in SLCHOST_CONF_REG, then set the EHS (Enable High-Speed) bit in CCCR at the Host side to output the corresponding signal at the rising clock edge.

SLCHOST_FRC_SDIO11 and SLCHOST_FRC_SDIO22 fields are five bits wide. The bits correspond to the CMD line and four DATA lines (0-3). Setting a bit causes the corresponding line to output at the rising clock edge or falling clock edge.

**Notes on priority setting**: The configuration of strapping pins has the lowest priority when controlling the sampling edge or driving edge. The lower-priority configuration takes effect only when the higher-priority configuration is not set. For example, the MTDO strapping value determines the sampling edge only when SCLHOST_FRC_POS_SAMP and SCLHOST_FRC_NEG_SAMP are not set.

## 26.3.7 Interrupt

Host and Slave can interrupt each other via the interrupt vector. Both Host and Slave have eight interrupt vectors. The interrupt is enabled by configuring the interrupt vector register (setting the enable bit to 1). The interrupt vector registers can clear themselves automatically, which means one interrupt at a time and no other configuration is required.

### 26.3.7.1 Host Interrupt

- *SLCOHOST_SLCO_RX_NEW_PACKET_INT* Slave has a packet to send.
- *SLCOHOST_SLCO_TX_OVF_INT* Slave receiving buffer overflow interrupt.
- *SLCOHOST_SLCO_RX_UDF_INT* Slave sending buffer underflow interrupt.
- *SLCOHOST_SLCO_TOHOST_BITn_INT* (*n*: 0 ~ 7) Slave interrupts Host.

### 26.3.7.2 Slave Interrupt

- *SLCOINT_SLCO_RX_DSCR_ERR_INT* Slave sending descriptor error.
- *SLCOINT_SLCO_TX_DSCR_ERR_INT* Slave receiving descriptor error.

- *SLCOINT_SLCO_RX_EOF_INT* Slave sending operation is finished.
- *SLCOINT_SLCO_RX_DONE_INT* A single buffer is sent by Slave.
- *SLCOINT_SLCO_TX_SUC_EOF_INT* Slave receiving operation is finished.
- *SLCOINT_SLCO_TX_DONE_INT* A single buffer is finished during receiving operation.
- *SLCOINT_SLCO_TX_OVF_INT* Slave receiving buffer overflow interrupt.
- *SLCOINT_SLCO_RX_UDF_INT* Slave sending buffer underflow interrupt.
- *SLCOINT_SLCO_TX_START_INT* Slave receiving interrupt initialization.
- *SLCOINT_SLCO_RX_START_INT* Slave sending interrupt initialization.
- *SLCOINT_SLC_FRHOST_BITn_INT* (*n*: 0 ~ 7) Host interrupts Slave.

## 26.4 Register Summary

The addresses in this section are relative to the SDIO Slave base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| **SDIO DMA (SLC) configuration registers** | | | |
| SLCCONF0_REG | SLCCONF0_SLC configuration | 0x3FF58000 | R/W |
| SLCORX_LINK_REG | Transmitting linked list configuration | 0x3FF5803C | R/W |
| SLCOTX_LINK_REG | Receiving linked list configuration | 0x3FF58040 | R/W |
| SLCINTVEC_TOHOST_REG | Interrupt sector for Slave to interrupt Host | 0x3FF5804C | WO |
| SLCOTOKEN1_REG | Number of receiving buffer | 0x3FF58054 | WO |
| SLCCONF1_REG | Control register | 0x3FF58060 | R/W |
| SLC_RX_DSCR_CONF_REG | DMA transmission configuration | 0x3FF58098 | R/W |
| SLCO_LEN_CONF_REG | Length control of the transmitting packets | 0x3FF580E4 | R/W |
| SLCO_LENGTH_REG | Length of the transmitting packets | 0x3FF580E8 | R/W |
| **Interrupt Registers** | | | |
| SLCOINT_RAW_REG | Raw interrupt status | 0x3FF58004 | RO |
| SLCOINT_ST_REG | Interrupt status | 0x3FF58008 | RO |
| SLCOINT_ENA_REG | Interrupt enable | 0x3FF5800C | R/W |
| SLCOINT_CLR_REG | Interrupt clear | 0x3FF58010 | WO |

| Name | Description | Address | Access |
|---|---|---|---|
| **SDIO SLC Host registers** | | | |
| SLCOHOST_TOKEN_RDATA | The accumulated number of Slave's receiving buffers | 0x3FF55044 | RO |
| SLCHOST_PKT_LEN_REG | Length of the transmitting packets | 0x3FF55060 | RO |
| SLCHOST_CONF_W0_REG | Host and Slave communication register0 | 0x3FF5506C | R/W |
| SLCHOST_CONF_W1_REG | Host and Slave communication register1 | 0x3FF55070 | R/W |
| SLCHOST_CONF_W2_REG | Host and Slave communication register2 | 0x3FF55074 | R/W |

| | | | |
|---|---|---|---|
| SLCHOST_CONF_W3_REG | Host and Slave communication register3 | 0x3FF55078 | R/W |
| SLCHOST_CONF_W4_REG | Host and Slave communication register4 | 0x3FF5507C | R/W |
| SLCHOST_CONF_W6_REG | Host and Slave communication register6 | 0x3FF55088 | R/W |
| SLCHOST_CONF_W8_REG | Host and Slave communication register8 | 0x3FF5509C | R/W |
| SLCHOST_CONF_W9_REG | Host and Slave communication register9 | 0x3FF550A0 | R/W |
| SLCHOST_CONF_W10_REG | Host and Slave communication register10 | 0x3FF550A4 | R/W |
| SLCHOST_CONF_W11_REG | Host and Slave communication register11 | 0x3FF550A8 | R/W |
| SLCHOST_CONF_W12_REG | Host and Slave communication register12 | 0x3FF550AC | R/W |
| SLCHOST_CONF_W13_REG | Host and Slave communication register13 | 0x3FF550B0 | R/W |
| SLCHOST_CONF_W14_REG | Host and Slave communication register14 | 0x3FF550B4 | R/W |
| SLCHOST_CONF_W15_REG | Host and Slave communication register15 | 0x3FF550B8 | R/W |
| SLCHOST_CONF_REG | Edge configuration | 0x3FF551F0 | R/W |
| **Interrupt Registers** | | | |
| SLCOHOST_INT_RAW_REG | Raw interrupt | 0x3FF55000 | RO |
| SLCOHOST_INT_ST_REG | Masked interrupt status | 0x3FF55058 | RO |
| SLCOHOST_INT_CLR_REG | Interrupt clear | 0x3FF550D4 | WO |
| SLCOHOST_FUNC1_INT_ENA_REG | Interrupt enable | 0x3FF550DC | R/W |
| SLCHOST_CONF_W7_REG | Interrupt vector for Host to interrupt Slave | 0x3FF5508C | WO |

| Name | Description | Address | Access |
|---|---|---|---|
| **SDIO HINF registers** | | | |
| HINF_CFG_DATA1_REG | SDIO specification configuration | 0x3FF4B004 | R/W |

## 26.5 SLC Registers

The addresses in this section are relative to the SDIO Slave base address (0x3FF5_8000) provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 26.4 *Register Summary*.

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

**Register 26.1. SLCCONF0_REG (0x0)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | SLCCONF0_SLC0_TOKEN_AUTO_CLR | (reserved) | | | | | | | SLCCONF0_SLC0_RX_AUTO_WRBACK | SLCCONF0_SLC0_RX_LOOP_TEST | SLCCONF0_SLC0_TX_LOOP_TEST | (reserved) | | SLCCONF0_SLC0_RX_RST | SLCCONF0_SLC0_TX_RST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | Reset |

**SLCCONF0_SLC0_TOKEN_AUTO_CLR** Please initialize to 0. Do not modify it. (R/W)

**SLCCONF0_SLC0_RX_AUTO_WRBACK** Allows changing the owner bit of the transmitting buffer's linked list when transmitting data. (R/W)

**SLCCONF0_SLC0_RX_LOOP_TEST** Loop around when the slave buffer finishes sending packets. When set to 1, hardware will not change the owner bit in the linked list. (R/W)

**SLCCONF0_SLC0_TX_LOOP_TEST** Loop around when the slave buffer finishes receiving packets. When set to 1, hardware will not change the owner bit in the linked list. (R/W)

**SLCCONF0_SLC0_RX_RST** Set this bit to reset the transmitting FSM. (R/W)

**SLCCONF0_SLC0_TX_RST** Set this bit to reset the receiving FSM. (R/W)

**Register 26.2. SLCOINT_RAW_REG (0x4)**

| 31–27 | 26–21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13–12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (reserved) | SLCOINT_SLCO_RX_DSCR_ERR_INT_RAW | SLCOINT_SLCO_TX_DSCR_ERR_INT_RAW | (reserved) | SLCOINT_SLCO_RX_EOF_INT_RAW | SLCOINT_SLCO_RX_DONE_INT_RAW | SLCOINT_SLCO_TX_SUC_EOF_INT_RAW | SLCOINT_SLCO_TX_DONE_INT_RAW | (reserved) | SLCOINT_SLCO_TX_OVF_INT_RAW | SLCOINT_SLCO_RX_UDF_INT_RAW | SLCOINT_SLCO_TX_START_INT_RAW | SLCOINT_SLCO_RX_START_INT_RAW | SLCOINT_SLC_FRHOST_BIT7_INT_RAW | SLCOINT_SLC_FRHOST_BIT6_INT_RAW | SLCOINT_SLC_FRHOST_BIT5_INT_RAW | SLCOINT_SLC_FRHOST_BIT4_INT_RAW | SLCOINT_SLC_FRHOST_BIT3_INT_RAW | SLCOINT_SLC_FRHOST_BIT2_INT_RAW | SLCOINT_SLC_FRHOST_BIT1_INT_RAW | SLCOINT_SLC_FRHOST_BIT0_INT_RAW | |
| 0x00 | 0 0 0 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 |

**SLCOINT_SLCO_RX_DSCR_ERR_INT_RAW** The raw interrupt bit for Slave sending descriptor error (RO)

**SLCOINT_SLCO_TX_DSCR_ERR_INT_RAW** The raw interrupt bit for Slave receiving descriptor error. (RO)

**SLCOINT_SLCO_RX_EOF_INT_RAW** The interrupt mark bit for Slave sending operation finished. (RO)

**SLCOINT_SLCO_RX_DONE_INT_RAW** The raw interrupt bit to mark single buffer as sent by Slave. (RO)

**SLCOINT_SLCO_TX_SUC_EOF_INT_RAW** The raw interrupt bit to mark Slave receiving operation as finished. (RO)

**SLCOINT_SLCO_TX_DONE_INT_RAW** The raw interrupt bit to mark a single buffer as finished during Slave receiving operation. (RO)

**SLCOINT_SLCO_TX_OVF_INT_RAW** The raw interrupt bit to mark Slave receiving buffer overflow. (RO)

**SLCOINT_SLCO_RX_UDF_INT_RAW** The raw interrupt bit for Slave sending buffer underflow. (RO)

**SLCOINT_SLCO_TX_START_INT_RAW** The raw interrupt bit for registering Slave receiving initialization interrupt. (RO)

**SLCOINT_SLCO_RX_START_INT_RAW** The raw interrupt bit to mark Slave sending initialization interrupt. (RO)

**SLCOINT_SLC_FRHOST_BIT7_INT_RAW** The interrupt mark bit 7 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT6_INT_RAW** The interrupt mark bit 6 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT5_INT_RAW** The interrupt mark bit 5 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT4_INT_RAW** The interrupt mark bit 4 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT3_INT_RAW** The interrupt mark bit 3 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT2_INT_RAW** The interrupt mark bit 2 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT1_INT_RAW** The interrupt mark bit 1 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT0_INT_RAW** The interrupt mark bit 0 for Host to interrupt Slave. (RO)

**Register 26.3. SLCOINT_ST_REG (0x8)**

| (reserved) | (reserved) | SLCOINT_SLC0_RX_DSCR_ERR_INT_ST | SLCOINT_SLC0_TX_DSCR_ERR_INT_ST | (reserved) | SLCOINT_SLC0_RX_EOF_INT_ST | SLCOINT_SLC0_RX_DONE_INT_ST | SLCOINT_SLC0_TX_SUC_EOF_INT_ST | SLCOINT_SLC0_TX_DONE_INT_ST | (reserved) | SLCOINT_SLC0_TX_OVF_INT_ST | SLCOINT_SLC0_RX_UDF_INT_ST | SLCOINT_SLC0_TX_START_INT_ST | SLCOINT_SLC0_RX_START_INT_ST | SLCOINT_SLC_FRHOST_BIT7_INT_ST | SLCOINT_SLC_FRHOST_BIT6_INT_ST | SLCOINT_SLC_FRHOST_BIT5_INT_ST | SLCOINT_SLC_FRHOST_BIT4_INT_ST | SLCOINT_SLC_FRHOST_BIT3_INT_ST | SLCOINT_SLC_FRHOST_BIT2_INT_ST | SLCOINT_SLC_FRHOST_BIT1_INT_ST | SLCOINT_SLC_FRHOST_BIT0_INT_ST | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 0 0 0 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 |

**SLCOINT_SLC0_RX_DSCR_ERR_INT_ST** The interrupt status bit for Slave sending descriptor error. (RO)

**SLCOINT_SLC0_TX_DSCR_ERR_INT_ST** The interrupt status bit for Slave receiving descriptor error. (RO)

**SLCOINT_SLC0_RX_EOF_INT_ST** The interrupt status bit for finished Slave sending operation. (RO)

**SLCOINT_SLC0_RX_DONE_INT_ST** The interrupt status bit for finished Slave sending operation. (RO)

**SLCOINT_SLC0_TX_SUC_EOF_INT_ST** The interrupt status bit for marking Slave receiving operation as finished. (RO)

**SLCOINT_SLC0_TX_DONE_INT_ST** The interrupt status bit for marking a single buffer as finished during the receiving operation. (RO)

**SLCOINT_SLC0_TX_OVF_INT_ST** The interrupt status bit for Slave receiving overflow interrupt. (RO)

**SLCOINT_SLC0_RX_UDF_INT_ST** The interrupt status bit for Slave sending buffer underflow. (RO)

**SLCOINT_SLC0_TX_START_INT_ST** The interrupt status bit for Slave receiving interrupt initialization. (RO)

**SLCOINT_SLC0_RX_START_INT_ST** The interrupt status bit for Slave sending interrupt initialization. (RO)

**SLCOINT_SLC_FRHOST_BIT7_INT_ST** The interrupt status bit 7 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT6_INT_ST** The interrupt status bit 6 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT5_INT_ST** The interrupt status bit 5 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT4_INT_ST** The interrupt status bit 4 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT3_INT_ST** The interrupt status bit 3 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT2_INT_ST** The interrupt status bit 2 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT1_INT_ST** The interrupt status bit 1 for Host to interrupt Slave. (RO)

**SLCOINT_SLC_FRHOST_BIT0_INT_ST** The interrupt status bit 0 for Host to interrupt Slave. (RO)

**Register 26.4. SLCOINT_ENA_REG (0xC)**

| 31–27 | 26–21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13–12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (reserved) | SLCOINT_SLCO_RX_DSCR_ERR_INT_ENA | SLCOINT_SLCO_TX_DSCR_ERR_INT_ENA | (reserved) | SLCOINT_SLCO_RX_EOF_INT_ENA | SLCOINT_SLCO_RX_DONE_INT_ENA | SLCOINT_SLCO_TX_SUC_EOF_INT_ENA | SLCOINT_SLCO_TX_DONE_INT_ENA | (reserved) | SLCOINT_SLCO_TX_OVF_INT_ENA | SLCOINT_SLCO_RX_UDF_INT_ENA | SLCOINT_SLCO_TX_START_INT_ENA | SLCOINT_SLCO_RX_START_INT_ENA | SLCOINT_SLC_FRHOST_BIT7_INT_ENA | SLCOINT_SLC_FRHOST_BIT6_INT_ENA | SLCOINT_SLC_FRHOST_BIT5_INT_ENA | SLCOINT_SLC_FRHOST_BIT4_INT_ENA | SLCOINT_SLC_FRHOST_BIT3_INT_ENA | SLCOINT_SLC_FRHOST_BIT2_INT_ENA | SLCOINT_SLC_FRHOST_BIT1_INT_ENA | SLCOINT_SLC_FRHOST_BIT0_INT_ENA | |
| 0x00 | 0 0 0 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 |

**SLCOINT_SLCO_RX_DSCR_ERR_INT_ENA** The interrupt enable bit for Slave sending linked list descriptor error. (R/W)

**SLCOINT_SLCO_TX_DSCR_ERR_INT_ENA** The interrupt enable bit for Slave receiving linked list descriptor error. (R/W)

**SLCOINT_SLCO_RX_EOF_INT_ENA** The interrupt enable bit for Slave sending operation completion. (R/W)

**SLCOINT_SLCO_RX_DONE_INT_ENA** The interrupt enable bit for single buffer's sent interrupt, in Slave sending mode. (R/W)

**SLCOINT_SLCO_TX_SUC_EOF_INT_ENA** The interrupt enable bit for Slave receiving operation completion. (R/W)

**SLCOINT_SLCO_TX_DONE_INT_ENA** The interrupt enable bit for single buffer's full event, in Slave receiving mode. (R/W)

**SLCOINT_SLCO_TX_OVF_INT_ENA** The interrupt enable bit for Slave receiving buffer overflow. (R/W)

**SLCOINT_SLCO_RX_UDF_INT_ENA** The interrupt enable bit for Slave sending buffer underflow. (R/W)

**SLCOINT_SLCO_TX_START_INT_ENA** The interrupt enable bit for Slave receiving operation initialization. (R/W)

**SLCOINT_SLCO_RX_START_INT_ENA** The interrupt enable bit for Slave sending operation initialization. (R/W)

**SLCOINT_SLC_FRHOST_BIT7_INT_ENA** The interrupt enable bit 7 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT6_INT_ENA** The interrupt enable bit 6 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT5_INT_ENA** The interrupt enable bit 5 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT4_INT_ENA** The interrupt enable bit 4 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT3_INT_ENA** The interrupt enable bit 3 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT2_INT_ENA** The interrupt enable bit 2 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT1_INT_ENA** The interrupt enable bit 1 for Host to interrupt Slave. (R/W)

**SLCOINT_SLC_FRHOST_BIT0_INT_ENA** The interrupt enable bit 0 for Host to interrupt Slave. (R/W)

Register 26.5. SLCOINT_CLR_REG (0x10)

| (reserved) | (reserved) | SLCOINT_SLCO_RX_DSCR_ERR_INT_CLR | SLCOINT_SLCO_TX_DSCR_ERR_INT_CLR | (reserved) | SLCOINT_SLCO_RX_EOF_INT_CLR | SLCOINT_SLCO_RX_DONE_INT_CLR | SLCOINT_SLCO_TX_SUC_EOF_INT_CLR | SLCOINT_SLCO_TX_DONE_INT_CLR | (reserved) | SLCOINT_SLCO_TX_OVF_INT_CLR | SLCOINT_SLCO_RX_UDF_INT_CLR | SLCOINT_SLCO_TX_START_INT_CLR | SLCOINT_SLCO_RX_START_INT_CLR | SLCOINT_SLC_FRHOST_BIT7_INT_CLR | SLCOINT_SLC_FRHOST_BIT6_INT_CLR | SLCOINT_SLC_FRHOST_BIT5_INT_CLR | SLCOINT_SLC_FRHOST_BIT4_INT_CLR | SLCOINT_SLC_FRHOST_BIT3_INT_CLR | SLCOINT_SLC_FRHOST_BIT2_INT_CLR | SLCOINT_SLC_FRHOST_BIT1_INT_CLR | SLCOINT_SLC_FRHOST_BIT0_INT_CLR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 0 0 0 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 |

**SLCOINT_SLCO_RX_DSCR_ERR_INT_CLR** Interrupt clear bit for Slave sending linked list descriptor error. (WO)

**SLCOINT_SLCO_TX_DSCR_ERR_INT_CLR** Interrupt clear bit for Slave receiving linked list descriptor error. (WO)

**SLCOINT_SLCO_RX_EOF_INT_CLR** Interrupt clear bit for Slave sending operation completion. (WO)

**SLCOINT_SLCO_RX_DONE_INT_CLR** Interrupt clear bit for single buffer's sent interrupt, in Slave sending mode. (WO)

**SLCOINT_SLCO_TX_SUC_EOF_INT_CLR** Interrupt clear bit for Slave receiving operation completion. (WO)

**SLCOINT_SLCO_TX_DONE_INT_CLR** Interrupt clear bit for single buffer's full event, in Slave receiving mode. (WO)

**SLCOINT_SLCO_TX_OVF_INT_CLR** Set this bit to clear the Slave receiving overflow interrupt. (WO)

**SLCOINT_SLCO_RX_UDF_INT_CLR** Set this bit to clear the Slave sending underflow interrupt. (WO)

**SLCOINT_SLCO_TX_START_INT_CLR** Set this bit to clear the interrupt for Slave receiving operation initialization. (WO)

**SLCOINT_SLCO_RX_START_INT_CLR** Set this bit to clear the interrupt for Slave sending operation initialization. (WO)

**SLCOINT_SLC_FRHOST_BIT7_INT_CLR** Set this bit to clear the SLCOINT_SLC_FRHOST_BIT7_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT6_INT_CLR** Set this bit to clear the SLCOINT_SLC_FRHOST_BIT6_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT5_INT_CLR** Set this bit to clear the SLCOINT_SLC_FRHOST_BIT5_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT4_INT_CLR** Set this bit to clear the SLCOINT_SLC_FRHOST_BIT4_INT interrupt. (WO)

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

### Register 26.5. SLCOINT_CLR_REG (0x10)

Continued from the previous page...

**SLCOINT_SLC_FRHOST_BIT3_INT_CLR** Set this bit to clear SLCOINT_SLC_FRHOST_BIT3_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT2_INT_CLR** Set this bit to clear SLCOINT_SLC_FRHOST_BIT2_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT1_INT_CLR** Set this bit to clear SLCOINT_SLC_FRHOST_BIT1_INT interrupt. (WO)

**SLCOINT_SLC_FRHOST_BIT0_INT_CLR** Set this bit to clear SLCOINT_SLC_FRHOST_BIT0_INT interrupt. (WO)

### Register 26.6. SLCORX_LINK_REG (0x3C)

| (reserved) | SLCORX_SLC0_RXLINK_RESTART | SLCORX_SLC0_RXLINK_START | SLCORX_SLC0_RXLINK_STOP | (reserved) | SLCORX_SLC0_RXLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**SLCORX_SLC0_RXLINK_RESTART** Set this bit to restart and continue the linked list operation for sending packets. (R/W)

**SLCORX_SLC0_RXLINK_START** Set this bit to start the linked list operation for sending packets. Sending will start from the address indicated by SLC0_RXLINK_ADDR. (R/W)

**SLCORX_SLC0_RXLINK_STOP** Set this bit to stop the linked list operation. (R/W)

**SLCORX_SLC0_RXLINK_ADDR** The lowest 20 bits in the initial address of Slave's sending linked list. (R/W)

Register 26.7. SLCOTX_LINK_REG (0x40)

| (reserved) | SLCOTX_SLCO_TXLINK_RESTART | SLCOTX_SLCO_TXLINK_START | SLCOTX_SLCO_TXLINK_STOP | (reserved) | SLCOTX_SLCO_TXLINK_ADDR | |
|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 … 20 | 19 … 0 | |
| 0 | 0 | 0 | 0 | 0 0 0 0 0 0 0 0 | 0x000000 | Reset |

**SLCOTX_SLCO_TXLINK_RESTART** Set this bit to restart and continue the linked list operation for receiving packets. (R/W)

**SLCOTX_SLCO_TXLINK_START** Set this bit to start the linked list operation for receiving packets. Receiving will start from the address indicated by SLCO_TXLINK_ADDR. (R/W)

**SLCOTX_SLCO_TXLINK_STOP** Set this bit to stop the linked list operation for receiving packets. (R/W)

**SLCOTX_SLCO_TXLINK_ADDR** The lowest 20 bits in the initial address of Slave's receiving linked list. (R/W)

Register 26.8. SLCINTVEC_TOHOST_REG (0x4C)

| (reserved) | (reserved) | (reserved) | SLCINTVEC_SLCO_TOHOST_INTVEC | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0 0 0 0 0 0 0 0 | 0x000 | 0x000 | Reset |

**SLCINTVEC_SLCO_TOHOST_INTVEC** The interrupt vector for Slave to interrupt Host. (WO)

### Register 26.9. SLCOTOKEN1_REG (0x54)

| (reserved) | SLCOTOKEN1_SLCO_TOKEN1 | (reserved) | SLCOTOKEN1_SLCO_TOKEN1_INC_MORE | (reserved) | SLCOTOKEN1_SLCO_TOKEN1_WDATA | |
|---|---|---|---|---|---|---|
| 31 28 | 27 16 | 15 | 14 | 13 12 | 11 0 | |
| 0x00 | 0x0000 | 0 | 0 | 0 0 | 0x0000 | Reset |

**SLCOTOKEN1_SLCO_TOKEN1** The accumulated number of buffers for receiving packets. (RO)

**SLCOTOKEN1_SLCO_TOKEN1_INC_MORE** Set this bit to add the value of SLCOTOKEN1_SLCO_TOKEN1_WDATA to that of SLCOTOKEN1_SLCO_TOKEN1. (WO)

**SLCOTOKEN1_SLCO_TOKEN1_WDATA** The number of available receiving buffers. (WO)

### Register 26.10. SLCCONF1_REG (0x60)

| (reserved) | (reserved) | (reserved) | SLCCONF1_SLCO_RX_STITCH_EN | SLCCONF1_SLCO_TX_STITCH_EN | SLCCONF1_SLCO_LEN_AUTO_CLR | (reserved) | (reserved) | |
|---|---|---|---|---|---|---|---|---|
| 31 23 | 22 16 | 15 7 | 6 | 5 | 4 | 3 | 2 0 | |
| 0x000 | 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 | 1 | 1 | 1 | 1 | 000 | Reset |

**SLCCONF1_SLCO_RX_STITCH_EN** Please initialize to 0. Do not modify it. (R/W)

**SLCCONF1_SLCO_TX_STITCH_EN** Please initialize to 0. Do not modify it. (R/W)

**SLCCONF1_SLCO_LEN_AUTO_CLR** Please initialize to 0. Do not modify it. (R/W)

Register 26.11. SLC_RX_DSCR_CONF_REG (0x98)

| (reserved) | SLC_SLC0_TOKEN_NO_REPLACE | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**SLC_SLC0_TOKEN_NO_REPLACE** Please initialize to 1. Do not modify it. (R/W)

Register 26.12. SLC0_LEN_CONF_REG (0xE4)

| (reserved) | (reserved) | SLC0_LEN_INC_MORE | (reserved) | SLC0_LEN_WDATA | |
|---|---|---|---|---|---|
| 31 … 29 | 28 … 23 | 22 | 21 … 20 | 19 … 0 | |
| 0x0 | 0 0 0 0 0 0 | 0 | 0 0 | 0x000000 | Reset |

**SLC0_LEN_INC_MORE** Set this bit to add the value of SLC0_LEN to that of SLC0_LEN_WDATA. (WO)

**SLC0_LEN_WDATA** The packet length sent. (WO)

Register 26.13. SLC0_LENGTH_REG (0xE8)

| (reserved) | SLC0_LEN | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0x0000 | 0x000000 | Reset |

**SLC0_LEN** Indicates the packet length sent by the Slave. (RO)

## 26.6 SLC Host Registers

The addresses in this section are relative to the SDIO Slave base address (0x3FF5_5000) provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 26.4 *Register Summary*.

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

**Register 26.14. SLCOHOST_TOKEN_RDATA (0x44)**

| (reserved) | HOSTREG_SLCO_TOKEN1 | (reserved) | |
|---|---|---|---|
| 31 28 | 27 16 | 15 0 | |
| 0x000 | 0x000 | 0x000 | Reset |

**HOSTREG_SLCO_TOKEN1** The accumulated number of Slave's receiving buffers. (RO)

**Register 26.15. SLCOHOST_INT_RAW_REG (0x50)**

| (reserved) | (reserved) | SLCOHOST_SLC0_RX_NEW_PACKET_INT_RAW | (reserved) | SLCOHOST_SLC0_TX_OVF_INT_RAW | SLCOHOST_SLC0_RX_UDF_INT_RAW | (reserved) | SLCOHOST_SLC0_TOHOST_BIT7_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT6_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT5_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT4_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT3_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT2_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT1_INT_RAW | SLCOHOST_SLC0_TOHOST_BIT0_INT_RAW | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 24 | 23 | 22 18 | 17 | 16 | 15 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 0 0 | 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 |

**SLCOHOST_SLC0_RX_NEW_PACKET_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_RX_NEW_PACKET_INT interrupt. (RO)

**SLCOHOST_SLC0_TX_OVF_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TX_OVF_INT interrupt. (RO)

**SLCOHOST_SLC0_RX_UDF_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_RX_UDF_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT7_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT7_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT6_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT6_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT5_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT5_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT4_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT4_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT3_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT3_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT2_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT2_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT1_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT1_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT0_INT_RAW** The raw interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT0_INT interrupt. (RO)

**Register 26.16. SLCOHOST_INT_ST_REG (0x58)**

| (reserved) | (reserved) | SLCOHOST_SLC0_RX_NEW_PACKET_INT_ST | (reserved) | SLCOHOST_SLC0_TX_OVF_INT_ST | SLCOHOST_SLC0_RX_UDF_INT_ST | (reserved) | SLCOHOST_SLC0_TOHOST_BIT7_INT_ST | SLCOHOST_SLC0_TOHOST_BIT6_INT_ST | SLCOHOST_SLC0_TOHOST_BIT5_INT_ST | SLCOHOST_SLC0_TOHOST_BIT4_INT_ST | SLCOHOST_SLC0_TOHOST_BIT3_INT_ST | SLCOHOST_SLC0_TOHOST_BIT2_INT_ST | SLCOHOST_SLC0_TOHOST_BIT1_INT_ST | SLCOHOST_SLC0_TOHOST_BIT0_INT_ST | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 24 | 23 | 22 18 | 17 | 16 | 15 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 0 0 | 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 |

**SLCOHOST_SLC0_RX_NEW_PACKET_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_RX_NEW_PACKET_INT interrupt. (RO)

**SLCOHOST_SLC0_TX_OVF_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TX_OVF_INT interrupt. (RO)

**SLCOHOST_SLC0_RX_UDF_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_RX_UDF_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT7_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT7_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT6_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT6_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT5_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT5_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT4_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT4_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT3_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT3_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT2_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT2_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT1_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT1_INT interrupt. (RO)

**SLCOHOST_SLC0_TOHOST_BIT0_INT_ST** The masked interrupt status bit for the SLCOHOST_SLC0_TOHOST_BIT0_INT interrupt. (RO)

**Register 26.17. SLCHOST_PKT_LEN_REG (0x60)**

| SLCHOST_HOSTREG_SLC0_LEN_CHECK | SLCHOST_HOSTREG_SLC0_LEN | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0x000 | 0x000 | Reset |

**SLCHOST_HOSTREG_SLC0_LEN_CHECK** Its value is HOSTREG_SLC0_LEN[9:0] plus HOSTREG_SLC0_LEN[19:10]. (RO)

**SLCHOST_HOSTREG_SLC0_LEN** The accumulated value of the data length sent by the Slave. The value gets updated only when the Host reads it.

**Register 26.18. SLCHOST_CONF_W0_REG (0x6C)**

| SLCHOST_CONF3 | SLCHOST_CONF2 | SLCHOST_CONF1 | SLCHOST_CONF0 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF3** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF2** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF1** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF0** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 26.19. SLCHOST_CONF_W1_REG (0x70)

| SLCHOST_CONF7 | SLCHOST_CONF6 | SLCHOST_CONF5 | SLCHOST_CONF4 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF7** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF6** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF5** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF4** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 26.20. SLCHOST_CONF_W2_REG (0x74)

| SLCHOST_CONF11 | SLCHOST_CONF10 | SLCHOST_CONF9 | SLCHOST_CONF8 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF11** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF10** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF9** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF8** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**Register 26.21. SLCHOST_CONF_W3_REG (0x78)**

| SLCHOST_CONF15 | SLCHOST_CONF14 | (reserved) | (reserved) | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x0 | 0xc0 | Reset |

**SLCHOST_CONF15** The information interaction register between Host and Slave. Both Host and Slave can be read from and written to this. (R/W)

**SLCHOST_CONF14** The information interaction register between Host and Slave. Both Host and Slave can be read from and written to this. (R/W)

**Register 26.22. SLCHOST_CONF_W4_REG (0x7C)**

| SLCHOST_CONF19 | SLCHOST_CONF18 | (reserved) | (reserved) | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x1 | 0xff | Reset |

**SLCHOST_CONF19** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF18** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 26.23. SLCHOST_CONF_W6_REG (0x88)

| SLCHOST_CONF27 | SLCHOST_CONF26 | SLCHOST_CONF25 | SLCHOST_CONF24 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF27** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF26** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF25** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF24** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 26.24. SLCHOST_CONF_W8_REG (0x9C)

| SLCHOST_CONF35 | SLCHOST_CONF34 | SLCHOST_CONF33 | SLCHOST_CONF32 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF35** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF34** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF33** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF32** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.25. SLCHOST_CONF_W9_REG (0xA0)

| SLCHOST_CONF39 | SLCHOST_CONF38 | SLCHOST_CONF37 | SLCHOST_CONF36 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF39** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF38** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF37** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF36** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.26. SLCHOST_CONF_W10_REG (0xA4)

| SLCHOST_CONF43 | SLCHOST_CONF42 | SLCHOST_CONF41 | SLCHOST_CONF40 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF43** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF42** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF41** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF40** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.27. SLCHOST_CONF_W11_REG (0xA8)

| SLCHOST_CONF47 | SLCHOST_CONF46 | SLCHOST_CONF45 | SLCHOST_CONF44 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF47** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF46** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF45** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF44** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.28. SLCHOST_CONF_W12_REG (0xAC)

| SLCHOST_CONF51 | SLCHOST_CONF50 | SLCHOST_CONF49 | SLCHOST_CONF48 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF51** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF50** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF49** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF48** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.29. SLCHOST_CONF_W13_REG (0xB0)

| SLCHOST_CONF55 | SLCHOST_CONF54 | SLCHOST_CONF53 | SLCHOST_CONF52 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF55** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF54** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF53** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF52** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

### Register 26.30. SLCHOST_CONF_W14_REG (0xB4)

| SLCHOST_CONF59 | SLCHOST_CONF58 | SLCHOST_CONF57 | SLCHOST_CONF56 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF59** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF58** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF57** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF56** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**Register 26.31. SLCHOST_CONF_W15_REG (0xB8)**

| SLCHOST_CONF63 | SLCHOST_CONF62 | SLCHOST_CONF61 | SLCHOST_CONF60 | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**SLCHOST_CONF63** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF62** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF61** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**SLCHOST_CONF60** The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

**Register 26.32. SLC0HOST_INT_CLR_REG (0xD4)**

| 31–26 | 25 | 24 | 23 | 22–18 | 17 | 16 | 15–8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | (reserved) | | SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR | (reserved) | SLC0HOST_SLC0_TX_OVF_INT_CLR | SLC0HOST_SLC0_RX_UDF_INT_CLR | (reserved) | SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR | SLC0HOST_SLC0_TOHOST_BIT0_INT_CLR | |
| 0x00 | 0 | 0 | 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 |

**SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_RX_NEW_PACKET_INT interrupt. (WO)

**SLC0HOST_SLC0_TX_OVF_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TX_OVF_INT interrupt. (WO)

**SLC0HOST_SLC0_RX_UDF_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_RX_UDF_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT7_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT6_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT5_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT4_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT3_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT2_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT1_INT interrupt. (WO)

**SLC0HOST_SLC0_TOHOST_BIT0_INT_CLR** Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT0_INT interrupt. (WO)

### Register 26.33. SLCOHOST_FUNC1_INT_ENA_REG (0xDC)

| (reserved) | (reserved) | SLCOHOST_FN1_SLCO_RX_NEW_PACKET_INT_ENA | (reserved) | SLCOHOST_FN1_SLCO_TX_OVF_INT_ENA | SLCOHOST_FN1_SLCO_RX_UDF_INT_ENA | (reserved) | SLCOHOST_FN1_SLCO_TOHOST_BIT7_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT6_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT5_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT4_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT3_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT2_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT1_INT_ENA | SLCOHOST_FN1_SLCO_TOHOST_BIT0_INT_ENA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 24 | 23 | 22 18 | 17 | 16 | 15 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 0 0 | 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 |

**SLCOHOST_FN1_SLCO_RX_NEW_PACKET_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_RX_NEW_PACKET_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TX_OVF_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TX_OVF_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_RX_UDF_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_RX_UDF_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT7_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT7_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT6_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT6_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT5_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT5_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT4_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT4_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT3_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT3_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT2_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT2_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT1_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT1_INT interrupt. (R/W)

**SLCOHOST_FN1_SLCO_TOHOST_BIT0_INT_ENA** The interrupt enable bit for the SLCOHOST_FN1_SLCO_TOHOST_BIT0_INT interrupt. (R/W)

Register 26.34. SLCHOST_CONF_W7_REG (0x8C)

| SLCHOST_CONF31 | (reserved) | SLCHOST_CONF29 | (reserved) | |
|---|---|---|---|---|
| 31 – 24 | 23 – 16 | 15 – 8 | 7 – 0 | |
| 0 0 0 0 0 0 0 0 | 0x000 | 0 0 0 0 0 0 0 0 | 0x000 | Reset |

**SLCHOST_CONF31** The interrupt vector used by Host to interrupt Slave. This bit will not be cleared automatically. (WO)

**SLCHOST_CONF29** The interrupt vector used by Host to interrupt Slave. This bit will not be cleared automatically. (WO)

Register 26.35. SLCHOST_CONF_REG (0x1F0)

| (reserved) | SLCHOST_HSPEED_CON_EN | (reserved) | SLCHOST_FRC_POS_SAMP | SLCHOST_FRC_NEG_SAMP | SLCHOST_FRC_SDIO20 | SLCHOST_FRC_SDIO11 | |
|---|---|---|---|---|---|---|---|
| 31 – 28 | 27 | 26 – 20 | 19 – 15 | 14 – 10 | 9 – 5 | 4 – 0 | |
| 0 0 0 0 | 0 | 0 0 0 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 |

**SLCHOST_HSPEED_CON_EN** Set this bit and HINF_HIGHSPEED_ENABLE, then set the EHS (Enable High-Speed) bit in CCCR at the Host side to output the corresponding signal at the rising clock edge. (R/W)

**SLCHOST_FRC_POS_SAMP** Set this bit to sample the corresponding signal at the rising clock edge. (R/W)

**SLCHOST_FRC_NEG_SAMP** Set this bit to sample the corresponding signal at the falling clock edge. (R/W)

**SLCHOST_FRC_SDIO20** Set this bit to output the corresponding signal at the rising clock edge. (R/W)

**SLCHOST_FRC_SDIO11** Set this bit to output the corresponding signal at the falling clock edge. (R/W)

## 26.7 HINF Registers

The addresses in this section are relative to the SDIO Slave base address (0x3FF4_B000) provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 26.4 *Register Summary*.

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

**Register 26.36. HINF_CFG_DATA1_REG (0x4)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | HINF_HIGHSPEED_ENABLE | HINF_SDIO_IOREADY1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**HINF_HIGHSPEED_ENABLE** Please initialize to 1. Do not modify it. (R/W)

**HINF_SDIO_IOREADY1** Please initialize to 1. Do not modify it. (R/W)

# Chapter 27

# SD/MMC Host Controller (SDHOST)

## 27.1 Overview

The ESP32 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 to be connected to SDIO memory cards, MMC cards and devices with a CE-ATA interface. It supports two external cards (Card0 and Card1).

## 27.2 Features

This module has the following features:

- Two external cards
- Supports SD Memory Card standard: versions 3.0 and 3.01
- Supports MMC: versions 4.41, 4.5, and 4.51
- Supports CE-ATA: version 1.1
- Supports 1-bit, 4-bit, and 8-bit (Card0 only) modes

The SD/MMC controller topology is shown in Figure 27.2-1. The controller supports two peripherals which cannot be functional at the same time.

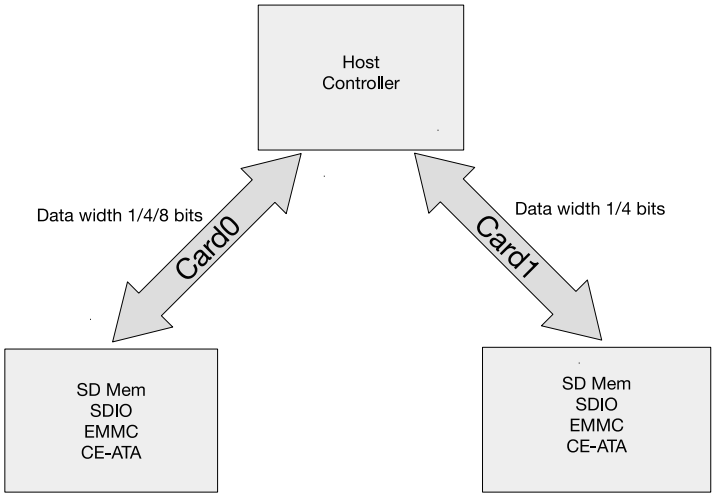



Figure 27.2-1. SD/MMC Controller Topology

## 27.3 SD/MMC External Interface Signals

The primary external interface signals, which enable the SD/MMC controller to communicate with an external device, are clock (clk), command (cmd) and data signals. Additional signals include the card interrupt, card detect, and write-protect signals. The direction of each signal is shown in Figure 27.3-1. The direction and description of each pin are listed in Table 27.3-1.

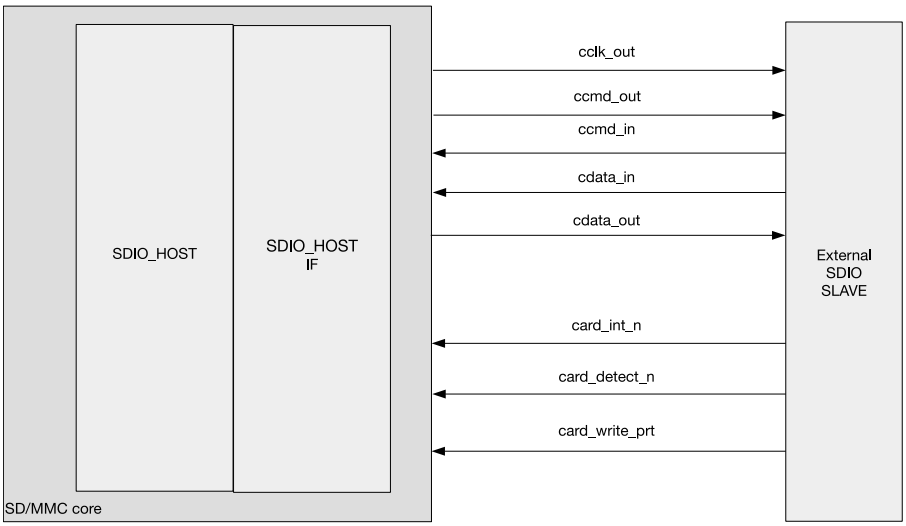



Figure 27.3-1. SD/MMC Controller External Interface Signals

Table 27.3-1. SD/MMC Signal Description

| Pin | Direction | Description |
|---|---|---|
| cclk_out | Output | Clock signals for slave device |
| ccmd | Duplex | Duplex command/response lines |
| cdata | Duplex | Duplex data read/write lines |
| card_detect_n | Input | Card detection input line |
| card_write_prt | Input | Card write protection status input |

## 27.4 Functional Description

### 27.4.1 SD/MMC Host Controller Architecture

The SD/MMC host controller consists of two main functional blocks, as shown in Figure 27.4-1:

- Bus Interface Unit (BIU): It provides APB interfaces for registers, data read and write operation by FIFO and DMA.
- Card Interface Unit (CIU): It handles external memory card interface protocols. It also provides clock control.

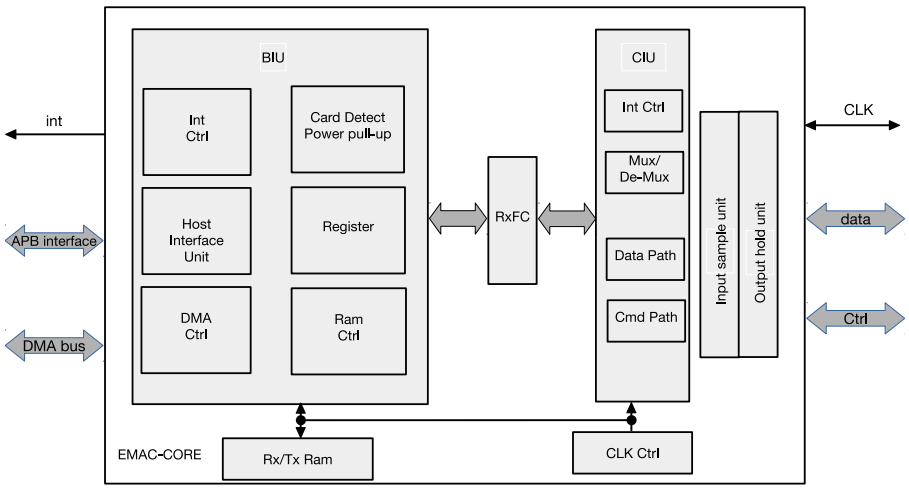



Figure 27.4-1. SDIO Host Block Diagram

#### 27.4.1.1 BIU

The BIU provides the access to registers and FIFO data through the Host Interface Unit (HIU). Additionally, it provides FIFO access to independent data through a DMA interface. The host interface can be configured as an APB interface. Figure 27.4-1 illustrates the internal components of the BIU. The BIU provides the following functions:

- Host interface
- DMA interface
- Interrupt control
- Register access
- FIFO access
- Power/pull-up control and card detection

#### 27.4.1.2 CIU

The CIU module implements the card-specific protocols. Within the CIU, the command path control unit and data path control unit prompt the controller to interface with the command and data ports, respectively, of the SD/MMC/CE-ATA cards. The CIU also provides clock control. Figure 27.4-1 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/demux unit

### 27.4.2 Command Path

The command path performs the following functions:

- Configures clock parameters
- Configures card command parameters
- Sends commands to card bus (ccmd_out line)
- Receives responses from card bus (ccmd_in line)
- Sends responses to BIU
- Drives the P-bit on the command line

The command path State Machine is shown in Figure 27.4-2.

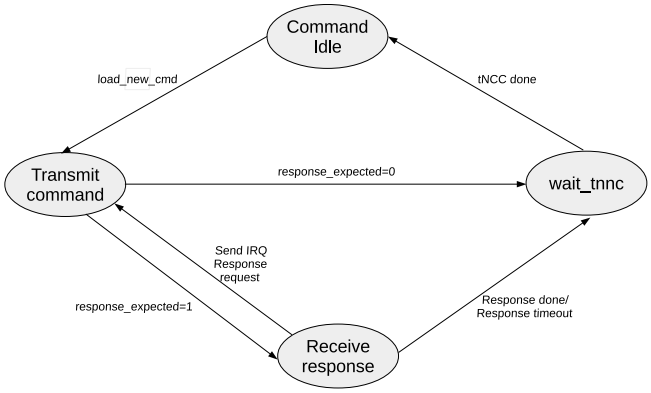



Figure 27.4-2. Command Path State Machine

## 27.4.3 Data Path

The data path block pops FIFO data and transmits them on cdata_out during a write-data transfer, or it receives data on cdata_in and pushes them into FIFO 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 data_expected bit is set in the Command register, the new command is a data-transfer command and the data path starts one of the following operations:

- Transmitting data if the read/write bit = 1
- Receiving data if read/write bit = 0

### 27.4.3.1 Data Transmit Operation

The data transmit state machine is illustrated in Figure 27.4-3. 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 are transmitted. Depending on the value of the transfer_mode bit in the Command 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 27.4-3.

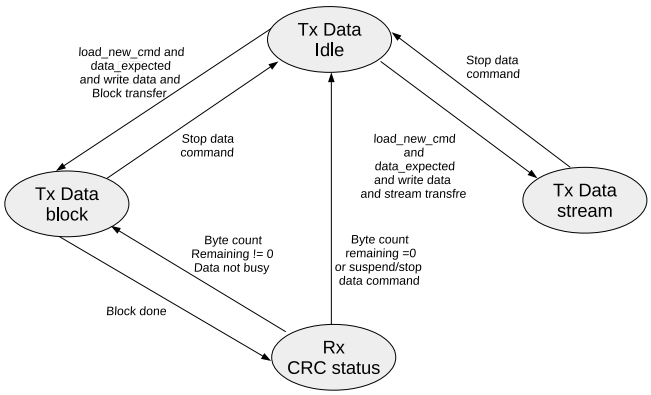



Figure 27.4-3. Data Transmit State Machine

### 27.4.3.2 Data Receive Operation

The data-receive state machine is illustrated in Figure 27.4-4. 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 are 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 transfer_mode bit in the Command 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 27.4-4.

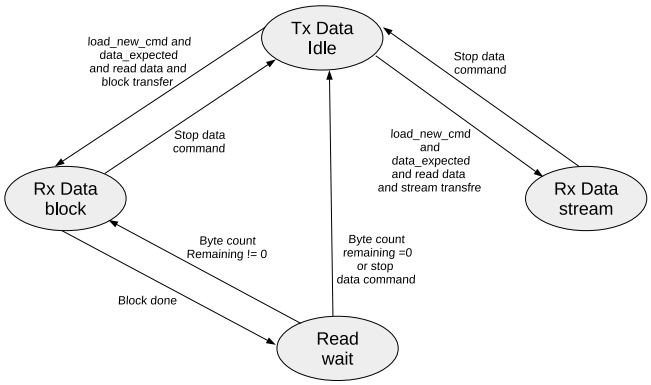



Figure 27.4-4. Data Receive State Machine

## 27.5 Software Restrictions for Proper CIU Operation

- 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 command at a time can be issued for data transfers.

- During an open-ended card-write operation, if the card clock is stopped due to FIFO being empty, the software must fill FIFO with data first, and then 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 FIFO, 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 stop_abort_cmd bit in the Command register, so that the CIU can stop the data transfer after issuing the card reset command.
- When the data's end bit error is set in the RINTSTS register, the CIU does not guarantee SDIO interrupts. In such a case, the software ignores SDIO interrupts and issues a stop/abort command to the card, so that the card stops sending read-data.
- If the card clock is stopped due to FIFO being full during a card read, the software will read at least two FIFO locations 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 should not be sent to another CE-ATA device.
- If a CE-ATA device's interrupts are enabled (nIEN=0), a new RW_BLK command should not be sent to the same device if the execution of a RW_BLK command is already in progress (the RW_BLK command used in this databook is the RW_MULTIPLE_BLOCK MMC command defined by the CE-ATA specifications). Only the CCSD can be sent while waiting for the CCS.
- If, however, a CE-ATA device's 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 in CE-ATA devices.
- The send_auto_stop signal is not supported (software should not set the send_auto_stop bit) in 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

## 27.6 RAM for Receiving and Sending Data

The submodule RAM is a buffer area for sending and receiving data. It can be divided into two units: the one is for sending data, and the other is for receiving data. The process of sending and receiving data can also be achieved by the CPU and DMA for reading and writing. The latter method is described in detail in Section 27.8.

### 27.6.1 Transmit RAM Module

There are two ways to enable a write operation: DMA and CPU read/write.

If SDIO-sending is enabled, data can be written to the transferred RAM module by APB interface or DMA. Data will be written from register EMAC_FIFO to the CPU, directly, by an APB interface.

### 27.6.2 Receive RAM Module

There are two ways to enable a read operation: DMA and CPU read/write.

When a subunit of the data path receives data, the subdata will be written onto the receive-RAM. Then, these subdata can be read either with the APB or the DMA method at the reading end. Register EMAC_FIFO can be read by the APB directly.

## 27.7 Descriptor Chain

Each linked list module consists of two parts: the linked list itself and a data buffer. In other words, each module points to a unique data buffer and the linked list that follows the module. Figure 27.7-1 shows the descriptor chain.

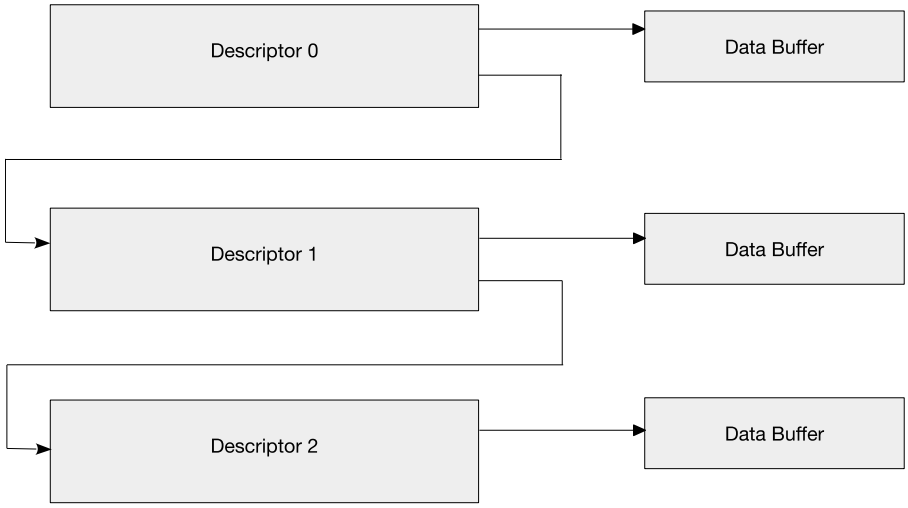



Figure 27.7-1. Descriptor Chain

## 27.8 The Structure of a Linked List

Each linked list consists of four words. As is shown below, Figure 27.8-1 demonstrates the linked list's structure, and Table 27.8-1, Table 27.8-2, Table 27.8-3, Table 27.8-4 provide the descriptions of linked lists.

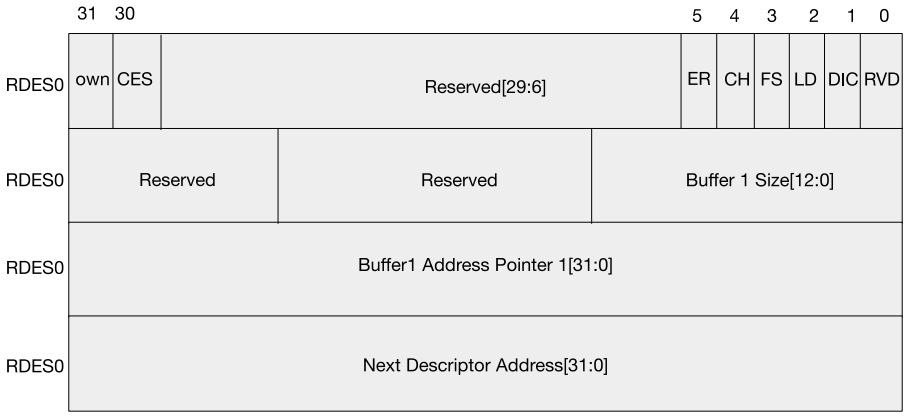



Figure 27.8-1. The Structure of a Linked List

The DES0 element contains control and status information.

Table 27.8-1. DES0

| Bits | Name | Description |
|---|---|---|
| 31 | OWN | When set, this bit indicates that the descriptor is owned by the DMAC. When reset, it indicates that the descriptor is owned by the Host. The DMAC clears this bit when it completes the data transfer. |
| 30 | CES (Card Error Summary) | These error bits indicate the status of the transition to or from the card.<br>The following bits are also present in RINTSTS, which indicates their digital logic OR gate.<br>• EBE: End Bit Error<br>• RTO: Response Time out<br>• RCRC: Response CRC<br>• SBE: Start Bit Error<br>• DRTO: Data Read Timeout<br>• DCRC: Data CRC for Receive<br>• RE: Response Error |
| 29:6 | Reserved | Reserved |
| 5 | ER (End of Ring) | When set, this bit indicates that the descriptor list has reached its final descriptor. The DMAC then returns to the base address of the list, creating a Descriptor Ring. |
| 4 | CH<br>(Second Address Chained) | When set, this bit indicates that the second address in the descriptor is the Next Descriptor address. When this bit is set, BS2 (DES1[25:13]) should be all zeros. |
| 3 | FD (First Descriptor) | When set, 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. |

| Bits | Name | Description |
|---|---|---|
| 2 | LD (Last Descriptor) | This bit is associated with the last block of a DMA transfer. When set, 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, this bit will prevent the setting of the TI/RI bit of the DMAC Status Register (IDSTS) for the data that ends in the buffer pointed by this descriptor. |
| 0 | Reserved | Reserved |

The DES1 element contains the buffer size.

Table 27.8-2. DES1

| Bits | Name | Description |
|---|---|---|
| 31:26 | Reserved | Reserved |
| 25:13 | Reserved | Reserved |
| 12:0 | BS1 (Buffer 1 Size) | Indicates the data buffer byte size, which must be a multiple of four. In the case where the buffer size is not a multiple of four, the resulting behavior is undefined. This field should not be zero. |

The DES2 element contains the address pointer to the data buffer.

Table 27.8-3. DES2

| Bits | Name | Description |
|---|---|---|
| 31:0 | Buffer Address Pointer 1 | These bits indicate the physical address of the data buffer. |

The DES3 element contains the address pointer to the next descriptor if the present descriptor is not the last one in a chained descriptor structure.

Table 27.8-4. DES3

| Bits | Name | Description |
|---|---|---|
| 31:0 | Next Descriptor Address | If the Second Address Chained (DES0[4]) bit is set, then this address contains the pointer to the physical memory where the Next Descriptor is present.<br>If this is not the last descriptor, then the Next Descriptor address pointer must be DES3[1:0] = 0. |

## 27.9 Initialization

### 27.9.1 DMAC Initialization

The DMAC initialization should proceed as follows:

- Write to the DMAC Bus Mode Register (BMOD_REG) will set the Host bus's access parameters.
- Write to the DMAC Interrupt Enable Register (IDINTEN) will mask any unnecessary interrupt causes.
- The software driver creates either the transmit or the receive descriptor list. Then, it writes to the DMAC Descriptor List Base Address Register (DBADDR), providing the DMAC with the starting address of the list.
- The DMAC engine attempts to acquire descriptors from descriptor lists.

### 27.9.2 DMAC Transmission Initialization

The DMAC transmission occurs as follows:

1. The Host sets up the elements (DES0-DES3) for transmission, and sets the OWN bit (DES0[31]). The Host also prepares the data buffer.
2. The Host programs the write-data command in the CMD register in BIU.
3. The Host also programs the required transmit threshold (TX_WMARK field in FIFOTH register).
4. The DMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMAC enters a suspend-state and asserts the Descriptor Unable interrupt in the IDSTS register. In such a case, the host needs to release the DMAC by writing any value to PLDMND_REG.
5. It then waits for the Command Done (CD) bit and no errors from BIU, which indicates that a transfer can be done.
6. Subsequently, the DMAC engine waits for a DMA interface request (dw_dma_req) from BIU. This request will be generated, based on the programmed transmit-threshold value. For the last bytes of data which cannot be accessed using a burst, single transfers are performed on the AHB Master Interface.
7. The DMAC fetches the transmit data from the data buffer in the Host memory and transfers them to FIFO for transmission to card.
8. When data span across multiple descriptors, the DMAC fetches the next descriptor and extends its operation using the following 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 IDSTS register by setting the Transmit Interrupt, if it has already been enabled. Also, the OWN bit is cleared by the DMAC by performing a write transaction to DES0.

### 27.9.3 DMAC Reception Initialization

The DMAC reception occurs as follows:

1. The Host sets up the element (DES0-DES3) for reception, and sets the OWN bit (DES0[31]).
2. The Host programs the read-data command in the CMD register in BIU.
3. Then, the Host programs the required level of the receive-threshold (RX_WMARK field in FIFOTH register).

4. The DMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMA enters a suspend-state and asserts the Descriptor Unable interrupt in the IDSTS register. In such a case, the host needs to release the DMAC by writing any value to PLDMND_REG.
5. It then waits for the Command Done (CD) bit and no errors from BIU, which indicates that a transfer can be done.
6. The DMAC engine then waits for a DMA interface request (dw_dma_req) from BIU. This request will be generated, based on the programmed receive-threshold value. For the last bytes of the data which cannot be accessed using a burst, single transfers are performed on the AHB.
7. The DMAC fetches the data from FIFO and transfers them to the Host memory.
8. When data span across multiple descriptors, the DMAC will fetch the next descriptor and extend its operation using the following 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 IDSTS register by setting Receive-Interrupt, if it has already been enabled. Also, the OWN bit is cleared by the DMAC by performing a write-transaction to DES0.

## 27.10 SD/MMC Timing

Figure 27.10-1 shows the timing diagram for SD/MMC in high-speed (HS) mode. Table 27.10-1 lists the timing requirements. These requirements are crucial for ensuring reliable and synchronized data communication in HS mode.

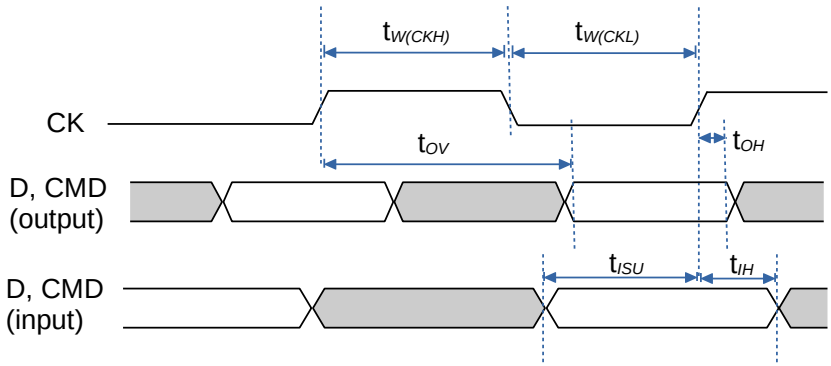



Figure 27.10-1. SD/MMC Timing in HS Mode

Table 27.10-1. SD/MMC Timing Requirements

| Symbol | Parameter | Conditions | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $t_{W(CKL)}$ | Clock Low Time | $f_{PP}$ = 80 MHz | 11.5 | 12.5 | — | ns |
| $t_{W(CKH)}$ | Clock High Time | $f_{PP}$ = 80 MHz | 11.5 | 12.5 | — | ns |
| $t_{ISU}$ | Input Setup Time in HS Mode | $f_{PP}$ = 80 MHz | 3.4 | — | — | ns |
| $t_{IH}$ | Input Hold Time in HS Mode | $f_{PP}$ = 80 MHz | 1.1 | — | — | ns |
| $t_{OV}$ | Output Valid Time in HS Mode | $f_{PP}$ = 80 MHz | — | — | 5.9 | ns |

| | | | | | | |
|---|---|---|---|---|---|---|
| $t_{OH}$ | Output Hold Time in HS Mode | $f_{PP}$ = 80 MHz | 0.5 | — | — | ns |

**Legend:**

- $f_{PP}$ - Clock frequency in data transfer mode.
- $t_{W(CKL)}$/$t_{W(CKH)}$ - Clock Low/High Time: $t_{W(CKL)}$/$t_{W(CKH)}$ represents the time that the clock signal (CK) should remain in the low or high state.
- $t_{ISU}$ - Input Setup Time in HS Mode: $t_{ISU}$ represents the setup time required for CMD and D (data lines) inputs.
- $t_{IH}$ - Input Hold Time in HS Mode: $t_{IH}$ specifies the hold time required for CMD and D inputs.
- $t_{OV}$ - Output Valid Time in HS Mode: $t_{OV}$ defines the time it takes for the CMD and D outputs to be ready.
- $t_{OH}$ - Output Hold Time in HS Mode: $t_{OH}$ specifies the hold time required for the CMD and D outputs to be valid.
- The timing of the CMD and D inputs and outputs are measured in relation to the clock signal CK.

## 27.11 Clock Phase Selection

If the setup time requirements for the input or output data signal are not met, users can specify the clock phase, as shown in the figure below.

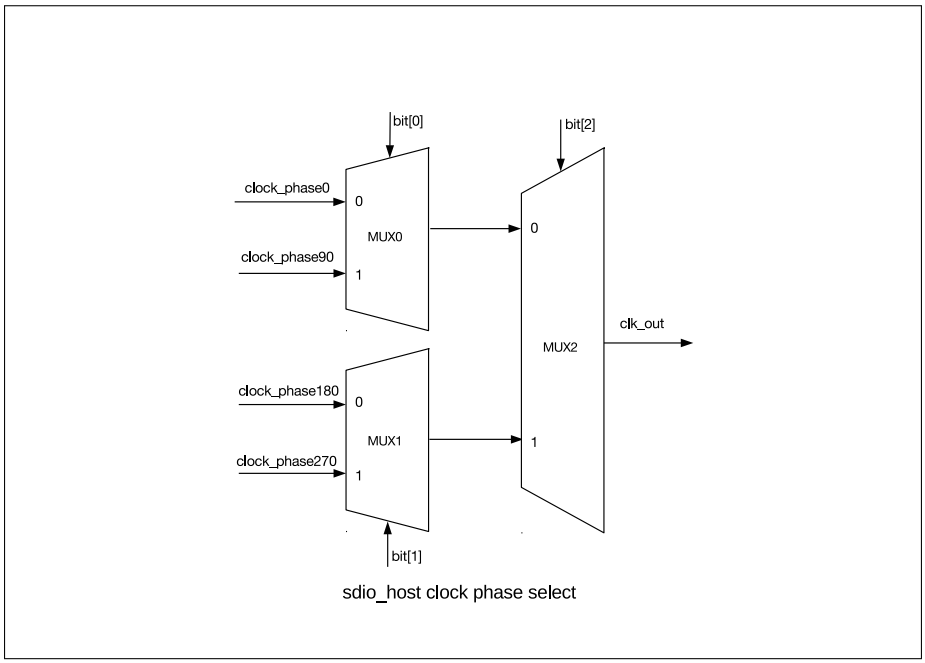



Figure 27.11-1. Clock Phase Selection

Please find detailed information on the clock phase selection register CLK_EDGE_SEL in Section Registers.

## 27.12 Interrupt

Interrupts can be generated as a result of various events. The IDSTS register contains all the bits that might cause an interrupt. The IDINTEN register contains an enable bit for each of the events that can cause an interrupt.

There are two groups of summary interrupts, "Normal" ones (bit8 NIS) and "Abnormal" ones (bit9 AIS), as outlined in the IDSTS register. Interrupts are cleared by writing 1 to the position of the corresponding bit. When all the enabled interrupts within a group are cleared, the corresponding summary bit is also cleared. When both summary bits are cleared, the interrupt signal dmac_intr_o is de-asserted (stops signalling).

Interrupts are not queued up, and if a new interrupt-event occurs before the driver has responded to it, no additional interrupts are generated. For example, the Receive Interrupt IDSTS[1] indicates that one or more data were transferred to the Host buffer.

An interrupt is generated only once for concurrent events. The driver must scan the IDSTS register for the interrupt cause.

## 27.13 Register Summary

The addresses in this section are relative to the SD/MMC base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| CTRL_REG | Control register | 0x0000 | R/W |
| CLKDIV_REG | Clock divider configuration register | 0x0008 | R/W |
| CLKSRC_REG | Clock source selection register | 0x000C | R/W |
| CLKENA_REG | Clock enable register | 0x0010 | R/W |
| TMOUT_REG | Data and response timeout configuration register | 0x0014 | R/W |
| CTYPE_REG | Card bus width configuration register | 0x0018 | R/W |
| BLKSIZ_REG | Card data block size configuration register | 0x001C | R/W |
| BYTCNT_REG | Data transfer length configuration register | 0x0020 | R/W |
| INTMASK_REG | SDIO interrupt mask register | 0x0024 | R/W |
| CMDARG_REG | Command argument data register | 0x0028 | R/W |
| CMD_REG | Command and boot configuration register | 0x002C | R/W |
| RESP0_REG | Response data register | 0x0030 | RO |
| RESP1_REG | Long response data register | 0x0034 | RO |
| RESP2_REG | Long response data register | 0x0038 | RO |
| RESP3_REG | Long response data register | 0x003C | RO |
| MINTSTS_REG | Masked interrupt status register | 0x0040 | RO |
| RINTSTS_REG | Raw interrupt status register | 0x0044 | R/W |
| STATUS_REG | SD/MMC status register | 0x0048 | RO |
| FIFOTH_REG | FIFO configuration register | 0x004C | R/W |
| CDETECT_REG | Card detect register | 0x0050 | RO |
| WRTPRT_REG | Card write protection (WP) status register | 0x0054 | RO |

| Name | Description | Address | Access |
|---|---|---|---|
| TCBCNT_REG | Transferred byte count register | 0x005C | RO |
| TBBCNT_REG | Transferred byte count register | 0x0060 | RO |
| DEBNCE_REG | Debounce filter time configuration register | 0x0064 | R/W |
| USRID_REG | User ID (scratchpad) register | 0x0068 | R/W |
| RST_N_REG | Card reset register | 0x0078 | R/W |
| BMOD_REG | Burst mode transfer configuration register | 0x0080 | R/W |
| PLDMND_REG | Poll demand configuration register | 0x0084 | WO |
| DBADDR_REG | Descriptor base address register | 0x0088 | R/W |
| IDSTS_REG | IDMAC status register | 0x008C | R/W |
| IDINTEN_REG | IDMAC interrupt enable register | 0x0090 | R/W |
| DSCADDR_REG | Host descriptor address pointer | 0x0094 | RO |
| BUFADDR_REG | Host buffer address pointer register | 0x0098 | RO |
| CLK_EDGE_SEL | Clock phase selection register | 0x0800 | R/W |

## 27.14 Registers

SD/MMC controller registers can be accessed by the APB bus of the CPU.

The addresses in this section are relative to the SD/MMC base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

**Register 27.1. CTRL_REG (0x0000)**

| (reserved) | (reserved) | (reserved) | CEATA_DEVICE_INTERRUPT_STATUS | SEND_AUTO_STOP_CCSD | SEND_CCSD | ABORT_READ_DATA | SEND_IRQ_RESPONSE | READ_WAIT | (reserved) | INT_ENABLE | (reserved) | DMA_RESET | FIFO_RESET | CONTROLLER_RESET | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 25 | 24 | 23 … 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x00 | 1 | 0x00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**CEATA_DEVICE_INTERRUPT_STATUS** 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)

**SEND_AUTO_STOP_CCSD** Always set send_auto_stop_ccsd and send_ccsd bits together; send_auto_stop_ccsd should not be set independently of send_ccsd. When set, SD/MMC automatically sends an internally-generated STOP command (CMD12) to the CE-ATA device. After sending this internally-generated STOP command, the Auto Command Done (ACD) bit in RINTSTS is set and an interrupt is generated for the host, in case the ACD interrupt is not masked. After sending the Command Completion Signal Disable (CCSD), SD/MMC automatically clears the send_auto_stop_ccsd bit. (R/W)

**SEND_CCSD** When set, SD/MMC sends CCSD to the CE-ATA device. 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. Once the CCSD pattern is sent to the device, SD/MMC automatically clears the send_ccsd bit. It also sets the Command Done (CD) bit in the RINTSTS register, and generates an interrupt for the host, in case the Command Done interrupt is not masked. NOTE: Once the send_ccsd bit is set, it takes two card clock cycles to drive the CCSD on the CMD line. Due to this, within the boundary conditions the CCSD may be sent to the CE-ATA device, even if the device has signalled CCS. (R/W)

**ABORT_READ_DATA** 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. This bit is automatically cleared once the data state machine is reset to idle. (R/W)

**SEND_IRQ_RESPONSE** Bit automatically clears once response is sent. To wait for MMC card interrupts, host issues CMD40 and waits for interrupt response from MMC card(s). In the meantime, if host wants SD/MMC to exit waiting for interrupt state, it can set this bit, at which time SD/MMC command state-machine sends CMD40 response on bus and returns to idle state. (R/W)

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

**Register 27.1. CTRL_REG (0x0000)**

**Continued from the previous page...**

**READ_WAIT** For sending read-wait to SDIO cards. (R/W)

**INT_ENABLE** Global interrupt enable/disable bit. 0: Disable; 1: Enable. (R/W)

**DMA_RESET** To reset DMA interface, firmware should set bit to 1. This bit is auto-cleared after two AHB clocks. (R/W)

**FIFO_RESET** To reset FIFO, firmware should set bit to 1. This bit is auto-cleared after completion of reset operation. Note: FIFO pointers will be out of reset after 2 cycles of system clocks in addition to synchronization delay (2 cycles of card clock), after the fifo_reset is cleared. (R/W)

**CONTROLLER_RESET** To reset controller, firmware should set this bit. This bit is auto-cleared after two AHB and two cclk_in clock cycles. (R/W)

**Register 27.2. CLKDIV_REG (0x0008)**

| CLK_DIVIDER3 | CLK_DIVIDER2 | CLK_DIVIDER1 | CLK_DIVIDER0 | |
|---|---|---|---|---|
| 31 24 | 23 16 | 15 8 | 7 0 | |
| 0x000 | 0x000 | 0x000 | 0x000 | Reset |

**CLK_DIVIDER3** Clock divider-3 value. Clock division factor is 2*$n$, where $n$=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

**CLK_DIVIDER2** Clock divider-2 value. Clock division factor is 2*$n$, where $n$=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

**CLK_DIVIDER1** Clock divider-1 value. Clock division factor is 2*$n$, where $n$=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

**CLK_DIVIDER0** Clock divider-0 value. Clock division factor is 2*$n$, where $n$=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

**Register 27.3. CLKSRC_REG (0x000C)**

| (reserved) | CLKSRC_REG | |
|---|---|---|
| 31 … 4 | 3 … 0 | |
| 0x000000 | 0x0 | Reset |

**CLKSRC_REG** Clock divider source for two SD cards is supported. Each card has two bits assigned to it. For example, bit[1:0] are assigned for card 0, bit[3:2] are assigned for card 1. Card 0 maps and internally routes clock divider[0:3] outputs to cclk_out[1:0] pins, depending on bit value.

00 : Clock divider 0;

01 : Clock divider 1;

10 : Clock divider 2;

11 : Clock divider 3.

In MMC-Ver3.3-only controller, only one clock divider is supported. The cclk_out is always from clock divider 0, and this register is not implemented. (R/W)

**Register 27.4. CLKENA_REG (0x0010)**

| (reserved) | CCLK_ENABEL | |
|---|---|---|
| 31 … 2 | 1 … 0 | |
| 0x00000 | 0x00000 | Reset |

**CCLK_ENABEL** Clock-enable control for two SD card clocks and one MMC card clock is supported.

0: Clock disabled;

1: Clock enabled.

In MMC-Ver3.3-only mode, since there is only one cclk_out, only cclk_enable[0] is used. (R/W)

### Register 27.5. TMOUT_REG (0x0014)

| DATA_TIMEOUT (31 : 8) | RESPONSE_TIMEOUT (7 : 0) | |
|---|---|---|
| 0xOFFFFFF | 0x040 | Reset |

**DATA_TIMEOUT** Value for card data read timeout. This value is also used for data starvation by host timeout. The timeout counter is started only after the card clock is stopped. This value is specified in number of card output clocks, i.e. cclk_out of the selected card.
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 needs to be disabled. (R/W)

**RESPONSE_TIMEOUT** Response timeout value. Value is specified in terms of number of card output clocks, i.e., cclk_out. (R/W)

### Register 27.6. CTYPE_REG (0x0018)

| (reserved) (31 : 18) | CARD_WIDTH8 (17 : 16) | (reserved) (15 : 2) | CARD_WIDTH4 (1 : 0) | |
|---|---|---|---|---|
| 0x00000 | 0x00000 | 0x00000 | 0x00000 | Reset |

**CARD_WIDTH8** One bit per card indicates if card is in 8-bit mode.
0: Non 8-bit mode;
1: 8-bit mode.
Bit[17:16] correspond to card[1:0] respectively. (R/W)

**CARD_WIDTH4** One bit per card indicates if card is 1-bit or 4-bit mode.
0: 1-bit mode;
1: 4-bit mode.
Bit[1:0] correspond to card[1:0] respectively. Only NUM_CARDS*2 number of bits are implemented. (R/W)

### Register 27.7. BLKSIZ_REG (0x001C)

| (reserved) (31 : 16) | BLOCK_SIZE (15 : 0) | |
|---|---|---|
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x00200 | Reset |

**BLOCK_SIZE** Block size. (R/W)

### Register 27.8. BYTCNT_REG (0x0020)

| 31 … 0 | |
|---|---|
| 0x000000200 | Reset |

**BYTCNT_REG** Number of bytes to be transferred, should be an integral multiple of Block Size for block transfers. For data transfers of undefined byte lengths, byte count should be set to 0. When byte count is set to 0, it is the responsibility of host to explicitly send stop/abort command to terminate data transfer. (R/W)

### Register 27.9. INTMASK_REG (0x0024)

| (reserved) | SDIO_INT_MASK | INT_MASK | |
|---|---|---|---|
| 31 … 18 | 17 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | 0x00000 | Reset |

**SDIO_INT_MASK** SDIO interrupt mask, one bit for each card. Bit[17:16] correspond to card[15:0] respectively. When masked, SDIO interrupt detection for that card is disabled. 0 masks an interrupt, and 1 enables an interrupt. In MMC-Ver3.3-only mode, these bits are always 0. (R/W)

**INT_MASK** These bits used to mask unwanted interrupts. A value of 0 masks interrupt, and a value of 1 enables the interrupt. (R/W)

Bit 15 (EBE): End-bit error, read/write (no CRC)
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation-by-host timeout/Volt_switch_int
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 (CD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect

### Register 27.10. CMDARG_REG (0x0028)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**CMDARG_REG** Value indicates command argument to be passed to the card. (R/W)

### Register 27.11. CMD_REG (0x002C)

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 … 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 … 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| START_CMD | (reserved) | USE_HOLE | (reserved) | (reserved) | (reserved) | (reserved) | (reserved) | CCS_EXPECTED | READ_CEATA_DEVICE | UPDATE_CLOCK_REGISTERS_ONLY | CARD_NUMBER | SEND_INITIALIZATION | STOP_ABORT_CMD | WAIT_PRVDATA_COMPLETE | SEND_AUTO_STOP | TRANSFER_MODE | READ/WRITE | DATA_EXPECTED | CHECK_RESPONSE_CRC | RESPONSE_LENGTH | RESPONSE_EXPECT | CMD_INDEX | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x00 | Reset |

**START_CMD** Start command. 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. Once command is sent and a response is received from SD_MMC_CEATA cards, Command Done bit is set in the raw interrupt Register. (R/W)

**USE_HOLE** Use Hold Register. (R/W) 0: CMD and DATA sent to card bypassing HOLD Register; 1: CMD and DATA sent to card through the HOLD Register.

**CCS_EXPECTED** Expected Command Completion Signal (CCS) configuration. (R/W)

0: Interrupts are not enabled in CE-ATA device (nIEN = 1 in ATA control register), or command does not expect CCS from device.

1: Interrupts are enabled in CE-ATA device (nIEN = 0), and RW_BLK command expects command completion signal from CE-ATA device.

If the command expects Command Completion Signal (CCS) from the CE-ATA device, the software should set this control bit. SD/MMC sets Data Transfer Over (DTO) bit in RINTSTS register and generates interrupt to host if Data Transfer Over interrupt is not masked.

**READ_CEATA_DEVICE** Read access flag. (R/W)

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.

Software should set this bit to indicate that CE-ATA device is being accessed for read transfer. This bit is used to disable read data timeout indication while performing CE-ATA read transfers. Maximum value of I/O transmission delay can be no less than 10 seconds. SD/MMC should not indicate read data timeout while waiting for data from CE-ATA device. (R/W)

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

**Register 27.11. CMD_REG (0x002C)**

**Continued from the previous page...**

**UPDATE_CLOCK_REGISTERS_ONLY** (R/W)

0: Normal command sequence.

1: Do not send commands, just update clock register value into card clock domain

Following register values are transferred into card clock domain: CLKDIV, CLRSRC, and CLKENA. Changes card clocks (change frequency, truncate off or on, and set low-frequency mode). This is provided in order to change clock frequency or stop clock without having to send command to cards.

During normal command sequence, when update_clock_registers_only = 0, following control registers are transferred from BIU to CIU: CMD, CMDARG, TMOUT, CTYPE, BLKSIZ, and BYTCNT. CIU uses new register values for new command sequence to card(s). When bit is set, there are no Command Done interrupts because no command is sent to SD_MMC_CEATA cards.

**CARD_NUMBER** Card number in use. Represents physical slot number of card being accessed. In MMC-Ver3.3-only mode, up to two cards are supported. In SD-only mode, up to two cards are supported. (R/W)

**SEND_INITIALIZATION** (R/W)

0: Do not send initialization sequence (80 clocks of 1) before sending this command.

1: Send initialization sequence before sending this command.

After power on, 80 clocks must be sent to card for initialization before sending any commands to card. Bit should be set while sending first command to card so that controller will initialize clocks before sending command to card.

**STOP_ABORT_CMD** (R/W)

0: Neither stop nor abort command can stop current data transfer. If abort is sent to function-number currently selected or 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. When open-ended or predefined data transfer is in progress, and host issues stop or abort command to stop data transfer, bit should be set so that command/data state-machines of CIU can return correctly to idle state.

**WAIT_PRVDATA_COMPLETE** (R/W)

0: Send command at once, even if previous data transfer has not completed;

1: Wait for previous data transfer to complete before sending Command.

The wait_prvdata_complete = 0 option is typically used to query status of card during data transfer or to stop current data transfer. card_number should be same as in previous command.

**SEND_AUTO_STOP** (R/W)

0: No stop command is sent at the end of data transfer;

1: Send stop command at the end of data transfer.

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

### Register 27.11. CMD_REG (0x002C)

Continued from the previous page ...

**TRANSFER_MODE** (R/W)

0: Block data transfer command;

1: Stream data transfer command. Don't care if no data expected.

**READ/WRITE** (R/W)

0: Read from card;

1: Write to card.

Don't care if no data is expected from card.

**DATA_EXPECTED** (R/W)

0: No data transfer expected.

1: Data transfer expected.

**CHECK_RESPONSE_CRC** (R/W)

0: Do not check;

1: Check response CRC.

Some of command responses do not return valid CRC bits. Software should disable CRC checks for those commands in order to disable CRC checking by controller.

**RESPONSE_LENGTH** (R/W)

0: Short response expected from card;

1: Long response expected from card.

**RESPONSE_EXPECT** (R/W)

0: No response expected from card;

1: Response expected from card.

**CMD_INDEX** Command index. (R/W)

### Register 27.12. RESP0_REG (0x0030)

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**RESP0_REG** Bit[31:0] of response. (RO)

### Register 27.13. RESP1_REG (0x0034)

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**RESP1_REG** Bit[63:32] of long response. (RO)

### Register 27.14. RESP2_REG (0x0038)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RESP2_REG** Bit[95:64] of long response. (RO)

### Register 27.15. RESP3_REG (0x003C)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**RESP3_REG** Bit[127:96] of long response. (RO)

### Register 27.16. MINTSTS_REG (0x0040)

| (reserved) | SDIO_INTERRUPT_MSK | INT_STATUS_MSK | |
|---|---|---|---|
| 31 … 18 | 17 … 16 | 15 … 0 | |
| 0 | 0x0 | 0x00000 | Reset |

**SDIO_INTERRUPT_MSK** Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. SDIO interrupt for card is enabled only if corresponding sdio_int_mask bit is set in Interrupt mask register (Setting mask bit enables interrupt). (RO)

**INT_STATUS_MSK** Interrupt enabled only if corresponding bit in interrupt mask register is set. (RO)

Bit 15 (EBE): End-bit error, read/write (no CRC)
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation by host timeout (HTO)
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 (CD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect

**Register 27.17. RINTSTS_REG (0x0044)**

| (reserved) | SDIO_INTERRUPT_RAW | INT_STATUS_RAW | |
|---|---|---|---|
| 31 18 | 17 16 | 15 0 | |
| 0x00000 | 0x0 | 0x00000 | Reset |

**SDIO_INTERRUPT_RAW** Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. Setting a bit clears the corresponding interrupt bit and writing 0 has no effect. (R/W)

0: No SDIO interrupt from card;

1: SDIO interrupt from card.

In MMC-Ver3.3-only mode, these bits are always 0. Bits are logged regardless of interrupt-mask status. (R/W)

**INT_STATUS_RAW** Setting a bit clears the corresponding interrupt and writing 0 has no effect. Bits are logged regardless of interrupt mask status. (R/W)

Bit 15 (EBE): End-bit error, read/write (no CRC)

Bit 14 (ACD): Auto command done

Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt

Bit 12 (HLE): Hardware locked write error

Bit 11 (FRUN): FIFO underrun/overrun error

Bit 10 (HTO): Data starvation by host timeout (HTO)

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 (CD): Command done

Bit 1 (RE): Response error

Bit 0 (CD): Card detect

**Register 27.18. STATUS_REG (0x0048)**

| (reserved) | (reserved) | FIFO_COUNT | RESPONSE_INDEX | DATA_STATE_MC_BUSY | DATA_BUSY | DATA_3_STATUS | COMMAND_FSM_STATES | FIFO_FULL | FIFO_EMPTY | FIFO_TX_WATERMARK | FIFO_RX_WATERMARK | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 … 17 | 16 … 11 | 10 | 9 | 8 | 7 … 4 | 3 | 2 | 1 | 0 | |
| 0 | 0 | 0x000 | 0x00 | 1 | 1 | 1 | 0x01 | 0 | 1 | 1 | 0 | Reset |

**FIFO_COUNT** FIFO count, number of filled locations in FIFO. (RO)

**RESPONSE_INDEX** Index of previous response, including any auto-stop sent by core. (RO)

**DATA_STATE_MC_BUSY** Data transmit or receive state-machine is busy. (RO)

**DATA_BUSY** Inverted version of raw selected card_data[0]. (RO)

0: Card data not busy;

1: Card data busy.

**DATA_3_STATUS** Raw selected card_data[3], checks whether card is present. (RO)

0: card not present;

1: card present.

**COMMAND_FSM_STATES** Command FSM states. (RO)

0: Idle

1: Send init sequence

2: Send cmd start bit

3: Send cmd tx bit

4: Send cmd index + arg

5: Send cmd crc7

6: Send cmd end bit

7: Receive resp start bit

8: Receive resp IRQ response

9: Receive resp tx bit

10: Receive resp cmd idx

11: Receive resp data

12: Receive resp crc7

13: Receive resp end bit

14: Cmd path wait NCC

15: Wait, cmd-to-response turnaround

**FIFO_FULL** FIFO is full status. (RO)

**FIFO_EMPTY** FIFO is empty status. (RO)

**FIFO_TX_WATERMARK** FIFO reached Transmit watermark level, not qualified with data transfer. (RO)

**FIFO_RX_WATERMARK** FIFO reached Receive watermark level, not qualified with data transfer. (RO)

**Register 27.19. FIFOTH_REG (0x004C)**

| (reserved) | DMA_MULTIPLE_TRANSACTION_SIZE | (reserved) | RX_WMARK | (reserved) | TX_WMARK | |
|---|---|---|---|---|---|---|
| 31 | 30 28 | 27 | 26 16 | 15 12 | 11 0 | |
| 0 | 0x0 | 0 | x x x x x x x x x x x | 0 0 0 0 | 0x0000 | Reset |

**DMA_MULTIPLE_TRANSACTION_SIZE** Burst size of multiple transaction, should be programmed same as DMA controller multiple-transaction-size SRC/DEST_MSIZE. 000: 1-byte transfer; 001: 4-byte transfer; 010: 8-byte transfer; 011: 16-byte transfer; 100: 32-byte transfer; 101: 64-byte transfer; 110: 128-byte transfer; 111: 256-byte transfer. (R/W)

**RX_WMARK** FIFO threshold watermark level when receiving data to card.When FIFO data count reaches greater than this number (FIFO_RX_WATERMARK), 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)

**TX_WMARK** FIFO threshold watermark level when transmitting data to card. When FIFO data count is less than or equal to this number (FIFO_TX_WATERMARK), DMA/FIFO request is raised. If Interrupt is enabled, then interrupt occurs. During end of packet, request or interrupt is generated, regardless of threshold programming.In non-DMA mode, when transmit 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 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)

### Register 27.20. CDETECT_REG (0x0050)

| (reserved) | CARD_DETECT_N | |
|---|---|---|
| 31 … 2 | 1 … 0 | |
| 0x0 | 0x0 | Reset |

**CARD_DETECT_N** Value on card_detect_n input ports (1 bit per card), read-only bits.0 represents presence of card. Only NUM_CARDS number of bits are implemented. (RO)

### Register 27.21. WRTPRT_REG (0x0054)

| (reserved) | WRITE_PROTECT | |
|---|---|---|
| 31 … 2 | 1 … 0 | |
| 0x0 | 0x0 | Reset |

**WRITE_PROTECT** Value on card_write_prt input ports (1 bit per card).1 represents write protection. Only NUM_CARDS number of bits are implemented. (RO)

### Register 27.22. TCBCNT_REG (0x005C)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TCBCNT_REG** Number of bytes transferred by CIU unit to card. (RO)

### Register 27.23. TBBCNT_REG (0x0060)

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**TBBCNT_REG** Number of bytes transferred between Host/DMA memory and BIU FIFO. (RO)

**Register 27.24. DEBNCE_REG (0x0064)**

| (reserved) | DEBOUNCE_COUNT | |
|---|---|---|
| 31 24 | 23 0 | |
| 0 0 0 0 0 0 0 0 | 0x0000000 | Reset |

**DEBOUNCE_COUNT** 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 27.25. USRID_REG (0x0068)**

| | |
|---|---|
| 31 0 | |
| 0x000000000 | Reset |

**USRID_REG** User identification register, value set by user. Default reset value can be picked by user while configuring core before synthesis. Can also be used as a scratchpad register by user. (R/W)

**Register 27.26. RST_N_REG (0x0078)**

| (reserved) | RST_CARD_RESET | |
|---|---|---|
| 31 2 | 1 0 | |
| 0 | 0x1 | Reset |

**RST_CARD_RESET** Hardware reset.1: Active mode; 0: Reset. These bits cause the cards to enter pre-idle state, which requires them to be re-initialized. CARD_RESET[0] should be set to 1'b0 to reset card0, CARD_RESET[1] should be set to 1'b0 to reset card1.The number of bits implemented is restricted to NUM_CARDS. (R/W)

**Register 27.27. BMOD_REG (0x0080)**

| (reserved) | BMOD_PBL | BMOD_DE | (reserved) | BMOD_FB | BMOD_SWR | |
|---|---|---|---|---|---|---|
| 31 … 11 | 10 … 8 | 7 | 6 … 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0x0 | 0 | 0x00 | 0 | 0 | Reset |

**BMOD_PBL** Programmable Burst Length. These bits indicate the maximum number of beats to be performed in one IDMAC transaction. The IDMAC will always attempt to burst as specified in PBL each time it starts a burst transfer on the host bus. The permissible values are 1, 4, 8, 16, 32, 64, 128 and 256. This value is the mirror of MSIZE of FIFOTH register. In order to change this value, write the required value to FIFOTH register. This is an encode value as follows:
000: 1-byte transfer; 001: 4-byte transfer; 010: 8-byte transfer; 011: 16-byte transfer; 100: 32-byte transfer; 101: 64-byte transfer; 110: 128-byte transfer; 111: 256-byte transfer.
PBL is a read-only value and is applicable only for data access, it does not apply to descriptor access. (R/W)

**BMOD_DE** IDMAC Enable. When set, the IDMAC is enabled. (R/W)

**BMOD_FB** Fixed Burst. Controls whether the AHB Master interface performs fixed burst transfers or not. When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB will use SINGLE and INCR burst transfer operations. (R/W)

**BMOD_SWR** Software Reset. When set, the DMA Controller resets all its internal registers. It is automatically cleared after one clock cycle. (R/W)

**Register 27.28. PLDMND_REG (0x0080)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**PLDMND_REG** Poll Demand. If the OWN bit of a descriptor is not set, the FSM goes to the Suspend state. The host needs to write any value into this register for the IDMAC FSM to resume normal descriptor fetch operation. This is a write only register, PD bit is write-only. (WO)

**Register 27.29. DBADDR_REG (0x0088)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**DBADDR_REG** Start of Descriptor List. Contains the base address of the First Descriptor. The LSB bits [1:0] are ignored and taken as all-zero by the IDMAC internally. Hence these LSB bits may be treated as read-only. (R/W)

**Register 27.30. IDSTS_REG (0x008C)**

| (reserved) | IDSTS_FSM | IDSTS_FBE_CODE | IDSTS_AIS | IDSTS_NIS | (reserved) | IDSTS_CES | IDSTS_DU | (reserved) | IDSTS_FBE | IDSTS_RI | IDSTS_TI | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 17 | 16 … 13 | 12 … 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 | 0x00 | 0x0 | 0 | 0 | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**IDSTS_FSM** DMAC FSM present state: (RO)
0: DMA_IDLE; 1: DMA_SUSPEND; 2: DESC_RD; 3: DESC_CHK; 4: DMA_RD_REQ_WAIT
5: DMA_WR_REQ_WAIT; 6: DMA_RD; 7: DMA_WR; 8: DESC_CLOSE.

**IDSTS_FBE_CODE** Fatal Bus Error Code. Indicates the type of error that caused a Bus Error. Valid only when the Fatal Bus Error bit IDSTS[2] is set. This field does not generate an interrupt. (RO)
3b001: Host Abort received during transmission;
3b010: Host Abort received during reception;
Others: Reserved.

**IDSTS_AIS** Abnormal Interrupt Summary. Logical OR of the following: IDSTS[2] : Fatal Bus Interrupt, IDSTS[4] : DU bit Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes AIS to be set is cleared. Writing 1 clears this bit. (R/W)

**IDSTS_NIS** Normal Interrupt Summary. Logical OR of the following: IDSTS[0] : Transmit Interrupt, IDSTS[1] : Receive Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes NIS to be set is cleared. Writing 1 clears this bit. (R/W)

**IDSTS_CES** Card Error Summary. Indicates the status of the transaction to/from the card, also present in RINTSTS. Indicates the logical OR of the following bits: EBE : End Bit Error, RTO : Response Timeout/Boot Ack Timeout, RCRC : Response CRC, SBE : Start Bit Error, DRTO : Data Read Timeout/BDS timeout, DCRC : Data CRC for Receive, RE : Response Error.
Writing 1 clears this bit. The abort condition of the IDMAC depends on the setting of this CES bit. If the CES bit is enabled, then the IDMAC aborts on a response error. (R/W)

**IDSTS_DU** Descriptor Unavailable Interrupt. This bit is set when the descriptor is unavailable due to OWN bit = 0 (DES0[31] =0). Writing 1 clears this bit. (R/W)

**IDSTS_FBE** Fatal Bus Error Interrupt. Indicates that a Bus Error occurred (IDSTS[12:10]) . When this bit is set, the DMA disables all its bus accesses. Writing 1 clears this bit. (R/W)

**IDSTS_RI** Receive Interrupt. Indicates the completion of data reception for a descriptor. Writing 1 clears this bit. (R/W)

**IDSTS_TI** Transmit Interrupt. Indicates that data transmission is finished for a descriptor. Writing 1 clears this bit. (R/W)

**Register 27.31. IDINTEN_REG (0x0090)**

| 31 | | | | | | | | | | | | | | | | | | | | | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | IDINTEN_AI | IDINTEN_NI | (reserved) | | IDINTEN_CES | IDINTEN_DU | (reserved) | IDINTEN_FBE | IDINTEN_RI | IDINTEN_TI | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**IDINTEN_AI** Abnormal Interrupt Summary Enable. (R/W)

When set, an abnormal interrupt is enabled. This bit enables the following bits:

IDINTEN[2]: Fatal Bus Error Interrupt;

IDINTEN[4]: DU Interrupt.

**IDINTEN_NI** Normal Interrupt Summary Enable. (R/W)

When set, a normal interrupt is enabled. When reset, a normal interrupt is disabled. This bit enables the following bits:

IDINTEN[0]: Transmit Interrupt;

IDINTEN[1]: Receive Interrupt.

**IDINTEN_CES** Card Error summary Interrupt Enable. When set, it enables the Card Interrupt summary. (R/W)

**IDINTEN_DU** Descriptor Unavailable Interrupt. When set along with Abnormal Interrupt Summary Enable, the DU interrupt is enabled. (R/W)

**IDINTEN_FBE** Fatal Bus Error Enable. When set with Abnormal Interrupt Summary Enable, the Fatal Bus Error Interrupt is enabled. When reset, Fatal Bus Error Enable Interrupt is disabled. (R/W)

**IDINTEN_RI** Receive Interrupt Enable. When set with Normal Interrupt Summary Enable, Receive Interrupt is enabled. When reset, Receive Interrupt is disabled. (R/W)

**IDINTEN_TI** Transmit Interrupt Enable. When set with Normal Interrupt Summary Enable, Transmit Interrupt is enabled. When reset, Transmit Interrupt is disabled. (R/W)

**Register 27.32. DSCADDR_REG (0x0094)**

| 31 | 0 | |
|---|---|---|
| 0x000000000 | | Reset |

**DSCADDR_REG** Host Descriptor Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the start address of the current descriptor read by the IDMAC. (RO)

**Register 27.33. BUFADDR_REG (0x0098)**

| 31 … 0 | |
|---|---|
| 0x000000000 | Reset |

**BUFADDR_REG** Host Buffer Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the current Data Buffer Address being accessed by the IDMAC. (RO)

**Register 27.34. CLK_EDGE_SEL (0x0800)**

| (reserved) | CCLKIN_EDGE_N | CCLKIN_EDGE_L | CCLKIN_EDGE_H | CCLKIN_EDGE_SLF_SEL | CCLKIN_EDGE_SAM_SEL | CCLKIN_EDGE_DRV_SEL | |
|---|---|---|---|---|---|---|---|
| 31 … 21 | 20 … 17 | 16 … 13 | 12 … 9 | 8 … 6 | 5 … 3 | 2 … 0 | |
| 0x000 | 0x1 | 0x0 | 0x1 | 0x0 | 0x0 | 0x0 | Reset |

**CCLKIN_EDGE_N** This value should be equal to CCLKIN_EDGE_L. (R/W)

**CCLKIN_EDGE_L** The low level of the divider clock. The value should be larger than CCLKIN_EDGE_H. (R/W)

**CCLKIN_EDGE_H** The high level of the divider clock. The value should be smaller than CCLKIN_EDGE_L. (R/W)

**CCLKIN_EDGE_SLF_SEL** It is used to select the clock phase of the internal signal from phase90, phase180, or phase270. (R/W)

**CCLKIN_EDGE_SAM_SEL** It is used to select the clock phase of the input signal from phase90, phase180, or phase270. (R/W)

**CCLKIN_EDGE_DRV_SEL** It is used to select the clock phase of the output signal from phase90, phase180, or phase270. (R/W)

# Chapter 28

# LED PWM Controller (LEDC)

## 28.1 Introduction

The LED_PWM controller is primarily designed to control the intensity of LEDs, although it can be used to generate PWM signals for other purposes as well. It has 16 channels which can generate independent waveforms that can be used to drive RGB LED devices. For maximum flexibility, the high-speed as well as the low-speed channels can be driven from one of four high-speed/low-speed timers. The PWM controller also has the ability to automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference. To increase resolution, the LED_PWM controller is also able to dither between two values, when a fractional PWM value is configured.

The LED_PWM controller has eight high-speed and eight low-speed PWM generators. In this document, they will be referred to as hsch*n* and lsch*n*, respectively. These channels can be driven from four timers which will be indicated by h_timer*x* and l_timer*x*.

## 28.2 Functional Description

### 28.2.1 Architecture

Figure 28.2-1 shows the architecture of the LED_PWM controller. As can be seen in the figure, the LED_PWM controller contains eight high-speed and eight low-speed channels. There are four high-speed clock modules for the high-speed channels, from which one h_timer*x* can be selected. There are also four low-speed clock modules for the low-speed channels, from which one l_timer*x* can be selected.

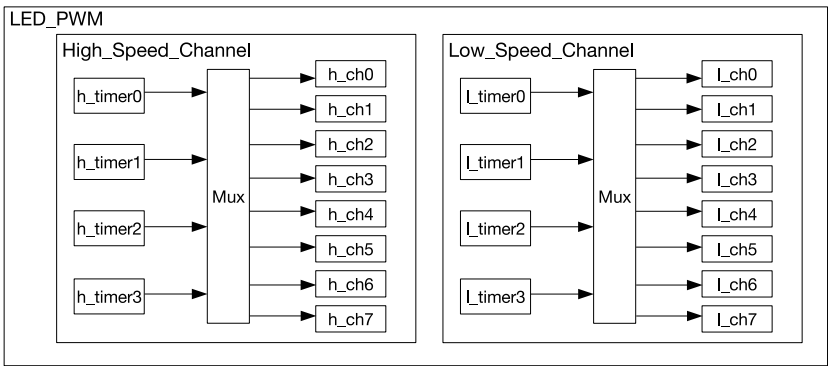



Figure 28.2-1. LED_PWM Architecture

Figure 28.2-2 illustrates a PWM channel with its selected timer; in this instance a high-speed channel and associated high-speed timer.

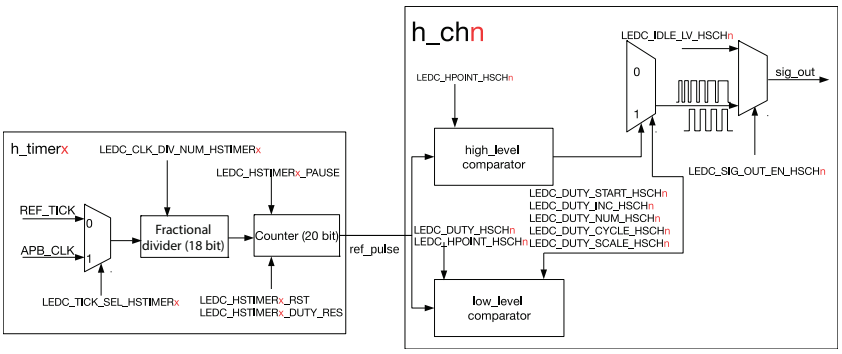



Figure 28.2-2. LED_PWM High-speed Channel Diagram

## 28.2.2 Timers

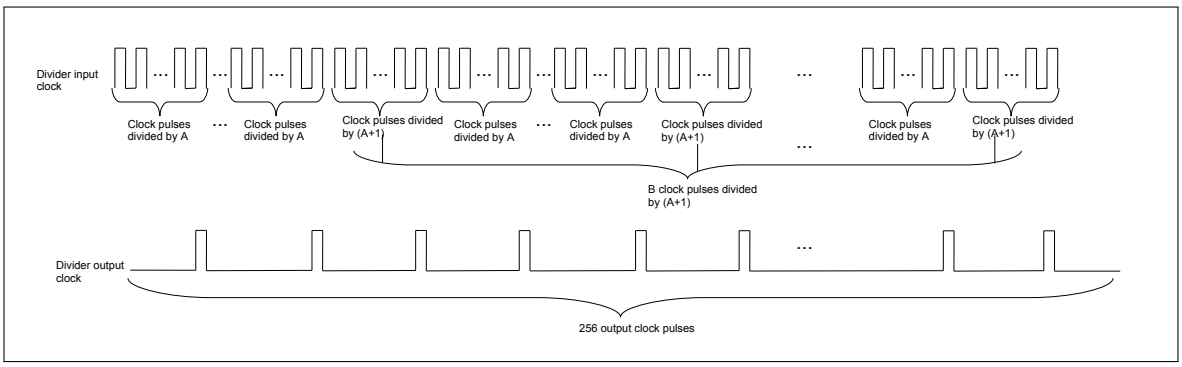



Figure 28.2-3. LED_PWM Divider

The clock of each high-speed timer, LEDC_CLK*x*, has two clock sources: REF_TICK or APB_CLK. For more information on the clock sources, please see Chapter Reset And Clock. The input clock is divided down by a divider first. The division factor is specified by LEDC_CLK_DIV_NUM_HSTIMER*x* which contains a fixed point number: the highest 10 bits represent the integer portion A, while the lowest eight bits contain the fractional portion B. The effective division factor LEDC_CLK_DIV*x* is as follows:

$$LEDC_CLK_DIVx = A + \frac{B}{256}$$

The division factor ranges from 1 ∼ 1023.

When the fractional part B is not 0, the input and output clock of the divider is shown as in figure 28.2-3. Among the 256 output clocks, B of them are divided by (A+1), whereas the remaining (256-B) are divided by A. Output clocks divided by (A+1) are evenly distributed in the total 256 output clocks.

The output clock of the divider is the base clock for the counter which will count up to the value specified in LEDC_HSTIMER*x*_DUTY_RES. An overflow interrupt will be generated once the counting value reaches $2^{LEDC_HSTIMERx_DUTY_RES} - 1$, at which point the counter restarts counting from 0. It is also possible to reset, suspend, and read the values of the counter by software.

The output signal of the timer is the 20-bit value generated by the counter. The cycle period of this signal defines the frequency of the signals of any PWM channels connected to this timer.

The frequency of a PWM generator output signal, sig_out*n*, depends on the frequency of the timer's clock source LEDC_CLK*x*, the division factor of the divider LEDC_CLK_DIV*x*, as well as the duty resolution (counter

width) LEDC_HSTIMERx_DUTY_RES:

$$f_{\text{sig_out}n} = \frac{f_{\text{LEDC_CLK}x}}{\text{LEDC_CLK_DIV}x \cdot 2^{\text{LEDC_HSTIMER}x\text{_DUTY_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC_HSTIMER}x\text{_DUTY_RES} = \log_2\left(\frac{f_{\text{LEDC_CLK}x}}{f_{\text{sig_out}n} \cdot \text{LEDC_CLK_DIV}x}\right)$$

Table 28.2-1 lists the commonly-used frequencies and their corresponding resolutions.

**Table 28.2-1. Commonly-used Frequencies and Resolutions**

| LEDC_CLKx | PWM Frequency | Highest Resolution (bit) [1] | Lowest Resolution (bit) [2] |
|---|---|---|---|
| APB_CLK (80 MHz) | 1 kHz | 16 | 7 |
| APB_CLK (80 MHz) | 5 kHz | 13 | 4 |
| APB_CLK (80 MHz) | 10 kHz | 12 | 3 |
| RC_FAST_CLK (8 MHz) | 1 kHz | 12 | 3 |
| RC_FAST_CLK (8 MHz) | 2 kHz | 11 | 2 |
| REF_TICK (1 MHz) | 1 kHz | 9 | 1 |

[1] The highest resolution is calculated when the clock divisor LEDC_CLK_DIVx is 1 and rounded down. 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_DIVx is $1023 + \frac{255}{256}$ and rounded up. If the lowest resolution calculated by the formula is lower than 0, then the lowest resolution should be 1.

The low-speed timers l_timerx on the low-speed channel differ from the high-speed timers h_timerx in two aspects:

1. Where the high-speed timer clock source can be clocked from REF_TICK or APB_CLK, the low-speed timers are sourced from either REF_TICK or SLOW_CLOCK. The SLOW_CLOCK source can be either APB_CLK (80 MHz) or 8 MHz, and can be selected using LEDC_APB_CLK_SEL.
2. The high-speed counter and divider are glitch-free, which means that if the software modifies the maximum counter or divisor value, the update will come into effect after the next overflow interrupt. In contrast, the low-speed counter and divider will update these values only when LEDC_LSTIMERx_PARA_UP is set.

## 28.2.3 Channels

A channel takes the 20-bit value from the counter of the selected high-speed timer and compares it to a set of two values in order to set the channel output. The first value it is compared to is the content of LEDC_HPOINT_HSCHn; if these two match, the output will be latched high. The second value is the sum of LEDC_HPOINT_HSCHn and LEDC_DUTY_HSCHn[24..4]. When this value is reached, the output is latched low.

By using these two values, the relative phase and the duty cycle of the PWM output can be set. Figure 28.2-4 illustrates this.

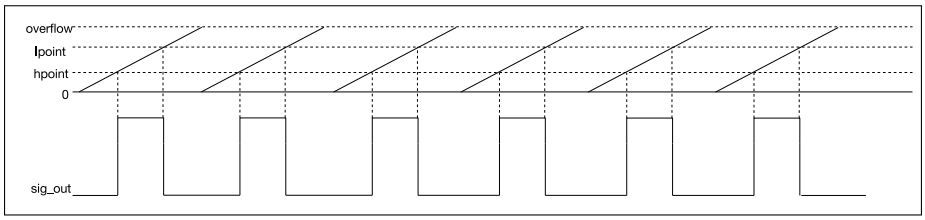



Figure 28.2-4. LED PWM Output Signal Diagram

LEDC_DUTY_HSCH*n* is a fixed-point register with four fractional bits. As mentioned before, when LEDC_DUTY_
HSCH*n*[24..4] is used in the PWM calculation directly, LEDC_DUTY_HSCH*n*[3..0] can be used to dither the output. If this value is non-zero, with a statistical chance of LEDC_DUTY_HSCH*n*[3..0]/16, the actual PWM pulse will be one cycle longer. This effectively increases the resolution of the PWM generator to 25 bits, but at the cost of a slight jitter in the duty cycle.

The channels also have the ability to automatically fade from one duty cycle value to another. This feature is enabled by setting LEDC_DUTY_START_HSCH*n*. When this bit is set, the PWM controller will automatically increment or decrement the value in LEDC_DUTY_HSCH*n*, depending on whether the bit LEDC_DUTY_INC_HSCH*n* is set or cleared, respectively. The speed the duty cycle changes is defined as such: every time the LEDC_DUTY_CYCLE_
HSCH*n* cycles, the content of LEDC_DUTY_SCALE_HSCH*n* is added to or subtracted from LEDC_DUTY_HSCH*n*[24..4]. The length of the fade can be limited by setting LEDC_DUTY_NUM_HSCH*n*: the fade will only last that number of cycles before finishing. A finished fade also generates an interrupt.

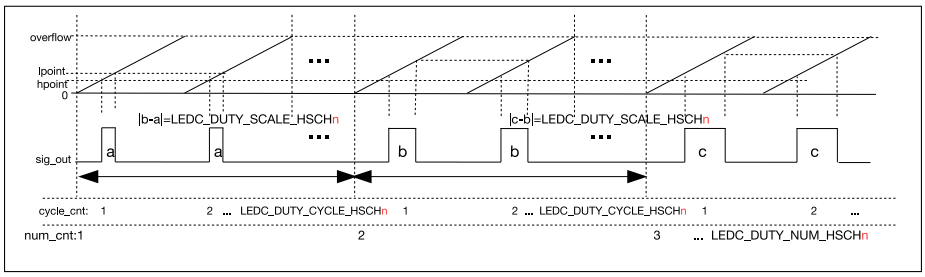



Figure 28.2-5. Output Signal Diagram of Fading Duty Cycle

Figure 28.2-5 is an illustration of this. In this configuration, LEDC_DUTY_NUM_HSCH*n*_R increases by LEDC_DUTY_
SCALE_HSCH*n* for every LEDC_DUTY_CYCLE_HSCH*n* clock cycles, which is reflected in the duty cycle of the output signal.

### Notes

- When the LEDC is in fade mode, it is not supported to perform operations (e.g., pause) on the process or configure the following registers:
  - LEDC_HPOINT_H/LSCH*n*
  - LEDC_DUTY_H/LSCH*n*

- LEDC_DUTY_START_H/LSCH*n*
- LEDC_DUTY_INC_H/LSCH*n*
- LEDC_DUTY_NUM_H/LSCH*n*
- LEDC_DUTY_CYCLE_H/LSCH*n*
- LEDC_DUTY_SCALE_H/LSCH*n*

Configure the second fade only after LEDC_DUTY_CHNG_END_HSCH*n* or LEDC_DUTY_CHNG_END_LSCH*n* interrupt is generated.

- When LEDC is in decremental fade mode and LEDC_DUTY_HSCH*n* is $2^{LEDC_HSTIMERx_DUTY_RES}$, LEDC_
DUTY_SCALE_HSCH*n* cannot be set to 1. When LEDC is in decremental fade mode and LEDC_DUTY_LSCH*n* is $2^{LEDC_LSTIMERx_DUTY_RES}$, LEDC_DUTY_SCALE_LSCH*n* cannot be set to 1.

### 28.2.4 Interrupts

- LEDC_DUTY_CHNG_END_LSCH*n*_INT: Triggered when a fade on a low-speed channel has finished.
- LEDC_DUTY_CHNG_END_HSCH*n*_INT: Triggered when a fade on a high-speed channel has finished.
- LEDC_HS_TIMER*x*_OVF_INT: Triggered when a high-speed timer has reached its maximum counter value.
- LEDC_LS_TIMER*x*_OVF_INT: Triggered when a low-speed timer has reached its maximum counter value.

## 28.3 Register Summary

The addresses in this section are relative to the LED PWM base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| LEDC_CONF_REG | Global ledc configuration register | 0x3FF59190 | R/W |
| LEDC_HSCH0_CONF0_REG | Configuration register 0 for high-speed channel 0 | 0x3FF59000 | R/W |
| LEDC_HSCH1_CONF0_REG | Configuration register 0 for high-speed channel 1 | 0x3FF59014 | R/W |
| LEDC_HSCH2_CONF0_REG | Configuration register 0 for high-speed channel 2 | 0x3FF59028 | R/W |
| LEDC_HSCH3_CONF0_REG | Configuration register 0 for high-speed channel 3 | 0x3FF5903C | R/W |
| LEDC_HSCH4_CONF0_REG | Configuration register 0 for high-speed channel 4 | 0x3FF59050 | R/W |
| LEDC_HSCH5_CONF0_REG | Configuration register 0 for high-speed channel 5 | 0x3FF59064 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| LEDC_HSCH6_CONF0_REG | Configuration register 0 for high-speed channel 6 | 0x3FF59078 | R/W |
| LEDC_HSCH7_CONF0_REG | Configuration register 0 for high-speed channel 7 | 0x3FF5908C | R/W |
| LEDC_HSCH0_CONF1_REG | Configuration register 1 for high-speed channel 0 | 0x3FF5900C | R/W |
| LEDC_HSCH1_CONF1_REG | Configuration register 1 for high-speed channel 1 | 0x3FF59020 | R/W |
| LEDC_HSCH2_CONF1_REG | Configuration register 1 for high-speed channel 2 | 0x3FF59034 | R/W |
| LEDC_HSCH3_CONF1_REG | Configuration register 1 for high-speed channel 3 | 0x3FF59048 | R/W |
| LEDC_HSCH4_CONF1_REG | Configuration register 1 for high-speed channel 4 | 0x3FF5905C | R/W |
| LEDC_HSCH5_CONF1_REG | Configuration register 1 for high-speed channel 5 | 0x3FF59070 | R/W |
| LEDC_HSCH6_CONF1_REG | Configuration register 1 for high-speed channel 6 | 0x3FF59084 | R/W |
| LEDC_HSCH7_CONF1_REG | Configuration register 1 for high-speed channel 7 | 0x3FF59098 | R/W |
| LEDC_LSCH0_CONF0_REG | Configuration register 0 for low-speed channel 0 | 0x3FF590A0 | R/W |
| LEDC_LSCH1_CONF0_REG | Configuration register 0 for low-speed channel 1 | 0x3FF590B4 | R/W |
| LEDC_LSCH2_CONF0_REG | Configuration register 0 for low-speed channel 2 | 0x3FF590C8 | R/W |
| LEDC_LSCH3_CONF0_REG | Configuration register 0 for low-speed channel 3 | 0x3FF590DC | R/W |
| LEDC_LSCH4_CONF0_REG | Configuration register 0 for low-speed channel 4 | 0x3FF590F0 | R/W |
| LEDC_LSCH5_CONF0_REG | Configuration register 0 for low-speed channel 5 | 0x3FF59104 | R/W |
| LEDC_LSCH6_CONF0_REG | Configuration register 0 for low-speed channel 6 | 0x3FF59118 | R/W |
| LEDC_LSCH7_CONF0_REG | Configuration register 0 for low-speed channel 7 | 0x3FF5912C | R/W |
| LEDC_LSCH0_CONF1_REG | Configuration register 1 for low-speed channel 0 | 0x3FF590AC | R/W |
| LEDC_LSCH1_CONF1_REG | Configuration register 1 for low-speed channel 1 | 0x3FF590C0 | R/W |
| LEDC_LSCH2_CONF1_REG | Configuration register 1 for low-speed channel 2 | 0x3FF590D4 | R/W |
| LEDC_LSCH3_CONF1_REG | Configuration register 1 for low-speed channel 3 | 0x3FF590E8 | R/W |
| LEDC_LSCH4_CONF1_REG | Configuration register 1 for low-speed channel 4 | 0x3FF590FC | R/W |
| LEDC_LSCH5_CONF1_REG | Configuration register 1 for low-speed channel 5 | 0x3FF59110 | R/W |
| LEDC_LSCH6_CONF1_REG | Configuration register 1 for low-speed channel 6 | 0x3FF59124 | R/W |
| LEDC_LSCH7_CONF1_REG | Configuration register 1 for low-speed channel 7 | 0x3FF59138 | R/W |
| **Duty-cycle registers** | | | |

| Name | Description | Address | Access |
|---|---|---|---|
| LEDC_HSCH0_DUTY_REG | Initial duty cycle for high-speed channel 0 | 0x3FF59008 | R/W |
| LEDC_HSCH1_DUTY_REG | Initial duty cycle for high-speed channel 1 | 0x3FF5901C | R/W |
| LEDC_HSCH2_DUTY_REG | Initial duty cycle for high-speed channel 2 | 0x3FF59030 | R/W |
| LEDC_HSCH3_DUTY_REG | Initial duty cycle for high-speed channel 3 | 0x3FF59044 | R/W |
| LEDC_HSCH4_DUTY_REG | Initial duty cycle for high-speed channel 4 | 0x3FF59058 | R/W |
| LEDC_HSCH5_DUTY_REG | Initial duty cycle for high-speed channel 5 | 0x3FF5906C | R/W |
| LEDC_HSCH6_DUTY_REG | Initial duty cycle for high-speed channel 6 | 0x3FF59080 | R/W |
| LEDC_HSCH7_DUTY_REG | Initial duty cycle for high-speed channel 7 | 0x3FF59094 | R/W |
| LEDC_HSCH0_DUTY_R_REG | Current duty cycle for high-speed channel 0 | 0x3FF59010 | RO |
| LEDC_HSCH1_DUTY_R_REG | Current duty cycle for high-speed channel 1 | 0x3FF59024 | RO |
| LEDC_HSCH2_DUTY_R_REG | Current duty cycle for high-speed channel 2 | 0x3FF59038 | RO |
| LEDC_HSCH3_DUTY_R_REG | Current duty cycle for high-speed channel 3 | 0x3FF5904C | RO |
| LEDC_HSCH4_DUTY_R_REG | Current duty cycle for high-speed channel 4 | 0x3FF59060 | RO |
| LEDC_HSCH5_DUTY_R_REG | Current duty cycle for high-speed channel 5 | 0x3FF59074 | RO |
| LEDC_HSCH6_DUTY_R_REG | Current duty cycle for high-speed channel 6 | 0x3FF59088 | RO |
| LEDC_HSCH7_DUTY_R_REG | Current duty cycle for high-speed channel 7 | 0x3FF5909C | RO |
| LEDC_LSCH0_DUTY_REG | Initial duty cycle for low-speed channel 0 | 0x3FF590A8 | R/W |
| LEDC_LSCH1_DUTY_REG | Initial duty cycle for low-speed channel 1 | 0x3FF590BC | R/W |
| LEDC_LSCH2_DUTY_REG | Initial duty cycle for low-speed channel 2 | 0x3FF590D0 | R/W |
| LEDC_LSCH3_DUTY_REG | Initial duty cycle for low-speed channel 3 | 0x3FF590E4 | R/W |
| LEDC_LSCH4_DUTY_REG | Initial duty cycle for low-speed channel 4 | 0x3FF590F8 | R/W |
| LEDC_LSCH5_DUTY_REG | Initial duty cycle for low-speed channel 5 | 0x3FF5910C | R/W |
| LEDC_LSCH6_DUTY_REG | Initial duty cycle for low-speed channel 6 | 0x3FF59120 | R/W |
| LEDC_LSCH7_DUTY_REG | Initial duty cycle for low-speed channel 7 | 0x3FF59134 | R/W |
| LEDC_LSCH0_DUTY_R_REG | Current duty cycle for low-speed channel 0 | 0x3FF590B0 | RO |
| LEDC_LSCH1_DUTY_R_REG | Current duty cycle for low-speed channel 1 | 0x3FF590C4 | RO |
| LEDC_LSCH2_DUTY_R_REG | Current duty cycle for low-speed channel 2 | 0x3FF590D8 | RO |
| LEDC_LSCH3_DUTY_R_REG | Current duty cycle for low-speed channel 3 | 0x3FF590EC | RO |
| LEDC_LSCH4_DUTY_R_REG | Current duty cycle for low-speed channel 4 | 0x3FF59100 | RO |
| LEDC_LSCH5_DUTY_R_REG | Current duty cycle for low-speed channel 5 | 0x3FF59114 | RO |
| LEDC_LSCH6_DUTY_R_REG | Current duty cycle for low-speed channel 6 | 0x3FF59128 | RO |
| LEDC_LSCH7_DUTY_R_REG | Current duty cycle for low-speed channel 7 | 0x3FF5913C | RO |
| LEDC_HSCH0_HPOINT_REG | High-level comparator value configuration register for high-speed channel 0 | 0x3FF59004 | R/W |
| LEDC_HSCH1_HPOINT_REG | High-level comparator value configuration register for high-speed channel 1 | 0x3FF59018 | R/W |
| LEDC_HSCH2_HPOINT_REG | High-level comparator value configuration register for high-speed channel 2 | 0x3FF5902C | R/W |
| LEDC_HSCH3_HPOINT_REG | High-level comparator value configuration register for high-speed channel 3 | 0x3FF59040 | R/W |
| LEDC_HSCH4_HPOINT_REG | High-level comparator value configuration register for high-speed channel 4 | 0x3FF59054 | R/W |

| Name | Description | Address | Access |
|---|---|---|---|
| LEDC_HSCH5_HPOINT_REG | High-level comparator value configuration register for high-speed channel 5 | 0x3FF59068 | R/W |
| LEDC_HSCH6_HPOINT_REG | High-level comparator value configuration register for high-speed channel 6 | 0x3FF5907C | R/W |
| LEDC_HSCH7_HPOINT_REG | High-level comparator value configuration register for high-speed channel 7 | 0x3FF59090 | R/W |
| LEDC_LSCH0_HPOINT_REG | High-level comparator value configuration register for low-speed channel 0 | 0x3FF590A4 | R/W |
| LEDC_LSCH1_HPOINT_REG | High-level comparator value configuration register for low-speed channel 1 | 0x3FF590B8 | R/W |
| LEDC_LSCH2_HPOINT_REG | High-level comparator value configuration register for low-speed channel 2 | 0x3FF590CC | R/W |
| LEDC_LSCH3_HPOINT_REG | High-level comparator value configuration register for low-speed channel 3 | 0x3FF590E0 | R/W |
| LEDC_LSCH4_HPOINT_REG | High-level comparator value configuration register for low-speed channel 4 | 0x3FF590F4 | R/W |
| LEDC_LSCH5_HPOINT_REG | High-level comparator value configuration register for low-speed channel 5 | 0x3FF59108 | R/W |
| LEDC_LSCH6_HPOINT_REG | High-level comparator value configuration register for low-speed channel 6 | 0x3FF5911C | R/W |
| LEDC_LSCH7_HPOINT_REG | High-level comparator value configuration register for low-speed channel 7 | 0x3FF59130 | R/W |
| **Timer registers** | | | |
| LEDC_HSTIMER0_CONF_REG | High-speed timer 0 configuration | 0x3FF59140 | R/W |
| LEDC_HSTIMER1_CONF_REG | High-speed timer 1 configuration | 0x3FF59148 | R/W |
| LEDC_HSTIMER2_CONF_REG | High-speed timer 2 configuration | 0x3FF59150 | R/W |
| LEDC_HSTIMER3_CONF_REG | High-speed timer 3 configuration | 0x3FF59158 | R/W |
| LEDC_HSTIMER0_VALUE_REG | High-speed timer 0 current counter value | 0x3FF59144 | RO |
| LEDC_HSTIMER1_VALUE_REG | High-speed timer 1 current counter value | 0x3FF5914C | RO |
| LEDC_HSTIMER2_VALUE_REG | High-speed timer 2 current counter value | 0x3FF59154 | RO |
| LEDC_HSTIMER3_VALUE_REG | High-speed timer 3 current counter value | 0x3FF5915C | RO |
| LEDC_LSTIMER0_CONF_REG | Low-speed timer 0 configuration | 0x3FF59160 | R/W |
| LEDC_LSTIMER1_CONF_REG | Low-speed timer 1 configuration | 0x3FF59168 | R/W |
| LEDC_LSTIMER2_CONF_REG | Low-speed timer 2 configuration | 0x3FF59170 | R/W |
| LEDC_LSTIMER3_CONF_REG | Low-speed timer 3 configuration | 0x3FF59178 | R/W |
| LEDC_LSTIMER0_VALUE_REG | Low-speed timer 0 current counter value | 0x3FF59164 | RO |
| LEDC_LSTIMER1_VALUE_REG | Low-speed timer 1 current counter value | 0x3FF5916C | RO |
| LEDC_LSTIMER2_VALUE_REG | Low-speed timer 2 current counter value | 0x3FF59174 | RO |
| LEDC_LSTIMER3_VALUE_REG | Low-speed timer 3 current counter value | 0x3FF5917C | RO |
| **Interrupt registers** | | | |
| LEDC_INT_RAW_REG | Raw interrupt status | 0x3FF59180 | RO |
| LEDC_INT_ST_REG | Masked interrupt status | 0x3FF59184 | RO |
| LEDC_INT_ENA_REG | Interrupt enable bits | 0x3FF59188 | R/W |
| LEDC_INT_CLR_REG | Interrupt clear bits | 0x3FF5918C | WO |

## 28.4 Registers

The addresses in this section are relative to the LED PWM base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 28.3 *Register Summary*.

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

**Register 28.1. LEDC_HSCH*n*_CONF0_REG (*n*: 0-7) (0x00+0x14**n*)**

| (reserved) | LEDC_IDLE_LV_HSCH*n* | LEDC_SIG_OUT_EN_HSCH*n* | LEDC_TIMER_SEL_HSCH*n* | |
|---|---|---|---|---|
| 31 … 4 | 3 | 2 | 1 … 0 | |
| 0x00000000 | 0 | 0 | 0 | Reset |

**LEDC_IDLE_LV_HSCH*n*** This bit is used to control the output value when high-speed channel *n* is inactive. (R/W)

**LEDC_SIG_OUT_EN_HSCH*n*** This is the output enable control bit for high-speed channel *n*. (R/W)

**LEDC_TIMER_SEL_HSCH*n*** There are four high-speed timers. These two bits are used to select one of them for high-speed channel *n*: (R/W)

0: select hstimer0;

1: select hstimer1;

2: select hstimer2;

3: select hstimer3.

**Register 28.2. LEDC_HSCH*n*_HPOINT_REG (*n*: 0-7) (0x04+0x14**n*)**

| (reserved) | LEDC_HPOINT_HSCH*n* | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0x0000 | 0x000000 | Reset |

**LEDC_HPOINT_HSCH*n*** The output value changes to high when htimerx(x=[0,3]), selected by high-speed channel *n*, has reached LEDC_HPOINT_HSCH*n*[19:0]. (R/W)

**Register 28.3. LEDC_HSCH*n*_DUTY_REG (*n*: 0-7) (0x08+0x14**n*)**

| (reserved) | LEDC_DUTY_HSCH*n* | |
|---|---|---|
| 31 … 25 | 24 … 0 | |
| 0x00 | 0x0000000 | Reset |

**LEDC_DUTY_HSCH*n*** The register is used to control output duty. When hstimerx(x=[0,3]), selected by high-speed channel *n*, has reached LEDC_LPOINT_HSCH*n*, the output signal changes to low. (R/W)

LEDC_LPOINT_HSCH*n*=LEDC_LPOINT_HSCH*n*[19:0]+LEDC_DUTY_HSCH*n*[24:4] (1)

LEDC_LPOINT_HSCH*n*=LEDC_LPOINT_HSCH*n*[19:0]+LEDC_DUTY_HSCH*n*[24:4] +1) (2)

See the Functional Description for more information on when (1) or (2) is chosen.

**Register 28.4. LEDC_HSCH*n*_CONF1_REG (*n*: 0-7) (0x0C+0x14**n*)**

| LEDC_DUTY_START_HSCH*n* / LEDC_DUTY_INC_HSCH*n* | | LEDC_DUTY_NUM_HSCH*n* | LEDC_DUTY_CYCLE_HSCH*n* | LEDC_DUTY_SCALE_HSCH*n* | |
|---|---|---|---|---|---|
| 31 | 30 | 29 … 20 | 19 … 10 | 9 … 0 | |
| 0 | 1 | 0x000 | 0x000 | 0x000 | Reset |

**LEDC_DUTY_START_HSCH*n*** When LEDC_DUTY_NUM_HSCH*n*, LEDC_DUTY_CYCLE_HSCH*n* and LEDC_DUTY_SCALE_HSCH*n* has been configured, these register will not take effect until LEDC_DUTY_START_HSCH*n* is set. This bit is automatically cleared by hardware. (R/W)

**LEDC_DUTY_INC_HSCH*n*** This register is used to increase or decrease the duty of output signal for high-speed channel *n*. (R/W)

**LEDC_DUTY_NUM_HSCH*n*** This register is used to control the number of times the duty cycle is increased or decreased for high-speed channel *n*. (R/W)

**LEDC_DUTY_CYCLE_HSCH*n*** This register is used to increase or decrease the duty cycle every time LEDC_DUTY_CYCLE_HSCH*n* cycles for high-speed channel *n*. (R/W)

**LEDC_DUTY_SCALE_HSCH*n*** This register is used to increase or decrease the step scale for high-speed channel *n*. (R/W)

**Register 28.5. LEDC_HSCHn_DUTY_R_REG (n: 0-7) (0x10+0x14*n)**

| (reserved) | LEDC_DUTY_HSCHn_R | |
|---|---|---|
| 31 … 25 | 24 … 0 | |
| 0x00 | 0x0000000 | Reset |

**LEDC_DUTY_HSCHn_R** This register represents the current duty cycle of the output signal for high-speed channel *n*. (RO)

**Register 28.6. LEDC_LSCHn_CONF0_REG (n: 0-7) (0xA0+0x14*n)**

| (reserved) | LEDC_PARA_UP_LSCHn | LEDC_IDLE_LV_LSCHn | LEDC_SIG_OUT_EN_LSCHn | LEDC_TIMER_SEL_LSCHn | |
|---|---|---|---|---|---|
| 31 … 5 | 4 | 3 | 2 | 1 … 0 | |
| 0x0000000 | 0 | 0 | 0 | 0 | Reset |

**LEDC_PARA_UP_LSCHn** This bit is used to update register LEDC_LSCHn_HPOINT and LEDC_LSCHn_DUTY for low-speed channel *n*. (R/W)

**LEDC_IDLE_LV_LSCHn** This bit is used to control the output value, when low-speed channel *n* is inactive. (R/W)

**LEDC_SIG_OUT_EN_LSCHn** This is the output enable control bit for low-speed channel *n*. (R/W)

**LEDC_TIMER_SEL_LSCHn** There are four low-speed timers, the two bits are used to select one of them for low-speed channel *n*. (R/W)

0: select lstimer0;

1: select lstimer1;

2: select lstimer2;

3: select lstimer3.

**Register 28.7. LEDC_LSCH*n*_HPOINT_REG (*n*: 0-7) (0xA4+0x14**n*)**

| (reserved) | LEDC_HPOINT_LSCH*n* | |
|---|---|---|
| 31 … 20 | 19 … 0 | |
| 0x0000 | 0x000000 | Reset |

**LEDC_HPOINT_LSCH*n*** The output value changes to high when lstimerx(x=[0,3]), selected by low-speed channel *n*, has reached LEDC_HPOINT_LSCH*n*[19:0]. (R/W)

**Register 28.8. LEDC_LSCH*n*_DUTY_REG (*n*: 0-7) (0xA8+0x14**n*)**

| (reserved) | LEDC_DUTY_LSCH*n* | |
|---|---|---|
| 31 … 25 | 24 … 0 | |
| 0x00 | 0x0000000 | Reset |

**LEDC_DUTY_LSCH*n*** The register is used to control output duty. When lstimerx(x=[0,3]), chosen by low-speed channel *n*, has reached LEDC_LPOINT_LSCH*n*,the output signal changes to low. (R/W)

LEDC_LPOINT_LSCH*n*=(LEDC_HPOINT_LSCH*n*[19:0]+LEDC_DUTY_LSCH*n*[24:4]) (1)

LEDC_LPOINT_LSCH*n*=(LEDC_HPOINT_LSCH*n*[19:0]+LEDC_DUTY_LSCH*n*[24:4] +1) (2)

See the Functional Description for more information on when (1) or (2) is chosen.

**Register 28.9. LEDC_LSCH*n*_CONF1_REG (*n*: 0-7) (0xAC+0x14**n*)**

| LEDC_DUTY_START_LSCH*n* / LEDC_DUTY_INC_LSCH*n* | | LEDC_DUTY_NUM_LSCH*n* | LEDC_DUTY_CYCLE_LSCH*n* | LEDC_DUTY_SCALE_LSCH*n* | |
|---|---|---|---|---|---|
| 31 | 30 | 29 … 20 | 19 … 10 | 9 … 0 | |
| 0 | 1 | 0x000 | 0x000 | 0x000 | Reset |

**LEDC_DUTY_START_LSCH*n*** When LEDC_DUTY_NUM_HSCH*n*, LEDC_DUTY_CYCLE_HSCH*n* and LEDC_DUTY_SCALE_HSCH*n* have been configured, these settings will not take effect until set LEDC_DUTY_START_HSCH*n*. This bit is automatically cleared by hardware. (R/W)

**LEDC_DUTY_INC_LSCH*n*** This register is used to increase or decrease the duty of output signal for low-speed channel *n*. (R/W)

**LEDC_DUTY_NUM_LSCH*n*** This register is used to control the number of times the duty cycle is increased or decreased for low-speed channel *n*. (R/W)

**LEDC_DUTY_CYCLE_LSCH*n*** This register is used to increase or decrease the duty every LEDC_DUTY_CYCLE_LSCH*n* cycles for low-speed channel *n*. (R/W)

**LEDC_DUTY_SCALE_LSCH*n*** This register is used to increase or decrease the step scale for low-speed channel *n*. (R/W)

**Register 28.10. LEDC_LSCH*n*_DUTY_R_REG (*n*: 0-7) (0xB0+0x14**n*)**

| (reserved) | LEDC_DUTY_LSCH*n*_R | |
|---|---|---|
| 31 … 25 | 24 … 0 | |
| 0x00 | 0x0000000 | Reset |

**LEDC_DUTY_LSCH*n*_R** This register represents the current duty of the output signal for low-speed channel *n*. (RO)

**Register 28.11. LEDC_HSTIMERx_CONF_REG (x: 0-3) (0x140+8*x)**

| (reserved) | LEDC_TICK_SEL_HSTIMERx | LEDC_HSTIMERx_RST | LEDC_HSTIMERx_PAUSE | LEDC_CLK_DIV_NUM_HSTIMERx | LEDC_HSTIMERx_DUTY_RES | |
|---|---|---|---|---|---|---|
| 31 26 | 25 | 24 | 23 | 22 5 | 4 0 | |
| 0x00 | 0 | 1 | 0 | 0x00000 | 0x00 | Reset |

**LEDC_TICK_SEL_HSTIMERx** This bit is used to select APB_CLK or REF_TICK for high-speed timer x. (R/W)

1: APB_CLK;

0: REF_TICK.

**LEDC_HSTIMERx_RST** This bit is used to reset high-speed timer x. The counter value will be 'zero' after reset. (R/W)

**LEDC_HSTIMERx_PAUSE** This bit is used to suspend the counter in high-speed timer x. (R/W)

**LEDC_CLK_DIV_NUM_HSTIMERx** This register is used to configure the division factor for the divider in high-speed timer x. The least significant eight bits represent the fractional part. (R/W)

**LEDC_HSTIMERx_DUTY_RES** This register is used to control the range of the counter in high-speed timer x. The counter range is [0, $2^{LEDC_HSTIMERx_DUTY_RES}$], the maximum bit width for counter is 20. (R/W)

**Register 28.12. LEDC_HSTIMERx_VALUE_REG (x: 0-3) (0x144+8*x)**

| (reserved) | LEDC_HSTIMERx_CNT | |
|---|---|---|
| 31 20 | 19 0 | |
| 0x0000 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**LEDC_HSTIMERx_CNT** Software can read this register to get the current counter value of high-speed timer x. (RO)

**Register 28.13. LEDC_LSTIMERx_CONF_REG (x: 0-3) (0x160+8*x)**

| (reserved) | LEDC_LSTIMERx_PARA_UP | LEDC_TICK_SEL_LSTIMERx | LEDC_LSTIMERx_RST | LEDC_LSTIMERx_PAUSE | LEDC_CLK_DIV_NUM_LSTIMERx | LEDC_LSTIMERx_DUTY_RES | |
|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 | 23 | 22 5 | 4 0 | |
| 0x00 | 0 | 0 | 1 | 0 | 0x00000 | 0x00 | Reset |

**LEDC_LSTIMERx_PARA_UP** Set this bit to update LEDC_CLK_DIV_NUM_LSTIMEx and LEDC_LSTIMERx_DUTY_RES. (R/W)

**LEDC_TICK_SEL_LSTIMERx** This bit is used to select RTC_SLOW_CLK or REF_TICK for low-speed timer x. (R/W)

1: RTC_SLOW_CLK;

0: REF_TICK.

**LEDC_LSTIMERx_RST** This bit is used to reset low-speed timer x. The counter will show 0 after reset. (R/W)

**LEDC_LSTIMERx_PAUSE** This bit is used to suspend the counter in low-speed timer x. (R/W)

**LEDC_CLK_DIV_NUM_LSTIMERx** This register is used to configure the division factor for the divider in low-speed timer x. The least significant eight bits represent the fractional part. (R/W)

**LEDC_LSTIMERx_DUTY_RES** This register is used to control the range of the counter in low-speed timer x. The counter range is [0, $2^{LEDC_LSTIMERx_DUTY_RES}$], the max bit width for counter is 20. (R/W)

**Register 28.14. LEDC_LSTIMERx_VALUE_REG (x: 0-3) (0x164+8*x)**

| (reserved) | LEDC_LSTIMERx_CNT | |
|---|---|---|
| 31 20 | 19 0 | |
| 0x0000 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Reset |

**LEDC_LSTIMERx_CNT** Software can read this register to get the current counter value of low-speed timer x. (RO)

**Register 28.15. LEDC_INT_RAW_REG (0x0180)**

| 31–24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | LEDC_DUTY_CHNG_END_LSCH7_INT_RAW | LEDC_DUTY_CHNG_END_LSCH6_INT_RAW | LEDC_DUTY_CHNG_END_LSCH5_INT_RAW | LEDC_DUTY_CHNG_END_LSCH4_INT_RAW | LEDC_DUTY_CHNG_END_LSCH3_INT_RAW | LEDC_DUTY_CHNG_END_LSCH2_INT_RAW | LEDC_DUTY_CHNG_END_LSCH1_INT_RAW | LEDC_DUTY_CHNG_END_LSCH0_INT_RAW | LEDC_DUTY_CHNG_END_HSCH7_INT_RAW | LEDC_DUTY_CHNG_END_HSCH6_INT_RAW | LEDC_DUTY_CHNG_END_HSCH5_INT_RAW | LEDC_DUTY_CHNG_END_HSCH4_INT_RAW | LEDC_DUTY_CHNG_END_HSCH3_INT_RAW | LEDC_DUTY_CHNG_END_HSCH2_INT_RAW | LEDC_DUTY_CHNG_END_HSCH1_INT_RAW | LEDC_DUTY_CHNG_END_HSCH0_INT_RAW | LEDC_LSTIMER3_OVF_INT_RAW | LEDC_LSTIMER2_OVF_INT_RAW | LEDC_LSTIMER1_OVF_INT_RAW | LEDC_LSTIMER0_OVF_INT_RAW | LEDC_HSTIMER3_OVF_INT_RAW | LEDC_HSTIMER2_OVF_INT_RAW | LEDC_HSTIMER1_OVF_INT_RAW | LEDC_HSTIMER0_OVF_INT_RAW | |
| 0 | 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_DUTY_CHNG_END_LSCH*n*_INT_RAW** The raw interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

**LEDC_DUTY_CHNG_END_HSCH*n*_INT_RAW** The raw interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

**LEDC_LSTIMER*x*_OVF_INT_RAW** The raw interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

**LEDC_HSTIMER*x*_OVF_INT_RAW** The raw interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

**Register 28.16. LEDC_INT_ST_REG (0x0184)**

| 31–24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | LEDC_DUTY_CHNG_END_LSCH7_INT_ST | LEDC_DUTY_CHNG_END_LSCH6_INT_ST | LEDC_DUTY_CHNG_END_LSCH5_INT_ST | LEDC_DUTY_CHNG_END_LSCH4_INT_ST | LEDC_DUTY_CHNG_END_LSCH3_INT_ST | LEDC_DUTY_CHNG_END_LSCH2_INT_ST | LEDC_DUTY_CHNG_END_LSCH1_INT_ST | LEDC_DUTY_CHNG_END_LSCH0_INT_ST | LEDC_DUTY_CHNG_END_HSCH7_INT_ST | LEDC_DUTY_CHNG_END_HSCH6_INT_ST | LEDC_DUTY_CHNG_END_HSCH5_INT_ST | LEDC_DUTY_CHNG_END_HSCH4_INT_ST | LEDC_DUTY_CHNG_END_HSCH3_INT_ST | LEDC_DUTY_CHNG_END_HSCH2_INT_ST | LEDC_DUTY_CHNG_END_HSCH1_INT_ST | LEDC_DUTY_CHNG_END_HSCH0_INT_ST | LEDC_LSTIMER3_OVF_INT_ST | LEDC_LSTIMER2_OVF_INT_ST | LEDC_LSTIMER1_OVF_INT_ST | LEDC_LSTIMER0_OVF_INT_ST | LEDC_HSTIMER3_OVF_INT_ST | LEDC_HSTIMER2_OVF_INT_ST | LEDC_HSTIMER1_OVF_INT_ST | LEDC_HSTIMER0_OVF_INT_ST | |
| 0 | 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_DUTY_CHNG_END_LSCH*n*_INT_ST** The masked interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

**LEDC_DUTY_CHNG_END_HSCH*n*_INT_ST** The masked interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

**LEDC_LSTIMER*x*_OVF_INT_ST** The masked interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

**LEDC_HSTIMER*x*_OVF_INT_ST** The masked interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

### Register 28.17. LEDC_INT_ENA_REG (0x0188)

| (reserved) | LEDC_DUTY_CHNG_END_LSCH7_INT_ENA | LEDC_DUTY_CHNG_END_LSCH6_INT_ENA | LEDC_DUTY_CHNG_END_LSCH5_INT_ENA | LEDC_DUTY_CHNG_END_LSCH4_INT_ENA | LEDC_DUTY_CHNG_END_LSCH3_INT_ENA | LEDC_DUTY_CHNG_END_LSCH2_INT_ENA | LEDC_DUTY_CHNG_END_LSCH1_INT_ENA | LEDC_DUTY_CHNG_END_LSCH0_INT_ENA | LEDC_DUTY_CHNG_END_HSCH7_INT_ENA | LEDC_DUTY_CHNG_END_HSCH6_INT_ENA | LEDC_DUTY_CHNG_END_HSCH5_INT_ENA | LEDC_DUTY_CHNG_END_HSCH4_INT_ENA | LEDC_DUTY_CHNG_END_HSCH3_INT_ENA | LEDC_DUTY_CHNG_END_HSCH2_INT_ENA | LEDC_DUTY_CHNG_END_HSCH1_INT_ENA | LEDC_DUTY_CHNG_END_HSCH0_INT_ENA | LEDC_LSTIMER3_OVF_INT_ENA | LEDC_LSTIMER2_OVF_INT_ENA | LEDC_LSTIMER1_OVF_INT_ENA | LEDC_LSTIMER0_OVF_INT_ENA | LEDC_HSTIMER3_OVF_INT_ENA | LEDC_HSTIMER2_OVF_INT_ENA | LEDC_HSTIMER1_OVF_INT_ENA | LEDC_HSTIMER0_OVF_INT_ENA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 – 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_DUTY_CHNG_END_LSCH*n*_INT_ENA** The interrupt enable bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (R/W)

**LEDC_DUTY_CHNG_END_HSCH*n*_INT_ENA** The interrupt enable bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (R/W)

**LEDC_LSTIMER*x*_OVF_INT_ENA** The interrupt enable bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (R/W)

**LEDC_HSTIMER*x*_OVF_INT_ENA** The interrupt enable bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (R/W)

### Register 28.18. LEDC_INT_CLR_REG (0x018C)

| (reserved) | LEDC_DUTY_CHNG_END_LSCH7_INT_CLR | LEDC_DUTY_CHNG_END_LSCH6_INT_CLR | LEDC_DUTY_CHNG_END_LSCH5_INT_CLR | LEDC_DUTY_CHNG_END_LSCH4_INT_CLR | LEDC_DUTY_CHNG_END_LSCH3_INT_CLR | LEDC_DUTY_CHNG_END_LSCH2_INT_CLR | LEDC_DUTY_CHNG_END_LSCH1_INT_CLR | LEDC_DUTY_CHNG_END_LSCH0_INT_CLR | LEDC_DUTY_CHNG_END_HSCH7_INT_CLR | LEDC_DUTY_CHNG_END_HSCH6_INT_CLR | LEDC_DUTY_CHNG_END_HSCH5_INT_CLR | LEDC_DUTY_CHNG_END_HSCH4_INT_CLR | LEDC_DUTY_CHNG_END_HSCH3_INT_CLR | LEDC_DUTY_CHNG_END_HSCH2_INT_CLR | LEDC_DUTY_CHNG_END_HSCH1_INT_CLR | LEDC_DUTY_CHNG_END_HSCH0_INT_CLR | LEDC_LSTIMER3_OVF_INT_CLR | LEDC_LSTIMER2_OVF_INT_CLR | LEDC_LSTIMER1_OVF_INT_CLR | LEDC_LSTIMER0_OVF_INT_CLR | LEDC_HSTIMER3_OVF_INT_CLR | LEDC_HSTIMER2_OVF_INT_CLR | LEDC_HSTIMER1_OVF_INT_CLR | LEDC_HSTIMER0_OVF_INT_CLR | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 – 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_DUTY_CHNG_END_LSCH*n*_INT_CLR** Set this bit to clear the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (WO)

**LEDC_DUTY_CHNG_END_HSCH*n*_INT_CLR** Set this bit to clear the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (WO)

**LEDC_LSTIMER*x*_OVF_INT_CLR** Set this bit to clear the LEDC_LSTIMER*x*_OVF_INT interrupt. (WO)

**LEDC_HSTIMER*x*_OVF_INT_CLR** Set this bit to clear the LEDC_HSTIMER*x*_OVF_INT interrupt. (WO)

**Register 28.19. LEDC_CONF_REG (0x0190)**

| (reserved) | LEDC_APB_CLK_SEL | |
|---|---|---|
| 31 … 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 | 0 | Reset |

**LEDC_APB_CLK_SEL** This bit is used to set the frequency of RTC_SLOW_CLK. (R/W)

0: 8 MHz;

1: 80 MHz.

# Chapter 29

# Motor Control PWM (MCPWM)

## 29.1 Introduction

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 timing and control resources inside are allocated into two major types of submodules: PWM timers and PWM operators. 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 necessary control resources to generate waveform pairs for one PWM channel. The MCPWM peripheral also contains a dedicated capture submodule that is used in systems where accurate timing of external events is important.

ESP32 contains two MCPWM peripherals: MCPWM0 and MCPWM1. Their control registers are located in 4-KB memory blocks starting at memory locations 0x3FF5E000 and 0x3FF6C000 respectively.

## 29.2 Features

Each MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, and a capture module. Figure 29.2-1 shows the submodules inside and the signals on the interface. PWM timers are used for generating timing references. The PWM operators generate 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 references to produce related PWM signals. PWM operators can also use different PWM timers' values to produce the PWM signals that work alone. Different PWM timers can also be synced together.

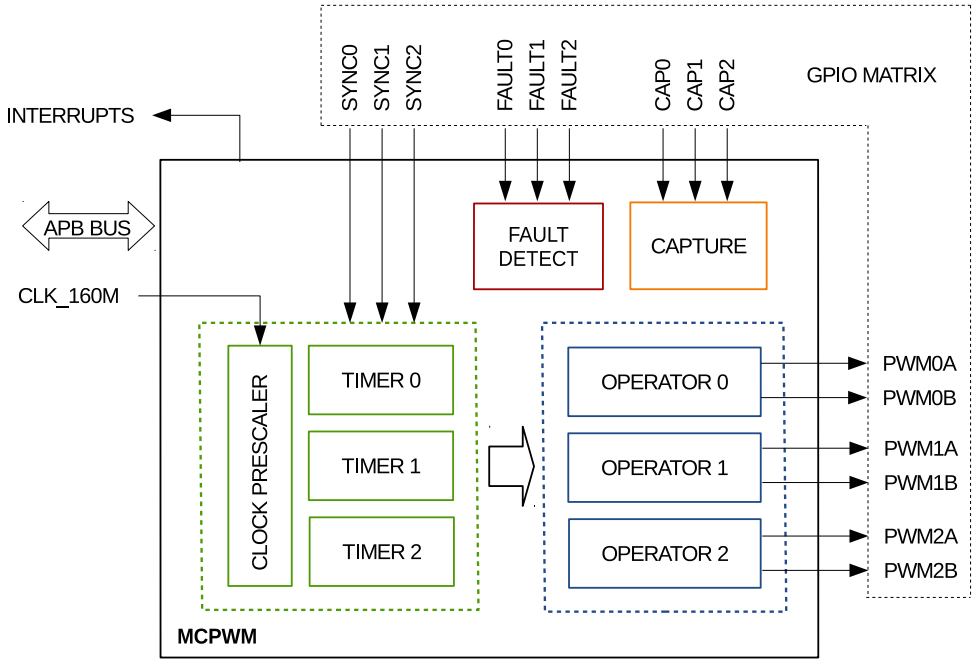



Figure 29.2-1. MCPWM Module Overview

An overview of the submodules' function in Figure 29.2-1 is shown below:

- 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 can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler' restart, so that the timer's clock can also be synced. The source of the sync can come from any GPIO or any other PWM timer's sync_out.
- 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 and asymmetric configuration.
  - Software, asynchronous override control of PWM signals.
  - 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 drives 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 allocated on fault condition 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
  - 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 from current/voltage sensors
  - Three individual capture channels, each of which has a time-stamp register (32 bits)
  - Selection of edge polarity and prescaling of input capture signal
  - The capture timer can sync with a PWM timer or external signals.
  - Interrupt on each of the three capture channels

# 29.3 Submodules

## 29.3.1 Overview

This section lists the configuration parameters of key submodules. For information on adjusting a specific parameter, e.g. synchronization source of PWM timer, please refer to Section 29.3.2 for details.

### 29.3.1.1 Prescaler Submodule

Figure 29.3-1. Prescaler Submodule

Configuration parameter:

- Scale the PWM clock according to CLK_160M.

#### 29.3.1.2 Timer Submodule

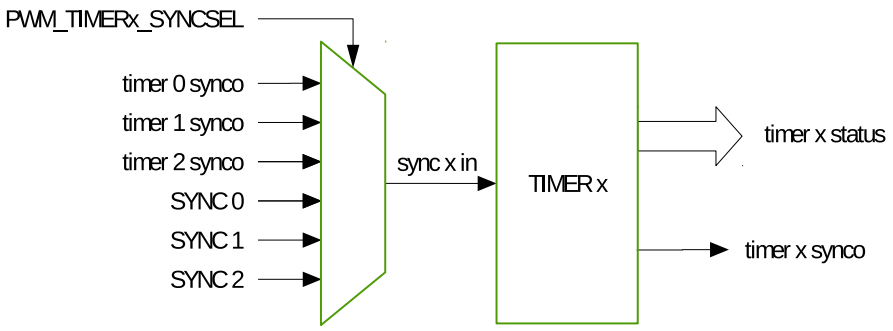



Figure 29.3-2. Timer Submodule

Configuration parameters:

- Set the PWM timer frequency or period.
- Configure the working mode for the timer:
  - Count-Up Mode: for asymmetric PWM outputs
  - Count-Down Mode: for asymmetric PWM outputs
  - Count-Up-Down Mode: for symmetric PWM outputs
- Configure the the reloading phase (including the value and the phase) used during software and hardware synchronization.
- Synchronize the PWM timers with each other. Either hardware or software synchronization may be used.
- Configure the source of the PWM timer's the synchronization input to one of the seven sources below:
  - The three PWM timer's synchronization outputs.
  - Three synchronization signals from the GPIO matrix: SYNC0, SYNC1, SYNC2.
  - No synchronization input signal selected
- Configure the source of the PWM timer's synchronization output to one of the four sources below:
  - Synchronization input signal
  - Event generated when value of the PWM timer is equal to zero
  - Event generated when value of the PWM timer is equal to period
  - No synchronization output generated
- Configure the method of period updating.

#### 29.3.1.3 Operator Submodule

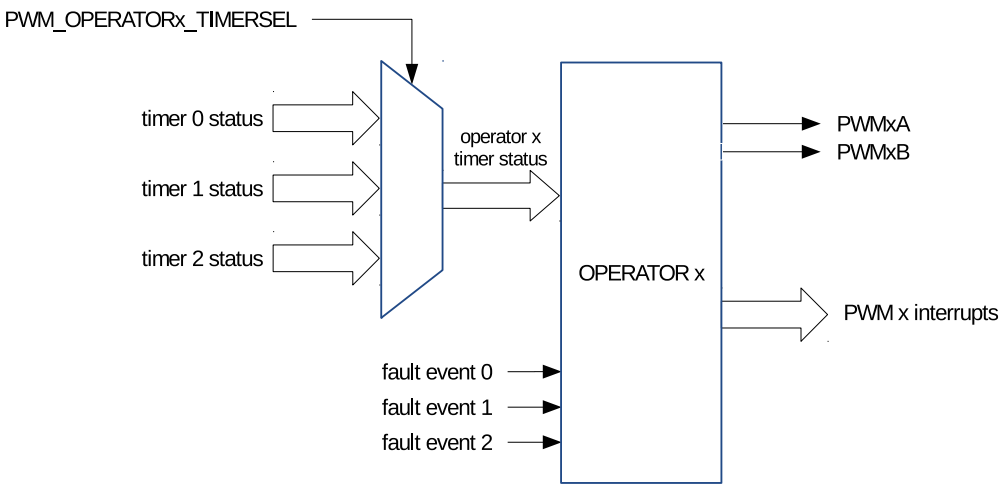



Figure 29.3-3. Operator Submodule

The configuration parameters of the operator submodule are shown in Table 29.3-1.

Table 29.3-1. Configuration Parameters of the Operator Submodule

| Submodule | Configuration Parameter or Option |
|---|---|
| PWM Generator | • Set up the PWM duty cycle for PWM*x*A and/or PWM*x*B output.<br>• Set up at which time the timing events occur.<br>• Define what action should be taken on timing events:<br>– Switch high or low PWM*x*A and/or PWM*x*B outputs<br>– Toggle PWM*x*A and/or PWM*x*B outputs<br>– Take no action on outputs<br>• Use direct s/w control to force the state of PWM outputs<br>• Add a dead time to raising and / or failing edge on PWM outputs.<br>• Configure update method for this submodule. |
| Dead Time Generator | • Control of complementary dead time relationship between upper and lower switches.<br>• Specify the dead time on rising edge.<br>• Specify the dead time on falling edge.<br>• Bypass the dead time generator module. The PWM waveform will pass through without inserting dead time.<br>• Allow PWM*x*B phase shifting with respect to the PWM*x*A output.<br>• Configure updating method for this submodule. |

| Submodule | Configuration Parameter or Option |
|---|---|
| PWM Carrier | • Enable carrier and set up carrier frequency.<br>• Configure duration of the first pulse in the carrier waveform.<br>• Set up the duty cycle of the following pulses.<br>• Bypass the PWM carrier module. The PWM waveform will be passed through without modification. |
| Fault Handler | • Configure if and how the PWM module should react the fault event signals.<br>• Specify the action taken when a fault event occurs:<br>– Force PWM*x*A and/or PWM*x*B high.<br>– Force PWM*x*A and/or PWM*x*B low.<br>– Configure PWM*x*A and/or PWM*x*B to ignore any fault event.<br>• Configure how often the PWM should react to fault events:<br>– One-shot<br>– Cycle-by-cycle<br>• Generate interrupts.<br>• Bypass the fault handler submodule entirely.<br>• Set up an option for cycle-by-cycle actions clearing.<br>• If desired, independently-configured actions can be taken when time-base counter is counting down or up. |

#### 29.3.1.4 Fault Detection Submodule

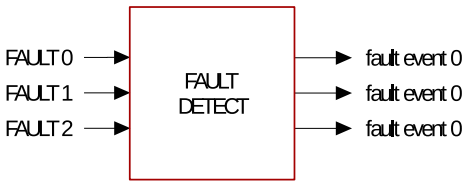



**Figure 29.3-4. Fault Detection Submodule**

Configuration parameters:

- Enable fault event generation and configure the polarity of fault event generation for every fault signal
- Generate fault event interrupts

#### 29.3.1.5 Capture Submodule

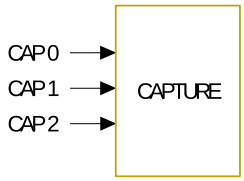



Figure 29.3-5. Capture Submodule

Configuration parameters:

- Select the edge polarity and prescaling of the capture input.
- Set up a software-triggered capture.
- Configure the capture timer's sync trigger and sync phase.
- Software syncs the capture timer.

### 29.3.2 PWM Timer Submodule

Each MCPWM module has three PWM timer submodules. Any of them can determine the necessary event timing for any of the three PWM operator submodules. Built-in synchronization logic allows multiple PWM timer submodules, in one or more MCPWM modules, to work together as a system, when using synchronization signals from the GPIO matrix.

#### 29.3.2.1 Configurations of the PWM Timer Submodule

Users can configure the following functions of the PWM timer submodule:

- Control how often events occur by specifying the PWM timer frequency or period.
- Configure a particular PWM timer to synchronize with other PWM timers or modules.
- Get a PWM timer in phase with other PWM timers or modules.
- Set one of 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 PWM_TIMER*x*_PRESCALE of register PWM_TIMER0_CFG0_REG. The PWM timer increments or decrements at a slower pace, depending on the setting of this register.

#### 29.3.2.2 PWM Timer's Working Modes and Timing Event Generation

The PWM timer has three working modes, selected by the PWM*x* timer mode register:

- Count-Up Mode:
  In this mode, the PWM timer increments from zero until reaching the value configured in the period register. Once done, the PWM timer returns to zero and starts increasing again. PWM period is equal to the value of period register + 1.

Note: The period register is PWM_TIMER*x*_PERIOD (*x* = 0, 1, 2), i.e., PWM_TIMER0_PERIOD, PWM_TIMER1_PERIOD, PWM_TIMER2_PERIOD.

- Count-Down Mode:
  The PWM timer decrements to zero, starting from the value configured in the period register. After reaching zero, it is set back to the period value. Then it starts to decrement again. In this case, the PWM period is also equal to the value of period register + 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. This pattern is then repeated. The PWM period is the result of (the value of period register × 2 + 1).

Figures 29.3-6 to 29.3-9 show PWM timer waveforms in different modes, including timer behavior during synchronization events.

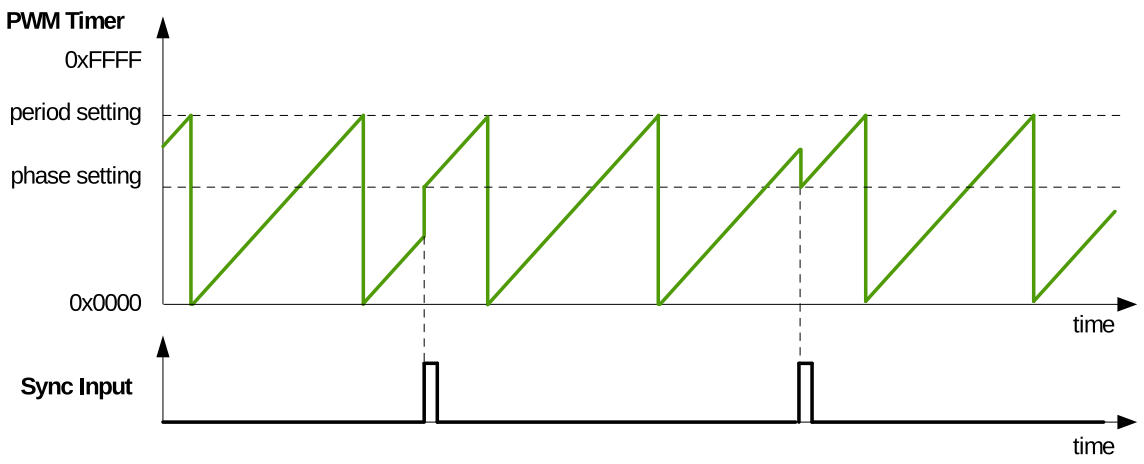



Figure 29.3-6. Count-Up Mode Waveform

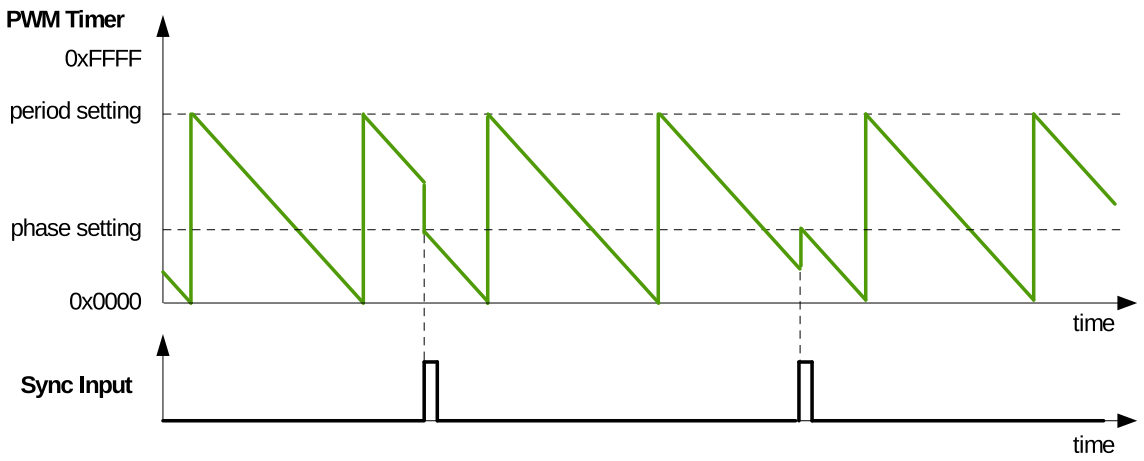



Figure 29.3-7. Count-Down Mode Waveforms

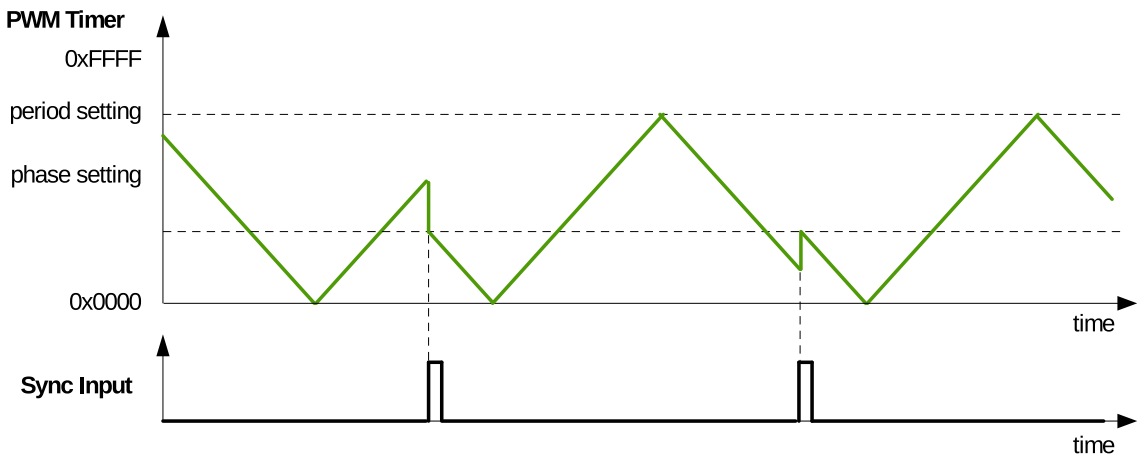



Figure 29.3-8. Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

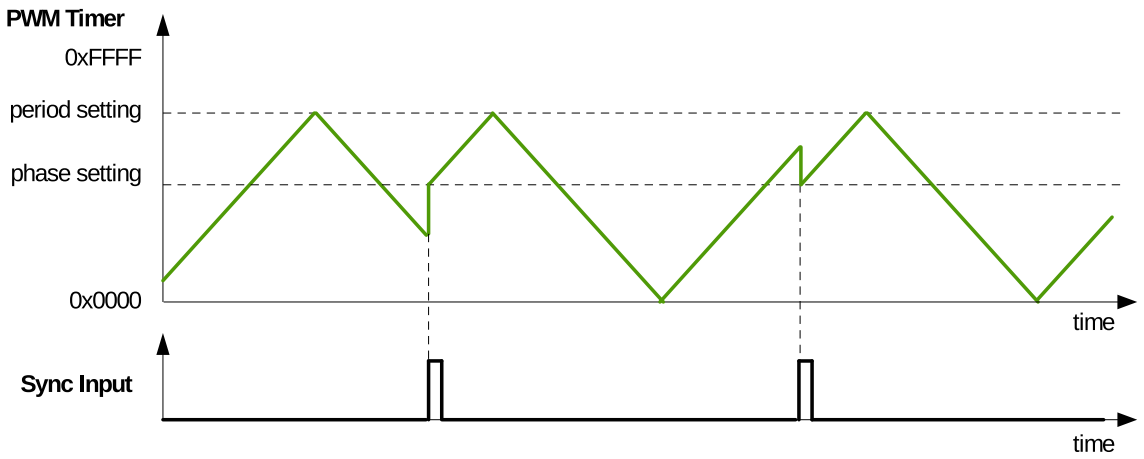



Figure 29.3-9. Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

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 equals to the value of the period register (PWM_TIMER*x*_PERIOD) and when the PWM timer is increasing.
- UTEZ
  The timing event generated when the PWM timer's value equals to zero and when the PWM timer is increasing.
- DTEP
  The timing event generated when the PWM timer's value equals to the value of the period register (PWM_TIMER*x*_PERIOD) and when the PWM timer is decreasing.
- DTEZ
  The timing event generated when the PWM timer's value equals to zero and when the PWM timer is decreasing.

Figures 29.3-10 to 29.3-12 show the timing waveforms of U/DTEP and U/DTEZ.

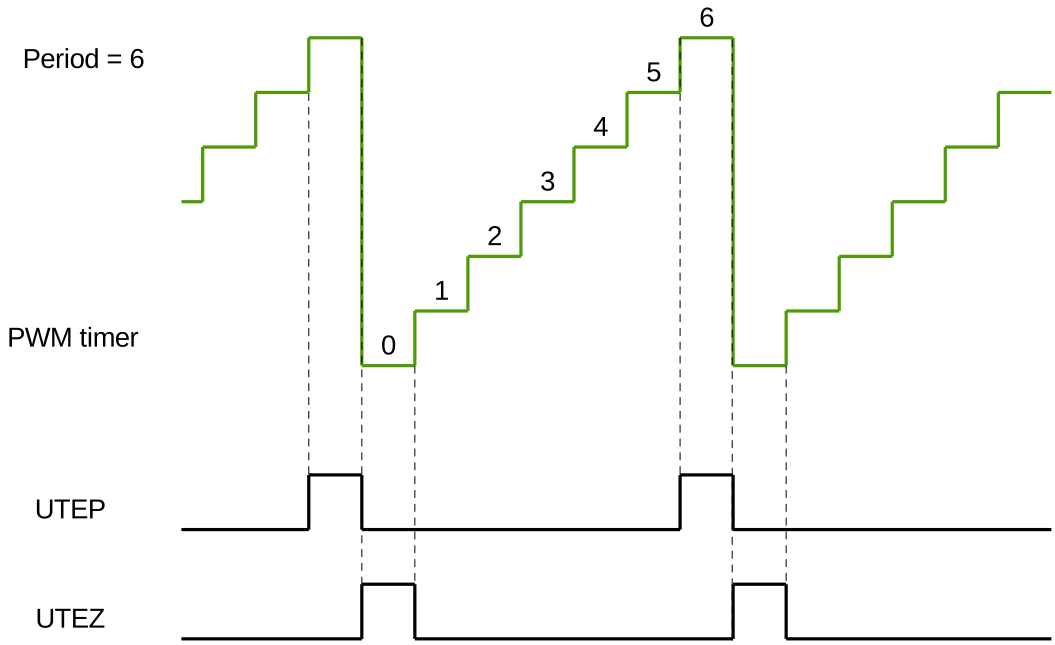



Figure 29.3-10. UTEP and UTEZ Generation in Count-Up Mode

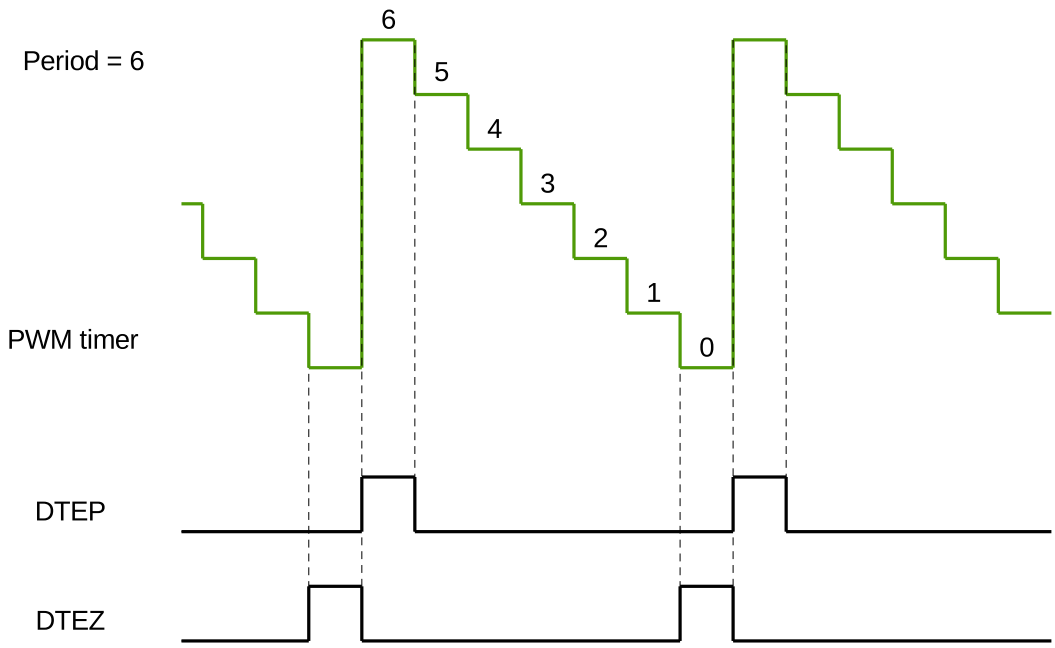



Figure 29.3-11. DTEP and DTEZ Generation in Count-Down Mode

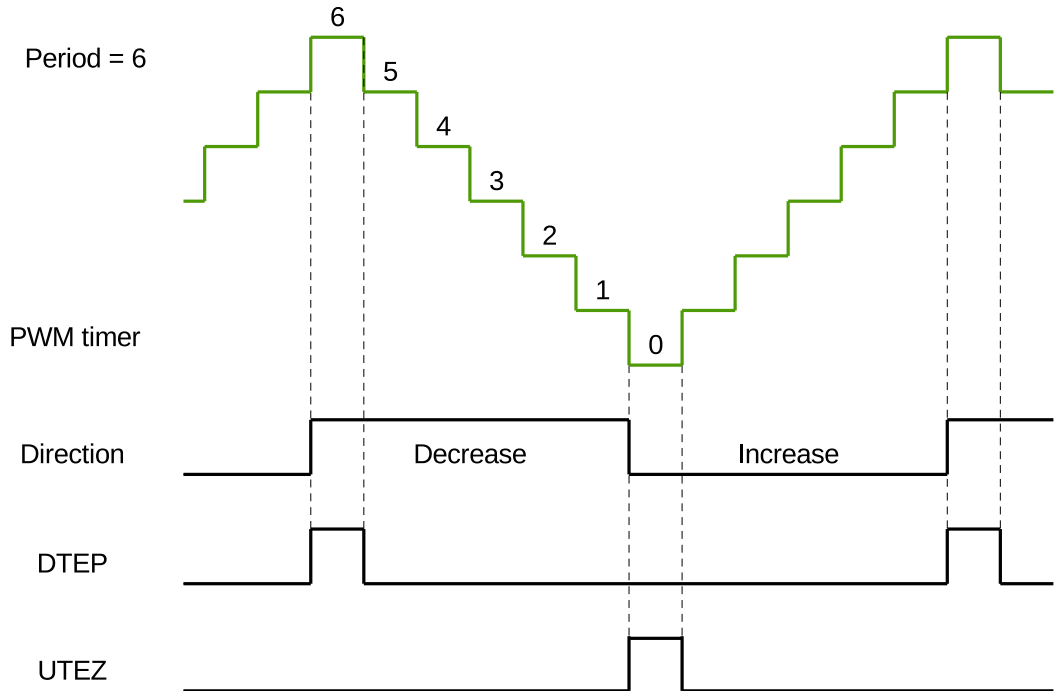



Figure 29.3-12. DTEP and UTEZ Generation in Count-Up-Down Mode

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.

#### 29.3.2.3 PWM Timer Shadow Register

The PWM timer's period register and the PWM timer's clock prescaler register have shadow registers. The purpose of a shadow register is to save a copy of the value to be written into the active register at a specific moment synchronized with the hardware. Both register types are defined as follows:

- Active Register
  This register is directly responsible for controlling all actions performed by hardware.
- Shadow Register
  It acts as a temporary buffer for a value to be written on the active register. Before this happens, the content of the shadow register has no direct effect on the controlled hardware. At a specific, user-configured point in time, the value saved in the shadow register is copied to the active register. This helps to prevent spurious 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 active register is determined by its specific update method register. The update can start when the PWM timer is equal to zero, when the PWM timer is equal to period,at a synchronization moment, or immediately. Software can trigger a globally forced update which will prompt all registers in the module to be updated according to shadow registers.

#### 29.3.2.4 PWM Timer Synchronization and Phase Locking

The PWM modules adopt a flexible synchronization method. Each PWM timer has a synchronization input and a synchronization output. The synchronization input can be selected from three synchronization outputs and three synchronization signals from the GPIO matrix. The synchronization output can be generated from the synchronization input signal, or when the PWM timer's value is equal to period or zero. Thus, the PWM timers can be chained together with their phase locked. During synchronization, the PWM timer clock prescaler will reset its counter in order to synchronize the PWM timer clock.

### 29.3.3 PWM Operator Submodule

The PWM Operator submodule has the following functions:

- Generates a PWM signal pair, based on timing references obtained from the corresponding PWM timer.
- Each signal out of the PWM signal pair includes a specific pattern of dead time.
- Superimposes a carrier on the PWM signal, if configured to do so.
- Handles response under fault conditions.

Figure 29.3-13 shows the block diagram of a PWM operator.

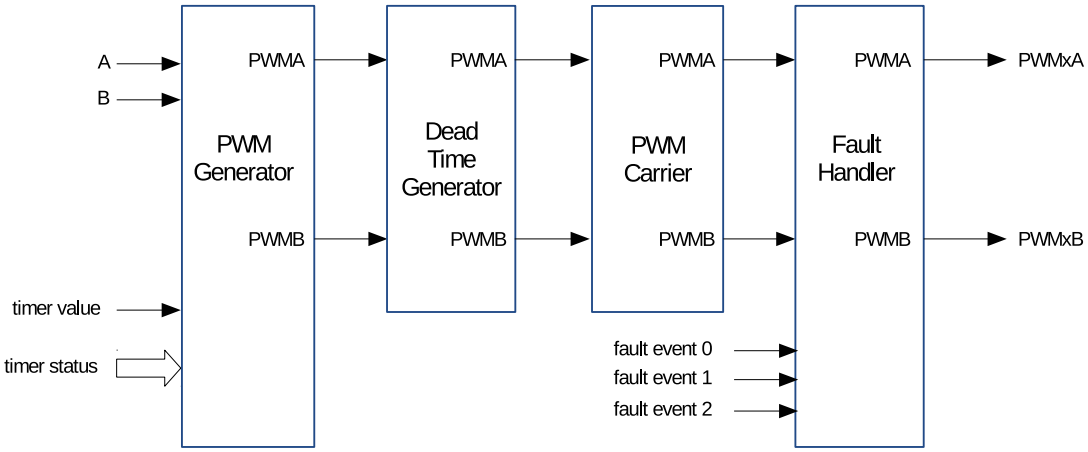



Figure 29.3-13. Submodules Inside the PWM Operator

#### 29.3.3.1 PWM Generator Submodule

**Purpose of the PWM Generator Submodule**

In this submodule, 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 submodule 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 satisfied:
  - 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/DT1, U/DT2 timing events based on fault or synchronization events.
- Management of priority when these timing events occur concurrently.
- Qualification and generation of set, clear and toggle actions, based on the timing events.
- Controlling of the PWM duty cycle, depending on configuration of the PWM generator submodule.
- Handling of new time stamp values, using shadow, registers to prevent glitches in the PWM cycle.

#### PWM Operator Shadow Registers

The time stamp registers A and B, as well as action configuration registers PWM_GENx_A_REG and PWM_GENx_B_REG are shadowed. Shadowing provides a way of updating registers in sync with the hardware. For a description of the shadow registers, please see 29.3.2.3.

#### Timing Events

For convenience, all timing signals and events are summarized in Table 29.3-2.

**Table 29.3-2. 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 A register | |
| DTEB | PWM timer value is equal to B register | |
| 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 A register | |
| UTEB | PWM timer value is equal to B register | |
| 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 PWMxA and PWMxB outputs. The change is done asynchronously. Software-force control is handled by the PWM_PWM_GENx_FORCE_REG registers.

The selection and configuration of T0/T1 in the PWM generator submodule is independent of the configuration of fault events in the fault handler submodule. 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 will always increment after a TEZ event. So when the PWM timer is in count-up-down mode, DTEP and UTEZ events will occur, while the events UTEP and DTEZ will never occur.

The PWM generator can handle multiple events at the same time. Events are prioritized by the hardware and relevant details are provided in Table 29.3-3 and Table 29.3-4. 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 direction.

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 29.3-3. Timing Events Priority When PWM Timer Increments**

| Priority Level | Event |
|---|---|
| 1 (highest) | Software-force event |
| 2 | UTEP |
| 3 | UT0 |
| 4 | UT1 |
| 5 | UTEB |
| 6 | UTEA |
| 7 (lowest) | UTEZ |

**Table 29.3-4. Timing Events Priority when PWM Timer Decrements**

| Priority level | Event |
|---|---|
| 1 (highest) | Software-force 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 29.3-10, 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 29.3-11, 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 submodule controls the behavior of outputs 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 direction (up or down). Knowing the counting direction, the submodule may then perform an independent action at each stage of the PWM timer counting up or down.

The following actions may be configured on outputs PWM*x*A and PWM*x*B:

- 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 high, pull it low, or vice versa.
- Do Nothing:
  Keep both outputs PWM*x*A and PWM*x*B unchanged. In this state, interrupts can still be triggered.

The configuration of actions on outputs is done by using registers PWN_GEN*x*_A_REG and PWN_GEN*x*_B_REG. So, the action to be taken on each output is set independently. Also there is great flexibility in selecting actions to be taken on a given output based on events. More specifically, any event listed in Table 29.3-2 can operate on either output PWM*x*A or PWM*x*B. To check out registers for particular generator 0, 1 or 2, please refer to register description in Section 29.4.

### Waveforms for Common Configurations

Figure 29.3-14 presents the symmetric PWM waveform generated when the PWM timer is counting up and down. 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.

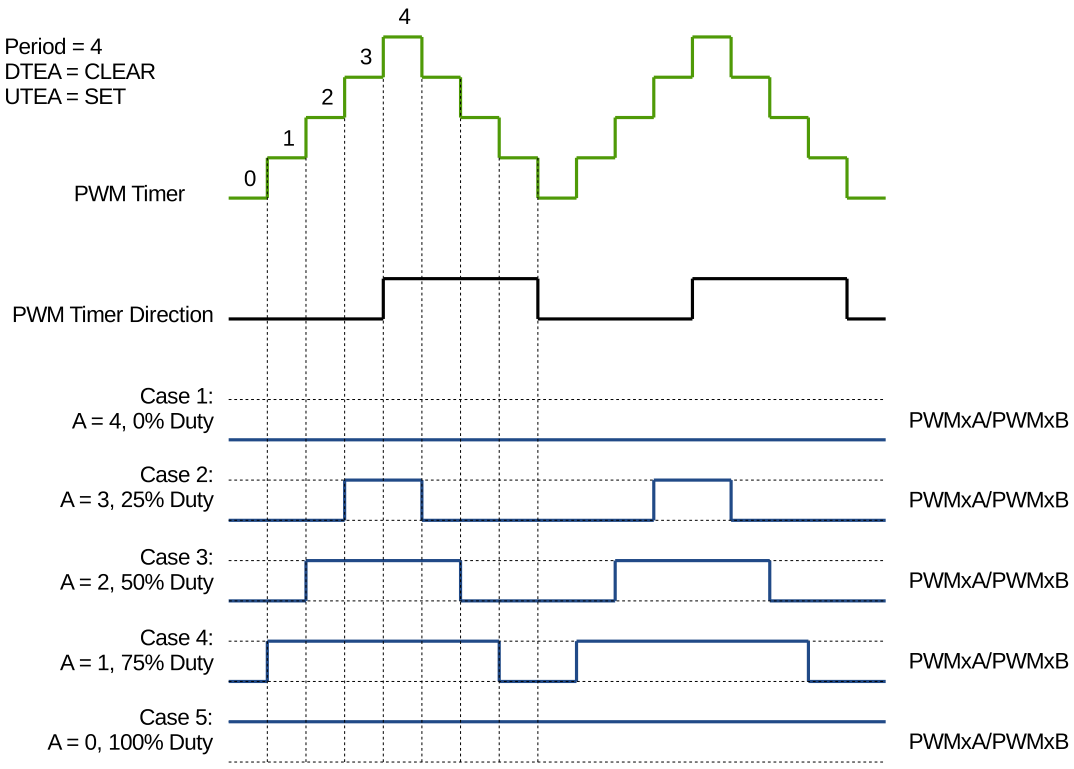



**Figure 29.3-14. Symmetrical Waveform in Count-Up-Down Mode**

The PWM waveforms in Figures 29.3-15 to 29.3-18 show some common PWM operator configurations. The following conventions are used in the figures:

- Period A and B refer to the values written in the corresponding registers.
- PWM*x*A and PWM*x*B are the output signals of PWM Operator *x*.

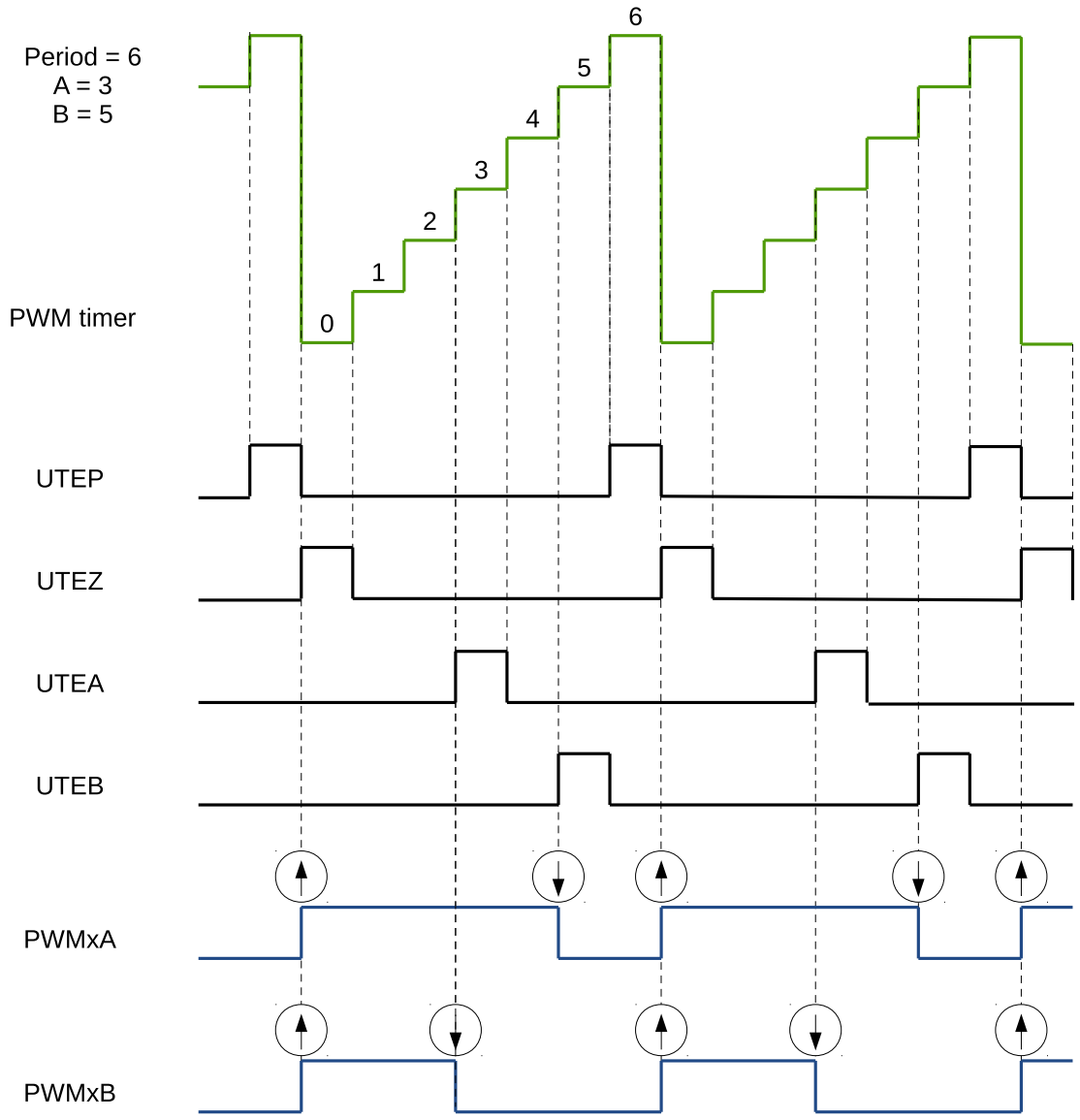



**Figure 29.3-15. Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWMxA and PWMxB — Active High**

The duty modulation for PWMxA is set by B, active high and proportional to B.
The duty modulation for PWMxB is set by A, active high and proportional to A.

$$Period = (PWM_TIMERx_PERIOD + 1) \times T_{PT_clk}$$

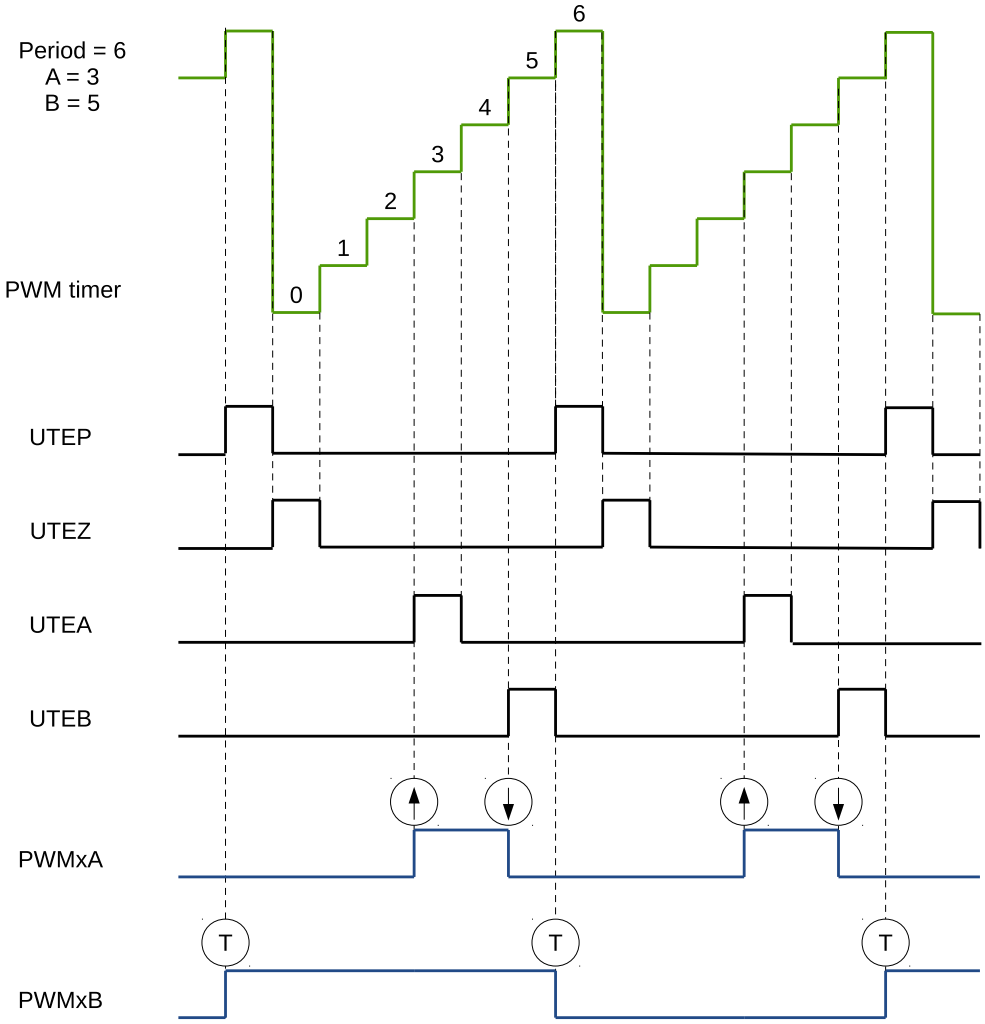



**Figure 29.3-16. Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWMxA**

Pulses may be generated anywhere within the PWM cycle (zero – period).
PWMxA's high time duty is proportional to (B – A).

$$Period = (PWM_TIMERx_PERIOD + 1) \times T_{PT_clk}$$

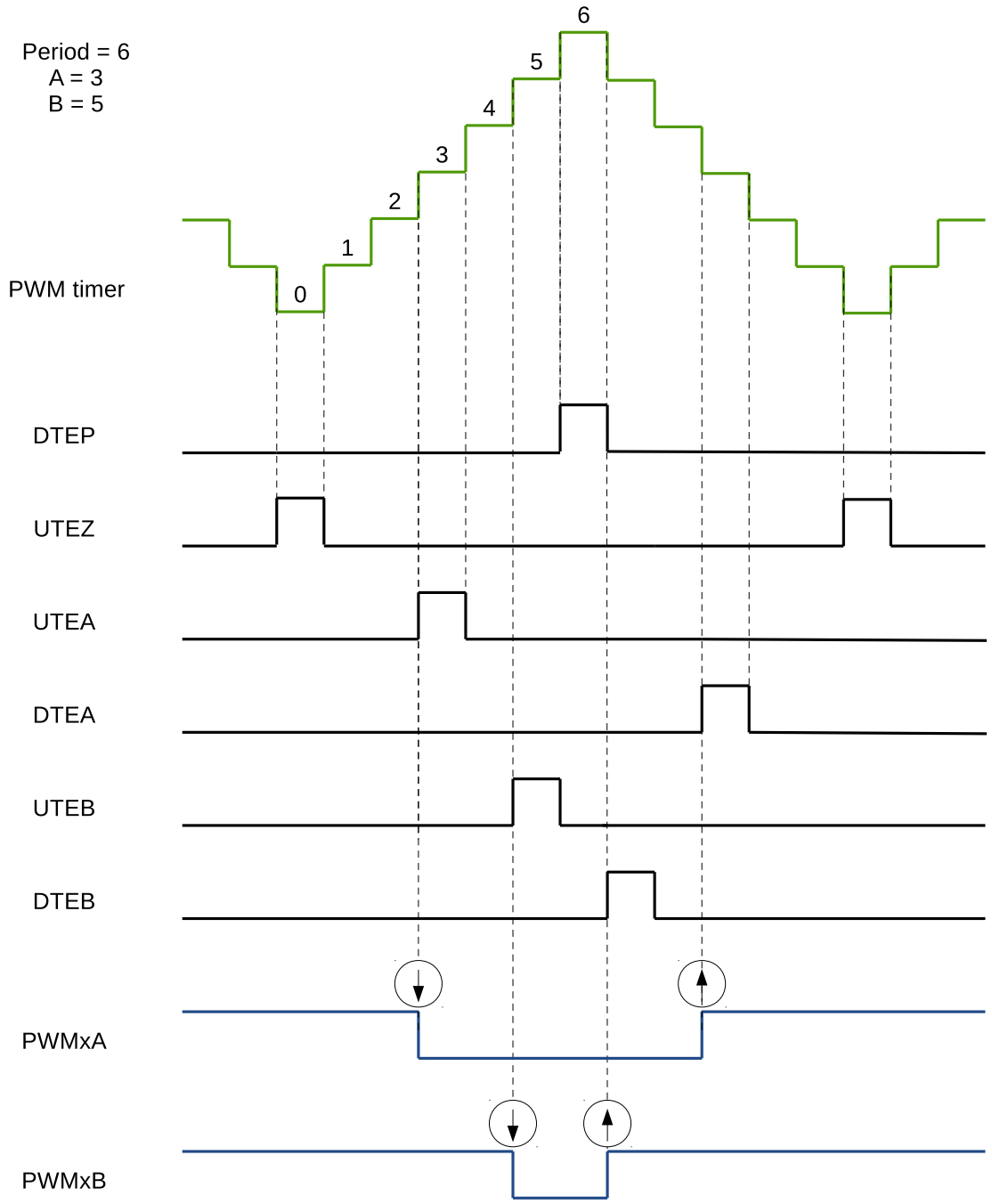



**Figure 29.3-17. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWMxA and PWMxB — Active High**

The duty modulation for PWMxA is set by A, active high and proportional to A.
The duty modulation for PWMxB is set by B, active high and proportional to B.
Outputs PWMxA and PWMxB can drive independent switches.

$$Period = (2 \times PWM_TIMERx_PERIOD + 1) \times T_{PT_clk}$$

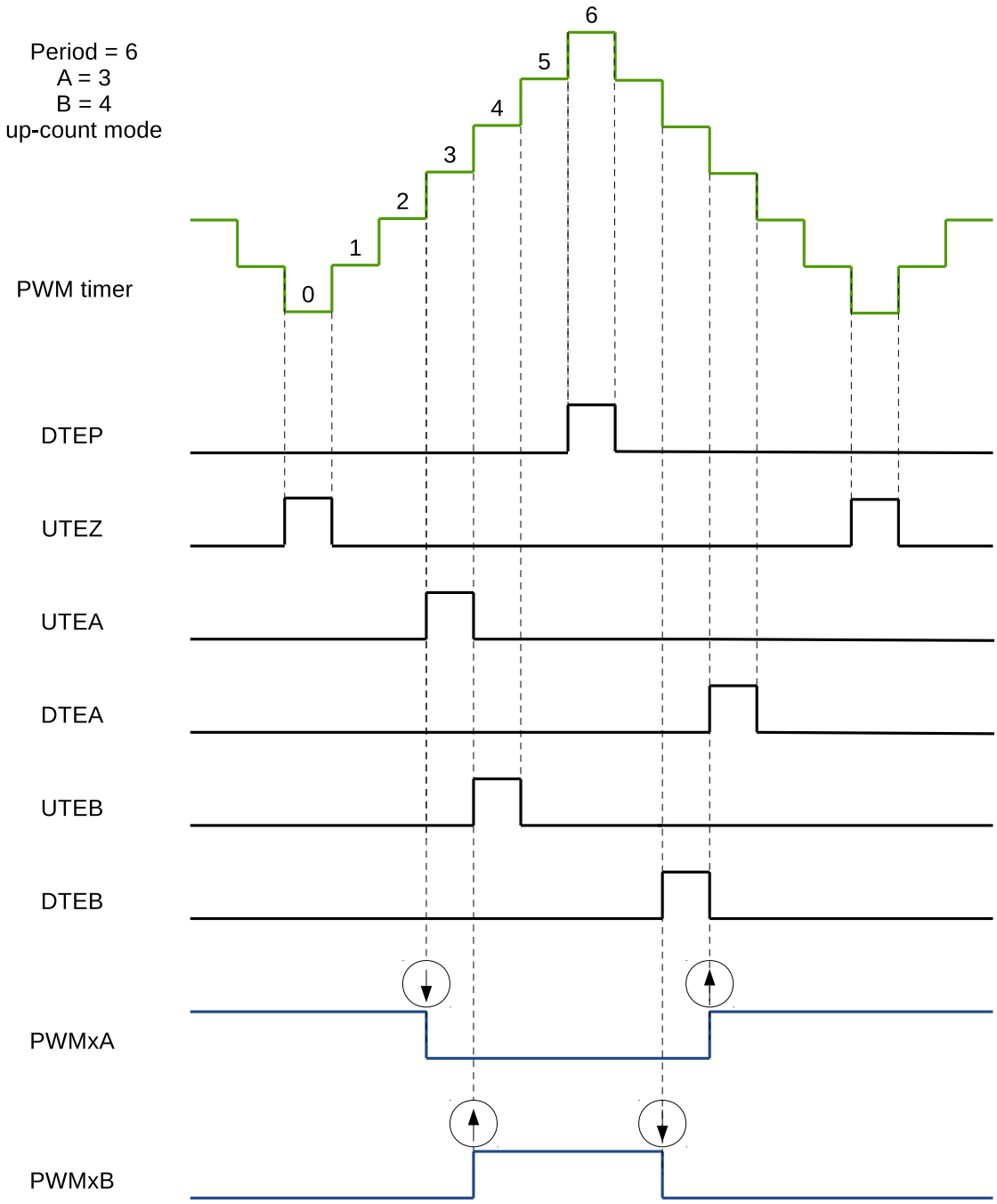



**Figure 29.3-18. Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary**

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 fully programmable by software. Use the dead-time generator module if another edge delay method is required.

$$Period = (2 \times PWM_TIMERx_PERIOD + 1) \times T_{PT_clk}$$

### 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, meaning 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 timing events or immediate events.

Figure 29.3-19 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.

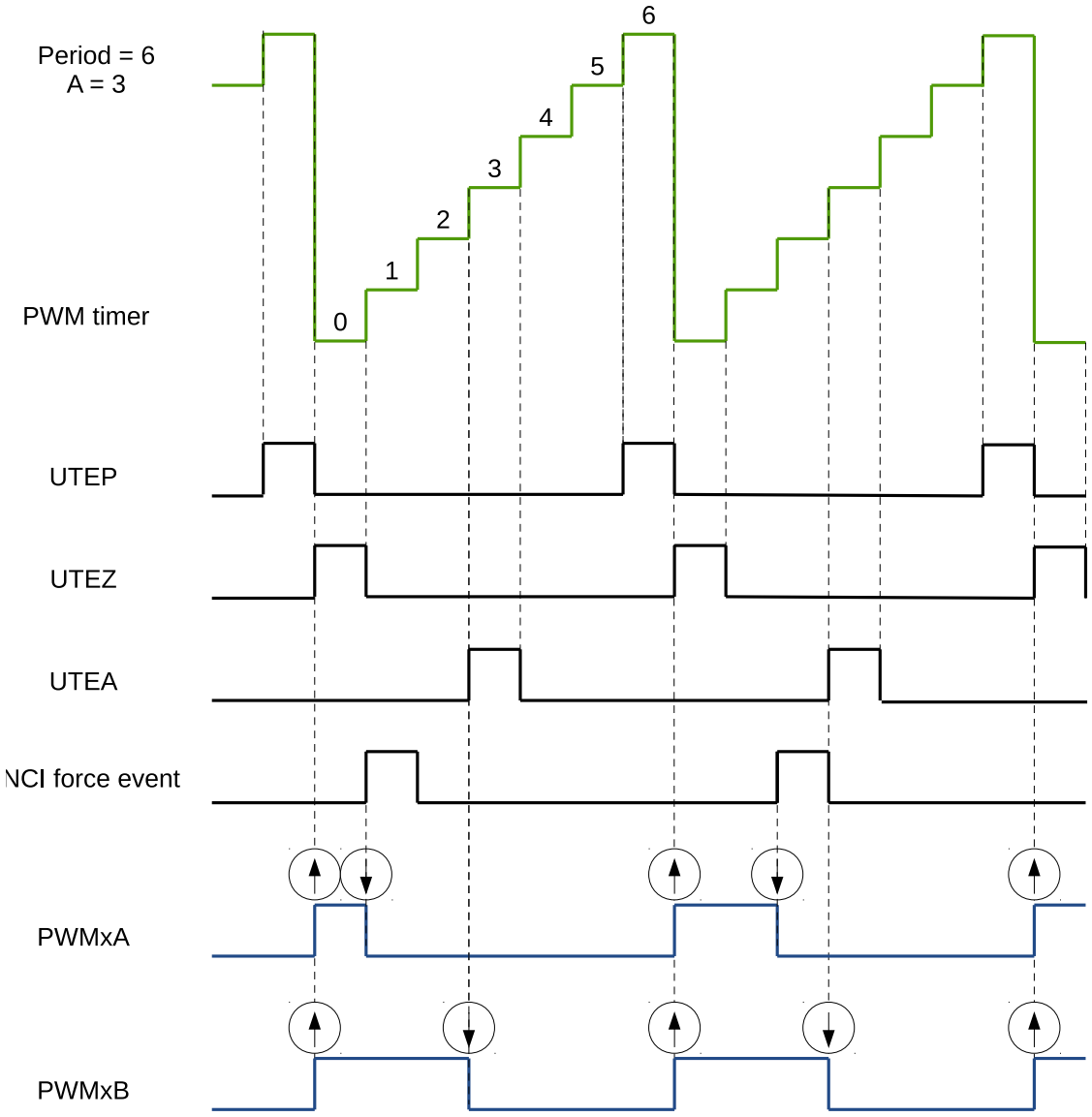



Figure 29.3-19. Example of an NCI Software-Force Event on PWM*x*A

Figure 29.3-20 shows a waveform of CNTU software-force events. UTEZ events are selected as triggers for

CNTU software-force events. CNTU is used to force the PWM*x*B output low. Forcing on PWM*x*A is disabled.

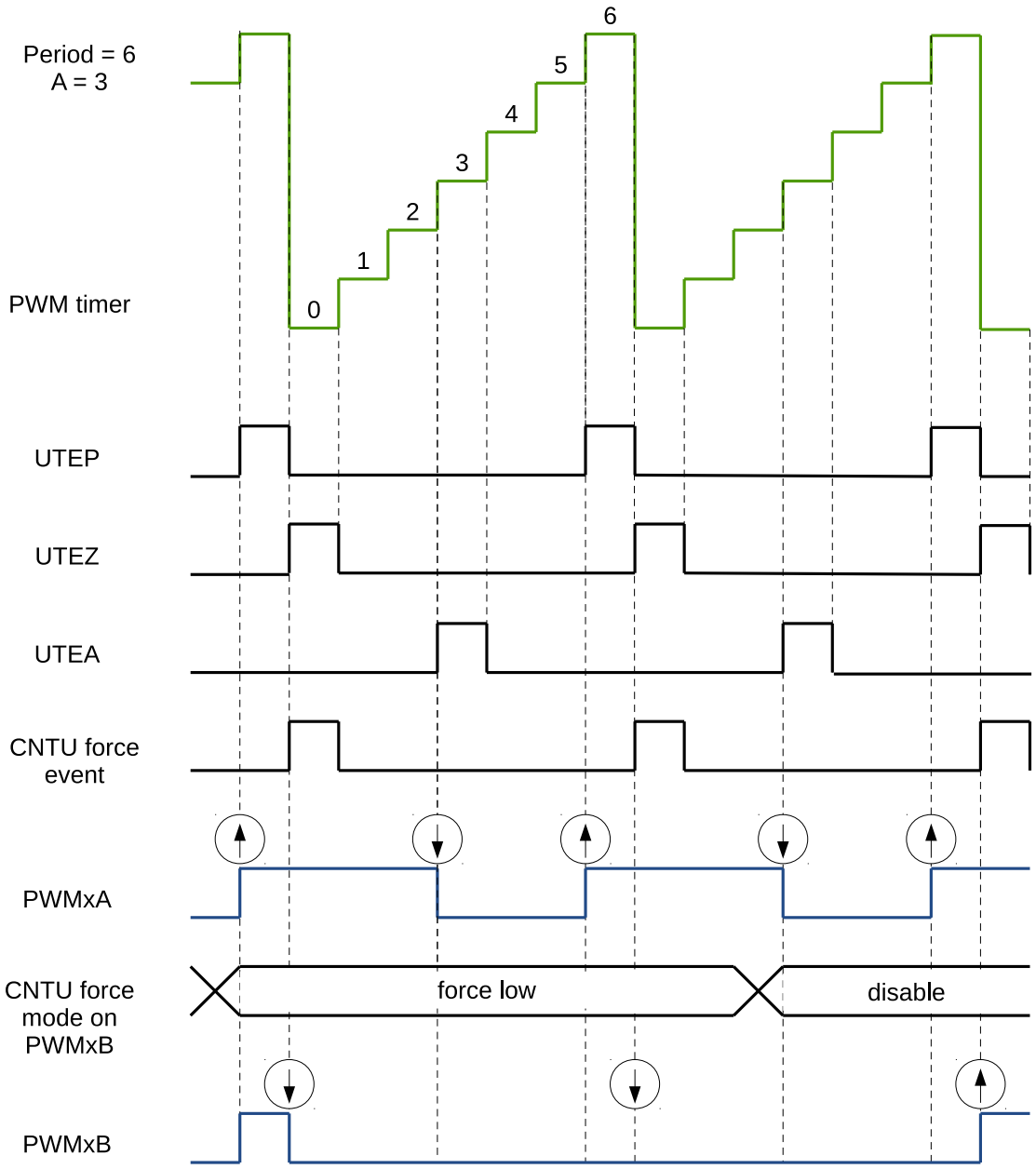



Figure 29.3-20. Example of a CNTU Software-Force Event on PWM*x*B

### 29.3.3.2 Dead Time Generator Submodule

#### Purpose of the Dead Time Generator Submodule

Several options to generate signals on PWM*x*A and PWM*x*B outputs, with a specific placement of signal edges, have been discussed in section 29.3.3.1. The required dead time is obtained by altering the edge placement between signals and by setting the signal's duty cycle. Another option is to control the dead time using a specialized submodule – the Dead Time Generator.

The key functions of the dead time generator submodule are as follows:

- Generating signal pairs (PWM*x*A and PWM*x*B) with a dead time from a single PWM*x*A input
- Creating a dead time by adding delay to signal edges:
  - Rising edge delay (RED)
  - Falling edge delay (FED)
- Configuring the signal pairs to be:
  - Active high complementary (AHC)
  - Active low complementary (ALC)
  - Active high (AH)
  - Active low (AL)
- This submodule may also be bypassed, if the dead time is configured directly in the generator submodule.

### Dead Time Generator's Shadow Registers

Delay registers RED and FED are shadowed with registers PWM_DT*x*_RED_CFG_REG and PWM_DT*x*_FED_CFG_REG. For the description of shadow registers, please see section 29.3.2.3.

### Highlights for Operation of the Dead Time Generator

Options for setting up the dead-time submodule are shown in Figure 29.3-21.

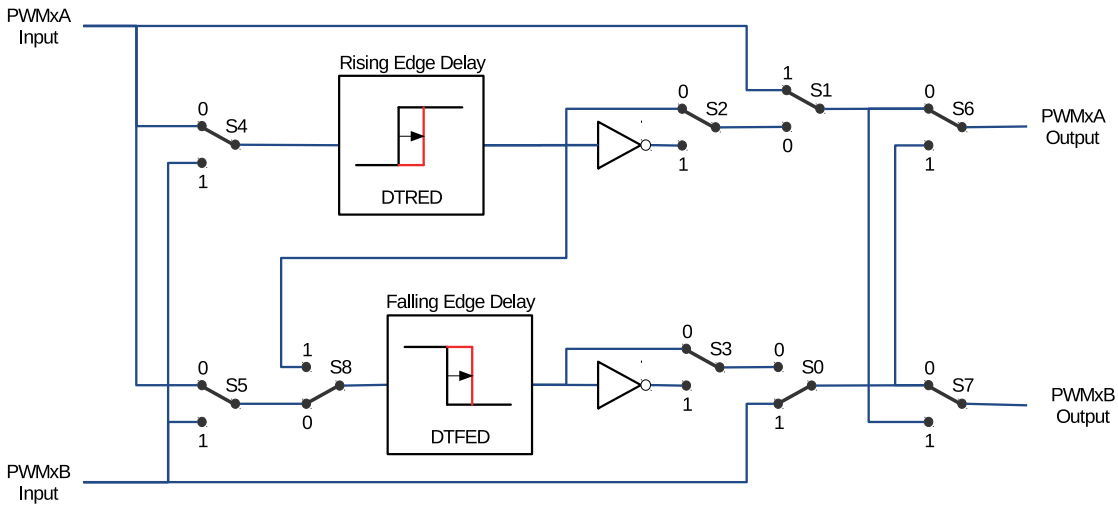



Figure 29.3-21. Options for Setting up the Dead Time Generator Submodule

S0-8 in the figure above are switches controlled by registers PWM_DT*x*_CFG_REG shown in Table 29.3-5.

Table 29.3-5. Dead Time Generator Switches Control Registers

| Switch | Register |
|---|---|
| S0 | PWM_DT*x*_B_OUTBYPASS |

| Switch | Register |
|---|---|
| S1 | PWM_DTx_A_OUTBYPASS |
| S2 | PWM_DTx_RED_OUTINVERT |
| S3 | PWM_DTx_FED_OUTINVERT |
| S4 | PWM_DTx_RED_INSEL |
| S5 | PWM_DTx_FED_INSEL |
| S6 | PWM_DTx_A_OUTSWAP |
| S7 | PWM_DTx_B_OUTSWAP |
| S8 | PWM_DTx_DEB_MODE |

All switch combinations are supported, but not all of them represent the typical modes of use. Table 29.3-6 documents some typical dead time configurations. In these configurations the position of S4 and S5 sets PWMxA as the common source of both falling-edge and rising-edge delay. The modes presented in table 29.3-6 may be categorized as follows:

- **Mode 1: Bypass delays on both falling (FED) as well as raising edge (RED)**
  In this mode the dead time submodule is disabled. Signals PWMxA and PWMxB pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**
  These 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 29.3-22 to 29.3-25.
- **Modes 6 and 7: Bypass delay on falling edge (FED) or rising edge (RED)**
  In these modes, either RED (Rising Edge Delay) or FED (Falling Edge Delay) is bypassed. As a result, the corresponding delay is not applied.

**Table 29.3-6. Typical Dead Time Generator Operating Modes**

| Mode | Mode Description | S0 | S1 | S2 | S3 |
|---|---|---|---|---|---|
| 1 | PWMxA and PWMxB Pass Through/No Delay | 1 | 1 | X | X |
| 2 | Active High Complementary (AHC), see Figure 29.3-22 | 0 | 0 | 0 | 1 |
| 3 | Active Low Complementary (ALC), see Figure 29.3-23 | 0 | 0 | 1 | 0 |
| 4 | Active High (AH), see Figure 29.3-24 | 0 | 0 | 0 | 0 |
| 5 | Active Low (AL), see Figure 29.3-25 | 0 | 0 | 1 | 1 |
| 6 | PWMxA Output = PWMxA In (No Delay)<br>PWMxB Output = PWMxA Input with Falling Edge Delay | 0 | 1 | 0 or 1 | 0 or 1 |
| 7 | PWMxA Output = PWMxA Input with Rising Edge Delay<br>PWMxB Output = PWMxB Input with No Delay | 1 | 0 | 0 or 1 | 0 or 1 |
| Note: For all the modes above, the position of the binary switches S4 to S8 is set to 0. | | | | | |

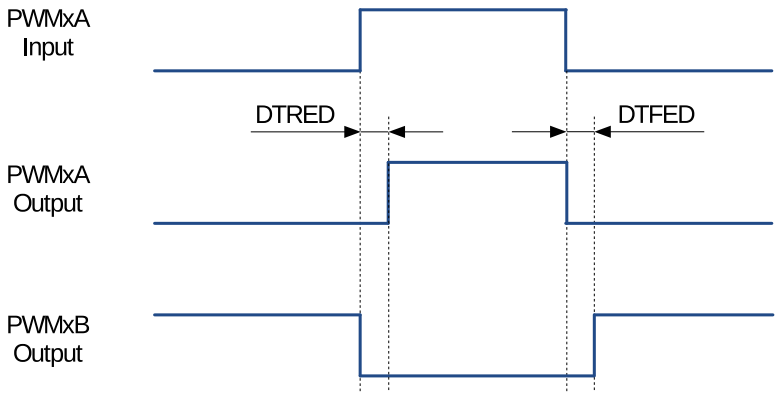



Figure 29.3-22. Active High Complementary (AHC) Dead Time Waveforms

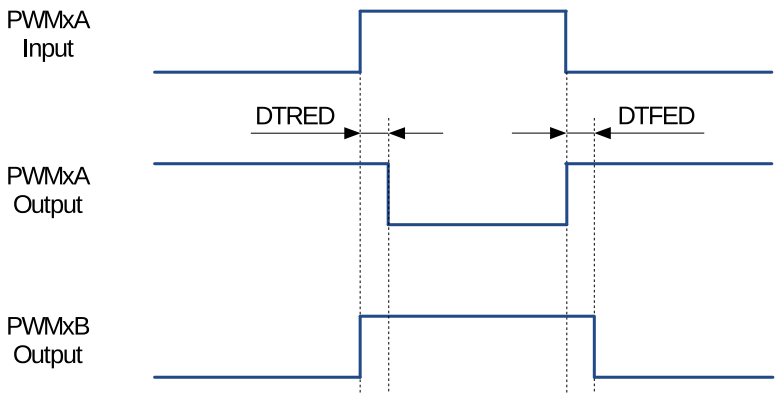



Figure 29.3-23. Active Low Complementary (ALC) Dead Time Waveforms

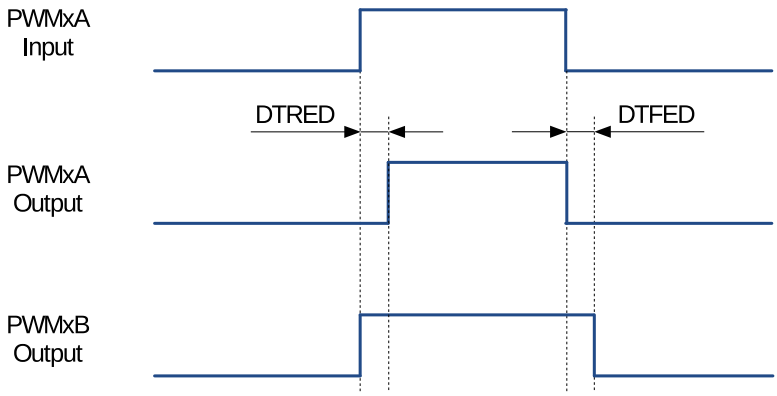



Figure 29.3-24. Active High (AH) Dead Time Waveforms

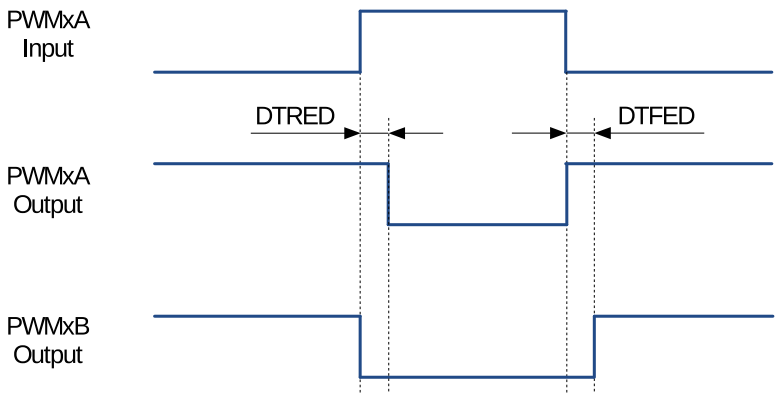



Figure 29.3-25. Active Low (AL) Dead Time Waveforms

Rising edge (RED) and falling edge (FED) delays may be set up independently. The delay value is programmed using the 16-bit registers PWM_DTx_RED and PWM_DTx_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 PWM_DTx_CLK_SEL.

To calculate the delay on falling edge (FED) and rising edge (RED), use the following formulas:

$$FED = PWM_DTx_FED \times T_{DT_clk}$$

$$RED = PWM_DTx_RED \times T_{DT_clk}$$

### 29.3.3.3 PWM Carrier Submodule

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

submodule 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 submodule 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 PWM_CARRIER*x*_PRESCALE and PWM_CARRIER*x*_DUTY bits in the PWM_CARRIER*x*_CFG_REG register. The purpose of one-shot pulses is to provide high-energy impulse to reliably turn on the power switch. Subsequent pulses sustain the power-on status. The width of a one-shot pulse is configurable with the PWM_CARRIER*x*_OSHTWTH bits. Enabling/disabling of the carrier submodule is done with the PWM_CARRIER*x*_EN bit.

### Waveform Examples

Figure 29.3-26 shows an example of waveforms, where a carrier is superimposed on original PWM pulses. This figure do not show the first one-shot pulse and the duty-cycle control. Related details are covered in the following two sections.

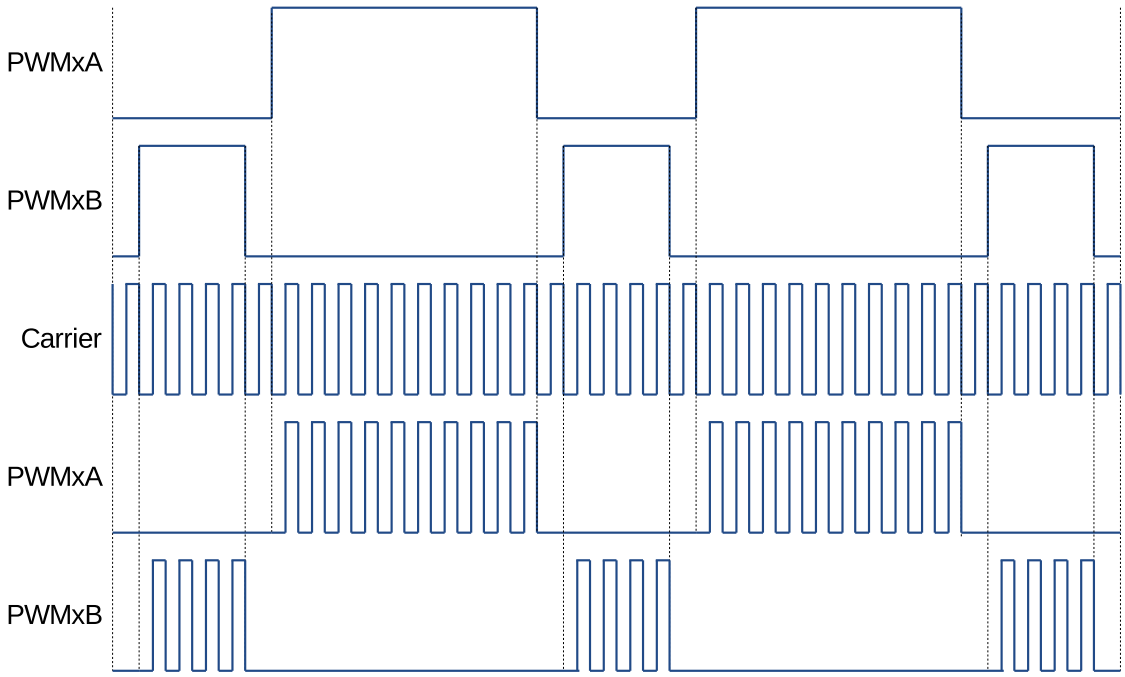



Figure 29.3-26. Example of Waveforms Showing PWM Carrier Action

### One-Shot Pulse

The width of the first pulse is configurable. It may assume one of 16 possible values and is described by the formula below:

$$T_{1stpulse} = T_{PWM_clk} \times 8 \times (PWM_CARRIERx_PRESCALE + 1) \times (PWM_CARRIERx_OSHTWTH + 1)$$

Where:

- $T_{PMW_clk}$ is the period of the PWM clock (PWM_clk).
- $(PWM_CARRIERx_OSHTWTH + 1)$ is the width of the first pulse (whose value ranges from 1 to 16).
- $(PWM_CARRIERx_PRESCALE + 1)$ is the PWM carrier clock's (PC_clk) prescaler value.

The first one-shot pulse and subsequent sustaining pulses are shown in Figure 29.3-27.

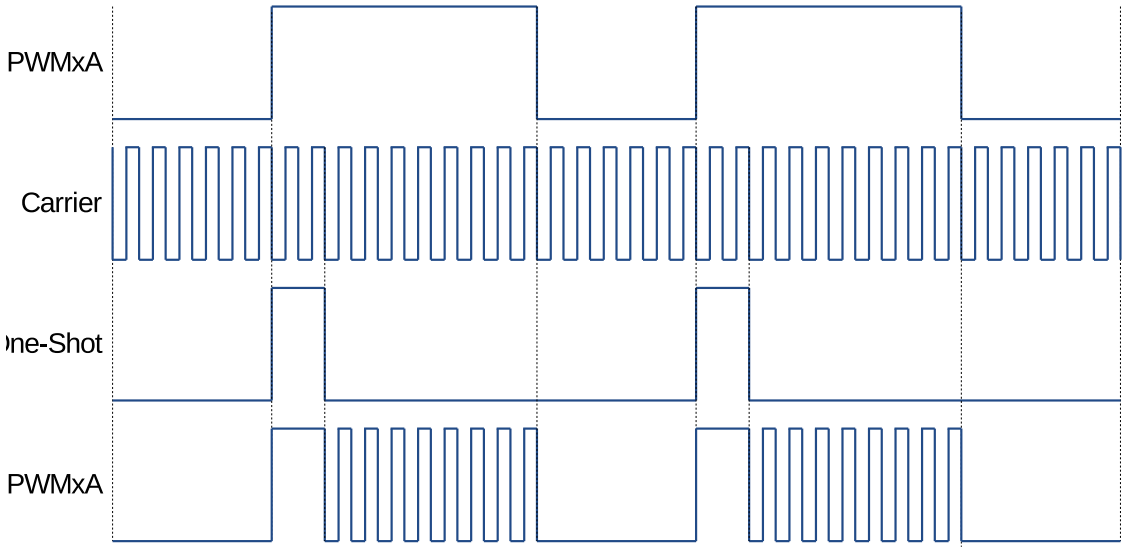



**Figure 29.3-27. Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Sub-module**

### 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 PWM_CARRIERx_DUTY, or bits [7:5] of register PWM_CARRIERx_CFG_REG.

Below is the formula for calculating the duty cycle:

$$Duty = PWM_CARRIERx_DUTY \div 8$$

All seven settings of the duty cycle are shown in Figure 29.3-28.

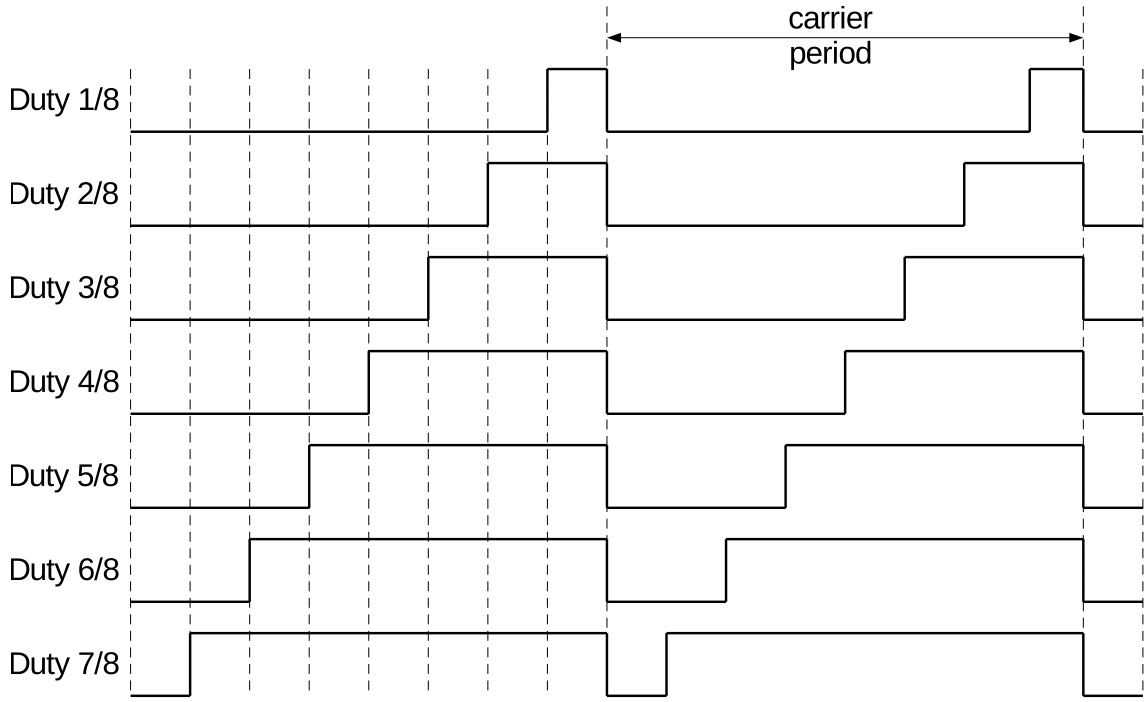



Figure 29.3-28. Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

### 29.3.3.4 Fault Handler Submodule

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 submodule to generate fault events. Fault events can then execute the user code to control MCPWM outputs in response to specific faults.

#### Function of Fault Handler Submodule

The key actions performed by the fault handler submodule 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 tripping (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 submodule function as required.

#### Operation and Configuration Tips

This section provides the operational tips and set-up options for the fault handler submodule.

Fault signals coming from pads 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 submodule 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 pads 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 handler submodule 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 registers PWM_FH*x*_A_CBC_U/D and PWM_FH*x*_B_CBC_U/D. Different actions can be indicted when the PWM timer is incrementing or decrementing. Different CBC action interrupts can be triggered for different fault events. Status register PWM_FH*x*_CBC_ON indicates whether a CBC action is on or off. When the fault event is no longer present, CBC actions on PWM*x*A/B will be cleared at a specified point, which is either a D/UTEP or D/UTEZ event. Register PWM_FH*x*_CBCPULSE determines at which event PWM*x*A and PWM*x*B will be able to resume normal actions. Therefore, in this mode, the CBC action is cleared or refreshed upon every PWM cycle.

- **One-Shot (OST) action:**
  When OST action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately, depending on the setting of registers PWM_FH*x*_A_OST_U/D and PWM_FH*x*_B_OST_U/D. Different actions can be configured when PWM timer is incrementing or decrementing. Different OST action interrupts can be triggered form different fault events. Status register PWM_FH*x*_OST_ON indicates whether an OST action is on or off. The OST actions on PWM*x*A/B are not automatically cleared when the fault event is no longer present. One-shot actions must be cleared manually by negating the value stored in register PWM_FH*x*_CLR_OST.

## 29.3.4 Capture Submodule

### 29.3.4.1 Introduction

The capture submodule 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 PAD input. Multiple capture channels can be sourced from the same PAD input, while prescaling for each channel can be set differently. Also, capture channels are sourced from different PADs. This provides several options for handling capture signals by hardware in the background, instead of having them processed directly by the CPU. A capture submodule has the following independent key resources:

- One 32-bit timer (counter) which can be synchronized with the PWM timer, another submodule 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.

#### 29.3.4.2 Capture Timer

The capture timer is a 32-bit counter incrementing continuously, once enabled. On the input it has an APB clock running typically at 80 MHz. At a sync event the counter is loaded with phase stored in register PWM_CAP_TIMER_PHASE_REG. Sync events can come from PWM timers sync-out, PWM module sync-in or software. The capture timer provides timing references for all three capture channels.

#### 29.3.4.3 Capture Channel

The capture signal coming to a capture channel will be inverted first, if needed, and then prescaled. Finally, specified edges of preprocessed capture signal will trigger capture events. When a capture event occurs, the capture timer's value is stored in time-stamp register PWM_CAP_CH*x*_REG. Different interrupts can be generated for different capture channels at capture events. The edge that triggers a capture event is recorded in register PWM_CAP*x*_EDGE. The capture event can be also forced by software.

## 29.4 Register Summary

The addresses in this section are relative to the MCPWM base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | PWM0 | PWM1 | Acc |
|---|---|---|---|---|
| **Prescaler configuration** | | | | |
| PWM_CLK_CFG_REG | Configuration of the prescaler | 0x3FF5E000 | 0x3FF6C000 | R/W |
| **PWM Timer 0 Configuration and status** | | | | |
| PWM_TIMER0_CFG0_REG | Timer period and update method | 0x3FF5E004 | 0x3FF6C004 | R/W |
| PWM_TIMER0_CFG1_REG | Working mode and start/stop control | 0x3FF5E008 | 0x3FF6C008 | R/W |
| PWM_TIMER0_SYNC_REG | Synchronization settings | 0x3FF5E00C | 0x3FF6C00C | R/W |
| PWM_TIMER0_STATUS_REG | Timer status | 0x3FF5E010 | 0x3FF6C010 | RO |
| **PWM Timer 1 Configuration and Status** | | | | |
| PWM_TIMER1_CFG0_REG | Timer update method and period | 0x3FF5E014 | 0x3FF6C014 | R/W |
| PWM_TIMER1_CFG1_REG | Working mode and start/stop control | 0x3FF5E018 | 0x3FF6C018 | R/W |
| PWM_TIMER1_SYNC_REG | Synchronization settings | 0x3FF5E01C | 0x3FF6C01C | R/W |
| PWM_TIMER1_STATUS_REG | Timer status | 0x3FF5E020 | 0x3FF6C020 | RO |
| **PWM Timer 2 Configuration and status** | | | | |
| PWM_TIMER2_CFG0_REG | Timer update method and period | 0x3FF5E024 | 0x3FF6C024 | R/W |
| PWM_TIMER2_CFG1_REG | Working mode and start/stop control | 0x3FF5E028 | 0x3FF6C028 | R/W |
| PWM_TIMER2_SYNC_REG | Synchronization settings | 0x3FF5E02C | 0x3FF6C02C | R/W |
| PWM_TIMER2_STATUS_REG | Timer status | 0x3FF5E030 | 0x3FF6C030 | RO |
| **Common configuration for PWM timers** | | | | |

| Name | Description | PWM0 | PWM1 | Acc |
|---|---|---|---|---|
| PWM_TIMER_SYNCI_CFG_REG | Synchronization input selection for timers | 0x3FF5E034 | 0x3FF6C034 | R/W |
| PWM_OPERATOR_TIMERSEL_REG | Select specific timer for PWM operators | 0x3FF5E038 | 0x3FF6C038 | R/W |
| **PWM Operator 0 Configuration and Status** | | | | |
| PWM_GEN0_STMP_CFG_REG | Transfer status and update method for time stamp registers A and B | 0x3FF5E03C | 0x3FF6C03C | R/W |
| PWM_GEN0_TSTMP_A_REG | Shadow register for register A | 0x3FF5E040 | 0x3FF6C040 | R/W |
| PWM_GEN0_TSTMP_B_REG | Shadow register for register B | 0x3FF5E044 | 0x3FF6C044 | R/W |
| PWM_GEN0_CFG0_REG | Fault event T0 and T1 handling | 0x3FF5E048 | 0x3FF6C048 | R/W |
| PWM_GEN0_FORCE_REG | Permissives to force PWM0A and PWM0B outputs by software | 0x3FF5E04C | 0x3FF6C04C | R/W |
| PWM_GEN0_A_REG | Actions triggered by events on PWM0A | 0x3FF5E050 | 0x3FF6C050 | R/W |
| PWM_GEN0_B_REG | Actions triggered by events on PWM0B | 0x3FF5E054 | 0x3FF6C054 | R/W |
| PWM_DT0_CFG_REG | Dead time type selection and configuration | 0x3FF5E058 | 0x3FF6C058 | R/W |
| PWM_DT0_FED_CFG_REG | Shadow register for falling edge delay (FED) | 0x3FF5E05C | 0x3FF6C05C | R/W |
| PWM_DT0_RED_CFG_REG | Shadow register for rising edge delay (RED) | 0x3FF5E060 | 0x3FF6C060 | R/W |
| PWM_CARRIER0_CFG_REG | Carrier enable and configuration | 0x3FF5E064 | 0x3FF6C064 | R/W |
| PWM_FH0_CFG0_REG | Actions on PWM0A and PWM0B on trip events | 0x3FF5E068 | 0x3FF6C068 | R/W |
| PWM_FH0_CFG1_REG | Software triggers for fault handler actions | 0x3FF5E06C | 0x3FF6C06C | R/W |
| PWM_FH0_STATUS_REG | Status of fault events | 0x3FF5E070 | 0x3FF6C070 | RO |
| **PWM Operator 1 Configuration and Status** | | | | |
| PWM_GEN1_STMP_CFG_REG | Transfer status and update method for time stamp registers A and B | 0x3FF5E074 | 0x3FF6C074 | R/W |
| PWM_GEN1_TSTMP_A_REG | Shadow register for register A | 0x3FF5E078 | 0x3FF6C078 | R/W |
| PWM_GEN1_TSTMP_B_REG | Shadow register for register B | 0x3FF5E07C | 0x3FF6C07C | R/W |
| PWM_GEN1_CFG0_REG | Fault event T0 and T1 handling | 0x3FF5E080 | 0x3FF6C080 | R/W |
| PWM_GEN1_FORCE_REG | Permissives to force PWM1A and PWM1B outputs by software | 0x3FF5E084 | 0x3FF6C084 | R/W |
| PWM_GEN1_A_REG | Actions triggered by events on PWM1A | 0x3FF5E088 | 0x3FF6C088 | R/W |
| PWM_GEN1_B_REG | Actions triggered by events on PWM1B | 0x3FF5E08C | 0x3FF6C08C | R/W |
| PWM_DT1_CFG_REG | Dead time type selection and configuration | 0x3FF5E090 | 0x3FF6C090 | R/W |
| PWM_DT1_FED_CFG_REG | Shadow register for FED | 0x3FF5E094 | 0x3FF6C094 | R/W |
| PWM_DT1_RED_CFG_REG | Shadow register for RED | 0x3FF5E098 | 0x3FF6C098 | R/W |

| Name | Description | PWM0 | PWM1 | Acc |
|---|---|---|---|---|
| PWM_CARRIER1_CFG_REG | Carrier enable and configuration | 0x3FF5E09C | 0x3FF6C09C | R/W |
| PWM_FH1_CFG0_REG | Actions on PWM1A and PWM1B on fault events | 0x3FF5E0A0 | 0x3FF6C0A0 | R/W |
| PWM_FH1_CFG1_REG | Software triggers for fault handler actions | 0x3FF5E0A4 | 0x3FF6C0A4 | R/W |
| PWM_FH1_STATUS_REG | Status of fault events | 0x3FF5E0A8 | 0x3FF6C0A8 | RO |
| **PWM Operator 2 Configuration and Status** | | | | |
| PWM_GEN2_STMP_CFG_REG | Transfer status and updating method for time stamp registers A and B | 0x3FF5E0AC | 0x3FF6C0AC | R/W |
| PWM_GEN2_TSTMP_A_REG | Shadow register for register A | 0x3FF5E0B0 | 0x3FF6C0B0 | R/W |
| PWM_GEN2_TSTMP_B_REG | Shadow register for register B | 0x3FF5E0B4 | 0x3FF6C0B4 | R/W |
| PWM_GEN2_CFG0_REG | Fault event T0 and T1 handling | 0x3FF5E080 | 0x3FF6C080 | R/W |
| PWM_GEN2_FORCE_REG | Permissives to force PWM2A and PWM2B outputs by software | 0x3FF5E0BC | 0x3FF6C0BC | R/W |
| PWM_GEN2_A_REG | Actions triggered by events on PWM2A | 0x3FF5E0C0 | 0x3FF6C0C0 | R/W |
| PWM_GEN2_B_REG | Actions triggered by events on PWM2B | 0x3FF5E0C4 | 0x3FF6C0C4 | R/W |
| PWM_DT2_CFG_REG | Dead time type selection and configuration | 0x3FF5E0C8 | 0x3FF6C0C8 | R/W |
| PWM_DT2_FED_CFG_REG | Shadow register for FED | 0x3FF5E0CC | 0x3FF6C0CC | R/W |
| PWM_DT2_RED_CFG_REG | Shadow register for RED | 0x3FF5E0D0 | 0x3FF6C0D0 | R/W |
| PWM_CARRIER2_CFG_REG | Carrier enable and configuration | 0x3FF5E0D4 | 0x3FF6C0D4 | R/W |
| PWM_FH2_CFG0_REG | Actions at PWM2A and PWM2B on trip events | 0x3FF5E0D8 | 0x3FF6C0D8 | R/W |
| PWM_FH2_CFG1_REG | Software triggers for fault handler actions | 0x3FF5E0DC | 0x3FF6C0DC | R/W |
| PWM_FH2_STATUS_REG | Status of fault events | 0x3FF5E0E0 | 0x3FF6C0E0 | RO |
| **Fault Detection Configuration and Status** | | | | |
| PWM_FAULT_DETECT_REG | Fault detection configuration and status | 0x3FF5E0E4 | 0x3FF6C0E4 | R/W |
| **Capture Configuration and Status** | | | | |
| PWM_CAP_TIMER_CFG_REG | Configure capture timer | 0x3FF5E0E8 | 0x3FF6C0E8 | R/W |
| PWM_CAP_TIMER_PHASE_REG | Phase for capture timer sync | 0x3FF5E0EC | 0x3FF6C0EC | R/W |
| PWM_CAP_CH0_CFG_REG | Capture channel 0 configuration and enable | 0x3FF5E0F0 | 0x3FF6C0F0 | R/W |
| PWM_CAP_CH1_CFG_REG | Capture channel 1 configuration and enable | 0x3FF5E0F4 | 0x3FF6C0F4 | R/W |
| PWM_CAP_CH2_CFG_REG | Capture channel 2 configuration and enable | 0x3FF5E0F8 | 0x3FF6C0F8 | R/W |
| PWM_CAP_CH0_REG | Value of last capture on channel 0 | 0x3FF5E0FC | 0x3FF6C0FC | RO |
| PWM_CAP_CH1_REG | Value of last capture on channel 1 | 0x3FF5E100 | 0x3FF6C100 | RO |
| PWM_CAP_CH2_REG | Value of last capture on channel 2 | 0x3FF5E104 | 0x3FF6C104 | RO |
| PWM_CAP_STATUS_REG | Edge of last capture trigger | 0x3FF5E108 | 0x3FF6C108 | RO |

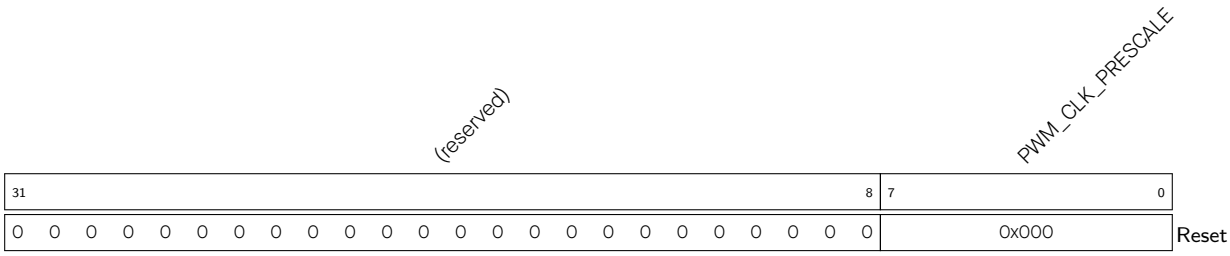
| Name | Description | PWM0 | PWM1 | Acc |
|---|---|---|---|---|
| **Enable update of active registers** | | | | |
| PWM_UPDATE_CFG_REG | Enable update | 0x3FF5E10C | 0x3FF6C10C | R/W |
| **Manage Interrupts** | | | | |
| INT_ENA_PWM_REG | Interrupt enable bits | 0x3FF5E110 | 0x3FF6C110 | R/W |
| INT_RAW_PWM_REG | Raw interrupt status | 0x3FF5E114 | 0x3FF6C114 | RO |
| INT_ST_PWM_REG | Masked interrupt status | 0x3FF5E118 | 0x3FF6C118 | RO |
| INT_CLR_PWM_REG | Interrupt clear bits | 0x3FF5E11C | 0x3FF6C11C | WO |

## 29.5 Registers

The addresses in this section are relative to the MCPWM base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 29.4 *Register Summary*.

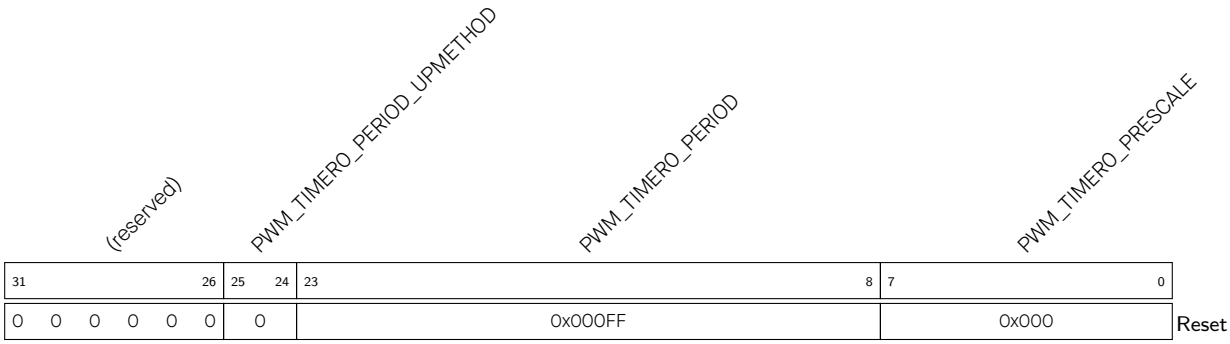
For how to program reserved fields, please refer to Section *Programming Reserved Register Field*.

**Register 29.1. PWM_CLK_CFG_REG (0x0000)**




**PWM_CLK_PRESCALE** Period of PWM_clk = 6.25ns * (PWM_CLK_PRESCALE + 1). (R/W)

**Register 29.2. PWM_TIMER0_CFG0_REG (0x0004)**




**PWM_TIMER0_PERIOD_UPMETHOD** Updating method for active register of PWM timer0 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. TEZ here and below means that the event that happens when the timer equals to zero. (R/W)

**PWM_TIMER0_PERIOD** Period shadow register of PWM timer0. (R/W)

**PWM_TIMER0_PRESCALE** Period of PT0_clk = Period of PWM_clk * (PWM_TIMER0_PRESCALE + 1). (R/W)

**Register 29.3. PWM_TIMER0_CFG1_REG (0x0008)**

| (reserved) | PWM_TIMER0_MOD | PWM_TIMER0_START | |
|---|---|---|---|
| 31 … 5 | 4 … 3 | 2 … 0 | |
| 0 0 0 0 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 | 0x0 | Reset |

**PWM_TIMER0_MOD** PWM timer0 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

**PWM_TIMER0_START** PWM timer0 start and stop control. 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. TEP here and below means the event that happens when the timer equals to period. (R/W)

**Register 29.4. PWM_TIMER0_SYNC_REG (0x000c)**

| (reserved) | PWM_TIMER0_PHASE_DIRECTION | PWM_TIMER0_PHASE | PWM_TIMER0_SYNCO_SEL | PWM_TIMER0_SYNC_SW | PWM_TIMER0_SYNCI_EN | |
|---|---|---|---|---|---|---|
| 31 … 21 | 20 | 19 … 4 | 3 … 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_TIMER0_PHASE_DIRECTION** 0: increase; 1: decrease. (R/W)

**PWM_TIMER0_PHASE** Phase for timer reload at sync event. (R/W)

**PWM_TIMER1_SYNCO_SEL** PWM timer0 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwise: sync_out is always 0. (R/W)

**PWM_TIMER1_SYNC_SW** Toggling this bit will trigger a software sync. (R/W)

**PWM_TIMER1_SYNCI_EN** When set, timer reloading with phase on sync input event is enabled. (R/W)

Register 29.5. PWM_TIMER0_STATUS_REG (0x0010)

| (reserved) | PWM_TIMER0_DIRECTION | PWM_TIMER0_VALUE | |
|---|---|---|---|
| 31 – 17 | 16 | 15 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | Reset |

**PWM_TIMER0_DIRECTION** Current direction of the PWM timer0 counter. 0: increment, 1: decrement. (RO)

**PWM_TIMER0_VALUE** Current value of the PWM timer0 counter. (RO)

Register 29.6. PWM_TIMER1_CFG0_REG (0x0014)

| (reserved) | PWM_TIMER1_PERIOD_UPMETHOD | PWM_TIMER1_PERIOD | PWM_TIMER1_PRESCALE | |
|---|---|---|---|---|
| 31 – 26 | 25 – 24 | 23 – 8 | 7 – 0 | |
| 0 0 0 0 0 0 | 0 | 0x000FF | 0x000 | Reset |

**PWM_TIMER1_PERIOD_UPMETHOD** Updating method for the active register of PWM timer1 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. (R/W)

**PWM_TIMER1_PERIOD** Period shadow register of the PWM timer1. (R/W)

**PWM_TIMER1_PRESCALE** Period of PT1_clk = Period of PWM_clk * (PWM_TIMER1_PRESCALE + 1) (R/W)

**Register 29.7. PWM_TIMER1_CFG1_REG (0x0018)**

| (reserved) | PWM_TIMER1_MOD | PWM_TIMER1_START | |
|---|---|---|---|
| 31 … 5 | 4 … 3 | 2 … 0 | |
| 0 0 0 0 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 | 0x0 | Reset |

**PWM_TIMER1_MOD** PWM timer1 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

**PWM_TIMER1_START** PWM timer1 start and stop control. 0: if PWM timer1 starts, then stops at TEZ; 1: if PWM timer1 starts, then stops at TEP; 2: PWM timer1 starts and runs on; 3: PWM timer1 starts and stops at the next TEZ; 4: PWM timer1 starts and stops at the next TEP. (R/W)

**Register 29.8. PWM_TIMER1_SYNC_REG (0x001c)**

| (reserved) | PWM_TIMER1_PHASE_DIRECTION | PWM_TIMER1_PHASE | PWM_TIMER1_SYNCO_SEL | PWM_TIMER1_SYNC_SW | PWM_TIMER1_SYNCI_EN | |
|---|---|---|---|---|---|---|
| 31 … 21 | 20 | 19 … 4 | 3 … 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_TIMER1_PHASE_DIRECTION** 0: increase; 1: decrease. (R/W)

**PWM_TIMER1_PHASE** Phase for timer reload at sync event. (R/W)

**PWM_TIMER1_SYNCO_SEL** PWM timer1 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwise: sync_out is always 0. (R/W)

**PWM_TIMER1_SYNC_SW** Toggling this bit will trigger a software sync. (R/W)

**PWM_TIMER1_SYNCI_EN** When set, timer reloading with phase at a sync input event is enabled. (R/W)

### Register 29.9. PWM_TIMER1_STATUS_REG (0x0020)

| (reserved) | PWM_TIMER1_DIRECTION | PWM_TIMER1_VALUE | |
|---|---|---|---|
| 31 … 17 | 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | Reset |

**PWM_TIMER1_DIRECTION** Current direction of the PWM timer1 counter. 0: increment 1: decrement. (RO)

**PWM_TIMER1_VALUE** Current value of the PWM timer1 counter. (RO)

### Register 29.10. PWM_TIMER2_CFG0_REG (0x0024)

| (reserved) | PWM_TIMER2_PERIOD_UPMETHOD | PWM_TIMER2_PERIOD | PWM_TIMER2_PRESCALE | |
|---|---|---|---|---|
| 31 … 26 | 25 … 24 | 23 … 8 | 7 … 0 | |
| 0 0 0 0 0 0 | 0 | 0x000FF | 0x000 | Reset |

**PWM_TIMER2_PERIOD_UPMETHOD** Updating method for active register of PWM timer2 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. (R/W)

**PWM_TIMER2_PERIOD** Period shadow register of PWM timer2. (R/W)

**PWM_TIMER2_PRESCALE** Period of PT2_clk = Period of PWM_clk * (PWM_TIMER2_PRESCALE + 1). (R/W)

**Register 29.11. PWM_TIMER2_CFG1_REG (0x0028)**

| (reserved) | PWM_TIMER2_MOD | PWM_TIMER2_START | |
|---|---|---|---|
| 31 … 5 | 4 … 3 | 2 … 0 | |
| 0 0 0 0 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 | 0x0 | Reset |

**PWM_TIMER2_MOD** PWM timer2 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

**PWM_TIMER2_START** PWM timer2 start and stop control. 0: if PWM timer2 starts, then stops at TEZ; 1: if PWM timer2 starts, then stops at TEP; 2: PWM timer2 starts and runs on; 3: PWM timer2 starts and stops at the next TEZ; 4: PWM timer2 starts and stops at the next TEP. (R/W)

**Register 29.12. PWM_TIMER2_SYNC_REG (0x002c)**

| (reserved) | PWM_TIMER2_PHASE_DIRECTION | PWM_TIMER2_PHASE | PWM_TIMER2_SYNCO_SEL | PWM_TIMER2_SYNC_SW | PWM_TIMER2_SYNCI_EN | |
|---|---|---|---|---|---|---|
| 31 … 21 | 20 | 19 … 4 | 3 … 2 | 1 | 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_TIMER2_PHASE_DIRECTION** 0: increase; 1: decrease. (R/W)

**PWM_TIMER2_PHASE** Phase for timer reload at sync event. (R/W)

**PWM_TIMER2_SYNCO_SEL** PWM timer2 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwise: sync_out is always 0. (R/W)

**PWM_TIMER2_SYNC_SW** Toggling this bit will trigger a software sync. (R/W)

**PWM_TIMER2_SYNCI_EN** When set, timer reloading with phase on sync input event is enabled. (R/W)

**Register 29.13. PWM_TIMER2_STATUS_REG (0x0030)**

| (reserved) | PWM_TIMER2_DIRECTION | PWM_TIMER2_VALUE | |
|---|---|---|---|
| 31 … 17 | 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | Reset |

**PWM_TIMER2_DIRECTION** Current direction of the PWM timer2 counter. 0: increment, 1: decrement. (RO)

**PWM_TIMER2_VALUE** Current value of the PWM timer2 counter. (RO)

**Register 29.14. PWM_TIMER_SYNCI_CFG_REG (0x0034)**

| (reserved) | PWM_EXTERNAL_SYNCI2_INVERT | PWM_EXTERNAL_SYNCI1_INVERT | PWM_EXTERNAL_SYNCI0_INVERT | PWM_TIMER2_SYNCISEL | PWM_TIMER1_SYNCISEL | PWM_TIMER0_SYNCISEL | |
|---|---|---|---|---|---|---|---|
| 31 … 12 | 11 | 10 | 9 | 8 … 6 | 5 … 3 | 2 … 0 | |
| 0 0 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 |

**PWM_EXTERNAL_SYNCI2_INVERT** Invert SYNC2 from GPIO matrix. (R/W)

**PWM_EXTERNAL_SYNCI1_INVERT** Invert SYNC1 from GPIO matrix. (R/W)

**PWM_EXTERNAL_SYNCI0_INVERT** Invert SYNC0 from GPIO matrix. (R/W)

**PWM_TIMER2_SYNCISEL** Select sync input for PWM timer2. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

**PWM_TIMER1_SYNCISEL** Select sync input for PWM timer1. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

**PWM_TIMER0_SYNCISEL** Select sync input for PWM timer0. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

### Register 29.15. PWM_OPERATOR_TIMERSEL_REG (0x0038)

| (reserved) | PWM_OPERATOR2_TIMERSEL | PWM_OPERATOR1_TIMERSEL | PWM_OPERATOR0_TIMERSEL | |
|---|---|---|---|---|
| 31 … 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 | Reset |

**PWM_OPERATOR2_TIMERSEL** Select the PWM timer for PWM operator2's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

**PWM_OPERATOR1_TIMERSEL** Select the PWM timer for PWM operator1's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

**PWM_OPERATOR0_TIMERSEL** Select the PWM timer for PWM operator0's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

### Register 29.16. PWM_GEN0_STMP_CFG_REG (0x003c)

| (reserved) | PWM_GEN0_B_SHDW_FULL | PWM_GEN0_A_SHDW_FULL | PWM_GEN0_B_UPMETHOD | PWM_GEN0_A_UPMETHOD | |
|---|---|---|---|---|---|
| 31 … 10 | 9 | 8 | 7 … 4 | 3 … 0 | |
| 0 0 0 0 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 |

**PWM_GEN0_B_SHDW_FULL** Set and reset by hardware. If set, PWM generator 0 time stamp B's shadow register.ister is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with Shadow register latest value. (RO)

**PWM_GEN0_A_SHDW_FULL** Set and reset by hardware. If set, PWM generator 0 time stamp A's shadow register.ister is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with Shadow register latest value. (RO)

**PWM_GEN0_B_UPMETHOD** Updating method for PWM generator 0 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)

**PWM_GEN0_A_UPMETHOD** Updating method for PWM generator 0 time stamp A's active register. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

### Register 29.17. PWM_GEN0_TSTMP_A_REG (0x0040)

| (reserved) | PWM_GEN0_A | |
|---|---|---|
| 31 — 16 | 15 — 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN0_A** PWM generator 0 time stamp A's shadow register. (R/W)

### Register 29.18. PWM_GEN0_TSTMP_B_REG (0x0044)

| (reserved) | PWM_GEN0_B | |
|---|---|---|
| 31 — 16 | 15 — 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN0_B** PWM generator 0 time stamp B's shadow register. (R/W)

### Register 29.19. PWM_GEN0_CFG0_REG (0x0048)

| (reserved) | PWM_GEN0_T1_SEL | PWM_GEN0_T0_SEL | PWM_GEN0_CFG_UPMETHOD | |
|---|---|---|---|---|
| 31 — 10 | 9 — 7 | 6 — 4 | 3 — 0 | |
| 0 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 |

**PWM_GEN0_T1_SEL** Source selection for PWM generator 0 event_t1, taking effect immediately. 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN0_T0_SEL** Source selection for PWM generator 0 event_t0, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN0_CFG_UPMETHOD** Updating method for PWM generator 0's active register of configuration. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

**Register 29.20. PWM_GEN0_FORCE_REG (0x004c)**

| (reserved) | PWM_GEN0_B_NCIFORCE_MODE | PWM_GEN0_B_NCIFORCE | PWM_GEN0_A_NCIFORCE_MODE | PWM_GEN0_A_NCIFORCE | PWM_GEN0_B_CNTUFORCE_MODE | PWM_GEN0_A_CNTUFORCE_MODE | PWM_GEN0_CNTUFORCE_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|
| 31 – 16 | 15 – 14 | 13 | 12 – 11 | 10 | 9 – 8 | 7 – 6 | 5 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x20 | Reset |

**PWM_GEN0_B_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM0B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN0_B_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM0B; a toggle will trigger a force event. (R/W)

**PWM_GEN0_A_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM0A, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN0_A_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM0A; a toggle will trigger a force event. (R/W)

**PWM_GEN0_B_CNTUFORCE_MODE** Continuous software-force mode for PWM0B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN0_A_CNTUFORCE_MODE** Continuous software-force mode for PWM0A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN0_CNTUFORCE_UPMETHOD** Updating method for continuous software force of PWM generator0. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer's value equals to that of register A/B.) (R/W)

**Register 29.21. PWM_GEN0_A_REG (0x0050)**

| (reserved) | PWM_GEN0_A_DT1 | PWM_GEN0_A_DT0 | PWM_GEN0_A_DTEB | PWM_GEN0_A_DTEA | PWM_GEN0_A_DTEP | PWM_GEN0_A_DTEZ | PWM_GEN0_A_UT1 | PWM_GEN0_A_UT0 | PWM_GEN0_A_UTEB | PWM_GEN0_A_UTEA | PWM_GEN0_A_UTEP | PWM_GEN0_A_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | Reset |

**PWM_GEN0_A_DT1** Action on PWM0A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN0_A_DT0** Action on PWM0A triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN0_A_DTEB** Action on PWM0A triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN0_A_DTEA** Action on PWM0A triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN0_A_DTEP** Action on PWM0A triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN0_A_DTEZ** Action on PWM0A triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN0_A_UT1** Action on PWM0A triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN0_A_UT0** Action on PWM0A triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN0_A_UTEB** Action on PWM0A triggered by event TEB when the timer increases. (R/W)

**PWM_GEN0_A_UTEA** Action on PWM0A triggered by event TEA when the timer increases. (R/W)

**PWM_GEN0_A_UTEP** Action on PWM0A triggered by event TEP when the timer increases. (R/W)

**PWM_GEN0_A_UTEZ** Action on PWM0A triggered by event TEZ when the timer increases. (R/W)

**Register 29.22. PWM_GEN0_B_REG (0x0054)**

| (reserved) | PWM_GEN0_B_DT1 | PWM_GEN0_B_DT0 | PWM_GEN0_B_DTEB | PWM_GEN0_B_DTEA | PWM_GEN0_B_DTEP | PWM_GEN0_B_DTEZ | PWM_GEN0_B_UT1 | PWM_GEN0_B_UT0 | PWM_GEN0_B_UTEB | PWM_GEN0_B_UTEA | PWM_GEN0_B_UTEP | PWM_GEN0_B_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | Reset |

**PWM_GEN0_B_DT1** Action on PWM0B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN0_B_DT0** Action on PWM0B triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN0_B_DTEB** Action on PWM0B triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN0_B_DTEA** Action on PWM0B triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN0_B_DTEP** Action on PWM0B triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN0_B_DTEZ** Action on PWM0B triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN0_B_UT1** Action on PWM0B triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN0_B_UT0** Action on PWM0B triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN0_B_UTEB** Action on PWM0B triggered by event TEB when the timer increases. (R/W)

**PWM_GEN0_B_UTEA** Action on PWM0B triggered by event TEA when the timer increases. (R/W)

**PWM_GEN0_B_UTEP** Action on PWM0B triggered by event TEP when the timer increases. (R/W)

**PWM_GEN0_B_UTEZ** Action on PWM0B triggered by event TEZ when the timer increases. (R/W)

**Register 29.23. PWM_DT0_CFG_REG (0x0058)**

| (reserved) | PWM_DT0_CLK_SEL | PWM_DT0_B_OUTBYPASS | PWM_DT0_A_OUTBYPASS | PWM_DT0_FED_OUTINVERT | PWM_DT0_RED_OUTINVERT | PWM_DT0_FED_INSEL | PWM_DT0_RED_INSEL | PWM_DT0_B_OUTSWAP | PWM_DT0_A_OUTSWAP | PWM_DT0_DEB_MODE | PWM_DT0_RED_UPMETHOD | PWM_DT0_FED_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 – 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 – 4 | 3 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_DT0_CLK_SEL** Dead time generator 0 clock selection. 0: PWM_clk, 1: PT_clk. (R/W)

**PWM_DT0_B_OUTBYPASS** S0 in Table 29.3-5. (R/W)

**PWM_DT0_A_OUTBYPASS** S1 in Table 29.3-5. (R/W)

**PWM_DT0_FED_OUTINVERT** S3 in Table 29.3-5. (R/W)

**PWM_DT0_RED_OUTINVERT** S2 in Table 29.3-5. (R/W)

**PWM_DT0_FED_INSEL** S5 in Table 29.3-5. (R/W)

**PWM_DT0_RED_INSEL** S4 in Table 29.3-5. (R/W)

**PWM_DT0_B_OUTSWAP** S7 in Table 29.3-5. (R/W)

**PWM_DT0_A_OUTSWAP** S6 in Table 29.3-5. (R/W)

**PWM_DT0_DEB_MODE** S8 in Table 29.3-5, dual-edge B mode. 0: FED/RED take effect on different paths separately, 1: FED/RED take effect on B path. (R/W)

**PWM_DT0_RED_UPMETHOD** Updating method for RED (rising edge delay) active register. 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)

**PWM_DT0_FED_UPMETHOD** Updating method for FED (falling edge delay) active register. 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)

**Register 29.24. PWM_DT0_FED_CFG_REG (0x005c)**

| (reserved) | PWM_DT0_FED | |
|---|---|---|
| 31 – 16 | 15 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT0_FED** Shadow register for FED. (R/W)

### Register 29.25. PWM_DT0_RED_CFG_REG (0x0060)

| (reserved) | PWM_DT0_RED | |
|---|---|---|
| 31 – 16 | 15 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT0_RED** Shadow register for RED. (R/W)

### Register 29.26. PWM_CARRIER0_CFG_REG (0x0064)

| (reserved) | PWM_CARRIER0_IN_INVERT | PWM_CARRIER0_OUT_INVERT | PWM_CARRIER0_OSHWTH | PWM_CARRIER0_DUTY | PWM_CARRIER0_PRESCALE | PWM_CARRIER0_EN | |
|---|---|---|---|---|---|---|---|
| 31 – 14 | 13 | 12 | 11 – 8 | 7 – 5 | 4 – 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 | Reset |

**PWM_CARRIER0_IN_INVERT** When set, invert the input of PWM0A and PWM0B for this submodule. (R/W)

**PWM_CARRIER0_OUT_INVERT** When set, invert the output of PWM0A and PWM0B for this submodule. (R/W)

**PWM_CARRIER0_OSHWTH** Width of the first pulse, in number of periods of the carrier. (R/W)

**PWM_CARRIER0_DUTY** Carrier duty selection. Duty = PWM_CARRIER0_DUTY/8. (R/W)

**PWM_CARRIER0_PRESCALE** PWM carrier0 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER0_PRESCALE + 1). (R/W)

**PWM_CARRIER0_EN** When set, carrier0 function is enabled. When cleared, carrier0 is bypassed. (R/W)

**Register 29.27. PWM_FH0_CFG0_REG (0x0068)**

| 31–24 | 23–22 | 21–20 | 19–18 | 17–16 | 15–14 | 13–12 | 11–10 | 9–8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | PWM_FH0_B_OST_U | PWM_FH0_B_OST_D | PWM_FH0_B_CBC_U | PWM_FH0_B_CBC_D | PWM_FH0_A_OST_U | PWM_FH0_A_OST_D | PWM_FH0_A_CBC_U | PWM_FH0_A_CBC_D | PWM_FH0_F0_OST | PWM_FH0_F1_OST | PWM_FH0_F2_OST | PWM_FH0_SW_OST | PWM_FH0_F0_CBC | PWM_FH0_F1_CBC | PWM_FH0_F2_CBC | PWM_FH0_SW_CBC | |
| 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 |

**PWM_FH0_B_OST_U** One-shot mode action on PWM0B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_B_OST_D** One-shot mode action on PWM0B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_B_CBC_U** Cycle-by-cycle mode action on PWM0B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_B_CBC_D** Cycle-by-cycle mode action on PWM0B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_A_OST_U** One-shot mode action on PWM0A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_A_OST_D** One-shot mode action on PWM0A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_A_CBC_U** Cycle-by-cycle mode action on PWM0A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_A_CBC_D** Cycle-by-cycle mode action on PWM0A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH0_F0_OST** event_f0 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_F1_OST** event_f1 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_F2_OST** event_f2 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_SW_OST** Enable register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_F0_CBC** event_f0 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_F1_CBC** event_f1 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_F2_CBC** event_f2 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH0_SW_CBC** Enable register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

### Register 29.28. PWM_FH0_CFG1_REG (0x006c)

| (reserved) | PWM_FH0_FORCE_OST | PWM_FH0_FORCE_CBC | PWM_FH0_CBCPULSE | PWM_FH0_CLR_OST | |
|---|---|---|---|---|---|
| 31 … 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 |

**PWM_FH0_FORCE_OST** A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

**PWM_FH0_FORCE_CBC** A toggle triggers a cycle-by-cycle mode action. (R/W)

**PWM_FH0_CBCPULSE** The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1: TEP. (R/W)

**PWM_FH0_CLR_OST** A toggle will clear on-going one-shot mode action. (R/W)

### Register 29.29. PWM_FH0_STATUS_REG (0x0070)

| (reserved) | PWM_FH0_OST_ON | PWM_FH0_CBC_ON | |
|---|---|---|---|
| 31 … 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 |

**PWM_FH0_OST_ON** Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

**PWM_FH0_CBC_ON** Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

Register 29.30. PWM_GEN1_STMP_CFG_REG (0x0074)

| (reserved) | PWM_GEN1_B_SHDW_FULL | PWM_GEN1_A_SHDW_FULL | PWM_GEN1_B_UPMETHOD | PWM_GEN1_A_UPMETHOD | |
|---|---|---|---|---|---|
| 31 … 10 | 9 | 8 | 7 … 4 | 3 … 0 | |
| 0 0 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 |

**PWM_GEN1_B_SHDW_FULL** Set and reset by hardware. If set, PWM generator 1 time stamp B's shadow register is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with shadow register's latest value. (RO)

**PWM_GEN1_A_SHDW_FULL** Set and reset by hardware. If set, PWM generator 1 time stamp A's shadow register is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with shadow register latest value. (RO)

**PWM_GEN1_B_UPMETHOD** Updating method for PWM generator 1 time stamp B's active register. 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)

**PWM_GEN1_A_UPMETHOD** Updating method for PWM generator 1 time stamp A's active register. 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)

Register 29.31. PWM_GEN1_TSTMP_A_REG (0x0078)

| (reserved) | PWM_GEN1_A | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN1_A** PWM generator 1 time stamp A's shadow register. (R/W)

Register 29.32. PWM_GEN1_TSTMP_B_REG (0x007c)

| (reserved) | PWM_GEN1_B | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN1_B** PWM generator 1 time stamp B's shadow register. (R/W)

**Register 29.33. PWM_GEN1_CFG0_REG (0x0080)**

| (reserved) | PWM_GEN1_T1_SEL | PWM_GEN1_T0_SEL | PWM_GEN1_CFG_UPMETHOD | |
|---|---|---|---|---|
| 31 … 10 | 9 … 7 | 6 … 4 | 3 … 0 | |
| 0 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 |

**PWM_GEN1_T1_SEL** Source selection for PWM generator1 event_t1, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN1_T0_SEL** Source selection for PWM generator1 event_t0, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN1_CFG_UPMETHOD** Updating method for PWM generator1's active register of configuration. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync. bit3: disable the update. (R/W)

**Register 29.34. PWM_GEN1_FORCE_REG (0x0084)**

| (reserved) | PWM_GEN1_B_NCIFORCE_MODE | PWM_GEN1_B_NCIFORCE | PWM_GEN1_A_NCIFORCE_MODE | PWM_GEN1_A_NCIFORCE | PWM_GEN1_B_CNTUFORCE_MODE | PWM_GEN1_A_CNTUFORCE_MODE | PWM_GEN1_CNTUFORCE_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|
| 31 … 16 | 15 … 14 | 13 | 12 … 11 | 10 | 9 … 8 | 7 … 6 | 5 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x20 | Reset |

**PWM_GEN1_B_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM1B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN1_B_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM1B; a toggle will trigger a force event. (R/W)

**PWM_GEN1_A_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM1A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN1_A_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM1A; a toggle will trigger a force event. (R/W)

**PWM_GEN1_B_CNTUFORCE_MODE** Continuous software-force mode for PWM1B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN1_A_CNTUFORCE_MODE** Continuous software-force mode for PWM1A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN1_CNTUFORCE_UPMETHOD** Updating method for continuous software force of PWM generator1. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer's value equals to that of register A/B). (R/W)

**Register 29.35. PWM_GEN1_A_REG (0x0088)**

| (reserved) | PWM_GEN1_A_DT1 | PWM_GEN1_A_DT0 | PWM_GEN1_A_DTEB | PWM_GEN1_A_DTEA | PWM_GEN1_A_DTEP | PWM_GEN1_A_DTEZ | PWM_GEN1_A_UT1 | PWM_GEN1_A_UT0 | PWM_GEN1_A_UTEB | PWM_GEN1_A_UTEA | PWM_GEN1_A_UTEP | PWM_GEN1_A_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | Reset |

**PWM_GEN1_A_DT1** Action on PWM1A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN1_A_DT0** Action on PWM1A triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN1_A_DTEB** Action on PWM1A triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN1_A_DTEA** Action on PWM1A triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN1_A_DTEP** Action on PWM1A triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN1_A_DTEZ** Action on PWM1A triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN1_A_UT1** Action on PWM1A triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN1_A_UT0** Action on PWM1A triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN1_A_UTEB** Action on PWM1A triggered by event TEB when the timer increases. (R/W)

**PWM_GEN1_A_UTEA** Action on PWM1A triggered by event TEA when the timer increases. (R/W)

**PWM_GEN1_A_UTEP** Action on PWM1A triggered by event TEP when the timer increases. (R/W)

**PWM_GEN1_A_UTEZ** Action on PWM1A triggered by event TEZ when the timer increases. (R/W)

**Register 29.36. PWM_GEN1_B_REG (0x008c)**

| (reserved) | PWM_GEN1_B_DT1 | PWM_GEN1_B_DT0 | PWM_GEN1_B_DTEB | PWM_GEN1_B_DTEA | PWM_GEN1_B_DTEP | PWM_GEN1_B_DTEZ | PWM_GEN1_B_UT1 | PWM_GEN1_B_UT0 | PWM_GEN1_B_UTEB | PWM_GEN1_B_UTEA | PWM_GEN1_B_UTEP | PWM_GEN1_B_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | Reset |

**PWM_GEN1_B_DT1** Action on PWM1B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN1_B_DT0** Action on PWM1B triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN1_B_DTEB** Action on PWM1B triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN1_B_DTEA** Action on PWM1B triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN1_B_DTEP** Action on PWM1B triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN1_B_DTEZ** Action on PWM1B triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN1_B_UT1** Action on PWM1B triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN1_B_UT0** Action on PWM1B triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN1_B_UTEB** Action on PWM1B triggered by event TEB when the timer increases. (R/W)

**PWM_GEN1_B_UTEA** Action on PWM1B triggered by event TEA when the timer increases. (R/W)

**PWM_GEN1_B_UTEP** Action on PWM1B triggered by event TEP when the timer increases. (R/W)

**PWM_GEN1_B_UTEZ** Action on PWM1B triggered by event TEZ when the timer increases. (R/W)

**Register 29.37. PWM_DT1_CFG_REG (0x0090)**

| (reserved) | PWM_DT1_CLK_SEL | PWM_DT1_B_OUTBYPASS | PWM_DT1_A_OUTBYPASS | PWM_DT1_FED_OUTINVERT | PWM_DT1_RED_OUTINVERT | PWM_DT1_FED_INSEL | PWM_DT1_RED_INSEL | PWM_DT1_B_OUTSWAP | PWM_DT1_A_OUTSWAP | PWM_DT1_DEB_MODE | PWM_DT1_RED_UPMETHOD | PWM_DT1_FED_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 – 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 – 4 | 3 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_DT1_CLK_SEL** Dead time generator 1 clock selection. 0: PWM_clk, 1: PT_clk. (R/W)

**PWM_DT1_B_OUTBYPASS** S0 in Table 29.3-5. (R/W)

**PWM_DT1_A_OUTBYPASS** S1 in Table 29.3-5. (R/W)

**PWM_DT1_FED_OUTINVERT** S3 in Table 29.3-5. (R/W)

**PWM_DT1_RED_OUTINVERT** S2 in Table 29.3-5. (R/W)

**PWM_DT1_FED_INSEL** S5 in Table 29.3-5. (R/W)

**PWM_DT1_RED_INSEL** S4 in Table 29.3-5. (R/W)

**PWM_DT1_B_OUTSWAP** S7 in Table 29.3-5. (R/W)

**PWM_DT1_A_OUTSWAP** S6 in Table 29.3-5. (R/W)

**PWM_DT1_DEB_MODE** S8 in Table 29.3-5; dual-edge B mode. 0: FED/RED take effect on different paths separately; 1: FED (falling edge delay)/RED (rising edge delay) take effect on B path. (R/W)

**PWM_DT1_RED_UPMETHOD** Updating method for RED active register. 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)

**PWM_DT1_FED_UPMETHOD** Updating method for FED active register. 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)

**Register 29.38. PWM_DT1_FED_CFG_REG (0x0094)**

| (reserved) | PWM_DT1_FED | |
|---|---|---|
| 31 – 16 | 15 – 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT1_FED** Shadow register for FED. (R/W)

### Register 29.39. PWM_DT1_RED_CFG_REG (0x0098)

| (reserved) | PWM_DT1_RED | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT1_RED** Shadow register for RED. (R/W)

### Register 29.40. PWM_CARRIER1_CFG_REG (0x009c)

| (reserved) | PWM_CARRIER1_IN_INVERT | PWM_CARRIER1_OUT_INVERT | PWM_CARRIER1_OSHWTH | PWM_CARRIER1_DUTY | PWM_CARRIER1_PRESCALE | PWM_CARRIER1_EN | |
|---|---|---|---|---|---|---|---|
| 31 … 14 | 13 | 12 | 11 … 8 | 7 … 5 | 4 … 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 | Reset |

**PWM_CARRIER1_IN_INVERT** When set, invert the input of PWM1A and PWM1B for this submodule. (R/W)

**PWM_CARRIER1_OUT_INVERT** When set, invert the output of PWM1A and PWM1B for this submodule. (R/W)

**PWM_CARRIER1_OSHWTH** Width of the first pulse in number of periods of the carrier. (R/W)

**PWM_CARRIER1_DUTY** Carrier duty selection. Duty = PWM_CARRIER1_DUTY/8. (R/W)

**PWM_CARRIER1_PRESCALE** PWM carrier1 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER1_PRESCALE + 1). (R/W)

**PWM_CARRIER1_EN** When set, carrier1 function is enabled. When cleared, carrier1 is bypassed. (R/W)

**Register 29.41. PWM_FH1_CFG0_REG (0x00a0)**

| (reserved) | PWM_FH1_B_OST_U | PWM_FH1_B_OST_D | PWM_FH1_B_CBC_U | PWM_FH1_B_CBC_D | PWM_FH1_A_OST_U | PWM_FH1_A_OST_D | PWM_FH1_A_CBC_U | PWM_FH1_A_CBC_D | PWM_FH1_F0_OST | PWM_FH1_F1_OST | PWM_FH1_F2_OST | PWM_FH1_SW_OST | PWM_FH1_F0_CBC | PWM_FH1_F1_CBC | PWM_FH1_F2_CBC | PWM_FH1_SW_CBC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 ... 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 | Reset |

**PWM_FH1_B_OST_U** One-shot mode action on PWM1B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_B_OST_D** One-shot mode action on PWM1B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_B_CBC_U** Cycle-by-cycle mode action on PWM1B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_B_CBC_D** Cycle-by-cycle mode action on PWM1B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_A_OST_U** One-shot mode action on PWM1A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_A_OST_D** One-shot mode action on PWM1A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_A_CBC_U** Cycle-by-cycle mode action on PWM1A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_A_CBC_D** Cycle-by-cycle mode action on PWM1A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH1_F0_OST** Enable event_f0 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_F1_OST** Enable event_f1 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_F2_OST** Enable event_f2 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_SW_OST** Enable the register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_F0_CBC** Enable event_f0 to trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_F1_CBC** Enable event_f1 to trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_F2_CBC** Enable event_f2 to will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH1_SW_CBC** Enable the register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

### Register 29.42. PWM_FH1_CFG1_REG (0x00a4)

| (reserved) | PWM_FH1_FORCE_OST | PWM_FH1_FORCE_CBC | PWM_FH1_CBCPULSE | PWM_FH1_CLR_OST | |
|---|---|---|---|---|---|
| 31 … 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 |

**PWM_FH1_FORCE_OST** A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

**PWM_FH1_FORCE_CBC** A toggle triggers a cycle-by-cycle mode action. (R/W)

**PWM_FH1_CBCPULSE** The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1: TEP. (R/W)

**PWM_FH1_CLR_OST** A toggle will clear on-going one-shot mode action. (R/W)

### Register 29.43. PWM_FH1_STATUS_REG (0x00a8)

| (reserved) | PWM_FH1_OST_ON | PWM_FH1_CBC_ON | |
|---|---|---|---|
| 31 … 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 |

**PWM_FH1_OST_ON** Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

**PWM_FH1_CBC_ON** Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

### Register 29.44. PWM_GEN2_STMP_CFG_REG (0x00ac)

| (reserved) | PWM_GEN2_B_SHDW_FULL | PWM_GEN2_A_SHDW_FULL | PWM_GEN2_B_UPMETHOD | PWM_GEN2_A_UPMETHOD | |
|---|---|---|---|---|---|
| 31 … 10 | 9 | 8 | 7 … 4 | 3 … 0 | |
| 0 0 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 |

**PWM_GEN2_B_SHDW_FULL** Set and reset by hardware. If set, PWM generator 2 time stamp B's shadow register is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with shadow register's latest value. (RO)

**PWM_GEN2_A_SHDW_FULL** Set and reset by hardware. If set, PWM generator 2 time stamp A's shadow register is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with shadow register's latest value. (RO)

**PWM_GEN2_B_UPMETHOD** Updating method for PWM generator 2 time stamp B's active register. 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)

**PWM_GEN2_A_UPMETHOD** Updating method for PWM generator 2 time stamp A's active register. 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)

### Register 29.45. PWM_GEN2_TSTMP_A_REG (0x00b0)

| (reserved) | PWM_GEN2_A | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN2_A** PWM generator 2 time stamp A's shadow register. (R/W)

### Register 29.46. PWM_GEN2_TSTMP_B_REG (0x00b4)

| (reserved) | PWM_GEN2_B | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_GEN2_B** PWM generator 2 time stamp B's shadow register. (R/W)

**Register 29.47. PWM_GEN2_CFG0_REG (0x00b8)**

| (reserved) | PWM_GEN2_T1_SEL | PWM_GEN2_T0_SEL | PWM_GEN2_CFG_UPMETHOD | |
|---|---|---|---|---|
| 31 … 10 | 9 … 7 | 6 … 4 | 3 … 0 | |
| 0 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 |

**PWM_GEN2_T1_SEL** Source selection for PWM generator2 event_t1, take effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN2_T0_SEL** Source selection for PWM generator2 event_t0, take effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

**PWM_GEN2_CFG_UPMETHOD** Updating method for PWM generator2's active register of configuration. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync. bit3: disable the update. (R/W)

**Register 29.48. PWM_GEN2_FORCE_REG (0x00bc)**

| (reserved) | PWM_GEN2_B_NCIFORCE_MODE | PWM_GEN2_B_NCIFORCE | PWM_GEN2_A_NCIFORCE_MODE | PWM_GEN2_A_NCIFORCE | PWM_GEN2_B_CNTUFORCE_MODE | PWM_GEN2_A_CNTUFORCE_MODE | PWM_GEN2_CNTUFORCE_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|
| 31 … 16 | 15 … 14 | 13 | 12 … 11 | 10 | 9 … 8 | 7 … 6 | 5 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0x20 | Reset |

**PWM_GEN2_B_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM2B, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN2_B_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM2B, a toggle will trigger a force event. (R/W)

**PWM_GEN2_A_NCIFORCE_MODE** Non-continuous immediate software-force mode for PWM2A, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN2_A_NCIFORCE** Trigger of non-continuous immediate software-force event for PWM2A, a toggle will trigger a force event. (R/W)

**PWM_GEN2_B_CNTUFORCE_MODE** Continuous software-force mode for PWM2B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN2_A_CNTUFORCE_MODE** Continuous software-force mode for PWM2A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

**PWM_GEN2_CNTUFORCE_UPMETHOD** Updating method for continuous software force of PWM generator2. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer value equals that of register A/B.) (R/W)

**Register 29.49. PWM_GEN2_A_REG (0x00c0)**

| (reserved) | PWM_GEN2_A_DT1 | PWM_GEN2_A_DT0 | PWM_GEN2_A_DTEB | PWM_GEN2_A_DTEA | PWM_GEN2_A_DTEP | PWM_GEN2_A_DTEZ | PWM_GEN2_A_UT1 | PWM_GEN2_A_UT0 | PWM_GEN2_A_UTEB | PWM_GEN2_A_UTEA | PWM_GEN2_A_UTEP | PWM_GEN2_A_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 | Reset |

**PWM_GEN2_A_DT1** Action on PWM2A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN2_A_DT0** Action on PWM2A triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN2_A_DTEB** Action on PWM2A triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN2_A_DTEA** Action on PWM2A triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN2_A_DTEP** Action on PWM2A triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN2_A_DTEZ** Action on PWM2A triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN2_A_UT1** Action on PWM2A triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN2_A_UT0** Action on PWM2A triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN2_A_UTEB** Action on PWM2A triggered by event TEB when the timer increases. (R/W)

**PWM_GEN2_A_UTEA** Action on PWM2A triggered by event TEA when the timer increases. (R/W)

**PWM_GEN2_A_UTEP** Action on PWM2A triggered by event TEP when the timer increases. (R/W)

**PWM_GEN2_A_UTEZ** Action on PWM2A triggered by event TEZ when the timer increases. (R/W)

**Register 29.50. PWM_GEN2_B_REG (0x00c4)**

| (reserved) | PWM_GEN2_B_DT1 | PWM_GEN2_B_DT0 | PWM_GEN2_B_DTEB | PWM_GEN2_B_DTEA | PWM_GEN2_B_DTEP | PWM_GEN2_B_DTEZ | PWM_GEN2_B_UT1 | PWM_GEN2_B_UT0 | PWM_GEN2_B_UTEB | PWM_GEN2_B_UTEA | PWM_GEN2_B_UTEP | PWM_GEN2_B_UTEZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 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 | Reset |

**PWM_GEN2_B_DT1** Action on PWM2B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

**PWM_GEN2_B_DT0** Action on PWM2B triggered by event_t0 when the timer decreases. (R/W)

**PWM_GEN2_B_DTEB** Action on PWM2B triggered by event TEB when the timer decreases. (R/W)

**PWM_GEN2_B_DTEA** Action on PWM2B triggered by event TEA when the timer decreases. (R/W)

**PWM_GEN2_B_DTEP** Action on PWM2B triggered by event TEP when the timer decreases. (R/W)

**PWM_GEN2_B_DTEZ** Action on PWM2B triggered by event TEZ when the timer decreases. (R/W)

**PWM_GEN2_B_UT1** Action on PWM2B triggered by event_t1 when the timer increases. (R/W)

**PWM_GEN2_B_UT0** Action on PWM2B triggered by event_t0 when the timer increases. (R/W)

**PWM_GEN2_B_UTEB** Action on PWM2B triggered by event TEB when the timer increases. (R/W)

**PWM_GEN2_B_UTEA** Action on PWM2B triggered by event TEA when the timer increases. (R/W)

**PWM_GEN2_B_UTEP** Action on PWM2B triggered by event TEP when the timer increases. (R/W)

**PWM_GEN2_B_UTEZ** Action on PWM2B triggered by event TEZ when the timer increases. (R/W)

**Register 29.51. PWM_DT2_CFG_REG (0x00c8)**

| (reserved) | PWM_DT2_CLK_SEL | PWM_DT2_B_OUTBYPASS | PWM_DT2_A_OUTBYPASS | PWM_DT2_FED_OUTINVERT | PWM_DT2_RED_OUTINVERT | PWM_DT2_FED_INSEL | PWM_DT2_RED_INSEL | PWM_DT2_B_OUTSWAP | PWM_DT2_A_OUTSWAP | PWM_DT2_DEB_MODE | PWM_DT2_RED_UPMETHOD | PWM_DT2_FED_UPMETHOD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 … 4 | 3 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**PWM_DT2_CLK_SEL** Dead time generator 1 clock selection. 0: PWM_clk; 1: PT_clk. (R/W)

**PWM_DT2_B_OUTBYPASS** S0 in Table 29.3-5. (R/W)

**PWM_DT2_A_OUTBYPASS** S1 in Table 29.3-5. (R/W)

**PWM_DT2_FED_OUTINVERT** S3 in Table 29.3-5. (R/W)

**PWM_DT2_RED_OUTINVERT** S2 in Table 29.3-5. (R/W)

**PWM_DT2_FED_INSEL** S5 in Table 29.3-5. (R/W)

**PWM_DT2_RED_INSEL** S4 in Table 29.3-5. (R/W)

**PWM_DT2_B_OUTSWAP** S7 in Table 29.3-5. (R/W)

**PWM_DT2_A_OUTSWAP** S6 in Table 29.3-5. (R/W)

**PWM_DT2_DEB_MODE** S8 in Table 29.3-5, dual-edge B mode, 0: FED/RED take effect on different path separately, 1: FED/RED take effect on B path. (R/W)

**PWM_DT2_RED_UPMETHOD** Updating method for RED (rising edge delay) active register. 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)

**PWM_DT2_FED_UPMETHOD** Updating method for FED (falling edge delay) active register. 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)

**Register 29.52. PWM_DT2_FED_CFG_REG (0x00cc)**

| (reserved) | PWM_DT2_FED | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT2_FED** Shadow register for FED. (R/W)

**Register 29.53. PWM_DT2_RED_CFG_REG (0x00d0)**

| (reserved) | PWM_DT2_RED | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 | Reset |

**PWM_DT2_RED** Shadow register for RED. (R/W)

**Register 29.54. PWM_CARRIER2_CFG_REG (0x00d4)**

| (reserved) | PWM_CARRIER2_IN_INVERT | PWM_CARRIER2_OUT_INVERT | PWM_CARRIER2_OSHWTH | PWM_CARRIER2_DUTY | PWM_CARRIER2_PRESCALE | PWM_CARRIER2_EN | |
|---|---|---|---|---|---|---|---|
| 31 … 14 | 13 | 12 | 11 … 8 | 7 … 5 | 4 … 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 | Reset |

**PWM_CARRIER2_IN_INVERT** When set, invert the input of PWM2A and PWM2B for this submodule. (R/W)

**PWM_CARRIER2_OUT_INVERT** When set, invert the output of PWM2A and PWM2B for this submodule. (R/W)

**PWM_CARRIER2_OSHWTH** Width of the first pulse in number of periods of the carrier. (R/W)

**PWM_CARRIER2_DUTY** Carrier duty selection. Duty = PWM_CARRIER2_DUTY / 8. (R/W)

**PWM_CARRIER2_PRESCALE** PWM carrier2 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER2_PRESCALE + 1). (R/W)

**PWM_CARRIER2_EN** When set, carrier2 function is enabled. When cleared, carrier2 is bypassed. (R/W)

**Register 29.55. PWM_FH2_CFG0_REG (0x00d8)**

| (reserved) | PWM_FH2_B_OST_U | PWM_FH2_B_OST_D | PWM_FH2_B_CBC_U | PWM_FH2_B_CBC_D | PWM_FH2_A_OST_U | PWM_FH2_A_OST_D | PWM_FH2_A_CBC_U | PWM_FH2_A_CBC_D | PWM_FH2_F0_OST | PWM_FH2_F1_OST | PWM_FH2_F2_OST | PWM_FH2_SW_OST | PWM_FH2_F0_CBC | PWM_FH2_F1_CBC | PWM_FH2_F2_CBC | PWM_FH2_SW_CBC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 | Reset |

**PWM_FH2_B_OST_U** One-shot mode action on PWM2B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_B_OST_D** One-shot mode action on PWM2B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_B_CBC_U** Cycle-by-cycle mode action on PWM2B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_B_CBC_D** Cycle-by-cycle mode action on PWM2B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_A_OST_U** One-shot mode action on PWM2A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_A_OST_D** One-shot mode action on PWM2A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_A_CBC_U** Cycle-by-cycle mode action on PWM2A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_A_CBC_D** Cycle-by-cycle mode action on PWM2A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

**PWM_FH2_F0_OST** event_f0 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_F1_OST** event_f1 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_F2_OST** event_f2 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_SW_OST** Enable register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_F0_CBC** event_f0 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_F1_CBC** event_f1 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_F2_CBC** event_f2 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

**PWM_FH2_SW_CBC** Enable register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

### Register 29.56. PWM_FH2_CFG1_REG (0x00dc)

| (reserved) | PWM_FH2_FORCE_OST | PWM_FH2_FORCE_CBC | PWM_FH2_CBCPULSE | PWM_FH2_CLR_OST | |
|---|---|---|---|---|---|
| 31 … 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 | 0 | Reset |

**PWM_FH2_FORCE_OST** A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

**PWM_FH2_FORCE_CBC** A toggle triggers a cycle-by-cycle mode action. (R/W)

**PWM_FH2_CBCPULSE** The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1:TEP. (R/W)

**PWM_FH2_CLR_OST** A toggle will clear on-going one-shot mode action. (R/W)

### Register 29.57. PWM_FH2_STATUS_REG (0x00e0)

| (reserved) | PWM_FH2_OST_ON | PWM_FH2_CBC_ON | |
|---|---|---|---|
| 31 … 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 | 0 | 0 | Reset |

**PWM_FH2_OST_ON** Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

**PWM_FH2_CBC_ON** Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

**Register 29.58. PWM_FAULT_DETECT_REG (0x00e4)**

| (reserved) | PWM_EVENT_F2 | PWM_EVENT_F1 | PWM_EVENT_F0 | PWM_F2_POLE | PWM_F1_POLE | PWM_F0_POLE | PWM_F2_EN | PWM_F1_EN | PWM_F0_EN | |
|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 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 |

**PWM_EVENT_F2** Set and reset by hardware. If set, event_f2 is on-going. (RO)

**PWM_EVENT_F1** Set and reset by hardware. If set, event_f1 is on-going. (RO)

**PWM_EVENT_F0** Set and reset by hardware. If set, event_f0 is on-going. (RO)

**PWM_F2_POLE** Set event_f2 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

**PWM_F1_POLE** Set event_f1 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

**PWM_F0_POLE** Set event_f0 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

**PWM_F2_EN** Set to enable the generation of event_f2. (R/W)

**PWM_F1_EN** Set to enable the generation of event_f1. (R/W)

**PWM_F0_EN** Set to enable the generation of event_f0. (R/W)

**Register 29.59. PWM_CAP_TIMER_CFG_REG (0x00e8)**

| (reserved) | PWM_CAP_SYNC_SW | PWM_CAP_SYNCI_SEL | PWM_CAP_SYNCI_EN | PWM_CAP_TIMER_EN | |
|---|---|---|---|---|---|
| 31 … 6 | 5 | 4 … 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 |

**PWM_CAP_SYNC_SW** Set this bit to force a capture timer sync; the capture timer is loaded with the value in the phase register. (WO)

**PWM_CAP_SYNCI_SEL** Capture module sync input selection. 0: none, 1: timer0 sync_out, 2: timer1 sync_out, 3: timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix. (R/W)

**PWM_CAP_SYNCI_EN** When set, the capture timer sync is enabled. (R/W)

**PWM_CAP_TIMER_EN** When set, the capture timer incrementing under APB_clk is enabled. (R/W)

### Register 29.60. PWM_CAP_TIMER_PHASE_REG (0x00ec)

| 31 … 0 | |
|---|---|
| 0 | Reset |

**PWM_CAP_TIMER_PHASE_REG** Phase value for the capture timer sync operation. (R/W)

### Register 29.61. PWM_CAP_CH0_CFG_REG (0x00f0)

| (reserved) | PWM_CAP0_SW | PWM_CAP0_IN_INVERT | PWM_CAP0_PRESCALE | PWM_CAP0_MODE | PWM_CAP0_EN | |
|---|---|---|---|---|---|---|
| 31 … 13 | 12 | 11 | 10 … 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 | Reset |

**PWM_CAP0_SW** When set, a software-forced capture on channel 0 is triggered. (WO)

**PWM_CAP0_IN_INVERT** When set, CAP0 form GPIO matrix is inverted before prescaling. (R/W)

**PWM_CAP0_PRESCALE** Prescaling value on the positive edge of CAP0. Prescaling value = PWM_CAP0_PRESCALE + 1. (R/W)

**PWM_CAP0_MODE** Edge of capture on channel 0 after prescaling. When bit0 is set to 1: enable capture on the negative edge; When bit1 is set to 1: enable capture on the positive edge. (R/W)

**PWM_CAP0_EN** When set, capture on channel 0 is enabled. (R/W)

**Register 29.62. PWM_CAP_CH1_CFG_REG (0x00f4)**

| (reserved) | PWM_CAP1_SW | PWM_CAP1_IN_INVERT | PWM_CAP1_PRESCALE | PWM_CAP1_MODE | PWM_CAP1_EN | |
|---|---|---|---|---|---|---|
| 31 … 13 | 12 | 11 | 10 … 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 | Reset |

**PWM_CAP1_SW** Write 1 will trigger a software-forced capture on channel 1. (WO)

**PWM_CAP1_IN_INVERT** When set, CAP1 form GPIO matrix is inverted before prescaling. (R/W)

**PWM_CAP1_PRESCALE** Value of prescale on the positive edge of CAP1. Prescale value = PWM_CAP1_PRESCALE + 1. (R/W)

**PWM_CAP1_MODE** Edge of capture on channel 1 after prescaling. When bit0 is set to 1: enable capture on the negative edge; When bit1 is set to 1: enable capture on the positive edge. (R/W)

**PWM_CAP1_EN** When set, capture on channel 1 is enabled. (R/W)

**Register 29.63. PWM_CAP_CH2_CFG_REG (0x00f8)**

| (reserved) | PWM_CAP2_SW | PWM_CAP2_IN_INVERT | PWM_CAP2_PRESCALE | PWM_CAP2_MODE | PWM_CAP2_EN | |
|---|---|---|---|---|---|---|
| 31 … 13 | 12 | 11 | 10 … 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 | Reset |

**PWM_CAP2_SW** When set, a software-forced capture on channel 2 is triggered. (WO)

**PWM_CAP2_IN_INVERT** When set, CAP2 form GPIO matrix is inverted before prescaling. (R/W)

**PWM_CAP2_PRESCALE** Prescaling value on the positive edge of CAP2. Prescaling value = PWM_CAP2_PRESCALE + 1. (R/W)

**PWM_CAP2_MODE** Edge of capture on channel 2 after prescaling. When bit0 is set to 1: enable capture on the negative edge; when bit1 is set to 1: enable capture on the positive edge. (R/W)

**PWM_CAP2_EN** When set, capture on channel 2 is enabled. (R/W)

### Register 29.64. PWM_CAP_CH0_REG (0x00fc)

| 31 … 0 | |
|---|---|
| 0 | Reset |

**PWM_CAP_CH0_REG** Value of the last capture on channel 0. (RO)

### Register 29.65. PWM_CAP_CH1_REG (0x0100)

| 31 … 0 | |
|---|---|
| 0 | Reset |

**PWM_CAP_CH1_REG** Value of the last capture on channel 1. (RO)

### Register 29.66. PWM_CAP_CH2_REG (0x0104)

| 31 … 0 | |
|---|---|
| 0 | Reset |

**PWM_CAP_CH2_REG** Value of the last capture on channel 2. (RO)

### Register 29.67. PWM_CAP_STATUS_REG (0x0108)

| | (reserved) | PWM_CAP2_EDGE | PWM_CAP1_EDGE | PWM_CAP0_EDGE | |
|---|---|---|---|---|---|
| | 31 … 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 |

**PWM_CAP2_EDGE** Edge of the last capture trigger on channel 2. 0: posedge; 1: negedge. (RO)

**PWM_CAP1_EDGE** Edge of the last capture trigger on channel 1. 0: posedge; 1: negedge. (RO)

**PWM_CAP0_EDGE** Edge of the last capture trigger on channel 0. 0: posedge; 1: negedge. (RO)

**Register 29.68. PWM_UPDATE_CFG_REG (0x010c)**

| 31 | | | | | | | | | | | | | | | | | | | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | | | | | | | | | | | | | | | | | | | | | | | PWM_OP2_FORCE_UP | PWM_OP2_UP_EN | PWM_OP1_FORCE_UP | PWM_OP1_UP_EN | PWM_OP0_FORCE_UP | PWM_OP0_UP_EN | PWM_GLOBAL_FORCE_UP | PWM_GLOBAL_UP_EN | |
| 0 | 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 | 1 | 0 | 1 | 0 | 1 | Reset |

**PWM_OP2_FORCE_UP** A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 2. (R/W)

**PWM_OP2_UP_EN** When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 2 are enabled (R/W)

**PWM_OP1_FORCE_UP** A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 1. (R/W)

**PWM_OP1_UP_EN** When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 1 are enabled. (R/W)

**PWM_OP0_FORCE_UP** A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 0. (R/W)

**PWM_OP0_UP_EN** When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 0 are enabled. (R/W)

**PWM_GLOBAL_FORCE_UP** A toggle (software negation of this bit's value) will trigger a forced update of all active registers in the MCPWM module. (R/W)

**PWM_GLOBAL_UP_EN** The global enable of update of all active registers in the MCPWM module. (R/W)

**Register 29.69. INT_ENA_PWM_REG (0x0110)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | INT_CAP2_INT_ENA | INT_CAP1_INT_ENA | INT_CAP0_INT_ENA | INT_FH2_OST_INT_ENA | INT_FH1_OST_INT_ENA | INT_FH0_OST_INT_ENA | INT_FH2_CBC_INT_ENA | INT_FH1_CBC_INT_ENA | INT_FH0_CBC_INT_ENA | INT_OP2_TEB_INT_ENA | INT_OP1_TEB_INT_ENA | INT_OP0_TEB_INT_ENA | INT_OP2_TEA_INT_ENA | INT_OP1_TEA_INT_ENA | INT_OP0_TEA_INT_ENA | INT_FAULT2_CLR_INT_ENA | INT_FAULT1_CLR_INT_ENA | INT_FAULT0_CLR_INT_ENA | INT_FAULT2_INT_ENA | INT_FAULT1_INT_ENA | INT_FAULT0_INT_ENA | INT_TIMER2_TEP_INT_ENA | INT_TIMER1_TEP_INT_ENA | INT_TIMER0_TEP_INT_ENA | INT_TIMER2_TEZ_INT_ENA | INT_TIMER1_TEZ_INT_ENA | INT_TIMER0_TEZ_INT_ENA | INT_TIMER2_STOP_INT_ENA | INT_TIMER1_STOP_INT_ENA | INT_TIMER0_STOP_INT_ENA | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**INT_CAP2_INT_ENA** The enable bit for the interrupt triggered by capture on channel 2. (R/W)

**INT_CAP1_INT_ENA** The enable bit for the interrupt triggered by capture on channel 1. (R/W)

**INT_CAP0_INT_ENA** The enable bit for the interrupt triggered by capture on channel 0. (R/W)

**INT_FH2_OST_INT_ENA** The enable bit for the interrupt triggered by a one-shot mode action on PWM2. (R/W)

**INT_FH1_OST_INT_ENA** The enable bit for the interrupt triggered by a one-shot mode action on PWM1. (R/W)

**INT_FH0_OST_INT_ENA** The enable bit for the interrupt triggered by a one-shot mode action on PWM0. (R/W)

**INT_FH2_CBC_INT_ENA** The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (R/W)

**INT_FH1_CBC_INT_ENA** The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (R/W)

**INT_FH0_CBC_INT_ENA** The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (R/W)

**INT_OP2_TEB_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 2 TEB event (R/W)

**INT_OP1_TEB_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 1 TEB event (R/W)

**INT_OP0_TEB_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 0 TEB event (R/W)

**INT_OP2_TEA_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 2 TEA event (R/W)

**INT_OP1_TEA_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 1 TEA event (R/W)

**INT_OP0_TEA_INT_ENA** The enable bit for the interrupt triggered by a PWM operator 0 TEA event (R/W)

**INT_FAULT2_CLR_INT_ENA** The enable bit for the interrupt triggered when event_f2 ends. (R/W)

**INT_FAULT1_CLR_INT_ENA** The enable bit for the interrupt triggered when event_f1 ends. (R/W)

**INT_FAULT0_CLR_INT_ENA** The enable bit for the interrupt triggered when event_f0 ends. (R/W)

**INT_FAULT2_INT_ENA** The enable bit for the interrupt triggered when event_f2 starts. (R/W)

**INT_FAULT1_INT_ENA** The enable bit for the interrupt triggered when event_f1 starts. (R/W)

**INT_FAULT0_INT_ENA** The enable bit for the interrupt triggered when event_f0 starts. (R/W)

**INT_TIMER2_TEP_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 2 TEP event. (R/W)

**INT_TIMER1_TEP_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 1 TEP event. (R/W)

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

Register 29.69. INT_ENA_PWM_REG (0x0110)

Continued from the previous page...

**INT_TIMER0_TEP_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 0 TEP event. (R/W)

**INT_TIMER2_TEZ_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 2 TEZ event. (R/W)

**INT_TIMER1_TEZ_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 1 TEZ event. (R/W)

**INT_TIMER0_TEZ_INT_ENA** The enable bit for the interrupt triggered by a PWM timer 0 TEZ event. (R/W)

**INT_TIMER2_STOP_INT_ENA** The enable bit for the interrupt triggered when the timer 2 stops. (R/W)

**INT_TIMER1_STOP_INT_ENA** The enable bit for the interrupt triggered when the timer 1 stops. (R/W)

**INT_TIMER0_STOP_INT_ENA** The enable bit for the interrupt triggered when the timer 0 stops. (R/W)

**Register 29.70. INT_RAW_PWM_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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | INT_CAP2_INT_RAW | INT_CAP1_INT_RAW | INT_CAP0_INT_RAW | INT_FH2_OST_INT_RAW | INT_FH1_OST_INT_RAW | INT_FH0_OST_INT_RAW | INT_FH2_CBC_INT_RAW | INT_FH1_CBC_INT_RAW | INT_FH0_CBC_INT_RAW | INT_OP2_TEB_INT_RAW | INT_OP1_TEB_INT_RAW | INT_OP0_TEB_INT_RAW | INT_OP2_TEA_INT_RAW | INT_OP1_TEA_INT_RAW | INT_OP0_TEA_INT_RAW | INT_FAULT2_CLR_INT_RAW | INT_FAULT1_CLR_INT_RAW | INT_FAULT0_CLR_INT_RAW | INT_FAULT2_INT_RAW | INT_FAULT1_INT_RAW | INT_FAULT0_INT_RAW | INT_TIMER2_TEP_INT_RAW | INT_TIMER1_TEP_INT_RAW | INT_TIMER0_TEP_INT_RAW | INT_TIMER2_TEZ_INT_RAW | INT_TIMER1_TEZ_INT_RAW | INT_TIMER0_TEZ_INT_RAW | INT_TIMER2_STOP_INT_RAW | INT_TIMER1_STOP_INT_RAW | INT_TIMER0_STOP_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**INT_CAP2_INT_RAW** The raw status bit for the interrupt triggered by capture on channel 2. (RO)

**INT_CAP1_INT_RAW** The raw status bit for the interrupt triggered by capture on channel 1. (RO)

**INT_CAP0_INT_RAW** The raw status bit for the interrupt triggered by capture on channel 0. (RO)

**INT_FH2_OST_INT_RAW** The raw status bit for the interrupt triggered by a one-shot mode action on PWM2. (RO)

**INT_FH1_OST_INT_RAW** The raw status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

**INT_FH0_OST_INT_RAW** The raw status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

**INT_FH2_CBC_INT_RAW** The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (RO)

**INT_FH1_CBC_INT_RAW** The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (RO)

**INT_FH0_CBC_INT_RAW** The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (RO)

**INT_OP2_TEB_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 2 TEB event. (RO)

**INT_OP1_TEB_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 1 TEB event. (RO)

**INT_OP0_TEB_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 0 TEB event. (RO)

**INT_OP2_TEA_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 2 TEA event. (RO)

**INT_OP1_TEA_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 1 TEA event. (RO)

**INT_OP0_TEA_INT_RAW** The raw status bit for the interrupt triggered by a PWM operator 0 TEA event. (RO)

**INT_FAULT2_CLR_INT_RAW** The raw status bit for the interrupt triggered when event_f2 ends. (RO)

**INT_FAULT1_CLR_INT_RAW** The raw status bit for the interrupt triggered when event_f1 ends. (RO)

**INT_FAULT0_CLR_INT_RAW** The raw status bit for the interrupt triggered when event_f0 ends. (RO)

**INT_FAULT2_INT_RAW** The raw status bit for the interrupt triggered when event_f2 starts. (RO)

**INT_FAULT1_INT_RAW** The raw status bit for the interrupt triggered when event_f1 starts. (RO)

**INT_FAULT0_INT_RAW** The raw status bit for the interrupt triggered when event_f0 starts. (RO)

**INT_TIMER2_TEP_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 2 TEP event. (RO)

**INT_TIMER1_TEP_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 1 TEP event. (RO)

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

Register 29.70. INT_RAW_PWM_REG (0x0114)

Continued from the previous page...

**INT_TIMER0_TEP_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 0 TEP event. (RO)

**INT_TIMER2_TEZ_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 2 TEZ event. (RO)

**INT_TIMER1_TEZ_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 1 TEZ event. (RO)

**INT_TIMER0_TEZ_INT_RAW** The raw status bit for the interrupt triggered by a PWM timer 0 TEZ event. (RO)

**INT_TIMER2_STOP_INT_RAW** The raw status bit for the interrupt triggered when the timer 2 stops. (RO)

**INT_TIMER1_STOP_INT_RAW** The raw status bit for the interrupt triggered when the timer 1 stops. (RO)

**INT_TIMER0_STOP_INT_RAW** The raw status bit for the interrupt triggered when the timer 0 stops. (RO)

**Register 29.71. INT_ST_PWM_REG (0x0118)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | INT_CAP2_INT_ST | INT_CAP1_INT_ST | INT_CAP0_INT_ST | INT_FH2_OST_INT_ST | INT_FH1_OST_INT_ST | INT_FH0_OST_INT_ST | INT_FH2_CBC_INT_ST | INT_FH1_CBC_INT_ST | INT_FH0_CBC_INT_ST | INT_OP2_TEB_INT_ST | INT_OP1_TEB_INT_ST | INT_OP0_TEB_INT_ST | INT_OP2_TEA_INT_ST | INT_OP1_TEA_INT_ST | INT_OP0_TEA_INT_ST | INT_FAULT2_CLR_INT_ST | INT_FAULT1_CLR_INT_ST | INT_FAULT0_CLR_INT_ST | INT_FAULT2_INT_ST | INT_FAULT1_INT_ST | INT_FAULT0_INT_ST | INT_TIMER2_TEP_INT_ST | INT_TIMER1_TEP_INT_ST | INT_TIMER0_TEP_INT_ST | INT_TIMER2_TEZ_INT_ST | INT_TIMER1_TEZ_INT_ST | INT_TIMER0_TEZ_INT_ST | INT_TIMER2_STOP_INT_ST | INT_TIMER1_STOP_INT_ST | INT_TIMER0_STOP_INT_ST | |
| 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**INT_CAP2_INT_ST** The masked status bit for the interrupt triggered by capture on channel 2. (RO)

**INT_CAP1_INT_ST** The masked status bit for the interrupt triggered by capture on channel 1. (RO)

**INT_CAP0_INT_ST** The masked status bit for the interrupt triggered by capture on channel 0. (RO)

**INT_FH2_OST_INT_ST** The masked status bit for the interrupt triggered by a one-shot mode action on PWM2. (RO)

**INT_FH1_OST_INT_ST** The masked status bit for the interrupt triggered by a one-shot mode action on PWM1. (RO)

**INT_FH0_OST_INT_ST** The masked status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

**INT_FH2_CBC_INT_ST** The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (RO)

**INT_FH1_CBC_INT_ST** The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (RO)

**INT_FH0_CBC_INT_ST** The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (RO)

**INT_OP2_TEB_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 2 TEB event. (RO)

**INT_OP1_TEB_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 1 TEB event. (RO)

**INT_OP0_TEB_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 0 TEB event. (RO)

**INT_OP2_TEA_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 2 TEA event. (RO)

**INT_OP1_TEA_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 1 TEA event. (RO)

**INT_OP0_TEA_INT_ST** The masked status bit for the interrupt triggered by a PWM operator 0 TEA event. (RO)

**INT_FAULT2_CLR_INT_ST** The masked status bit for the interrupt triggered when event_f2 ends. (RO)

**INT_FAULT1_CLR_INT_ST** The masked status bit for the interrupt triggered when event_f1 ends. (RO)

**INT_FAULT0_CLR_INT_ST** The masked status bit for the interrupt triggered when event_f0 ends. (RO)

**INT_FAULT2_INT_ST** The masked status bit for the interrupt triggered when event_f2 starts. (RO)

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

Register 29.71. INT_ST_PWM_REG (0x0118)

Continued from the previous page...

**INT_FAULT1_INT_ST** The masked status bit for the interrupt triggered when event_f1 starts. (RO)

**INT_FAULT0_INT_ST** The masked status bit for the interrupt triggered when event_f0 starts. (RO)

**INT_TIMER2_TEP_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 2 TEP event. (RO)

**INT_TIMER1_TEP_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 1 TEP event. (RO)

**INT_TIMER0_TEP_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 0 TEP event. (RO)

**INT_TIMER2_TEZ_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 2 TEZ event. (RO)

**INT_TIMER1_TEZ_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 1 TEZ event. (RO)

**INT_TIMER0_TEZ_INT_ST** The masked status bit for the interrupt triggered by a PWM timer 0 TEZ event. (RO)

**INT_TIMER2_STOP_INT_ST** The masked status bit for the interrupt triggered when the timer 2 stops. (RO)

**INT_TIMER1_STOP_INT_ST** The masked status bit for the interrupt triggered when the timer 1 stops. (RO)

**INT_TIMER0_STOP_INT_ST** The masked status bit for the interrupt triggered when the timer 0 stops. (RO)

**Register 29.72. INT_CLR_PWM_REG (0x011c)**

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (reserved) | | INT_CAP2_INT_CLR | INT_CAP1_INT_CLR | INT_CAP0_INT_CLR | INT_FH2_OST_INT_CLR | INT_FH1_OST_INT_CLR | INT_FH0_OST_INT_CLR | INT_FH2_CBC_INT_CLR | INT_FH1_CBC_INT_CLR | INT_FH0_CBC_INT_CLR | INT_OP2_TEB_INT_CLR | INT_OP1_TEB_INT_CLR | INT_OP0_TEB_INT_CLR | INT_OP2_TEA_INT_CLR | INT_OP1_TEA_INT_CLR | INT_OP0_TEA_INT_CLR | INT_FAULT2_CLR_INT_CLR | INT_FAULT1_CLR_INT_CLR | INT_FAULT0_CLR_INT_CLR | INT_FAULT2_INT_CLR | INT_FAULT1_INT_CLR | INT_FAULT0_INT_CLR | INT_TIMER2_TEP_INT_CLR | INT_TIMER1_TEP_INT_CLR | INT_TIMER0_TEP_INT_CLR | INT_TIMER2_TEZ_INT_CLR | INT_TIMER1_TEZ_INT_CLR | INT_TIMER0_TEZ_INT_CLR | INT_TIMER2_STOP_INT_CLR | INT_TIMER1_STOP_INT_CLR | INT_TIMER0_STOP_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**INT_CAP2_INT_CLR** Set this bit to clear interrupt triggered by capture on channel 2. (WO)

**INT_CAP1_INT_CLR** Set this bit to clear interrupt triggered by capture on channel 1. (WO)

**INT_CAP0_INT_CLR** Set this bit to clear interrupt triggered by capture on channel 0. (WO)

**INT_FH2_OST_INT_CLR** Set this bit to clear interrupt triggered by a one-shot mode action on PWM2. (WO)

**INT_FH1_OST_INT_CLR** Set this bit to clear interrupt triggered by a one-shot mode action on PWM1. (WO)

**INT_FH0_OST_INT_CLR** Set this bit to clear interrupt triggered by a one-shot mode action on PWM0. (WO)

**INT_FH2_CBC_INT_CLR** Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM2. (WO)

**INT_FH1_CBC_INT_CLR** Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM1. (WO)

**INT_FH0_CBC_INT_CLR** Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM0. (WO)

**INT_OP2_TEB_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 2 TEB event. (WO)

**INT_OP1_TEB_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 1 TEB event. (WO)

**INT_OP0_TEB_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 0 TEB event. (WO)

**INT_OP2_TEA_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 2 TEA event. (WO)

**INT_OP1_TEA_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 1 TEA event. (WO)

**INT_OP0_TEA_INT_CLR** Set this bit to clear interrupt triggered by a PWM operator 0 TEA event. (WO)

**INT_FAULT2_CLR_INT_CLR** Set this bit to clear interrupt triggered when event_f2 ends. (WO)

**INT_FAULT1_CLR_INT_CLR** Set this bit to clear interrupt triggered when event_f1 ends. (WO)

**INT_FAULT0_CLR_INT_CLR** Set this bit to clear interrupt triggered when event_f0 ends. (WO)

**INT_FAULT2_INT_CLR** Set this bit to clear interrupt triggered when event_f2 starts. (WO)

**INT_FAULT1_INT_CLR** Set this bit to clear interrupt triggered when event_f1 starts. (WO)

**INT_FAULT0_INT_CLR** Set this bit to clear interrupt triggered when event_f0 starts. (WO)

**INT_TIMER2_TEP_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 2 TEP event. (WO)

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

Register 29.72. INT_CLR_PWM_REG (0x011c)

Continued from the previous page...

**INT_TIMER1_TEP_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 1 TEP event. (WO)

**INT_TIMER0_TEP_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 0 TEP event. (WO)

**INT_TIMER2_TEZ_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 2 TEZ event. (WO)

**INT_TIMER1_TEZ_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 1 TEZ event. (WO)

**INT_TIMER0_TEZ_INT_CLR** Set this bit to clear interrupt triggered by a PWM timer 0 TEZ event. (WO)

**INT_TIMER2_STOP_INT_CLR** Set this bit to clear interrupt triggered when the timer 2 stops. (WO)

**INT_TIMER1_STOP_INT_CLR** Set this bit to clear interrupt triggered when the timer 1 stops. (WO)

**INT_TIMER0_STOP_INT_CLR** Set this bit to clear interrupt triggered when the timer 0 stops. (WO)

# Chapter 30

# Remote Control Peripheral (RMT)

## 30.1 Introduction

The RMT (Remote Control) module is primarily designed to send and receive infrared remote control signals that implement on-off keying in a carrier frequency, but due to its design it can be used to generate various types of signals. An RMT transmitter does this by reading consecutive duration values of an active and inactive output from the built-in RAM block, optionally modulating it with a carrier wave. A receiver will inspect its input signal, optionally filtering it, and will place the lengths of time the signal is active and inactive in the RAM block.

The RMT module has eight channels, numbered zero to seven; registers, signals and blocks that are duplicated in each channel are indicated by an *n* which is used as a placeholder for the channel number.

## 30.2 Functional Description

### 30.2.1 RMT Architecture

The RMT module contains eight channels. Each channel has both a transmitter and a receiver, but only one of them can be active in every channel. The eight channels share a 512x32-bit RAM block which can be read and written by the processor cores over the APB bus, read by the transmitters, and written by the receivers. The transmitted signal can optionally be modulated by a carrier wave. Each channel is clocked by a divided-down signal derived from either the APB bus clock or REF_TICK.

### 30.2.2 RMT RAM

The data structure in RAM is shown in Figure 30.2-2. Each 32-bit value contains two 16-bit entries, with two fields in every entry, "level" and "period". "Level" indicates whether a high-/low-level value was received or is going to be sent, while "period" points out the divider-clock cycles for which the level lasts. A zero period is interpreted as an end-marker: the transmitter will stop transmitting once it has read this, and the receiver will write this, once it has detected that the signal it received has gone idle.

Normally, only one block of 64x32-bit worth of data can be sent or received. If the data size is larger than this block size, blocks can be extended or the channel can be configured for the wraparound mode.

The RMT RAM can be accessed via the APB bus. The initial address is 0x3FF56800. The RAM block is divided into eight 64x32-bit blocks. By default, each channel uses one block (block zero for channel zero, block one for channel one, and so on). Users can extend the memory to a specific channel by configuring the RMT_MEM_SIZE_CH*n* register; setting this to >1 will prompt the channel to use the memory of subsequent

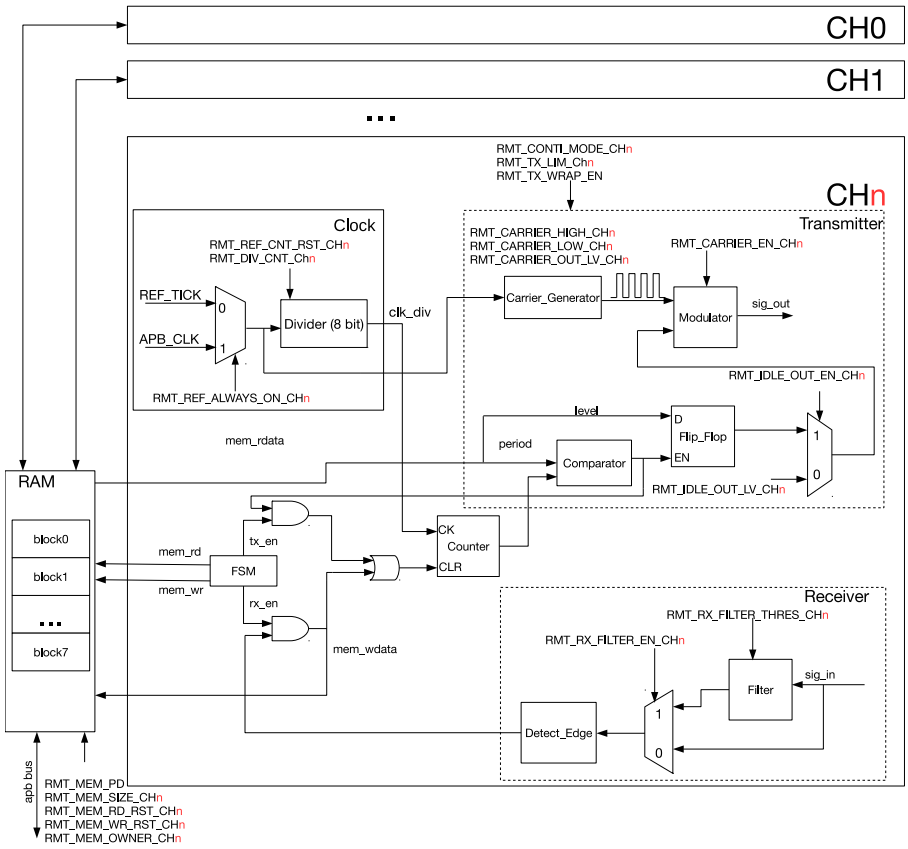



Figure 30.2-1. RMT Architecture

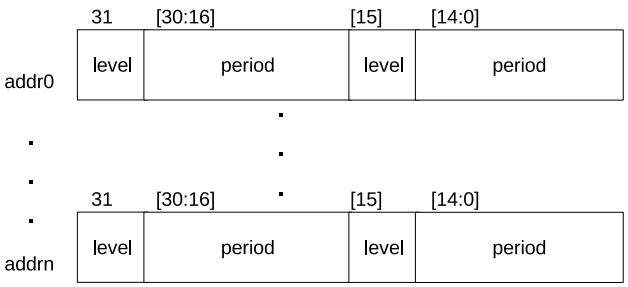



Figure 30.2-2. Data Structure

channels as well. The RAM address range of channel *n* is start_addr_CH*n* to end_addr_CH*n*, which is defined by:

start_addr_ch*n* $= 0x3FF56800 + 64 * 4 * n$, and
end_addr_ch*n* $= 0x3FF56800 + (64 * 4 * n + 64 * 4 *$ RMT_MEM_SIZE_CH*n*$)\text{mod}(512 * 4) - 4$

To protect a receiver from overwriting the blocks a transmitter is about to transmit, RMT_MEM_OWNER_CH*n* can be configured to designate the owner, be it a transmitter or receiver, of channel *n*'s RAM block. This way, if this ownership is violated, the RMT_CH*n*_ERR interrupt will be generated.

**Note:** When enabling the continuous transmission mode by setting RMT_REG_TX_CONTI_MODE, the transmitter will transmit the data on the channel continuously, that is, from the first byte to the last one, then from the first to the last again, and so on. In this mode, there will be an idle level lasting one clk_div cycle between N and N+1 transmissions.

### 30.2.3 Clock

The main clock of a channel is generated by taking either the 80 MHz APB clock or REF_TICK (usually 1MHz), according to the state of RMT_REF_ALWAYS_ON_CH*n*. (For more information on clock sources, please see Chapter Reset And Clock.) Then, the aforementioned state gets scaled down using a configurable 8-bit divider to create the channel clock which is used by both the carrier wave generator and the counter. The divider value can be set by configuring RMT_DIV_CNT_CH*n*.

### 30.2.4 Transmitter

When the RMT_TX_START_CH*n* register is 1, the transmitter of channel *n* will start reading and sending data from RAM. The transmitter will receive a 32-bit value each time it reads from RAM. Of these 32 bits, the low 16-bit entry is sent first and the high entry second.

To transmit more data than can be fitted in the channel's RAM, the wraparound mode can be enabled. In this mode, when the transmitter has reached the last entry in the channel's memory, it will loop back to the first byte. To use this mechanism for sending more data than can be fitted in the channel's RAM, fill the RAM with the initial events and set RMT_CH*n*_TX_LIM_REG to cause an RMT_CH*n*_TX_THR_EVENT_INT interrupt before the wraparound happens. Then, when the interrupt happens, the already sent data should be replaced by subsequent events, so that when the wraparound happens the transmitter will seamlessly continue sending the new events.

With or without the wraparound mode enabled, transmission ends when an entry with zero length is encountered. When this happens, the transmitter will generate an RMT_CH*n*_TX_END_INT interrupt and return to the idle state. When a transmitter is in the idle state, the output level defaults to end-mark 0. Users can also configure RMT_IDLE_OUT_EN_CH*n* and RMT_IDLE_OUT_LV_CH*n* to control the output level manually.

The output of the transmitter can be modulated using a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier frequency and duty cycle can be configured by adjusting the carrier's high and low durations in channel-clock cycles, in RMT_CARRIER_HIGH_CH*n* and RMT_CARRIER_LOW_CH*n*.

### 30.2.5 Receiver

When RMT_RX_EN_CH*n* is set to 1, the receiver in channel *n* becomes active, measuring the duration between input signal edges. These will be written as period/level value pairs to the channel RAM in the same fashion as the transmitter sends them. Receiving ends, when the receiver detects no change in signal level for more than RMT_IDLE_THRES_CH*n* channel clock ticks. The receiver will write a final entry with 0 period, generate an RMT_CH*n*_RX_END_INT_RAW interrupt and return to the idle state.

The receiver has an input signal filter which can be configured using RMT_RX_FILTER_EN_CH*n*: The filter will remove pulses with a length of less than RMT_RX_FILTER_THRES_CH*n* in APB clock periods.

When the RMT module is inactive, the RAM can be put into low-power mode by setting the RMT_MEM_PD register to 1.

### 30.2.6 Interrupts

- 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*n*_TX_END_INT: Triggered when the transmitter has finished transmitting the signal.
- RMT_CH*n*_RX_END_INT: Triggered when the receiver has finished receiving a signal.

## 30.3 Register Summary

The addresses in this section are relative to the RMT base address provided in Table 3.3-6 in Chapter 3 *System and Memory*.

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

| Name | Description | Address | Access |
|---|---|---|---|
| **Configuration registers** | | | |
| RMT_CH0CONF0_REG | Channel 0 config register 0 | 0x3FF56020 | R/W |
| RMT_CH0CONF1_REG | Channel 0 config register 1 | 0x3FF56024 | R/W |
| RMT_CH1CONF0_REG | Channel 1 config register 0 | 0x3FF56028 | R/W |
| RMT_CH1CONF1_REG | Channel 1 config register 1 | 0x3FF5602C | R/W |
| RMT_CH2CONF0_REG | Channel 2 config register 0 | 0x3FF56030 | R/W |
| RMT_CH2CONF1_REG | Channel 2 config register 1 | 0x3FF56034 | R/W |
| RMT_CH3CONF0_REG | Channel 3 config register 0 | 0x3FF56038 | R/W |
| RMT_CH3CONF1_REG | Channel 3 config register 1 | 0x3FF5603C | R/W |
| RMT_CH4CONF0_REG | Channel 4 config register 0 | 0x3FF56040 | R/W |
| RMT_CH4CONF1_REG | Channel 4 config register 1 | 0x3FF56044 | R/W |
| RMT_CH5CONF0_REG | Channel 5 config register 0 | 0x3FF56048 | R/W |
| RMT_CH5CONF1_REG | Channel 5 config register 1 | 0x3FF5604C | R/W |
| RMT_CH6CONF0_REG | Channel 6 config register 0 | 0x3FF56050 | R/W |
| RMT_CH6CONF1_REG | Channel 6 config register 1 | 0x3FF56054 | R/W |
| RMT_CH7CONF0_REG | Channel 7 config register 0 | 0x3FF56058 | R/W |
| RMT_CH7CONF1_REG | Channel 7 config register 1 | 0x3FF5605C | R/W |
| **Interrupt registers** | | | |
| RMT_INT_RAW_REG | Raw interrupt status | 0x3FF560A0 | RO |
| RMT_INT_ST_REG | Masked interrupt status | 0x3FF560A4 | RO |
| RMT_INT_ENA_REG | Interrupt enable bits | 0x3FF560A8 | R/W |
| RMT_INT_CLR_REG | Interrupt clear bits | 0x3FF560AC | WO |
| **Carrier wave duty cycle registers** | | | |
| RMT_CH0CARRIER_DUTY_REG | Channel 0 duty cycle configuration register | 0x3FF560B0 | R/W |
| RMT_CH1CARRIER_DUTY_REG | Channel 1 duty cycle configuration register | 0x3FF560B4 | R/W |
| RMT_CH2CARRIER_DUTY_REG | Channel 2 duty cycle configuration register | 0x3FF560B8 | R/W |
| RMT_CH3CARRIER_DUTY_REG | Channel 3 duty cycle configuration register | 0x3FF560BC | R/W |
| RMT_CH4CARRIER_DUTY_REG | Channel 4 duty cycle configuration register | 0x3FF560C0 | R/W |
| RMT_CH5CARRIER_DUTY_REG | Channel 5 duty cycle configuration register | 0x3FF560C4 | R/W |
| RMT_CH6CARRIER_DUTY_REG | Channel 6 duty cycle configuration register | 0x3FF560C8 | R/W |
| RMT_CH7CARRIER_DUTY_REG | Channel 7 duty cycle configuration register | 0x3FF560CC | R/W |
| **Tx event configuration registers** | | | |
| RMT_CH0_TX_LIM_REG | Channel 0 Tx event configuration register | 0x3FF560D0 | R/W |
| RMT_CH1_TX_LIM_REG | Channel 1 Tx event configuration register | 0x3FF560D4 | R/W |

| | | | |
|---|---|---|---|
| RMT_CH2_TX_LIM_REG | Channel 2 Tx event configuration register | 0x3FF560D8 | R/W |
| RMT_CH3_TX_LIM_REG | Channel 3 Tx event configuration register | 0x3FF560DC | R/W |
| RMT_CH4_TX_LIM_REG | Channel 4 Tx event configuration register | 0x3FF560E0 | R/W |
| RMT_CH5_TX_LIM_REG | Channel 5 Tx event configuration register | 0x3FF560E4 | R/W |
| RMT_CH6_TX_LIM_REG | Channel 6 Tx event configuration register | 0x3FF560E8 | R/W |
| RMT_CH7_TX_LIM_REG | Channel 7 Tx event configuration register | 0x3FF560EC | R/W |
| **Other registers** | | | |
| RMT_APB_CONF_REG | RMT-wide configuration register | 0x3FF560F0 | R/W |

## 30.4 Registers

The addresses in this section are relative to the RMT base address provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 30.3 *Register Summary*.

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

**Register 30.1. RMT_CH*n*CONF0_REG (*n*: 0-7) (0x0020+8**n*)**

| (reserved) | RMT_MEM_PD | RMT_CARRIER_OUT_LV_CH*n* | RMT_CARRIER_EN_CH*n* | RMT_MEM_SIZE_CH*n* | RMT_IDLE_THRES_CH*n* | RMT_DIV_CNT_CH*n* | |
|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 24 | 23 8 | 7 0 | |
| 0x0 | 0 | 1 | 1 | 0x01 | 0x01000 | 0x002 | Reset |

**RMT_MEM_PD** This bit is used to power down the entire RMT RAM block. (It only exists in RMT_CHOCONF0). 1: power down memory; 0: power up memory. (R/W)

**RMT_CARRIER_OUT_LV_CH*n*** This bit is used for configuration when the carrier wave is being transmitted. Transmit on low output level with 0, and transmit on high output level with 1. (R/W)

**RMT_CARRIER_EN_CH*n*** This is the carrier modulation enable-control bit for channel *n*. Carrier modulation is enabled with 1, while carrier modulation is disabled with 0. (R/W)

**RMT_MEM_SIZE_CH*n*** This register is used to configure the amount of memory blocks allocated to channel *n*. (R/W)

**RMT_IDLE_THRES_CH*n*** In receive mode, when no edge is detected on the input signal for longer than REG_IDLE_THRES_CH*n* channel clock cycles, the receive process is finished. (R/W)

**RMT_DIV_CNT_CH*n*** This register is used to set the divider for the channel clock of channel *n*. (R/W)

**Register 30.2. RMT_CH*n*CONF1_REG (*n*: 0-7) (0x0024+8**n*)**

| (reserved) | RMT_IDLE_OUT_EN_CH*n* | RMT_IDLE_OUT_LV_CH*n* | RMT_REF_ALWAYS_ON_CH*n* | RMT_REF_CNT_RST_CH*n* | RMT_RX_FILTER_THRES_CH*n* | RMT_RX_FILTER_EN_CH*n* | RMT_TX_CONTI_MODE_CH*n* | RMT_MEM_OWNER_CH*n* | (reserved) | RMT_MEM_RD_RST_CH*n* | RMT_MEM_WR_RST_CH*n* | RMT_RX_EN_CH*n* | RMT_TX_START_CH*n* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 20 | 19 | 18 | 17 | 16 | 15 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| 0x0000 | 0 | 0 | 0 | 0 | 0x00F | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | Reset |

**RMT_IDLE_OUT_EN_CH*n*** This is the output enable-control bit for channel *n* in IDLE state. (R/W)

**RMT_IDLE_OUT_LV_CH*n*** This bit configures the level of output signals in channel *n* when the latter is in IDLE state. (R/W)

**RMT_REF_ALWAYS_ON_CH*n*** This bit is used to select the channel's base clock. 1:clk_apb; 0:clk_ref. (R/W)

**RMT_REF_CNT_RST_CH*n*** Setting this bit resets the clock divider of channel *n*. (R/W)

**RMT_RX_FILTER_THRES_CH*n*** In receive mode, channel *n* ignores input pulse when the pulse width is smaller than this value in APB clock periods. (R/W)

**RMT_RX_FILTER_EN_CH*n*** This is the receive filter's enable-bit for channel *n*. (R/W)

**RMT_TX_CONTI_MODE_CH*n*** If this bit is set, instead of going to an idle state when transmission ends, the transmitter will restart transmission. This results in a repeating output signal. (R/W)

**RMT_MEM_OWNER_CH*n*** This bit marks channel *n*'s RAM block ownership. Number 1 indicates that the receiver is using the RAM, while 0 indicates that the transmitter is using the RAM. (R/W)

**RMT_MEM_RD_RST_CH*n*** Set this bit to reset the read-RAM address for channel *n* by accessing the transmitter. (R/W)

**RMT_MEM_WR_RST_CH*n*** Set this bit to reset the write-RAM address for channel *n* by accessing the receiver. (R/W)

**RMT_RX_EN_CH*n*** Set this bit to enable receiving data on channel *n*. (R/W)

**RMT_TX_START_CH*n*** Set this bit to start sending data on channel *n*. (R/W)

### Register 30.3. RMT_INT_RAW_REG (0x00A0)

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RMT_CH7_TX_THR_EVENT_INT_RAW | RMT_CH6_TX_THR_EVENT_INT_RAW | RMT_CH5_TX_THR_EVENT_INT_RAW | RMT_CH4_TX_THR_EVENT_INT_RAW | RMT_CH3_TX_THR_EVENT_INT_RAW | RMT_CH2_TX_THR_EVENT_INT_RAW | RMT_CH1_TX_THR_EVENT_INT_RAW | RMT_CH0_TX_THR_EVENT_INT_RAW | RMT_CH7_ERR_INT_RAW | RMT_CH7_RX_END_INT_RAW | RMT_CH7_TX_END_INT_RAW | RMT_CH6_ERR_INT_RAW | RMT_CH6_RX_END_INT_RAW | RMT_CH6_TX_END_INT_RAW | RMT_CH5_ERR_INT_RAW | RMT_CH5_RX_END_INT_RAW | RMT_CH5_TX_END_INT_RAW | RMT_CH4_ERR_INT_RAW | RMT_CH4_RX_END_INT_RAW | RMT_CH4_TX_END_INT_RAW | RMT_CH3_ERR_INT_RAW | RMT_CH3_RX_END_INT_RAW | RMT_CH3_TX_END_INT_RAW | RMT_CH2_ERR_INT_RAW | RMT_CH2_RX_END_INT_RAW | RMT_CH2_TX_END_INT_RAW | RMT_CH1_ERR_INT_RAW | RMT_CH1_RX_END_INT_RAW | RMT_CH1_TX_END_INT_RAW | RMT_CH0_ERR_INT_RAW | RMT_CH0_RX_END_INT_RAW | RMT_CH0_TX_END_INT_RAW | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RMT_CH*n*_TX_THR_EVENT_INT_RAW** The raw interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

**RMT_CH*n*_ERR_INT_RAW** The raw interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

**RMT_CH*n*_RX_END_INT_RAW** The raw interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

**RMT_CH*n*_TX_END_INT_RAW** The raw interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

### Register 30.4. RMT_INT_ST_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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RMT_CH7_TX_THR_EVENT_INT_ST | RMT_CH6_TX_THR_EVENT_INT_ST | RMT_CH5_TX_THR_EVENT_INT_ST | RMT_CH4_TX_THR_EVENT_INT_ST | RMT_CH3_TX_THR_EVENT_INT_ST | RMT_CH2_TX_THR_EVENT_INT_ST | RMT_CH1_TX_THR_EVENT_INT_ST | RMT_CH0_TX_THR_EVENT_INT_ST | RMT_CH7_ERR_INT_ST | RMT_CH7_RX_END_INT_ST | RMT_CH7_TX_END_INT_ST | RMT_CH6_ERR_INT_ST | RMT_CH6_RX_END_INT_ST | RMT_CH6_TX_END_INT_ST | RMT_CH5_ERR_INT_ST | RMT_CH5_RX_END_INT_ST | RMT_CH5_TX_END_INT_ST | RMT_CH4_ERR_INT_ST | RMT_CH4_RX_END_INT_ST | RMT_CH4_TX_END_INT_ST | RMT_CH3_ERR_INT_ST | RMT_CH3_RX_END_INT_ST | RMT_CH3_TX_END_INT_ST | RMT_CH2_ERR_INT_ST | RMT_CH2_RX_END_INT_ST | RMT_CH2_TX_END_INT_ST | RMT_CH1_ERR_INT_ST | RMT_CH1_RX_END_INT_ST | RMT_CH1_TX_END_INT_ST | RMT_CH0_ERR_INT_ST | RMT_CH0_RX_END_INT_ST | RMT_CH0_TX_END_INT_ST | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RMT_CH*n*_TX_THR_EVENT_INT_ST** The masked interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

**RMT_CH*n*_ERR_INT_ST** The masked interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

**RMT_CH*n*_RX_END_INT_ST** The masked interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

**RMT_CH*n*_TX_END_INT_ST** The masked interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

### Register 30.5. RMT_INT_ENA_REG (0x00A8)

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RMT_CH7_TX_THR_EVENT_INT_ENA | RMT_CH6_TX_THR_EVENT_INT_ENA | RMT_CH5_TX_THR_EVENT_INT_ENA | RMT_CH4_TX_THR_EVENT_INT_ENA | RMT_CH3_TX_THR_EVENT_INT_ENA | RMT_CH2_TX_THR_EVENT_INT_ENA | RMT_CH1_TX_THR_EVENT_INT_ENA | RMT_CH0_TX_THR_EVENT_INT_ENA | RMT_CH7_ERR_INT_ENA | RMT_CH7_RX_END_INT_ENA | RMT_CH7_TX_END_INT_ENA | RMT_CH6_ERR_INT_ENA | RMT_CH6_RX_END_INT_ENA | RMT_CH6_TX_END_INT_ENA | RMT_CH5_ERR_INT_ENA | RMT_CH5_RX_END_INT_ENA | RMT_CH5_TX_END_INT_ENA | RMT_CH4_ERR_INT_ENA | RMT_CH4_RX_END_INT_ENA | RMT_CH4_TX_END_INT_ENA | RMT_CH3_ERR_INT_ENA | RMT_CH3_RX_END_INT_ENA | RMT_CH3_TX_END_INT_ENA | RMT_CH2_ERR_INT_ENA | RMT_CH2_RX_END_INT_ENA | RMT_CH2_TX_END_INT_ENA | RMT_CH1_ERR_INT_ENA | RMT_CH1_RX_END_INT_ENA | RMT_CH1_TX_END_INT_ENA | RMT_CH0_ERR_INT_ENA | RMT_CH0_RX_END_INT_ENA | RMT_CH0_TX_END_INT_ENA | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

**RMT_CH*n*_TX_THR_EVENT_INT_ENA** The interrupt enable bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (R/W)

**RMT_CH*n*_ERR_INT_ENA** The interrupt enable bit for the RMT_CH*n*_ERROR_INT interrupt. (R/W)

**RMT_CH*n*_RX_END_INT_ENA** The interrupt enable bit for the RMT_CH*n*_RX_END_INT interrupt. (R/W)

**RMT_CH*n*_TX_END_INT_ENA** The interrupt enable bit for the RMT_CH*n*_TX_END_INT interrupt. (R/W)

### Register 30.6. RMT_INT_CLR_REG (0x00AC)

| 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 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RMT_CH7_TX_THR_EVENT_INT_CLR | RMT_CH6_TX_THR_EVENT_INT_CLR | RMT_CH5_TX_THR_EVENT_INT_CLR | RMT_CH4_TX_THR_EVENT_INT_CLR | RMT_CH3_TX_THR_EVENT_INT_CLR | RMT_CH2_TX_THR_EVENT_INT_CLR | RMT_CH1_TX_THR_EVENT_INT_CLR | RMT_CH0_TX_THR_EVENT_INT_CLR | RMT_CH7_ERR_INT_CLR | RMT_CH7_RX_END_INT_CLR | RMT_CH7_TX_END_INT_CLR | RMT_CH6_ERR_INT_CLR | RMT_CH6_RX_END_INT_CLR | RMT_CH6_TX_END_INT_CLR | RMT_CH5_ERR_INT_CLR | RMT_CH5_RX_END_INT_CLR | RMT_CH5_TX_END_INT_CLR | RMT_CH4_ERR_INT_CLR | RMT_CH4_RX_END_INT_CLR | RMT_CH4_TX_END_INT_CLR | RMT_CH3_ERR_INT_CLR | RMT_CH3_RX_END_INT_CLR | RMT_CH3_TX_END_INT_CLR | RMT_CH2_ERR_INT_CLR | RMT_CH2_RX_END_INT_CLR | RMT_CH2_TX_END_INT_CLR | RMT_CH1_ERR_INT_CLR | RMT_CH1_RX_END_INT_CLR | RMT_CH1_TX_END_INT_CLR | RMT_CH0_ERR_INT_CLR | RMT_CH0_RX_END_INT_CLR | RMT_CH0_TX_END_INT_CLR | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 |

**RMT_CH*n*_TX_THR_EVENT_INT_CLR** Set this bit to clear the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (WO)

**RMT_CH*n*_ERR_INT_CLR** Set this bit to clear the RMT_CH*n*_ERR_INT interrupt. (WO)

**RMT_CH*n*_RX_END_INT_CLR** Set this bit to clear the RMT_CH*n*_RX_END_INT interrupt. (WO)

**RMT_CH*n*_TX_END_INT_CLR** Set this bit to clear the RMT_CH*n*_TX_END_INT interrupt. (WO)

**Register 30.7. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-7) (0x00B0+4**n*)**

| RMT_CARRIER_HIGH_CH*n* (31:16) | RMT_CARRIER_LOW_CH*n* (15:0) | |
|---|---|---|
| 0x00040 | 0x00040 | Reset |

**RMT_CARRIER_HIGH_CH*n*** This field is used to configure the carrier wave's high-level clock period for channel *n*. The clock source can be either REF_TICK or APB_CLK. (R/W)

**RMT_CARRIER_LOW_CH*n*** This field is used to configure the carrier wave's low-level clock period for channel *n*. The clock source can be either REF_TICK or APB_CLK. (R/W)

**Register 30.8. RMT_CH*n*_TX_LIM_REG (*n*: 0-7) (0x00D0+4**n*)**

| (reserved) (31:9) | RMT_TX_LIM_CH*n* (8:0) | |
|---|---|---|
| 0x000000 | 0x080 | Reset |

**RMT_TX_LIM_CH*n*** When channel *n* sends more entries than specified here, it produces a TX_THR_EVENT interrupt. (R/W)

**Register 30.9. RMT_APB_CONF_REG (0x00F0)**

| (reserved) (31:2) | RMT_MEM_TX_WRAP_EN (1) | RMT_MEM_ACCESS_EN (0) | |
|---|---|---|---|
| 0x00000000 | 0 | 0 | Reset |

**RMT_MEM_TX_WRAP_EN** This bit enables wraparound mode: when the transmitter of a channel has reached the end of its memory block, it will resume sending at the start of its memory region. (R/W)

**RMT_MEM_ACCESS_EN** This bit must be 1 in order to access the RMT memory.

# Part VI

# Analog Signal Processing

This part describes components related to analog-to-digital conversion, on-chip sensors, and features such as temperature sensing, demonstrating the system's capabilities in handling analog signals.

# Chapter 31

# On-Chip Sensors and Analog Signal Processing

## 31.1 Introduction

ESP32 has a capacitive touch sensor with up to 10 inputs.

The processing of analog signals is done by two successive approximation ADCs (SAR ADC). There are five controllers dedicated to operating ADCs. This provides flexibility when it comes to converting analog inputs in both high-performance and low-power modes, with minimum processor overhead.

ESP32 is also capable of generating analog signals, using two independent DACs and a cosine waveform generator.

## 31.2 Capacitive Touch Sensor

### 31.2.1 Introduction

A touch-sensor system is built on a substrate which carries electrodes and relevant connections under a protective flat surface; see Figure 31.2-1. When a user touches the surface, the capacitance variation is triggered and a binary signal is generated to indicate whether the touch is valid.

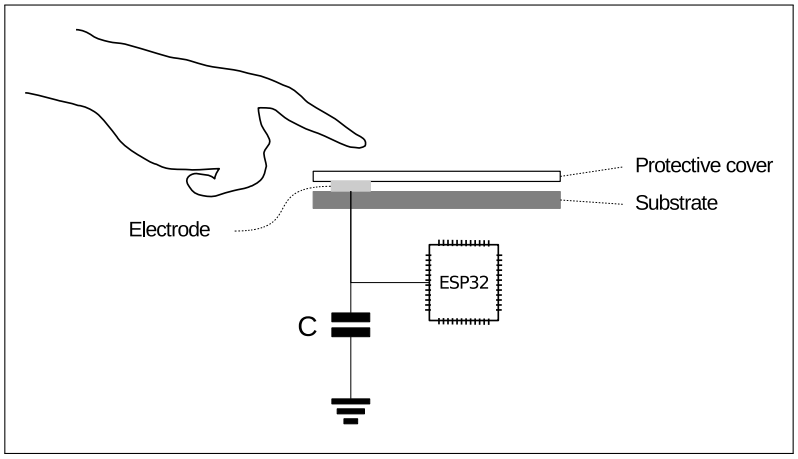



Figure 31.2-1. Touch Sensor

### 31.2.2 Features

- Up to 10 capacitive touch pads / GPIOs
- The sensing pads can be arranged in different combinations, so that a larger area or more points can be detected.

- The touch pad sensing process is under the control of a hardware-implemented finite-state machine (FSM) which is initiated by software or a dedicated hardware timer.
- Information that a pad has been touched can be obtained:
  - by checking touch-sensor registers directly through software,
  - from an interrupt triggered by a touch detection,
  - by waking up the CPU from deep sleep upon touch detection.
- Support for low-power operation in the following scenarios:
  - CPU waiting in deep sleep and saving power until touch detection and subsequent wake up
  - Touch detection managed by the ULP coprocessor
    The user program in ULP coprocessor can trigger a scanning process by checking and writing into specific registers, in order to verify whether the touch threshold is reached.

**Note:**
ESP32 Touch Sensor has not passed the Conducted Susceptibility (CS) test for now, and thus has limited application scenarios.

### 31.2.3 Available GPIOs

All 10 available sensing GPIOs (pads) are listed in Table 31.2-1.

**Table 31.2-1. ESP32 Capacitive Sensing Touch Pads**

| Touch Sensing Signal Name | Pin Name |
|---|---|
| T0 | GPIO4 |
| T1 | GPIO0 |
| T2 | GPIO2 |
| T3 | MTDO |
| T4 | MTCK |
| T5 | MTDI |
| T6 | MTMS |
| T7 | GPIO27 |
| T8 | 32K_XN |
| T9 | 32K_XP |

### 31.2.4 Functional Description

The internal structure of the touch sensor is shown in Figure 31.2-2. The operating flow is shown in Figure 31.2-3.

The capacitance of a touch pad is periodically charged and discharged. The chart "Pad Voltage" shows the charge/discharge voltage that swings from DREFH (reference voltage high) to DREFL (reference voltage low). During each swing, the touch sensor generates an output pulse, shown in the chart as "OUT". The swing slope is different when the pad is touched (high capacitance) and when it is not (low capacitance). By

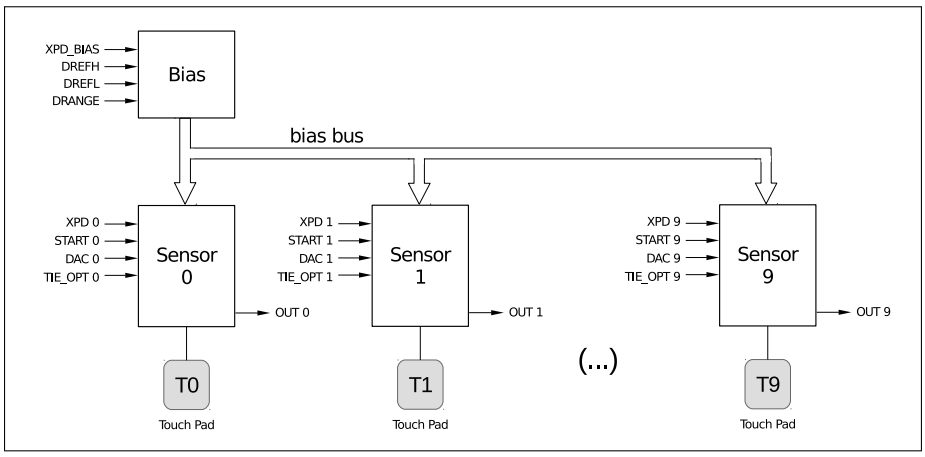



Figure 31.2-2. Touch Sensor Structure

comparing the difference between the output pulse counts during the same time interval, we can conclude whether the touch pad has been touched. TIE_OPT is used to establish the initial voltage level that starts the charge/discharge cycle.

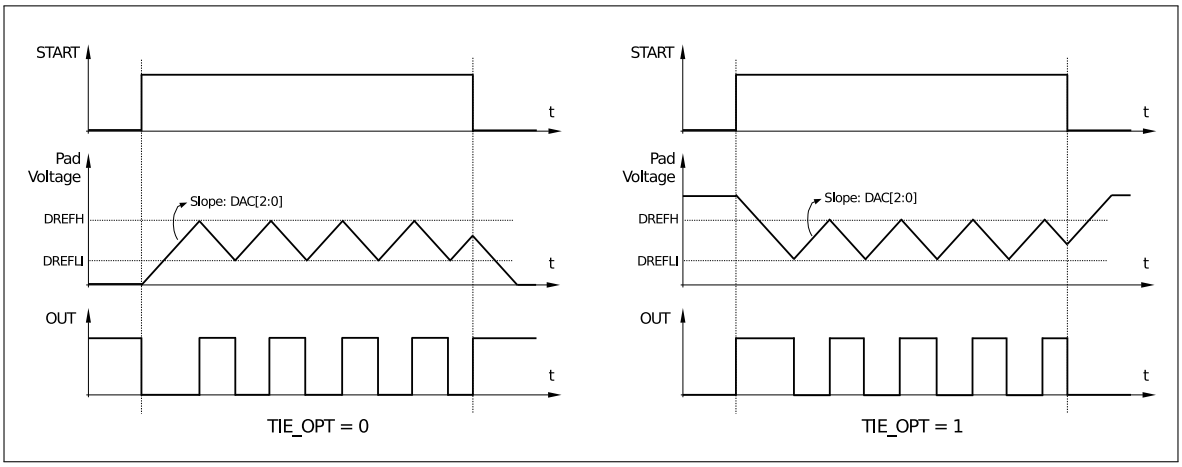



Figure 31.2-3. Touch Sensor Operating Flow

## 31.2.5 Touch FSM

The Touch FSM performs a measurement sequence described in section 31.2.4. Software can operate the Touch FSM through dedicated registers. The internal structure of a touch FSM is shown in Figure 31.2-4.

The functions of Touch FSM include:

- Receipt of a start signal, either from software or a timer
  - when SENS_SAR_TOUCH_START_FORCE=1, SENS_SAR_TOUCH_START_EN is used to initiate a single measurement
  - when SENS_SAR_TOUCH_START_FORCE=0, measurement is triggered periodically with a timer.

  The Touch FSM can be active in sleep mode. The SENS_SAR_TOUCH_SLEEP_CYCLES register can be used to set the cycles. The sensor is operated by RTC_FAST_CLK, which normally runs at 8 MHz. More information on that can be found in chapter Reset and Clock.

- Generation of XPD_TOUCH_BIAS / TOUCH_XPD / TOUCH_START with adjustable timing sequence
  To select enabled pads, TOUCH_XPD / TOUCH_START is masked by the 10-bit register SENS_SAR_TOUCH _PAD_WORKEN.
- Counting of pulses on TOUCH0_OUT ~ TOUCH9_OUT
  The result can be read from SENS_SAR_TOUCH_MEAS_OUT*n*. All ten touch sensors can work simultaneously.
- Generation of a wakeup interrupt
  The FSM regards the touch pads as "touched", if the number of counted pulses is below the threshold. The 10-bit registers SENS_TOUCH_PAD_OUTEN1 & SENS_TOUCH_PAD_OUTEN2 define two sets of touch pads, i.e. SET1 & SET2. If at least one of the pads in SET1 is "touched", the wakeup interrupt will be generated by default. It is also possible to configure the wakeup interrupt to be generated only when pads from both sets are "touched".

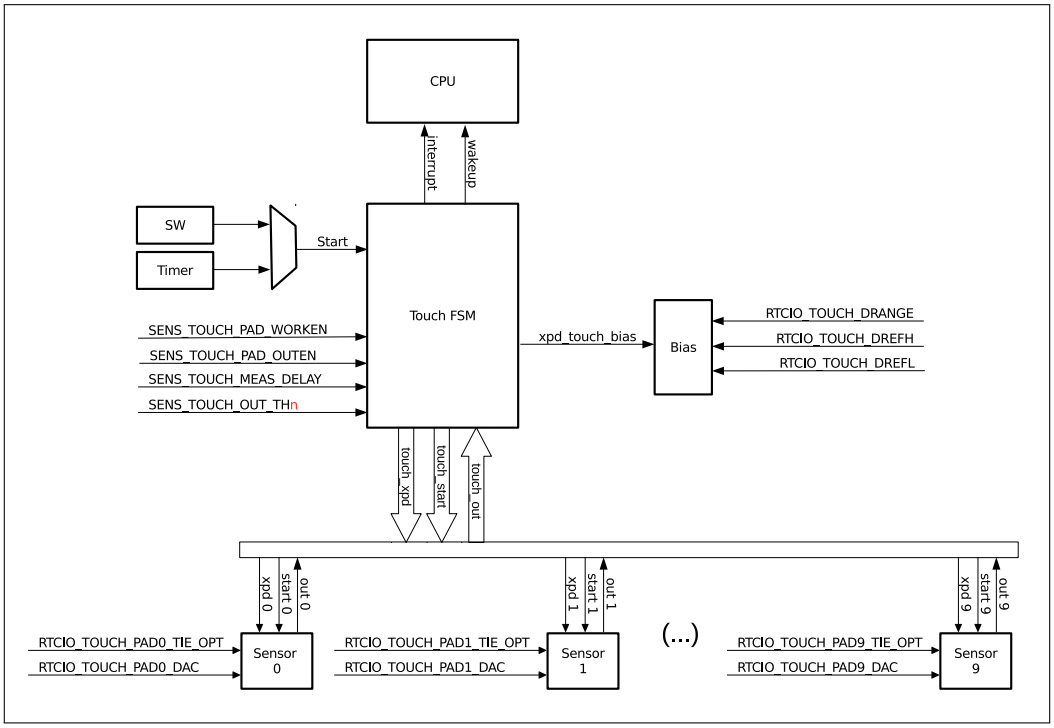



Figure 31.2-4. Touch FSM Structure

# 31.3 SAR ADC

## 31.3.1 Introduction

ESP32 integrates two 12-bit SAR ADCs. They are managed by five SAR ADC controllers, and are able to measure signals from one to 18 analog pads.

The SAR ADC controllers have specialized uses. Two of them support high-performance multiple-channel scanning. Another two are used for low-power operation during Deep-sleep, and the last one is dedicated to PWDET / PKDET (power and peak detection). A diagram of the SAR ADCs is shown in Figure 31.3-1.

**Note:**
PWDET/PKDET controller is for Wi-Fi internal use only. If Wi-Fi module is using the SAR ADC2, users can not measure the analog signal from the pins using SAR ADC2. After SAR ADC2 is released by Wi-Fi, users can use SAR ADC2 normally.

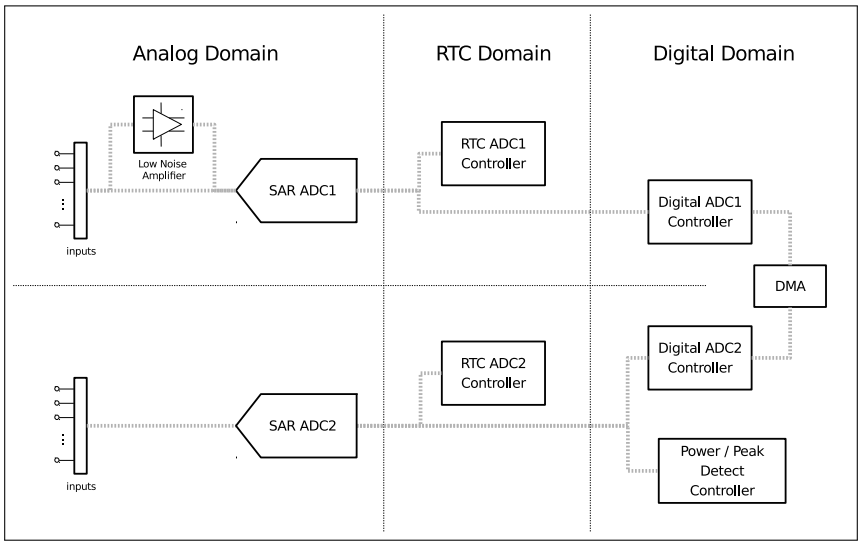



Figure 31.3-1. SAR ADC Depiction

## 31.3.2 Features

- Two SAR ADCs, with simultaneous sampling and conversion
- Up to five SAR ADC controllers for different purposes (e.g. high performance, low power or PWDET / PKDET).
- Up to 18 analog input pads
- 12-bit, 11-bit, 10-bit, 9-bit configurable resolution
- DMA support (available on one controller)
- Multiple channel-scanning modes (available on two controllers)
- Operation during Deep-sleep (available on one controller)
- Controlled by a ULP coprocessor (available on two controllers)

## 31.3.3 Outline of Function

The SAR ADC module's major components, and their interconnections, are shown in Figure 31.3-2.

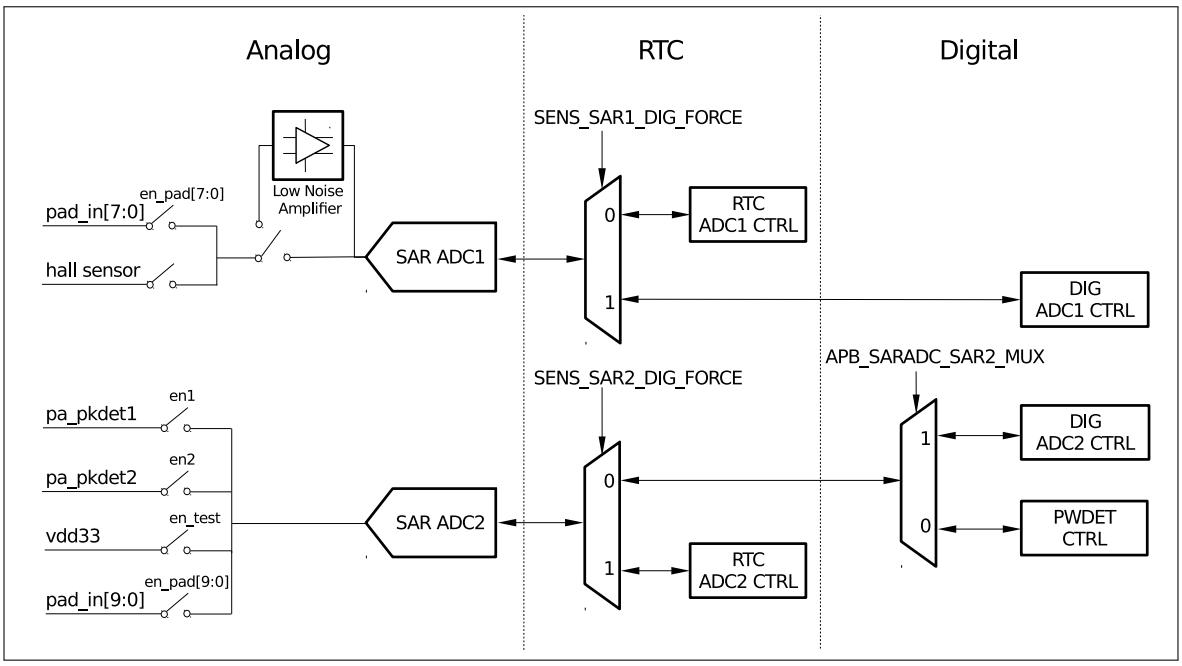



Figure 31.3-2. SAR ADC Outline of Function

Table 31.3-1 lists all the analog signals that may be sent to the SAR ADC module via the ADC channels.

Table 31.3-1. Inputs of SAR ADC

| Signal Name | ADC Channel # | Processed by |
|---|---|---|
| VDET_2 | 7 | SAR ADC1 |
| VDET_1 | 6 | |
| 32K_XN | 5 | |
| 32K_XP | 4 | |
| SENSOR_VN | 3 | |
| SENSOR_CAPN | 2 | |
| SENSOR_CAPP | 1 | |
| SENSOR_VP | 0 | |
| GPIO26 | 9 | SAR ADC2 |
| GPIO25 | 8 | |
| GPIO27 | 7 | |
| MTMS | 6 | |
| MTDI | 5 | |
| MTCK | 4 | |
| MTDO | 3 | |
| GPIO2 | 2 | |
| GPIO0 | 1 | |
| GPIO4 | 0 | |

**Note:**

- Some of the SAR ADC2 pins are used as strapping pins (GPIO0, GPIO2, and GPIO15), thus can not be used freely.

There are five ADC controllers in ESP32: RTC ADC1 CTRL, RTC ADC2 CTRL, DIG ADC1 CTRL, DIG ADC2 CTRL and PWDET CTRL. The differences between them are summarized in Table 31.3-2.

**Table 31.3-2. ESP32 SAR ADC Controllers**

| | RTC ADC1 | RTC ADC2 | DIG ADC1 | DIG ADC2 | PWDET |
|---|---|---|---|---|---|
| DAC | Y | - | - | - | - |
| Support deep sleep | Y | Y | - | - | - |
| ULP coprocessor | Y | Y | - | - | - |
| PWDET/PKDET | - | - | - | - | Y |
| DMA | - | - | Y | - | - |

## 31.3.4 RTC SAR ADC Controllers

The purpose of SAR ADC controllers in the RTC power domain – RTC ADC1 CTRL and RTC ADC2 CTRL – is to provide ADC measurement with minimal power consumption in a low frequency.

The outline of a single controller's function is shown in Figure 31.3-3. For each controller, the start of analog-to-digital conversion can be triggered by register SENS_SAR_MEASn_START_SAR. The measurement's result can be obtained from register SENS_SAR_MEASn_DATA_SAR.

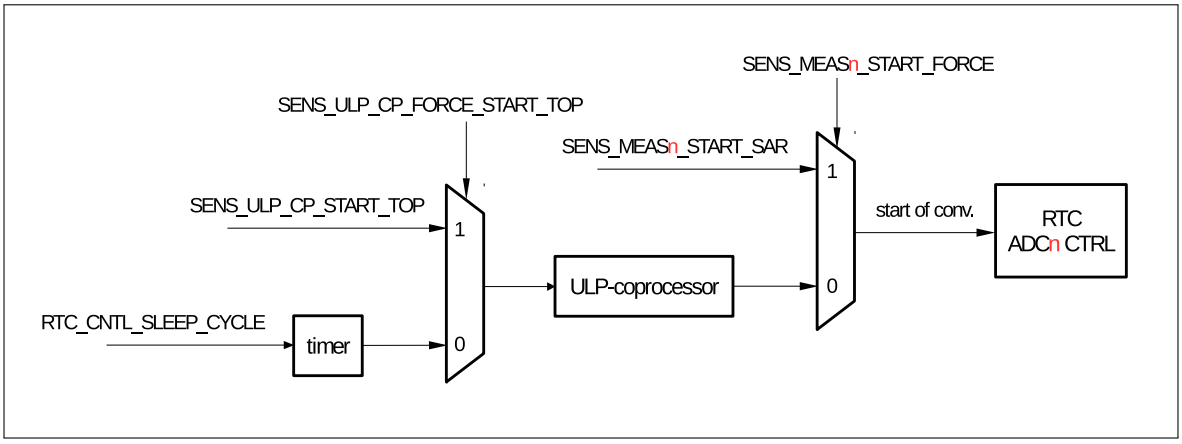



**Figure 31.3-3. RTC SAR ADC Outline of Function**

The controllers are intertwined with the ULP coprocessor, as the ULP coprocessor has a built-in instruction to start an ADC measurement. In many cases, the controllers need to cooperate with the ULP coprocessor, e.g.:

- when periodically monitoring a channel during deep sleep, where the ULP coprocessor is the only trigger source during this mode;
- when scanning channels continuously in a sequence. Continuous scanning or DMA is not supported by the controllers. However, it is possible with the help of the ULP coprocessor.

## 31.3.5 DIG SAR ADC Controllers

Compared to RTC SAR ADC controllers, DIG SAR ADC controllers have optimized performance and throughput. Some of their features are:

- High performance; the clock is much faster, therefore, the sample rate is highly increased.

- Multiple-channel scanning mode; there is a pattern table that defines the measurement rule for each SAR ADC. The scanning mode can be configured as a single mode, double mode, or alternate mode.
- The scanning can be started by software or I2S.
- DMA support; an interrupt will be generated when scanning is finished.

**Note:**
We do not use the term "start of conversion" in this section, because there is no direct access to starting a single SAR analog-to-digital conversion. We use "start of scan" instead, which implies that we expect to scan a sequence of channels with DIG ADC controllers.

Figure 31.3-4 shows a diagram of DIG SAR ADC controllers.

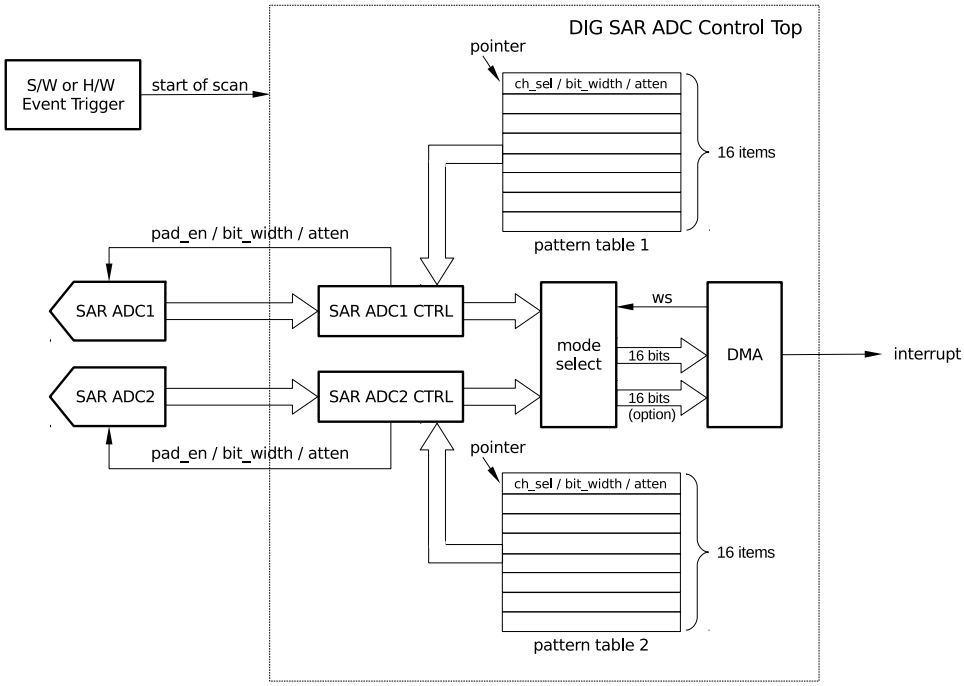



**Figure 31.3-4. Diagram of DIG SAR ADC Controllers**

The pattern tables contain the measurement rules mentioned above. Each table has 16 items which store information on channel selection, resolution and attenuation. When scanning starts, the controller reads measurement rules one-by-one from a pattern table. For each controller the scanning sequence includes 16 different rules at most, before repeating itself.

The 8-bit item (the pattern table register) is composed of three fields that contain channel, resolution and attenuation information, as shown in Table 31.3-3.

**Table 31.3-3. Fields of the Pattern Table Register**

| Pattern Table Register [7:0] | | |
|---|---|---|
| ch_sel[3:0] | bit_width[1:0] | atten[1:0] |
| channel to be scanned | resolution | attenuation |

There are three scanning modes: single mode, double mode and alternate mode.

- Single mode: channels of either SAR ADC1 or SAR ADC2 will be scanned.
- Double mode: channels of SAR ADC1 and SAR ADC2 will be scanned simultaneously.
- Alternate mode: channels of SAR ADC1 and SAR ADC2 will be scanned alternately.

ESP32 supports up to a 12-bit SAR ADC resolution. The 16-bit data in DMA is composed of the ADC result and some necessary information related to the scanning mode:

- For single mode, only 4-bit information on channel selection is added.
- For double mode or alternate mode, 4-bit information on channel selection is added plus one extra bit indicating which SAR ADC was selected.

For each scanning mode there is a corresponding data format, called Type I and Type II. Both data formats are described in Tables 31.3-4 and 31.3-5.

**Table 31.3-4. Fields of Type I DMA Data Format**

| **Type I** DMA Data Format [15:0] | |
|---|---|
| ch_sel[3:0] | data[11:0] |
| channel | SAR ADC data |

**Table 31.3-5. Fields of Type II DMA Data Format**

| **Type II** DMA Data Format [15:0] | | |
|---|---|---|
| sar_sel | ch_sel[3:0] | SAR ADC data[10:0] |
| SAR ADCn | channel | SAR ADC data |

For Type I the resolution of SAR ADC is up to 12 bits, while for Type II the resolution is 11 bits at most.

DIG SAR ADC Controllers allow the use of I2S for direct memory access. The WS signal of I2S acts as a measurement-trigger signal. The DATA signal provides the information that the measurement result is ready. Software can configure APB_SARADC_DATA_TO_I2S, in order to connect ADC to I2S.

# 31.4 DAC

## 31.4.1 Introduction

Two 8-bit DAC channels can be used to convert digital values into analog output signals (up to two of them). The design structure is composed of integrated resistor strings and a buffer. This dual DAC supports power supply and uses it as input voltage reference. The dual DAC also supports independent or simultaneous signal conversions inside of its channels.

## 31.4.2 Features

The features of DAC are as follows:

- Two 8-bit DAC channels

- Independent or simultaneous conversion in channels
- Voltage reference from the VDD3P3_RTC pin
- Cosine waveform (CW) generator
- DMA capability
- Start of conversion can be triggered by software or SAR ADC FSM (please refer to the SAR ADC chapter for more details)
- Can be fully controlled by the ULP coprocessor

A diagram showing the DAC channel's function is presented in Figure 31.4-1. For a detailed description, see the sections below.

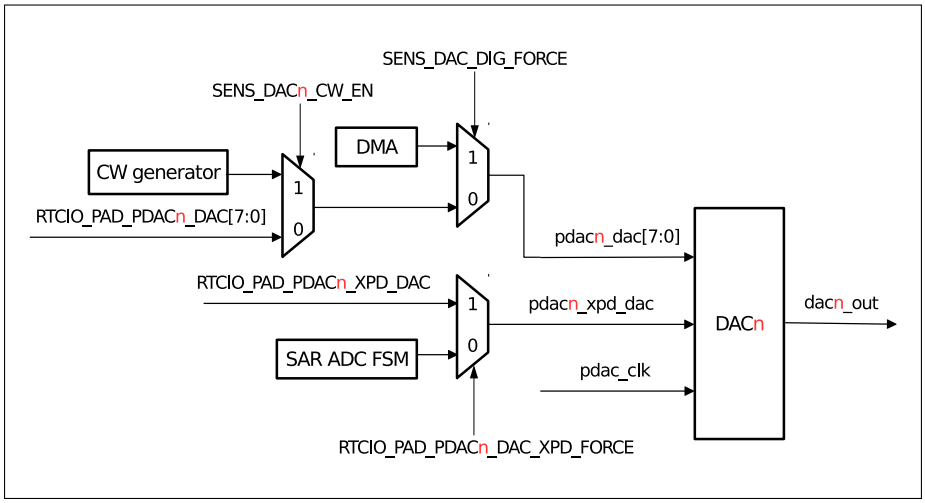



Figure 31.4-1. Diagram of DAC Function

## 31.4.3 Structure

The two 8-bit DAC channels can be configured independently. For each DAC channel, the output analog voltage can be calculated as follows:

$$\text{DAC}n\text{_OUT} = \text{VDD3P3_RTC} \cdot \text{PDAC}n\text{_DAC}/255$$

- VDD3P3_RTC is the voltage on pin VDD3P3_RTC (typically 3.3V).
- PDAC*n*_DAC has multiple sources: CW generator, register RTCIO_PAD_DAC*n*_REG, and DMA.

The start of conversion is determined by register RTCIO_PAD_PDAC*n*_XPD_DAC. The conversion process itself is controlled by software or SAR ADC FSM; see Figure 31.4-1.

## 31.4.4 Cosine Waveform Generator

The cosine waveform (CW) generator can be used to generate a cosine / sine tone. A diagram showing cosine waveform generator's function is presented in Figure 31.4-2.

The CW generator has the following features:

- Adjustable frequency

The frequency of CW can be adjusted by register SENS_SAR_SW_FSTEP[15:0]:

$$\text{freq} = \text{dig_clk_rtc_freq} \cdot \text{SENS_SAR_SW_FSTEP}/65536$$

The frequency of dig_clk_rtc is typically 8 MHz.

- Scaling
  Configuring register SENS_SAR_DAC_SCALE*n*[1:0]; the amplitude of a CW can be multiplied by 1, 1/2, 1/4 or 1/8.
- DC offset
  The offset may be introduced by register SENS_SAR_DAC_DC*n*[7:0]. The result will be saturated.
- Phase shift
  A phase-shift of 0 / 90 / 180 / 270 degrees can be added by setting register SENS_SAR_DAC_INV*n*[1:0].

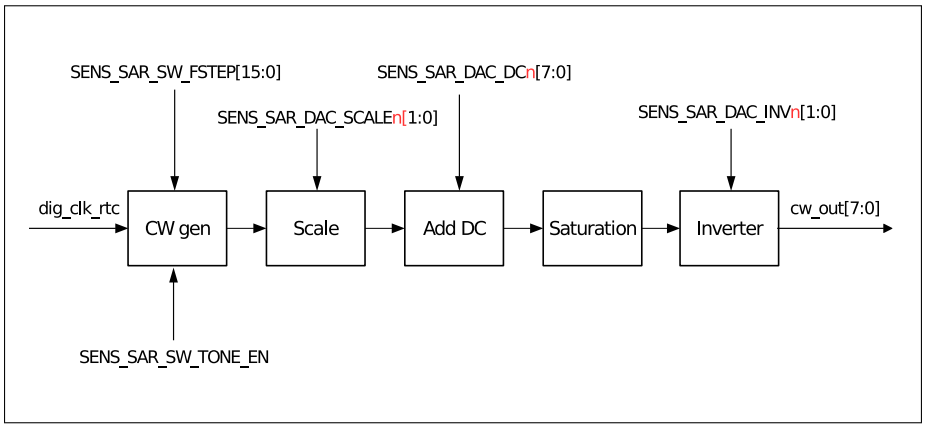



Figure 31.4-2. Cosine Waveform (CW) Generator

### 31.4.5 DMA support

A DMA controller with dual DMA channels can be used to set the output of two DAC channels. By configuring SENS_SAR_DAC_DIG_FORCE, I2S_clk can be connected to DAC clk, and I2S_DATA_OUT can be connected to DAC_DATA for direct memory access.

For details, please refer to chapter DMA.

## 31.5 Register Summary

Note: The registers listed below have been grouped, according to their functionality. This particular grouping does not reflect the exact sequential order of their place in memory.

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

### 31.5.1 Sensors

| Name | Description | Address | Access |
|---|---|---|---|
| **Touch pad setup and control registers** | | | |
| SENS_SAR_TOUCH_CTRL1_REG | Touch pad control | 0x3FF48858 | R/W |

| | | | |
|---|---|---|---|
| SENS_SAR_TOUCH_CTRL2_REG | Touch pad control and status | 0x3FF48884 | RO |
| SENS_SAR_TOUCH_ENABLE_REG | Wakeup interrupt control and working set | 0x3FF4888C | R/W |
| SENS_SAR_TOUCH_THRES1_REG | Threshold setup for pads 0 and 1 | 0x3FF4885C | R/W |
| SENS_SAR_TOUCH_THRES2_REG | Threshold setup for pads 2 and 3 | 0x3FF48860 | R/W |
| SENS_SAR_TOUCH_THRES3_REG | Threshold setup for pads 4 and 5 | 0x3FF48864 | R/W |
| SENS_SAR_TOUCH_THRES4_REG | Threshold setup for pads 6 and 7 | 0x3FF48868 | R/W |
| SENS_SAR_TOUCH_THRES5_REG | Threshold setup for pads 8 and 9 | 0x3FF4886C | R/W |
| SENS_SAR_TOUCH_OUT1_REG | Counters for pads 0 and 1 | 0x3FF48870 | RO |
| SENS_SAR_TOUCH_OUT2_REG | Counters for pads 2 and 3 | 0x3FF48874 | RO |
| SENS_SAR_TOUCH_OUT3_REG | Counters for pads 4 and 5 | 0x3FF48878 | RO |
| SENS_SAR_TOUCH_OUT4_REG | Counters for pads 6 and 6 | 0x3FF4887C | RO |
| SENS_SAR_TOUCH_OUT5_REG | Counters for pads 8 and 9 | 0x3FF48880 | RO |
| **SAR ADC control register** | | | |
| SENS_SAR_START_FORCE_REG | SAR ADC1 and ADC2 control | 0x3FF4882C | R/W |
| **SAR ADC1 control registers** | | | |
| SENS_SAR_READ_CTRL_REG | SAR ADC1 data and sampling control | 0x3FF48800 | R/W |
| SENS_SAR_MEAS_START1_REG | SAR ADC1 conversion control and status | 0x3FF48854 | RO |
| **SAR ADC2 control registers** | | | |
| SENS_SAR_READ_CTRL2_REG | SAR ADC2 data and sampling control | 0x3FF48890 | R/W |
| SENS_SAR_MEAS_START2_REG | SAR ADC2 conversion control and status | 0x3FF48894 | RO |
| **ULP coprocessor configuration register** | | | |
| SENS_ULP_CP_SLEEP_CYC0_REG | Sleep cycles for ULP coprocessor | 0x3FF48818 | R/W |
| **Pad attenuation configuration registers** | | | |
| SENS_SAR_ATTEN1_REG | 2-bit attenuation for each pad | 0x3FF48834 | R/W |
| SENS_SAR_ATTEN2_REG | 2-bit attenuation for each pad | 0x3FF48838 | R/W |
| **DAC control registers** | | | |
| SENS_SAR_DAC_CTRL1_REG | DAC control | 0x3FF48898 | R/W |
| SENS_SAR_DAC_CTRL2_REG | DAC output control | 0x3FF4889C | R/W |

## 31.5.2 Advanced Peripheral Bus

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

| Name | Description | Address | Access |
|---|---|---|---|
| **SAR ADC1 and ADC2 common configuration registers** | | | |
| APB_SARADC_CTRL_REG | SAR ADC common configuration | 0x60002610 | R/W |
| APB_SARADC_CTRL2_REG | SAR ADC common configuration | 0x60002614 | R/W |
| APB_SARADC_FSM_REG | SAR ADC FSM sample cycles configuration | 0x60002618 | R/W |
| **SAR ADC1 pattern table registers** | | | |
| APB_SARADC_SAR1_PATT_TAB1_REG | Items 0 - 3 of pattern table | 0x6000261C | R/W |
| APB_SARADC_SAR1_PATT_TAB2_REG | Items 4 - 7 of pattern table | 0x60002620 | R/W |
| APB_SARADC_SAR1_PATT_TAB3_REG | Items 8 - 11 of pattern table | 0x60002624 | R/W |
| APB_SARADC_SAR1_PATT_TAB4_REG | Items 12 - 15 of pattern table | 0x60002628 | R/W |
| **SAR ADC2 pattern table registers** | | | |

| | | | |
|---|---|---|---|
| APB_SARADC_SAR2_PATT_TAB1_REG | Items 0 - 3 of pattern table | 0x6000262C | R/W |
| APB_SARADC_SAR2_PATT_TAB2_REG | Items 4 - 7 of pattern table | 0x60002630 | R/W |
| APB_SARADC_SAR2_PATT_TAB3_REG | Items 8 - 11 of pattern table | 0x60002634 | R/W |
| APB_SARADC_SAR2_PATT_TAB4_REG | Items 12 - 15 of pattern table | 0x60002638 | R/W |

### 31.5.3 RTC I/O

For details, please refer to Section Register Summary in Chapter IO_MUX and GPIO Matrix.

# 31.6 Registers

## 31.6.1 Sensors

The addresses in this section are relative to (the RTC base address + 0x0800) provided in Table 3.3-6 in Chapter 3 *System and Memory*. The absolute register addresses are listed in Section 31.5.1 *Sensors*.

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

**Register 31.1. SENS_SAR_READ_CTRL_REG (0x0000)**

| (reserved) | SENS_SAR1_DATA_INV | SENS_SAR1_DIG_FORCE | (reserved) | SENS_SAR1_SAMPLE_BIT | SENS_SAR1_SAMPLE_CYCLE | SENS_SAR1_CLK_DIV | |
|---|---|---|---|---|---|---|---|
| 31 29 | 28 | 27 | 26 18 | 17 16 | 15 8 | 7 0 | |
| 0 0 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 | 3 | 9 | 2 | Reset |

**SENS_SAR1_DATA_INV** Invert SAR ADC1 data. (R/W)

**SENS_SAR1_DIG_FORCE** 1: SAR ADC1 controlled by DIG ADC1 CTR, 0: SAR ADC1 controlled by RTC ADC1 CTRL. (R/W)

**SENS_SAR1_SAMPLE_BIT** Bit width of SAR ADC1, 00: for 9-bit, 01: for 10-bit, 10: for 11-bit, 11: for 12-bit. (R/W)

**SENS_SAR1_SAMPLE_CYCLE** Sample cycles for SAR ADC1. (R/W)

**SENS_SAR1_CLK_DIV** Clock divider. (R/W)

**Register 31.2. SENS_ULP_CP_SLEEP_CYC0_REG (0x0018)**

| 31 0 | |
|---|---|
| 200 | Reset |

**SENS_ULP_CP_SLEEP_CYC0_REG** Sleep cycles for ULP coprocessor timer. (R/W)

**Register 31.3. SENS_SAR_START_FORCE_REG (0x002c)**

| (reserved) | SENS_SAR1_STOP | SENS_SAR2_STOP | SENS_PC_INIT | (reserved) | SENS_ULP_CP_START_TOP | SENS_ULP_CP_FORCE_START_TOP | SENS_SAR2_PWDET_CCT | SENS_SAR2_EN_TEST | SENS_SAR2_BIT_WIDTH | SENS_SAR1_BIT_WIDTH | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 24 | 23 | 22 | 21 … 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 0 | 0 | 1 1 | 1 1 | Reset |

**SENS_SAR1_STOP** Stop SAR ADC1 conversion. (R/W)

**SENS_SAR2_STOP** Stop SAR ADC2 conversion. (R/W)

**SENS_PC_INIT** Initialized PC for ULP coprocessor. (R/W)

**SENS_ULP_CP_START_TOP** Write 1 to start ULP coprocessor; it is active only when reg_ulp_cp_force_start_top = 1. (R/W)

**SENS_ULP_CP_FORCE_START_TOP** 1: ULP coprocessor is started by SW, 0: ULP coprocessor is started by timer. (R/W)

**SENS_SAR2_PWDET_CCT** SAR2_PWDET_CCT, PA power detector capacitance tuning. (R/W)

**SENS_SAR2_EN_TEST** SAR2_EN_TEST is active only when reg_sar2_dig_force = 0. (R/W)

**SENS_SAR2_BIT_WIDTH** Bit width of SAR ADC2, 00: 9 bits, 01: 10 bits, 10: 11 bits, 11: 12 bits. (R/W)

**SENS_SAR1_BIT_WIDTH** Bit width of SAR ADC1, 00: 9 bits, 01: 10 bits, 10: 11 bits, 11: 12 bits. (R/W)

**Register 31.4. SENS_SAR_ATTEN1_REG (0x0034)**

| 31 … 0 | |
|---|---|
| 0xFFFFFFFF | Reset |

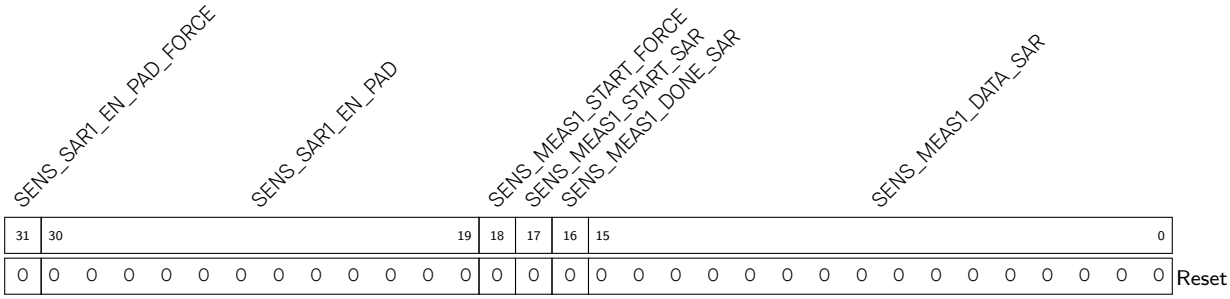
**SENS_SAR_ATTEN1_REG** 2-bit attenuation for each pad, 11: 1 dB, 10: 6 dB, 01: 3 dB, 00: 0 dB, [1:0] is used for ADC1_CH0, [3:2] is used for ADC1_CH1, etc. (R/W)

**Register 31.5. SENS_SAR_ATTEN2_REG (0x0038)**

| 31 … 0 | |
|---|---|
| 0xFFFFFFFF | Reset |

**SENS_SAR_ATTEN2_REG** 2-bit attenuation for each pad, 11: 1 dB, 10: 6 dB, 01: 3 dB, 00: 0 dB, [1:0] is used for ADC2_CH0, [3:2] is used for ADC2_CH1, etc (R/W)

**Register 31.6. SENS_SAR_MEAS_START1_REG (0x0054)**




**SENS_SAR1_EN_PAD_FORCE** 1: SAR ADC1 pad enable bitmap is controlled by SW, 0: SAR ADC1 pad enable bitmap is controlled by ULP coprocessor. (R/W)

**SENS_SAR1_EN_PAD** SAR ADC1 pad enable bitmap; active only when reg_sar1_en_pad_force = 1. (R/W)

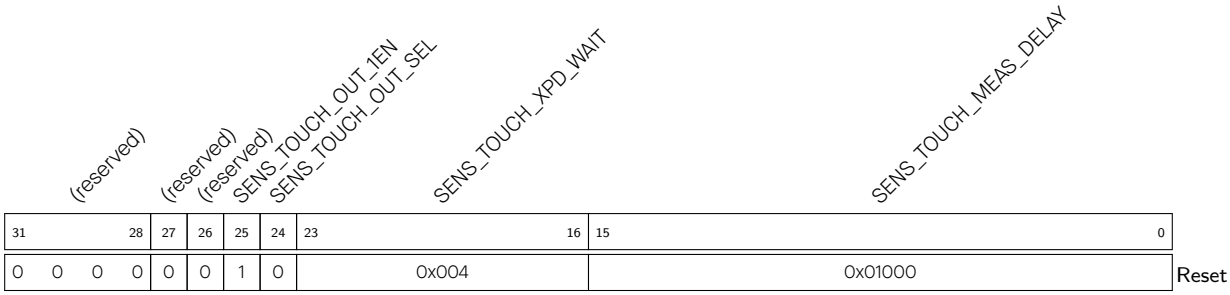
**SENS_MEAS1_START_FORCE** 1: SAR ADC1 controller (in RTC) is started by SW, 0: SAR ADC1 controller is started by ULP coprocessor. (R/W)

**SENS_MEAS1_START_SAR** SAR ADC1 controller (in RTC) starts conversion; active only when reg_meas1_start_force = 1. (R/W)

**SENS_MEAS1_DONE_SAR** SAR ADC1 conversion-done indication. (RO)

**SENS_MEAS1_DATA_SAR** SAR ADC1 data. (RO)

**Register 31.7. SENS_SAR_TOUCH_CTRL1_REG (0x0058)**




**SENS_TOUCH_OUT_1EN** 1: wakeup interrupt is generated if SET1 is touched, 0: wakeup interrupt is generated only if both SET1 & SET2 are touched. (R/W)

**SENS_TOUCH_OUT_SEL** 1: the touch pad is considered touched when the value of the counter is greater than the threshold, 0: the touch pad is considered touched when the value of the counter is less than the threshold. (R/W)

**SENS_TOUCH_XPD_WAIT** The waiting time (in 8 MHz cycles) between TOUCH_START and TOUCH_XPD. (R/W)

**SENS_TOUCH_MEAS_DELAY** The measurement's duration (in 8 MHz cycles). (R/W)

### Register 31.8. SENS_SAR_TOUCH_THRES1_REG (0x005c)

| SENS_TOUCH_OUT_TH0 | SENS_TOUCH_OUT_TH1 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_OUT_TH0** The threshold for touch pad 0. (R/W)

**SENS_TOUCH_OUT_TH1** The threshold for touch pad 1. (R/W)

### Register 31.9. SENS_SAR_TOUCH_THRES2_REG (0x0060)

| SENS_TOUCH_OUT_TH2 | SENS_TOUCH_OUT_TH3 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_OUT_TH2** The threshold for touch pad 2. (R/W)

**SENS_TOUCH_OUT_TH3** The threshold for touch pad 3. (R/W)

### Register 31.10. SENS_SAR_TOUCH_THRES3_REG (0x0064)

| SENS_TOUCH_OUT_TH4 | SENS_TOUCH_OUT_TH5 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_OUT_TH4** The threshold for touch pad 4. (R/W)

**SENS_TOUCH_OUT_TH5** The threshold for touch pad 5. (R/W)

### Register 31.11. SENS_SAR_TOUCH_THRES4_REG (0x0068)

| SENS_TOUCH_OUT_TH6 | SENS_TOUCH_OUT_TH7 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_OUT_TH6** The threshold for touch pad 6. (R/W)

**SENS_TOUCH_OUT_TH7** The threshold for touch pad 7. (R/W)

### Register 31.12. SENS_SAR_TOUCH_THRES5_REG (0x006c)

| SENS_TOUCH_OUT_TH8 | SENS_TOUCH_OUT_TH9 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_OUT_TH8** The threshold for touch pad 8. (R/W)

**SENS_TOUCH_OUT_TH9** The threshold for touch pad 9. (R/W)

### Register 31.13. SENS_SAR_TOUCH_OUT1_REG (0x0070)

| SENS_TOUCH_MEAS_OUT0 | SENS_TOUCH_MEAS_OUT1 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_MEAS_OUT0** The counter for touch pad 0. (RO)

**SENS_TOUCH_MEAS_OUT1** The counter for touch pad 1. (RO)

**Register 31.14. SENS_SAR_TOUCH_OUT2_REG (0x0074)**

| SENS_TOUCH_MEAS_OUT2 | SENS_TOUCH_MEAS_OUT3 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_MEAS_OUT2** The counter for touch pad 2. (RO)

**SENS_TOUCH_MEAS_OUT3** The counter for touch pad 3. (RO)

**Register 31.15. SENS_SAR_TOUCH_OUT3_REG (0x0078)**

| SENS_TOUCH_MEAS_OUT4 | SENS_TOUCH_MEAS_OUT5 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_MEAS_OUT4** The counter for touch pad 4. (RO)

**SENS_TOUCH_MEAS_OUT5** The counter for touch pad 5. (RO)

**Register 31.16. SENS_SAR_TOUCH_OUT4_REG (0x007c)**

| SENS_TOUCH_MEAS_OUT6 | SENS_TOUCH_MEAS_OUT7 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_MEAS_OUT6** The counter for touch pad 6. (RO)

**SENS_TOUCH_MEAS_OUT7** The counter for touch pad 7. (RO)

**Register 31.17. SENS_SAR_TOUCH_OUT5_REG (0x0080)**

| SENS_TOUCH_MEAS_OUT8 | SENS_TOUCH_MEAS_OUT9 | |
|---|---|---|
| 31 … 16 | 15 … 0 | |
| 0x00000 | 0x00000 | Reset |

**SENS_TOUCH_MEAS_OUT8** The counter for touch pad 8. (RO)

**SENS_TOUCH_MEAS_OUT9** The counter for touch pad 9. (RO)

**Register 31.18. SENS_SAR_TOUCH_CTRL2_REG (0x0084)**

| (reserved) | SENS_TOUCH_MEAS_EN_CLR | SENS_TOUCH_SLEEP_CYCLES | SENS_TOUCH_START_FORCE | SENS_TOUCH_START_EN | SENS_TOUCH_START_FSM_EN | SENS_TOUCH_MEAS_DONE | SENS_TOUCH_MEAS_EN | |
|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 … 14 | 13 | 12 | 11 | 10 | 9 … 0 | |
| 0 | 0 | 0x00100 | 0 | 0 | 1 | 0 | 0x000 | Reset |

**SENS_TOUCH_MEAS_EN_CLR** Set to clear reg_touch_meas_en. (WO)

**SENS_TOUCH_SLEEP_CYCLES** Sleep cycles for timer. (R/W)

**SENS_TOUCH_START_FORCE** 1: starts the Touch FSM via software; 0: starts the Touch FSM via timer. (R/W)

**SENS_TOUCH_START_EN** 1: starts the Touch FSM; this is valid when reg_touch_start_force is set. (R/W)

**SENS_TOUCH_START_FSM_EN** 1: TOUCH_START & TOUCH_XPD are controlled by the Touch FSM; 0: TOUCH_START & TOUCH_XPD are controlled by registers. (R/W)

**SENS_TOUCH_MEAS_DONE** Set to 1 by FSM, indicating that touch measurement is done. (RO)

**SENS_TOUCH_MEAS_EN** 10-bit register indicating which pads are touched. (RO)

**Register 31.19. SENS_SAR_TOUCH_ENABLE_REG (0x008c)**

| (reserved) | SENS_TOUCH_PAD_OUTEN1 | SENS_TOUCH_PAD_OUTEN2 | SENS_TOUCH_PAD_WORKEN | |
|---|---|---|---|---|
| 31 30 | 29 20 | 19 10 | 9 0 | |
| 0 0 | 0x3FF | 0x3FF | 0x3FF | Reset |

**SENS_TOUCH_PAD_OUTEN1** Bitmap defining SET1 for generating a wakeup interrupt; SET1 is considered touched if at least one of the touch pads in SET1 is touched. (R/W)

**SENS_TOUCH_PAD_OUTEN2** Bitmap defining SET2 for generating a wakeup interrupt; SET2 is considered touched if at least one of the touch pads in SET2 is touched. (R/W)

**SENS_TOUCH_PAD_WORKEN** Bitmap defining the working set during measurement. (R/W)

**Register 31.20. SENS_SAR_READ_CTRL2_REG (0x0090)**

| (reserved) | SENS_SAR2_DATA_INV | SENS_SAR2_DIG_FORCE | (reserved) | SENS_SAR2_SAMPLE_BIT | SENS_SAR2_SAMPLE_CYCLE | SENS_SAR2_CLK_DIV | |
|---|---|---|---|---|---|---|---|
| 31 30 | 29 | 28 | 27 18 | 17 16 | 15 8 | 7 0 | |
| 0 0 | 0 | 0 | 0 0 0 0 0 0 0 0 0 0 | 3 | 9 | 2 | Reset |

**SENS_SAR2_DATA_INV** Invert SAR ADC2 data. (R/W)

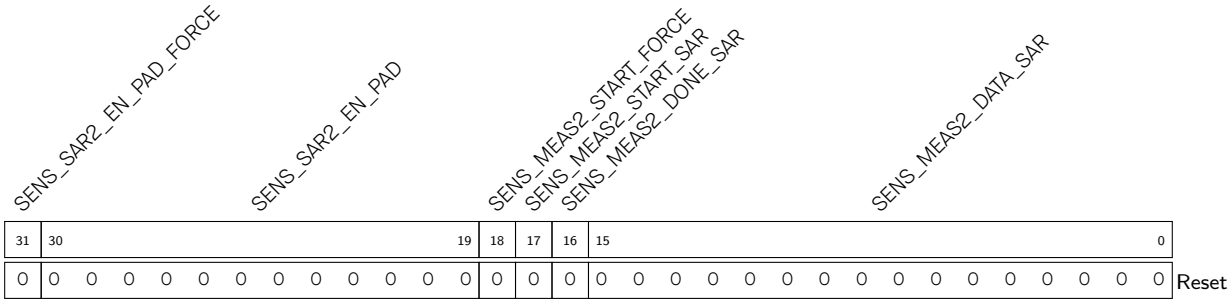
**SENS_SAR2_DIG_FORCE** 1: SAR ADC2 controlled by DIG ADC2 CTRL or PWDET CTRL, 0: SAR ADC2 controlled by RTC ADC2 CTRL (R/W)

**SENS_SAR2_SAMPLE_BIT** Bit width of SAR ADC2, 00: for 9-bit, 01: for 10-bit, 10: for 11-bit, 11: for 12-bit. (R/W)

**SENS_SAR2_SAMPLE_CYCLE** Sample cycles of SAR ADC2. (R/W)

**SENS_SAR2_CLK_DIV** Clock divider. (R/W)

**Register 31.21. SENS_SAR_MEAS_START2_REG (0x0094)**




**SENS_SAR2_EN_PAD_FORCE** 1: SAR ADC2 pad enable bitmap is controlled by SW, 0: SAR ADC2 pad enable bitmap is controlled by ULP coprocessor. (R/W)

**SENS_SAR2_EN_PAD** SAR ADC2 pad enable bitmap; active only when reg_sar2_en_pad_force = 1. (R/W)

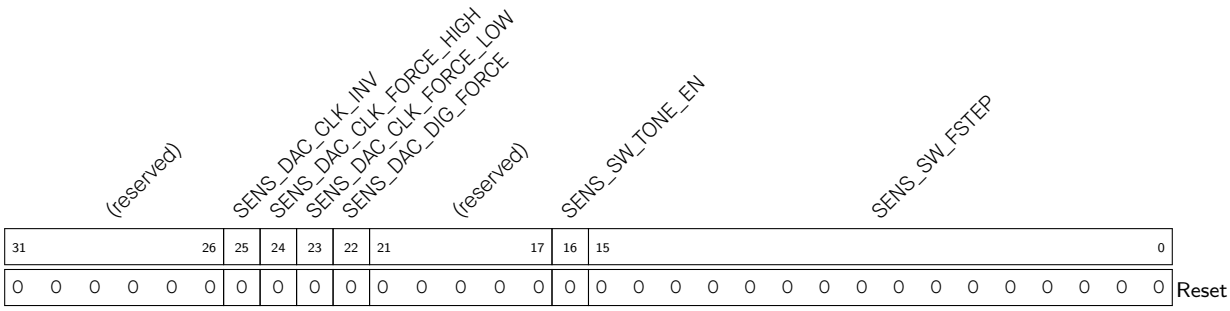
**SENS_MEAS2_START_FORCE** 1: SAR ADC2 controller (in RTC) is started by SW, 0: SAR ADC2 controller is started by ULP coprocessor. (R/W)

**SENS_MEAS2_START_SAR** SAR ADC2 controller (in RTC) starts conversion; active only when reg_meas2_start_force = 1. (R/W)

**SENS_MEAS2_DONE_SAR** SAR ADC2-conversion-done indication. (RO)

**SENS_MEAS2_DATA_SAR** SAR ADC2 data. (RO)

**Register 31.22. SENS_SAR_DAC_CTRL1_REG (0x0098)**




**SENS_DAC_CLK_INV** 1: inverts PDAC_CLK, 0: no inversion. (R/W)

**SENS_DAC_CLK_FORCE_HIGH** forces PDAC_CLK to be 1. (R/W)

**SENS_DAC_CLK_FORCE_LOW** forces PDAC_CLK to be 0. (R/W)

**SENS_DAC_DIG_FORCE** 1: DAC1 & DAC2 use DMA, 0: DAC1 & DAC2 do not use DMA. (R/W)

**SENS_SW_TONE_EN** 1: enable CW generator, 0: disable CW generator. (R/W)

**SENS_SW_FSTEP** Frequency step for CW generator; can be used to adjust the frequency. (R/W)

**Register 31.23. SENS_SAR_DAC_CTRL2_REG (0x009c)**

| (reserved) | SENS_DAC_CW_EN2 | SENS_DAC_CW_EN1 | SENS_DAC_INV2 | SENS_DAC_INV1 | SENS_DAC_SCALE2 | SENS_DAC_SCALE1 | SENS_DAC_DC2 | SENS_DAC_DC1 | |
|---|---|---|---|---|---|---|---|---|---|
| 31 26 | 25 | 24 | 23 22 | 21 20 | 19 18 | 17 16 | 15 8 | 7 0 | |
| 0 0 0 0 0 0 | 1 | 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 | Reset |

**SENS_DAC_CW_EN2** 1: selects CW generator as source for PDAC2_DAC[7:0], 0: selects register reg_pdac2_dac[7:0] as source for PDAC2_DAC[7:0]. (R/W)

**SENS_DAC_CW_EN1** 1: selects CW generator as source for PDAC1_DAC[7:0], 0: selects register reg_pdac1_dac[7:0] as source for PDAC1_DAC[7:0]. (R/W)

**SENS_DAC_INV2** DAC2, 00: does not invert any bits, 01: inverts all bits, 10: inverts MSB, 11: inverts all bits except for MSB. (R/W)

**SENS_DAC_INV1** DAC1, 00: does not invert any bits, 01: inverts all bits, 10: inverts MSB, 11: inverts all bits except for MSB. (R/W)

**SENS_DAC_SCALE2** DAC2, 00: no scale; 01: scale to 1/2; 10: scale to 1/4; 11: scale to 1/8. (R/W)

**SENS_DAC_SCALE1** DAC1, 00: no scale; 01: scale to 1/2; 10: scale to 1/4; 11: scale to 1/8. (R/W)

**SENS_DAC_DC2** DC offset for DAC2 CW generator. (R/W)

**SENS_DAC_DC1** DC offset for DAC1 CW generator. (R/W)

## 31.6.2 Advanced Peripheral Bus

The addresses in this section are relative to the base address of 0x6000_2600 (by AHB bus). The absolute register addresses are listed in Section 31.5.2 *Advanced Peripheral Bus*.

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

**Register 31.24. APB_SARADC_CTRL_REG (0x10)**

| (reserved) | APB_SARADC_DATA_TO_I2S | APB_SARADC_DATA_SAR_SEL | APB_SARADC_SAR2_PATT_P_CLEAR | APB_SARADC_SAR1_PATT_P_CLEAR | APB_SARADC_SAR2_PATT_LEN | APB_SARADC_SAR1_PATT_LEN | APB_SARADC_SAR_CLK_DIV | APB_SARADC_SAR_CLK_GATED | APB_SARADC_SAR_SEL | APB_SARADC_WORK_MODE | APB_SARADC_SAR2_MUX | APB_SARADC_START | APB_SARADC_START_FORCE | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 27 | 26 | 25 | 24 | 23 | 22 19 | 18 15 | 14 7 | 6 | 5 | 4 3 | 2 | 1 | 0 | |
| 0 0 0 0 0 | 0 | 0 | 0 | 0 | 15 | 15 | 4 | 1 | 0 | 0 | 0 | 0 | 0 | Reset |

**APB_SARADC_DATA_TO_I2S** 1: I2S input data is from SAR ADC (for DMA), 0: I2S input data is from GPIO matrix. (R/W)

**APB_SARADC_DATA_SAR_SEL** 1: sar_sel will be coded by the MSB of the 16-bit output data, in this case, the resolution should not contain more than 11 bits; 0: using 12-bit SAR ADC resolution. (R/W)

**APB_SARADC_SAR2_PATT_P_CLEAR** Clears the pointer of pattern table for DIG ADC2 CTRL. (R/W)

**APB_SARADC_SAR1_PATT_P_CLEAR** Clears the pointer of pattern table for DIG ADC1 CTRL. (R/W)

**APB_SARADC_SAR2_PATT_LEN** SAR ADC2, 0 - 15 means pattern table length of 1 - 16. (R/W)

**APB_SARADC_SAR1_PATT_LEN** SAR ADC1, 0 - 15 means pattern table length of 1 - 16. (R/W)

**APB_SARADC_SAR_CLK_DIV** SAR clock divider. (R/W)

**APB_SARADC_SAR_CLK_GATED** Reserved. Please initialize to 0b1 (R/W)

**APB_SARADC_SAR_SEL** 0: SAR1, 1: SAR2, this setting is applicable in the single SAR mode. (R/W)

**APB_SARADC_WORK_MODE** 0: single mode, 1: double mode, 2: alternate mode. (R/W)

**APB_SARADC_SAR2_MUX** 1: SAR ADC2 is controlled by DIG ADC2 CTRL, 0: SAR ADC2 is controlled by PWDET CTRL. (R/W)

**APB_SARADC_START** Reserved. Please initialize to 0 (R/W)

**APB_SARADC_START_FORCE** Reserved. Please initialize to 0 (R/W)

### Register 31.25. APB_SARADC_CTRL2_REG (0x14)

| (reserved) | APB_SARADC_SAR2_INV | APB_SARADC_SAR1_INV | APB_SARADC_MAX_MEAS_NUM | APB_SARADC_MEAS_NUM_LIMIT | |
|---|---|---|---|---|---|
| 31 … 11 | 10 | 9 | 8 … 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 | 255 | 0 | Reset |

**APB_SARADC_SAR2_INV** 1: data to DIG ADC2 CTRL is inverted, 0: data is not inverted. (R/W)

**APB_SARADC_SAR1_INV** 1: data to DIG ADC1 CTRL is inverted, 0: data is not inverted. (R/W)

**APB_SARADC_MAX_MEAS_NUM** Max conversion number. (R/W)

**APB_SARADC_MEAS_NUM_LIMIT** Reserved. Please initialize to 0b1 (R/W)

### Register 31.26. APB_SARADC_FSM_REG (0x18)

| APB_SARADC_SAMPLE_CYCLE | (reserved) | (reserved) | (reserved) | |
|---|---|---|---|---|
| 31 … 24 | 23 … 16 | 15 … 8 | 7 … 0 | |
| 2 | 8 | 255 | 8 | Reset |

**APB_SARADC_SAMPLE_CYCLE** Sample cycles. (R/W)

### Register 31.27. APB_SARADC_SAR1_PATT_TAB1_REG (0x1C)

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR1_PATT_TAB1_REG** Pattern tables 0 - 3 for SAR ADC1, one byte for each pattern table: [31:28] pattern0_channel, [27:26] pattern0_bit_width, [25:24] pattern0_attenuation, [23:20] pattern1_channel, etc. (R/W)

### Register 31.28. APB_SARADC_SAR1_PATT_TAB2_REG (0x20)

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR1_PATT_TAB2_REG** Pattern tables 4 - 7 for SAR ADC1, one byte for each pattern table: [31:28] pattern4_channel, [27:26] pattern4_bit_width, [25:24] pattern4_attenuation, [23:20] pattern5_channel, etc. (R/W)

### Register 31.29. APB_SARADC_SAR1_PATT_TAB3_REG (0x24)

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR1_PATT_TAB3_REG** Pattern tables 8 - 11 for SAR ADC1, one byte for each pattern table: [31:28] pattern8_channel, [27:26] pattern8_bit_width, [25:24] pattern8_attenuation, [23:20] pattern9_channel, etc. (R/W)

### Register 31.30. APB_SARADC_SAR1_PATT_TAB4_REG (0x28)

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR1_PATT_TAB4_REG** Pattern tables 12 - 15 for SAR ADC1, one byte for each pattern table: [31:28] pattern12_channel, [27:26] pattern12_bit_width, [25:24] pattern12_attenuation, [23:20] pattern13_channel, etc. (R/W)

### Register 31.31. APB_SARADC_SAR2_PATT_TAB1_REG (0x2C)

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR2_PATT_TAB1_REG** Pattern tables 0 - 3 for SAR ADC2, one byte for each pattern table: [31:28] pattern0_channel, [27:26] pattern0_bit_width, [25:24] pattern0_attenuation, [23:20] pattern1_channel, etc. (R/W)

**Register 31.32. APB_SARADC_SAR2_PATT_TAB2_REG (0x30)**

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR2_PATT_TAB2_REG** Pattern tables 4 - 7 for SAR ADC2, one byte for each pattern table: [31:28] pattern4_channel, [27:26] pattern4_bit_width, [25:24] pattern4_attenuation, [23:20] pattern5_channel, etc. (R/W)

**Register 31.33. APB_SARADC_SAR2_PATT_TAB3_REG (0x34)**

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR2_PATT_TAB3_REG** Pattern tables 8 - 11 for SAR ADC2, one byte for each pattern table: [31:28] pattern8_channel, [27:26] pattern8_bit_width, [25:24] pattern8_attenuation, [23:20] pattern9_channel, etc. (R/W)

**Register 31.34. APB_SARADC_SAR2_PATT_TAB4_REG (0x38)**

| 31 … 0 | |
|---|---|
| 0x00F0F0F0F | Reset |

**APB_SARADC_SAR2_PATT_TAB4_REG** Pattern tables 12 - 15 for SAR ADC2, one byte for each pattern table: [31:28] pattern12_channel, [27:26] pattern12_bit_width, [25:24] pattern12_attenuation, [23:20] pattern13_channel, etc. (R/W)

### 31.6.3 RTC I/O

For details, please refer to Section Registers in Chapter IO_MUX and GPIO Matrix.

# Part VII

# Appendix

This part contains the following information starting from the next page:


- *Related Documentation and Resources*
- *Glossary*
- *Programming Reserved Register Field*
- *Interrupt Configuration Registers*
- *Revision History*

# Related Documentation and Resources

## Related Documentation

- ESP32 Series Datasheet – Specifications of the ESP32 hardware.
- *ESP32 Hardware Design Guidelines* – Guidelines on how to integrate the ESP32 into your hardware product.
- ESP32 ECO and Workarounds for Bugs – Correction of ESP32 design errors.
- *ESP32 Series SoC Errata* – Descriptions of known errors in ESP32 series of SoCs.
- *Certificates*
  https://espressif.com/en/support/documents/certificates
- *ESP32 Product/Process Change Notifications (PCN)*
  https://espressif.com/en/support/documents/pcns
- *ESP32 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://espressif.com/en/support/documents/advisories
- *Documentation Updates and Update Notification Subscription*
  https://espressif.com/en/support/download/documents

## Developer Zone

- ESP-IDF Programming Guide for ESP32 – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.
  https://github.com/espressif
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- *ESP-FAQ* – A summary document of frequently asked questions released by Espressif.
  https://espressif.com/projects/esp-faq/en/latest/index.html
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.
  https://espressif.com/en/support/download/sdks-demos

## Products

- *ESP32 Series SoCs* – Browse through all ESP32 SoCs.
  https://espressif.com/en/products/socs?id=ESP32
- *ESP32 Series Modules* – Browse through all ESP32-based modules.
  https://espressif.com/en/products/modules?id=ESP32
- *ESP32 Series DevKits* – Browse through all ESP32-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32
- *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 Accelerator |
| AHB | Advanced High-performance Bus. For more information, please refer to its specification. |
| APB | Advanced Peripheral Bus. For more information, please refer to its specification. |
| DMA | DMA Controller |
| DPORT | Short for Data Port. The full name is Xtensa Local Memory Interface (XLMI) Port. |
| eFuse | eFuse Controller |
| EMAC | Ethernet MAC |
| I2C | I2C Controller |
| I2S | I2S Controller |
| LEDC | LED_PWM Controller |
| MCPWM | Motor Control PWM |
| MMU | Memory Management Unit |
| MPU | Memory Protection Unit |
| PCNT | Pulse Count Controller |
| PERI | Peripheral |
| PID | Process Identifier |
| PMU | Power Management Unit |
| RMT | Remote Control Peripheral |
| RNG | Random Number Generator |
| RSA | RSA Accelerator |
| RTC | Real Time Controller. A group of circuits in SoC that keeps working in any chip mode and at any time. |
| SDHOST | SD/MMC Host Controller |
| SHA | SHA Accelerator |
| SPI | SPI Controller |
| TIMG | Timer Group |
| TWAI | Two-wire Automotive Interface |
| UART | UART Controller |
| ULP Coprocessor | Ultra-low-power Coprocessor |
| WDT | Watchdog Timers |

## Abbreviations Related to Registers

| | |
|---|---|
| REG | **Register**. |
| SYSREG | **System registers** are a group of registers that control system reset, memory, clocks, software interrupts, power management, clock gating, etc. |

ISO **Isolation**. If a peripheral or other chip component is powered down, the pins, if any, to which its output signals are routed will go into a floating state. ISO registers isolate such pins and keep them at a certain determined value, so that the other non-powered-down peripherals/devices attached to these pins are not affected.

NMI **Non-maskable interrupt** is a hardware interrupt that cannot be disabled or ignored by the CPU instructions. Such interrupts exist to signal the occurrence of a critical error.

W1TS Abbreviation added to names of registers/fields to indicate that such register/field should be used to set a field in a corresponding register with a similar name. For example, the register `GPIO_ENABLE_W1TS_REG` should be used to set the corresponding fields in the register `GPIO_ENABLE_REG`.

W1TC Same as *W1TS*, but used to clear a field in a corresponding register.

## Access Types for Registers

Sections *Register Summary* and *Register Description* in TRM chapters specify access types for registers and their fields.

Most frequently used access types and their combinations are as follows:

- RO
- WO
- WT
- R/W
- R/W1
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

Descriptions of all access types are provided below.

R **Read.** User application can read from this register/field; usually combined with other access types.

RO **Read only.** User application can only read from this register/field.

HRO **Hardware Read Only.** Only hardware can read from this register/field; used for storing default settings for variable parameters.

W **Write.** User application can write to this register/field; usually combined with other access types.

WO **Write only.** User application can only write to this register/field.

W1 **Write Once.** User application can write to this register/field only once; only allowed to write 1; writing 0 is invalid.

SS **Self set.** On a specified event, hardware automatically writes 1 to this register/field; used with 1-bit fields.

SC **Self clear.** On a specified event, hardware automatically writes 0 to this register/field; used with 1-bit and multi-bit fields.

SM **Self modify.** On a specified event, hardware automatically writes a specified value to this register/field; used with multi-bit fields.

SU **Self update.** On a specified event, hardware automatically updates this register/field; used with multi-bit fields.

RS **Read to set.** If user application reads from this register/field, hardware automatically writes 1 to it.

RC **Read to clear.** If user application reads from this register/field, hardware automatically writes 0 to it.

RF **Read from FIFO.** If user application writes new data to FIFO, the register/field automatically reads it.

WF **Write to FIFO.** If user application writes new data to this register/field, it automatically passes the data to FIFO via APB bus.

WS **Write any value to set.** If user application writes to this register/field, hardware automatically sets this register/field.

W1S **Write 1 to set.** If user application writes 1 to this register/field, hardware automatically sets this register/field.

W0S **Write 0 to set.** If user application writes 0 to this register/field, hardware automatically sets this register/field.

WC **Write any value to clear.** If user application writes to this register/field, hardware automatically clears this register/field.

W1C **Write 1 to clear.** If user application writes 1 to this register/field, hardware automatically clears this register/field.

W0C **Write 0 to clear.** If user application writes 0 to this register/field, hardware automatically clears this register/field.

WT **Write 1 to trigger an event.** If user application writes 1 to this field, this action triggers an event (pulse in the APB bus) or clears a corresponding WTC field (see WTC).

WTC **Write to clear.** Hardware automatically clears this field if user application writes 1 to the corresponding WT field (see WT).

W1T **Write 1 to toggle.** If user application writes 1 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

W0T **Write 0 to toggle.** If user application writes 0 to this field, hardware automatically inverts the corresponding field; otherwise - no effect.

WL **Write if a lock is deactivated.** If the lock is deactivated, user application can write to this register/field.

varies **The access type varies.** Different fields of this register might have different access types.

# Programming Reserved Register Field

## Introduction

A field in a register is reserved if the field is not open to users, or produces unpredictable results if configured to values other than defaults.

## Programming Reserved Register Field

The reserved fields should not be modified. It is not possible to write only part of a register since registers must always be written as a whole. As a result, to write an entire register that contains reserved fields, you can choose one of the following two options:

1. Read the value of the register, modify only the fields you want to configure and then write back the value so that reserved fields are untouched.

   OR

2. Modify only the fields you want to configure and write back the default value of the reserved fields. The default value of a field is provided in the "Reset" line of a register diagram. For example, the default value of Field_A in Register X is 1.

**Register 31.35. Register X (Address)**

| (reserved) | | | | | | | | | | | | Field_C | | | | | | | | | | | | | | | | | | | | | | | | Field_B | Field_A | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | | | | | | | | | | | 20 | 19 | | | | 16 | 15 | | | | | | | | | | | | | | | 2 | 1 | 0 | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000 | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Reset | | | | |

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 31.6-4.
- **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 31.6-4. Configuration of ENA/RAW/ST Registers**

| ENA Bit Value | RAW Bit Value | ST Bit Value |
|---|---|---|
| 0 | Ignored | 0 |
| 1 | 0 | 0 |
| | 1 | 1 |

# Revision History

| Date | Version | Release notes |
|---|---|---|
| 2026.03 | v5.7 | • Chapter 24 *Ethernet Media Access Controller (EMAC)*: Updated the bit[31:29] of the register DMAMISSEDFR_REG to reserved, and updated the field Missed_FC to bit[15:0] |
| 2025.11 | v5.6 | • Chapter 9 *Low-Power Management (RTC_CNTL)*: Modified the description of sleep rejection sources |
| 2025.08 | v5.5 | • Chapter 21 *I2C Controller (I2C)*:<br>– Added bit[9:0] to the register I2C_FIFO_CONF_REG; added bit[2:0] and bit4 to the register I2C_INT_RAW_REG, I2C_INT_CLR_REG, I2C_INT_ENA_REG, and I2C_INT_STATUS_REG<br>– Added registers I2C_CTR_REG, I2C_SR_REG, and I2C_TO_REG in Section 21.4 *Register Summary*<br>• Chapter 26 *SDIO Slave Controller (SDIO)*: Added bit[3:0] to the register SLCCONF1_REG<br>• Chapter 1 *ULP Coprocessor (ULP)*: Updated the bit width of the SLEEP instruction |
| 2025.05 | v5.4 | • Chapter 21 *I2C Controller (I2C)*: Added Register<br>• Chapter 12 *DPort Registers*: Added access exception registers and descriptions in Section 12.4 *Register Summary*<br>• Chapter 19 *UART Controller (UART)*: Updated Figure 19.3-2 *UART Shared RAM* |
| 2025.01 | v5.3 | • Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*:<br>– Added a footnote in Section 6.1 *Overview*<br>– Added Figure 6.1-2 *Internal Structure of a Pad*<br>– Updated Section 6.9 *Peripheral Signal List*<br>– Updated Section 6.12 *Register Summary* and Section 6.13 *Registers* to show only the GPIOs that are available for users<br>• Chapter 21 *I2C Controller (I2C)*: Added a note about NON-FIFO mode in Section 21.3.2 *Architecture* |

**Cont'd on next page**

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2024.08 | v5.2 | • Chapter 24 *Ethernet Media Access Controller (EMAC)*: Added a note about RMII reference clock in Section 24.6.2.1<br>• Chapter 23 *Pulse Count Controller (PCNT)*: Corrected the address of PCNT_U*n*_CNT_REG (*n*: 0-7)<br>• Chapter 28 *LED PWM Controller (LEDC)*: Corrected the addresses of some registers<br>• Chapter 9 *Low-Power Management (RTC_CNTL)*:<br>– Updated the description of predefined low-power modes<br>– Added a note about EXT1 under Table 9.3-2<br>• Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*:<br>– Updated the description of register GPIO_STATUS_REG, GPIO_STATUS_W1TS_REG, GPIO_STATUS_W1TC_REG, GPIO_STATUS1_REG, GPIO_STATUS1_W1TS_REG, and GPIO_STATUS1_W1TC_REG<br>– Updated the description of pad hold control in Section 6.7 |
| 2024.04 | v5.1 | • Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*:<br>– Updated the description of registers RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7) and RTCIO_TOUCH_PAD*m*_REG (*m* = 8, 9)<br>– Updated the description in Section 6.7<br>• Chapter 20 *SPI Controller (SPI)*: Labelled bit12 ~ bit15 in SPI_CTRL2_REG as reserved<br>• Chapter 26 *SDIO Slave Controller (SDIO)*: Fixed a few typos<br>• Chapter 24 *Ethernet Media Access Controller (EMAC)*: Labelled register EMACCSTATUS_REG as reserved<br>• Chapter 22 *I2S Controller (I2S)*: Updated interrupt description in Section 22.6.2<br>• Chapter 28 *LED PWM Controller (LEDC)*: Updated the note about not supporting to pause a fade process in Section 28.2.3, and the lowest resolution in Table 28.2-1<br>• Chapter 10 *Timer Group (TIMG)*: Updated the description of TIMG*n*_WDT_CLK_PRESCALE<br>• Chapter 31 *On-Chip Sensors and Analog Signal Processing*: Updated register APB_SARADC_FSM_REG<br>• Chapter 9 *Low-Power Management (RTC_CNTL)*: Updated description of RTC_CNTL_WDTWPROTECT_REG |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2023.07 | v5.0 | • Updated register prefix APB_CTRL to SYSCON<br>• Section 3.3.4 *Cache*: Added description that the cache block size is 32 bytes<br>• Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*: Split the original RTCIO_TOUCH_PAD*n*_REG (*n*: 0-9) into RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7) and RTCIO_TOUCH_PAD*m*_REG (*m*: 8-9); In RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7), added bits [27-31], [12-16]<br>• Revised Section 6.5.2 *Analog Function Description*<br>• Chapter 12 *DPort Registers*: Added description of two registers DPORT_PRO_CACHE_CTRL1_REG and DPORT_APP_CACHE_CTRL1_REG<br>• Section 12.3.7 *Peripheral Clock Gating and Reset*: Added a note about enabling clock and releasing reset state before using a peripheral<br>• Section 2.5 *SPI DMA Interface*: Changed the data size for a single transfer to "four bytes aligned"<br>• Chapter 24 *Ethernet Media Access Controller (EMAC)*: Removed contents about timestamp/PTP, as the feature is not supported in hardware<br>• Chapter 23 *Pulse Count Controller (PCNT)*: Added the description about the PCNT_CLK_EN bit |
| 2023.04 | v4.9 | • Removed contents about hall sensor, including relevant registers, signals, etc, acc to PCN20221202<br>• Renamed PLL_D2_CLK to PLL_F160M_CLK throughout the document<br>• Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*: Added TWAI signals in Table 6.9-1<br>• Added descriptions about the break condition and updated the maximum length of stop bits and related descriptions in Chapter 19 *UART Controller (UART)*<br>• Added the formula to calculate duty cycle resolution and updated Table *Commonly-used Frequencies and Resolutions* in Chapter 28 *LED PWM Controller (LEDC)*<br>• Chapter 31 *On-Chip Sensors and Analog Signal Processing*:<br>  – Added a note about limited applications of touch sensor in Section 31.2.2 *Features*<br>  – Removed internal signals vdd33, pa_pkdet1, pa_pkdet2<br>• Added description about "reject sleep" in Chapter 9 *Low-Power Management (RTC_CNTL)* |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2022.12 | v4.8 | Updated the following chapters:<br>• Updated section 7.2.4.2, added register summary and description section in Chapter 7 *Reset and Clock*<br>• Updated the description of MIICSRCLK and MIIBUSY field in Chapter 24 *Ethernet Media Access Controller (EMAC)*<br>• Added the descriptions of the UART_MEM_TX_STATUS_REG and UART_MEM_RX_STATUS_REG registers in Chapter 19 *UART Controller (UART)*<br>• Updated some clock names in Chapter 18 *Random Number Generator (RNG)*<br>• Updated Table 31.3-1 in Chapter 31 *On-Chip Sensors and Analog Signal Processing*<br>• Updated some clock names in Chapter 9 *Low-Power Management (RTC_CNTL)*, and added description of RTC_CNTL_MAIN_STATE_IN_IDLE<br>• Renamed the RTC8M_CLK clock RC_FAST_CLK in Chapter 5 *eFuse Controller (EFUSE)*<br>• Added clarification about relative address in the Registers sections of all chapters |
| 2022.08 | v4.7 | • Updated Table 3.3-6 in Chapter 3 *System and Memory*<br>• Updated Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*:<br>– Updated the description in Section 6.2.2<br>– Updated the description of GPIO_PIN*n*_INT_ENA and IO_MUX_PIN_CTRL<br>• Added a note about in_link descriptor in Section 2.4 in Chapter 2 *DMA Controller (DMA)*<br>• Updated clock names:<br>– RTC8M_CLK: renamed as RC_FAST_CLK<br>– RTC8M_D256_CLK: renamed as RC_FAST_DIV_CLK<br>– RTC_CLK: renamed as RC_SLOW_CLK<br>– SLOW_CLK: renamed as RTC_SLOW_CLK<br>– FAST_CLK: renamed as RTC_FAST_CLK<br>• Updated the description below Figure 23.2-2 in Chapter 23 *Pulse Count Controller (PCNT)*<br>• Updated Section 10.3 in Chapter 10 *Timer Group (TIMG)*<br>• Added two notes in Chapter 31 *On-Chip Sensors and Analog Signal Processing* |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2021.11 | v4.6 | Updated Table 3.3-6 in Chapter 3 *System and Memory*<br>Updated Table 8.3-1 in Chapter 8 *Interrupt Matrix (INTERRUPT)*<br>Updated Figure 7.1-1 in Chapter 7 *Reset and Clock*<br>Updated Table 6.9-1 and description of IO_MUX_PIN_CTRL in Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*<br>Major updates to Chapter 12 *DPort Registers*<br>Added a note below Table 22.1-1 in Chapter 22 *I2S Controller (I2S)*<br>Updated Section 26.3.6 and description of register SLCHOST_CONF_REG in Chapter 26 *SDIO Slave Controller (SDIO)*<br>Updated description in Chapter 17 *External Memory Encryption and Decryption (FLASH)*<br>Updated Section 31.4.3 in Chapter 31 *On-Chip Sensors and Analog Signal Processing*<br>Provided address clarification for Section Registers in Chapter 6, Chapter 20, Chapter 22, Chapter 30, Chapter 31, and Chapter 1<br>Added new terms to Glossary<br>Fixed typos |
| 2021.07 | v4.5 | Added Section 24.6.4<br>Added a note below Table 6.11-1<br>Added the base address information of interrupt matrix registers in Chapter 12 *DPort Registers* and in Chapter 8 *Interrupt Matrix (INTERRUPT)*<br>Updated the description in Section 7.2.7<br>Updated pin function numbers starting from Function0 in Chapter 6 *IO MUX and GPIO Matrix (GPIO, IO MUX)*<br>Updated Table 28.2-1 in Chapter 28 *LED PWM Controller (LEDC)*<br>Updated base address for Ethernet MAC to 0x3FF6_9000 in section 24.9<br>Updated the description of UART_SW_RTS and UART_RX_FLOW_THRHD fields in Chapter 19 *UART Controller (UART)*<br>Updated the feature description in Chapter 25 *Two-Wire Automotive Interface (TWAI)*<br>Updated description in Chapter 18 *Random Number Generator (RNG)*<br>Updated the description in Section 1.4.2<br>Fixed typos in Table 8.3-1 and in Section 2.4 |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2021.03 | v4.4 | Updated description to PWM_TIMER0_SYNC_REG ~ PWM_TIMER2_SYNC_REG registers in Chapter 29 *Motor Control PWM (MCPWM)*<br>Updated the description of register TIMG*n*_RTCCALICFG_REG and register TIMG*n*_RTCCALICFG1_REG in Chapter 10 *Timer Group (TIMG)*<br>Updated the trademark symbol for Chapter 25 *Two-Wire Automotive Interface (TWAI)*<br>Updated the description to RTC_CNTL_WDTCONFIG0_REG in Chapter 9: *Low-Power Management (RTC_CNTL)*<br>Updated the description to EMACGMIIADDR_REG in Chapter 24: *Ethernet Media Access Controller (EMAC)*<br>Updated the access to UART_FIFO_REG in Table 19.4-1: *UART Register Summary*<br>Updated the description to IO_MUX_*x*_REG (*x*: 0-19, 20[1], 21-23, 25-27, 32-39) and IO_MUX_*x*_REG (*x*: 0-19, 20[1], 21-23, 25-27, 32-39) in Chapter 6: *IO MUX and GPIO Matrix (GPIO, IO MUX)*<br>Updated the description of register RTC_CNTL_WDTCONFIG0_REG in Chapter 9 *Low-Power Management (RTC_CNTL)*<br>Renamed Section: Documentation Conversions to Section: Glossary<br>Updated the notes below Table 6.10-1: *IO_MUX Pin Summary* |
| 2020.09 | v4.3 | Added Chapter TWAI<br>Added section 18.4 *Programming Procedure* and updated some description in Chapter 18 *Random Number Generator (RNG)*<br>Added information about uart_download_dis in Chapter *eFuse Controller*<br>Updated the description of SPI_ADDR_REG and SPI_SLV_WR_STATUS_REG |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2020.06 | v4.2 | Added Section: Documentation Conventions<br>Changes to Chapter *System and Memory*:<br>• Added a note about DPORT address and AHB address under Table 3.3-6<br>Changes to Chapter *Reset and Clock*:<br>• Updated Table 7.2-2 by adding PLL_CLK frequency of 480 MHz and CPU_CLK frequency of 240 MHz<br>• Updated Table 7.2-4 by modifying the APB_CLK frequency to 80 MHz when the CPU_CLK source is PLL_CLK<br>Changes to Chapter *IO_MUX and GPIO Matrix*:<br>• Fixed a typo in Section 6.4.2: For inputs, the SIG_IN_SEL register must be cleared to route the input directly to the peripheral.<br>• Changed the reset values for MTCK, MTMS, GPIO27 in Table 6.10-1<br>• Updated description of register FUN_DRV<br>Changes to Chapter *I2S*:<br>• Updated Section 22.4.1.1<br>Changes to Chapter *UART Controllers*:<br>• Updated the description of registers UART_FIFO_REG and UART_RX_TOUT_THRHD<br>Changes to Chapter *LED_PWM*:<br>• Added Table 28.2-1<br>Changes to Chapter *MCPWM*:<br>• Corrected the PWM period in Count-Up, Count-Down, and Count-Up-Down modes<br>Changes to Chapter *PULSE_CNT*:<br>• Added description of register PCNT_U*n*_STATUS_REG<br>Changes to Chapter *eFuse Controller*:<br>• Combined system parameters "32pad" and "chip_version" into one: pkg_version<br>• Updated description of registers EFUSE_RD_CHIP_VER_PKG and EFUSE_CHIP_VER_PKG<br>Changes to Chapter *Low-Power Management*:<br>• Added description of register RTC_CNTL_WDTCONFIG0_REG<br>• Modified description of register RTC_CNTL_WDTCONFIG*n*_REG<br>Changes to Chapter *ULP Coprocessor*:<br>• Updated description in sections 1.4.13 and 1.4.14<br>• Fixed typos |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2019.11 | v4.1 | Changes to Chapter *IO_MUX and GPIO Matrix*:<br>• Updated Table 6.11-1;<br>• Added field RTCIO_TOUCH_PAD*n*_FUN_SEL in register RTCIO_TOUCH_PAD*n*_REG;<br>Changes to Chapter *SPI*:<br>• Fixed incorrect SPI2/SPI3 addresses in Table 20.7;<br>Changes to Chapter *I2C Controller*:<br>• Removed I2C_SLAVE_TRAN_COMP_INT interrupt;<br>Changes to Chapter *I2S*:<br>• Added a note under Table 22.1-1;<br>Changes to Chapter *UART Controllers*:<br>• Fixed errors in the description of register UART_FORCE_XOFF and UART_FORCE_XON;<br>• Fixed errors in the description of register ART_SWFC_CONF_REG;<br>Changes to Chapter *Remote Control Peripheral*:<br>• Updated Figure 30.2-1;<br>Changes to Chapter *PULSE_CNT*:<br>• Updated Figure 23.2-1;<br>• Fixed typos in the description of register PCNT_U*n*_CONF0_REG;<br>Changes to Chapter *eFuse Controller*:<br>• Added/updated eight system parameters and updated the relevant register description;<br>• Updated the configuration values in Table 5.3-4;<br>• Modified the bit width of system parameter flash_crypt_cnt to 7 bits;<br>Changes to Chapter *PID/MPU/MMU*:<br>• Added a note under Table 4.3-8;<br>Changes to Chapter *On-Chip Sensors and Analog Signal Processing*:<br>• Fixed typos in the description of registers SENS_SAR2_BIT_WIDTH and SENS_SAR1_BIT_WIDTH;<br>Changes to Chapter *ULP Coprocessor*:<br>• Corrected the OpCode for REG_WR;<br>• Updated Section 1.6.2.4;<br>• Fixed typos in the description of registers RTC_I2C_RX_LSB_FIRST and RTC_I2C_TX_LSB_FIRST;<br>• Removed the description of registers RTC_I2C_SLAVE_TRAN_COMP_INT_ENA and RTC_I2C_SLAVE_TRAN_COMP_INT_ST;<br>Changes to Chapter *Low-Power Management*:<br>• Updated the default value and description of register RTC_CNTL_DBROWN_OUT_THRES;<br>• Updated the description of register RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA;<br>Added documentation feedback hyperlink. |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2018.12 | v4.0 | Updated some register names in Chapter IO_MUX and GPIO Matrix to be consistent with the header files;<br>Changes to Chapter 20 SPI:<br>• Updated Section 20.3;<br>• Updated Section 20.5.1;<br>• Updated Section 20.8;<br>Changes to Chapter 19 UART Controllers:<br>• Removed support for 4 STOP bits;<br>• Added a note at the end of section 19.3.2;<br>• Updated the description of register UART_DLO_EN;<br>Added a note at the end of section 9.3.10 in Chapter 9: Low-Power Management. |
| 2018.10 | v3.9 | Updated Figure 21.3-3: I2C Sequence Chart, in Chapter 21: I2C Controller. |
| 2018.09 | v3.8 | Updated the description of register TIMG*n*_T*x*_ALARM_EN;<br>Added description of ULP wakeup time in section 1.5: ULP Program Execution. |
| 2018.08 | v3.7 | Updated the description of register UART_RX_GAP_TOUT. |
| 2018.08 | v3.6 | Updated the conditions of jumps to relative address in section 1.4.6;<br>Updated the description of register UART_ACTIVE_THRESHOLD. |
| 2018.07 | v3.5 | Changes to chapter 30 RMT:<br>• Updated RAM start address in section 30.2.2: RMT RAM;<br>• Corrected several wrong addresses of RMT registers;<br>• Updated the description of register RMT_APB_CONF_REG.<br>Updated the description of registers UART_RX_TOUT_THRHD, UART_RXFIFO_FULL_INT_CLR, UART_RXFIFO_FULL_INT_CLR. |
| 2018.06 | v3.4 | Updated the images in Section 6.8: ESP32 I/O Pad Power Supplies;<br>Updated Section 21.3.3: I2C Bus Timing;<br>Added notes to Section 28.2.3: LED_PWM Channels;<br>Updated the "Maximum count value" in Section 23.2.3: Pulse Counter Watchpoints;<br>Removed the description of the temperature sensor and LNA. |
| 2018.05 | v3.3 | Updated the addresses of registers in the Register Summary Section and the Registers Section of Chapter Low-Power Management. |
| 2018.04 | v3.2 | Updated Figure 26.3-3 CMD53 Content;<br>Added seven registers in Chapter Ethernet MAC:<br>• DMAOPERATION_MODE_REG;<br>• DMAIN_EN_REG,<br>• DMAMISSEDFR_REG,<br>• PMT_RWUFFR_REG,<br>• PMT_CSR_REG,<br>• EMACLPI_CSR_REG, and<br>• EMACLPITIMERSCONTROL_REG. |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| 2018.04 | v3.1 | Updated Figure 30.2-1 RMT Architecture;<br>Added a note to Section 6.7;<br>Added the function description for the bits of the register in Section 6.46. |
| 2018.03 | v3.0 | Updated the instruction layout diagram of ST in Section 1.4.2;<br>Added description of registers EMACADDR2HIGH_REG to EMACADDR7LOW_REG in Section 24.9 and Section 24.10. |
| 2018.02 | v2.9 | Updated Sections 6.2.2, 6.2.3, 6.3.2;<br>Added registers I2S_FIFO_WR_REG and I2S_FIFO_RD_REG in Section I2S Registers. |
| 2018.01 | v2.8 | Added Chapter Ethernet MAC.<br>Added the description of system parameter BLK3_part_reserve in Chapter eFuse Controller. |
| 2017.12 | v2.7 | Added Subsection Cache in Section System and Memory;<br>Updated Section Timers and the naming of several registers in LED_PWM;<br>Updated the description of console_debug_disable in Chapter eFuse Controller. |
| 2017.11 | v2.6 | Updated Chapter Remote Controller Peripheral:<br>• Updated Figure 30.2-1 RMT Architecture;<br>• Updated section RMT RAM;<br>• Updated section Transmitter;<br>• Updated the description of RMT_CH*n*_TX_THR_EVENT_INT.<br>Added notes in Section UART RAM and Register UART_CONF0_REG. |
| 2017.11 | v2.5 | Updated the addresses for register SPI_CTRL_REG in Section SPI Register Summary;<br>Added Section Clock Phase Selection in Chapter SD/MMC Host Controller, and a description of register CLK_EDGE_SEL;<br>Major revision on Chapter I2C Controller. |
| 2017.09 | v2.4 | Added the description of register SLC0HOST_TOKEN_RDATA in Chapter SDIO Slave;<br>Added notes in Section The Clock of I2S Module;<br>Added a note in Section GP-SPI Master Mode;<br>Added Chapter DPort Register;<br>Added Chapter DMA Controller. |
| 2017.08 | v2.3 | Added Chapter Flash Encryption/Decryption. |
| 2017.07 | v2.2 | Added Chapter Low-Power Management. |
| 2017.07 | v2.1 | Updated the addresses of the GPIO configuration/data registers and the GPIO RTC function configuration registers in Chapter IO_MUX and GPIO Matrix;<br>Added Chapter PID Controller. |
| 2017.07 | v2.0 | Added Chapter SDIO Slave. |
| 2017.06 | v1.9 | Updated Chapter IO_MUX and GPIO Matrix;<br>Added Chapter MCPWM. |
| 2017.06 | v1.8 | Added register I2S_STATE_REG in Chapter I2S;<br>Updated Chapter IO_MUX and GPIO Matrix; |

Cont'd on next page

Cont'd from previous page

| Date | Version | Release notes |
|---|---|---|
| | | Added Chapter ULP Coprocessor. |
| 2017.05 | v1.7 | Added Chapter On-Chip Sensors and Analog Signal Processing;<br>Added Section Audio PLL;<br>Updated Section eFuse Controller Register Summary;<br>Updated Sections I2S PDM and LCD MODE;<br>Updated Section: Communication Format Supported by GP-SPI Slave. |
| 2017.03 | v1.6 | Added Chapter SD/MMC Host Controller;<br>Added register IO_MUX_PIN_CTRL in Chapter IO_MUX and GPIO Matrix. |
| 2017.03 | v1.5 | Added Chapter I2S. |
| 2017.01 | v1.4 | Added Chapter SPI;<br>Added Chapter UART Controllers. |
| 2016.12 | v1.3 | Added Chapter eFuse Controller;<br>Added Chapter RSA Accelerator;<br>Added Chapter Random Number Generator;<br>Updated Section I2C Controller Interrupt and Section I2C Controller Registers. |
| 2016.11 | v1.2 | Added Chapter PID/MPU/MMU;<br>Updated Section IO_MUX and GPIO Matrix Register Summary;<br>Updated Section LED_PWM Register Summary. |
| 2016.09 | v1.1 | Added Chapter I2C Controller. |
| 2016.08 | v1.0 | Initial release. |

## Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

ALL THIRD PARTY'S INFORMATION IN THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES TO ITS AUTHENTICITY AND ACCURACY.

NO WARRANTY IS PROVIDED TO THIS DOCUMENT FOR ITS MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, NOR DOES ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

**Copyright © 2026 Espressif Systems (Shanghai) Co., Ltd. All rights reserved.**

www.espressif.com