

# ESP8684

## Technical Reference Manual Version 1.3



ESPRESSIF

## About This Document

The **ESP8684 Technical Reference Manual** is targeted at developers working on low level software projects that use the ESP8684 SoC. It describes the hardware modules listed below for the ESP8684 SoC and other products in ESP8684 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.	ESP8684 Chapters	Progress
<b>Part I. Microprocessor and Master</b>		
1	ESP-RISC-V CPU	Published
2	GDMA Controller (GDMA)	Published
<b>Part II. Memory Organization</b>		
3	System and Memory	Published
4	eFuse Controller (eFuse)	Published
<b>Part III. System Component</b>		
5	IO MUX and GPIO Matrix (GPIO, IO MUX)	Published
6	Reset and Clock	Published
7	Chip Boot Control	Published
8	Interrupt Matrix (INTMTRX)	Published
9	Low-power Management (RTC_CNTL)	Published
10	System Timer (SYSTIMER)	Published
11	Timer Group (TIMG)	Published
12	Watchdog Timers (WDT)	Published
13	System Registers (SYSTEM)	Published
14	Debug Assistant (ASSIST_DEBUG)	Published
<b>Part IV. Cryptography/Security Component</b>		
15	ECC Hardware Accelerator (ECC)	Published
16	SHA Accelerator (SHA)	Published
17	External Memory Encryption and Decryption (XTS_AES)	Published
18	Random Number Generator (RNG)	Published
<b>Part V. Connectivity Interface</b>		
19	UART Controller (UART)	Published
20	SPI Controller (SPI)	Published
21	I2C Master Controller (I2C)	Published
22	LED PWM Controller (LEDC)	Published
<b>Part VI. Analog Signal Processing</b>		
23	On-Chip Sensor 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/esp8684\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/documentation/esp8684_technical_reference_manual_en.pdf)



# Contents

<b>I</b>	<b>Microprocessor and Master</b>	<b>21</b>
<b>1</b>	<b>ESP-RISC-V CPU</b>	<b>22</b>
1.1	Overview	22
1.2	Features	22
1.3	Address Map	23
1.4	Configuration and Status Registers (CSRs)	23
1.4.1	Register Summary	23
1.4.2	Register Description	25
1.5	Interrupt Controller	33
1.5.1	Features	33
1.5.2	Functional Description	33
1.5.3	Suggested Operation	35
1.5.3.1	Latency Aspects	35
1.5.3.2	Configuration Procedure	35
1.5.4	Register Summary	36
1.5.5	Register Description	37
1.6	Debug	38
1.6.1	Overview	38
1.6.2	Features	39
1.6.3	Functional Description	39
1.6.4	Register Summary	39
1.6.5	Register Description	39
1.7	Hardware Trigger	42
1.7.1	Features	42
1.7.2	Functional Description	42
1.7.3	Trigger Execution Flow	43
1.7.4	Register Summary	43
1.7.5	Register Description	44
1.8	Memory Protection	48
1.8.1	Overview	48
1.8.2	Features	48
1.8.3	Functional Description	48
1.8.4	Register Summary	48
1.8.5	Register Description	50
<b>2</b>	<b>GDMA Controller (GDMA)</b>	<b>51</b>
2.1	Overview	51
2.2	Features	51
2.3	Architecture	52
2.4	Functional Description	52
2.4.1	Data Transfer Between Peripheral and Memory	52

2.4.2	Memory-to-Memory Data Transfer	53
2.4.3	Linked List	53
2.4.4	Enabling GDMA	54
2.4.5	Linked List Reading Process	55
2.4.6	EOF	56
2.4.7	Accessing Internal RAM	56
2.4.8	Arbitration	57
2.5	GDMA Interrupts	57
2.6	Programming Procedures	58
2.6.1	Programming Procedure for GDMA Clock and Reset	58
2.6.2	Programming Procedure for GDMA's Transmit Channel	58
2.6.3	Programming Procedure for GDMA's Receive Channel	58
2.6.4	Programming Procedure for Memory-to-Memory Transfer	59
2.7	Register Summary	60
2.8	Registers	62
<b>II</b>	<b>Memory Organization</b>	<b>80</b>
<b>3</b>	<b>System and Memory</b>	<b>81</b>
3.1	Overview	81
3.2	Features	81
3.3	Functional Description	82
3.3.1	Address Mapping	82
3.3.2	Internal Memory	83
3.3.3	External Memory	84
3.3.3.1	External Memory Address Mapping	84
3.3.3.2	Cache	85
3.3.3.3	Cache Operations	85
3.3.4	GDMA Address Space	86
3.3.5	Modules/Peripherals	86
3.3.5.1	Module/Peripheral Address Mapping	86
<b>4</b>	<b>eFuse Controller (eFuse)</b>	<b>88</b>
4.1	Overview	88
4.2	Features	88
4.3	Functional Description	88
4.3.1	Structure	88
4.3.1.1	EFUSE_WR_DIS	91
4.3.1.2	EFUSE_RD_DIS	91
4.3.1.3	Data Storage	91
4.3.2	Programming of Parameters	92
4.3.3	User Read of Parameters	94
4.3.4	eFuse VDDQ Timing	95
4.3.5	Parameters Used by Hardware Modules	95
4.3.6	Interrupts	96
4.4	Register Summary	97

4.5	Registers	99
<b>III</b>	<b>System Component</b>	<b>117</b>
<b>5</b>	<b>IO MUX and GPIO Matrix (GPIO, IO MUX)</b>	<b>118</b>
5.1	Overview	118
5.2	Features	118
5.3	Architectural Overview	119
5.4	Peripheral Input via GPIO Matrix	120
5.4.1	Overview	120
5.4.2	Signal Synchronization	120
5.4.3	Functional Description	121
5.4.4	Simple GPIO Input	122
5.5	Peripheral Output via GPIO Matrix	123
5.5.1	Overview	123
5.5.2	Functional Description	123
5.5.3	Simple GPIO Output	124
5.6	Direct Input and Output via IO MUX	124
5.6.1	Overview	124
5.6.2	Functional Description	125
5.7	Analog Functions of GPIO Pins	125
5.8	Pin Functions in Light-sleep	125
5.9	Pin Hold Feature	126
5.10	Power Supplies and Management of GPIO Pins	126
5.10.1	Power Supplies of GPIO Pins	126
5.10.2	Power Supply Management	126
5.11	Peripheral Signal List	126
5.12	IO MUX Functions List	133
5.13	Analog Functions List	134
5.14	Register Summary	134
5.14.1	GPIO Matrix Register Summary	134
5.14.2	IO MUX Register Summary	136
5.15	Registers	137
5.15.1	GPIO Matrix Registers	137
5.15.2	IO MUX Registers	144
<b>6</b>	<b>Reset and Clock</b>	<b>147</b>
6.1	Reset	147
6.1.1	Overview	147
6.1.2	Architectural Overview	147
6.1.3	Features	147
6.1.4	Functional Description	148
6.2	Clock	149
6.2.1	Overview	149
6.2.2	Architectural Overview	149
6.2.3	Features	149

6.2.4	Functional Description	150
6.2.4.1	CPU Clock	150
6.2.4.2	Peripheral Clock	150
6.2.4.3	Wireless Clock	153
6.2.4.4	RTC Clock	154
<b>7</b>	<b>Chip Boot Control</b>	<b>155</b>
7.1	Overview	155
7.2	Features	155
7.3	Functional Description	155
7.3.1	Default Configuration	156
7.3.2	Boot Mode Control	156
7.3.3	ROM Code Printing Control	158
<b>8</b>	<b>Interrupt Matrix (INTMTRX)</b>	<b>159</b>
8.1	Overview	159
8.2	Features	159
8.3	Functional Description	160
8.3.1	Peripheral Interrupt Sources	160
8.3.2	CPU Interrupts	163
8.3.3	Allocate Peripheral Interrupt Source to CPU Interrupt	163
8.3.3.1	Allocate one peripheral interrupt source (Source_ <b>X</b> ) to CPU	163
8.3.3.2	Allocate multiple peripheral interrupt sources (Source_ <b>Xn</b> ) to CPU	163
8.3.3.3	Disable CPU peripheral interrupt source (Source_ <b>X</b> )	163
8.3.4	Query Current Interrupt Status of Peripheral Interrupt Source	163
8.4	Register Summary	165
8.5	Registers	168
<b>9</b>	<b>Low-power Management (RTC_CNTL)</b>	<b>173</b>
9.1	Introduction	173
9.2	Features	173
9.3	Functional Description	173
9.3.1	Power Management Unit (PMU)	175
9.3.2	Low-Power Clocks	176
9.3.3	Timers	177
9.3.4	Voltage Regulators	178
9.3.4.1	Digital System Voltage Regulator	178
9.3.4.2	Low-power Voltage Regulator	179
9.4	Brownout Detector	179
9.5	Power Modes Management	181
9.5.1	Power Domains	181
9.5.2	Pre-defined Power Modes	181
9.5.3	Wakeup Sources	182
9.5.4	Reject Sleep	183
9.6	Register Summary	184
9.7	Registers	186

<b>10</b>	<b>System Timer (SYSTIMER)</b>	217
10.1	Overview	217
10.2	Features	217
10.3	Clock Source Selection	218
10.4	Functional Description	218
10.4.1	Counter	218
10.4.2	Comparator and Alarm	219
10.4.3	Synchronization Operation	220
10.4.4	Interrupt	221
10.5	Programming Procedure	221
10.5.1	Read Current Count Value	221
10.5.2	Configure One-Time Alarm in Target Mode	221
10.5.3	Configure Periodic Alarms in Period Mode	221
10.5.4	Update After Light-sleep	222
10.6	Register Summary	222
10.7	Registers	224
<b>11</b>	<b>Timer Group (TIMG)</b>	235
11.1	Overview	235
11.2	Features	235
11.3	Functional Description	236
11.3.1	16-bit Prescaler and Clock Selection	236
11.3.2	54-bit Time-base Counter	236
11.3.3	Alarm Generation	237
11.3.4	Timer Reload	238
11.3.5	RTC_SLOW_CLK Frequency Calculation	238
11.3.6	Interrupts	238
11.4	Configuration and Usage	239
11.4.1	Timer as a Simple Clock	239
11.4.2	Timer as One-shot Alarm	239
11.4.3	Timer as Periodic Alarm	240
11.4.4	RTC_SLOW_CLK Frequency Calculation	240
11.5	Register Summary	242
11.6	Registers	244
<b>12</b>	<b>Watchdog Timers (WDT)</b>	254
12.1	Overview	254
12.2	Digital Watchdog Timers	256
12.2.1	Features	256
12.2.2	Functional Description	256
12.2.2.1	Clock Source and 32-Bit Counter	257
12.2.2.2	Stages and Timeout Actions	257
12.2.2.3	Write Protection	258
12.2.2.4	Flash Boot Protection	258
12.3	Super Watchdog	258
12.3.1	Features	259



12.3.2	Super Watchdog Controller	259
12.3.2.1	Structure	259
12.3.2.2	Workflow	259
12.4	Interrupts	260
12.5	Registers	260
<b>13</b>	<b>System Registers (SYSTEM)</b>	261
13.1	Overview	261
13.2	Features	261
13.3	Function Description	261
13.3.1	System and Memory Registers	261
13.3.1.1	Internal Memory	261
13.3.1.2	External Memory	262
13.3.2	Clock Registers	262
13.3.3	Interrupt Signal Registers	263
13.3.4	Peripheral Clock Gating and Reset Registers	263
13.4	Register Summary	265
13.5	Registers	266
<b>14</b>	<b>Debug Assistant (ASSIST_DEBUG)</b>	275
14.1	Overview	275
14.2	Features	275
14.3	Functional Description	275
14.3.1	SP Monitoring	275
14.3.2	PC Logging	275
14.3.3	CPU Debugging Status Logging	275
14.4	Recommended Operation	276
14.4.1	SP Monitoring	276
14.4.2	PC Logging Configuration Process	276
14.5	Register Summary	277
14.6	Registers	278
<b>IV</b>	<b>Cryptography/Security Component</b>	284
<b>15</b>	<b>ECC Hardware Accelerator (ECC)</b>	285
15.1	Introduction	285
15.2	Features	285
15.3	Terminology	285
15.3.1	ECC Basics	285
15.3.1.1	Elliptic Curve and Points on the Curves	285
15.3.1.2	Affine Coordinates and Jacobian Coordinates	285
15.3.2	ECC Definitions	286
15.3.2.1	Memory Blocks	286
15.3.2.2	Data and Data Block	286
15.3.2.3	Write Data	286
15.3.2.4	Read Data	287

15.3.2.5	Standard Calculation and Jacobian Calculation	287
15.4	Function Description	287
15.4.1	Key Size	287
15.4.2	Working Modes	287
15.4.2.1	Base Point Multiplication (Point Multi Mode)	288
15.4.2.2	Finite Field Division (Division Mode)	288
15.4.2.3	Base Point Verification (Point Verif Mode)	288
15.4.2.4	Base Point Verification + Base Point Multiplication (Point Verif + Multi Mode)	288
15.4.2.5	Jacobian Point Multiplication (Jacobian Point Multi Mode)	289
15.4.2.6	Jacobian Point Verification (Jacobian Point Verif Mode)	289
15.4.2.7	Base Point Verification + Jacobian Point Multiplication (Point Verif + Jacobian Point Multi Mode)	289
15.5	Clocks and Resets	290
15.6	Interrupts	290
15.7	Programming Procedures	290
15.8	Register Summary	291
15.9	Registers	292
<b>16</b>	<b>SHA Accelerator (SHA)</b>	295
16.1	Introduction	295
16.2	Features	295
16.3	Working Modes	295
16.4	Function Description	297
16.4.1	Preprocessing	297
16.4.1.1	Padding the Message	297
16.4.1.2	Parsing the Message	297
16.4.1.3	Setting the Initial Hash Value	298
16.4.2	Hash Operation	298
16.4.2.1	Typical SHA Mode Process	298
16.4.2.2	DMA-SHA Mode Process	299
16.4.3	Message Digest	300
16.4.4	Interrupt	301
16.5	Register Summary	301
16.6	Registers	302
<b>17</b>	<b>External Memory Encryption and Decryption (XTS_AES)</b>	306
17.1	Overview	306
17.2	Features	306
17.3	Module Structure	306
17.4	Functional Description	307
17.4.1	XTS Algorithm	307
17.4.2	Key	307
17.4.3	Target Memory Space	308
17.4.4	Data Writing	309
17.4.5	Manual Encryption Block	309
17.4.6	Auto Decryption Block	310

17.5	Software Process	310
17.6	Register Summary	312
17.7	Registers	313
<b>18</b>	<b>Random Number Generator (RNG)</b>	<b>316</b>
18.1	Introduction	316
18.2	Features	316
18.3	Functional Description	316
18.4	Programming Procedure	317
18.5	Register Summary	317
18.6	Register	318
<b>V</b>	<b>Connectivity Interface</b>	<b>319</b>
<b>19</b>	<b>UART Controller (UART)</b>	<b>320</b>
19.1	Overview	320
19.2	Features	320
19.3	UART Architecture	321
19.4	Functional Description	323
19.4.1	Clock and Reset	323
19.4.2	UART RAM	324
19.4.3	Baud Rate Generation and Detection	325
19.4.3.1	Baud Rate Generation	325
19.4.3.2	Baud Rate Detection	326
19.4.4	UART Data Frame	327
19.4.5	AT_CMD Character Structure	327
19.4.6	RS485	328
19.4.6.1	Driver Control	328
19.4.6.2	Turnaround Delay	328
19.4.6.3	Bus Snooping	329
19.4.7	IrDA	329
19.4.8	Wake-up	330
19.4.9	Flow Control	330
19.4.9.1	Hardware Flow Control	331
19.4.9.2	Software Flow Control	332
19.4.10	UART Interrupts	333
19.5	Programming Procedures	334
19.5.1	Register Type	334
19.5.1.1	Synchronous Registers	334
19.5.1.2	Static Registers	335
19.5.1.3	Immediate Registers	336
19.5.2	Detailed Steps	336
19.5.2.1	Initializing UART $n$	337
19.5.2.2	Configuring UART $n$ Communication	338
19.5.2.3	Enabling UART $n$	338
19.6	Register Summary	339

19.7	Registers	341
<b>20</b>	<b>SPI Controller (SPI)</b>	<b>361</b>
20.1	Overview	361
20.2	Glossary	361
20.3	Features	362
20.4	Architectural Overview	363
20.5	Functional Description	363
20.5.1	Data Modes	364
20.5.2	Introduction to FSPI Bus Signals	364
20.5.3	Bit Read/Write Order Control	366
20.5.4	Transfer Modes	368
20.5.5	CPU-Controlled Data Transfer	368
20.5.5.1	CPU-Controlled Master Mode	368
20.5.5.2	CPU-Controlled Slave Mode	370
20.5.6	DMA-Controlled Data Transfer	371
20.5.6.1	GDMA Configuration	371
20.5.6.2	GDMA TX/RX Buffer Length Control	372
20.5.7	Data Flow Control in GP-SPI2 Master and Slave Modes	373
20.5.7.1	GP-SPI2 Functional Blocks	373
20.5.7.2	Data Flow Control in Master Mode	374
20.5.7.3	Data Flow Control in Slave Mode	375
20.5.8	GP-SPI2 Works as a Master	376
20.5.8.1	State Machine	376
20.5.8.2	Register Configuration for State and Bit Mode Control	378
20.5.8.3	Full-Duplex Communication (1-bit Mode Only)	381
20.5.8.4	Half-Duplex Communication (1/2/4-bit Mode)	382
20.5.8.5	DMA-Controlled Configurable Segmented Transfer	384
20.5.9	GP-SPI2 Works as a Slave	387
20.5.9.1	Communication Formats	388
20.5.9.2	Supported CMD Values in Half-Duplex Communication	389
20.5.9.3	Slave Single Transfer and Slave Segmented Transfer	391
20.5.9.4	Configuration of Slave Single Transfer	392
20.5.9.5	Configuration of Slave Segmented Transfer in Half-Duplex	392
20.5.9.6	Configuration of Slave Segmented Transfer in Full-Duplex	393
20.6	CS Setup Time and Hold Time Control	393
20.7	GP-SPI2 Clock Control	394
20.7.1	Clock Phase and Polarity	395
20.7.2	Clock Control in Master Mode	396
20.7.3	Clock Control in Slave Mode	397
20.8	GP-SPI2 Timing Compensation	397
20.9	Interrupts	397
20.10	Register Summary	400
20.11	Registers	401
<b>21</b>	<b>I2C Master Controller (I2C)</b>	<b>427</b>

21.1	Overview	427
21.2	Features	427
21.3	I2C Architecture	428
21.4	Functional Description	429
21.4.1	Clock Configuration	430
21.4.2	SCL and SDA Noise Filtering	430
21.4.3	Generating SCL Pulses in Idle State	430
21.4.4	Synchronization	430
21.4.5	Open-Drain Output	431
21.4.6	Timing Parameter Configuration	432
21.4.7	Timeout Control	433
21.4.8	Command Configuration	433
21.4.9	TX/RX RAM Data Storage	435
21.4.10	Data Conversion	436
21.4.11	Addressing Mode	436
21.4.12	Starting of the I2C Master Controller	436
21.5	Programming Example	436
21.5.1	I2C <sub>master</sub> Writes to I2C <sub>slave</sub> with a 7-bit Address in One Command Sequence	436
21.5.1.1	Introduction	437
21.5.1.2	Configuration Example	437
21.5.2	I2C <sub>master</sub> Writes to I2C <sub>slave</sub> with a 10-bit Address in One Command Sequence	438
21.5.2.1	Introduction	438
21.5.2.2	Configuration Example	439
21.5.3	I2C <sub>master</sub> Writes to I2C <sub>slave</sub> with Two 7-bit Addresses in One Command Sequence	440
21.5.3.1	Introduction	440
21.5.3.2	Configuration Example	440
21.5.4	I2C <sub>master</sub> Writes to I2C <sub>slave</sub> with a 7-bit Address in Multiple Command Sequences	442
21.5.4.1	Introduction	442
21.5.4.2	Configuration Example	443
21.5.5	I2C <sub>master</sub> Reads I2C <sub>slave</sub> with a 7-bit Address in One Command Sequence	444
21.5.5.1	Introduction	444
21.5.5.2	Configuration Example	445
21.5.6	I2C <sub>master</sub> Reads I2C <sub>slave</sub> with a 10-bit Address in One Command Sequence	446
21.5.6.1	Introduction	446
21.5.6.2	Configuration Example	446
21.5.7	I2C <sub>master</sub> Reads I2C <sub>slave</sub> with Two 7-bit Addresses in One Command Sequence	448
21.5.7.1	Introduction	448
21.5.7.2	Configuration Example	448
21.5.8	I2C <sub>master</sub> Reads I2C <sub>slave</sub> with a 7-bit Address in Multiple Command Sequences	450
21.5.8.1	Introduction	450
21.5.8.2	Configuration Example	451
21.6	Interrupts	452
21.7	Register Summary	454
21.8	Registers	456

## 22 LED PWM Controller (LEDC) 475

22.1	Overview	475
22.2	Features	475
22.3	Functional Description	477
22.3.1	Architecture	477
22.3.2	Timers	477
22.3.2.1	Clock Source	477
22.3.2.2	Clock Divider Configuration	478
22.3.2.3	14-bit Counter	479
22.3.3	PWM Generators	480
22.3.4	Duty Cycle Fading	481
22.3.5	Interrupts	482
22.4	Register Summary	483
22.5	Registers	485
<b>VI</b>	<b>Analog Signal Processing</b>	492
<b>23</b>	<b>On-Chip Sensor and Analog Signal Processing</b>	493
23.1	Overview	493
23.2	SAR ADC	493
23.2.1	Overview	493
23.2.2	Features	493
23.2.3	Functional Description	493
23.2.3.1	Input Signals	494
23.2.3.2	ADC Conversion and Attenuation	495
23.2.3.3	DIG ADC Controller	495
23.2.3.4	DIG ADC Clock	496
23.2.3.5	DIG ADC FSM	496
23.2.3.6	ADC Filters	499
23.2.3.7	Threshold Monitoring	499
23.3	Temperature Sensor	499
23.3.1	Overview	499
23.3.2	Features	499
23.3.3	Functional Description	500
23.4	Interrupts	500
23.5	Register Summary	500
23.6	Register	501
<b>VII</b>	<b>Appendix</b>	512
	<b>Related Documentation and Resources</b>	513
	<b>Glossary</b>	514
	Abbreviations for Peripherals	514
	Abbreviations Related to Registers	514
	Access Types for Registers	516

<b>Programming Reserved Register Field</b>	518
Introduction	518
Programming Reserved Register Field	518
<b>Interrupt Configuration Registers</b>	519
<b>Revision History</b>	520

## List of Tables

1.3-1	CPU Address Map	23
1.5-1	ID wise map of Interrupt Trap-Vector Addresses	34
1.7-1	NAPOT encoding for maddress	43
2.4-1	Selecting Peripherals via Register Configuration	53
2.4-2	Descriptor Field Alignment Requirements	56
3.3-1	Internal Memory Address Mapping	83
3.3-2	External Memory Address Mapping	84
3.3-3	Module/Peripheral Address Mapping	86
4.3-1	Parameters in BLOCK0	89
4.3-2	Parameters in BLOCK1 to BLOCK3	90
4.3-3	Registers information	94
4.3-4	Configuration of Default VDDQ Timing Parameters	95
5.8-1	Bits Used to Control IO MUX Functions in Light-sleep Mode	125
5.11-1	Peripheral Signals via GPIO Matrix	128
5.12-1	IO MUX Pin Functions	133
5.13-1	Analog Functions of IO MUX Pins	134
6.1-1	Reset Sources	148
6.2-1	CPU Clock Source	150
6.2-2	CPU Clock Frequency	150
6.2-3	Peripheral Clocks	152
6.2-4	APB_CLK Clock Frequency	153
6.2-5	CRYPTO_CLK Frequency	153
6.2-6	MSPI_CLK Frequency	153
7.3-1	Default Configuration of Strapping Pins	156
7.3-2	Boot Mode Control	156
7.3-3	ROM Code Printing Control	158
8.3-1	CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources	161
9.3-1	Low-power Clocks	177
9.3-2	The Triggering Conditions for the RTC Timer	177
9.5-1	Predefined Power Modes	181
9.5-2	Wakeup Source	182
9.5-3	Reject Source	183
10.4-1	UNIT <sub>n</sub> Configuration Bits	219
10.4-2	Trigger Point	220
10.4-3	Synchronization Operation	220
11.3-1	Alarm Generation When Up-Down Counter Increments	237
11.3-2	Alarm Generation When Up-Down Counter Decrements	237



12.2-1	Timeout Actions	258
13.3-1	Memory Controlling Bit	262
13.3-2	Clock Gating and Reset Bits	263
15.3-1	ECC Accelerator Memory Blocks	286
15.4-1	Choose ECC Accelerator Key Size	287
15.4-2	ECC Accelerator's Working Modes	288
16.3-1	SHA Accelerator Working Mode	295
16.3-2	SHA Hash Algorithm Selection	297
16.4-1	The Storage and Length of Message Digest from Different Algorithms	300
17.4-1	<i>Key</i> Generated Based on <i>Key<sub>A</sub></i> , <i>Key<sub>B</sub></i>	308
17.4-2	Mapping Between Offsets and Registers	309
19.5-1	UART <sub><i>n</i></sub> Synchronous Registers	334
19.5-2	UART <sub><i>n</i></sub> Static Registers	335
20.5-1	Data Modes Supported by GP-SPI2	364
20.5-2	Functional Description of FSPI Bus Signals	364
20.5-3	Signals Used in Various SPI Modes	365
20.5-4	Bit Order Control in GP-SPI2 Master and Slave Modes	367
20.5-5	Supported Transfers in Master and Slave Modes	368
20.5-6	Interrupt Trigger Condition on GP-SPI2 Data Transfer in Slave Mode	372
20.5-7	Registers Used for State Control in 1/2/4-bit Modes	378
20.5-7	Registers Used for State Control in 1/2/4-bit Modes	379
20.5-8	Sending Sequence of Command Value	380
20.5-9	Sending Sequence of Address Value	380
20.5-10	BM Table for CONF State	386
20.5-11	An Example of CONF buffer <sub><i>i</i></sub> in Segment <sub><i>i</i></sub>	386
20.5-12	BM Bit Value v.s. Register to Be Updated in This Example	387
20.5-13	Supported CMD Values in SPI Mode	390
20.5-13	Supported CMD Values in SPI Mode	391
20.5-14	Supported CMD Values in QPI Mode	391
20.7-1	Clock Phase and Polarity Configuration in Master Mode	396
20.7-2	Clock Phase and Polarity Configuration in Slave Mode	397
20.9-1	GP-SPI2 Master Mode Interrupts	398
20.9-2	GP-SPI2 Slave Mode Interrupts	399
21.4-1	I2C Registers that Need Synchronization	430
22.3-1	Commonly-used Frequencies and Resolutions	480
23.2-1	SAR ADC Input Signals	495
23.3-1	Temperature Offset	500
23.6-4	Configuration of ENA/RAW/ST Registers	519

## List of Figures

1.1-1	CPU Block Diagram	22
1.6-1	Debug System Overview	38
2.1-1	Modules with GDMA Feature and GDMA Channels	51
2.3-1	GDMA Engine Architecture	52
2.4-1	Structure of a Linked List	53
2.4-2	Relationship among Linked Lists	55
3.2-1	System Structure and Address Mapping	82
3.3-1	Cache Structure	85
4.3-1	Shift Register Circuit (former 32-byte)	92
4.3-2	Shift Register Circuit (latter 12-byte)	92
5.3-1	Architecture of IO MUX and GPIO Matrix	119
5.3-2	Internal Structure of a Pad	120
5.4-1	GPIO Input Synchronized on APB Clock Rising Edge or on Falling Edge	121
5.4-2	Filter Timing of GPIO Input Signals	122
6.1-1	Reset Types	147
6.2-1	System Clock	149
7.3-1	Chip Boot Flow	157
8.2-1	Interrupt Matrix Structure	160
9.3-1	Low-power Management Schematics	174
9.3-2	Power Management Unit Workflow	176
9.3-3	RTC_SLOW_CLK and RTC_FAST_CLK	177
9.3-4	Digital System Regulator	178
9.3-5	Low-power voltage regulator	179
9.4-1	Brownout detector	179
9.4-2	Brownout handling	180
10.1-1	System Timer Structure	217
10.4-1	System Timer Alarm Generation	218
11.1-1	Timer Group Overview	235
11.3-1	Timer Group Architecture	236
12.1-1	Watchdog Timers Overview	254
12.2-1	Digital Watchdog Timers in ESP8684	256
12.3-1	Super Watchdog Controller Structure	259
17.3-1	Architecture of the External Memory Encryption and Decryption	307
18.3-1	Noise Source	316

19.3-1	UART Architecture Overview	321
19.3-2	UART Architecture	322
19.4-1	UART Controllers Sharing RAM	324
19.4-2	UART Controllers Division	325
19.4-3	The Timing Diagram of Weak UART Signals Along Falling Edges	326
19.4-4	Structure of UART Data Frame	327
19.4-5	AT_CMD Character Structure	327
19.4-6	Driver Control Diagram in RS485 Mode	328
19.4-7	The Timing Diagram of Encoding and Decoding in SIR mode	329
19.4-8	IrDA Encoding and Decoding Diagram	330
19.4-9	Hardware Flow Control Diagram	331
19.4-10	Connection between Hardware Flow Control Signals	332
19.5-1	UART Programming Procedures	337
20.4-1	SPI Module Overview	363
20.5-1	Data Buffer Used in CPU-Controlled Transfer	368
20.5-2	GP-SPI2 Block Diagram	373
20.5-3	Data Flow Control in GP-SPI2 Master Mode	374
20.5-4	Data Flow Control in GP-SPI2 Slave Mode	375
20.5-5	GP-SPI2 State Machine in Master Mode	377
20.5-6	Full-Duplex Communication Between GP-SPI2 Master and a Slave	381
20.5-7	Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode	383
20.5-8	SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash	384
20.5-9	Configurable Segmented Transfer in DMA-Controlled Master Mode	384
20.6-1	Recommended CS Timing and Settings When Accessing External RAM	394
20.6-2	Recommended CS Timing and Settings When Accessing Flash	394
20.7-1	SPI Clock Mode 0 or 2	395
20.7-2	SPI Clock Mode 1 or 3	396
21.3-1	I2C Master Architecture	428
21.3-2	I2C Protocol Timing (Cited from Fig. 31 in <a href="#">The I2C-bus specification Version 2.1</a> )	429
21.3-3	I2C Timing Parameters (Cited from Table 5 in <a href="#">The I2C-bus specification Version 2.1</a> )	429
21.4-1	I2C Timing Diagram	432
21.4-2	Structure of I2C Command Registers	434
21.5-1	I2C <sub>master</sub> Writing to I2C <sub>slave</sub> with a 7-bit Address	437
21.5-2	I2C <sub>master</sub> Writing to a Slave with a 10-bit Address	438
21.5-3	I2C <sub>master</sub> Writing to I2C <sub>slave</sub> with Two 7-bit Addresses	440
21.5-4	I2C <sub>master</sub> Writing to I2C <sub>slave</sub> with a 7-bit Address in Multiple Sequences	442
21.5-5	I2C <sub>master</sub> Reading I2C <sub>slave</sub> with a 7-bit Address	444
21.5-6	I2C <sub>master</sub> Reading I2C <sub>slave</sub> with a 10-bit Address	446
21.5-7	I2C <sub>master</sub> Reading N Bytes of Data from addrM of I2C <sub>slave</sub> with a 7-bit Address	448
21.5-8	I2C <sub>master</sub> Reading I2C <sub>slave</sub> with a 7-bit Address in Segments	450
22.2-1	LED PWM Architecture	476
22.3-1	LED PWM Generator Diagram	477
22.3-2	Frequency Division When LEDC_CLK_DIV is a Non-Integer Value	479
22.3-3	LED_PWM Output Signal Diagram	481

22.3-4	Output Signal Diagram of Fading Duty Cycle	482
23.2-1	SAR ADC Function Overview	494
23.2-2	Diagram of DIG ADC FSM	496
23.2-3	APB_SARADC_SAR_PATT_TAB1_REG and Pattern Table Entry 0 - Entry 3	497
23.2-4	APB_SARADC_SAR_PATT_TAB2_REG and Pattern Table Entry 4 - Entry 7	497
23.2-5	Pattern Table Entry	498
23.2-6	cmd0 Configuration	498
23.2-7	cmd1 Configuration	498

# Part I

## Microprocessor and Master

This part covers the essential processing elements of the system, diving into the ESP-RISC-V CPU and GDMA Controller (GDMA). Details include CPU instruction extensions and controllers for Direct Memory Access (DMA).

# Chapter 1

## ESP-RISC-V CPU

### 1.1 Overview

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

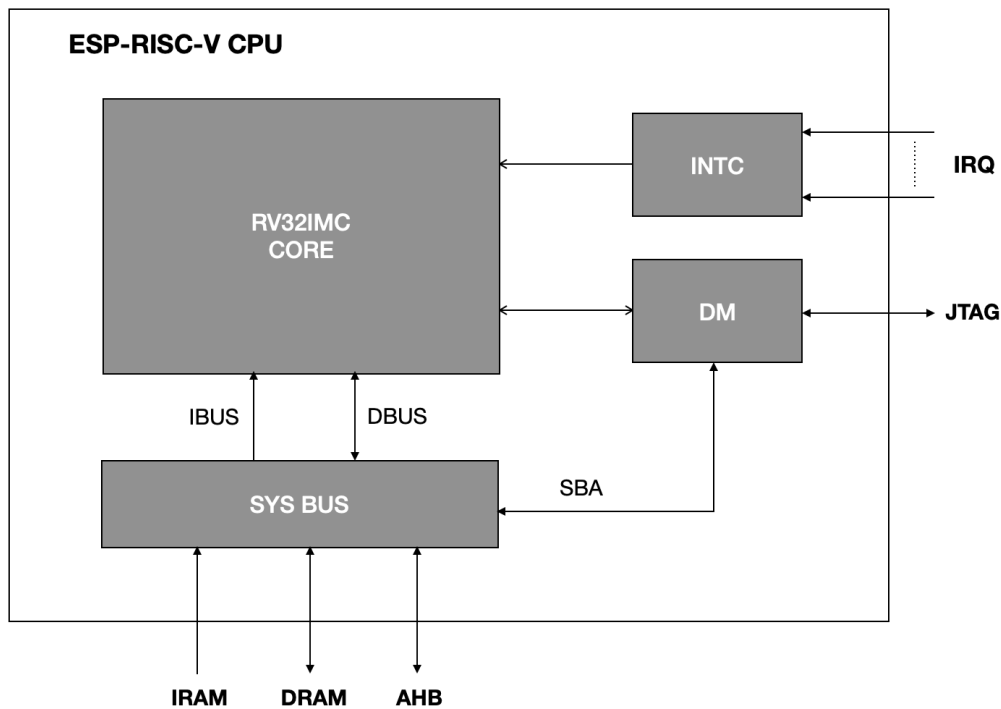


Figure 1.1-1. CPU Block Diagram

### 1.2 Features

The ESP-RISC-V CPU has the following features:

- Operating clock frequency up to 120 MHz
- Zero wait cycle access to on-chip SRAM and Cache for program and data access over IRAM/DRAM interface
- Interrupt controller (INTC) with up to 31 vectored interrupts with programmable priority and threshold levels

- Debug module (DM) compliant with RISC-V debug specification v0.13 with external debugger support over an industry-standard JTAG/USB port
- Debugger direct system bus access (SBA) to memory and peripherals
- Hardware trigger compliant to RISC-V debug specification v0.13 with up to 2 breakpoints/watchpoints
- Physical memory protection (PMP) for up to 16 regions
- 32-bit AHB system bus for peripheral access
- Configurable events for core performance metrics

## 1.3 Address Map

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

**Table 1.3-1. CPU Address Map**

Name	Description	Starting Address	Ending Address	Access
IRAM	Instruction Address Map	0x4000_0000	0x47FF_FFFF	R/W
DRAM	Data Address Map	0x3800_0000	0x3FFF_FFFF	R/W
DM	Debug Address Map	0x2000_0000	0x27FF_FFFF	R/W
AHB	AHB Address Map	*default	*default	R/W

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

## 1.4 Configuration and Status Registers (CSRs)

### 1.4.1 Register Summary

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

Name	Description	Address	Access
<b>Machine Information CSRs</b>			
<a href="#">mvendorid</a>	Machine Vendor ID	0xF11	RO
<a href="#">marchid</a>	Machine Architecture ID	0xF12	RO
<a href="#">mimpid</a>	Machine Implementation ID	0xF13	RO
<a href="#">mhartid</a>	Machine Hart ID	0xF14	RO
<b>Machine Trap Setup CSRs</b>			
<a href="#">mstatus</a>	Machine-Mode Status	0x300	R/W

Name	Description	Address	Access
<a href="#">misa</a> <sup>1</sup>	Machine ISA	0x301	R/W
<a href="#">mtvec</a> <sup>2</sup>	Machine Trap Vector	0x305	R/W
<b>Machine Trap Handling CSRs</b>			
<a href="#">mscratch</a>	Machine Scratch	0x340	R/W
<a href="#">mepc</a>	Machine Trap Program Counter	0x341	R/W
<a href="#">mcause</a> <sup>3</sup>	Machine Trap Cause	0x342	R/W
<a href="#">mtval</a>	Machine Trap Value	0x343	R/W
<b>Physical Memory Protection (PMP) CSRs</b>			
<a href="#">pmpcfg0</a>	Physical memory protection configuration	0x3A0	R/W
<a href="#">pmpcfg1</a>	Physical memory protection configuration	0x3A1	R/W
<a href="#">pmpcfg2</a>	Physical memory protection configuration	0x3A2	R/W
<a href="#">pmpcfg3</a>	Physical memory protection configuration	0x3A3	R/W
<a href="#">pmpaddr0</a>	Physical memory protection address	0x3B0	R/W
<a href="#">pmpaddr1</a>	Physical memory protection address	0x3B1	R/W
<a href="#">pmpaddr2</a>	Physical memory protection address	0x3B2	R/W
<a href="#">pmpaddr3</a>	Physical memory protection address	0x3B3	RO
<a href="#">pmpaddr4</a>	Physical memory protection address	0x3B4	RO
<a href="#">pmpaddr5</a>	Physical memory protection address	0x3B5	RO
<a href="#">pmpaddr6</a>	Physical memory protection address	0x3B6	RO
<a href="#">pmpaddr7</a>	Physical memory protection address	0x3B7	RO
<a href="#">pmpaddr8</a>	Physical memory protection address	0x3B8	RO
<a href="#">pmpaddr9</a>	Physical memory protection address	0x3B9	RO
<a href="#">pmpaddr10</a>	Physical memory protection address	0x3BA	RO
<a href="#">pmpaddr11</a>	Physical memory protection address	0x3BB	RO
<a href="#">pmpaddr12</a>	Physical memory protection address	0x3BC	RO
<a href="#">pmpaddr13</a>	Physical memory protection address	0x3BD	RO
<a href="#">pmpaddr14</a>	Physical memory protection address	0x3BE	RO
<a href="#">pmpaddr15</a>	Physical memory protection address	0x3BF	RO
<b>Trigger Module CSRs (shared with Debug Mode)</b>			
<a href="#">tselect</a>	Trigger Select Register	0x7A0	R/W
<a href="#">tdata1</a>	Trigger Abstract Data 1	0x7A1	R/W
<a href="#">tdata2</a>	Trigger Abstract Data 2	0x7A2	R/W
<a href="#">tcontrol</a>	Global Trigger Control	0x7A5	R/W
<b>Debug Mode CSRs</b>			
<a href="#">dcsr</a>	Debug Control and Status	0x7B0	R/W
<a href="#">dpc</a>	Debug PC	0x7B1	R/W
<a href="#">dscratch0</a>	Debug Scratch Register 0	0x7B2	R/W
<a href="#">dscratch1</a>	Debug Scratch Register 1	0x7B3	R/W
<b>Performance Counter CSRs (Custom) <sup>4</sup></b>			

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

<sup>2</sup>[mtvec](#) only provides configuration for trap handling in vectored mode with the base address aligned to 256 bytes

<sup>3</sup>External interrupt IDs reflected in [mcause](#) include even those IDs which have been reserved by RISC-V standard for core internal sources.

<sup>4</sup>These custom CSRs have been implemented in the address space reserved by RISC-V standard for custom use

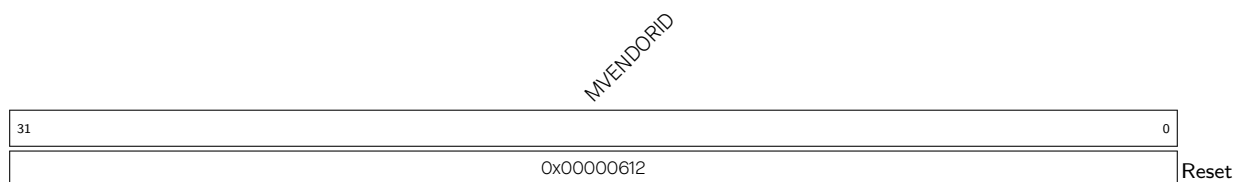


Name	Description	Address	Access
<a href="#">mpcer</a>	Machine Performance Counter Event	0x7E0	R/W
<a href="#">mpcmr</a>	Machine Performance Counter Mode	0x7E1	R/W
<a href="#">mpccr</a>	Machine Performance Counter Count	0x7E2	R/W
<b>GPIO Access CSRs (Custom)</b>			
<a href="#">cpu_gpio_oen</a>	GPIO Output Enable	0x803	R/W
<a href="#">cpu_gpio_in</a>	GPIO Input Value	0x804	RO
<a href="#">cpu_gpio_out</a>	GPIO Output Value	0x805	R/W

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

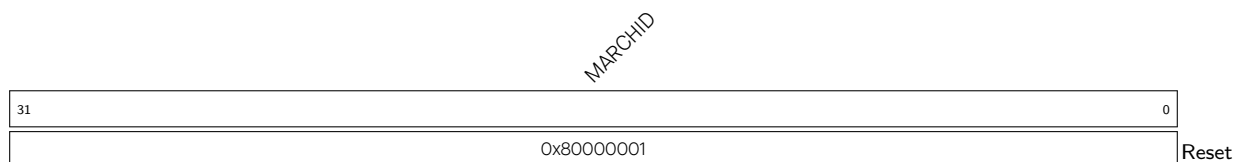
## 1.4.2 Register Description

Register 1.1. mvendorid (0xF11)



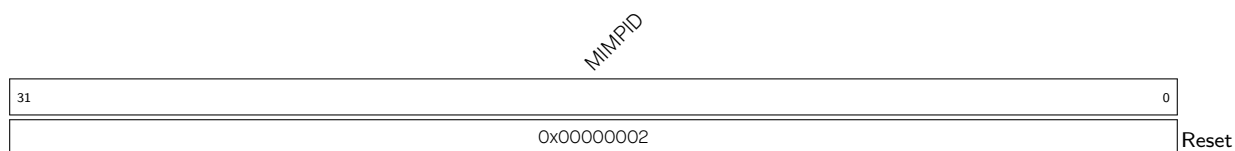
**MVENDORID** Vendor ID. (RO)

Register 1.2. marchid (0xF12)



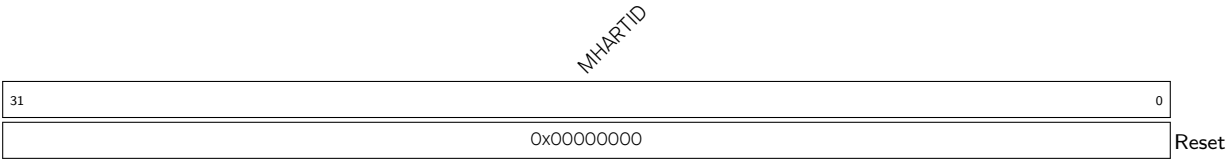
**MARCHID** Architecture ID. (RO)

Register 1.3. mimpid (0xF13)



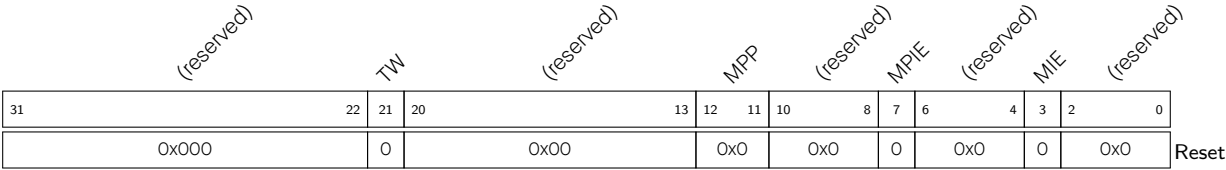
**MIMPID** Implementation ID. (RO)

Register 1.4. mhartid (0xF14)



**MHARTID** Hart ID. (RO)

Register 1.5. mstatus (0x300)



**MIE** Global machine-mode interrupt enable. (R/W)

**MPIE** Previous MIE. (R/W)

**MPP** Machine previous privilege mode. (R/W)

Possible values:

- 0x0: User-mode
- 0x3: Machine-mode

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

**TW** Timeout wait. (R/W)

If this bit is set, executing WFI (Wait-for-Interrupt) instruction in User-mode will cause illegal instruction exception.

**Register 1.6. misa (0x301)**

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

Reset

**MXL** Machine XLEN = 1 (32-bit). (RO)**Z** Reserved = 0. (RO)**Y** Reserved = 0. (RO)**X** Non-standard extensions present = 0. (RO)**W** Reserved = 0. (RO)**V** Reserved = 0. (RO)**U** User-mode implemented = 1. (RO)**T** Reserved = 0. (RO)**S** Supervisor-mode implemented = 0. (RO)**R** Reserved = 0. (RO)**Q** Quad-precision floating-point extension = 0. (RO)**P** Reserved = 0. (RO)**O** Reserved = 0. (RO)**N** User-level interrupts supported = 0. (RO)**M** Integer Multiply/Divide extension = 1. (RO)**L** Reserved = 0. (RO)**K** Reserved = 0. (RO)**J** Reserved = 0. (RO)**I** RV32I base ISA = 1. (RO)**H** Hypervisor extension = 0. (RO)**G** Additional standard extensions present = 0. (RO)**F** Single-precision floating-point extension = 0. (RO)**E** RV32E base ISA = 0. (RO)**D** Double-precision floating-point extension = 0. (RO)**C** Compressed Extension = 1. (RO)**B** Reserved = 0. (RO)**A** Atomic Extension = 0. (RO)

**Register 1.7. mtvec (0x305)**

BASE								(reserved)				MODE		
31						8	7			2	1	0		
0x0000000								0x00				0x1		Reset

**MODE** Only vectored mode **0x1** is available. (RO)

**BASE** Higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

**Register 1.8. mscratch (0x340)**

MSCRATCH																															
31																															0
0x00000000																															
Reset																															

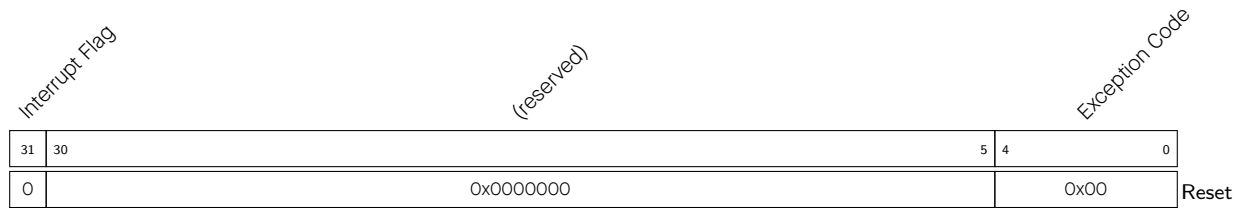
**MSCRATCH** Machine scratch register for custom use. (R/W)

**Register 1.9. mepc (0x341)**

MEPC																															
310																															
0x00000000Reset																															

**MEPC** Machine trap/exception program counter. (R/W)

This is automatically updated with address of the instruction which was about to be executed while CPU encountered the most recent trap.

**Register 1.10. mcause (0x342)**

**Exception Code** This field is automatically updated with unique ID of the most recent exception or interrupt due to which CPU entered trap. (R/W)

Possible exception IDs are:

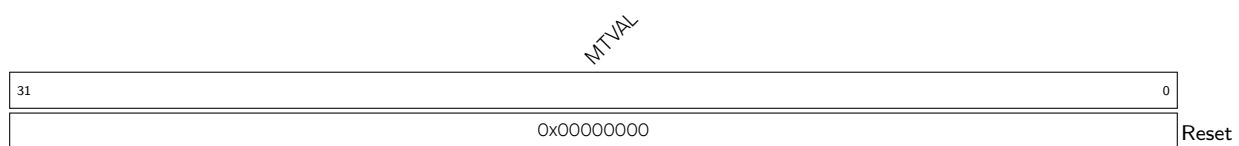
- 0x1: PMP Instruction access fault
- 0x2: Illegal Instruction
- 0x3: Hardware Breakpoint/Watchpoint or EBREAK
- 0x5: PMP Load access fault
- 0x7: PMP Store access fault
- 0x8: ECALL from U mode
- 0xb: ECALL from M mode

*Note: Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.*

**Interrupt Flag** This flag is automatically updated when CPU enters trap. (R/W)

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

*Note: The interrupt controller is using up IDs in range 1-31 for all external interrupt sources. This is different from the RISC-V standard which has reserved IDs in range 0-15 for core internal interrupt sources.*

**Register 1.11. mtval (0x343)**

**MTVAL** Machine trap value. (R/W)

This is automatically updated with an exception dependent data which may be useful for handling that exception.

Data is to be interpreted depending upon exception IDs:

- 0x1: Faulting virtual address of instruction
- 0x2: Faulting instruction opcode
- 0x5: Faulting data address of load operation
- 0x7: Faulting data address of store operation

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

**Register 1.12. mpcer (0x7E0)**

(reserved)											INST_COMP (BRANCH_TAKEN BRANCH JMP_UNCOND STORE LOAD IDLE JMP_HAZARD LD_HAZARD INST CYCLE												
31											11	10	9	8	7	6	5	4	3	2	1	0	
0x000												0	0	0	0	0	0	0	0	0	0	0	Reset

**INST\_COMP** Count Compressed Instructions. (R/W)**BRANCH\_TAKEN** Count Branches Taken. (R/W)**BRANCH** Count Branches. (R/W)**JMP\_UNCOND** Count Unconditional Jumps. (R/W)**STORE** Count Stores. (R/W)**LOAD** Count Loads. (R/W)**IDLE** Count IDLE Cycles. (R/W)**JMP\_HAZARD** Count Jump Hazards. (R/W)**LD\_HAZARD** Count Load Hazards. (R/W)**INST** Count Instructions. (R/W)**CYCLE** Count Clock Cycles. Cycle count does not increment during WFI mode. (R/W)

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

**Register 1.13. mpcmr (0x7E1)**

(reserved)										COUNT_SAT COUNT_EN			
31										2	1	0	
0											1	1	Reset

**COUNT\_SAT** Counter Saturation Control. (R/W)

Possible values:

- 0: Overflow on maximum value
- 1: Halt on maximum value

**COUNT\_EN** Counter Enable Control. (R/W)

Possible values:

- 0: Disabled
- 1: Enabled

**Register 1.14. mpccr (0x7E2)**

MPCCR																															
31																														0	
0x00000000																															
Reset																															

**MPCCR** Machine Performance Counter Value. (R/W)

**Register 1.15. cpu\_gpio\_oen (0x803)**

(reserved)																CPU_GPIO_OEN[7] CPU_GPIO_OEN[6] CPU_GPIO_OEN[5] CPU_GPIO_OEN[4] CPU_GPIO_OEN[3] CPU_GPIO_OEN[2] CPU_GPIO_OEN[1] CPU_GPIO_OEN[0]										
31																8	7	6	5	4	3	2	1	0		
0																	0	0	0	0	0	0	0	0	0	Reset

**CPU\_GPIO\_OEN** GPIO<sub>n</sub> (n=0 ~ 21) Output Enable. CPU\_GPIO\_OEN[7:0] correspond to output enable signals cpu\_gpio\_out\_oen[7:0] in Table 5.11-1 *Peripheral Signals via GPIO Matrix*.

CPU\_GPIO\_OEN value matches that of cpu\_gpio\_out\_oen.

CPU\_GPIO\_OEN is the enable signal of [CPU\\_GPIO\\_OUT](#). (R/W)

- 0: GPIO output disable
- 1: GPIO output enable

**Register 1.16. cpu\_gpio\_in (0x804)**

(reserved)								CPU_GPIO_IN[7] CPU_GPIO_IN[6] CPU_GPIO_IN[5] CPU_GPIO_IN[4] CPU_GPIO_IN[3] CPU_GPIO_IN[2] CPU_GPIO_IN[1] CPU_GPIO_IN[0]										
31								8	7	6	5	4	3	2	1	0		
0									0	0	0	0	0	0	0	0	0	Reset

**CPU\_GPIO\_IN** GPIO<sub>n</sub> (n=0 ~ 21) Input Value. It is a CPU CSR to read input value (1=high, 0=low) from SoC GPIO pin.

CPU\_GPIO\_IN[7:0] correspond to input signals cpu\_gpio\_in[7:0] in Table 5.11-1 *Peripheral Signals via GPIO Matrix*.

CPU\_GPIO\_IN[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section ?? in Chapter [IO MUX and GPIO Matrix \(GPIO, IO MUX\)](#). (RO)

Register 1.17. cpu\_gpio\_out (0x805)

(reserved)								CPU_GPIO_OUT[7] CPU_GPIO_OUT[6] CPU_GPIO_OUT[5] CPU_GPIO_OUT[4] CPU_GPIO_OUT[3] CPU_GPIO_OUT[2] CPU_GPIO_OUT[1] CPU_GPIO_OUT[0]									
31								8	7	6	5	4	3	2	1	0	
0									0	0	0	0	0	0	0	0	Reset

**CPU\_GPIO\_OUT** GPIO<sub>n</sub> (n=0 ~ 21) Output Value. It is a CPU CSR to write value (1=high, 0=low) to SoC GPIO pin. The value takes effect only when [CPU\\_GPIO\\_OEN](#) is set.

CPU\_GPIO\_OUT[7:0] correspond to output signals cpu\_gpio\_out[7:0] in Table 5.11-1 *Peripheral Signals via GPIO Matrix*.

CPU\_GPIO\_OUT[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section ?? in Chapter *IO MUX and GPIO Matrix (GPIO, IO MUX)*. (R/W)



## 1.5 Interrupt Controller

### 1.5.1 Features

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

- Up to 31 asynchronous interrupts with unique IDs (1-31)
- Configurable via read/write to memory mapped registers
- 15 levels of priority, programmable for each interrupt
- Support for both level and edge type interrupt sources
- Programmable global threshold for masking interrupts with lower priority
- Interrupts IDs mapped to trap-vector address offsets

### 1.5.2 Functional Description

Each interrupt ID has 5 properties associated with it:

1. Enable State (0-1):
  - Determines if an interrupt is enabled to be captured and serviced by the CPU.
  - Programmed by writing the corresponding bit in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_ENABLE\\_REG](#).
2. Type (0-1):
  - Enables latching the state of an interrupt signal on its rising edge.
  - Programmed by writing the corresponding bit in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_TYPE\\_REG](#).
  - An interrupt for which type is kept 0 is referred as a 'level' type interrupt.
  - An interrupt for which type is set to 1 is referred as an 'edge' type interrupt.
3. Priority (1-15):
  - Determines which interrupt, among multiple pending interrupts, the CPU will service first.
  - Programmed by writing to the [INTERRUPT\\_CORE0\\_CPU\\_INT\\_PRI\\_n\\_REG](#) for a particular interrupt ID *n* in range (1-31).
  - Enabled interrupts with priorities zero or less than the threshold value in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_THRESH\\_REG](#) are masked.
  - Priority levels increase from 1 (lowest) to 15 (highest).
  - Interrupts with same priority are statically prioritized by their IDs, lowest ID having highest priority.
4. Pending State (0-1):
  - Reflects the captured state of an enabled and unmasked interrupt signal.
  - For each interrupt ID, the corresponding bit in read-only [INTERRUPT\\_CORE0\\_CPU\\_INT\\_EIP\\_STATUS\\_REG](#) gives its pending state.

- A pending interrupt will cause CPU to enter trap if no other pending interrupt has higher priority.
- A pending interrupt is said to be 'claimed' if it preempts the CPU and causes it to jump to the corresponding trap vector address.
- All pending interrupts which are yet to be serviced are termed as 'unclaimed'.

#### 5. Clear State (0-1):

- Toggling this will clear the pending state of claimed edge-type interrupts only.
- Toggled by first setting and then clearing the corresponding bit in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_CLEAR\\_REG](#).
- Pending state of a level type interrupt is unaffected by this and must be cleared from source.
- Pending state of an unclaimed edge type interrupt can be flushed, if required, by first clearing the corresponding bit in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_ENABLE\\_REG](#) and then toggling same bit in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_CLEAR\\_REG](#).

When CPU services a pending interrupt, it:

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

Table 1.5-1 shows the mapping of each interrupt ID with the corresponding trap-vector address. In short, the word aligned trap address for an interrupt with a certain  $ID = i$  can be calculated as  $(mtvec + 4i)$ .

*Note:  $ID = 0$  is unavailable and therefore cannot be used for capturing interrupts. This is because the corresponding trap vector address  $(mtvec + 0x00)$  is reserved for exceptions.*

**Table 1.5-1. ID wise map of Interrupt Trap-Vector Addresses**

ID	Address	ID	Address	ID	Address	ID	Address
0	NA	8	$mtvec + 0x20$	16	$mtvec + 0x40$	24	$mtvec + 0x60$
1	$mtvec + 0x04$	9	$mtvec + 0x24$	17	$mtvec + 0x44$	25	$mtvec + 0x64$
2	$mtvec + 0x08$	10	$mtvec + 0x28$	18	$mtvec + 0x48$	26	$mtvec + 0x68$
3	$mtvec + 0x0c$	11	$mtvec + 0x2c$	19	$mtvec + 0x4c$	27	$mtvec + 0x6c$
4	$mtvec + 0x10$	12	$mtvec + 0x30$	20	$mtvec + 0x50$	28	$mtvec + 0x70$
5	$mtvec + 0x14$	13	$mtvec + 0x34$	21	$mtvec + 0x54$	29	$mtvec + 0x74$
6	$mtvec + 0x18$	14	$mtvec + 0x38$	22	$mtvec + 0x58$	30	$mtvec + 0x78$
7	$mtvec + 0x1c$	15	$mtvec + 0x3c$	23	$mtvec + 0x5c$	31	$mtvec + 0x7c$

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

Upon execution of MRET instruction, the CPU:

- copies the state of [MPIE](#) back into [MIE](#), and subsequently clears [MPIE](#). This means that if previously [MPIE](#) was set, then, after MRET, [MIE](#) will be set, thereby enabling interrupts globally.

- jumps to the address stored in [mepc](#) and resumes execution.

It is possible to perform software assisted nesting of interrupts inside an ISR as explained in [1.5.3](#).

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

- Only if an interrupt has non-zero priority, higher or equal to the value in the threshold register, will it be reflected in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_EIP\\_STATUS\\_REG](#).
- If an interrupt is visible in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_EIP\\_STATUS\\_REG](#) and has yet to be serviced, then it's possible to mask it (and thereby prevent the CPU from servicing it) by either lowering the value of its priority or increasing the global threshold.
- If an interrupt, visible in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_EIP\\_STATUS\\_REG](#), is to be flushed (and prevented from being serviced at all), then it must be disabled (and cleared if it is of edge type).

## 1.5.3 Suggested Operation

### 1.5.3.1 Latency Aspects

There is latency involved while configuring the Interrupt Controller.

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

Whenever any of its registers are modified, the Interrupt Controller enters into transient state, which may take up to 4 cycles for it to settle down into steady state again. During this transient state, the ordering of interrupts may not be predictable, and therefore, a few safety measures need to be taken in software to avoid any synchronization issues.

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

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

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

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

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

### 1.5.3.2 Configuration Procedure

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

done.

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

1. save the state of [MIE](#) and clear [MIE](#) to 0
2. depending upon the type of the interrupt (edge/level), set/unset the  $n$ th bit of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_TYPE\\_REG](#)
3. set the priority by writing a value to [INTERRUPT\\_CORE0\\_CPU\\_INT\\_PRI\\_n\\_REG](#) in range 1 (lowest) to 15 (highest)
4. set the  $n$ th bit of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_ENABLE\\_REG](#)
5. execute FENCE instruction
6. restore the state of [MIE](#)

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

Upon entering into an ISR, software must toggle the  $n$ th bit of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_CLEAR\\_REG](#) if the interrupt is of edge type, or clear the source of the interrupt if it is of level type.

Software may also update the value of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_THRESH\\_REG](#) and program [MIE](#)=1 for allowing higher priority interrupts to preempt the current ISR (nesting), however, before doing so, all the state CSRs must be saved ([mepc](#), [mstatus](#), [mcause](#), etc.) since they will get overwritten due to occurrence of such an interrupt. Later, when exiting the ISR, the values of these CSRs must be restored.

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

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

1. save the state of [MIE](#) and clear [MIE](#) to 0
2. check if the interrupt is pending in [INTERRUPT\\_CORE0\\_CPU\\_INT\\_EIP\\_STATUS\\_REG](#)
3. set/unset the  $n$ th bit of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_ENABLE\\_REG](#)
4. if the interrupt is of edge type and was found to be pending in step 2 above,  $n$ th bit of [INTERRUPT\\_CORE0\\_CPU\\_INT\\_CLEAR\\_REG](#) must be toggled, so that its pending status gets flushed
5. execute FENCE instruction
6. restore the state of [MIE](#)

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

## 1.5.4 Register Summary

The addresses in this section are relative to Interrupt Controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

For the complete list of interrupt registers and detailed configuration information, please refer to Chapter 8 *Interrupt Matrix (INTMTRX)*, section 8.4, register group "CPU Interrupt Registers".

### 1.5.5 Register Description

The addresses in this section are relative to Interrupt Controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

For the complete list of interrupt registers and detailed configuration information, please refer to Chapter 8 *Interrupt Matrix (INTMTRX)*, section 8.4, register group "CPU interrupt register".

## 1.6 Debug

### 1.6.1 Overview

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

Figure 1.6-1 below shows the main components of External Debug Support.

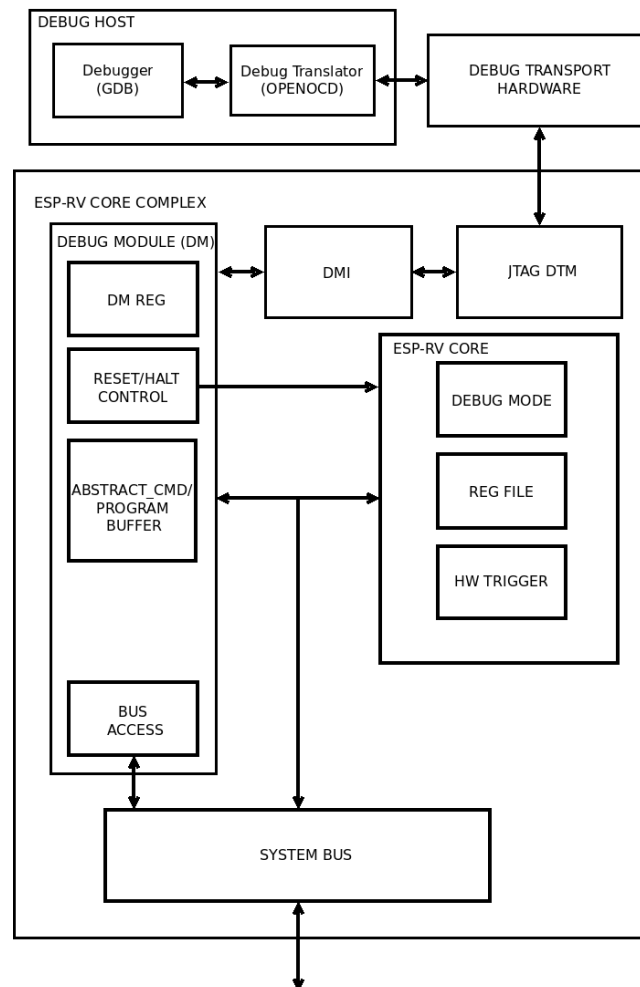


Figure 1.6-1. Debug System Overview

The user interacts with the Debug Host (e.g., laptop), which is running a debugger (e.g., gdb). The debugger communicates with a Debug Translator (e.g., OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (e.g., Olimex USB-JTAG adapter). The Debug Transport Hardware connects the Debug Host to the ESP-RV Core's Debug Transport Module (DTM) through standard JTAG interface. The DTM provides access to the Debug Module (DM) using the Debug Module Interface (DMI).

The DM allows the debugger to halt the core. Abstract commands provide access to its GPRs (general purpose registers). The Program Buffer allows the debugger to execute arbitrary code on the core, which allows access to additional CPU core state. Alternatively, additional abstract commands can provide access to additional CPU core state. ESP-RV core contains Trigger Module supporting two triggers. When trigger conditions are met, cores will halt spontaneously and inform the debug module that they have halted.

System bus access block allows memory and peripheral register access without using RISC-V core.

## 1.6.2 Features

Basic debug functionality has the following features:

- Halt and resume CPU core
- Access to CSR and GPR
- Debug from the first instruction after reset
- Core reset control
- Software breakpoint
- Hardware single-stepping
- 16-word program buffer
- System bus access
- Support for two hardware triggers

## 1.6.3 Functional Description

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

## 1.6.4 Register Summary

Below is the list of Debug CSR's supported by ESP-RV core.

Name	Description	Address	Access
<a href="#">dcsr</a>	Debug Control and Status	0x7B0	R/W
<a href="#">dpc</a>	Debug PC	0x7B1	R/W
<a href="#">dscratch0</a>	Debug Scratch Register 0	0x7B2	R/W
<a href="#">dscratch1</a>	Debug Scratch Register 1	0x7B3	R/W

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

## 1.6.5 Register Description

Below are the details of Debug CSR's supported by ESP-RV core.

**Register 1.18. dcsr (0x7B0)**

xdebugver				reserved				ebreakm		reserved		ebreaku		reserved		stopcount		stoptime		cause		reserved		step	prv	
31		28		27				16		15	14	13	12	11	10	9	8			6	5		3	2	1	0
4				0				0	0	0	0	0	0	0	0	0	0		0		0	0	0	0	Reset	

Reset

**xdebugver** Debug version. (RO)

- 4: External debug support exists

**ebreakm** When 1, ebreak instructions in Machine-Mode enter Debug Mode. (R/W)

**ebreaku** When 1, ebreak instructions in User/Application-Mode enter Debug Mode. (R/W)

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

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

**cause** Explains why Debug Mode was entered. When there are multiple reasons to enter Debug Mode in a single cycle, the cause with the highest priority number is the one written.

1. An ebreak instruction was executed. (priority 3)
2. The Trigger Module caused a halt. (priority 4)
3. haltreq was set. (priority 2)
4. The CPU core single stepped because step was set. (priority 1)

Other values are reserved for future use. (RO)

**step** When set and not in Debug Mode, the core will only execute a single instruction and then enter Debug Mode. Interrupts are **enabled\*** when this bit is set. If the instruction does not complete due to an exception, the core will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set. (R/W)

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

**\*Note:** Different from RISC-V Debug specification 0.13

**Register 1.19. dpc (0x7B1)**

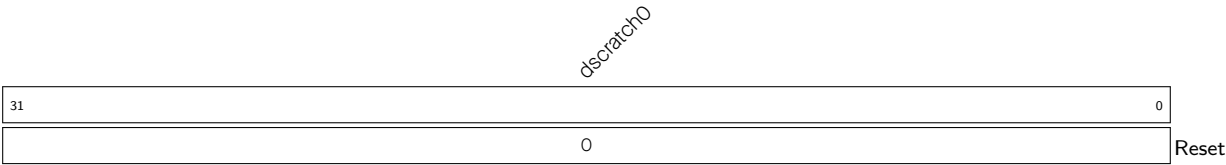
dpc																													
31																													0
0																													

Reset

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

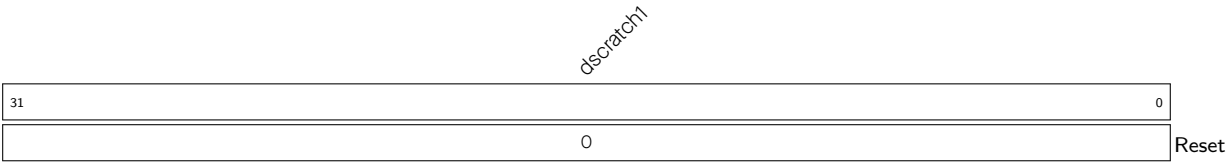


Register 1.20. dscratch0 (0x7B2)



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

Register 1.21. dscratch1 (0x7B3)



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

## 1.7 Hardware Trigger

### 1.7.1 Features

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

- Two independent trigger units
- Matching the address of program counter or load-store accesses
- Execution preemption by causing breakpoint exception
- Halting execution and transferring control to debugger
- Support for NAPOT (naturally aligned power of two) address encoding

### 1.7.2 Functional Description

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

To choose a particular trigger unit write the index (0-7) of that unit into [tselect](#) CSR. When [tselect](#) is written with a valid index, the abstract CSRs [tdata1](#) and [tdata2](#) are automatically mapped to reflect internal registers of that trigger unit. Each trigger unit has two internal registers, namely [mcontrol](#) and [maddress](#), which are mapped to [tdata1](#) and [tdata2](#), respectively.

Writing larger than allowed indexes to [tselect](#) will clip the written value to the largest valid index, which can be read back. This property may be used for enumerating the number of available triggers during initialization or when using a debugger.

Since software or debugger may need to know the type of the selected trigger to correctly interpret [tdata1](#) and [tdata2](#), the 4 bits (31-28) of [tdata1](#) encodes the type of the selected trigger. This type field is read-only and always provides a value of 0x2 for every trigger, which stands for match type trigger, hence, it is inferred that [tdata1](#) and [tdata2](#) are to be interpreted as [mcontrol](#) and [maddress](#). The information regarding other possible values can be found in the RISC-V Debug Specification v0.13, but this trigger module only supports type 0x2.

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

Each trigger unit can be configured to either cause breakpoint exception or enter debug mode, by writing to the action bit of [mcontrol](#). This bit can only be written from debugger, thus by default a trigger, if enabled, will cause breakpoint exception.

[mcontrol](#) for each trigger unit has a [hit](#) bit which may be read, after CPU halts or enters exception, to find out if this was the trigger unit that fired. This bit is set as soon as the corresponding trigger fires, but it has to be manually cleared before resuming operation. Although, failing to clear it doesn't affect normal execution in any way.

Each trigger unit only supports match on address, although this address could either be that of a load/store access or the virtual address of an instruction. The address and size of a region are specified by writing to maddress (tdata2) CSR for the selected trigger unit. Larger than 1 byte region sizes are specified through NAPOT (naturally aligned power of two) encoding (see Table 1.7-1) and enabled by setting match bit in mcontrol. Note that for NAPOT encoded addresses, by definition, the start address is constrained to be aligned to (i.e., an integer multiple of) the region size.

**Table 1.7-1. NAPOT encoding for maddress**

maddress (31-0)	Start Address	Size (bytes)
aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaaa01	aaa...aaaaaaaaa00	4
aaa...aaaaaaaaa011	aaa...aaaaaaaaa000	8
aaa...aaaaaaaaa0111	aaa...aaaaaaaaa0000	16
....		
a01...111111111	a00...000000000	$2^{31}$

tcontrol CSR is common to all trigger units. It is used for preventing triggers from causing repeated exceptions in machine-mode while execution is happening inside a trap handler. This also disables breakpoint exceptions inside ISRs by default, although, it is possible to manually enable this right before entering an ISR, for debugging purposes. This CSR is not relevant if a trigger is configured to enter debug mode.

### 1.7.3 Trigger Execution Flow

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

- **dpc** is set to current PC (in decode stage)
- cause field in **dcsr** is set to 2, which means halt due to trigger
- **hit** bit is set to 1, corresponding to the trigger(s) which fired

When hart goes into trap due to the firing of a trigger (**action** = 0) :

- **mepc** is set to current PC (in decode stage)
- **mcause** is set to 3, which means breakpoint exception
- **mpte** is set to the value in **mte** right before trap
- **mte** is set to 0
- **hit** bit is set to 1, corresponding to the trigger(s) which fired

*Note : If two different triggers fire at the same time, one with action = 0 and another with action = 1, then hart is halted and enters debug mode.*

### 1.7.4 Register Summary

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

Name	Description	Address	Access
<a href="#">tselect</a>	Trigger Select Register	0x7A0	R/W
<a href="#">tdata1</a>	Trigger Abstract Data 1	0x7A1	R/W
<a href="#">tdata2</a>	Trigger Abstract Data 2	0x7A2	R/W
<a href="#">tcontrol</a>	Global Trigger Control	0x7A5	R/W

## 1.7.5 Register Description

### Register 1.22. tselect (0x7A0)

(reserved)																															tselect				
31																															3	2	0		
0x00000000																															0x0			Reset	

**tselect** Index (0-7) of the selected trigger unit. (R/W)

### Register 1.23. tdata1 (0x7A1)

type		dmode		data	
31	28	27	26	0	
0x2		0	0x3e00000		Reset

**type** Type of trigger. (RO)

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

**dmode** This is set to 1 if a trigger is being used by the debugger. (R/W \*)

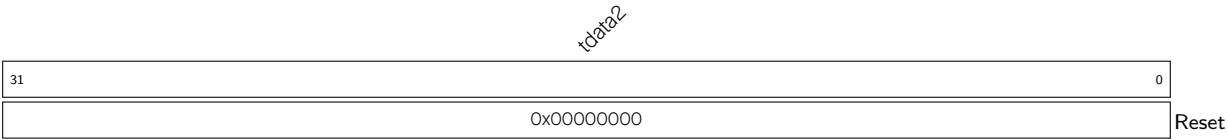
- 0: Both Debug and machine-mode can write the tdata1 and tdata2 registers at the selected tselect.
- 1: Only Debug Mode can write the tdata1 and tdata2 registers at the selected tselect. Writes from other modes are ignored.

\* Note : Only writable from debug mode.

**data** Abstract tdata1 content. (R/W)

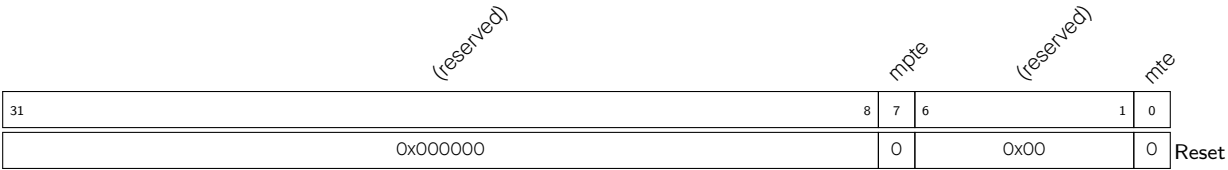
This will always be interpreted as fields of [mcontrol](#) since only match type (0x2) triggers are supported.

Register 1.24. tdata2 (0x7A2)



**tdata2** Abstract tdata2 content. (R/W)  
This will always be interpreted as [maddress](#) since only match type (0x2) triggers are supported.

Register 1.25. tcontrol (0x7A5)



**mpte** Machine-mode previous trigger enable bit. (R/W)

- When CPU is taking a machine-mode trap, the value of [mte](#) is automatically pushed into this.
- When CPU is executing MRET, its value is popped back into [mte](#), so this becomes 0.

**mte** Machine-mode trigger enable bit. (R/W)

- When CPU is taking a machine-mode trap, its value is automatically pushed into [mpte](#), so this becomes 0 and triggers with [action](#)=0 are disabled globally.
- When CPU is executing MRET, the value of [mpte](#) is automatically popped back into this.

### Register 1.26. mcontrol (0x7A1)

[illegible]

**dmode** Same as **dmode** in **tdata1**.

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

**action** Write this for configuring the selected trigger to perform one of the available actions when firing. (R/W)  
Valid options are:

- 0x0: cause breakpoint exception.
- 0x1: enter debug mode (only valid when dmode = 1)

*Note : Writing an invalid value will set this to the default value 0x0.*

**match** Write this for configuring the selected trigger to perform one of the available matching operations on a data/instruction address. (R/W) Valid options are:

- 0x0: exact byte match, i.e., address corresponding to one of the bytes in an access must match the value of `maddress` exactly.
- 0x1: NAPOT match, i.e., at least one of the bytes of an access must lie in the NAPOT region specified in `maddress`.

*Note : Writing a larger value will clip it to the largest possible value 0x1.*

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

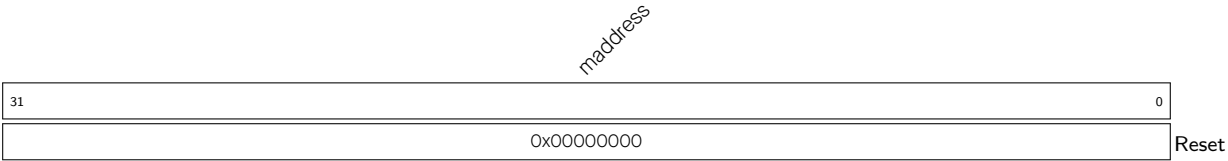
**u** Set this for enabling selected trigger to operate in user-mode. (R/W)

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

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

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

Register 1.27. maddress (0x7A2)



**maddress** Address used by the selected trigger when performing match operation. (R/W)  
This is decoded as NAPOT when [match](#)=1 in [mcontrol](#).

## 1.8 Memory Protection

### 1.8.1 Overview

The CPU core includes a physical memory protection unit, which can be used by software to set memory access privileges (read, write and execute permissions) for required memory regions. It supports 16 memory regions, of which some address regions have been hard coded to values in accordance with ESP8684 memory map and the rest of address regions are kept programmable to split SRAM into separate IRAM/DRAM regions as per software code size.

It is fully compliant to the Physical Memory Protection (PMP) description specified in RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. However, in order to save area, values of 13 **pmpaddrX** registers (refer [Register Summary](#)) have been hard-coded. Details are provided in next sub-section.

For detailed understanding of the RISC-V PMP concept, please refer to RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10.

### 1.8.2 Features

The memory protection unit has the following features:

- Support for 16 PMP entries
- Programmable **pmpaddr0-2**
- Hard-coded **pmpaddr3-15** as per ESP8684 memory map

### 1.8.3 Functional Description

Software can program the PMP unit's configuration and address registers in order to contain faults and support secure execution. PMP CSR's can only be programmed in machine-mode. Once enabled, write, read and execute permission checks are applied to all the accesses in user-mode as per programmed values of enabled **pmcfgX** and **pmpaddrX** registers.

By default, PMP grants permission to all accesses in machine-mode and revokes permission of all access in user-mode. This implies that it is mandatory to program address range and valid permissions in **pmcfgX** and **pmpaddr** registers for any valid access to pass through in user-mode. However, it is not required for machine-mode as PMP permits all accesses to go through by default. In cases where PMP checks are also required in machine-mode, software can set the lock bit of required PMP entry to enable permission checks on it. Once lock bit is set, it can only be cleared through CPU reset.

When any instruction is being fetched from memory region without execute permissions, exception is generated at processor level and exception cause is set as instruction access fault in **mcause** CSR. Similarly, any load/store access without valid read/write permissions, will result in exception generation with **mcause** updated as load access and store access fault respectively. In case of load/store access faults, violating address is captured in **mtval** CSR.

### 1.8.4 Register Summary

Below is a list of PMP CSRs supported by the CPU. These are only accessible from machine-mode. As mentioned earlier, **pmpaddrX0-2** are kept programmable to split the SRAM region as per software



requirements. **pmpaddrX3-15** are hard coded with values specified in "CSR Reset Value" column. These hard-coded values have been derived to match the SoC memory map specified in "PMP Region" column. To enable any PMP region, the A field in associated **pmpcfgX** register should always be programmed with value specified in "Address Matching Mode" column.

Name	Description	CSR Address	CSR Reset Value	CSR Access	Address Matching Mode	PMP Region
<a href="#">pmpcfg0</a>	PMP config register	0x3A0	0x0	R/W	-	-
<a href="#">pmpcfg1</a>	PMP config register	0x3A1	0x0	R/W	-	-
<a href="#">pmpcfg2</a>	PMP config register	0x3A2	0x0	R/W	-	-
<a href="#">pmpcfg3</a>	PMP config register	0x3A3	0x0	R/W	-	-
<a href="#">pmpaddr0</a>	PMP address register	0x3B0	0x0	R/W	OFF	IRAM Base Address
<a href="#">pmpaddr1</a>	PMP address register	0x3B1	0x0	R/W	TOR	IRAM End Address
<a href="#">pmpaddr2</a>	PMP address register	0x3B2	0x0	R/W	OFF	DRAM Base Address
<a href="#">pmpaddr3</a>	PMP address register	0x3B3	0x0FF38000	RO	TOR	DRAM End Address 0x3FCDFFFF
<a href="#">pmpaddr4</a>	PMP address register	0x3B4	0x08FFFFFF	RO	NAPOT	0x20000000 - 0x27FFFFFF (128 MB)
<a href="#">pmpaddr5</a>	PMP address register	0x3B5	0x0F07FFFF	RO	NAPOT	0x3C000000 - 0x3C3FFFFFF (4 MB)
<a href="#">pmpaddr6</a>	PMP address register	0x3B6	0x0FFC0000	RO	OFF	0x3FF00000
<a href="#">pmpaddr7</a>	PMP address register	0x3B7	0x0FFD4000	RO	TOR	0x3FF00000 - 0x3FF4FFFF (320 KB)
<a href="#">pmpaddr8</a>	PMP address register	0x3B8	0x10000000	RO	OFF	0x40000000
<a href="#">pmpaddr9</a>	PMP address register	0x3B9	0x10024000	RO	TOR	0x40000000 - 0x4008FFFF (576 KB)
<a href="#">pmpaddr10</a>	PMP address register	0x3BA	0x1087FFFF	RO	NAPOT	0x42000000 - 0x423FFFFFF (4 MB)
<a href="#">pmpaddr11</a>	PMP address register	0x3BB	0x1801FFFF	RO	NAPOT	0x60000000 - 0x600FFFFFF (1 MB)
<a href="#">pmpaddr12</a>	PMP address register	0x3BC	0x100DF7FF	RO	NAPOT	0x4037C000 - 0x4037FFFF (16 KB)

Name	Description	CSR Address	CSR Reset Value	CSR Access	Address Matching Mode	PMP Region
<a href="#">pmpaddr13</a>	PMP address register	0x3BD	0x3FFFFFFF	RO	NA4	0xFFFFFFFF (4 Byte)
<a href="#">pmpaddr14</a>	PMP address register	0x3BE	0x0	RO	OFF	0x0
<a href="#">pmpaddr15</a>	PMP address register	0x3BF	0x3FFFFFFF	RO	TOR	0xFFFFFFFFE (4 GB)

### 1.8.5 Register Description

PMP unit implements all `pmcfg0-3` and `pmpaddr0-15` CSRs as defined in RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.10.

## Chapter 2

## GDMA Controller (GDMA)

### 2.1 Overview

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

The GDMA controller in ESP8684 has two independent channels, i.e. one transmit channel (i.e., Tx channel 0) and one receive channel (i.e., Rx channel 0). These two channels are shared by peripherals with GDMA feature, namely SPI2, and SHA. Users can assign the two channels to any of these peripherals.

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

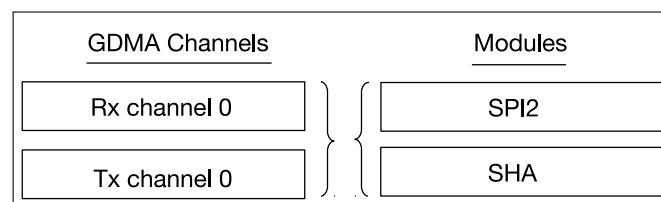


Figure 2.1-1. Modules with GDMA Feature and GDMA Channels

### 2.2 Features

The GDMA controller has the following features:

- Programmable length of data to be transferred in bytes
- Linked list of descriptors
- INCR burst transfer when accessing internal RAM
- Access to an address space of 256 KB at most in internal RAM
- One transmit channel and one receive channel
- Software-configurable selection of peripheral requesting its service
- Fixed channel priority and round-robin channel arbitration
- AHB bus architecture

## 2.3 Architecture

In ESP8684, all modules that need high-speed data transfer support GDMA. The GDMA controller and CPU data bus have access to the same address space in internal RAM. Figure 2.3-1 shows the basic architecture of the GDMA engine.

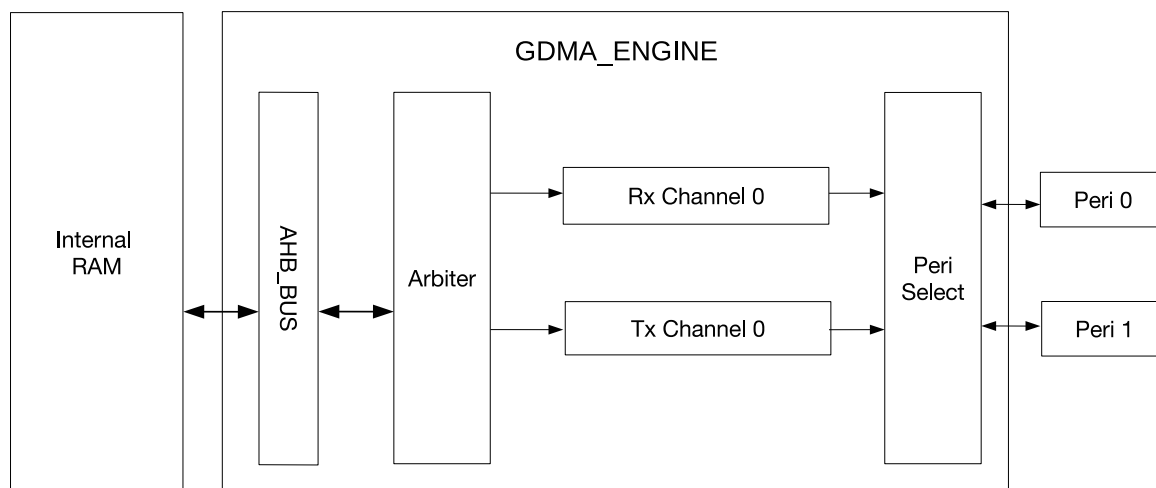


Figure 2.3-1. GDMA Engine Architecture

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

The GDMA engine reads data from or writes data to internal RAM via the AHB\_BUS. Before this, the GDMA controller uses fixed-priority arbitration scheme for channels requesting read or write access. For available address range of Internal RAM, please see Chapter 3 *System and Memory*.

Software can use the GDMA engine through linked lists. These linked lists, stored in internal RAM, consist of outlink and inlink. The GDMA controller reads an outlink (i.e. a linked list of transmit descriptors) from internal RAM and transmits data in corresponding RAM according to the outlink, or reads an inlink (i.e., a linked list of receive descriptors) and stores received data into specific address space in RAM according to the inlink.

## 2.4 Functional Description

### 2.4.1 Data Transfer Between Peripheral and Memory

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

Every transmit and receive channel can be connected to any peripheral with GDMA feature. Table 2.4-1 illustrates how to select the peripheral to be connected via registers. When one channel is connected to a peripheral, the other channel can not be connected to that peripheral.

Table 2.4-1. Selecting Peripherals via Register Configuration

GDMA_PERI_IN_SEL_CHO GDMA_PERI_OUT_SEL_CHO	Peripheral
0	SPI2
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	SHA
8	Reserved
9 ~ 63	Invalid

## 2.4.2 Memory-to-Memory Data Transfer

The GDMA controller also allows memory-to-memory data transfer. Such data transfer can be enabled by setting `GDMA_MEM_TRANS_EN_CHO`, which connects the output of transmit channel 0 to the input of receive channel 0.

## 2.4.3 Linked List

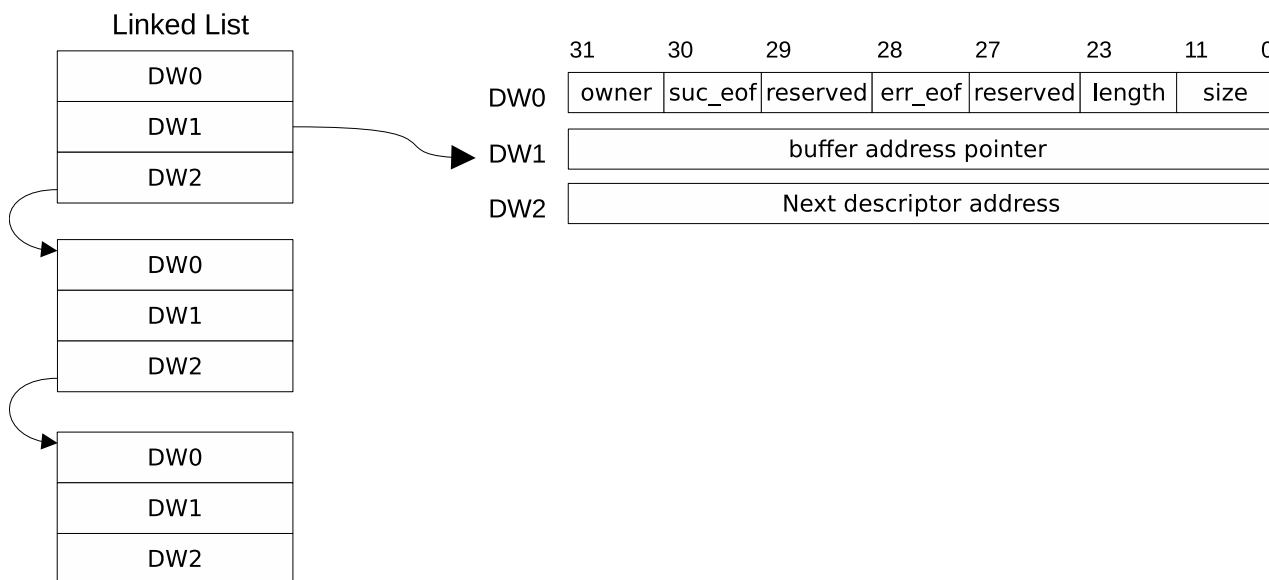


Figure 2.4-1. Structure of a Linked List

Figure 2.4-1 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of three words. Linked lists should be in internal RAM for the GDMA engine to be able to use them. The meaning of each field is as follows:

- Owner (DW0) [31]: Specifies who is allowed to access the buffer that this descriptor points to.  
1'b0: CPU can access the buffer;

1'b1: The GDMA controller can access the buffer.

When the GDMA controller stops using the buffer, this bit in a receive descriptor is automatically cleared by hardware, and this bit in a transmit descriptor is automatically cleared by hardware only if [GDMA\\_OUT\\_AUTO\\_WRBK\\_CHO](#) is set to 1. Before software loads a linked list, this bit should be set to 1.

**Note:** GDMA\_OUT is the prefix of transmit channel registers, and GDMA\_IN is the prefix of receive channel registers.

- suc\_eof (DW0) [30]: Specifies whether the GDMA\_IN\_SUC\_EOF\_CHO\_INT or GDMA\_OUT\_EOF\_CHO\_INT interrupt will be triggered when the data corresponding to this descriptor has been received or transmitted.

1'b0: No interrupt will be triggered after the current descriptor's successful transfer;

1'b1: An interrupt will be triggered after the current descriptor's successful transfer.

For receive descriptors, software needs to clear this bit to 0, and hardware will set it to 1 after receiving data containing the EOF flag.

For transmit descriptors, software needs to set this bit to 1 as needed.

If software configures this bit to 1 in a descriptor, the GDMA will include the EOF flag in the data sent to the corresponding peripheral, indicating to the peripheral that this data segment marks the end of one transfer phase.

- Reserved (DW0) [29]: Reserved. Value of this bit does not matter.
- err\_eof (DW0) [28]: Specifies whether the received data has errors.  
When an error is detected in the received data segment corresponding to a descriptor, this bit in the receive descriptor is set to 1 by hardware.
- Reserved (DW0) [27:24]: Reserved.
- Length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to.  
This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored into the buffer.
- Size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to. Size should be larger than or equal to length.
- Buffer address pointer (DW1): Address of the buffer. This field can only point to internal RAM.
- Next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value is 0. This field can only point to internal RAM.

When a data frame or packet has been received, the suc\_eof bit in the current receive descriptor will be set to 1, and the GDMA controller stops data transmission to the buffer pointed by the current receive descriptor.

Even if the length of data received is smaller than the size of the buffer, data received in the next transaction would not be stored in the available space of this buffer. The data would rather be stored in the buffer pointed by the next receive descriptor.

## 2.4.4 Enabling GDMA

Software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures [GDMA\\_INLINK\\_ADDR\\_CHO](#) field with address of the first receive descriptor, and sets [GDMA\\_INLINK\\_START\\_CHO](#) bit to enable GDMA. When the GDMA controller transmits data, software loads an

outlink, prepares data to be transmitted, configures `GDMA_OUTLINK_ADDR_CHO` field with address of the first transmit descriptor, and sets `GDMA_OUTLINK_START_CHO` bit to enable GDMA. `GDMA_INLINK_START_CHO` bit and `GDMA_OUTLINK_START_CHO` bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a DMA transfer that is already started. Naively, it would seem to be possible to do this only by setting the next descriptor address pointer field (DW2) at the end of the current list to the first descriptor of the to-be-added list. However, this strategy fails if the existing DMA transfer is almost or entirely finished. Instead, the GDMA engine has specialized logic to make sure a DMA transfer can be continued or restarted: if it is still ongoing, it will make sure to take the appended descriptors into account; if the transfer has already finished, it will restart with the new descriptors. This is implemented in the Restart function.

When using the Restart function, software needs to rewrite address of the first descriptor in the new list to DW2 of the last descriptor in the loaded list, and set `GDMA_INLINK_RESTART_CHO` bit or `GDMA_OUTLINK_RESTART_CHO` bit (these two bits are cleared automatically by hardware). As shown in Figure 2.4-2, by doing so hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

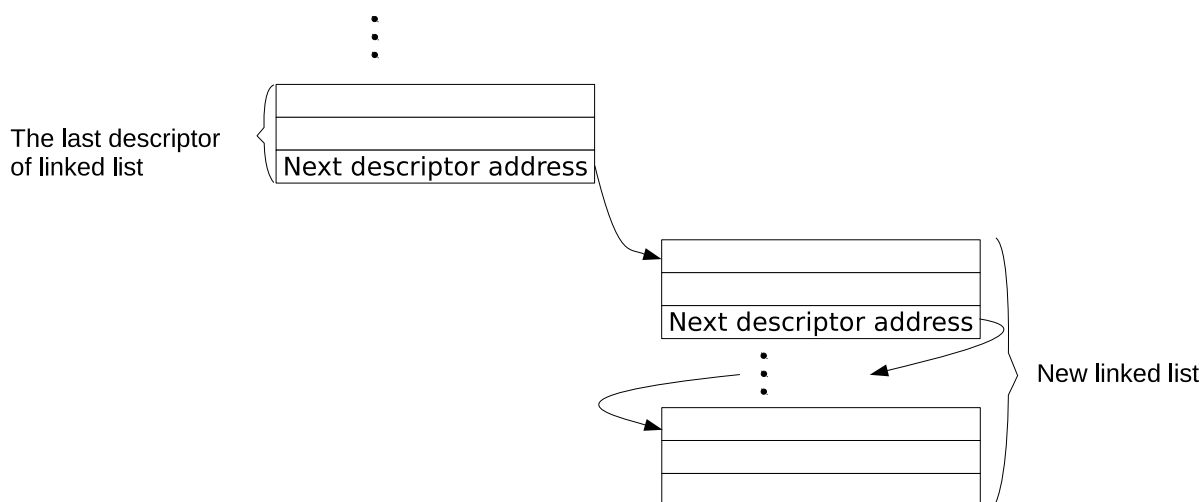


Figure 2.4-2. Relationship among Linked Lists

## 2.4.5 Linked List Reading Process

Once configured and enabled by software, the GDMA controller starts to read the linked list from internal RAM. The GDMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, will the corresponding GDMA channel transfer data. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either `GDMA_IN_DSCR_ERR_CHO_INT` or `GDMA_OUT_DSCR_ERR_CHO_INT`), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when `GDMA_IN_CHECK_OWNER_CHO` or `GDMA_OUT_CHECK_OWNER_CHO` is set to 1. If the owner bit is 0, the buffer should be accessed by the CPU. In this case, the owner bit fails the check. The owner bit check will be skipped if `GDMA_IN_CHECK_OWNER_CHO` or `GDMA_OUT_CHECK_OWNER_CHO` is 0;
- Buffer address pointer (DW1) check. If the buffer address pointer does not point to `0x3FCA0000 ~ 0x3FCDFFFF` (please refer to Section 2.4.7), it fails the check.

After software detects a descriptor error interrupt, it must reset the corresponding channel, and reconfigure this channel, and enable GDMA. For details, see Section 2.6.2, Section 2.6.3, and Section 2.6.4.

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

## 2.4.6 EOF

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

Before the GDMA controller transmits data, [GDMA\\_OUT\\_TOTAL\\_EOF\\_CHO\\_INT\\_ENA](#) bit should be set to enable GDMA\_OUT\_EOF\_CHO\_INT interrupt. If data in the buffer pointed by the last descriptor (with EOF) has been transmitted, a GDMA\_OUT\_EOF\_CHO\_INT interrupt is generated.

Before the GDMA controller receives data, [GDMA\\_IN\\_SUC\\_EOF\\_CHO\\_INT\\_ENA](#) bit should be set to enable GDMA\_IN\_SUC\_EOF\_CHO\_INT interrupt. If a data segment with an EOF flag has been received successfully, a GDMA\_IN\_SUC\_EOF\_CHO\_INT interrupt is generated. In addition, the GDMA controller also supports GDMA\_IN\_ERR\_CHO\_EOF\_INT interrupt. This interrupt is enabled by setting [GDMA\\_IN\\_ERR\\_EOF\\_CHO\\_INT\\_ENA](#) bit, and it indicates that a data segment corresponding to a descriptor has been received with errors.

When detecting a GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT or a GDMA\_IN\_SUC\_EOF\_CHO\_INT interrupt, software can record the value of [GDMA\\_OUT\\_EOF\\_DES\\_ADDR\\_CHO](#) or [GDMA\\_IN\\_SUC\\_EOF\\_DES\\_ADDR\\_CHO](#) field, i.e., address of the last descriptor. Therefore, software can tell which descriptors have been used and reclaim them.

**Note:** In this chapter, EOF of transmit descriptors refers to suc\_eof, while EOF of receive descriptors refers to both suc\_eof and err\_eof.

## 2.4.7 Accessing Internal RAM

Any transmit and receive channel of GDMA can access 0x3FCA0000 ~ 0x3FCDFFFF in internal RAM. To improve data transfer efficiency, GDMA can send data in burst mode, which is disabled by default. This mode is enabled for receive channel by setting [GDMA\\_IN\\_DATA\\_BURST\\_EN\\_CHO](#), and enabled for transmit channel by setting [GDMA\\_OUT\\_DATA\\_BURST\\_EN\\_CHO](#).

**Table 2.4-2. Descriptor Field Alignment Requirements**

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

<sup>1</sup> "—" means no alignment requirements.

Table 2.4-2 lists the requirements for descriptor field alignment when accessing internal RAM.

When burst mode is disabled, size, length, and buffer address pointer in both transmit and receive descriptors do not need to be word-aligned. That is to say, GDMA can read data of specified length (1 ~ 4095 bytes) from



any start addresses in the accessible address range, or write received data of the specified length (1 ~ 4095 bytes) to any contiguous addresses in the accessible address range.

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

## 2.4.8 Arbitration

To ensure timely response to peripherals running at a high speed with low latency (such as SPI), the GDMA controller implements a fixed-priority channel arbitration scheme. That is to say, each channel can be assigned a priority from 0 ~ 9. The larger the number, the higher the priority, and the more timely the response. When several channels are assigned the same priority, the GDMA controller adopts a round-robin arbitration scheme.

Please note that the overall throughput of peripherals with GDMA feature cannot exceed the maximum bandwidth of the GDMA. Otherwise, requests from low-priority peripherals might not be responded to in time.

## 2.5 GDMA Interrupts

- DMA\_INFIFO\_OVF\_CHO\_INT: Triggered when the RX FIFO of GDMA overflows.
- GDMA\_INFIFO\_UDF\_CHO\_INT: Triggered when the RX FIFO of GDMA underflows.
- GDMA\_IN\_DSCR\_EMPTY\_CHO\_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel 0.
- GDMA\_IN\_DSCR\_ERR\_CHO\_INT: Triggered when an error is detected in a receive descriptor on receive channel 0.
- GDMA\_IN\_ERR\_EOF\_CHO\_INT: Triggered when an error is detected in the data segment corresponding to a descriptor received via receive channel 0. This interrupt is used only for UHCIO peripheral (UART0 or UART1).
- GDMA\_IN\_SUC\_EOF\_CHO\_INT: Triggered when the suc\_eof bit in a receive descriptor is 1 and the data corresponding to this receive descriptor has been received via receive channel 0.
- GDMA\_IN\_DONE\_CHO\_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel 0.
- GMA\_OUTFIFO\_OVF\_CHO\_INT: Triggered when the TX FIFO of GDMA overflows.
- GDMA\_OUTFIFO\_UDF\_CHO\_INT: Triggered when the TX FIFO of GDMA underflows.
- GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel 0.
- GDMA\_OUT\_DSCR\_ERR\_CHO\_INT: Triggered when an error is detected in a transmit descriptor on transmit channel 0.
- GDMA\_OUT\_EOF\_CHO\_INT: Triggered when EOF in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel 0. If [GDMA\\_OUT\\_EOF\\_MODE\\_CHO](#) is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters GDMA's transmit

channel; if [GDMA\\_OUT\\_EOF\\_MODE\\_CHO](#) is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.

- [GDMA\\_OUT\\_DONE\\_CHO\\_INT](#): Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel 0.

## 2.6 Programming Procedures

### 2.6.1 Programming Procedure for GDMA Clock and Reset

GDMA's clock and reset should be configured as follows:

1. Set [SYSTEM\\_DMA\\_CLK\\_EN](#) to enable GDMA's clock;
2. Clear [SYSTEM\\_DMA\\_RST](#) to reset GDMA.

### 2.6.2 Programming Procedure for GDMA's Transmit Channel

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

1. Set [GDMA\\_OUT\\_RST\\_CHO](#) first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer;
2. Load an outlink, and configure [GDMA\\_OUTLINK\\_ADDR\\_CHO](#) with address of the first transmit descriptor;
3. Configure [GDMA\\_PERI\\_OUT\\_SEL\\_CHO](#) with the value corresponding to the peripheral to be connected, as shown in Table 2.4-1;
4. Set [GDMA\\_OUTLINK\\_START\\_CHO](#) to enable GDMA's transmit channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2 or SHA). See details in individual chapters of these peripherals;
6. Wait for [GDMA\\_OUT\\_TOTAL\\_EOF\\_CHO\\_INT](#) interrupt, which indicates the completion of data transfer.

### 2.6.3 Programming Procedure for GDMA's Receive Channel

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

1. Set [GDMA\\_IN\\_RST\\_CHO](#) first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer;
2. Load an inlink, and configure [GDMA\\_INLINK\\_ADDR\\_CHO](#) with address of the first receive descriptor;
3. Configure [GDMA\\_PERI\\_IN\\_SEL\\_CHO](#) with the value corresponding to the peripheral to be connected, as shown in Table 2.4-1;
4. Set [GDMA\\_INLINK\\_START\\_CHO](#) to enable GDMA's receive channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2). See details in individual chapters of these peripherals;

## 2.6.4 Programming Procedure for Memory-to-Memory Transfer

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

1. Set [GDMA\\_OUT\\_RST\\_CHO](#) first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer;
2. Set [GDMA\\_IN\\_RST\\_CHO](#) first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer;
3. Load an outlink, and configure [GDMA\\_OUTLINK\\_ADDR\\_CHO](#) with address of the first transmit descriptor;
4. Load an inlink, and configure [GDMA\\_INLINK\\_ADDR\\_CHO](#) with address of the first receive descriptor;
5. Set [GDMA\\_MEM\\_TRANS\\_EN\\_CHO](#) to enable memory-to-memory transfer;
6. Set [GDMA\\_OUTLINK\\_START\\_CHO](#) to enable GDMA's transmit channel for data transfer;
7. Set [GDMA\\_INLINK\\_START\\_CHO](#) to enable GDMA's receive channel for data transfer;
8. If the `suc_eof` bit is set in a transmit descriptor, a `GDMA_IN_SUC_EOF_CHO_INT` interrupt will be triggered when the data segment corresponding to this descriptor has been transmitted.

## 2.7 Register Summary

The addresses in this section are relative to GDMA Controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

The abbreviations given in Column **Access** are explained in Section [Access Types for Registers](#).

Name	Description	Address	Access
<b>Interrupt Registers</b>			
<a href="#">GDMA_INT_RAW_CHO_REG</a>	Raw status interrupt of RX channel 0	0x0000	R/WTC/SS
<a href="#">GDMA_INT_ST_CHO_REG</a>	Masked interrupt of RX channel 0	0x0004	RO
<a href="#">GDMA_INT_ENA_CHO_REG</a>	Interrupt enable bits of RX channel 0	0x0008	R/W
<a href="#">GDMA_INT_CLR_CHO_REG</a>	Interrupt clear bits of RX channel 0	0x000C	WT
<b>Configuration Registers</b>			
<a href="#">GDMA_MISC_CONF_REG</a>	Miscellaneous register	0x0044	R/W
<a href="#">GDMA_IN_CONFO_CHO_REG</a>	Configuration register 0 of RX channel 0	0x0070	R/W
<a href="#">GDMA_IN_CONF1_CHO_REG</a>	Configuration register 1 of RX channel 0	0x0074	R/W
<a href="#">GDMA_IN_POP_CHO_REG</a>	Pop control register of RX channel 0	0x007C	varies
<a href="#">GDMA_IN_LINK_CHO_REG</a>	Link descriptor configuration and control register of RX channel 0	0x0080	varies
<a href="#">GDMA_OUT_CONFO_CHO_REG</a>	Configuration register 0 of TX channel 0	0x00D0	R/W
<a href="#">GDMA_OUT_CONF1_CHO_REG</a>	Configuration register 1 of TX channel 0	0x00D4	R/W
<a href="#">GDMA_OUT_PUSH_CHO_REG</a>	Push control register of TX channel 0	0x00DC	varies
<a href="#">GDMA_OUT_LINK_CHO_REG</a>	Link descriptor configuration and control register of TX channel 0	0x00E0	varies
<b>Status Registers</b>			
<a href="#">GDMA_INFIFO_STATUS_CHO_REG</a>	RX FIFO status of RX channel 0	0x0078	RO
<a href="#">GDMA_IN_STATE_CHO_REG</a>	Receive status of RX channel 0	0x0084	RO
<a href="#">GDMA_IN_SUC_EOF_DES_ADDR_CHO_REG</a>	Inlink descriptor address when EOF occurs of RX channel 0	0x0088	RO
<a href="#">GDMA_IN_ERR_EOF_DES_ADDR_CHO_REG</a>	Inlink descriptor address when errors occur of RX channel 0	0x008C	RO
<a href="#">GDMA_IN_DSCR_CHO_REG</a>	Address of the next receive descriptor pointed by the current pre-read receive descriptor on RX channel 0	0x0090	RO
<a href="#">GDMA_IN_DSCR_BFO_CHO_REG</a>	Address of the current pre-read receive descriptor on RX channel 0	0x0094	RO
<a href="#">GDMA_IN_DSCR_BF1_CHO_REG</a>	Address of the previous pre-read receive descriptor on RX channel 0	0x0098	RO
<a href="#">GDMA_OUTFIFO_STATUS_CHO_REG</a>	TX FIFO status of TX channel 0	0x00D8	RO
<a href="#">GDMA_OUT_STATE_CHO_REG</a>	Transmit status of TX channel 0	0x00E4	RO
<a href="#">GDMA_OUT_EOF_DES_ADDR_CHO_REG</a>	Outlink descriptor address when EOF occurs of TX channel 0	0x00E8	RO
<a href="#">GDMA_OUT_EOF_BFR_DES_ADDR_CHO_REG</a>	The last outlink descriptor address when EOF occurs of TX channel 0	0x00EC	RO

Name	Description	Address	Access
<a href="#">GDMA_OUT_DSCR_CHO_REG</a>	Address of the next transmit descriptor pointed by the current pre-read transmit descriptor on TX channel 0	0x00F0	RO
<a href="#">GDMA_OUT_DSCR_BFO_CHO_REG</a>	Address of the current pre-read transmit descriptor on TX channel 0	0x00F4	RO
<a href="#">GDMA_OUT_DSCR_BF1_CHO_REG</a>	Address of the previous pre-read transmit descriptor on TX channel 0	0x00F8	RO
<b>Priority Registers</b>			
<a href="#">GDMA_IN_PRI_CHO_REG</a>	Priority register of RX channel 0	0x009C	R/W
<a href="#">GDMA_OUT_PRI_CHO_REG</a>	Priority register of TX channel 0	0x00FC	R/W
<b>Peripheral Select Registers</b>			
<a href="#">GDMA_IN_PERI_SEL_CHO_REG</a>	Peripheral selection of RX channel 0	0x00A0	R/W
<a href="#">GDMA_OUT_PERI_SEL_CHO_REG</a>	Peripheral selection of TX channel 0	0x0100	R/W
<b>Version Registers</b>			
<a href="#">GDMA_DATE_REG</a>	Version control register	0x0048	R/W

## 2.8 Registers

The addresses in this section are relative to GDMA Controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 2.1. GDMA\_INT\_RAW\_CHO\_REG (0x0000)

(reserved)																												GDMA_OUT_FIFO_UDF_CHO_INT_RAW GDMA_OUT_FIFO_OVF_CHO_INT_RAW GDMA_INFIFO_UDF_CHO_INT_RAW GDMA_INFIFO_OVF_CHO_INT_RAW GDMA_OUT_TOTAL_EOF_CHO_INT_RAW GDMA_IN_DSCR_EMPTY_CHO_INT_RAW GDMA_OUT_DSCR_ERR_CHO_INT_RAW GDMA_OUT_DONE_EOF_CHO_INT_RAW GDMA_IN_DSCR_ERR_CHO_INT_RAW GDMA_IN_DONE_EOF_CHO_INT_RAW GDMA_IN_SUC_EOF_CHO_INT_RAW GDMA_IN_DONE_CHO_INT_RAW													
31														13														12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0													

**GDMA\_IN\_DONE\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when the last data pointed by one receive descriptor has been received for RX channel 0. (R/WTC/SS)

**GDMA\_IN\_SUC\_EOF\_CHO\_INT\_RAW** The raw interrupt bit turns to high level for RX channel 0 when the last data pointed by one receive descriptor has been received and the suc\_eof bit in this descriptor is 1. (R/WTC/SS)

**GDMA\_IN\_ERR\_EOF\_CHO\_INT\_RAW** Reserved. (R/WTC/SS)

**GDMA\_OUT\_DONE\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when the last data pointed by one transmit descriptor has been transmitted to peripherals for TX channel 0. (R/WTC/SS)

**GDMA\_OUT\_EOF\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when the last data pointed by one transmit descriptor has been read from memory for TX channel 0. (R/WTC/SS)

**GDMA\_IN\_DSCR\_ERR\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when detecting receive descriptor error, including owner error, the second and third word error of receive descriptor for RX channel 0. (R/WTC/SS)

**GDMA\_OUT\_DSCR\_ERR\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when detecting transmit descriptor error, including owner error, the second and third word error of transmit descriptor for TX channel 0. (R/WTC/SS)

Continued on the next page...

**Register 2.1. GDMA\_INT\_RAW\_CHO\_REG (0x0000)**

Continued from the previous page...

**GDMA\_IN\_DSCR\_EMPTY\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when RX buffer pointed by inlink is full and receiving data is not completed, but there is no more inlink for RX channel 0. (R/WTC/SS)

**GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT\_RAW** The raw interrupt bit turns to high level when data corresponding to an outlink (includes one descriptor or few descriptors) is transmitted out for TX channel 0. (R/WTC/SS)

**GDMA\_INFIFO\_OVF\_CHO\_INT\_RAW** This raw interrupt bit turns to high level when level 1 FIFO of RX channel 0 is overflow. (R/WTC/SS)

**GDMA\_INFIFO\_UDF\_CHO\_INT\_RAW** This raw interrupt bit turns to high level when level 1 FIFO of RX channel 0 is underflow. (R/WTC/SS)

**GDMA\_OUTFIFO\_OVF\_CHO\_INT\_RAW** This raw interrupt bit turns to high level when level 1 FIFO of TX channel 0 is overflow. (R/WTC/SS)

**GDMA\_OUTFIFO\_UDF\_CHO\_INT\_RAW** This raw interrupt bit turns to high level when level 1 FIFO of TX channel 0 is underflow. (R/WTC/SS)

Register 2.2. GDMA\_INT\_ST\_CHO\_REG (0x0004)

(reserved)																								GDMA_OUTFIFO_UDF_CHO_INT_ST GDMA_OUTFIFO_OVF_CHO_INT_ST GDMA_INFIFO_UDF_CHO_INT_ST GDMA_INFIFO_OVF_CHO_INT_ST GDMA_OUT_TOTAL_EOF_CHO_INT_ST GDMA_IN_DSCR_ERR_CHO_INT_ST GDMA_OUT_DSCR_EMPTY_CHO_INT_ST GDMA_OUT_DSCR_ERR_CHO_INT_ST GDMA_IN_DONE_EOF_CHO_INT_ST GDMA_IN_SUC_EOF_CHO_INT_ST GDMA_IN_DONE_EOF_CHO_INT_ST												
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	Reset									

**GDMA\_IN\_DONE\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_IN\_DONE\_CH\_INT interrupt. (RO)

**GDMA\_IN\_SUC\_EOF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_IN\_SUC\_EOF\_CH\_INT interrupt. (RO)

**GDMA\_IN\_ERR\_EOF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_IN\_ERR\_EOF\_CH\_INT interrupt. (RO)

**GDMA\_OUT\_DONE\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUT\_DONE\_CH\_INT interrupt. (RO)

**GDMA\_OUT\_EOF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUT\_EOF\_CH\_INT interrupt. (RO)

**GDMA\_IN\_DSCR\_ERR\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_IN\_DSCR\_ERR\_CH\_INT interrupt. (RO)

**GDMA\_OUT\_DSCR\_ERR\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUT\_DSCR\_ERR\_CH\_INT interrupt. (RO)

**GDMA\_IN\_DSCR\_EMPTY\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_IN\_DSCR\_EMPTY\_CH\_INT interrupt. (RO)

**GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUT\_TOTAL\_EOF\_CH\_INT interrupt. (RO)

**GDMA\_INFIFO\_OVF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_INFIFO\_OVF\_L1\_CH\_INT interrupt. (RO)

**GDMA\_INFIFO\_UDF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_INFIFO\_UDF\_L1\_CH\_INT interrupt. (RO)

**GDMA\_OUTFIFO\_OVF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUTFIFO\_OVF\_L1\_CH\_INT interrupt. (RO)

**GDMA\_OUTFIFO\_UDF\_CHO\_INT\_ST** The raw interrupt status bit for the GDMA\_OUTFIFO\_UDF\_L1\_CH\_INT interrupt. (RO)



Register 2.3. GDMA\_INT\_ENA\_CHO\_REG (0x0008)

(reserved)																												GDMA_OUTFIFO_UDF_CHO_INT_ENA GDMA_OUTFIFO_OVF_CHO_INT_ENA GDMA_INFIFO_UDF_CHO_INT_ENA GDMA_INFIFO_OVF_CHO_INT_ENA GDMA_OUT_TOTAL_EOF_CHO_INT_ENA GDMA_IN_DSCR_EMPTY_CHO_INT_ENA GDMA_OUT_DSCR_ERR_CHO_INT_ENA GDMA_OUT_DONE_CHO_INT_ENA GDMA_IN_ERR_EOF_CHO_INT_ENA GDMA_IN_SUC_EOF_CHO_INT_ENA GDMA_IN_DONE_CHO_INT_ENA													
31																13												12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														

**GDMA\_IN\_DONE\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_IN\_DONE\_CH\_INT interrupt. (R/W)

**GDMA\_IN\_SUC\_EOF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_IN\_SUC\_EOF\_CH\_INT interrupt. (R/W)

**GDMA\_IN\_ERR\_EOF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_IN\_ERR\_EOF\_CH\_INT interrupt. (R/W)

**GDMA\_OUT\_DONE\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUT\_DONE\_CH\_INT interrupt. (R/W)

**GDMA\_OUT\_EOF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUT\_EOF\_CH\_INT interrupt. (R/W)

**GDMA\_IN\_DSCR\_ERR\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_IN\_DSCR\_ERR\_CH\_INT interrupt. (R/W)

**GDMA\_OUT\_DSCR\_ERR\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUT\_DSCR\_ERR\_CH\_INT interrupt. (R/W)

**GDMA\_IN\_DSCR\_EMPTY\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_IN\_DSCR\_EMPTY\_CH\_INT interrupt. (R/W)

**GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUT\_TOTAL\_EOF\_CH\_INT interrupt. (R/W)

**GDMA\_INFIFO\_OVF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_INFIFO\_OVF\_L1\_CH\_INT interrupt. (R/W)

**GDMA\_INFIFO\_UDF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_INFIFO\_UDF\_L1\_CH\_INT interrupt. (R/W)

**GDMA\_OUTFIFO\_OVF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUTFIFO\_OVF\_L1\_CH\_INT interrupt. (R/W)

**GDMA\_OUTFIFO\_UDF\_CHO\_INT\_ENA** The interrupt enable bit for the GDMA\_OUTFIFO\_UDF\_L1\_CH\_INT interrupt. (R/W)

Register 2.4. GDMA\_INT\_CLR\_CHO\_REG (0x000C)

(reserved)																				GDMA_OUTFIFO_UDF_CHO_INT_CLR GDMA_OUTFIFO_OVF_CHO_INT_CLR GDMA_INFIFO_UDF_CHO_INT_CLR GDMA_INFIFO_OVF_CHO_INT_CLR GDMA_OUT_TOTAL_EOF_CHO_INT_CLR GDMA_IN_DSCR_EMPTY_CHO_INT_CLR GDMA_OUT_DSCR_ERR_CHO_INT_CLR GDMA_IN_DSCR_EOF_CHO_INT_CLR GDMA_OUT_DONE_CHO_INT_CLR GDMA_IN_ERR_EOF_CHO_INT_CLR GDMA_IN_SUC_EOF_CHO_INT_CLR GDMA_IN_DONE_CHO_INT_CLR											
31													13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset				
0 0 0 0 0 0 0 0 0 0 0 0 0													0	0	0	0	0	0	0	0	0	0	0	0							

**GDMA\_IN\_DONE\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_IN\_DONE\_CH\_INT interrupt. (WT)

**GDMA\_IN\_SUC\_EOF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_IN\_SUC\_EOF\_CH\_INT interrupt. (WT)

**GDMA\_IN\_ERR\_EOF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_IN\_ERR\_EOF\_CH\_INT interrupt. (WT)

**GDMA\_OUT\_DONE\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUT\_DONE\_CH\_INT interrupt. (WT)

**GDMA\_OUT\_EOF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUT\_EOF\_CH\_INT interrupt. (WT)

**GDMA\_IN\_DSCR\_ERR\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_IN\_DSCR\_ERR\_CH\_INT interrupt. (WT)

**GDMA\_OUT\_DSCR\_ERR\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUT\_DSCR\_ERR\_CH\_INT interrupt. (WT)

**GDMA\_IN\_DSCR\_EMPTY\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_IN\_DSCR\_EMPTY\_CH\_INT interrupt. (WT)

**GDMA\_OUT\_TOTAL\_EOF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUT\_TOTAL\_EOF\_CH\_INT interrupt. (WT)

**GDMA\_INFIFO\_OVF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_INFIFO\_OVF\_L1\_CH\_INT interrupt. (WT)

**GDMA\_INFIFO\_UDF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_INFIFO\_UDF\_L1\_CH\_INT interrupt. (WT)

**GDMA\_OUTFIFO\_OVF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUTFIFO\_OVF\_L1\_CH\_INT interrupt. (WT)

**GDMA\_OUTFIFO\_UDF\_CHO\_INT\_CLR** Set this bit to clear the GDMA\_OUTFIFO\_UDF\_L1\_CH\_INT interrupt. (WT)

**Register 2.5. GDMA\_MISC\_CONF\_REG (0x0044)**

(reserved)																										GDMA_CLK_EN GDMA_ARB_PRI_DIS (reserved) GDMA_AHBM_RST_INTER				
31																								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

**GDMA\_AHBM\_RST\_INTER** Set this bit, then clear this bit to reset the internal ahb FSM. (R/W)

**GDMA\_ARB\_PRI\_DIS** Set this bit to disable priority arbitration function. (R/W)

**GDMA\_CLK\_EN** 0: Enable the clock only when application writes registers. 1: Force the clock on for registers. (R/W)

**Register 2.6. GDMA\_IN\_CONFO\_CHO\_REG (0x0070)**

(reserved)																										GDMA_MEM_TRANS_EN_CHO GDMA_IN_DATA_BURST_EN_CHO GDMA_INDSOR_BURST_EN_CHO GDMA_IN_LOOP_TEST_CHO GDMA_IN_RST_CHO						
31																										5	4	3	2	1	0	Reset
0 0																										0	0	0	0	0		

**GDMA\_IN\_RST\_CHO** This bit is used to reset GDMA channel 0 RX FSM and RX FIFO pointer. (R/W)

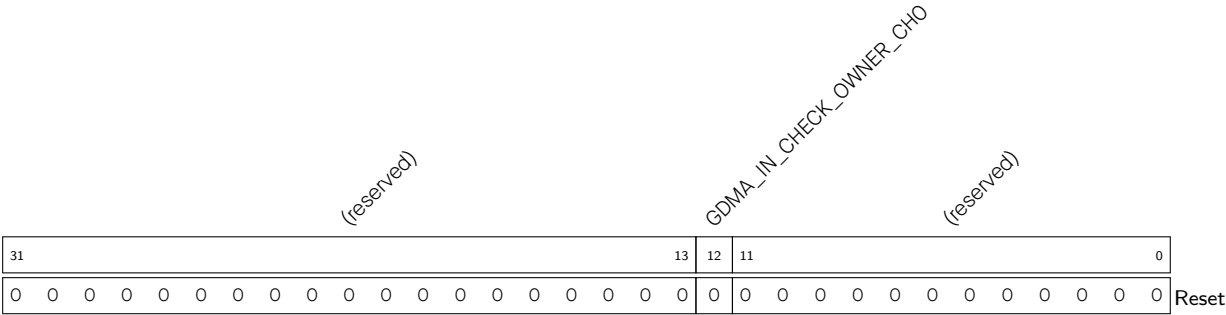
**GDMA\_IN\_LOOP\_TEST\_CHO** Reserved. (R/W)

**GDMA\_INDSOR\_BURST\_EN\_CHO** Set this bit to 1 to enable INCR burst transfer for RX channel 0 reading descriptor when accessing internal RAM. (R/W)

**GDMA\_IN\_DATA\_BURST\_EN\_CHO** Set this bit to 1 to enable INCR burst transfer for RX channel 0 receiving data when accessing internal RAM. (R/W)

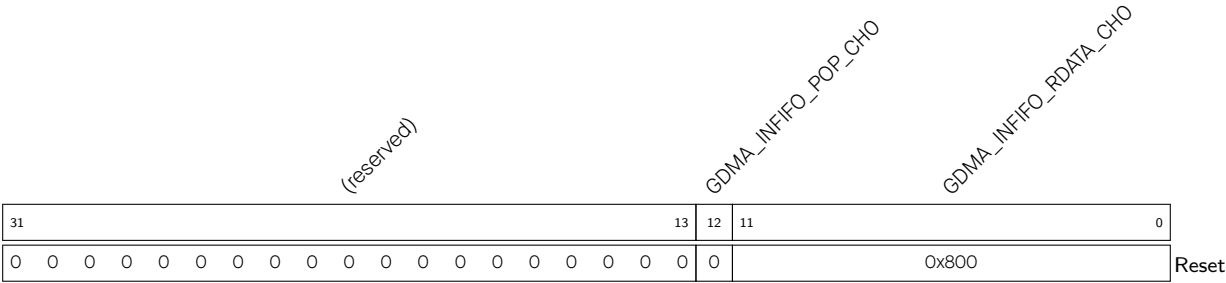
**GDMA\_MEM\_TRANS\_EN\_CHO** Set this bit 1 to enable automatic transmitting data from memory to memory via GDMA. (R/W)

Register 2.7. GDMA\_IN\_CONF1\_CHO\_REG (0x0074)



**GDMA\_IN\_CHECK\_OWNER\_CHO** Set this bit to enable checking the owner attribute of the descriptor. (R/W)

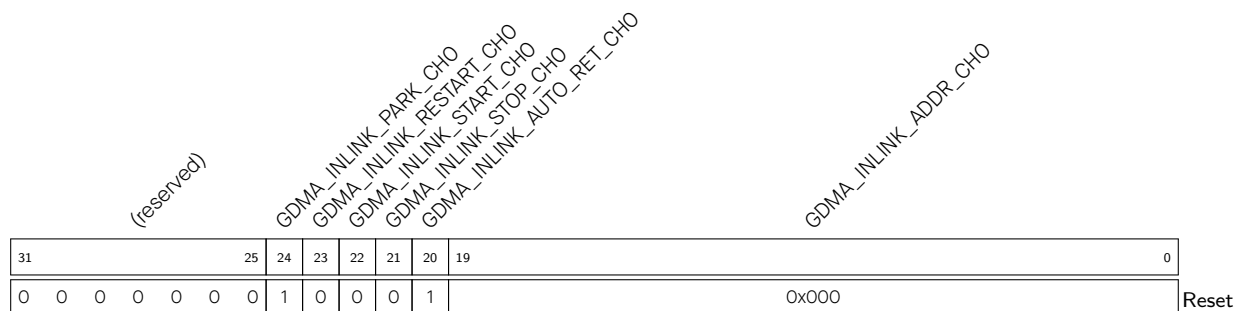
Register 2.8. GDMA\_IN\_POP\_CHO\_REG (0x007C)



**GDMA\_INFIFO\_RDATA\_CHO** This register stores the data popping from GDMA FIFO (intended for debugging). (RO)

**GDMA\_INFIFO\_POP\_CHO** Set this bit to pop data from GDMA FIFO (intended for debugging). (R/W/SC)

### Register 2.9. GDMA\_IN\_LINK\_CHO\_REG (0x0080)



**GDMA\_INLINK\_ADDR\_CHO** This register stores the 20 least significant bits of the first receive descriptor's address. (R/W)

**GDMA\_INLINK\_AUTO\_RET\_CHO** Set this bit to return to current receive descriptor's address, when there are some errors in current receiving data. (R/W)

**GDMA\_INLINK\_STOP\_CHO** Set this bit to stop GDMA's receive channel from receiving data.  
(R/W/SC)

**GDMA\_INLINK\_START\_CHO** Set this bit to enable GDMA's receive channel for data transfer.  
(R/W/SC)

**GDMA\_INLINK\_RESTART\_CHO** Set this bit to mount a new receive descriptor. (R/W/SC)

**GDMA\_INLINK\_PARK\_CHO** 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

### Register 2.10. GDMA\_OUT\_CONFO\_CHO\_REG (0x00D0)

Diagram illustrating the structure of the **GDMA\_OUT\_TEST\_CHO** register (32 bits wide):

- Bits 31 to 6: (reserved)
- Bit 5: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**
- Bit 4: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**
- Bit 3: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**
- Bit 2: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**
- Bit 1: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**
- Bit 0: **GDMA\_OUT\_DATA\_BURST\_EN\_CHO**

**GDMA\_OUT\_RST\_CHO** This bit is used to reset GDMA channel 0 TX FSM and TX FIFO pointer. (R/W)

GDMA\_OUT\_LOOP\_TEST\_CHO Reserved. (R/W)

**GDMA\_OUT\_AUTO\_WBACK\_CHO** Set this bit to enable automatic outlink-writeback when all the data in TX buffer has been transmitted. (R/W)

<b>GDMA_OUT_EOF_MODE_CHO</b>	EOF flag generation mode when transmitting data. 1: EOF flag for TX channel 0 is generated when data need to transmit has been popped from FIFO in GDMA. (R/W)
------------------------------	--

**GDMA\_OUTDSCR\_BURST\_EN\_CHO** Set this bit to 1 to enable INCR burst transfer for TX channel 0 reading descriptor when accessing internal RAM. (R/W)

**GDMA\_OUT\_DATA\_BURST\_EN\_CHO** Set this bit to 1 to enable INCR burst transfer for TX channel 0 transmitting data when accessing internal RAM. (R/W)

### Register 2.11. GDMA\_OUT\_CONF1\_CHO\_REG (0x00D4)

[illegible]

**GDMA\_OUT\_CHECK\_OWNER\_CHO** Set this bit to enable checking the owner attribute of the descriptor. (R/W)

**Register 2.12. GDMA\_OUT\_PUSH\_CHO\_REG (0x00DC)**

(reserved)										GDMA_OUTFIFO_PUSH_CHO		GDMA_OUTFIFO_WDATA_CHO	
31											9	8	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
												0x0	Reset

**GDMA\_OUTFIFO\_WDATA\_CHO** This register stores the data that need to be pushed into GDMA FIFO. (R/W)

**GDMA\_OUTFIFO\_PUSH\_CHO** Set this bit to push data into GDMA FIFO. (R/W/SC)

**Register 2.13. GDMA\_OUT\_LINK\_CHO\_REG (0x00E0)**

(reserved)																GDMA_OUTLINK_PARK_CHO				GDMA_OUTLINK_RESTART_CHO				GDMA_OUTLINK_START_CHO				GDMA_OUTLINK_STOP_CHO				GDMA_OUTLINK_ADDR_CHO																																																																								
31								24								23	22	21	20	19	0																																																																																			
0								0								0								0								0								0								1								0								0								0								0x000																								Reset

**GDMA\_OUTLINK\_ADDR\_CHO** This register stores the 20 least significant bits of the first transmit descriptor's address. (R/W)

**GDMA\_OUTLINK\_STOP\_CHO** Set this bit to stop GDMA's receive channel from receiving data. (R/W/SC)

**GDMA\_OUTLINK\_START\_CHO** Set this bit to enable GDMA's transmit channel for data transfer. (R/W/SC)

**GDMA\_OUTLINK\_RESTART\_CHO** Set this bit to restart a new outlink from the last address. (R/W/SC)

**GDMA\_OUTLINK\_PARK\_CHO** 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

### Register 2.14. GDMA\_INFIFO\_STATUS\_CHO\_REG (0x0078)

[illegible]

**GDMA\_INFIFO\_FULL\_CHO** L1 RX FIFO full signal for RX channel 0. (RO)

**GDMA\_INFIFO\_EMPTY\_CHO** L1 RX FIFO empty signal for RX channel 0. (RO)

**GDMA\_INFIFO\_CNT\_CHO** The register stores the byte number of the data in L1 RX FIFO for RX channel 0. (RO)

GDMA IN REMAIN UNDER 1B CHO Reserved. (RO)

GDMA IN REMAIN UNDER 2B CHO Reserved. (RO)

GDMA IN REMAIN UNDER 3B CHO Reserved. (RO)

GDMA IN REMAIN UNDER 4B CHO Reserved. (RO)

GDMA\_IN\_BUF\_HUNGRY\_CHO Reserved. (RO)

### Register 2.15. GDMA IN STATE CHO REG (0x0084)

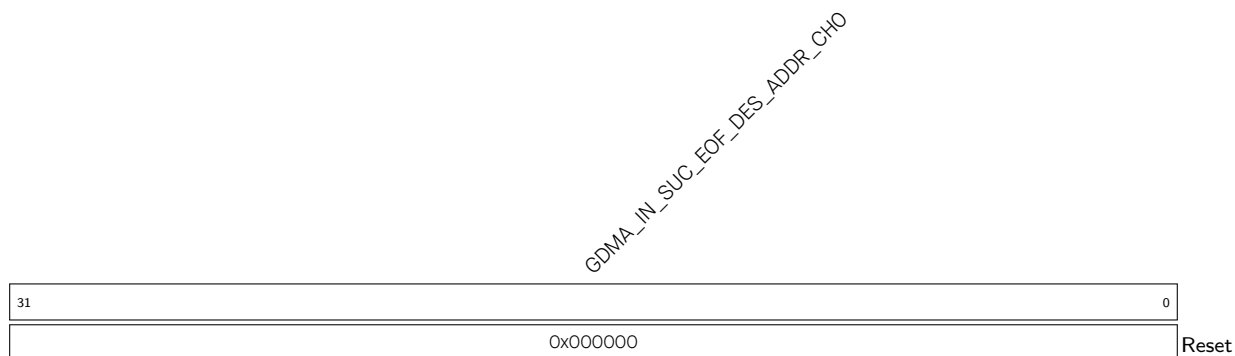
[illegible]

**GDMA\_INLINK\_DSCR\_ADDR\_CHO** This register stores the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)

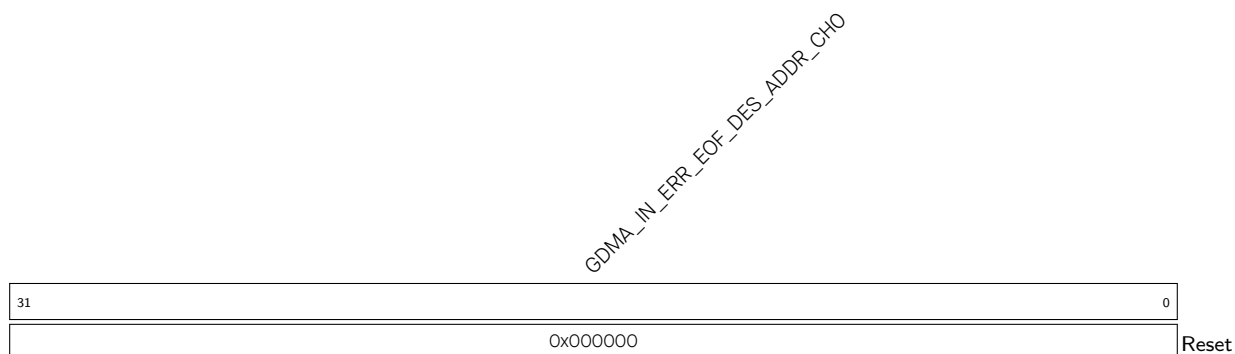
GDMA\_IN\_DSCR\_STATE\_CHO Reserved. (RO)

GDMA\_IN\_STATE\_CHO Reserved. (RO)

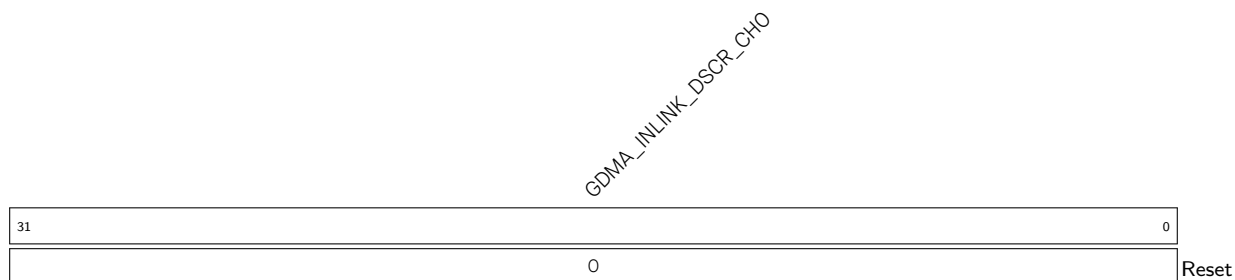


**Register 2.16. GDMA\_IN\_SUC\_EOF\_DES\_ADDR\_CHO\_REG (0x0088)**

**GDMA\_IN\_SUC\_EOF\_DES\_ADDR\_CHO** This register stores the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

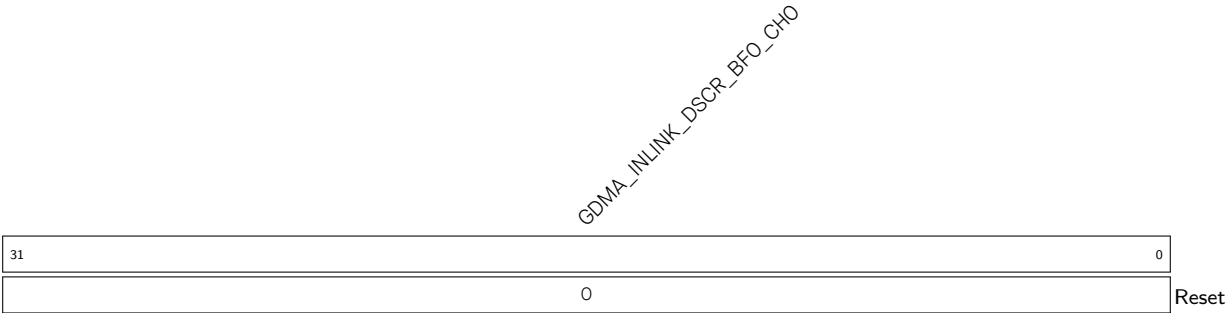
**Register 2.17. GDMA\_IN\_ERR\_EOF\_DES\_ADDR\_CHO\_REG (0x008C)**

**GDMA\_IN\_ERR\_EOF\_DES\_ADDR\_CHO** Reserved. (RO)

**Register 2.18. GDMA\_IN\_DSCR\_CHO\_REG (0x0090)**

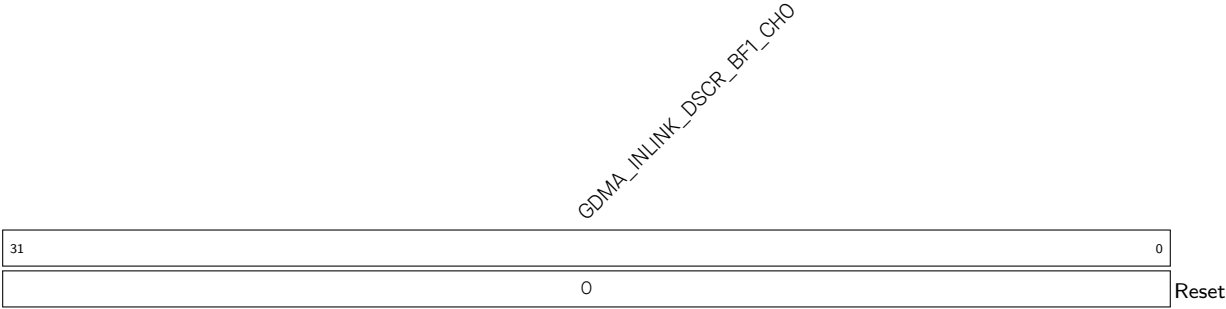
**GDMA\_INLINK\_DSCR\_CHO** Represents the address of the next receive descriptor x+1 pointed by the current receive descriptor that is pre-read. (RO)

Register 2.19. GDMA\_IN\_DSCR\_BFO\_CHO\_REG (0x0094)



**GDMA\_INLINK\_DSCR\_BFO\_CHO** Represents the address of the current receive descriptor x that is pre-read. (RO)

Register 2.20. GDMA\_IN\_DSCR\_BF1\_CHO\_REG (0x0098)



**GDMA\_INLINK\_DSCR\_BF1\_CHO** Represents the address of the previous receive descriptor x-1 that is pre-read. (RO)

Register 2.21. GDMA\_OUTFIFO\_STATUS\_CHO\_REG (0x00D8)

(reserved)						GDMA_OUT_REMAIN_UNDER_4B_CHO GDMA_OUT_REMAIN_UNDER_3B_CHO GDMA_OUT_REMAIN_UNDER_2B_CHO GDMA_OUT_REMAIN_UNDER_1B_CHO																(reserved)								GDMA_OUTFIFO_CNT_CHO		GDMA_OUTFIFO_EMPTY_CHO GDMA_OUTFIFO_FULL_CHO	
31	27	26	25	24	23	22											8	7						2	1	0							
0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Reset	

Reset

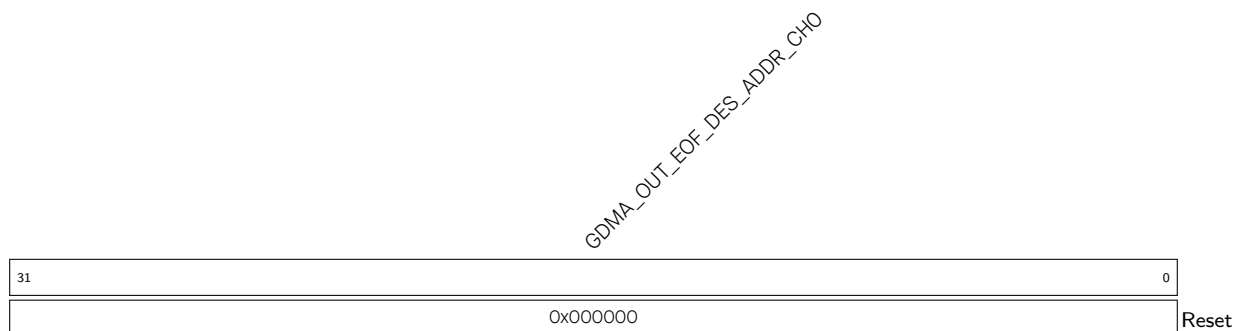
**GDMA\_OUTFIFO\_FULL\_CHO** L1 TX FIFO full signal for TX channel 0. (RO)**GDMA\_OUTFIFO\_EMPTY\_CHO** L1 TX FIFO empty signal for TX channel 0. (RO)**GDMA\_OUTFIFO\_CNT\_CHO** The register stores the byte number of the data in L1 TX FIFO for TX channel 0. (RO)**GDMA\_OUT\_REMAIN\_UNDER\_1B\_CHO** Reserved. (RO)**GDMA\_OUT\_REMAIN\_UNDER\_2B\_CHO** Reserved. (RO)**GDMA\_OUT\_REMAIN\_UNDER\_3B\_CHO** Reserved. (RO)**GDMA\_OUT\_REMAIN\_UNDER\_4B\_CHO** Reserved. (RO)

Register 2.22. GDMA\_OUT\_STATE\_CHO\_REG (0x00E4)

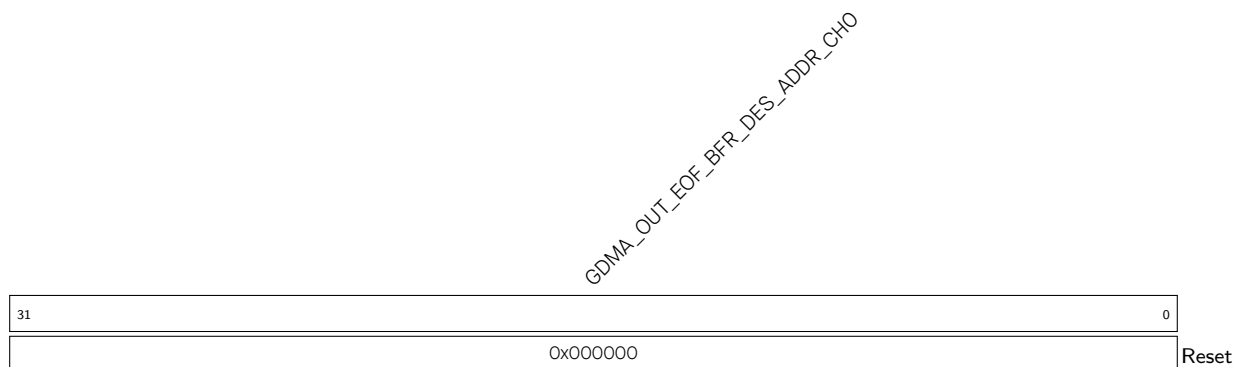
(reserved)										GDMA_OUT_STATE_CHO				GDMA_OUTLINK_DSCR_ADDR_CHO																			
31										23	22	20		19	18	17																	0
0 0 0 0 0 0 0 0 0 0										0		0		0																	Reset		

Reset

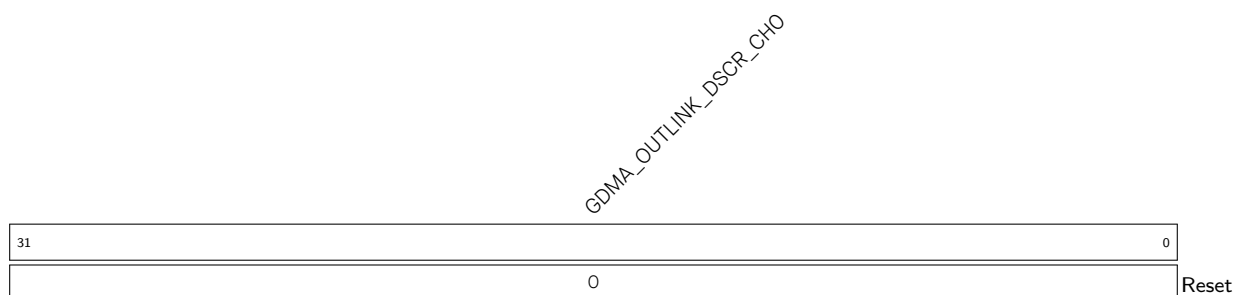
**GDMA\_OUTLINK\_DSCR\_ADDR\_CHO** This register stores the lower 18 bits of the next receive descriptor address that is pre-read (but not processed yet). If the current receive descriptor is the last descriptor, then this field represents the address of the current receive descriptor. (RO)**GDMA\_OUT\_DSCR\_STATE\_CHO** Reserved. (RO)**GDMA\_OUT\_STATE\_CHO** Reserved. (RO)

**Register 2.23. GDMA\_OUT\_EOF\_DES\_ADDR\_CHO\_REG (0x00E8)**

**GDMA\_OUT\_EOF\_DES\_ADDR\_CHO** This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

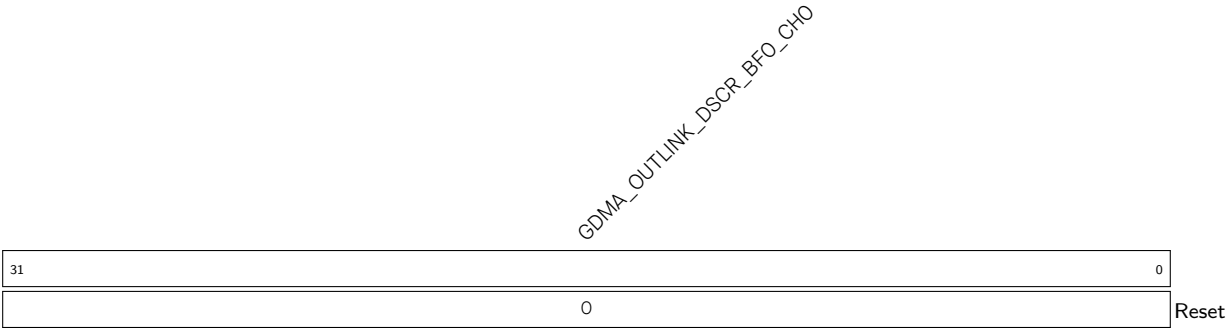
**Register 2.24. GDMA\_OUT\_EOF\_BFR\_DES\_ADDR\_CHO\_REG (0x00EC)**

**GDMA\_OUT\_EOF\_BFR\_DES\_ADDR\_CHO** This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

**Register 2.25. GDMA\_OUT\_DSCR\_CHO\_REG (0x00F0)**

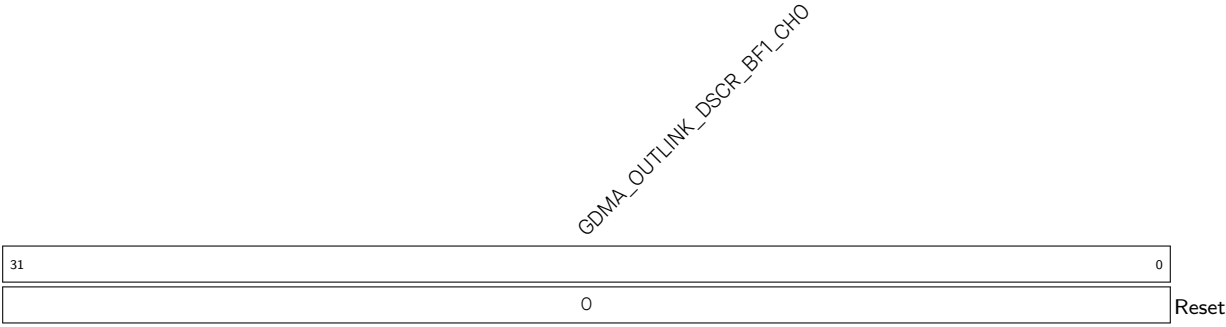
**GDMA\_OUTLINK\_DSCR\_CHO** Represents the address of the next transmit descriptor y+1 pointed by the current transmit descriptor that is pre-read. (RO)

Register 2.26. GDMA\_OUT\_DSCR\_BFO\_CHO\_REG (0x00F4)



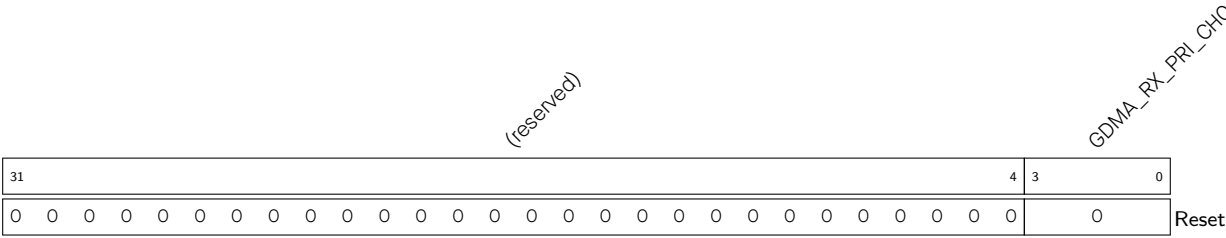
**GDMA\_OUTLINK\_DSCR\_BFO\_CHO** Represents the address of the current transmit descriptor y that is pre-read. (RO)

Register 2.27. GDMA\_OUT\_DSCR\_BF1\_CHO\_REG (0x00F8)



**GDMA\_OUTLINK\_DSCR\_BF1\_CHO** Represents the address of the previous transmit descriptor y-1 that is pre-read. (RO)

Register 2.28. GDMA\_IN\_PRI\_CHO\_REG (0x009C)



**GDMA\_RX\_PRI\_CHO** The priority of RX channel 0. The larger the value, the higher the priority. (R/W)

### Register 2.29. GDMA\_OUT\_PRI\_CHO\_REG (0x00FC)

[illegible]

**GDMA\_TX\_PRI\_CHO** The priority of TX channel 0. The larger the value, the higher the priority. (R/W)

### Register 2.30. GDMA\_IN\_PERI\_SEL\_CHO\_REG (0x00A0)

Diagram illustrating the structure of the GDMA\_PERL\_IN\_SEL\_CH0 register. The register is 32 bits wide, divided into two main sections:

- (reserved)**: The upper portion of the register, spanning from bit 31 down to bit 5.
- GDMA\_PERL\_IN\_SEL\_CH0**: The lower portion of the register, spanning from bit 5 down to bit 0. This section is further divided into:
  - A 26-bit field (bits 31-6) containing 26 zeros.
  - A 6-bit field (bits 5-0) containing the value 0x3f.

The label "Reset" is positioned at the bottom right of the diagram.

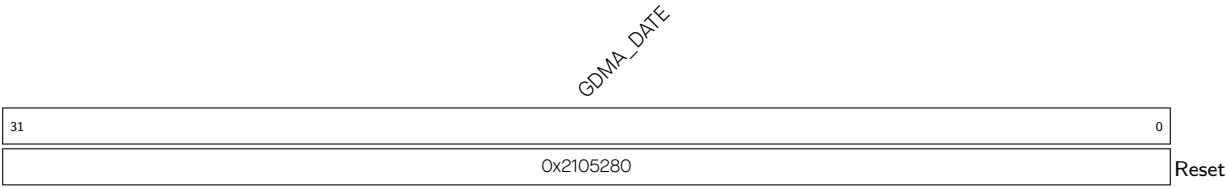
**GDMA\_PERI\_IN\_SEL\_CHO** This register is used to select peripheral for RX channel 0. 0: SPI2. 1: reserved. 2: reserved. 3: reserved. 4: reserved. 5: reserved. 6: reserved. 7: reserved. 8: reserved; 9 ~ 63: Invalid. (R/W)

### Register 2.31. GDMA\_OUT\_PERI\_SEL\_CHO\_REG (0x0100)

Diagram illustrating the structure of the `GDMA_PERI_OUT_SEL_OUT` register. The register is 32 bits wide, divided into two 16-bit sections. The top 16 bits are labeled `(reserved)`. The bottom 16 bits are labeled `GDMA_PERI_OUT_SEL_OUT`. The bottom 16 bits are further divided into a 5-bit field labeled `GDMA_PERI_OUT_SEL_OUT` and a 11-bit field labeled `Reset`.

**GDMA\_PERI\_OUT\_SEL\_CHO** This register is used to select peripheral for TX channel 0. 0: SPI2. 1: reserved. 2: reserved. 3: reserved. 4: reserved. 5: reserved. 6: reserved. 7: SHA. 8: reserved; 9 ~ 63: Invalid. (R/W)

Register 2.32. GDMA\_DATE\_REG (0x0048)



**GDMA\_DATE** This is the version control register. (R/W)

## Part II

# Memory Organization

This part provides insights into the system's memory structure, discussing system memory and eFuse, offering a framework for understanding memory-related subsystems.



## Chapter 3

# System and Memory

### 3.1 Overview

The ESP8684 is an ultra-low-power and highly-integrated system with a 32-bit RISC-V single-core processor and a four-stage pipeline that operates at up to 120 MHz. All internal memory, external memory, and peripherals are located on the CPU buses.

### 3.2 Features

ESP8684's system and memory has the following features:

- **Address Space**
  - 848 KB of internal memory address space accessed from the instruction bus
  - 576 KB of internal memory address space accessed from the data bus
  - 828 KB of peripheral address space
  - 4 MB of external memory virtual address space accessed from the instruction bus
  - 4 MB of external memory virtual address space accessed from the data bus
  - 576 KB of internal DMA address space
- **Internal Memory**
  - 576 KB of internal ROM
  - 272 KB of internal SRAM
- **External Memory**
  - Supports up to 16 MB external flash
- **Peripheral Space**
  - 22 modules/peripherals in total
- **GDMA**
  - 2 GDMA-supported modules/peripherals

Figure 3.2-1 illustrates the system structure and address mapping.

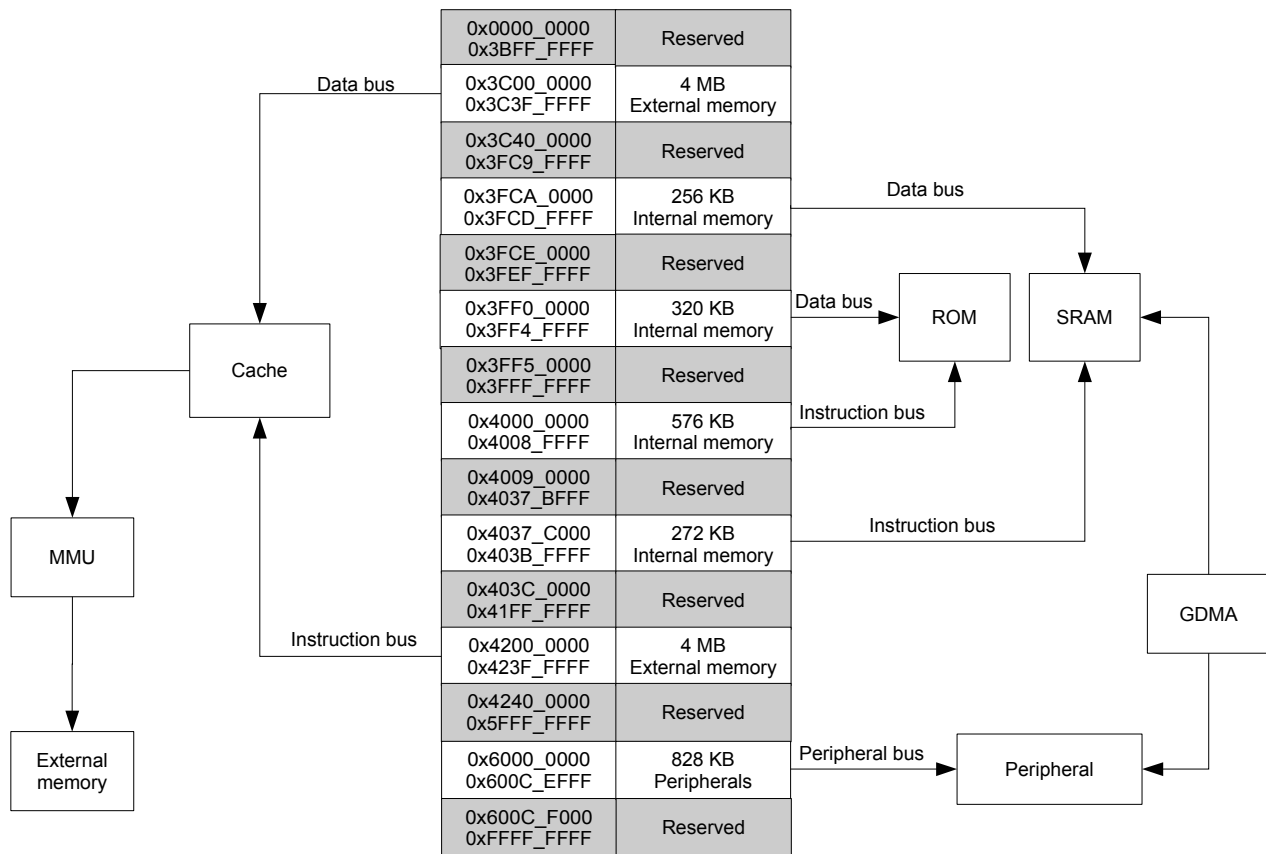


Figure 3.2-1. System Structure and Address Mapping

**Note:**

- The address space with gray background is not available to users.
- The range of addresses available in the address space may be larger than the actual available memory of a particular type.

## 3.3 Functional Description

### 3.3.1 Address Mapping

Addresses below 0x4000\_0000 are accessed using the data bus. Addresses in the range of 0x4000\_0000 ~ 0x4FFF\_FFFF are accessed using the instruction bus. Addresses over and including 0x5000\_0000 are accessed using the peripheral bus.

Both the data bus and instruction bus are little-endian. The CPU can access data via the data bus using single-byte, double-byte, four-byte alignment. The CPU can also access data via the instruction bus, but only in four-byte aligned manner.

The CPU can:

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

- directly access modules/peripherals via the peripheral bus.

Figure 3.2-1 shows the address ranges on the data bus, instruction bus and peripheral bus as well as their corresponding target memory.

Some internal and external memory can be accessed via both the data bus and instruction bus. In such cases, the CPU can access the same memory using multiple addresses.

### 3.3.2 Internal Memory

The ESP8684 consists of the following two types of internal memory:

- Internal ROM (576 KB): The Internal ROM of the ESP8684 is a read-only memory which cannot be programmed. Internal ROM contains the ROM code (software instructions and some software read-only data) of some low-level system software.
- Internal SRAM (272 KB): The Internal Static RAM (SRAM) is a volatile memory that can be quickly accessed by the CPU (generally within a single CPU clock cycle).
  - A part of the SRAM can be configured to operate as a cache for external memory access.
  - Some parts of the SRAM can only be accessed via the CPU's instruction bus.
  - Some parts of the SRAM can be accessed via both the CPU's instruction bus and the CPU's data bus.

Based on the two different types of internal memory described above, the internal memory of the ESP8684 is split into two segments: Internal ROM (576 KB) and Internal SRAM (272 KB).

However, within each segment, there may be different bus access restrictions (e.g., some parts of the segment may only be accessible by the CPU's data bus). Therefore, segments are also further divided into parts. Table 3.3-1 describes each part of internal memory and their address ranges on the data bus and instruction bus.

**Table 3.3-1. Internal Memory Address Mapping**

Bus Type	Boundary Address		Size (KB)	Target
	Low Address	High Address		
Data bus	0x3FF0_0000	0x3FF4_FFFF	320	Internal ROM 1
	0x3FCA_0000	0x3FCD_FFFF	256	Internal SRAM 1
Instruction bus	0x4000_0000	0x4003_FFFF	256	Internal ROM 0
	0x4004_0000	0x4008_FFFF	320	Internal ROM 1
	0x4037_C000	0x4037_FFFF	16	Internal SRAM 0
	0x4038_0000	0x403B_FFFF	256	Internal SRAM 1

#### 1. Internal ROM 0

Internal ROM 0 is a 256 KB, read-only memory space, addressed by the CPU only through the instruction bus via 0x4000\_0000 ~ 0x4003\_FFFF, as shown in Table 3.3-1.

#### 2. Internal ROM 1

Internal ROM 1 is a 320 KB, read-only memory space, addressed by the CPU through the instruction bus via 0x4004\_0000 ~ 0x4008\_FFFF or through the data bus via 0x3FF0\_0000 ~ 0x3FF4\_FFFF in the same order, as shown in Table 3.3-1.

This means, for example, address 0x4004\_0000 and 0x3FF0\_0000 correspond to the same word, 0x4004\_0004 and 0x3FF0\_0004 correspond to the same word, 0x4004\_0008 and 0x3FF0\_0008 correspond to the same word, etc.

### 3. Internal SRAM 0

Internal SRAM 0 is a 16 KB, read-and-write memory space, addressed by the CPU through the instruction bus via the range described in Table 3.3-1.

This memory can be configured as instruction cache to store cache instructions or read-only data of the external memory. In this case, the configured memory cannot be accessed by the CPU.

### 4. Internal SRAM 1

Internal SRAM 1 is a 256 KB, read-and-write memory space, addressed by the CPU through the data bus or instruction bus, in the same order (the same meaning as the explanation in 3.3.2 *Internal ROM 1*), via the ranges described in Table 3.3-1.

## 3.3.3 External Memory

ESP8684 supports SPI, Dual SPI, Quad SPI, and QPI interfaces that allow connection to multiple external flash chips. The chip supports hardware manual encryption and automatic decryption based on XTS-AES algorithm to protect user programs and data in the external flash.

### 3.3.3.1 External Memory Address Mapping

The CPU accesses the external memory via the cache. According to the MMU (Memory Management Unit) settings, the cache maps the CPU's address to the external memory's physical address. Due to this address mapping, the ESP8684 can address up to 16 MB external flash.

Using the cache, ESP8684 is able to support the following address space mappings. Note that the instruction bus address space (4 MB) and the data bus address space (4 MB) is always shared.

- Up to 4 MB instruction bus address space can be mapped into the external flash. The mapped address space is organized as individual 64-KB blocks.
- Up to 4 MB data bus (read-only) address space can be mapped into the external flash. The mapped address space is organized as individual 64-KB blocks.

Table 3.3-2 lists the mapping between the cache and the corresponding address ranges on the data bus and instruction bus.

**Table 3.3-2. External Memory Address Mapping**

Bus Type	Boundary Address		Size (MB)	Target
	Low Address	High Address		
Data bus (read-only)	0x3C00_0000	0x3C3F_FFFF	4	Uniform Cache

Cont'd on next page

Table 3.3-2 – cont'd from previous page

Bus Type	Boundary Address		Size (MB)	Target
	Low Address	High Address		
Instruction bus	0x4200_0000	0x423F_FFFF	4	Uniform Cache

### 3.3.3.2 Cache

As shown in Figure 3.3-1, ESP8684 has a read-only uniform cache which is four-way set-associative, its size is 16 KB and its block size is 32 bytes. When cache is active, some internal memory space will be occupied by cache (see Internal SRAM 0 in Section 3.3.2).

The uniform cache is accessible by the instruction bus and the data bus at the same time, but can only respond to one of them at a time. When a cache miss occurs, the cache controller will initiate a request to the external memory.

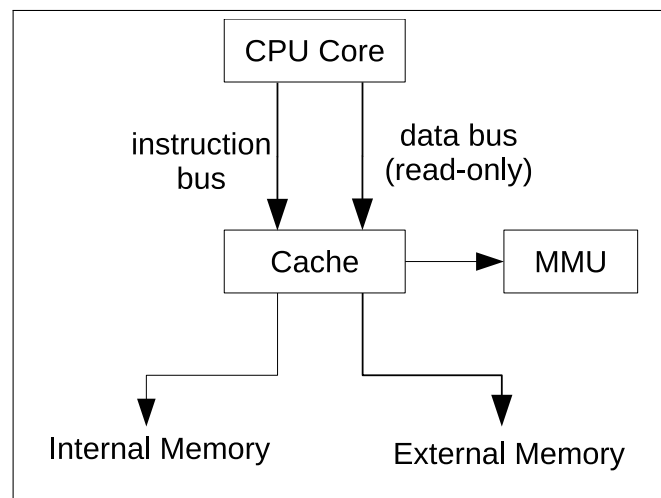


Figure 3.3-1. Cache Structure

### 3.3.3.3 Cache Operations

ESP8684 cache supports the following operations:

1. **Invalidate:** This operation is used to clear valid data in the cache. After this operation is completed, the data will only be stored in the external memory. The CPU needs to access the external memory in order to read this data. There are two types of invalidate-operation: automatic invalidation (Auto-Invalidate) and manual invalidation (Manual-Invalidate). Manual-Invalidate is performed only on data in the specified area in the cache, while Auto-Invalidate is performed on all data in the cache.
2. **Preload:** This operation is used to load instructions and data into the cache in advance. The minimum unit of preload-operation is one block (32 bytes). There are two types of preload-operation: manual preload (Manual-Preload) and automatic preload (Auto-Preload). Manual-Preload means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Auto-Preload means the hardware prefetches a piece of continuous data according to the current address where the cache hits or misses (depending on configuration).
3. **Lock/Unlock:** The lock operation is used to prevent the data in the cache from being easily replaced. There are two types of lock: prelock and manual lock. When prelock is enabled, the cache locks the

data in the specified area when filling the missing data to cache memory, while the data outside the specified area will not be locked. When manual lock is enabled, the cache checks the data that is already in the cache memory and only locks the data in the specified area, and leaves the data outside the specified area unlocked. When there are missing data, the cache will replace the data in the unlocked way first, so the data in the locked way is always stored in the cache and will not be replaced. But when all ways within the cache are locked, the cache will replace data, as if it was not locked. Unlocking is the reverse of locking, except that it only can be done manually.

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

### 3.3.4 GDMA Address Space

The GDMA (General Direct Memory Access) peripheral in ESP8684 can provide DMA (Direct Memory Access) services including:

- Data transfers between different locations of internal memory;
- Data transfers between modules/peripherals and internal memory.

The GDMA can read and write to Internal SRAM 1 via the same address as the data bus. Specifically, GDMA accesses Internal SRAM 1 via 0x3FCA\_0000 ~ 0x3FCD\_FFFF. Note that GDMA cannot access the internal memory occupied by the cache.

There are two peripherals/modules that can work together with GDMA, i.e., SPI2 and SHA Accelerator. These two peripherals share one channel in GDMA and cannot enable GDMA function at the same time.

Peripherals/modules with GDMA function can access any memory available to GDMA. For more information, please refer to Chapter 2 [GDMA Controller \(GDMA\)](#).

### 3.3.5 Modules/Peripherals

The CPU can access modules/peripherals via 0x6000\_0000 ~ 0x600C\_EFFF shared by the peripheral bus.

#### 3.3.5.1 Module/Peripheral Address Mapping

Table 3.3-3 lists all the modules/peripherals and their respective address ranges. Note that the address space of specific modules/peripherals is defined by "Boundary Address" (including both Low Address and High Address).

**Table 3.3-3. Module/Peripheral Address Mapping**

Target	Boundary Address		Size (KB)	Notes
	Low Address	High Address		
<a href="#">UART Controller 0</a>	0x6000_0000	0x6000_0FFF	4	
Reserved	0x6000_1000	0x6000_1FFF		
<a href="#">SPI Controller 1</a>	0x6000_2000	0x6000_2FFF	4	
<a href="#">SPI Controller 0</a>	0x6000_3000	0x6000_3FFF	4	
<a href="#">GPIO</a>	0x6000_4000	0x6000_4FFF	4	

Continued on the next page...

Table 3.3-3 – Continued from the previous page

Target	Boundary Address		Size (KB)	Notes
	Low Address	High Address		
Reserved	0x6000_5000	0x6000_7FFF		
<a href="#">Low-Power Management</a>	0x6000_8000	0x6000_8FFF	4	
<a href="#">IO MUX</a>	0x6000_9000	0x6000_9FFF	4	
Reserved	0x6000_A000	0x6000_CFFF		
MISC	0x6000_D000	0x6000_DFFF	4	
Reserved	0x6000_E000	0x6000_FFFF		
<a href="#">UART Controller 1</a>	0x6001_0000	0x6001_0FFF	4	
Reserved	0x6001_1000	0x6001_2FFF		
<a href="#">I2C Controller</a>	0x6001_3000	0x6001_3FFF	4	
Reserved	0x6001_4000	0x6001_8FFF		
<a href="#">LED PWM Controller</a>	0x6001_9000	0x6001_9FFF	4	
Reserved	0x6001_A000	0x6001_EFFF		
<a href="#">Timer Group 0</a>	0x6001_F000	0x6001_FFFF	4	
Reserved	0x6002_0000	0x6002_2FFF		
<a href="#">System Timer</a>	0x6002_3000	0x6002_3FFF	4	
<a href="#">SPI Controller 2</a>	0x6002_4000	0x6002_4FFF	4	
Reserved	0x6002_5000	0x6002_5FFF		
SYSCON	0x6002_6000	0x6002_6FFF	4	
Reserved	0x6002_7000	0x6003_AFFF		
<a href="#">SHA Accelerator</a>	0x6003_B000	0x6003_BFFF	4	
<a href="#">ECC Accelerator</a>	0x6003_E000	0x6003_EFFF	4	
Reserved	0x6002_C000	0x6003_EFFF		
<a href="#">GDMA Controller</a>	0x6003_F000	0x6003_FFFF	4	
<a href="#">ADC Controller</a>	0x6004_0000	0x6004_0FFF	4	
Reserved	0x6004_1000	0x600B_FFFF		
<a href="#">System Registers</a>	0x600C_0000	0x600C_0FFF	4	
<a href="#">Sensitive Registers</a>	0x600C_1000	0x600C_1FFF	4	
<a href="#">Interrupt Matrix</a>	0x600C_2000	0x600C_2FFF	4	
Reserved	0x600C_3000	0x600C_3FFF		
Configure Cache	0x600C_4000	0x600C_DFFF	40	
Reserved	0x600C_E000	0x600C_DFFF		
<a href="#">Debug Assist</a>	0x600C_E000	0x600C_EFFF	4	

## Chapter 4

### eFuse Controller (eFuse)

#### 4.1 Overview

ESP8684 contains a 1024-bit eFuse memory to store parameters and user data. The parameters include control parameters for some hardware modules, system data parameters and keys used for the decryption module. Once an eFuse bit is programmed to 1, it can never be reverted to 0. The eFuse controller programs individual bits of parameters in eFuse according to user configurations. From outside the chip, eFuse data can only be read via the eFuse Controller. If read-protection for some data is not enabled, that data is readable from outside the chip. If read-protection is enabled, that data can not be read from outside the chip. In all cases, however, some keys stored in eFuse can still be used internally by hardware cryptography modules such as Digital Signature, HMAC, etc., without exposing this data to the outside world.

#### 4.2 Features

The eFuse controller has the following features:

- 1024-bit one-time programmable storage, in which 256-bit is reserved for users
- Configurable write protection
- Configurable read protection
- Various hardware encoding schemes against data corruption in the eFuse memory

#### 4.3 Functional Description

##### 4.3.1 Structure

eFuse data is organized in 4 blocks (BLOCK0 ~ BLOCK3).

BLOCK0 holds control parameters for most hardware modules.

Table 4.3-1 lists all the parameters in BLOCK0 that can be accessed (read and used) by users and their offsets, bit widths, as well as information on whether they can be used by hardware, which bits are write-protected, and corresponding descriptions.

The [EFUSE\\_WR\\_DIS](#) parameter is used to control the writing of other parameters, while [EFUSE\\_RD\\_DIS](#) is used to disable users from reading BLOCK3. For more information on these two parameters, please see Section 4.3.1.1 and Section 4.3.1.2.



Table 4.3-1. Parameters in BLOCK0

Parameters	Bit Width	Accessible by Hardware	Write Protection by <a href="#">EFUSE_WR_DIS</a> Bit Number	Description
<a href="#">EFUSE_WR_DIS</a>	8	Y	N/A	Disable writing of individual eFuses
<a href="#">EFUSE_RD_DIS</a>	2	Y	0	Disable users from reading eFuse BLOCK3
<a href="#">EFUSE_WDT_DELAY_SEL</a>	2	Y	1	Represent RTC watchdog timeout threshold
<a href="#">EFUSE_DIS_PAD_JTAG</a>	1	Y	1	Disable JTAG permanently
<a href="#">EFUSE_DIS_DOWNLOAD_ICACHE</a>	1	Y	1	Disable iCache in download mode
<a href="#">EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT</a>	1	Y	2	Disable manual flash encryption in download boot modes
<a href="#">EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT</a>	3	Y	2	Enable SPI boot encryption and decryption
<a href="#">EFUSE_XTS_KEY_LENGTH_256</a>	1	Y	2	Represent key length for XTS_AES
<a href="#">EFUSE_UART_PRINT_CONTROL</a>	2	N	3	Represent UART boot message output mode
<a href="#">EFUSE_FORCE_SEND_RESUME</a>	1	N	3	Force ROM code to send an SPI flash resume command during SPI boot
<a href="#">EFUSE_DIS_DOWNLOAD_MODE</a>	1	N	3	Disable all Download modes
<a href="#">EFUSE_DIS_DIRECT_BOOT</a>	1	N	3	Disable Direct_boot mode
<a href="#">EFUSE_ENABLE_SECURITY_DOWNLOAD</a>	1	N	3	Enable UART secure download mode
<a href="#">EFUSE_FALSH_TPUW</a>	4	N	3	Represents flash startup delay after SoC is powered up
<a href="#">EFUSE_SECURE_BOOT_EN</a>	1	N	2	Enable secure boot
<a href="#">EFUSE_SECURE_VERSION</a>	4	N	4	Secure version
<a href="#">EFUSE_CUSTOM_MAC_USED</a>	1	N	4	Enable customized MAC writing

Table 4.3-2 lists parameter information stored in BLOCK1 ~ BLOCK3.

Table 4.3-2. Parameters in BLOCK1 to BLOCK3

BLOCK	Parameters	Bit Width	Accessible by Hardware	Write Protection by <a href="#">EFUSE_WR_DIS</a> Bit Number	Read Protection by <a href="#">EFUSE_RD_DIS</a> Bit Number	Description
BLOCK1	EFUSE_CUSTOMED_MAC	88	N	5	N/A	Customize MAC address or user data
BLOCK2	EFUSE_SYS_DATA_PART1	48	N	6	N/A	MAC address
		208	N	6	N/A	System data
BLOCK3	EFUSE_KEY0	128	Y	7	[0]	KEY or user data
		128	Y	7	[1]	KEY or user data

BLOCK1 ~ BLOCK3 use the RS coding scheme, so there are some restrictions on writing to these parameters. For more detailed information, please refer to Section 4.3.1.3 and Section 4.3.2.

### 4.3.1.1 EFUSE\_WR\_DIS

Parameter [EFUSE\\_WR\\_DIS](#) determines whether individual eFuse parameters are write-protected. After [EFUSE\\_WR\\_DIS](#) has been programmed, execute an eFuse read operation so that the write-protection status would take effect.

Column "Write Protection by [EFUSE\\_WR\\_DIS](#) Bit Number" in Table 4.3-1 and Table 4.3-2 lists the specific bits in [EFUSE\\_WR\\_DIS](#) that disable writing.

When the write-protect bit of a parameter is set to 0, it means that this parameter is not write-protected and can be programmed, unless it has been programmed before.

When the write-protect bit of a parameter is set to 1, it means that this parameter is write-protected and none of its bits can be modified, with non-programmed bits always remaining 0 while programmed bits always remaining 1. That is to say, if a parameter is write-protected, it will always remain in this state and cannot be changed.

### 4.3.1.2 EFUSE\_RD\_DIS

Only parameters in BLOCK3 can be read-protected to prevent any access from outside of the chip as shown in column "Read Protection by [EFUSE\\_RD\\_DIS](#) Bit Number" of Table 4.3-2. After [EFUSE\\_RD\\_DIS](#) has been programmed, execute an eFuse read operation so that the read-protection status would take effect.

If the corresponding [EFUSE\\_RD\\_DIS](#) bit is 0, then the eFuse block can be read by users; if the corresponding [EFUSE\\_RD\\_DIS](#) bit is 1, then the parameter controlled by this bit is user protected.

Other parameters that are not in BLOCK3 can always be read by users.

### 4.3.1.3 Data Storage

Internally, eFuse controller uses hardware encoding schemes to protect data from corruption, which are invisible for users.

All BLOCK0 parameters except for [EFUSE\\_WR\\_DIS](#) are stored with four backups, meaning each bit is stored four times. This backup scheme is not visible to users.

Except for [EFUSE\\_WR\\_DIS](#) which is 8-bit, all other parameters in BLOCK0 are 32-bit. Therefore, BLOCK0 occupies  $(8 + 32 * 4 = 136)$  bits of storage totally in eFuse memory.

BLOCK1 ~ BLOCK3 use RS (44, 32) coding scheme that supports up to 6 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is  $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ .

The shift register circuit shown in Figure 4.3-1 and 4.3-2 processes 32-byte data using RS (44, 32). This coding scheme encodes 32 bytes of data into 44 bytes:

- Byte 0 ~ 31 is the data itself
- Byte 32 ~ 43 is the parity byte stored in the 8-bit trigger DFF1, DFF2,..., DFF12 (where  $gf\_mul\_n$  (n is an integer) is the result of multiplying a byte of data in the  $GF(2^8)$  field with the element  $\alpha^n$ )

Then, hardware will write the 44-byte data to eFuse memory. The eFuse controller will automatically decode the data and correct errors when reading the eFuse block.

Because the RS check codes are generated on the entire 256-bit eFuse block, each block can only be written once.

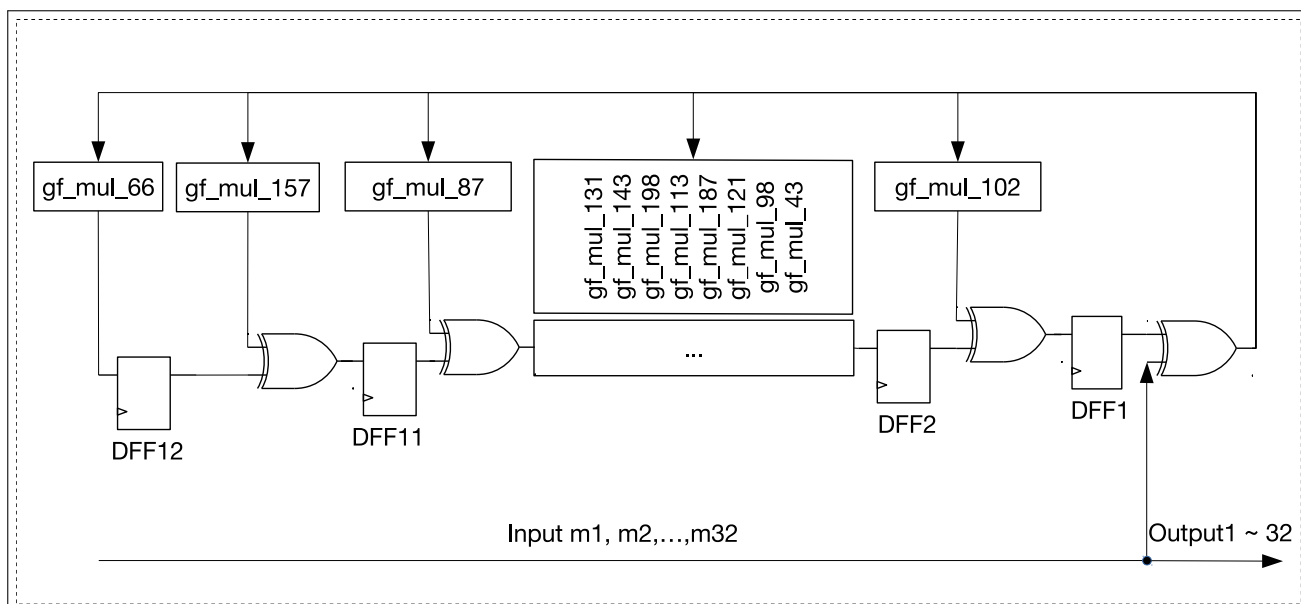


Figure 4.3-1. Shift Register Circuit (former 32-byte)

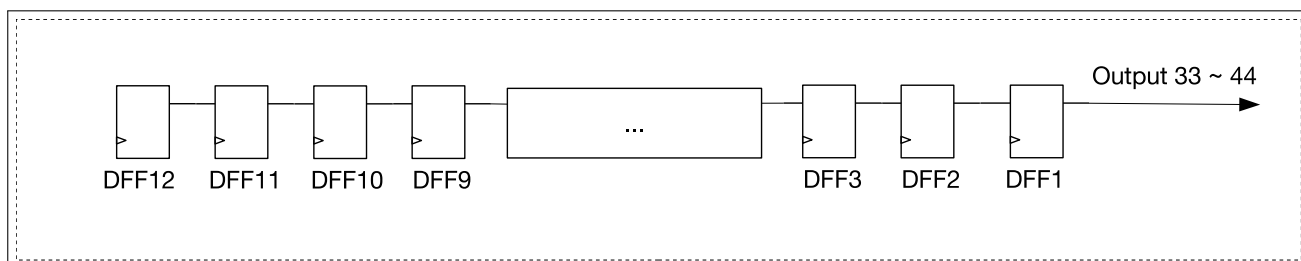


Figure 4.3-2. Shift Register Circuit (latter 12-byte)

Since the size of BLOCK1 is less than 256-bit, the unused bits will be treated as 0 by hardware during the RS (44, 32) decoding. Thus the final coding result will not be affected.

Among blocks using the RS (44, 32) coding scheme, the parameter in BLOCK1 is 88-bit, and the RS check code is 96-bit, so BLOCK1 occupies  $(88 + 96 = 184)$  bits in eFuse memory. The parameter in BLOCK2 and BLOCK3 is 256-bit respectively, and the RS check code is 96-bit, so these two blocks occupy  $((256 + 96) * 2 = 704)$  bits in eFuse memory.

## 4.3.2 Programming of Parameters

The eFuse controller can only program eFuse parameters in one block at a time. BLOCK0 ~ BLOCK3 share the same address range to store the parameters to be programmed. Configure parameter `EFUSE_BLK_NUM` to indicate which block should be programmed.

### Programming BLOCK0

Set the `EFUSE_BLK_NUM` field to 0. The parameters to be programmed in BLOCK0 are stored in `EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA1_REG`. The data in `EFUSE_PGM_DATA2_REG ~ EFUSE_PGM_DATA7_REG` and `EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG` registers does not affect the programming of BLOCK0.

### Programming BLOCK1

Set the [EFUSE\\_BLK\\_NUM](#) field to 1. The parameters to be programmed in BLOCK1 are stored in [EFUSE\\_PGM\\_DATA0\\_REG ~ EFUSE\\_PGM\\_DATA2\\_REG](#), while the corresponding RS check codes are stored in [EFUSE\\_PGM\\_CHECK\\_VALUE0\\_REG ~ EFUSE\\_PGM\\_CHECK\\_VALUE2\\_REG](#). The data in [EFUSE\\_PGM\\_DATA3\\_REG ~ EFUSE\\_PGM\\_DATA7\\_REG](#) registers does not affect the programming of BLOCK1.

### Programming BLOCK2 ~ 3

Set the [EFUSE\\_BLK\\_NUM](#) field to 2 or 3, respectively. The parameters to be programmed are stored in [EFUSE\\_PGM\\_DATA0\\_REG ~ EFUSE\\_PGM\\_DATA7\\_REG](#), while the corresponding RS check code is stored in [EFUSE\\_PGM\\_CHECK\\_VALUE0\\_REG ~ EFUSE\\_PGM\\_CHECK\\_VALUE2\\_REG](#).

### Programming process

The process of programming parameters is as follows:

1. Configure the value of parameter [EFUSE\\_BLK\\_NUM](#) to determine the block to be programmed.
2. Write the parameters to be programmed to registers [EFUSE\\_PGM\\_DATA0\\_REG ~ EFUSE\\_PGM\\_DATA7\\_REG](#) and [EFUSE\\_PGM\\_CHECK\\_VALUE0\\_REG ~ EFUSE\\_PGM\\_CHECK\\_VALUE2\\_REG](#).
3. Make sure the eFuse programming voltage VDDQ is configured correctly as described in Section 4.3.4.
4. Set the field [EFUSE\\_OP\\_CODE](#) of register [EFUSE\\_CONF\\_REG](#) to 0x5A5A.
5. Set the field [EFUSE\\_PGM\\_CMD](#) of register [EFUSE\\_CMD\\_REG](#) to 1.
6. Poll register [EFUSE\\_CMD\\_REG](#) until it is 0x0, or wait for a PGM\_DONE interrupt. For more information on how to identify a PGM/READ\_DONE interrupt, please see the end of Section 4.3.3.
7. Clear the parameters in [EFUSE\\_PGM\\_DATA0\\_REG ~ EFUSE\\_PGM\\_DATA7\\_REG](#) and [EFUSE\\_PGM\\_CHECK\\_VALUE0\\_REG ~ EFUSE\\_PGM\\_CHECK\\_VALUE2\\_REG](#).
8. Trigger an eFuse read operation (see Section 4.3.3) to update eFuse registers with the new values.
9. Check corresponding error registers. If the value read is not 0, above 1 ~ 7 steps should be executed again to prevent programming insufficiency. For different eFuse blocks, the corresponding error registers that need to be checked are listed as follows:
  - BLOCK0: [EFUSE\\_RD\\_REPEAT\\_ERR\\_REG](#)
  - BLOCK1: [EFUSE\\_RD\\_RS\\_ERR\\_REG](#)[3:0]
  - BLOCK2: [EFUSE\\_RD\\_RS\\_ERR\\_REG](#)[7:4]
  - BLOCK3: [EFUSE\\_RD\\_RS\\_ERR\\_REG](#)[11:8]

### Restrictions

In BLOCK0, each bit can be programmed separately. However, we recommend to minimize programming cycles and program all the bits of a parameter in one programming action. In addition, after all parameters controlled by a certain bit of [EFUSE\\_WR\\_DIS](#) are programmed, that bit should be immediately programmed, or you could even program these parameters and the controlling bit at the same time. By doing so, programming can be effectively protected by preventing messing the programming of [EFUSE\\_WR\\_DIS](#) with the bits controlled by it. Repeated programming of already programmed bits is strictly forbidden. Otherwise, programming errors will occur.

BLOCK2 cannot be programmed by users as it has been programmed at manufacturing.

Both BLOCK1 and BLOCK3 can only be programmed once. Repeated programming is not allowed.

### 4.3.3 User Read of Parameters

Users cannot read data programmed in the eFuse controller directly. The eFuse controller reads all programmed data and stores the results to their corresponding programming registers in its memory space. Then, users can read eFuse bits by reading the registers that start with EFUSE\_RD\_. Details are provided in Table 4.3-3.

Table 4.3-3. Registers information

BLOCK	Read Registers	Programming Registers
0	<a href="#">EFUSE_RD_WR_DIS_REG</a>	<a href="#">EFUSE_PGM_DATA0_REG</a>
0	<a href="#">EFUSE_RD_REPEAT_DATA0_REG</a>	<a href="#">EFUSE_PGM_DATA1_REG</a>
1	<a href="#">EFUSE_RD_BLK1_DATA0 ~ 2_REG</a>	<a href="#">EFUSE_PGM_DATA0 ~ 2_REG</a>
2	<a href="#">EFUSE_RD_BLK2_DATA0 ~ 7_REG</a>	<a href="#">EFUSE_PGM_DATA0 ~ 7_REG</a>
3	<a href="#">EFUSE_RD_BLK3_DATA0 ~ 7_REG</a>	<a href="#">EFUSE_PGM_DATA0 ~ 7_REG</a>

#### Updating eFuse controller read registers

The eFuse controller reads eFuse memory to update corresponding registers. This read operation happens on system reset and can also be triggered manually by users as needed (e.g., if new eFuse values have been programmed). The process of triggering a read operation by users is as follows:

1. Set the field [EFUSE\\_OP\\_CODE](#) of register [EFUSE\\_CONF\\_REG](#) to 0x5AA5.
2. Set the field [EFUSE\\_READ\\_CMD](#) of register [EFUSE\\_CMD\\_REG](#) to 1.
3. Poll register [EFUSE\\_CMD\\_REG](#) until it is 0x0, or wait for a READ\_DONE interrupt. Information on how to identify a PGM/READ\_DONE interrupt is provided below in this section.
4. Read the values of each parameter from eFuse memory.

The eFuse controller read registers will hold all values until the next read operation.

#### Error detection

Error record registers allow users to detect if there are any inconsistencies between the parameters stored in the eFuse memory and the parameters read by the eFuse controller.

The [EFUSE\\_RD\\_REPEAT\\_ERR\\_REG](#) register indicates if there are any errors of programmed parameters (except for [EFUSE\\_WR\\_DIS](#)) in BLOCK0 (value 1 indicates an error is detected, and the bit becomes invalid; value 0 indicates no error).

The [EFUSE\\_RD\\_RS\\_ERR\\_REG](#) register stores the number of corrected bytes as well as the result of RS decoding during eFuse reading BLOCK1 ~ BLOCK3.

The values of above registers will be updated every time after the eFuse controller read registers have been updated.

#### Identifying programming/read operation

The methods to identify the completion of a programming/read operation are described below. Please note that bit 1 corresponds to a programming operation, and bit 0 corresponds to a read operation.

- Method 1:
  1. Poll bit 1/0 in register [EFUSE\\_INT\\_RAW\\_REG](#) until it becomes 1, which represents the completion of a program/read operation.
- Method 2:
  1. Set bit 1/0 in register [EFUSE\\_INT\\_ENA\\_REG](#) to 1 to enable the eFuse controller to post a PGM/READ\_DONE interrupt.
  2. Configure the Interrupt Matrix to enable the CPU to respond to eFuse controller interrupt signals, see Chapter 8 [Interrupt Matrix \(INTMTRX\)](#).
  3. Wait for the PGM/READ\_DONE interrupt.
  4. Set bit 1/0 in register [EFUSE\\_INT\\_CLR\\_REG](#) to 1 to clear the PGM/READ\_DONE interrupt.

#### Attention

When the eFuse controller updating registers, the [EFUSE\\_PGM\\_DATA<sub>n</sub>\\_REG](#) (n=0,1,..., 7) register will be re-used. Therefore, please do not write meaningful data to such register before the eFuse controller starts to update registers.

During the chip boot time, the eFuse controller will update eFuse data automatically and write it to registers that users can access. Users can get eFuse memory data by reading such registers. That is to say, it is no need to drive the eFuse controller manually again to update read registers.

### 4.3.4 eFuse VDDQ Timing

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

- [EFUSE\\_DAC\\_NUM](#) (the rising period of VDDQ): The default value of VDDQ is 2.5 V and the voltage increases by 0.01 V in each clock cycle. Thus, the default value of this parameter is 255.
- [EFUSE\\_DAC\\_CLK\\_DIV](#) (the clock divisor of VDDQ): The clock period to program VDDQ should be larger than 1  $\mu$ s.
- [EFUSE\\_PWR\\_ON\\_NUM](#) (the power-up time for VDDQ): The programming voltage should be stabilized after this time, which means the value of this parameter should be configured to exceed the result of [EFUSE\\_DAC\\_CLK\\_DIV](#) times [EFUSE\\_DAC\\_NUM](#).
- [EFUSE\\_PWR\\_OFF\\_NUM](#) (the power-down time for VDDQ): The value of this parameter should be larger than 10  $\mu$ s.

Table 4.3-4. Configuration of Default VDDQ Timing Parameters

<a href="#">EFUSE_DAC_NUM</a>	<a href="#">EFUSE_DAC_CLK_DIV</a>	<a href="#">EFUSE_PWR_ON_NUM</a>	<a href="#">EFUSE_PWR_OFF_NUM</a>
0xFF	0x28	0x3000	0x190

### 4.3.5 Parameters Used by Hardware Modules

Some hardware modules are directly connected to the eFuse peripheral in order to use the parameters listed in Table 4.3-1 and Table 4.3-2, specifically those marked with "Y" in columns "Accessible by Hardware". Users cannot intervene in this process.

### 4.3.6 Interrupts

- PGM\_DONE interrupt: Triggered when eFuse programming has finished. To enable this interrupt, set the [EFUSE\\_PGM\\_DONE\\_INT\\_ENA](#) field of register [EFUSE\\_INT\\_ENA\\_REG](#) to 1.
- READ\_DONE interrupt: Triggered when eFuse read has finished. To enable this interrupt, set the [EFUSE\\_READ\\_DONE\\_INT\\_ENA](#) field of register [EFUSE\\_INT\\_ENA\\_REG](#) to 1.



## 4.4 Register Summary

The addresses in this section are relative to eFuse controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

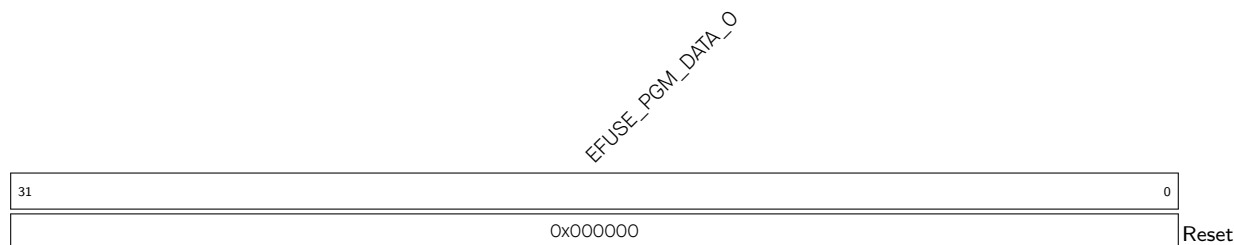
Name	Description	Address	Access
<b>PGM Data Register</b>			
<a href="#">EFUSE_PGM_DATA0_REG</a>	Register 0 that configures data to be programmed	0x0000	R/W
<a href="#">EFUSE_PGM_DATA1_REG</a>	Register 1 that configures data to be programmed	0x0004	R/W
<a href="#">EFUSE_PGM_DATA2_REG</a>	Register 2 that configures data to be programmed	0x0008	R/W
<a href="#">EFUSE_PGM_DATA3_REG</a>	Register 3 that configures data to be programmed	0x000C	R/W
<a href="#">EFUSE_PGM_DATA4_REG</a>	Register 4 that configures data to be programmed	0x0010	R/W
<a href="#">EFUSE_PGM_DATA5_REG</a>	Register 5 that configures data to be programmed	0x0014	R/W
<a href="#">EFUSE_PGM_DATA6_REG</a>	Register 6 that configures data to be programmed	0x0018	R/W
<a href="#">EFUSE_PGM_DATA7_REG</a>	Register 7 that configures data to be programmed	0x001C	R/W
<a href="#">EFUSE_PGM_CHECK_VALUE0_REG</a>	Register 0 that configures the RS code to be programmed	0x0020	R/W
<a href="#">EFUSE_PGM_CHECK_VALUE1_REG</a>	Register 1 that configures the RS code to be programmed	0x0024	R/W
<a href="#">EFUSE_PGM_CHECK_VALUE2_REG</a>	Register 2 that configures the RS code to be programmed	0x0028	R/W
<b>Read Data Register</b>			
<a href="#">EFUSE_RD_WR_DIS_REG</a>	Register 0 of BLOCK0 wr_dis data	0x002C	RO
<a href="#">EFUSE_RD_REPEAT_DATA0_REG</a>	Register 1 of BLOCK0 data	0x0030	RO
<a href="#">EFUSE_RD_BLK1_DATA0_REG</a>	Register 0 of BLOCK1 data	0x0034	RO
<a href="#">EFUSE_RD_BLK1_DATA1_REG</a>	Register 1 of BLOCK1 data	0x0038	RO
<a href="#">EFUSE_RD_BLK1_DATA2_REG</a>	Register 2 of BLOCK1 data	0x003C	RO
<a href="#">EFUSE_RD_BLK2_DATA0_REG</a>	Register 0 of BLOCK2 data	0x0040	RO
<a href="#">EFUSE_RD_BLK2_DATA1_REG</a>	Register 1 of BLOCK2 data	0x0044	RO
<a href="#">EFUSE_RD_BLK2_DATA2_REG</a>	Register 2 of BLOCK2 data	0x0048	RO
<a href="#">EFUSE_RD_BLK2_DATA3_REG</a>	Register 3 of BLOCK2 data	0x004C	RO
<a href="#">EFUSE_RD_BLK2_DATA4_REG</a>	Register 4 of BLOCK2 data	0x0050	RO
<a href="#">EFUSE_RD_BLK2_DATA5_REG</a>	Register 5 of BLOCK2 data	0x0054	RO
<a href="#">EFUSE_RD_BLK2_DATA6_REG</a>	Register 6 of BLOCK2 data	0x0058	RO
<a href="#">EFUSE_RD_BLK2_DATA7_REG</a>	Register 7 of BLOCK2 data	0x005C	RO

Name	Description	Address	Access
<a href="#">EFUSE_RD_BLK3_DATA0_REG</a>	Register 0 of BLOCK3 data	0x0060	RO
<a href="#">EFUSE_RD_BLK3_DATA1_REG</a>	Register 1 of BLOCK3 data	0x0064	RO
<a href="#">EFUSE_RD_BLK3_DATA2_REG</a>	Register 2 of BLOCK3 data	0x0068	RO
<a href="#">EFUSE_RD_BLK3_DATA3_REG</a>	Register 3 of BLOCK3 data	0x006C	RO
<a href="#">EFUSE_RD_BLK3_DATA4_REG</a>	Register 4 of BLOCK3 data	0x0070	RO
<a href="#">EFUSE_RD_BLK3_DATA5_REG</a>	Register 5 of BLOCK3 data	0x0074	RO
<a href="#">EFUSE_RD_BLK3_DATA6_REG</a>	Register 6 of BLOCK3 data	0x0078	RO
<a href="#">EFUSE_RD_BLK3_DATA7_REG</a>	Register 7 of BLOCK3 data	0x007C	RO
<b>Report Register</b>			
<a href="#">EFUSE_RD_REPEAT_ERR_REG</a>	Register 0 with programming error record of BLOCK0	0x0080	RO
<a href="#">EFUSE_RD_RS_ERR_REG</a>	Register 0 with programming error record of BLOCK1-3	0x0084	RO
<b>Configuration Register</b>			
<a href="#">EFUSE_CLK_REG</a>	eFuse clock configuration register	0x0088	R/W
<a href="#">EFUSE_CONF_REG</a>	eFuse operation mode configuration register	0x008C	R/W
<a href="#">EFUSE_CMD_REG</a>	eFuse command register	0x0094	Varies
<a href="#">EFUSE_DAC_CONF_REG</a>	Controls the eFuse programming voltage	0x0108	R/W
<a href="#">EFUSE_RD_TIM_CONF_REG</a>	Configures read timing parameters	0x010C	R/W
<a href="#">EFUSE_WR_TIM_CONF1_REG</a>	Configuration register 1 of eFuse programming timing parameters	0x0114	R/W
<a href="#">EFUSE_WR_TIM_CONF2_REG</a>	Configuration register 2 of eFuse programming timing parameters	0x0118	R/W
<b>Status Register</b>			
<a href="#">EFUSE_STATUS_REG</a>	eFuse status register	0x0090	RO
<b>Interrupt Register</b>			
<a href="#">EFUSE_INT_RAW_REG</a>	eFuse raw interrupt register	0x0098	R/WTC/SS
<a href="#">EFUSE_INT_ST_REG</a>	eFuse interrupt status register	0x009C	RO
<a href="#">EFUSE_INT_ENA_REG</a>	eFuse interrupt enable register	0x0100	R/W
<a href="#">EFUSE_INT_CLR_REG</a>	eFuse interrupt clear register	0x0104	WT
<b>Version Register</b>			
<a href="#">EFUSE_DATE_REG</a>	eFuse version register	0x01FC	R/W

## 4.5 Registers

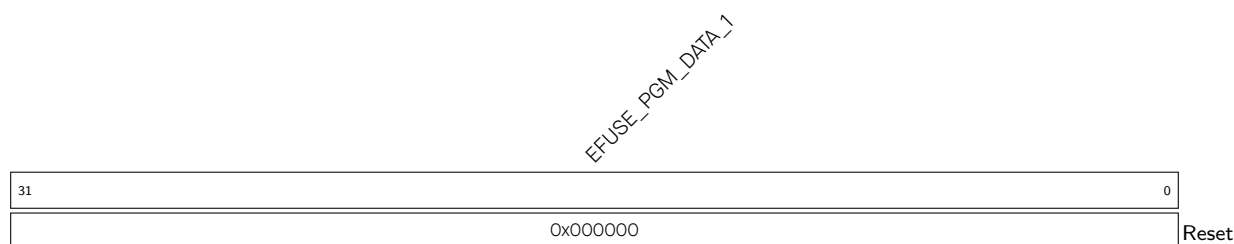
The addresses in this section are relative to eFuse controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 4.1. EFUSE\_PGM\_DATA0\_REG (0x0000)**



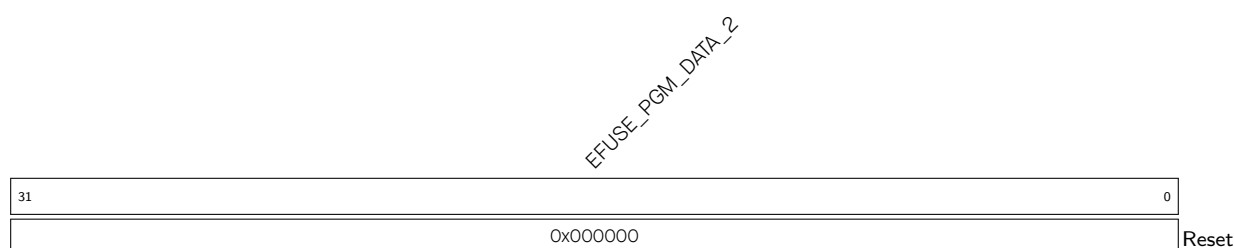
**EFUSE\_PGM\_DATA\_0** Configures the content of the 0th 32-bit data to be programmed. (R/W)

**Register 4.2. EFUSE\_PGM\_DATA1\_REG (0x0004)**

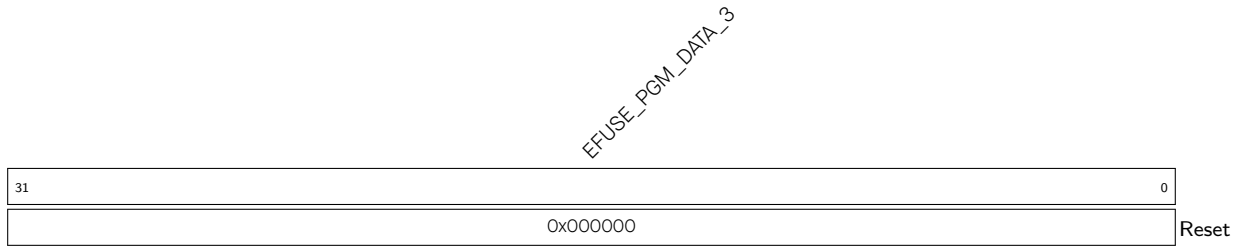


**EFUSE\_PGM\_DATA\_1** Configures the content of the 1st 32-bit data to be programmed. (R/W)

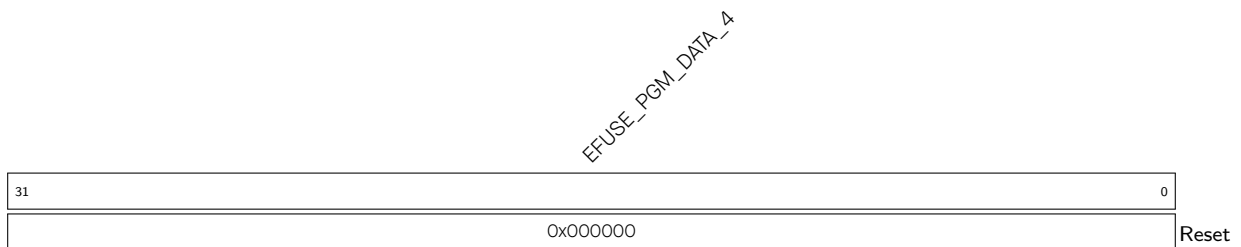
**Register 4.3. EFUSE\_PGM\_DATA2\_REG (0x0008)**



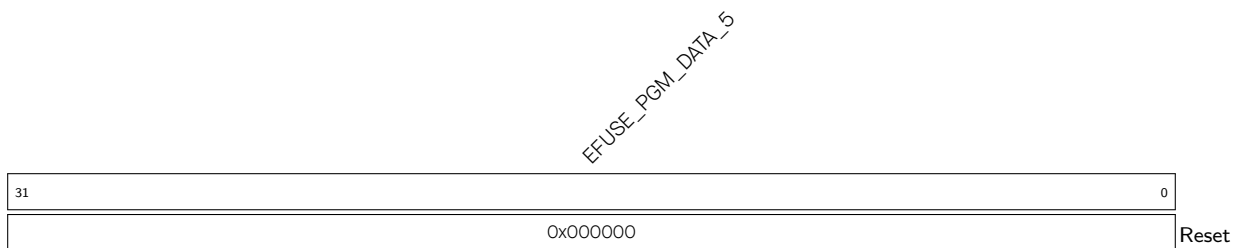
**EFUSE\_PGM\_DATA\_2** Configures the content of the 2nd 32-bit data to be programmed. (R/W)

**Register 4.4. EFUSE\_PGM\_DATA3\_REG (0x000C)**

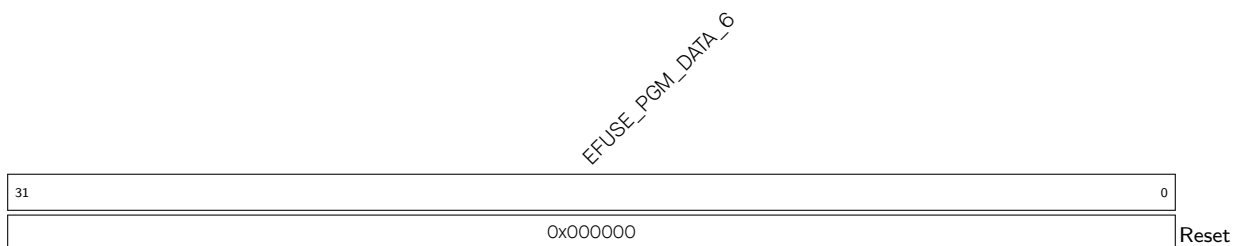
**EFUSE\_PGM\_DATA\_3** Configures the content of the 3rd 32-bit data to be programmed. (R/W)

**Register 4.5. EFUSE\_PGM\_DATA4\_REG (0x0010)**

**EFUSE\_PGM\_DATA\_4** Configures the content of the 4th 32-bit data to be programmed. (R/W)

**Register 4.6. EFUSE\_PGM\_DATA5\_REG (0x0014)**

**EFUSE\_PGM\_DATA\_5** Configures the content of the 5th 32-bit data to be programmed. (R/W)

**Register 4.7. EFUSE\_PGM\_DATA6\_REG (0x0018)**

**EFUSE\_PGM\_DATA\_6** Configures the content of the 6th 32-bit data to be programmed. (R/W)

**Register 4.8. EFUSE\_PGM\_DATA7\_REG (0x001C)**

<i>EFUSE_PGM_DATA_7</i>	
31	0
0x000000	
Reset	

**EFUSE\_PGM\_DATA\_7** Configures the content of the 7th 32-bit data to be programmed. (R/W)

**Register 4.9. EFUSE\_PGM\_CHECK\_VALUE0\_REG (0x0020)**

<i>EFUSE_PGM_RS_DATA_0</i>	
31	0
0x000000	
Reset	

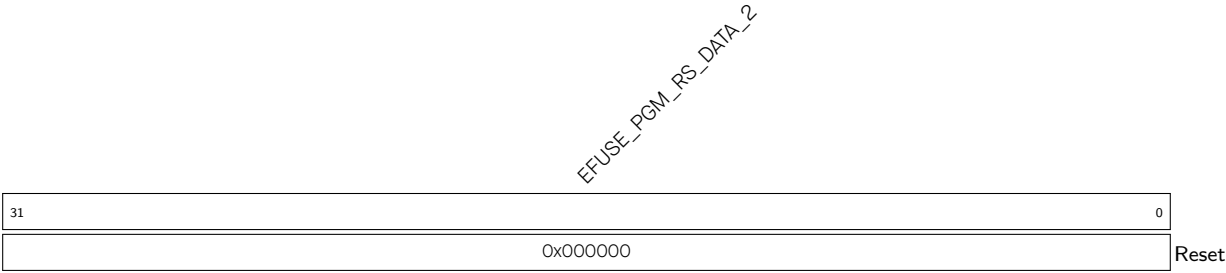
**EFUSE\_PGM\_RS\_DATA\_0** Configures the content of the 0th 32-bit RS code to be programmed.  
(R/W)

**Register 4.10. EFUSE\_PGM\_CHECK\_VALUE1\_REG (0x0024)**

<i>EFUSE_PGM_RS_DATA_1</i>	
31	0
0x000000	
Reset	

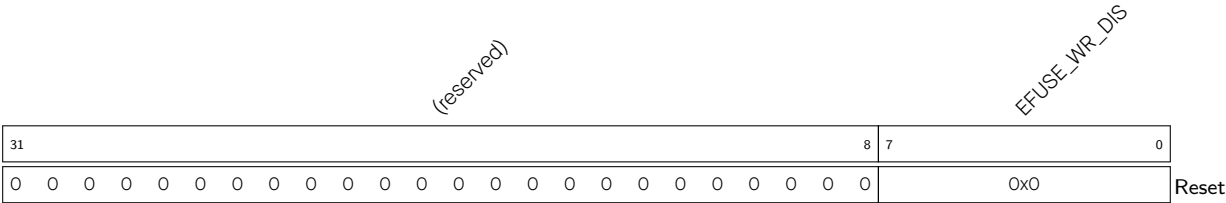
**EFUSE\_PGM\_RS\_DATA\_1** Configures the content of the 1st 32-bit RS code to be programmed.  
(R/W)

Register 4.11. EFUSE\_PGM\_CHECK\_VALUE2\_REG (0x0028)



**EFUSE\_PGM\_RS\_DATA\_2** Configures the content of the 2nd 32-bit RS code to be programmed.  
(R/W)

Register 4.12. EFUSE\_RD\_WR\_DIS\_REG (0x002C)



**EFUSE\_WR\_DIS** Represents whether programming of individual eFuses is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

Register 4.13. EFUSE\_RD\_REPEAT\_DATA0\_REG (0x0030)

EFUSE_RPT4_RESERVED		EFUSE_CUSTOM_MAC_USED		EFUSE_SECURE_VERSION		EFUSE_SECURE_BOOT_EN		EFUSE_FLASH_TPUW		EFUSE_ENABLE_SECURITY_DOWNLOAD		EFUSE_DIS_DIRECT_BOOT		EFUSE_DIS_DOWNLOAD_MODE		EFUSE_FORCE_SEND_RESUME		EFUSE_UART_PRINT_CONTROL		EFUSE_XTS_KEY_LENGTH_256		EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT		EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT		EFUSE_DIS_DOWNLOAD_ICACHE		EFUSE_WDT_DELAY_SEL		EFUSE_RD_DIS	
31	27	26	25	22	21	20	17	16	15	14	13	12	11	10	9	7	6	5	4	3	2	1	0	Reset							
0x0		0	0x0		0	0x0		0	0	0	0	0x0	0	0x0		0	0	0	0	0x0	0	Reset									

**EFUSE\_RD\_DIS** Represents whether reading of the high/low 128 bits is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

**EFUSE\_WDT\_DELAY\_SEL** Represents RTC watchdog timeout threshold. Measurement unit: slow clock cycle. 0: 40000. 1: 80000. 2: 160000. 3: 320000. (RO)

**EFUSE\_DIS\_PAD\_JTAG** Represents whether pad JTAG is permanently disabled or enabled. 1: Disabled. 0: Enabled. (RO)

**EFUSE\_DIS\_DOWNLOAD\_ICACHE** Represents whether iCache is disabled or enabled in download mode. 1: Disabled. 0: Enabled. (RO)

**EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT** Represents whether manual flash encryption is disabled or enabled in download boot modes. 1: Disabled. 0: Enabled. (RO)

**EFUSE\_SPI\_BOOT\_ENCRYPT\_DECRYPT\_CNT** Represents whether SPI boot encryption and decryption are disabled or enabled. Odd number of 1: Enabled. Even number of 1: Disabled. (RO)

**EFUSE\_XTS\_KEY\_LENGTH\_256** Represents key length for XTS\_AES. 1: All 256 bits of BLOCK3. 0: The lower 128 bits of BLOCK3. (RO)

**EFUSE\_UART\_PRINT\_CONTROL** Represents UART boot message output mode. 2'b00: Force print; 2'b01: Low-level print, controlled by GPIO8; 2'b10: High-level print, controlled by GPIO8; 2'b11: Print force disable. (RO)

**EFUSE\_FORCE\_SEND\_RESUME** Represents whether to force ROM code to send an SPI flash resume command during SPI boot. 1: Send. 0: Not send. (RO)

**EFUSE\_DIS\_DOWNLOAD\_MODE** Represents whether all Download modes are disabled or enabled (boot\_mode[3:0] = 0, 1, 2, 4, 5, 6, 7). 1: Disabled. 0: Enabled. (RO)

**EFUSE\_DIS\_DIRECT\_BOOT** Represents whether Direct\_boot mode is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

**EFUSE\_ENABLE\_SECURITY\_DOWNLOAD** Represents whether UART secure download mode is enabled or disabled (read/write flash only). 1: Enabled. 0: Disabled. (RO)

Continued on the next page...

**Register 4.13. EFUSE\_RD\_REPEAT\_DATA0\_REG (0x0030)**

Continued from the previous page...

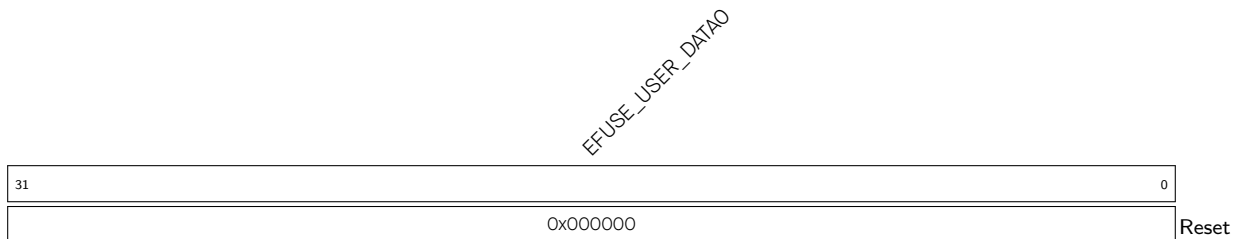
**EFUSE\_FLASH\_TPUW** Represents flash startup delay after SoC is powered up. Measurement unit: ms. If the value is less than 15, it represents the delay. If the value is equal to or larger than 15, the delay is 30 ms. (RO)

**EFUSE\_SECURE\_BOOT\_EN** Represents whether secure boot is enabled or disabled. 1: Enabled. 0: Disabled. (RO)

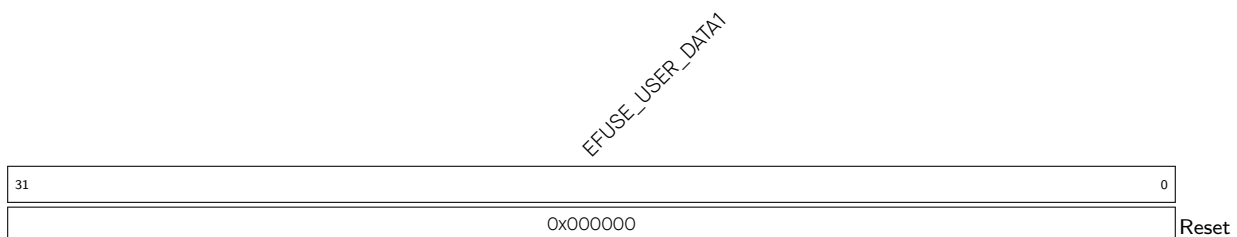
**EFUSE\_SECURE\_VERSION** Represents the secure version used by ESP-IDF anti-rollback feature. (RO)

**EFUSE\_CUSTOM\_MAC\_USED** Represents whether the MAC customized by users is used or not. 1: Used. 0: Not used. (RO)

**EFUSE\_RPT4\_RESERVED** Reserved (used for four backups method). (RO)

**Register 4.14. EFUSE\_RD\_BLK1\_DATA0\_REG (0x0034)**

**EFUSE\_USER\_DATA0** Stores the 0th 32-bit of user data. (RO)

**Register 4.15. EFUSE\_RD\_BLK1\_DATA1\_REG (0x0038)**

**EFUSE\_USER\_DATA1** Stores the 1st 32-bit of user data. (RO)



Register 4.16. EFUSE\_RD\_BLK1\_DATA2\_REG (0x003C)

(reserved)								EFUSE_USER_DATA2																																										
31								24																					23																					0
0	0	0	0	0	0	0	0	0x0000																							Reset																			

**EFUSE\_USER\_DATA2** Stores the bits [64:87] of user data. (RO)

Register 4.17. EFUSE\_RD\_BLK2\_DATA0\_REG (0x0040)

EFUSE_MAC_ID_LOW																															
31																															0
0x000000																															
Reset																															

**EFUSE\_MAC\_ID\_LOW** Stores the lower 32-bit of MAC ID. (RO)

Register 4.18. EFUSE\_RD\_BLK2\_DATA1\_REG (0x0044)

EFUSE_SYS_DATA0																EFUSE_MAC_ID_HIGH														
31																16	15													0
0x00																0x00														Reset

**EFUSE\_MAC\_ID\_HIGH** Stores the higher 16-bit of MAC ID. (RO)

**EFUSE\_SYS\_DATA0** Stores the 0th 16-bit of system data. (RO)

Register 4.19. EFUSE\_RD\_BLK2\_DATA2\_REG (0x0048)

EFUSE_SYS_DATA1																															
31																															0
0x000000																															
Reset																															

**EFUSE\_SYS\_DATA1** Stores the 0th 32-bit of system data. (RO)

**Register 4.20. EFUSE\_RD\_BLK2\_DATA3\_REG (0x004C)**

<i>EFUSE_SYS_DATA2</i>	
31	0
0x000000	
Reset	

**EFUSE\_SYS\_DATA2** Stores the 1st 32-bit of system data. (RO)

**Register 4.21. EFUSE\_RD\_BLK2\_DATA4\_REG (0x0050)**

<i>EFUSE_SYS_DATA3</i>	
31	0
0x000000	
Reset	

**EFUSE\_SYS\_DATA3** Stores the 2nd 32-bit of system data. (RO)

**Register 4.22. EFUSE\_RD\_BLK2\_DATA5\_REG (0x0054)**

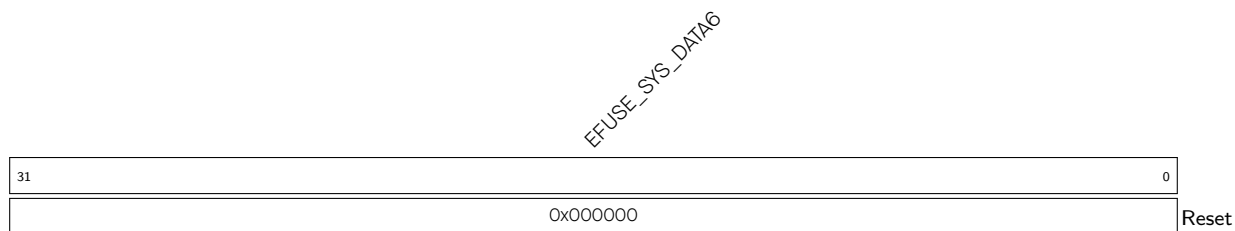
<i>EFUSE_SYS_DATA4</i>	
31	0
0x000000	
Reset	

**EFUSE\_SYS\_DATA4** Stores the 3rd 32-bit of system data. (RO)

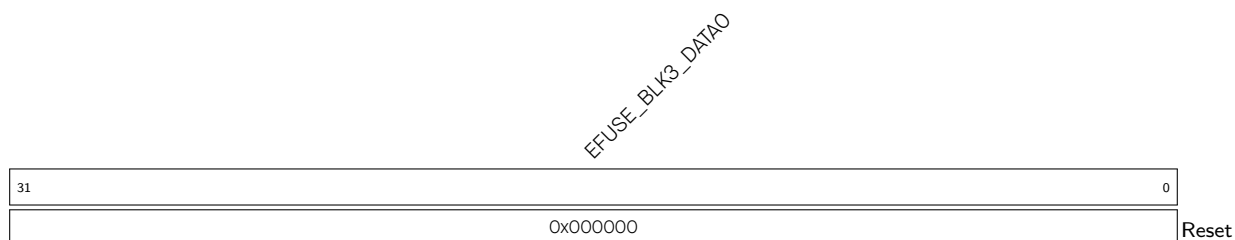
**Register 4.23. EFUSE\_RD\_BLK2\_DATA6\_REG (0x0058)**

<i>EFUSE_SYS_DATA5</i>	
31	0
0x000000	
Reset	

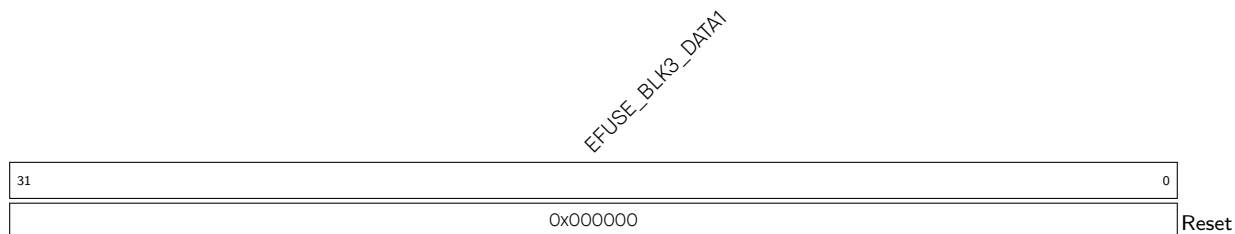
**EFUSE\_SYS\_DATA5** Stores the 4th 32-bit of system data. (RO)

**Register 4.24. EFUSE\_RD\_BLK2\_DATA7\_REG (0x005C)**

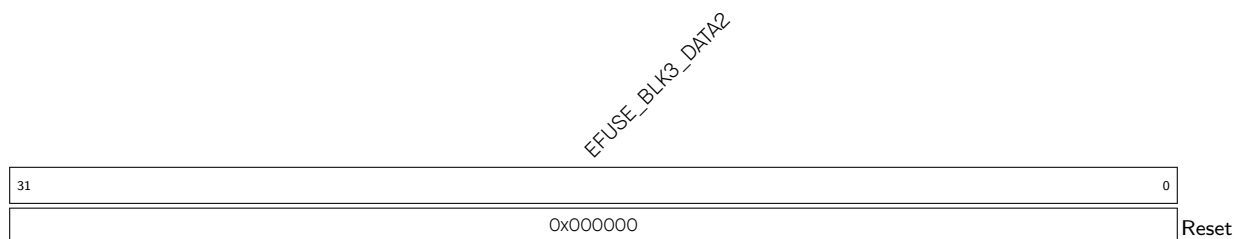
**EFUSE\_SYS\_DATA6** Stores the 5th 32-bit of system data. (RO)

**Register 4.25. EFUSE\_RD\_BLK3\_DATA0\_REG (0x0060)**

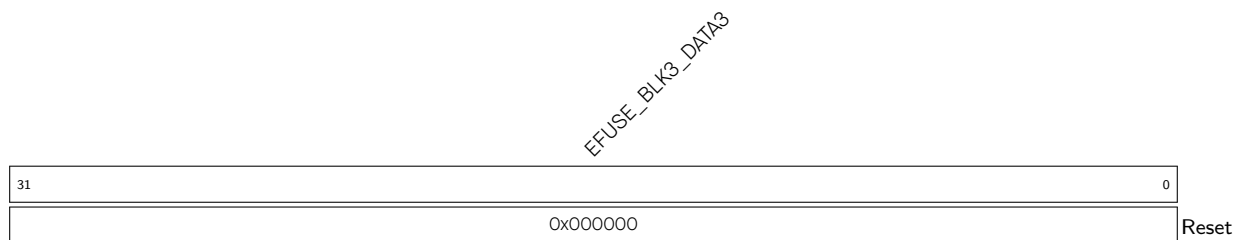
**EFUSE\_BLK3\_DATA0** Stores the 0th 32-bit of BLOCK3. (RO)

**Register 4.26. EFUSE\_RD\_BLK3\_DATA1\_REG (0x0064)**

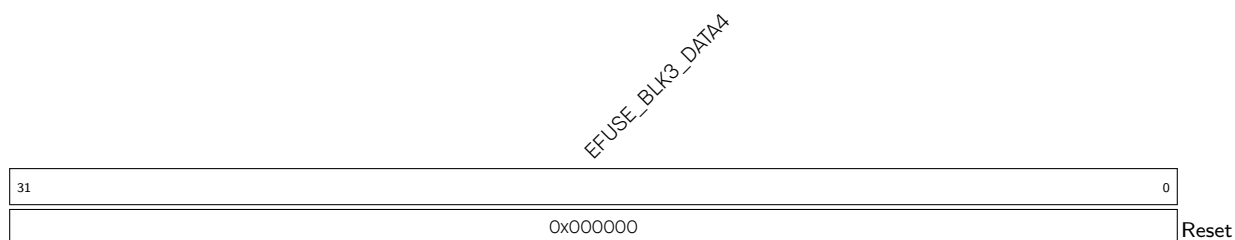
**EFUSE\_BLK3\_DATA1** Stores the 1st 32-bit of BLOCK3. (RO)

**Register 4.27. EFUSE\_RD\_BLK3\_DATA2\_REG (0x0068)**

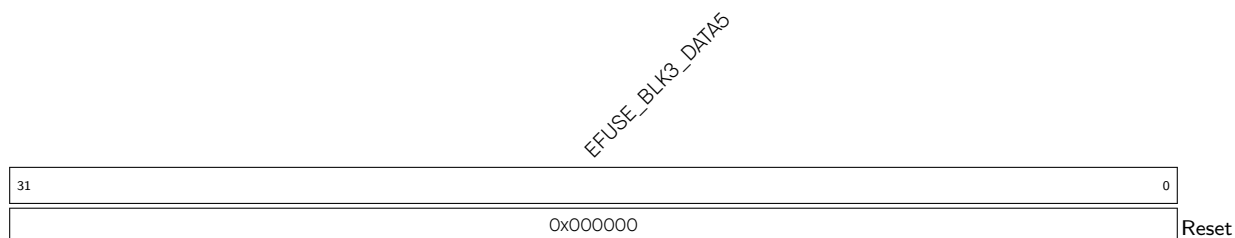
**EFUSE\_BLK3\_DATA2** Stores the 2nd 32-bit of BLOCK3. (RO)

**Register 4.28. EFUSE\_RD\_BLK3\_DATA3\_REG (0x006C)**

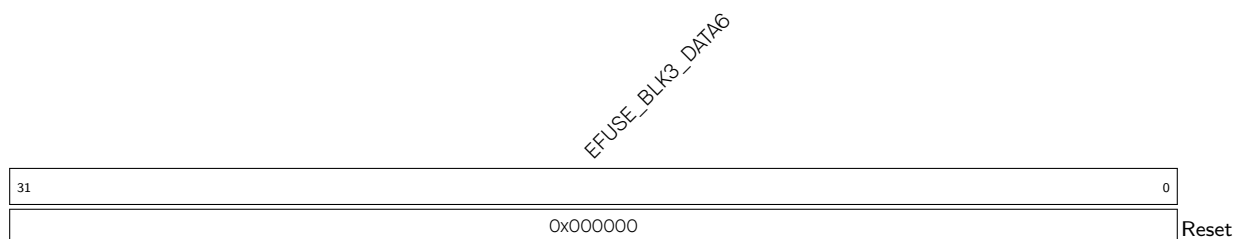
**EFUSE\_BLK3\_DATA3** Stores the 3rd 32-bit of BLOCK3. (RO)

**Register 4.29. EFUSE\_RD\_BLK3\_DATA4\_REG (0x0070)**

**EFUSE\_BLK3\_DATA4** Stores the 4th 32-bit of BLOCK3. (RO)

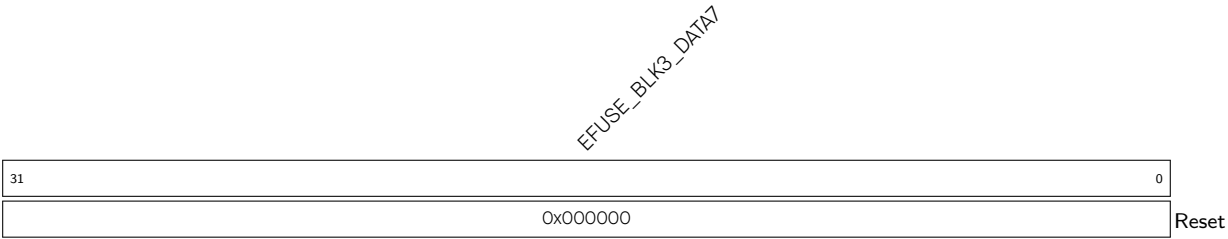
**Register 4.30. EFUSE\_RD\_BLK3\_DATA5\_REG (0x0074)**

**EFUSE\_BLK3\_DATA5** Stores the 5th 32-bit of BLOCK3. (RO)

**Register 4.31. EFUSE\_RD\_BLK3\_DATA6\_REG (0x0078)**

**EFUSE\_BLK3\_DATA6** Stores the 6th 32-bit of BLOCK3. (RO)

Register 4.32. EFUSE\_RD\_BLK3\_DATA7\_REG (0x007C)



**EFUSE\_BLK3\_DATA7** Stores the 7th 32-bit of BLOCK3. (RO)

### Register 4.33. EFUSE\_RD\_REPEAT\_ERR\_REG (0x0080)

[illegible]

**EFUSE\_RD\_DIS\_ERR** If any bit in RD\_DIS\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_WDT\_DELAY\_SEL\_ERR** If any bit in WDT\_DELAY\_SEL\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_DIS\_PAD\_JTAG\_ERR** If any bit in DIS\_PAD\_JTAG\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_DIS\_DOWNLOAD\_ICACHE\_ERR** If any bit in DIS\_DOWN\_ICACHE\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT\_ERR** If any bit in DIS\_DOWNLOAD\_MANUAL\_ENCRYPT\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_SPI\_BOOT\_ENCRYPT\_DECRYPT\_CNT\_ERR** If any bit in SPI\_BOOT\_ENCRYPT\_DECRYPT\_CNT\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_XTS\_KEY\_LENGTH\_256\_ERR** If any bit in XTS\_KEY\_LENGTH\_256\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_UART\_PRINT\_CONTROL\_ERR** If any bit in UART\_PRINT\_CONTROL\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_FORCE\_SEND\_RESUME\_ERR** If any bit in FORCE\_SEND\_RESUME\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_DIS\_DOWNLOAD\_MODE\_ERR** If any bit in DIS\_DOWNLOAD\_MODE\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_DIS\_DIRECT\_BOOT\_ERR** If any bit in DIS\_DIRECT\_BOOT\_ERR is 1, then it indicates a programming error of this parameter. (RO)

Continued on the next page...

**Register 4.33. EFUSE\_RD\_REPEAT\_ERR\_REG (0x0080)**

Continued from the previous page...

**EFUSE\_ENABLE\_SECURITY\_DOWNLOAD\_ERR** If any bit in ENABLE\_SECURITY\_DOWNLOAD\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_FLASH\_TPUW\_ERR** If any bit in FLASH\_TPUW\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_SECURE\_BOOT\_EN\_ERR** If any bit in SECURE\_BOOT\_EN\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_SECURE\_VERSION\_ERR** If any bit in SECURE\_VERSION\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_CUSTOM\_MAC\_USED\_ERR** If any bit in CUSTOM\_MAC\_USED\_ERR is 1, then it indicates a programming error of this parameter. (RO)

**EFUSE\_RPT4\_RESERVED\_ERR** Reserved. (RO)

**Register 4.34. EFUSE\_RD\_RS\_ERR\_REG (0x0084)**

(reserved)																				EFUSE_BLK3_FAIL				EFUSE_BLK3_ERR_NUM				EFUSE_BLK2_FAIL				EFUSE_BLK2_ERR_NUM				EFUSE_BLK1_FAIL				EFUSE_BLK1_ERR_NUM																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31																				12				11	10	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_BLK1\_ERR\_NUM** The value of this signal means the number of error bytes in BLOCK1. (RO)

**EFUSE\_BLK1\_FAIL** 0: Means no failure and that the data of BLOCK1 is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

**EFUSE\_BLK2\_ERR\_NUM** The value of this signal means the number of error bytes in BLOCK2. (RO)

**EFUSE\_BLK2\_FAIL** 0: Means no failure and that the data of BLOCK2 is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

**EFUSE\_BLK3\_ERR\_NUM** The value of this signal means the number of error bytes in BLOCK3. (RO)

**EFUSE\_BLK3\_FAIL** 0: Means no failure and that the BLOCK3 data is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

Register 4.35. EFUSE\_CLK\_REG (0x0088)

(reserved)																EFUSE_CLK_EN																(reserved)																EFUSE_EFUSE_MEM_FORCE_PU EFUSE_MEM_CLK_FORCE_ON EFUSE_EFUSE_MEM_FORCE_PD			
31																17	16	15																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	0	0	0	1	0	Reset											

Reset

**EFUSE\_EFUSE\_MEM\_FORCE\_PD** Set this bit to force SRAM in eFuse controller into power-saving mode. (R/W)

**EFUSE\_MEM\_CLK\_FORCE\_ON** Set this bit to force to activate clock signal of SRAM in eFuse controller. (R/W)

**EFUSE\_EFUSE\_MEM\_FORCE\_PU** Set this bit to force SRAM in eFuse controller into working mode. (R/W)

**EFUSE\_CLK\_EN** Set this bit to force enable clock signal of eFuse configuration register. (R/W)

Register 4.36. EFUSE\_CONF\_REG (0x008C)

(reserved)																EFUSE_OP_CODE															
31															16	15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00														Reset	

Reset

**EFUSE\_OP\_CODE** 0x5A5A: Operate programming command. 0x5AA5: Operate read command. (R/W)



Register 4.37. EFUSE\_CMD\_REG (0x0094)

(reserved)																												EFUSE_BLK_NUM			EFUSE_PGM_CMD		EFUSE_READ_CMD	
31																												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	0x0	0	0	Reset

**EFUSE\_READ\_CMD** Set this bit to send read command. (R/W/SC)

**EFUSE\_PGM\_CMD** Set this bit to send programming command. (R/W/SC)

**EFUSE\_BLK\_NUM** The serial number of the block to be programmed. Value 0-3 corresponds to block number 0-3 respectively. (R/W)

Register 4.38. EFUSE\_DAC\_CONF\_REG (0x0108)

(reserved)																EFUSE_OE_CLR		EFUSE_DAC_NUM		EFUSE_DAC_CLK_PAD_SEL		EFUSE_DAC_CLK_DIV				
31																18	17	16			9	8	7			0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	0	28	Reset						

**EFUSE\_DAC\_CLK\_DIV** Controls the division factor of the rising clock of the programming voltage. (R/W)

**EFUSE\_DAC\_CLK\_PAD\_SEL** Don't care. (R/W)

**EFUSE\_DAC\_NUM** Controls the rising period of the programming voltage. (R/W)

**EFUSE\_OE\_CLR** Reduces the power supply of the programming voltage. (R/W)

Register 4.39. EFUSE\_RD\_TIM\_CONF\_REG (0x010C)

EFUSE_READ_INIT_NUM																								(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
31												24												23																								0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0x12												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0											

**EFUSE\_READ\_INIT\_NUM** Configures the waiting time of reading eFuse memory. (R/W)

**Register 4.40. EFUSE\_WR\_TIM\_CONF1\_REG (0x0114)**

(reserved)								EFUSE_PWR_ON_NUM															(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
31								24								23								8								7								0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
0								0								0								0								0								0								0								0x3000								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0								0							

**EFUSE\_PWR\_ON\_NUM** Configures the power up time for VDDQ. (R/W)

**Register 4.41. EFUSE\_WR\_TIM\_CONF2\_REG (0x0118)**

(reserved)																EFUSE_PWR_OFF_NUM																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x190																Reset															

**EFUSE\_PWR\_OFF\_NUM** Configures the power outage time for VDDQ. (R/W)

**Register 4.42. EFUSE\_STATUS\_REG (0x0090)**

(reserved)																EFUSE_BLK0_VALID_BIT_CNT				(reserved)				EFUSE_STATE			
31																16	15	10	9	4	3	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0x0	Reset		

**EFUSE\_STATE** Indicates the state of the eFuse controller state machine. (RO)

**EFUSE\_BLK0\_VALID\_BIT\_CNT** Records the number of bits with a value of '1' in BLOCK0. (RO)

**Register 4.43. EFUSE\_INT\_RAW\_REG (0x0098)**

(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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**EFUSE\_READ\_DONE\_INT\_RAW** The raw bit signal for read\_done interrupt. (R/WTC/SS)

**EFUSE\_PGM\_DONE\_INT\_RAW** The raw bit signal for pgm\_done interrupt. (R/WTC/SS)

**Register 4.44. EFUSE\_INT\_ST\_REG (0x009C)**

(reserved)																															EFUSE_PGM_DONE_INT_ST		EFUSE_READ_DONE_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
31																														2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**EFUSE\_READ\_DONE\_INT\_ST** The status signal for read\_done interrupt. (RO)

**EFUSE\_PGM\_DONE\_INT\_ST** The status signal for pgm\_done interrupt. (RO)

**Register 4.45. EFUSE\_INT\_ENA\_REG (0x0100)**

(reserved)																															EFUSE_PGM_DONE_INT_ENA		EFUSE_READ_DONE_INT_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**EFUSE\_READ\_DONE\_INT\_ENA** The enable signal for read\_done interrupt. (R/W)

**EFUSE\_PGM\_DONE\_INT\_ENA** The enable signal for pgm\_done interrupt. (R/W)

Register 4.46. EFUSE\_INT\_CLR\_REG (0x0104)

(reserved)																															EFUSE_PGM_DONE_INT_CLR EFUSE_READ_DONE_INT_CLR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**EFUSE\_READ\_DONE\_INT\_CLR** The clear signal for read\_done interrupt. (WT)

**EFUSE\_PGM\_DONE\_INT\_CLR** The clear signal for pgm\_done interrupt. (WT)

Register 4.47. EFUSE\_DATE\_REG (0x01FC)

(reserved)				EFUSE_DATE																							31	28	27	0
0	0	0	0	0x2108190																										Reset

**EFUSE\_DATE** Stores eFuse controller register version. (R/W)

## Part III

# System Component

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

## Chapter 5

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

## 5.1 Overview

The ESP8684 chip features 21 GPIO pins. Each pin can be used as a general-purpose I/O, or to be connected to an internal peripheral signal. Through GPIO matrix and IO MUX, peripheral input signals can be from any IO pins, and peripheral output signals can be routed to any IO pins. Together these modules provide highly configurable I/O.

**Note:**

- The 21 GPIO pins are numbered 0 ~ 20.
- For chip variants with a SiP flash built in, GPIO11~ GPIO17 are dedicated to connecting SiP flash, not for other uses. The remaining 14 GPIO pins (numbered 0 ~ 10, 18 ~ 20) are configurable by users.

## 5.2 Features

**GPIO matrix has the following features:**

- A full-switching matrix between the peripheral input/output signals and the GPIO pins.
- 33 peripheral input signals can be sourced from the input of any GPIO pins.
- The output of any GPIO pins can be from any of the 61 peripheral output signals.
- Supports signal synchronization for peripheral inputs based on APB clock bus.
- Provides input signal filter.
- Supports GPIO simple input and output.

**IO MUX has the following features:**

- Provides one configuration register `IO_MUX_GPIO $n$ _REG` for each GPIO pin. The pin can be configured to
  - perform GPIO function routed by GPIO matrix;
  - or perform direct connection bypassing GPIO matrix.
- Supports some high-speed digital signals (SPI, JTAG, UART) bypassing GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to peripherals.

## 5.3 Architectural Overview

Figure 5.3-1 shows in details how IO MUX and GPIO matrix route signals from pins to peripherals, and from peripherals to pins.

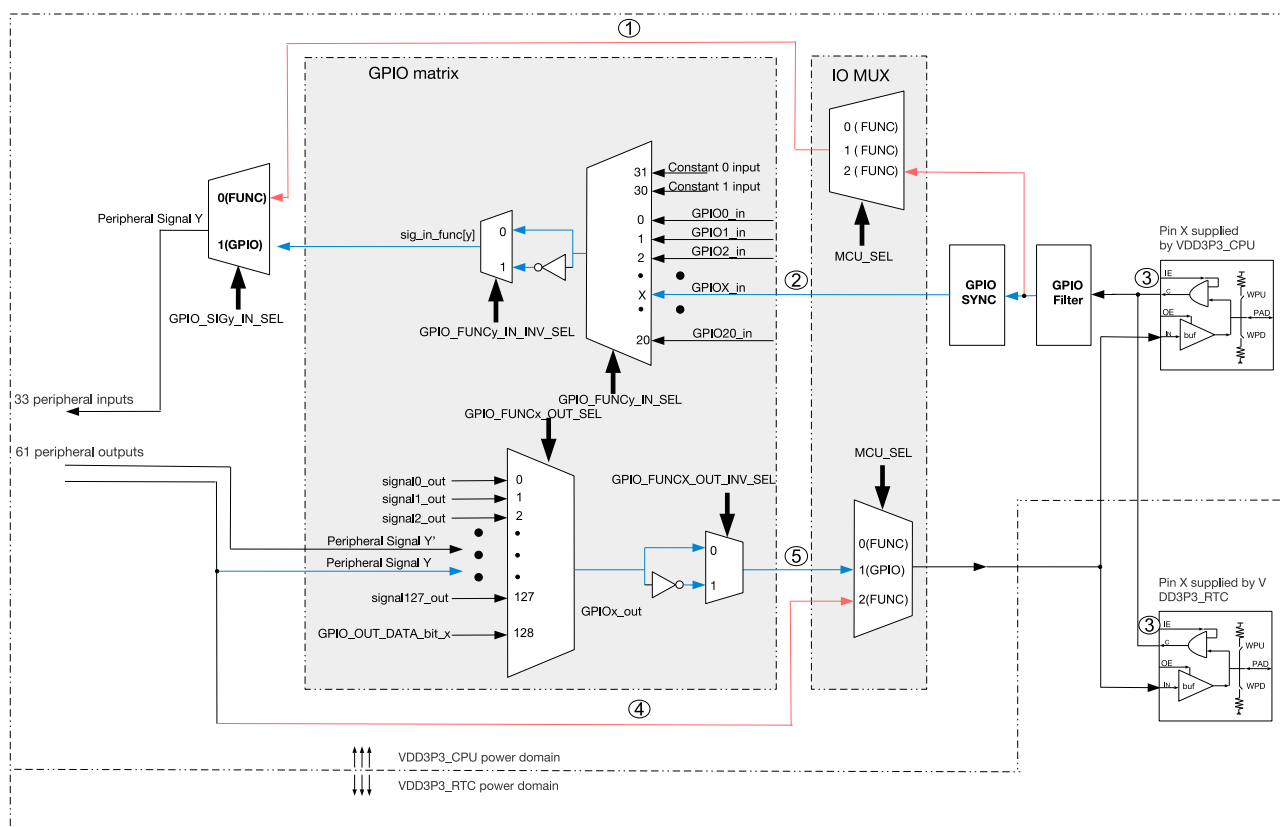


Figure 5.3-1. Architecture of IO MUX and GPIO Matrix

1. Only part of peripheral input signals (marked “yes” in column “Direct input through IO MUX” in Table 5.11-1) can bypass GPIO matrix. The other input signals can only be routed to peripherals via GPIO matrix.
2. There are only 21 inputs from GPIO SYNC to GPIO matrix, since ESP8684 provides 21 GPIO pins in total. Note, for chip variants with SiP flash, there are only 14 inputs from GPIO SYNC to GPIO matrix in total.
3. The pins supplied by VDD3P3\_CPU or by VDD3P3\_RTC are controlled by the signals: IE, OE, WPU, and WPD.
4. Only part of peripheral outputs (marked “yes” in column “Direct output through IO MUX” in Table 5.11-1) can be routed to pins bypassing GPIO matrix. The other output signals can only be routed to pins via GPIO matrix.
5. There are 21 outputs (corresponding to GPIO X: 0 ~ 20) from GPIO matrix to IO MUX. Note, for chip variants with SiP flash, there are only 14 outputs (corresponding to GPIO X: 0 ~ 10, 18 ~ 20) from GPIO matrix to IO MUX in total.

Figure 5.3-2 shows the internal structure of a pad, which is an electrical interface between the chip logic and the GPIO pin. The structure is applicable to all 21 GPIO pins and can be controlled by IE, OE, WPU, and WPD signals.

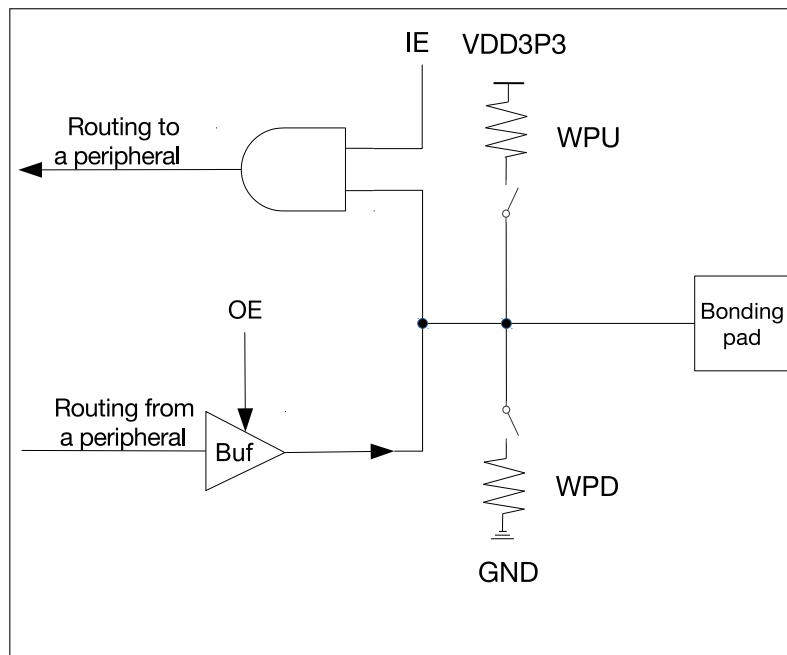


Figure 5.3-2. Internal Structure of a Pad

**Note:**

- 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.

## 5.4 Peripheral Input via GPIO Matrix

### 5.4.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 21 GPIOs (0 ~ 20), see Table 5.11-1. Meanwhile, register corresponding to the peripheral signal should be set to receive input signal via GPIO matrix.

### 5.4.2 Signal Synchronization

When signals are directed from pins using GPIO matrix, the signals will be synchronized to the APB bus clock by GPIO SYNC hardware, then go to GPIO matrix. This synchronization applies to all GPIO matrix signals but does not apply when using IO MUX, see Figure 5.3-1.



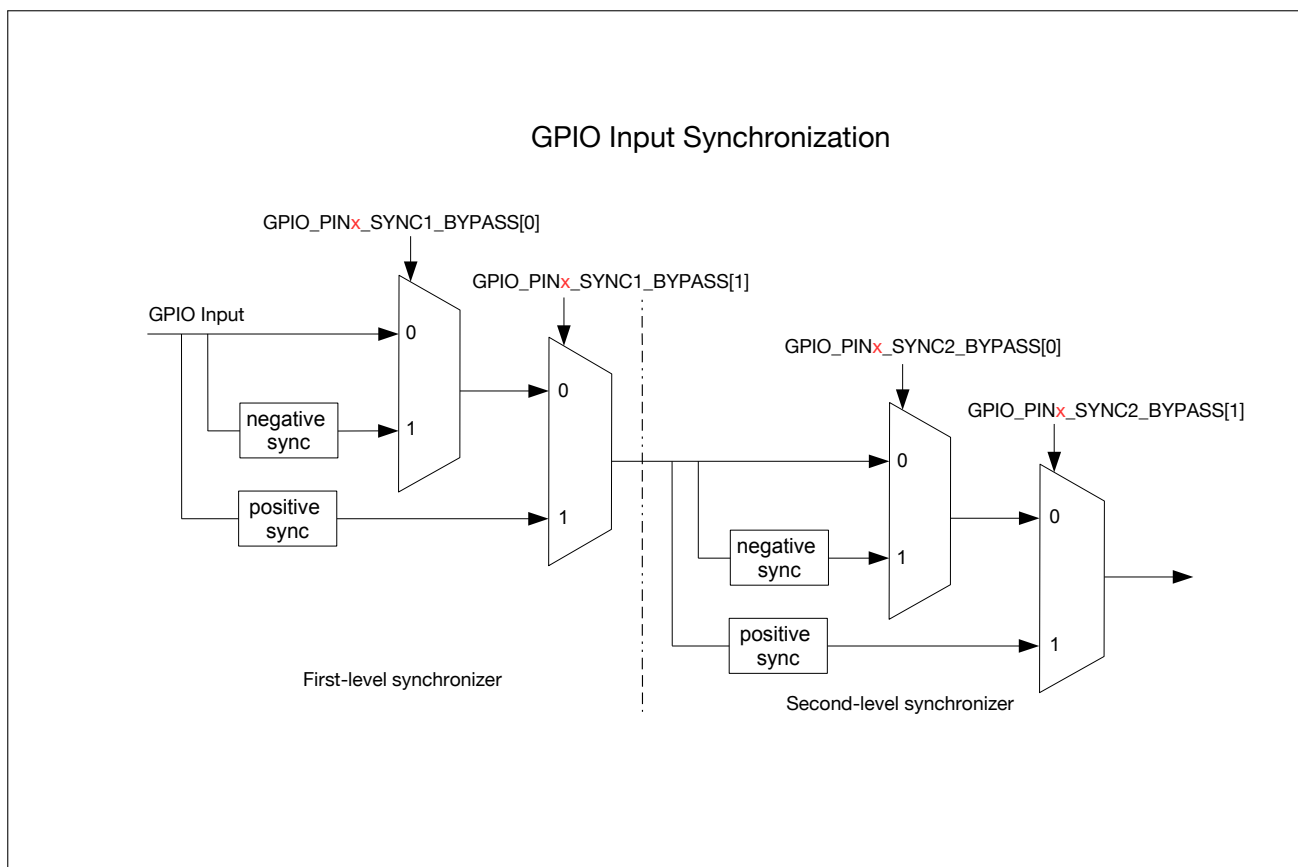


Figure 5.4-1. GPIO Input Synchronized on APB Clock Rising Edge or on Falling Edge

Figure 5.4-1 shows the functionality of GPIO SYNC. In the figure, negative sync and positive sync mean GPIO input is synchronized on APB clock falling edge and on APB clock rising edge, respectively.

The synchronization function is disabled by default, i.e., `GPIO_PINx_SYNC1/2_BYPASS[1:0] = 0`. But when an asynchronous peripheral signal is connected to the pin, this signal should be synchronized by two-level synchronization (i.e., first-level synchronizer and second-level synchronizer) to reduce the probability of causing metastability. For more information, see Step 3 in the following section.

### 5.4.3 Functional Description

To read GPIO pin  $X^1$  into peripheral signal  $Y$ , follow the steps below:

1. Configure register `GPIO_FUNCy_IN_SEL_CFG_REG` corresponding to peripheral signal  $Y$  in GPIO matrix:
  - Set `GPIO_SIGy_IN_SEL` to enable peripheral signal input via GPIO matrix.
  - Set `GPIO_FUNCy_IN_SEL` to the desired GPIO pin, i.e.  $X$  here.

**Note that** some peripheral signals have no valid `GPIO_SIGy_IN_SEL` bit, namely, these peripherals can only receive input signals via GPIO matrix.

2. Optionally enable the filter for pin input signals by setting `IO_MUX_GPIOn_FILTER_EN`. Only the signals with a valid width of more than two clock cycles can be sampled, see Figure 5.4-2.

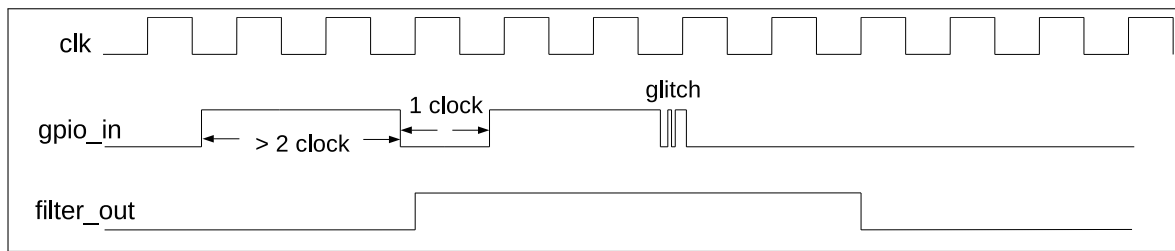


Figure 5.4-2. Filter Timing of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set `GPIO_PINx_REG` corresponding to GPIO pin `X` as follows:

- Set `GPIO_PINx_SYNC1_BYPASS` to enable input signal synchronized on rising edge or on falling edge in the first-level synchronizer, see Figure 5.4-1.
- Set `GPIO_PINx_SYNC2_BYPASS` to enable input signal synchronized on rising edge or on falling edge in the second-level synchronizer, see Figure 5.4-1.

4. Configure IO MUX register to enable pin input. For this end, please set `IO_MUX_GPIOx_REG` corresponding to GPIO pin `X` as follows:

- Set `IO_MUX_GPIOx_FUN_IE` to enable input<sup>2</sup>.
- Set or clear `IO_MUX_GPIOx_FUN_WPU` and `IO_MUX_GPIOx_FUN_WPD`, as desired, to enable or disable pull-up and pull-down resistors.

For example, to connect UART0 DSR input signal<sup>3</sup> (UODSR\_in, signal index 8) to GPIO7, please follow the steps below. Note that GPIO7 is also named as MTDO pin.

1. Set `GPIO_SIG8_IN_SEL` bit in register `GPIO_FUNC8_IN_SEL_CFG_REG` to enable peripheral signal input via GPIO matrix.
2. Set `GPIO_FUNC8_IN_SEL` in register `GPIO_FUNC8_IN_SEL_CFG_REG` to 7, i.e. select GPIO7.
3. Set `IO_MUX_GPIO7_FUN_IE` in register `IO_MUX_GPIO7_REG` to enable pin input.

**Note:**

1. One input pin can be connected to multiple peripheral input signals.
2. The input signal can be inverted by configuring `GPIO_FUNCy_IN_INV_SEL`.
3. It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special `GPIO_FUNCy_IN_SEL` input, instead of a GPIO number:
  - When `GPIO_FUNCy_IN_SEL` is set to 0x1F, input signal is always 0.
  - When `GPIO_FUNCy_IN_SEL` is set to 0x1E, input signal is always 1.

### 5.4.4 Simple GPIO Input

GPIO matrix can also be used for simple GPIO input. For this case, the input value of one GPIO pin can be read at any time without routing the GPIO input to any peripherals. `GPIO_IN_REG` holds the input values of each GPIO pin.

To implement simple GPIO input, follow the steps below:

- Set `IO_MUX_GPIOx_FUN_IE` in register `IO_MUX_GPIOx_REG`, to enable pin input.
- Read the GPIO input from `GPIO_IN_REG[x]`.

## 5.5 Peripheral Output via GPIO Matrix

### 5.5.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (only signals with a name assigned in the column “Output signal” in Table 5.11-1) to one of the 21 GPIOs (0 ~ 20). Note, for chip variants with SiP flash, output signals can only be mapped to 14 GPIO pins, i.e. GPIO0 ~ GPIO10, GPIO18 ~ GPIO20.

The output signal is routed from the peripheral into GPIO matrix and then into IO MUX. IO MUX must be configured to set the chosen pin to GPIO function. This enables the output GPIO signal to be connected to the pin.

**Note:**

There is a range of peripheral output signals (97 ~ 100 in Table 5.11-1) which are not connected to any peripheral, but to the input signals (97 ~ 100) directly. This feature can be used to input a signal from one GPIO pin and output directly to another GPIO pin.

### 5.5.2 Functional Description

The 61 output signals (signals with a name assigned in the column “Output signal” in Table 5.11-1) can be set to go through GPIO matrix into IO MUX and then to a pin. Figure 5.3-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pin *X*<sup>1</sup>, follow these steps:

1. Configure register `GPIO_FUNCx_OUT_SEL_CFG_REG` and `GPIO_ENABLE_REG[x]` corresponding to GPIO pin *X* in GPIO matrix. Recommended operation: use corresponding `WITS` (write 1 to set) and `WITC` (write 1 to clear) registers to set or clear `GPIO_ENABLE_REG`.
  - Set the `GPIO_FUNCx_OUT_SEL` field in register `GPIO_FUNCx_OUT_SEL_CFG_REG` to the index of the desired peripheral output signal *Y*.
  - If the signal should always be enabled as an output, set the `GPIO_FUNCx_OEN_SEL` bit in register `GPIO_FUNCx_OUT_SEL_CFG_REG` and the bit in register `GPIO_ENABLE_WITS_REG`, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the `SPIQ_oe` in column “Output enable signal when `GPIO_FUNCn_OEN_SEL = 0`” in Table 5.11-1), clear `GPIO_FUNCx_OEN_SEL` bit instead.
  - Set the corresponding bit in register `GPIO_ENABLE_WITC_REG` to disable the output from the GPIO pin.
2. For an open drain output, set the `GPIO_PINx_PAD_DRIVER` bit in register `GPIO_PINx_REG` corresponding to GPIO pin *X*.
3. Configure IO MUX register to enable output via GPIO matrix. Set the `IO_MUX_GPIOx_REG` corresponding to GPIO pin *X* as follows:

- Set the field `IO_MUX_GPIOx_MCU_SEL` to desired IO MUX function corresponding to GPIO pin`x`. This is Function 1 (GPIO function), numeric value 1, for all pins.
- Set the `IO_MUX_GPIOx_FUN_DRV` field to the desired value for output strength (0 ~ 3). The higher the driver strength, the more current can be sourced/sunk from the pin.
  - 0: ~5 mA
  - 1: ~10 mA
  - 2: ~20 mA (default)
  - 3: ~40 mA
- If using open drain mode, set/clear the `IO_MUX_GPIOx_FUN_WPU` and `IO_MUX_GPIOx_FUN_WPD` bits to enable/disable the internal pull-up/pull-down resistors.

**Note:**

1. The output signal from a single peripheral can be sent to multiple pins simultaneously.
2. The output signal can be inverted by setting `GPIO_FUNCx_OUT_INV_SEL` bit.

### 5.5.3 Simple GPIO Output

GPIO matrix can also be used for simple GPIO output. For this case, one GPIO pin can be configured to directly output desired value, without routing any peripheral output signal to this pin. This can be done as below:

- Set GPIO matrix `GPIO_FUNCn_OUT_SEL` with a special peripheral index 128 (0x80);
- Set the corresponding bit in `GPIO_OUT_REG` register to the desired GPIO output value.

**Note:**

- `GPIO_OUT_REG[0] ~ GPIO_OUT_REG[20]` correspond to GPIO0 ~ GPIO20, respectively. `GPIO_OUT_REG[24:21]` are invalid.
- Recommended operation: use corresponding W1TS and W1TC registers, such as `GPIO_OUT_W1TS/GPIO_OUT_W1TC` to set or clear the registers `GPIO_OUT_REG`.

## 5.6 Direct Input and Output via IO MUX

### 5.6.1 Overview

Some high speed digital signals (SPI and JTAG) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to peripherals.

This option is less flexible than routing signals via GPIO matrix, as the IO MUX register for each GPIO pin can only select from a limited number of functions, but high-frequency digital performance can be improved.

## 5.6.2 Functional Description

Two registers must be configured in order to bypass GPIO matrix for peripheral input signals:

1. `IO_MUX_GPIO $n$ _MCU_SEL` for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Section 5.12.
2. Clear `GPIO_SIG $n$ _IN_SEL` to route the input directly to the peripheral.

To bypass GPIO matrix for peripheral output signals, `IO_MUX_GPIO $n$ _MCU_SEL` for the GPIO pin must be set to the required pin function.

**Note:**

Not all signals can be directly connected to peripheral via IO MUX. Some input/output signals can only be connected to peripheral via GPIO matrix.

## 5.7 Analog Functions of GPIO Pins

Some GPIO pins in ESP8684 provide analog functions. When the pin is used for analog purpose, make sure that pull-up and pull-down resistors are disabled by following configuration:

- Set `IO_MUX_GPIO $n$ _MCU_SEL` to 1, and clear `IO_MUX_GPIO $n$ _FUN_IE`, `IO_MUX_GPIO $n$ _FUN_WPU`, `IO_MUX_GPIO $n$ _FUN_WPD`.
- Write 1 to `GPIO_ENABLE_WITC $[n]$` , to clear output enable.

See Table 5.13-1 for analog functions of ESP8684 pins.

## 5.8 Pin Functions in Light-sleep

Pins may provide different functions when ESP8684 is in Light-sleep mode. If `IO_MUX_SLP_SEL` in register `IO_MUX_ $n$ _REG` for a GPIO pin is set to 1, a different set of bits will be used to control the pin when the chip is in Light-sleep mode.

Table 5.8-1. Bits Used to Control IO MUX Functions in Light-sleep Mode

IO MUX Functions	Normal Execution OR <code>IO_MUX_SLP_SEL</code> = 0	Light-sleep Mode AND <code>IO_MUX_SLP_SEL</code> = 1
Output Drive Strength	<code>IO_MUX_FUN_DRV</code>	<code>IO_MUX_MCU_DRV</code>
Pull-up Resistor	<code>IO_MUX_FUN_WPU</code>	<code>IO_MUX_MCU_WPU</code>
Pull-down Resistor	<code>IO_MUX_FUN_WPD</code>	<code>IO_MUX_MCU_WPD</code>
Output Enable	<code>OEN_SEL</code> from GPIO matrix *	<code>IO_MUX_MCU_OE</code>

**Note:**

If `IO_MUX_SLP_SEL` is set to 0, pin functions remain the same in both normal execution and Light-sleep mode. Please refer to Section 5.5.2 for how to enable output in normal execution.

## 5.9 Pin Hold Feature

Each GPIO pin (including the RTC pins: GPIO0 ~ GPIO5) 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/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.

### Note:

- For digital pins (GPIO6 ~ 20), to maintain pin input/output status in Deep-sleep mode, users can set RTC\_CNTL\_DIG\_PAD\_HOLD[n] in register RTC\_CNTL\_DIG\_PAD\_HOLD\_REG to 1 before powering down. To disable the hold function after the chip is woken up, users can set RTC\_CNTL\_DIG\_PAD\_HOLD[n] to 0.
- For RTC pins (GPIO0 ~ 5), the input and output values are controlled by the corresponding bits of register RTC\_CNTL\_RTC\_PAD\_HOLD\_REG, and users can set it to 1 to hold the value or set it to 0 to unhold the value.

## 5.10 Power Supplies and Management of GPIO Pins

### 5.10.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in [ESP8684 Datasheet](#). All the pins can be used to wake up the chip from Light-sleep mode, but only the pins (GPIO0 ~ GPIO5) in VDD3P3\_RTC domain can be used to wake up the chip from Deep-sleep mode.

### 5.10.2 Power Supply Management

Each ESP8684 pin is connected to one of the two different power domains.

- VDD3P3\_RTC: the input power supply for both RTC and CPU
- VDD3P3\_CPU: the input power supply for CPU

## 5.11 Peripheral Signal List

Table 5.11-1 shows the peripheral input/output signals via GPIO matrix.

Please pay attention to the configuration of the bit `GPIO_FUNCn_OEN_SEL`:

- `GPIO_FUNCn_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_FUNCn_OEN_SEL` = 0: use the output enable signal from peripheral, for example `SPIQ_oe` in the column “Output enable signal when `GPIO_FUNCn_OEN_SEL` = 0” of Table 5.11-1. 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 the "Output enable signal when `GPIO_FUNCn_OEN_SEL` = 0", it indicates that once the register `GPIO_FUNCn_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 5.11-1 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 5.11-1 are valid output signals.

Table 5.11-1. Peripheral Signals via GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when <b>GPIO_FUNC<sub>n</sub>_OEN_SEL = 0</b>	Direct Output via IO MUX
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe	yes
1	SPID_in	0	yes	SPID_out	SPID_oe	yes
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe	yes
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe	yes
4	-	-	-	SPICLK_out_mux	SPICLK_oe	yes
5	-	-	-	SPICS0_out	SPICS0_oe	yes
6	UORXD_in	0	yes	UOTXD_out	1'd1	yes
7	UOCTS_in	0	no	UORTS_out	1'd1	no
8	UODSR_in	0	no	UODTR_out	1'd1	no
9	U1RXD_in	0	no	U1TXD_out	1'd1	no
10	U1CTS_in	0	no	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	-	-	-	-	-	-
13	-	-	-	-	-	-
14	-	-	-	-	-	-
15	-	-	-	SPIQ_monitor	1'd1	no
16	-	-	-	SPID_monitor	1'd1	no
17	-	-	-	SPIHD_monitor	1'd1	no
18	-	-	-	SPIWP_monitor	1'd1	no
19	-	-	-	SPICS1_out	SPICS1_oe	no
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-
24	-	-	-	-	-	-



Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when <code>GPIO_FUNC<math>n</math>_OEN_SEL = 0</code>	Direct Output via IO MUX
25	-	-	-	-	-	-
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	cpu_gpio_in0	0	no	cpu_gpio_out0	cpu_gpio_out_oen0	no
29	cpu_gpio_in1	0	no	cpu_gpio_out1	cpu_gpio_out_oen1	no
30	cpu_gpio_in2	0	no	cpu_gpio_out2	cpu_gpio_out_oen2	no
31	cpu_gpio_in3	0	no	cpu_gpio_out3	cpu_gpio_out_oen3	no
32	cpu_gpio_in4	0	no	cpu_gpio_out4	cpu_gpio_out_oen4	no
33	cpu_gpio_in5	0	no	cpu_gpio_out5	cpu_gpio_out_oen5	no
34	cpu_gpio_in6	0	no	cpu_gpio_out6	cpu_gpio_out_oen6	no
35	cpu_gpio_in7	0	no	cpu_gpio_out7	cpu_gpio_out_oen7	no
36	-	-	-	-	-	-
37	-	-	-	-	-	-
38	-	-	-	-	-	-
39	-	-	-	-	-	-
40	-	-	-	-	-	-
41	-	-	-	-	-	-
42	-	-	-	-	-	-
43	-	-	-	-	-	-
44	-	-	-	-	-	-
45	ext_adc_start	0	no	ledc_ls_sig_out0	1'd1	no
46	-	-	-	ledc_ls_sig_out1	1'd1	no
47	-	-	-	ledc_ls_sig_out2	1'd1	no
48	-	-	-	ledc_ls_sig_out3	1'd1	no
49	-	-	-	ledc_ls_sig_out4	1'd1	no
50	-	-	-	ledc_ls_sig_out5	1'd1	no
51	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when <a href="#">GPIO_FUNC</a> <a href="#">n_OEN_SEL</a> = 0	Direct Output via IO MUX
52	-	-	-	-	-	-
53	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
54	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
55	-	-	-	-	-	-
56	-	-	-	-	-	-
57	-	-	-	-	-	-
58	-	-	-	-	-	-
59	-	-	-	-	-	-
60	-	-	-	-	-	-
61	-	-	-	-	-	-
62	-	-	-	-	-	-
63	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
64	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
65	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes
66	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
67	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes
68	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
69	-	-	-	FSPICS1_out	FSPICS1_oe	no
70	-	-	-	FSPICS2_out	FSPICS2_oe	no
71	-	-	-	FSPICS3_out	FSPICS3_oe	no
72	-	-	-	FSPICS4_out	FSPICS4_oe	no
73	-	-	-	FSPICS5_out	FSPICS5_oe	no
74	-	-	-	-	-	-
75	-	-	-	-	-	-
76	-	-	-	-	-	-
77	-	-	-	-	-	-
78	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when <a href="#">GPIO_FUNC</a> <i>n</i> _OEN_SEL = 0	Direct Output via IO MUX
79	-	-	-	-	-	-
80	-	-	-	-	-	-
81	-	-	-	-	-	-
82	-	-	-	-	-	-
83	-	-	-	-	-	-
84	-	-	-	-	-	-
85	-	-	-	-	-	-
86	-	-	-	-	-	-
87	-	-	-	-	-	-
88	-	-	-	-	-	-
89	-	-	-	ant_sel0	1'd1	no
90	-	-	-	ant_sel1	1'd1	no
91	-	-	-	ant_sel2	1'd1	no
92	-	-	-	ant_sel3	1'd1	no
93	-	-	-	ant_sel4	1'd1	no
94	-	-	-	ant_sel5	1'd1	no
95	-	-	-	ant_sel6	1'd1	no
96	-	-	-	ant_sel7	1'd1	no
97	sig_in_func_97	0	no	sig_in_func97	1'd1	no
98	sig_in_func_98	0	no	sig_in_func98	1'd1	no
99	sig_in_func_99	0	no	sig_in_func99	1'd1	no
100	sig_in_func_100	0	no	sig_in_func100	1'd1	no
101	-	-	-	-	-	-
102	-	-	-	-	-	-
103	-	-	-	-	-	-
104	-	-	-	-	-	-
105	-	-	-	-	-	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when <code>GPIO_FUNC<math>\overline{n}</math>_OEN_SEL = 0</code>	Direct Output via IO MUX
106	-	-	-	-	-	-
107	-	-	-	-	-	-
108	-	-	-	-	-	-
109	-	-	-	-	-	-
110	-	-	-	-	-	-
111	-	-	-	-	-	-
112	-	-	-	-	-	-
113	-	-	-	-	-	-
114	-	-	-	-	-	-
115	-	-	-	-	-	-
116	-	-	-	-	-	-
117	-	-	-	-	-	-
118	-	-	-	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	-	-	-	-	-	-
122	-	-	-	-	-	-
123	-	-	-	CLK_OUT_out1	1'd1	no
124	-	-	-	CLK_OUT_out2	1'd1	no
125	-	-	-	CLK_OUT_out3	1'd1	no
126	-	-	-	-	-	-
127	-	-	-	-	-	-

## 5.12 IO MUX Functions List

Table 5.12-1 shows the IO MUX functions of each pin.

Table 5.12-1. IO MUX Pin Functions

Pin No.	Pin Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Note
0	GPIO0	GPIO0	GPIO0	-	-	2	0	R
1	GPIO1	GPIO1	GPIO1	-	-	2	0	R
2	GPIO2	GPIO2	GPIO2	FSPIQ	-	2	1	R
3	GPIO3	GPIO3	GPIO3	-	-	2	1	R
4	MTMS	MTMS	GPIO4	FSPIHD	-	2	1	R
5	MTDI	MTDI	GPIO5	FSPIWP	-	2	1	R
6	MTCK	MTCK	GPIO6	FSPICLK	-	2	1*	-
7	MTDO	MTDO	GPIO7	FSPID	-	2	1	-
8	GPIO8	GPIO8	GPIO8	-	-	2	1	-
9	GPIO9	GPIO9	GPIO9	-	-	2	3	-
10	GPIO10	GPIO10	GPIO10	FSPICSO	-	2	1	-
11	VDD_SPI	GPIO11	GPIO11	-	-	2	0	S
12	SPIHD	SPIHD	GPIO12	-	-	2	3	S
13	SPIWP	SPIWP	GPIO13	-	-	2	3	S
14	SPICSO	SPICSO	GPIO14	-	-	2	3	S
15	SPICLK	SPICLK	GPIO15	-	-	2	3	S
16	SPID	SPID	GPIO16	-	-	2	3	S
17	SPIQ	SPIQ	GPIO17	-	-	2	3	S
18	GPIO18	GPIO18	GPIO18	-	-	2	0	-
19	UORXD	UORXD	GPIO19	-	-	2	3	-
20	UOTXD	UOTXD	GPIO20	-	-	2	4	-

### Drive Strength

“DRV” column shows the drive strength of each pin after reset:

- 0 - Drive current = ~5 mA
- 1 - Drive current = ~10 mA
- 2 - Drive current = ~20 mA (default)
- 3 - Drive current = ~40 mA

### Reset Configurations

“Reset” column shows the default configuration of each pin after reset:

- 0 - IE = 0 (input disabled)
- 1 - IE = 1 (input enabled)
- 2 - IE = 1, WPD = 1 (input enabled, pull-down resistor enabled)

- **3** - IE = 1, WPU = 1 (input enabled, pull-up resistor enabled)
- **4** - OE = 1, WPU = 1 (output enabled, pull-up resistor enabled)
- **1\*** - If eFuse bit EFUSE\_DIS\_PAD\_JTAG = 1, the pin MTCK is left floating after reset, i.e. IE = 1. If eFuse bit EFUSE\_DIS\_PAD\_JTAG = 0, the pin MTCK is connected to internal pull-up resistor, i.e. IE = 1, WPU = 1.

**Note:**

- **R** - Pins in VDD3P3\_RTC domain, and part of them have analog functions, see Table 5.13-1.
- **S** - For chip variants with SiP flash, these pins are only used to connect SiP flash, i.e. only Function 0 is available. For chip variants without SiP flash, these pins can be used as normal pins, i.e. all the functions are available.

## 5.13 Analog Functions List

Table 5.13-1 shows the IO MUX pins with analog functions.

**Table 5.13-1. Analog Functions of IO MUX Pins**

GPIO No.	Pin Name	Analog Function
0	GPIO0	ADC1_CH0
1	GPIO1	ADC1_CH1
2	GPIO2	ADC1_CH2
3	GPIO3	ADC1_CH3
4	MTMS	ADC1_CH4
5	MTDI	ADC2_CH0

## 5.14 Register Summary

### 5.14.1 GPIO Matrix Register Summary

The addresses in this section are relative to GPIO base address provided in Table 3.3-3 in Chapter 3 [System and Memory](#).

The abbreviations given in Column **Access** are explained in Section [Access Types for Registers](#).

**Note:** For chip variants with SiP flash, only 14 GPIO pins are available, i.e. GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20. For this case:

- **Configuration Registers:** can only be configured for GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20;
- **Pin Configuration Registers:** only GPIO\_PIN0\_REG ~ GPIO\_PIN10\_REG and GPIO\_PIN18\_REG ~ GPIO\_PIN20\_REG are available;
- **Input Configuration Registers:** can only be configured for GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20;
- **Output Configuration Registers:** only GPIO\_FUNC0\_OUT\_SEL\_CFG\_REG ~ GPIO\_FUNC10\_OUT\_SEL\_CFG\_REG and GPIO\_PIN18\_OUT\_SEL\_CFG\_REG ~ GPIO\_PIN20\_OUT\_SEL\_CFG\_REG are available;

Name	Description	Address	Access
<b>Configuration Registers</b>			
<a href="#">GPIO_OUT_REG</a>	GPIO output register	0x0004	R/W/SS
<a href="#">GPIO_OUT_WITS_REG</a>	GPIO output set register	0x0008	WT
<a href="#">GPIO_OUT_WITC_REG</a>	GPIO output clear register	0x000C	WT
<a href="#">GPIO_ENABLE_REG</a>	GPIO output enable register	0x0020	R/W/SS
<a href="#">GPIO_ENABLE_WITS_REG</a>	GPIO output enable set register	0x0024	WT
<a href="#">GPIO_ENABLE_WITC_REG</a>	GPIO output enable clear register	0x0028	WT
<a href="#">GPIO_STRAP_REG</a>	Pin strapping register	0x0038	RO
<a href="#">GPIO_IN_REG</a>	GPIO input register	0x003C	RO
<a href="#">GPIO_STATUS_REG</a>	GPIO interrupt status register	0x0044	R/W/SS
<a href="#">GPIO_STATUS_WITS_REG</a>	GPIO interrupt status set register	0x0048	WT
<a href="#">GPIO_STATUS_WITC_REG</a>	GPIO interrupt status clear register	0x004C	WT
<a href="#">GPIO_PCPU_INT_REG</a>	GPIO CPU interrupt status register	0x005C	RO
<a href="#">GPIO_STATUS_NEXT_REG</a>	GPIO interrupt source register	0x014C	RO
<b>Pin Configuration Registers</b>			
<a href="#">GPIO_PIN0_REG</a>	GPIO pin 0 configuration register	0x0074	R/W
<a href="#">GPIO_PIN1_REG</a>	GPIO pin 1 configuration register	0x0078	R/W
<a href="#">GPIO_PIN2_REG</a>	GPIO pin 2 configuration register	0x007C	R/W
<a href="#">GPIO_PIN3_REG</a>	GPIO pin 3 configuration register	0x0080	R/W
<a href="#">GPIO_PIN4_REG</a>	GPIO pin 4 configuration register	0x0084	R/W
<a href="#">GPIO_PIN5_REG</a>	GPIO pin 5 configuration register	0x0088	R/W
<a href="#">GPIO_PIN6_REG</a>	GPIO pin 6 configuration register	0x008C	R/W
<a href="#">GPIO_PIN7_REG</a>	GPIO pin 7 configuration register	0x0090	R/W
<a href="#">GPIO_PIN8_REG</a>	GPIO pin 8 configuration register	0x0094	R/W
<a href="#">GPIO_PIN9_REG</a>	GPIO pin 9 configuration register	0x0098	R/W
<a href="#">GPIO_PIN10_REG</a>	GPIO pin 10 configuration register	0x009C	R/W
<a href="#">GPIO_PIN11_REG</a>	GPIO pin 11 configuration register	0x00A0	R/W
<a href="#">GPIO_PIN12_REG</a>	GPIO pin 12 configuration register	0x00A4	R/W
<a href="#">GPIO_PIN13_REG</a>	GPIO pin 13 configuration register	0x00A8	R/W
<a href="#">GPIO_PIN14_REG</a>	GPIO pin 14 configuration register	0x00AC	R/W
<a href="#">GPIO_PIN15_REG</a>	GPIO pin 15 configuration register	0x00B0	R/W
<a href="#">GPIO_PIN16_REG</a>	GPIO pin 16 configuration register	0x00B4	R/W
<a href="#">GPIO_PIN17_REG</a>	GPIO pin 17 configuration register	0x00B8	R/W
<a href="#">GPIO_PIN18_REG</a>	GPIO pin 18 configuration register	0x00BC	R/W
<a href="#">GPIO_PIN19_REG</a>	GPIO pin 19 configuration register	0x00C0	R/W
<a href="#">GPIO_PIN20_REG</a>	GPIO pin 20 configuration register	0x00C4	R/W
<b>Input Function Configuration Registers</b>			
<a href="#">GPIO_FUNC0_IN_SEL_CFG_REG</a>	Configuration register for input signal 0	0x0154	R/W
<a href="#">GPIO_FUNC1_IN_SEL_CFG_REG</a>	Configuration register for input signal 1	0x0158	R/W
...	...	...	...
<a href="#">GPIO_FUNC126_IN_SEL_CFG_REG</a>	Configuration register for input signal 126	0x034C	R/W
<a href="#">GPIO_FUNC127_IN_SEL_CFG_REG</a>	Configuration register for input signal 127	0x0350	R/W
<b>Output Function Configuration Registers</b>			

Name	Description	Address	Access
<a href="#">GPIO_FUNC0_OUT_SEL_CFG_REG</a>	Configuration register for GPIO0 output	0x0554	R/W
<a href="#">GPIO_FUNC1_OUT_SEL_CFG_REG</a>	Configuration register for GPIO1 output	0x0558	R/W
<a href="#">GPIO_FUNC2_OUT_SEL_CFG_REG</a>	Configuration register for GPIO2 output	0x055C	R/W
<a href="#">GPIO_FUNC3_OUT_SEL_CFG_REG</a>	Configuration register for GPIO3 output	0x0560	R/W
<a href="#">GPIO_FUNC4_OUT_SEL_CFG_REG</a>	Configuration register for GPIO4 output	0x0564	R/W
<a href="#">GPIO_FUNC5_OUT_SEL_CFG_REG</a>	Configuration register for GPIO5 output	0x0568	R/W
<a href="#">GPIO_FUNC6_OUT_SEL_CFG_REG</a>	Configuration register for GPIO6 output	0x056C	R/W
<a href="#">GPIO_FUNC7_OUT_SEL_CFG_REG</a>	Configuration register for GPIO7 output	0x0570	R/W
<a href="#">GPIO_FUNC8_OUT_SEL_CFG_REG</a>	Configuration register for GPIO8 output	0x0574	R/W
<a href="#">GPIO_FUNC9_OUT_SEL_CFG_REG</a>	Configuration register for GPIO9 output	0x0578	R/W
<a href="#">GPIO_FUNC10_OUT_SEL_CFG_REG</a>	Configuration register for GPIO10 output	0x057C	R/W
<a href="#">GPIO_FUNC11_OUT_SEL_CFG_REG</a>	Configuration register for GPIO11 output	0x0580	R/W
<a href="#">GPIO_FUNC12_OUT_SEL_CFG_REG</a>	Configuration register for GPIO12 output	0x0584	R/W
<a href="#">GPIO_FUNC13_OUT_SEL_CFG_REG</a>	Configuration register for GPIO13 output	0x0588	R/W
<a href="#">GPIO_FUNC14_OUT_SEL_CFG_REG</a>	Configuration register for GPIO14 output	0x058C	R/W
<a href="#">GPIO_FUNC15_OUT_SEL_CFG_REG</a>	Configuration register for GPIO15 output	0x0590	R/W
<a href="#">GPIO_FUNC16_OUT_SEL_CFG_REG</a>	Configuration register for GPIO16 output	0x0594	R/W
<a href="#">GPIO_FUNC17_OUT_SEL_CFG_REG</a>	Configuration register for GPIO17 output	0x0598	R/W
<a href="#">GPIO_FUNC18_OUT_SEL_CFG_REG</a>	Configuration register for GPIO18 output	0x059C	R/W
<a href="#">GPIO_FUNC19_OUT_SEL_CFG_REG</a>	Configuration register for GPIO19 output	0x05A0	R/W
<a href="#">GPIO_FUNC20_OUT_SEL_CFG_REG</a>	Configuration register for GPIO20 output	0x05A4	R/W
<b>Version Register</b>			
<a href="#">GPIO_DATE_REG</a>	GPIO version register	0x06FC	R/W
<b>Clock Gate Register</b>			
<a href="#">GPIO_CLOCK_GATE_REG</a>	GPIO clock gate register	0x062C	R/W

## 5.14.2 IO MUX Register Summary

The addresses in this section are relative to the IO MUX base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

**Note:** For chip variants with SiP flash, only 14 GPIO pins are available, i.e. GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20. For this case, IO\_MUX\_GPIO11\_REG ~ IO\_MUX\_GPIO17\_REG are not configurable.

Name	Description	Address	Access
<b>Configuration Registers</b>			
<a href="#">IO_MUX_PIN_CTRL_REG</a>	Clock output configuration Register	0x0000	R/W
<a href="#">IO_MUX_GPIO0_REG</a>	IO MUX configuration register for pin GPIO0	0x0004	R/W
<a href="#">IO_MUX_GPIO1_REG</a>	IO MUX configuration register for pin GPIO1	0x0008	R/W
<a href="#">IO_MUX_GPIO2_REG</a>	IO MUX configuration register for pin GPIO2	0x000C	R/W
<a href="#">IO_MUX_GPIO3_REG</a>	IO MUX configuration register for pin GPIO3	0x0010	R/W
<a href="#">IO_MUX_GPIO4_REG</a>	IO MUX configuration register for pin MTMS	0x0014	R/W
<a href="#">IO_MUX_GPIO5_REG</a>	IO MUX configuration register for pin MTDI	0x0018	R/W



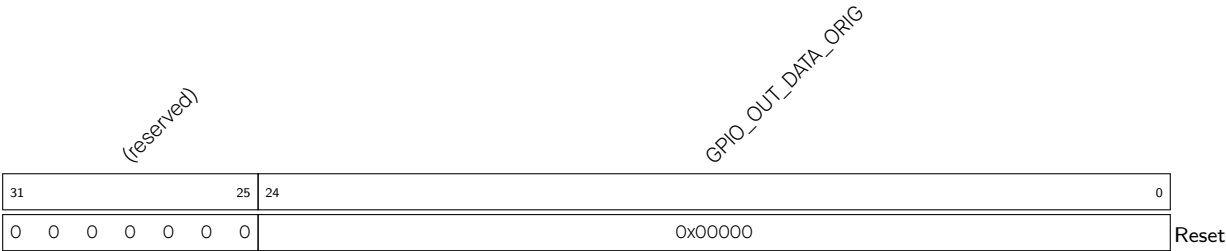
Name	Description	Address	Access
<a href="#">IO_MUX_GPIO6_REG</a>	IO MUX configuration register for pin MTCK	0x001C	R/W
<a href="#">IO_MUX_GPIO7_REG</a>	IO MUX configuration register for pin MTDO	0x0020	R/W
<a href="#">IO_MUX_GPIO8_REG</a>	IO MUX configuration register for pin GPIO8	0x0024	R/W
<a href="#">IO_MUX_GPIO9_REG</a>	IO MUX configuration register for pin GPIO9	0x0028	R/W
<a href="#">IO_MUX_GPIO10_REG</a>	IO MUX configuration register for pin GPIO10	0x002C	R/W
<a href="#">IO_MUX_GPIO11_REG</a>	IO MUX configuration register for pin VDD_SPI	0x0030	R/W
<a href="#">IO_MUX_GPIO12_REG</a>	IO MUX configuration register for pin SPIHD	0x0034	R/W
<a href="#">IO_MUX_GPIO13_REG</a>	IO MUX configuration register for pin SPIWP	0x0038	R/W
<a href="#">IO_MUX_GPIO14_REG</a>	IO MUX configuration register for pin SPICSO	0x003C	R/W
<a href="#">IO_MUX_GPIO15_REG</a>	IO MUX configuration register for pin SPICLK	0x0040	R/W
<a href="#">IO_MUX_GPIO16_REG</a>	IO MUX configuration register for pin SPID	0x0044	R/W
<a href="#">IO_MUX_GPIO17_REG</a>	IO MUX configuration register for pin SPIQ	0x0048	R/W
<a href="#">IO_MUX_GPIO18_REG</a>	IO MUX configuration register for pin GPIO18	0x004C	R/W
<a href="#">IO_MUX_GPIO19_REG</a>	IO MUX configuration register for pin UORXD	0x0050	R/W
<a href="#">IO_MUX_GPIO20_REG</a>	IO MUX configuration register for pin UOTXD	0x0054	R/W
Version Register			
<a href="#">IO_MUX_DATE_REG</a>	IO MUX Version Control Register	0x00FC	R/W

## 5.15 Registers

### 5.15.1 GPIO Matrix Registers

The addresses in this section are relative to GPIO base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 5.1. GPIO\_OUT\_REG (0x0004)



**GPIO\_OUT\_DATA\_ORIG** GPIO0 ~ 20 output value in simple GPIO output mode. The values of bit0 ~ bit20 correspond to the output value of GPIO0 ~ GPIO20 respectively, and bit21 ~ bit24 are invalid. (R/W/SS)

**Register 5.2. GPIO\_OUT\_W1TS\_REG (0x0008)**

(reserved)								GPIO_OUT_W1TS																					
3125								240																					
0000000								0x00000Reset																					

**GPIO\_OUT\_W1TS** GPIO0 ~ 20 output set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in [GPIO\\_OUT\\_REG](#) will be set to 1. Recommended operation: use this register to set [GPIO\\_OUT\\_REG](#). (WT)

**Register 5.3. GPIO\_OUT\_W1TC\_REG (0x000C)**

(reserved)								GPIO_OUT_W1TC																					
3125								240																					
0000000								0x00000Reset																					

**GPIO\_OUT\_W1TC** GPIO0 ~ 20 output clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in [GPIO\\_OUT\\_REG](#) will be cleared. Recommended operation: use this register to clear [GPIO\\_OUT\\_REG](#). (WT)

**Register 5.4. GPIO\_ENABLE\_REG (0x0020)**

(reserved)								GPIO_ENABLE_DATA																					
3125								240																					
0000000								0x00000Reset																					

**GPIO\_ENABLE\_DATA** GPIO output enable register for GPIO0 ~ 20. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (R/W/SS)

(reserved)

GPIO\_ENABLE\_WTTS

Reset

**GPIO\_ENABLE\_WITS** GPIO0 ~ 20 output enable set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in [GPIO\\_ENABLE\\_REG](#) will be set to 1. Recommended operation: use this register to set [GPIO\\_ENABLE\\_REG](#). (WT)

(reserved)

GPIO\_ENABLE\_W1TC

Reset

**GPIO\_ENABLE\_WITC** GPIO0 ~ 20 output enable clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in **GPIO\_ENABLE\_REG** will be cleared. Recommended operation: use this register to clear **GPIO\_ENABLE\_REG**. (WT)

(reserved)

GPIO\_STRAPPING

Reset

**GPIO\_STRAPPING** GPIO strapping values. (RO)

- bit 2: GPIO8
- bit 3: GPIO9

### Register 5.8. GPIO\_IN\_REG (0x003C)

(reserved)								GPIO_IN_DATA_NEXT																										
31								25	24																									0
0	0	0	0	0	0	0	0	0x000000																								Reset		

**GPIO\_IN\_DATA\_NEXT** GPIO0 ~ 20 input value. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. Each bit represents a pin input value, 1 for high level and 0 for low level.  
(RO)

### Register 5.9. GPIO\_STATUS\_REG (0x0044)

Diagram of the `GPIO_STATUS_INTERRUPT` register. The register is 32 bits wide. Bits 31 down to 25 are reserved. Bits 24 down to 0 are used for the `GPIO_STATUS_INTERRUPT`. The reset value is 0x000000.

31	25	24	0
(reserved)		GPIO_STATUS_INTERRUPT	
0 0 0 0 0 0 0		0x000000	
Reset			

**GPIO\_STATUS\_INTERRUPT** GPIO0 ~ 20 interrupt status register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (R/W/SS)

### Register 5.10. GPIO\_STATUS\_W1TS\_REG (0x0048)

31	25	24	0
(reserved)		GPIO_STATUS_WTS	
0	0	0	0
0 0 0 0 0 0 0		0x000000	

Reset

**GPIO\_STATUS\_W1TS** GPIO0 ~ 20 interrupt status set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in **GPIO\_STATUS\_INTERRUPT** will be set to 1. Recommended operation: use this register to set **GPIO\_STATUS\_INTERRUPT**. (WT)

**Register 5.11. GPIO\_STATUS\_W1TC\_REG (0x004C)**

(reserved)								GPIO_STATUS_W1TC																					
3125								240																					
0000000								0x00000Reset																					

**GPIO\_STATUS\_W1TC** GPIO0 ~ 20 interrupt status clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in [GPIO\\_STATUS\\_INTERRUPT](#) will be cleared. Recommended operation: use this register to clear [GPIO\\_STATUS\\_INTERRUPT](#). (WT)

**Register 5.12. GPIO\_PCPU\_INT\_REG (0x005C)**

(reserved)								GPIO_PROCPU_INT																						
31					25	24																							0	
0	0	0	0	0	0	0	0	0x00000																						Reset

**GPIO\_PROCPU\_INT** GPIO0 ~ 20 CPU interrupt status. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. This interrupt status is corresponding to the bit in [GPIO\\_STATUS\\_REG](#) when assert (high) enable signal (bit13 of [GPIO\\_PIN<sub>n</sub>\\_REG](#)). (RO)

Register 5.13. GPIO\_PIN $n$ \_REG ( $n$ : 0-20) (0x0074+4\* $n$ )

(reserved)																GPIO_PIN <sub>n</sub> _INT_ENA				GPIO_PIN <sub>n</sub> _CONFIG		GPIO_PIN <sub>n</sub> _WAKEUP_ENABLE		(reserved)		GPIO_PIN <sub>n</sub> _SYNC1_BYPASS		GPIO_PIN <sub>n</sub> _PAD_DRIVER		GPIO_PIN <sub>n</sub> _SYNC2_BYPASS					
31																18	17				13	12	11	10	9			7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0		0x0		0	0x0		0	0	0x0	0	0x0		Reset							

**GPIO\_PIN $n$ \_SYNC2\_BYPASS** For the second-level synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

**GPIO\_PIN<sub>n</sub>\_PAD\_DRIVER** Pin drive selection. 0: normal output; 1: open drain output. (R/W)

**GPIO\_PIN<sub>n</sub>\_SYNC1\_BYPASS** For the first-level synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

**GPIO\_PIN*****n*****\_INT\_TYPE** Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

**GPIO\_PIN<sub>n</sub>\_WAKEUP\_ENABLE** GPIO wake-up enable bit, only wakes up the CPU from Light-sleep.  
(R/W)

**GPIO\_PIN<sub>n</sub>\_CONFIG** reserved (R/W)

**GPIO\_PIN*****n*****\_INT\_ENA** Interrupt enable bits. bit13: CPU interrupt enabled; bit14: CPU non-maskable interrupt enabled. (R/W)

### Register 5.14. GPIO\_STATUS\_NEXT\_REG (0x014C)

Diagram illustrating the structure of the `GPIO_STATUS_INTERRUPT` register. The register is 32 bits wide, divided into two 16-bit sections. The top 16 bits are labeled `(reserved)`. The bottom 16 bits are labeled `GPIO_STATUS_INTERRUPT_NEXT`. The bottom 16 bits are further divided into two 8-bit sections, each labeled `0`. The bottom 8 bits are labeled `Reset`.

**GPIO\_STATUS\_INTERRUPT\_NEXT** Interrupt source signal of GPIO0 ~ 20, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (RO)

Register 5.15. GPIO\_FUNC $n$ \_IN\_SEL\_CFG\_REG ( $n$ : 0-127) (0x0154+4\* $n$ )

(reserved)																												GPIO_SIG <sub>n</sub> _IN_SEL		GPIO_FUNC <sub>n</sub> _IN_INV_SEL		GPIO_FUNC <sub>n</sub> _IN_SEL	
31																												7	6	5	4	0	
0 0																												0	0	0x0		Reset	

**GPIO\_FUNC $n$ \_IN\_SEL** Selection control for peripheral input signal  $n$ , selects a pin from the 21 GPIO matrix pins to connect this input signal. Or selects 0x1E for a constantly high input or 0x1F for a constantly low input. (R/W)

**GPIO\_FUNC $n$ \_IN\_INV\_SEL** Invert the input value. 1: invert enabled; 0: invert disabled. (R/W)

**GPIO\_SIG $n$ \_IN\_SEL** Bypass GPIO matrix. 1: route signals via GPIO matrix, 0: connect signals directly to peripheral configured in IO MUX. (R/W)

Register 5.16. GPIO\_FUNC $n$ \_OUT\_SEL\_CFG\_REG ( $n$ : 0-20) (0x0554+4\* $n$ )

(reserved)																GPIO_FUNC $n$ _OEN_INV_SEL		GPIO_FUNC $n$ _OEN_SEL		GPIO_FUNC $n$ _OUT_INV_SEL		GPIO_FUNC $n$ _OUT_SEL	
31																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	0x80		Reset

**GPIO\_FUNC $n$ \_OUT\_SEL** Selection control for GPIO output  $n$ . If a value  $Y$  ( $0 \leq Y < 128$ ) is written to this field, the peripheral output signal  $Y$  will be connected to GPIO output  $n$ . If a value 128 is written to this field, bit  $n$  of [GPIO\\_OUT\\_REG](#) and [GPIO\\_ENABLE\\_REG](#) will be selected as the output value and output enable. (R/W)

**GPIO\_FUNC $n$ \_OUT\_INV\_SEL** 0: Do not invert the output value; 1: Invert the output value. (R/W)

**GPIO\_FUNC $n$ \_OEN\_SEL** 0: Use output enable signal from peripheral; 1: Force the output enable signal to be sourced from bit  $n$  of [GPIO\\_ENABLE\\_REG](#). (R/W)

**GPIO\_FUNC $n$ \_OEN\_INV\_SEL** 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

### Register 5.17. GPIO\_CLOCK\_GATE\_REG (0x062C)

[illegible]

**GPIO\_CLK\_EN** Clock gating enable bit. If set to 1, the clock is free running. (R/W)

### Register 5.18. GPIO\_DATE\_REG (0x06FC)

(reserved)				GPIO_DATE_REG																												
31	28	27																													0	
0	0	0	0	0x2006130																												Reset

**GPIO\_DATE\_REG** Version control register (R/W)

### 5.15.2 IO MUX Registers

The addresses in this section are relative to the IO MUX base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

### Register 5.19. IO\_MUX\_PIN\_CTRL\_REG (0x0000)

(reserved)																IO_MUX_CLK_OUT3		IO_MUX_CLK_OUT2		IO_MUX_CLK_OUT1		
31121187430																						Reset
00																						

**IO\_MUX\_CLK\_OUT<sub>x</sub>** If you want to output clock for I2S to CLK\_OUT\_out<sub>x</sub>, set IO\_MUX\_CLK\_OUT<sub>x</sub> to 0x0. CLK\_OUT\_out<sub>x</sub> can be found in Table 5.11-1. (R/W)



Register 5.20. IO\_MUX\_GPIO*n*\_REG (*n*: 0-20) (0x0004+4\**n*)

[illegible]

**IO\_MUX\_GPIO<sub>n</sub>MCU\_OE** Output enable of the pin in sleep mode. 1: output enabled; 0: output disabled. (R/W)

**IO\_MUX\_GPIO $n$ \_SLP\_SEL** Sleep mode selection of this pin. Set to 1 to put the pin in sleep mode.  
(R/W)

**IO\_MUX\_GPIO $n$ \_MCU\_WPD** Pull-down enable of the pin in sleep mode. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

**IO\_MUX\_GPIO $n$ \_MCU\_WPU** Pull-up enable of the pin during sleep mode. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

**IO\_MUX\_GPIO<sub>n</sub>MCU\_IE** Input enable of the pin during sleep mode. 1: input enabled; 0: input disabled. (R/W)

**IO\_MUX\_GPIO $n$ \_MCU\_DRV** Configures the drive strength of GPIO $n$  during sleep mode.

0: ~5 mA

1:  $\sim 10$  mA

2: ~ 20 mA

3: ~40 mA

(R/W)

**IO\_MUX\_GPIO<sub>n</sub>\_FUN\_WPD** Pull-down enable of the pin. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

**IO\_MUX\_GPIO<sub>n</sub>\_FUN\_WPU** Pull-up enable of the pin. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

**IO\_MUX\_GPIO<sub>n</sub>\_FUN\_IE** Input enable of the pin. 1: input enabled; 0: input disabled. (R/W)

**IO\_MUX\_GPIO*n*\_FUN\_DRV** Select the drive strength of the pin. 0: ~ 5 mA; 1: ~ 10 mA; 2: ~ 20 mA; 3: ~ 40 mA. (R/W)

**IO\_MUX\_GPIO $n$ \_MCU\_SEL** Select IO MUX function for this pin. 0: Select Function 0; 1: Select Function 1; etc. (R/W)

**IO\_MUX\_GPIO***n*\_FILTER\_EN Enable filter for pin input signals. 1: filter enabled; 0: filter disabled.  
(R/W)

Register 5.21. IO\_MUX\_DATE\_REG (0x00FC)

(reserved)				IO_MUX_DATE_REG																										
31	28	27	0																											
0	0	0	0	0x2006050																										Reset

IO\_MUX\_DATE\_REG Version control register (R/W)

## Chapter 6

## Reset and Clock

### 6.1 Reset

#### 6.1.1 Overview

ESP8684 provides four types of reset that occur at different levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset. All reset types mentioned above (except Chip Reset) maintain the data stored in internal memory. Figure 6.1-1 shows the scope of affected subsystems by each type of reset.

#### 6.1.2 Architectural Overview

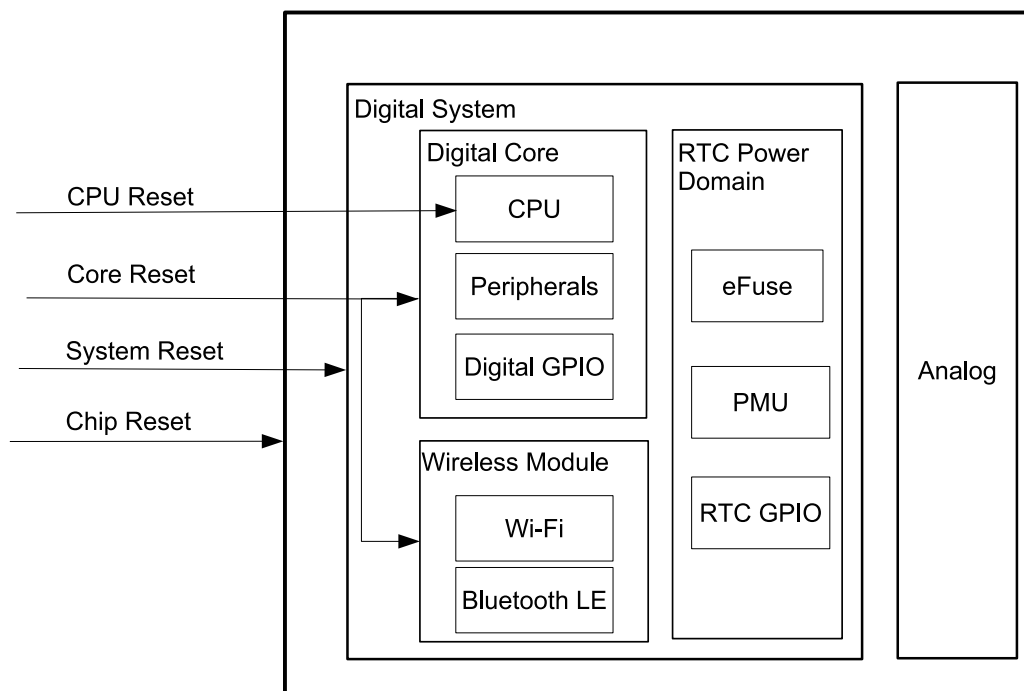


Figure 6.1-1. Reset Types

#### 6.1.3 Features

- Support four reset types:
  - CPU Reset: Only resets CPU core. Once such reset is released, the instructions from the CPU reset vector will be executed.
  - Core Reset: Resets the whole digital system except RTC, including CPU, peripherals, Wi-Fi, Bluetooth® LE, and digital GPIOs.

- System Reset: Resets the whole digital system, including RTC.
- Chip Reset: Resets the whole chip.
- Support software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 9 *Low-power Management (RTC\_CNTL)*.
  - Hardware Reset: triggered directly by the hardware.

**Note:**

If CPU is reset, [SENSITIVE registers](#) will be reset, too.

## 6.1.4 Functional Description

CPU will be reset immediately when any of the resets above occurs. Users can get reset source codes by reading register `RTC_CNTL_RESET_CAUSE_PROCPU` after the reset is released.

Table 6.1-1 lists possible reset sources and the types of reset they trigger.

**Table 6.1-1. Reset Sources**

Code	Source	Reset Type	Comments
0x01	Chip reset <sup>1</sup>	Chip Reset	—
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by brown-out detector <sup>2</sup>
0x10	RWDT system reset	System Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>
0x12	Analog Super Watchdog reset	System Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>
0x13	Clock glitch reset	System Reset	—
0x03	Software system reset	Core Reset	Triggered by configuring <code>RTC_CNTL_SW_SYS_RST</code>
0x05	Deep-sleep reset	Core Reset	See Chapter 9 <i>Low-power Management (RTC_CNTL)</i>
0x07	MWDT0 core reset	Core Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>
0x09	RWDT core reset	Core Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>
0x14	eFuse reset	Core Reset	Triggered by eFuse CRC error
0x18	JTAG reset	CPU Reset	Triggered by JTAG
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>
0x0C	Software CPU reset	CPU Reset	Triggered by configuring <code>RTC_CNTL_SW_PROCPU_RST</code>
0x0D	RWDT CPU reset	CPU Reset	See Chapter 12 <i>Watchdog Timers (WDT)</i>

<sup>1</sup> Chip Reset can be triggered by the following two sources:

- Triggered by chip power-on.
- Triggered by brown-out detector.

<sup>2</sup> Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on the configuration of `RTC_CNTL_BROWN_OUT_RST_SEL`. See Chapter 9 *Low-power Management (RTC\_CNTL)*.

## 6.2 Clock

### 6.2.1 Overview

ESP8684 clocks are mainly sourced from external oscillator (OSC), RC, and PLL circuit, and then processed by the dividers or selectors, which allows most functional modules to select their working clock according to their power consumption and performance requirements. Figure 6.2-1 shows the system clock structure.

### 6.2.2 Architectural Overview

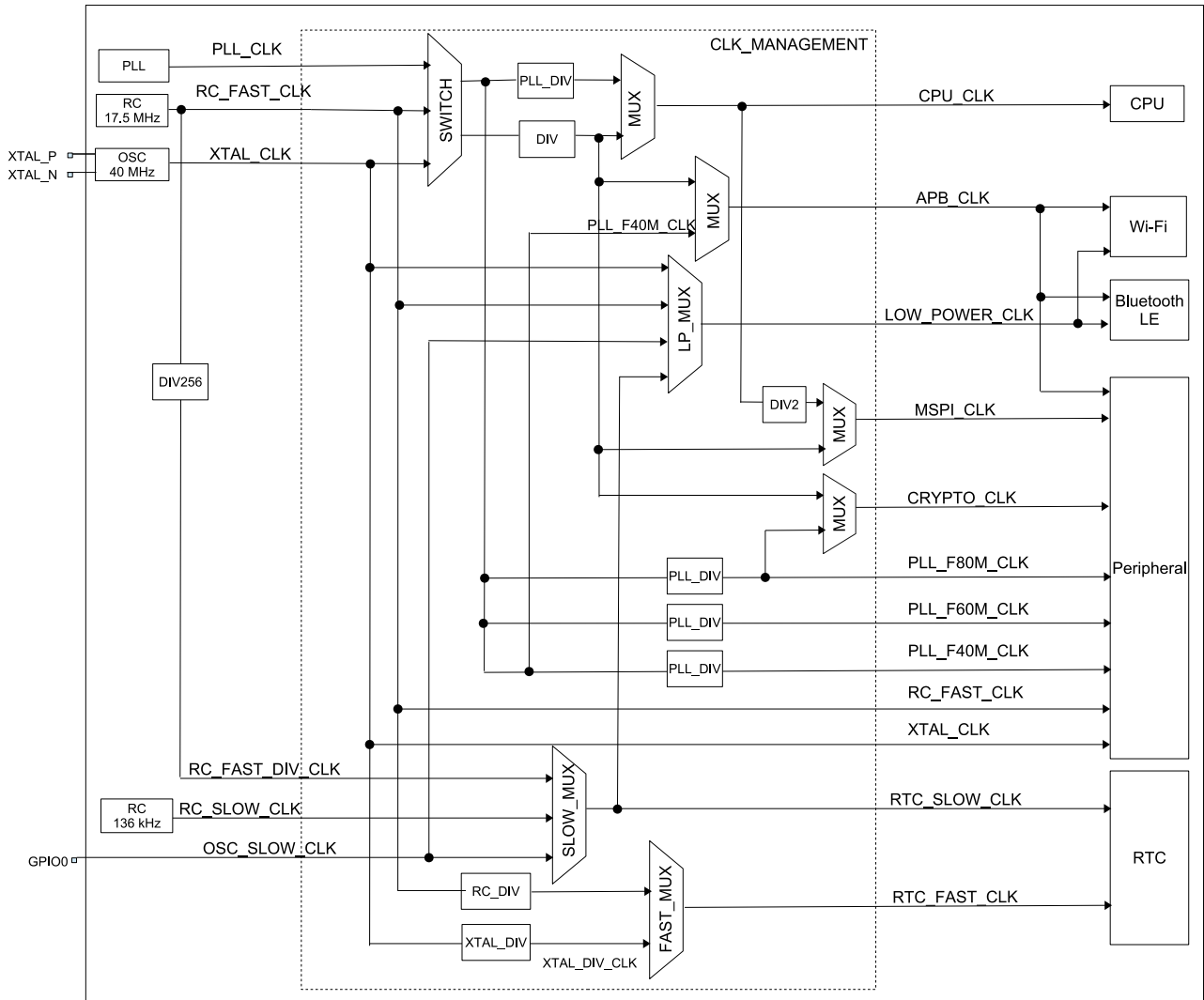


Figure 6.2-1. System Clock

### 6.2.3 Features

ESP8684 clocks can be classified in two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL\_CLK (480 MHz): internal PLL clock

- XTAL\_CLK (26 MHz or 40 MHz): external crystal clock
- Slow speed clocks for low-power devices, such as RTC module and low-power peripherals
  - OSC\_SLOW\_CLK (usually 32 kHz): external slow clock from GPIO0
  - RC\_FAST\_CLK (17.5 MHz by default): internal fast RC oscillator with adjustable frequency
  - FOSC\_DIV\_CLK: internal fast RC oscillator derived from RC\_FAST\_CLK divided by 256
  - RC\_SLOW\_CLK (136 kHz by default): internal slow RC oscillator with adjustable frequency

## 6.2.4 Functional Description

### 6.2.4.1 CPU Clock

As Figure 6.2-1 shows, CPU\_CLK is the master clock for CPU and it can be as high as 120 MHz when CPU works in high performance mode. Alternatively, CPU can run at lower frequencies, such as at 2 MHz, to lower power consumption. Users can set PLL\_CLK, RC\_FAST\_CLK or XTAL\_CLK as CPU\_CLK clock source by configuring SYSTEM\_SOC\_CLK\_SEL, see Table 6.2-1 and Table 6.2-2. By default, the CPU clock is sourced from XTAL\_CLK with a divider of 2, i.e. the CPU clock is 20 MHz.

Table 6.2-1. CPU Clock Source

SYSTEM_SOC_CLK_SEL	CPU Clock Source
0	XTAL_CLK
1	PLL_CLK
2	RC_FAST_CLK

Table 6.2-2. CPU Clock Frequency

CPU Clock Source	SEL_0*	SEL_1*	CPU Clock Frequency
XTAL_CLK	0	-	$CPU\_CLK = XTAL\_CLK / (SYSTEM\_PRE\_DIV\_CNT + 1)$ SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
PLL_CLK	1	0	$CPU\_CLK = PLL\_CLK / 6$ CPU_CLK frequency is 80 MHz
PLL_CLK	1	1	$CPU\_CLK = PLL\_CLK / 4$ CPU_CLK frequency is 120 MHz
RC_FAST_CLK	2	-	$CPU\_CLK = RC\_FAST\_CLK / (SYSTEM\_PRE\_DIV\_CNT + 1)$ SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1

\* The value of SYSTEM\_SOC\_CLK\_SEL

\* The value of SYSTEM\_CPUPERIOD\_SEL

### 6.2.4.2 Peripheral Clock

Peripheral clocks are classified into two categories:

- Bus clock: APB\_CLK

- Functional clocks: CRYPTO\_CLK, PLL\_F80M\_CLK, PLL\_F60M\_CLK, PLL\_F40M\_CLK, MSPI\_CLK, XTAL\_CLK, and RC\_FAST\_CLK.

Table 6.2-3 shows which clock can be used by each peripheral.

Table 6.2-3. Peripheral Clocks

Peripheral	XTAL_CLK	RC_FAST_CLK	PLL_F40M_CLK	PLL_F60M_CLK	PLL_F80M_CLK	(RTC) RTC_FAST_CLK	CRYPTO_CLK	MSPI_CLK
Timer Group	Y		Y					
UART	Y	Y	Y					
I2C	Y	Y						
SPI	Y		Y					
LEDC	Y	Y		Y				
SAR ADC	Y				Y			
Temperature sensor	Y	Y						
System Timer	Y							
Crypto							Y	
MSPI								Y
eFuse						Y		



**APB\_CLK**

The frequency of APB\_CLK is determined by the clock source of CPU\_CLK as shown in Table 6.2-4.

**Table 6.2-4. APB\_CLK Clock Frequency**

CPU_CLK Source	APB_CLK Frequency
PLL_CLK	40 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

**CRYPTO\_CLK**

The frequency of CRYPTO\_CLK is determined by the CPU\_CLK source, as shown in Table 6.2-5.

**Table 6.2-5. CRYPTO\_CLK Frequency**

CPU_CLK Source	CRYPTO_CLK Frequency
PLL_CLK	80 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

**MSPI\_CLK**

The frequency of MSPI\_CLK is determined by the CPU\_CLK source, as shown in Table 6.2-6.

**Table 6.2-6. MSPI\_CLK Frequency**

CPU_CLK Source	MSPI_CLK Frequency
PLL_CLK	CPU_CLK/2
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

**PLL\_F80M\_CLK, PLL\_F60M\_CLK, PLL\_F40M\_CLK**

PLL\_F80M\_CLK, PLL\_F60M\_CLK, and PLL\_F40M\_CLK are divided from PLL\_CLK according to current PLL frequency.

**6.2.4.3 Wireless Clock**

The wireless clock (LOW\_POWER\_CLK) in ESP8684 is used for Wi-Fi and Bluetooth LE in low-power mode. The clock source of LOW\_POWER\_CLK can be:

- OSC\_SLOW\_CLK
- XTAL\_CLK
- RC\_FAST\_CLK
- RTC\_SLOW\_CLK (the low clock selected by RTC)

**Note:** Wi-Fi and Bluetooth LE can only work when CPU\_CLK uses PLL\_CLK as its clock source. Suspending PLL\_CLK requires that Wi-Fi and Bluetooth LE have entered low-power mode first.

#### 6.2.4.4 RTC Clock

RTC module can operate when most other clocks are stopped. RTC clocks include RTC\_SLOW\_CLK and RTC\_FAST\_CLK.

The clock sources for RTC\_SLOW\_CLK and RTC\_FAST\_CLK are low-frequency clocks:

- RTC\_SLOW\_CLK, used to clock RTC timer, RTC watch dog, and low-power controller, can be derived from:
  - OSC\_SLOW\_CLK
  - RC\_SLOW\_CLK
  - or FOSC\_DIV\_CLK
- RTC\_FAST\_CLK, used to clock RTC peripherals and on-chip sensor module, can be derived from
  - XTAL\_CLK divided by 2
  - or RC\_FAST\_CLK divided by N

## Chapter 7

### Chip Boot Control

#### 7.1 Overview

Strapping pins are the specific chip pins used to control the following functions during chip power-on or hardware reset of ESP8684:

- control chip boot mode
- enable or disable ROM code printing to UART

ESP8684 has two strapping pins:

- GPIO8
- GPIO9

During Chip Reset (see Chapter [6 Reset and Clock](#)), hardware captures samples and stores the voltage level of strapping pins as strapping bit of “0” or “1” in latches, and holds these bits until the chip is powered down. Software can read the latch status (strapping value) from [GPIO\\_STRAPPING](#).

#### 7.2 Features

- Control of chip function on boot with strapping pins:
  - GPIO8
  - GPIO9
- Able to control chip boot mode:
  - SPI Boot mode
  - Download Boot mode
- Able to control ROM code printing to UART
- Allow the reading of strapping pin values from [GPIO\\_STRAPPING](#)

#### 7.3 Functional Description

This section provides description of the chip functions and the pattern of the strapping pins values to invoke each function.

**Notice:**

Only documented patterns should be used. If some pattern is not documented, it may trigger unexpected behavior.

### 7.3.1 Default Configuration

By default, GPIO9 is connected to the chip's internal pull-up resistor. If GPIO9 is not connected or connected to an external high-impedance circuit, the internal weak pull-up determines the default input level of this strapping pin (see Table 7.3-1).

**Table 7.3-1. Default Configuration of Strapping Pins**

Strapping Pin	Default Configuration
GPIO8	N/A
GPIO9	Pull-up

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP8684. After the reset is released, the strapping pins work as normal-function pins.

### 7.3.2 Boot Mode Control

The values of GPIO2, GPIO3, GPIO8, and GPIO9 at reset determine the boot mode after the reset is released. Table 7.3-2 shows the strapping pin values of GPIO2, GPIO3, GPIO8, and GPIO9, and the associated boot modes.

**Table 7.3-2. Boot Mode Control**

Boot Mode	GPIO9	GPIO8	GPIO3	GPIO2
SPI Boot mode	1	x <sup>1</sup>	x	x
Joint Download Boot mode <sup>2</sup>	0	1	x	x
SPI Download Boot mode <sup>3</sup>	0	0	0	1
Invalid Combination <sup>4</sup>	0	0	x	0

<sup>1</sup> x: values that have no effect on the result and can therefore be ignored.

<sup>2</sup> Joint Download Boot mode: Joint Download Boot mode supports the following download method:

- UART Download Boot

<sup>3</sup> SPI Download Boot mode: GPIO3 and GPIO2 need to be reserved only when using SPI Download Boot mode. GPIO3 and GPIO2 are floating by default and are in a high-impedance state at reset.

<sup>4</sup> Invalid Combination: This combination can trigger unexpected behavior and should be avoided.

In SPI Boot mode, the ROM bootloader loads and executes the program from SPI flash to boot the system. SPI Boot mode can be further classified as follows:

- Normal Flash Boot: supports Secure Boot. The ROM bootloader loads the program from flash into RAM and executes it. In most practical scenarios, this program is the 2nd stage bootloader, which later boots the target application.

- Direct Boot: does not support Secure Boot and programs run directly from flash. To enable this mode, make sure that the first two words of the bin file downloaded to flash are 0xaedb041d. For more detailed process, see Figure 7.3-1.

In Joint Download Boot mode, users can download binary files into flash using UART0 interface. It is also possible to load binary files into SRAM and execute it from SRAM.

In SPI Download Boot mode, users can download binary files into flash using SPI interface. It is also possible to download binary files into SRAM and execute it from SRAM.

Figure 7.3-1 shows the detailed boot flow of the chip.

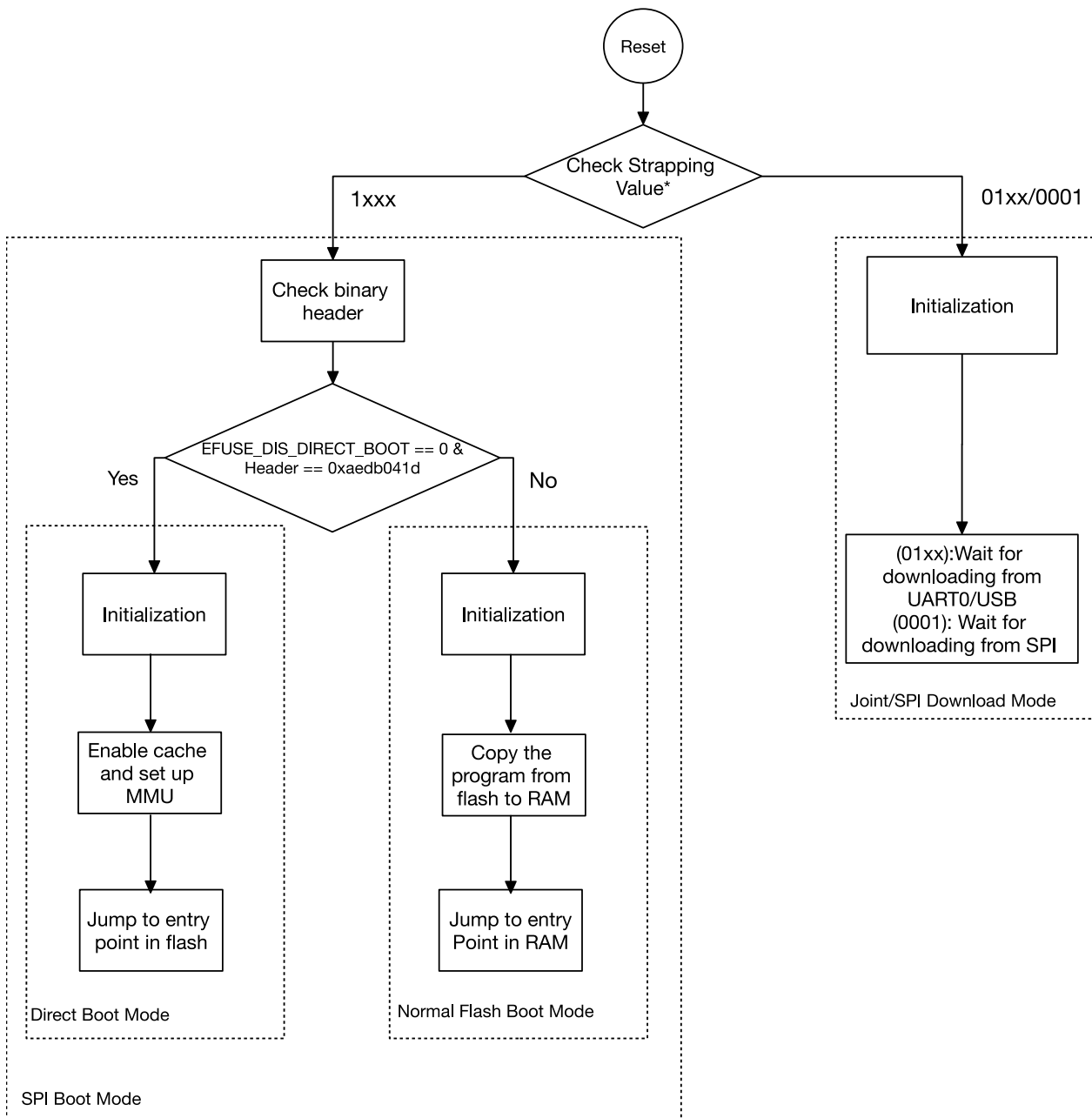


Figure 7.3-1. Chip Boot Flow

The following registers control boot mode behaviors:

- [RTC\\_CNTL\\_FORCE\\_DOWNLOAD\\_BOOT](#)

Software can force switch the chip from SPI Boot mode to Joint Download Boot mode by setting register [RTC\\_CNTL\\_FORCE\\_DOWNLOAD\\_BOOT](#) and triggering a CPU reset. In this case, hardware overwrites [GPIO\\_STRAPPING](#)[3:2] from “1x” to “01”.

- [EFUSE\\_DIS\\_DOWNLOAD\\_MODE](#)

If this eFuse is 1, Joint Download Boot mode is disabled. [GPIO\\_STRAPPING](#) will not be overwritten by [RTC\\_CNTL\\_FORCE\\_DOWNLOAD\\_BOOT](#).

- [EFUSE\\_ENABLE\\_SECURITY\\_DOWNLOAD](#)

If this eFuse is 1, Joint Download Boot mode only allows reading, writing, and erasing plaintext flash and does not support any SRAM or register operations. Ignore this eFuse if Joint Download Boot mode is disabled.

- [EFUSE\\_DIS\\_DIRECT\\_BOOT](#)

If this eFuse is 1, Direct Boot mode is disabled.

### 7.3.3 ROM Code Printing Control

GPIO8 controls ROM code printing during the early SPI boot process. This GPIO is used together with [EFUSE\\_UART\\_PRINT\\_CONTROL](#).

**Table 7.3-3. ROM Code Printing Control**

eFuse <sup>1</sup>	GPIO8	ROM Code Printing Behavior
0	x	ROM code always prints to UART during boot. The value of GPIO8 is ignored.
1	0	Print is enabled during boot.
	1	Print is disabled during boot.
2	0	Print is disabled during boot.
	1	Print is enabled during boot.
3	x	Print is always disabled during boot. The value of GPIO8 is ignored.

<sup>1</sup> eFuse: [EFUSE\\_UART\\_PRINT\\_CONTROL](#)

## Chapter 8

# Interrupt Matrix (INTMTRX)

## 8.1 Overview

The interrupt matrix embedded in ESP8684 independently routes peripheral interrupt sources to the ESP-RISC-V CPU's peripheral interrupts, to timely inform CPU to process the coming interrupts.

The ESP8684 has 43 peripheral interrupt sources. To map them to 31 CPU interrupts, this interrupt matrix is needed.

**Note:**

This chapter focuses on how to map peripheral interrupt sources to CPU interrupts. For more details about interrupt configuration, vector, and ISA suggested operations, please refer to Chapter 1 *ESP-RISC-V CPU*.

## 8.2 Features

Interrupt matrix has the following features:

- 43 peripheral interrupt sources as input
- 31 CPU peripheral interrupts as output
- Able to query current status of peripheral interrupt sources
- Configurable priority, type, threshold, and enable signal of CPU interrupts

Figure 8.2-1 shows the structure of the interrupt matrix.

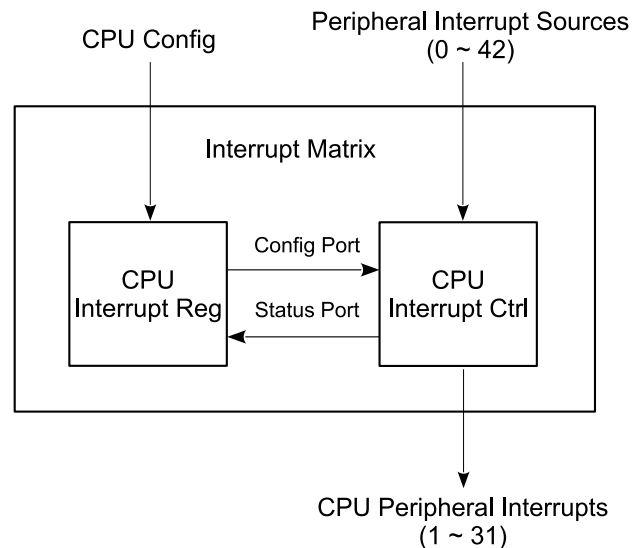


Figure 8.2-1. Interrupt Matrix Structure

## 8.3 Functional Description

### 8.3.1 Peripheral Interrupt Sources

The ESP8684 has 43 peripheral interrupt sources in total. Table 8.3-1 lists all these sources and their configuration/status registers.

- Column “Index”: Peripheral interrupt source index, can be 0 ~ 42.
- Column “Chapter”: In which chapter the interrupt source is described in details.
- Column “Source”: Name of the peripheral interrupt source.
- Column “Configuration Register”: Registers used to configure routing of the peripheral interrupt sources to CPU peripheral interrupts
- Column “Status Register”: Registers used for indicating the interrupt status of peripheral interrupt sources.
  - Column “Status Register - Bit”: Bit position in status register, indicating the interrupt status.
  - Column “Status Register - Name”: Name of status registers.



Table 8.3-1. CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources

Index	Chapter	Source	Configuration Register	Bit	Status Register Name
0	N/A	reserved	reserved	0	INTERRUPT_CORE0_INTR_STATUS_0_REG
1	N/A	reserved	reserved	1	
2	N/A	reserved	reserved	2	
3	N/A	reserved	reserved	3	
4	N/A	reserved	reserved	4	
5	N/A	reserved	reserved	5	
6	N/A	reserved	reserved	6	
7	N/A	reserved	reserved	7	
8	N/A	reserved	reserved	8	
9	N/A	reserved	reserved	9	
10	N/A	reserved	reserved	10	
11	N/A	reserved	reserved	11	
12	N/A	reserved	reserved	12	
13	<a href="#">IO MUX and GPIO Matrix (GPIO, IO MUX)</a>	GPIO_PROCPU_INTR	<a href="#">INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_MAP_REG</a>	13	
14	N/A	reserved	reserved	14	
15	N/A	reserved	reserved	15	
16	<a href="#">SPI Controller (SPI)</a>	GPSPi2_INTR_2	<a href="#">INTERRUPT_CORE0_SPI_INTR_2_MAP_REG</a>	16	
17	<a href="#">UART Controller (UART)</a>	UART_INTR	<a href="#">INTERRUPT_CORE0_UART_INTR_MAP_REG</a>	17	
18	<a href="#">UART Controller (UART)</a>	UART1_INTR	<a href="#">INTERRUPT_CORE0_UART1_INTR_MAP_REG</a>	18	
19	<a href="#">LED PWM Controller (LEDC)</a>	LEDC_INTR	<a href="#">INTERRUPT_CORE0_LEDC_INT_MAP_REG</a>	19	
20	<a href="#">eFuse Controller (eFuse)</a>	EFUSE_INTR	<a href="#">INTERRUPT_CORE0_EFUSE_INT_MAP_REG</a>	20	
21	<a href="#">Low-power Management (RTC_CNTL)</a>	RTC_CNTL_INTR	<a href="#">INTERRUPT_CORE0_RTC_CORE_INTR_MAP_REG</a>	21	
22	<a href="#">I2C Master Controller (I2C)</a>	I2C_EXT0_INTR	<a href="#">INTERRUPT_CORE0_I2C_EXT0_INTR_MAP_REG</a>	22	
23	<a href="#">Timer Group (TIMG)</a>	TG_TO_INTR	<a href="#">INTERRUPT_CORE0_TG_TO_INT_MAP_REG</a>	23	
24	<a href="#">Timer Group (TIMG)</a>	TG_WDT_INTR	<a href="#">INTERRUPT_CORE0_TG_WDT_INT_MAP_REG</a>	24	
25	N/A	reserved	reserved	25	
26	<a href="#">System Timer (SYSTIMER)</a>	SYSTIMER_TARGET0_INTR	<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET0_INT_MAP_REG</a>	26	
27	<a href="#">System Timer (SYSTIMER)</a>	SYSTIMER_TARGET1_INTR	<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET1_INT_MAP_REG</a>	27	
28	<a href="#">System Timer (SYSTIMER)</a>	SYSTIMER_TARGET2_INTR	<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET2_INT_MAP_REG</a>	28	
29	N/A	reserved	reserved	29	
30	N/A	reserved	reserved	30	
31	N/A	reserved	reserved	31	

Index	Chapter	Source	Configuration Register	Status Register	
				Bit	Name
32	<a href="#">On-Chip Sensor and Analog Signal Processing</a>	DIGITAL_ADC_INTR	<a href="#">INTERRUPT_CORE0_APB_ADC_INT_MAP_REG</a>	0	<a href="#">INTERRUPT_CORE0_INTR_STATUS_1_REG</a>
33	<a href="#">GDMA Controller (GDMA)</a>	GDMA_CH0_INTR	<a href="#">INTERRUPT_CORE0_DMA_CH0_INT_MAP_REG</a>	1	
34	<a href="#">SHA Accelerator (SHA)</a>	SHA_INTR	<a href="#">INTERRUPT_CORE0_SHA_INTR_MAP_REG</a>	2	
35	<a href="#">ECC Hardware Accelerator (ECC)</a>	ECC_INTR	<a href="#">INTERRUPT_CORE0_ECC_INTR_MAP_REG</a>	3	
36	<a href="#">System Registers (SYSTEM)</a>	SW_INTR_0	<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG</a>	4	
37	<a href="#">System Registers (SYSTEM)</a>	SW_INTR_1	<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG</a>	5	
38	<a href="#">System Registers (SYSTEM)</a>	SW_INTR_2	<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG</a>	6	
39	<a href="#">System Registers (SYSTEM)</a>	SW_INTR_3	<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG</a>	7	
40	<a href="#">Debug Assistant (ASSIST_DEBUG)</a>	ASSIST_DEBUG_INTR	<a href="#">INTERRUPT_CORE0_ASSIST_DEBUG_INTR_MAP_REG</a>	8	
41	N/A	PERI_VIO_SIZE_INTR	<a href="#">INTERRUPT_CORE0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG</a>	9	
42	N/A	reserved	reserved	10	

### 8.3.2 CPU Interrupts

The ESP8684 implements its interrupt mechanism using an interrupt controller instead of RISC-V Privileged ISA specification. The ESP-RISC-V CPU has 31 interrupts, with unique IDs (1 ~ 31). Each CPU interrupt has the following properties.

- Priority levels from 1 (lowest) to 15 (highest).
- Configurable as high-level triggered or rising-edge triggered.
- Programmable threshold for masking interrupts with lower priority.

**Note:**

For detailed information about how to configure CPU interrupts, see Chapter 1 [ESP-RISC-V CPU](#).

### 8.3.3 Allocate Peripheral Interrupt Source to CPU Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- Source\_*X*: stands for a peripheral interrupt source, wherein *X* means the index of this interrupt source in Table 8.3-1.
- [INTERRUPT\\_CORE0\\_SOURCE\\_X\\_MAP\\_REG](#): stands for a configuration register, mapping Source\_*X* to CPU interrupt.
- Num\_P: the ID of CPU interrupts, can be 1 ~ 31.
- Interrupt\_P: stands for the CPU interrupt with ID = Num\_P.

#### 8.3.3.1 Allocate one peripheral interrupt source (Source\_*X*) to CPU

Setting the corresponding configuration register [INTERRUPT\\_CORE0\\_SOURCE\\_X\\_MAP\\_REG](#) of Source\_*X* to Num\_P allocates this interrupt source to Interrupt\_P.

#### 8.3.3.2 Allocate multiple peripheral interrupt sources (Source\_*Xn*) to CPU

Setting the corresponding configuration register [INTERRUPT\\_CORE0\\_SOURCE\\_Xn\\_MAP\\_REG](#) of each interrupt source to the same Num\_P allocates multiple sources to the same Interrupt\_P. Any of these sources can trigger CPU Interrupt\_P. When an interrupt signal is generated, interrupt service routine (ISR) should check the interrupt status registers to figure out which peripheral generated the interrupt. For more information, see Chapter 1 [ESP-RISC-V CPU](#).

#### 8.3.3.3 Disable CPU peripheral interrupt source (Source\_*X*)

Clearing the configuration register [INTERRUPT\\_CORE0\\_SOURCE\\_X\\_MAP\\_REG](#) disables the corresponding interrupt source.

### 8.3.4 Query Current Interrupt Status of Peripheral Interrupt Source

Users can query current interrupt status of a peripheral interrupt source by reading the bit value in [INTERRUPT\\_CORE0\\_INTR\\_STATUS\\_n\\_REG](#) (read only). For the mapping between

[INTERRUPT\\_CORE0\\_INTR\\_STATUS\\_n\\_REG](#) and peripheral interrupt sources, please refer to Table 8.3-1.

## 8.4 Register Summary

The addresses in this section are relative to the interrupt matrix base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Interrupt source mapping register</b>			
<a href="#">INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_MAP_REG</a>	GPIO_INTERRUPT_PRO mapping register	0x0034	R/W
<a href="#">INTERRUPT_CORE0_SPI_INTR_2_MAP_REG</a>	SPI_INTR_2 mapping register	0x0040	R/W
<a href="#">INTERRUPT_CORE0_UART_INTR_MAP_REG</a>	UART_INTR mapping register	0x0044	R/W
<a href="#">INTERRUPT_CORE0_UART1_INTR_MAP_REG</a>	UART1_INTR mapping register	0x0048	R/W
<a href="#">INTERRUPT_CORE0_LEDC_INT_MAP_REG</a>	LEDC_INT mapping register	0x004C	R/W
<a href="#">INTERRUPT_CORE0_EFUSE_INT_MAP_REG</a>	EFUSE_INT mapping register	0x0050	R/W
<a href="#">INTERRUPT_CORE0_RTC_CORE_INTR_MAP_REG</a>	RTC_CORE_INTR mapping register	0x0054	R/W
<a href="#">INTERRUPT_CORE0_I2C_EXT0_INTR_MAP_REG</a>	I2C_EXT0_INTR mapping register	0x0058	R/W
<a href="#">INTERRUPT_CORE0_TG_TO_INT_MAP_REG</a>	TG_TO_INT mapping register	0x005C	R/W
<a href="#">INTERRUPT_CORE0_TG_WDT_INT_MAP_REG</a>	TG_WDT_INT mapping register	0x0060	R/W
<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET0_INT_MAP_REG</a>	SYSTIMER_TARGET0_INT mapping register	0x0068	R/W
<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET1_INT_MAP_REG</a>	SYSTIMER_TARGET1_INT mapping register	0x006C	R/W
<a href="#">INTERRUPT_CORE0_SYSTIMER_TARGET2_INT_MAP_REG</a>	SYSTIMER_TARGET2_INT mapping register	0x0070	R/W
<a href="#">INTERRUPT_CORE0_APB_ADC_INT_MAP_REG</a>	APB_ADC_INT mapping register	0x0080	R/W
<a href="#">INTERRUPT_CORE0_DMA_CHO_INT_MAP_REG</a>	DMA_CHO_INT mapping register	0x0084	R/W
<a href="#">INTERRUPT_CORE0_SHA_INT_MAP_REG</a>	SHA_INT mapping register	0x0088	R/W
<a href="#">INTERRUPT_CORE0_ECC_INT_MAP_REG</a>	ECC_INT mapping register	0x008C	R/W
<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG</a>	CPU_INTR_FROM_CPU_0 mapping register	0x0090	R/W
<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG</a>	CPU_INTR_FROM_CPU_1 mapping register	0x0094	R/W
<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG</a>	CPU_INTR_FROM_CPU_2 mapping register	0x0098	R/W
<a href="#">INTERRUPT_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG</a>	CPU_INTR_FROM_CPU_3 mapping register	0x009C	R/W
<a href="#">INTERRUPT_CORE0_ASSIST_DEBUG_INTR_MAP_REG</a>	ASSIST_DEBUG_INTR mapping register	0x00A0	R/W
<a href="#">INTERRUPT_CORE0_SIZE_INTR_MAP_REG</a>	PIF_PMS_MONITOR_VIOLATE_SIZE_INTR mapping register	0x00A4	R/W

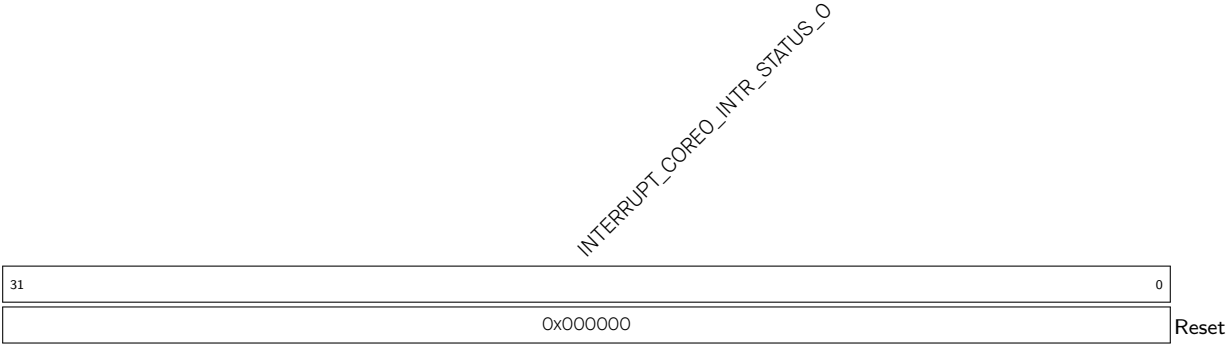
Name	Description	Address	Access
<b>Interrupt source status register</b>			
<a href="#">INTERRUPT_CORE0_INTR_STATUS_0_REG</a>	Interrupt source status register 0	0x00AC	RO
<a href="#">INTERRUPT_CORE0_INTR_STATUS_1_REG</a>	Interrupt source status register 1	0x00B0	RO
<b>Clock register</b>			
<a href="#">INTERRUPT_CORE0_CLOCK_GATE_REG</a>	Clock register	0x00B4	R/W
<b>CPU interrupt register</b>			
<a href="#">INTERRUPT_CORE0_CPU_INT_ENABLE_REG</a>	Enable register for CPU interrupts	0x00B8	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_TYPE_REG</a>	Type configuration register for CPU interrupts	0x00BC	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_CLEAR_REG</a>	CPU interrupt clear register	0x00C0	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG</a>	Pending status register for CPU interrupts	0x00C4	RO
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_1_REG</a>	Priority configuration register for CPU interrupt 1	0x00CC	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_2_REG</a>	Priority configuration register for CPU interrupt 2	0x00D0	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_3_REG</a>	Priority configuration register for CPU interrupt 3	0x00D4	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_4_REG</a>	Priority configuration register for CPU interrupt 4	0x00D8	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_5_REG</a>	Priority configuration register for CPU interrupt 5	0x00DC	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_6_REG</a>	Priority configuration register for CPU interrupt 6	0x00E0	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_7_REG</a>	Priority configuration register for CPU interrupt 7	0x00E4	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_8_REG</a>	Priority configuration register for CPU interrupt 8	0x00E8	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_9_REG</a>	Priority configuration register for CPU interrupt 9	0x00EC	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_10_REG</a>	Priority configuration register for CPU interrupt 10	0x00F0	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_11_REG</a>	Priority configuration register for CPU interrupt 11	0x00F4	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_12_REG</a>	Priority configuration register for CPU interrupt 12	0x00F8	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_13_REG</a>	Priority configuration register for CPU interrupt 13	0x00FC	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_14_REG</a>	Priority configuration register for CPU interrupt 14	0x0100	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_15_REG</a>	Priority configuration register for CPU interrupt 15	0x0104	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_16_REG</a>	Priority configuration register for CPU interrupt 16	0x0108	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_17_REG</a>	Priority configuration register for CPU interrupt 17	0x010C	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_18_REG</a>	Priority configuration register for CPU interrupt 18	0x0110	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_19_REG</a>	Priority configuration register for CPU interrupt 19	0x0114	R/W

Name	Description	Address	Access
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_20_REG</a>	Priority configuration register for CPU interrupt 20	0x0118	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_21_REG</a>	Priority configuration register for CPU interrupt 21	0x011C	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_22_REG</a>	Priority configuration register for CPU interrupt 22	0x0120	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_23_REG</a>	Priority configuration register for CPU interrupt 23	0x0124	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_24_REG</a>	Priority configuration register for CPU interrupt 24	0x0128	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_25_REG</a>	Priority configuration register for CPU interrupt 25	0x012C	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_26_REG</a>	Priority configuration register for CPU interrupt 26	0x0130	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_27_REG</a>	Priority configuration register for CPU interrupt 27	0x0134	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_28_REG</a>	Priority configuration register for CPU interrupt 28	0x0138	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_29_REG</a>	Priority configuration register for CPU interrupt 29	0x013C	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_30_REG</a>	Priority configuration register for CPU interrupt 30	0x0140	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_PRI_31_REG</a>	Priority configuration register for CPU interrupt 31	0x0144	R/W
<a href="#">INTERRUPT_CORE0_CPU_INT_THRESH_REG</a>	Threshold configuration register for CPU interrupts	0x0148	R/W
<b>Version register</b>			
<a href="#">INTERRUPT_CORE0_INTERRUPT_DATE_REG</a>	Version control register	0x07FC	R/W



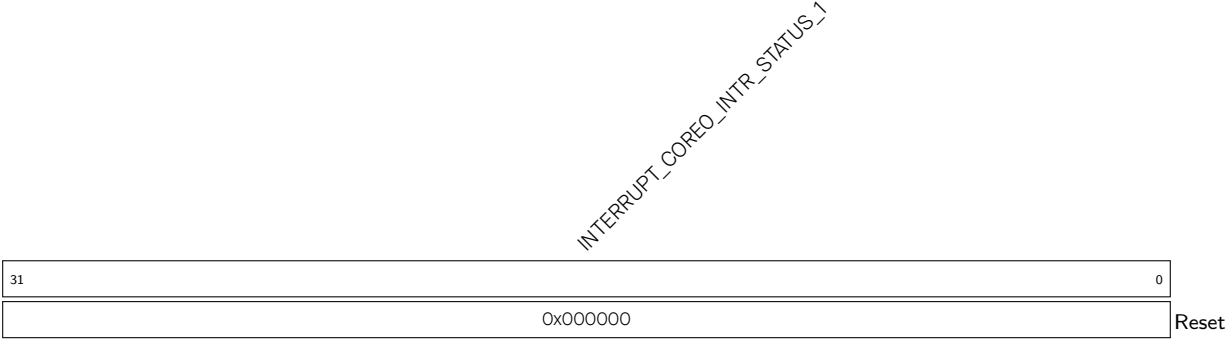


Register 8.24. INTERRUPT\_CORE0\_INTR\_STATUS\_0\_REG (0x00AC)



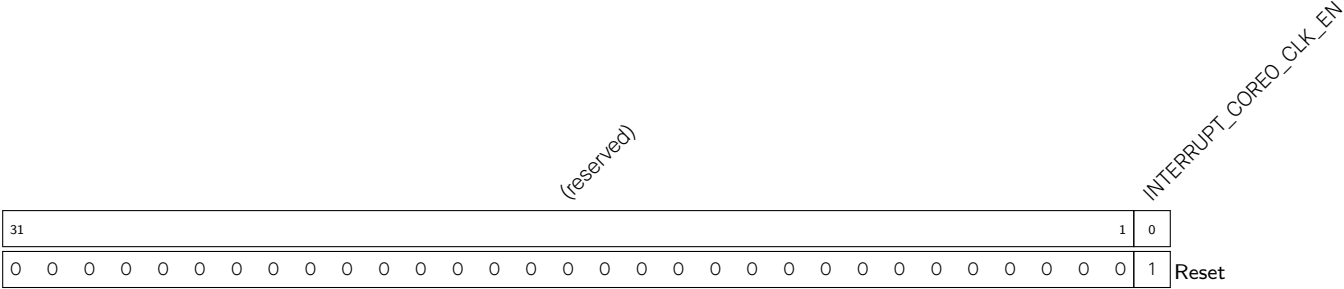
**INTERRUPT\_CORE0\_INTR\_STATUS\_0** This register stores the status of the first 32 interrupt sources: 0 ~ 31. If the bit is 1 here, it means the corresponding source triggered an interrupt. (RO)

Register 8.25. INTERRUPT\_CORE0\_INTR\_STATUS\_1\_REG (0x00B0)

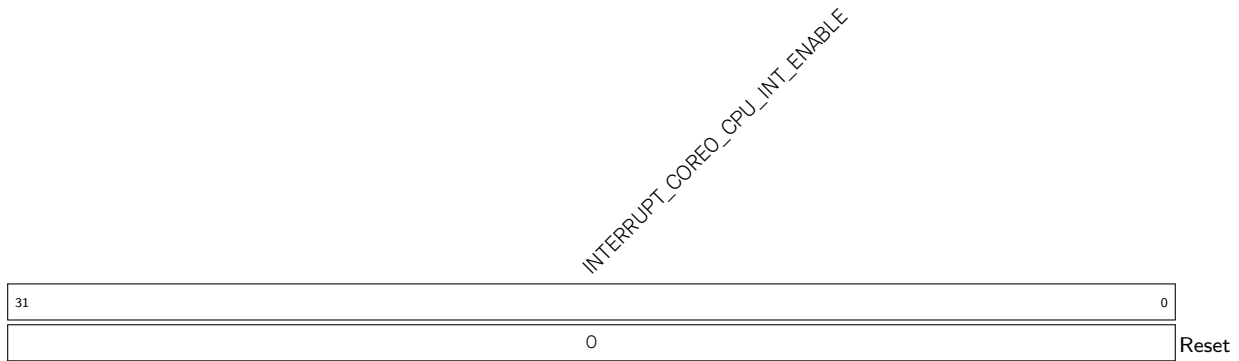


**INTERRUPT\_CORE0\_INTR\_STATUS\_1** This register stores the status of interrupt sources: 32 ~ 42. If the bit is 1 here, it means the corresponding source triggered an interrupt. (RO)

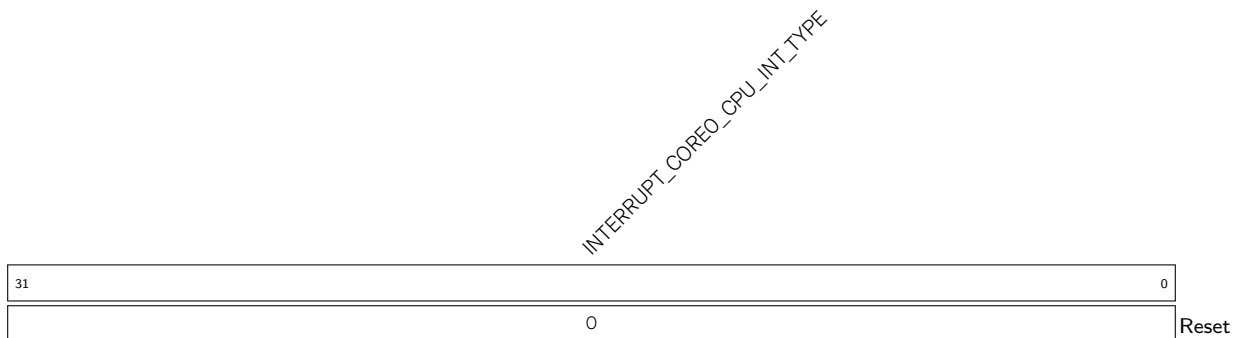
Register 8.26. INTERRUPT\_CORE0\_CLOCK\_GATE\_REG (0x00B4)



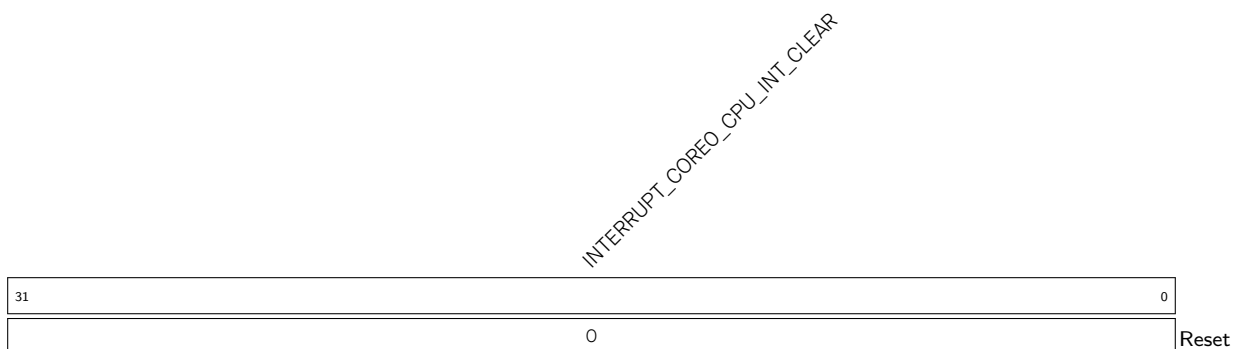
**INTERRUPT\_CORE0\_CLK\_EN** Set 1 to force interrupt register clock-gate on. (R/W)

**Register 8.27. INTERRUPT\_CORE0\_CPU\_INT\_ENABLE\_REG (0x00B8)**

**INTERRUPT\_CORE0\_CPU\_INT\_ENABLE** Writing 1 to the bit here enables its corresponding CPU interrupt. For more information about how to use this register, see Chapter 1 [ESP-RISC-V CPU](#). (R/W)

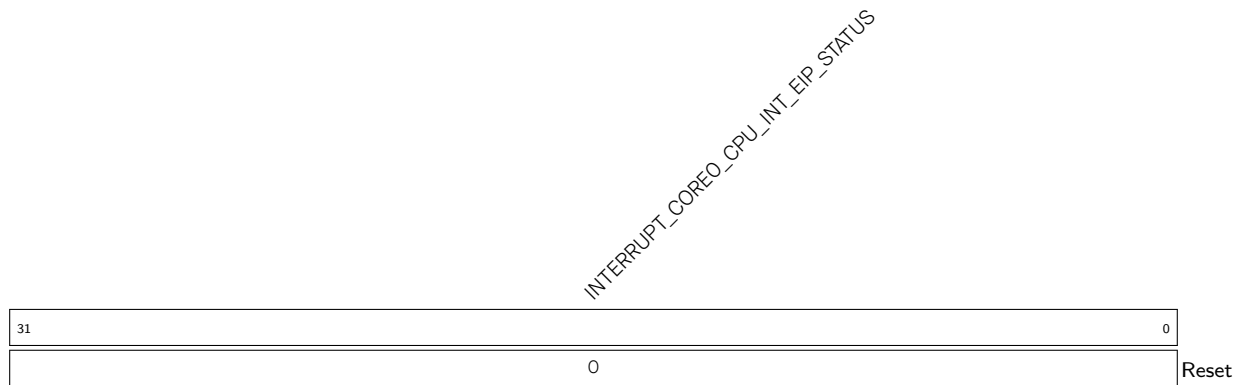
**Register 8.28. INTERRUPT\_CORE0\_CPU\_INT\_TYPE\_REG (0x00BC)**

**INTERRUPT\_CORE0\_CPU\_INT\_TYPE** Configure CPU interrupt type. 0: level-triggered; 1: edge-triggered. For more information about how to use this register, see Chapter 1 [ESP-RISC-V CPU](#). (R/W)

**Register 8.29. INTERRUPT\_CORE0\_CPU\_INT\_CLEAR\_REG (0x00C0)**

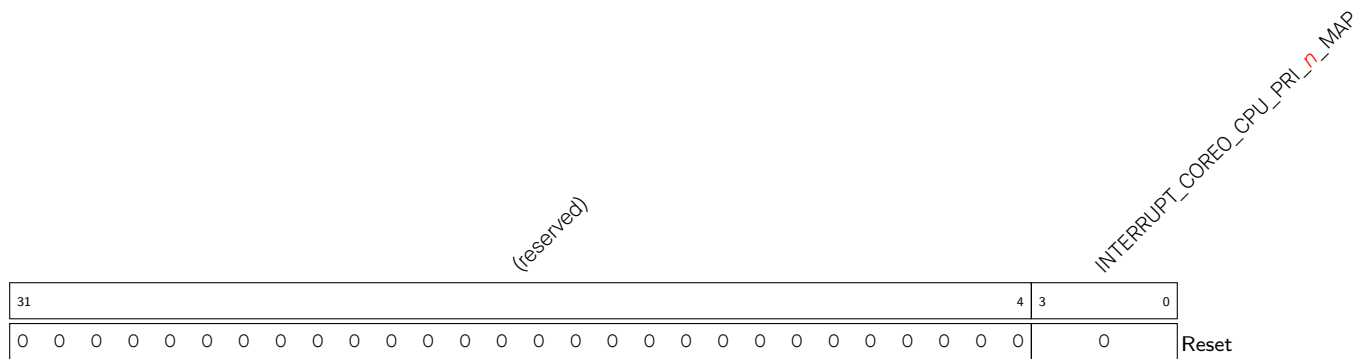
**INTERRUPT\_CORE0\_CPU\_INT\_CLEAR** Writing 1 to the bit here clears its corresponding CPU interrupt. For more information about how to use this register, see Chapter 1 [ESP-RISC-V CPU](#). (R/W)

### Register 8.30. INTERRUPT\_CORE0\_CPU\_INT\_EIP\_STATUS\_REG (0x00C4)



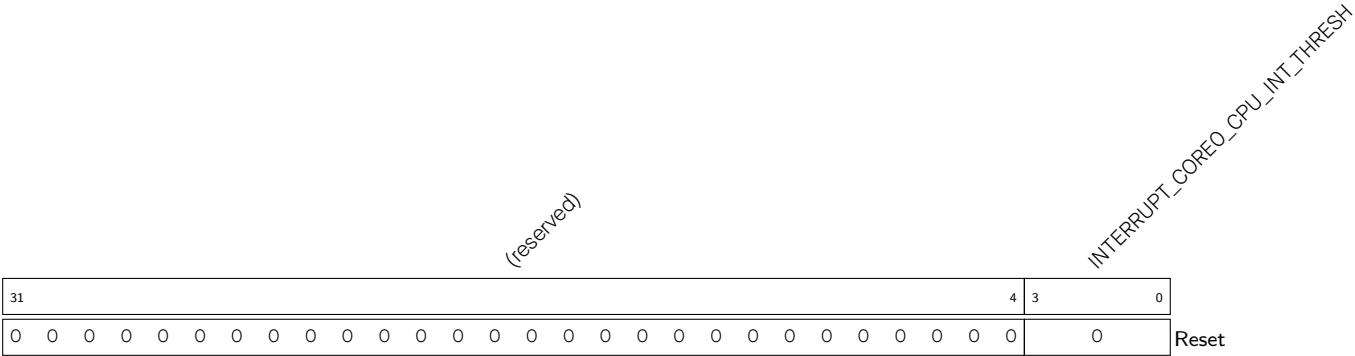
**INTERRUPT\_CORE0\_CPU\_INT\_EIP\_STATUS** Store the pending status of CPU interrupts. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (RO)

**Register 8.31. INTERRUPT\_CORE0\_CPU\_INT\_PRI\_n\_REG (n: 1 - 31)(0x00C8 + 0x4\*n)**



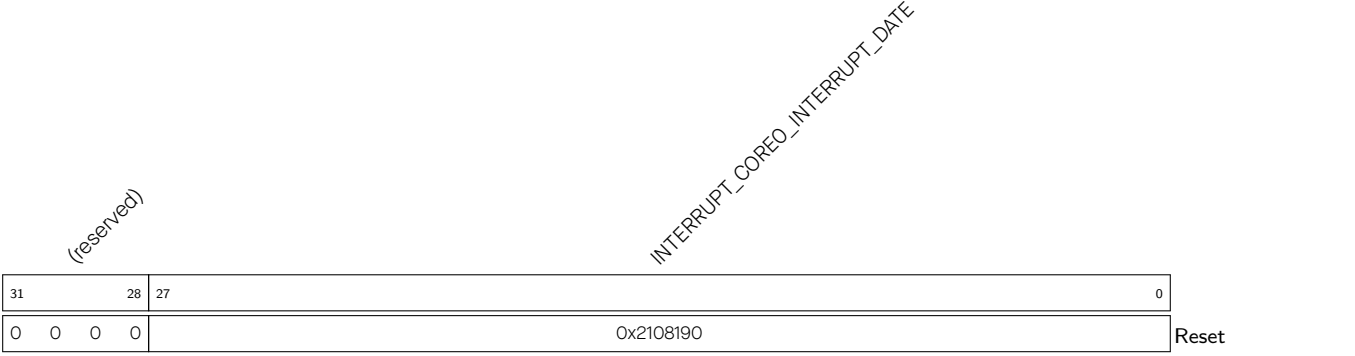
**INTERRUPT\_CORE0\_CPU\_PRI\_***n***\_MAP** Set the priority for CPU interrupt *n*. The priority here can be 1 (lowest) ~ 15 (highest). For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.32. INTERRUPT\_CORE0\_CPU\_INT\_THRESH\_REG (0x0148)



**INTERRUPT\_CORE0\_CPU\_INT\_THRESH** Set threshold for interrupt assertion to CPU. Only when the interrupt priority is equal to or higher than this threshold, CPU will respond to this interrupt. For more information about how to use this register, see Chapter 1 [ESP-RISC-V CPU](#). (R/W)

Register 8.33. INTERRUPT\_CORE0\_INTERRUPT\_DATE\_REG (0x07FC)



**INTERRUPT\_CORE0\_INTERRUPT\_DATE** Version control register. (R/W)

## Chapter 9

# Low-power Management (RTC\_CNTL)

## 9.1 Introduction

ESP8684 has an advanced Power Management Unit (PMU), which can flexibly power up different power domains of the chip, to achieve the best balance among chip performance, power consumption, and wakeup latency. To simplify power management for typical scenarios, ESP8684 has predefined four power modes, which are preset configurations that power up different combinations of power domains. On top of that, the chip also allows the users to independently power up any particular power domain to meet more complex requirements.

## 9.2 Features

ESP8684's low-power management supports the following features:

- 4 x predefined power modes to simplify power management for typical scenarios
- 8 x 32-bit retention registers

In this chapter, we first introduce the working process of ESP8684's low-power management, then introduce the predefined power modes of the chip.

## 9.3 Functional Description

ESP8684's low-power management involves the following components:

- Power management unit: controls the power supply to three power domain categories:
  - Real Time Controller (RTC)
  - Digital
  - Analog

For a complete list of 6 power domains grouped in these three power domain categories, see [Section 9.5.1](#).

- Power isolation unit: isolates different power domains, so any powered down power domain does not affect the powered up ones.
- Low-power clocks: provide clocks to power domains working in low-power modes.
- RTC timer: logs the status of the RTC main state machine in dedicated registers.

- 8 x 32-bit “always-on” retention registers: These registers are always powered up and are not affected by any low-power modes, thus can be used for storing data that cannot be lost.
- 6 x “always-on” pins: These pins are always powered up and are not affected by any low-power modes, which makes them suitable for working as wakeup sources when the chip is working in the low-power modes (for details, please refer to Section 9.5.3), or can be used as regular GPIOs (for details, please refer to Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)*).
- Voltage regulators: regulate the power supply to different power domains.

The schematic diagram of ESP8684's low-power management is shown in Figure 9.3-1.

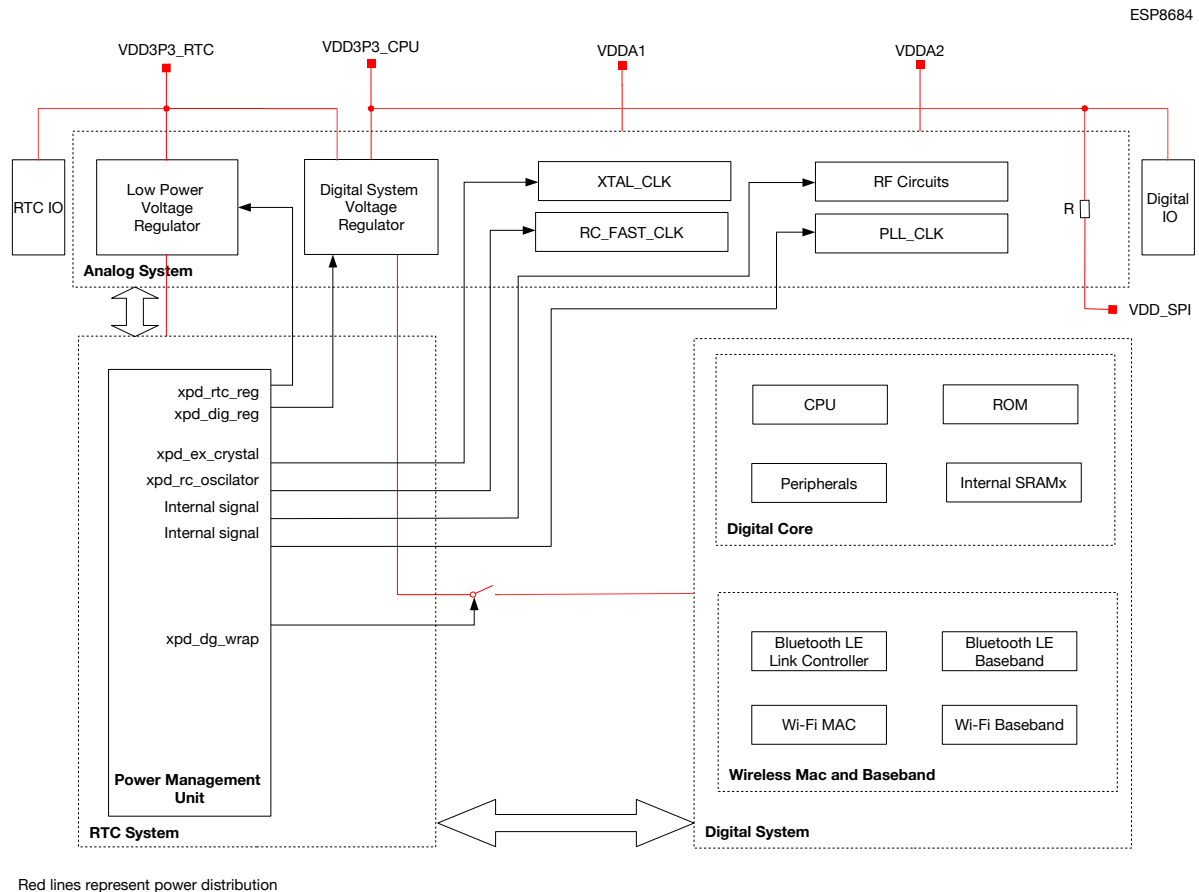


Figure 9.3-1. Low-power Management Schematics

**Note:**

- Power domains are enclosed with dashed lines. For more information about different power domains, please check Section 9.5.1.
- Switches in the above diagram can be controlled by Register [RTC\\_CNTL\\_DIG\\_PWC\\_REG](#).
- Signals in the above diagram are described below:
  - xpd\_rtc\_reg:
    - \* When [RTC\\_CNTL\\_REGULATOR\\_FORCE\\_PU](#) is set to 1, low power voltage regulator is always-on;
    - \* Otherwise, the low power voltage regulator is off when chip enters sleep.
  - xpd\_dig\_reg:
    - \* When [RTC\\_CNTL\\_DG\\_WRAP\\_PD\\_EN](#) is enabled, the digital system voltage regulator is off when the

- chip enters sleep;
- \* Otherwise, the digital system voltage regulator is always-on.
- xpd\_ex\_crystal:
  - \* When [RTC\\_CNTL\\_XTL\\_FORCE\\_PU](#) is set to 1, the external main crystal clock is always-on;
  - \* Otherwise, the external main crystal clock is off when chip enters sleep.
- xpd\_rc\_oscillator:
  - \* when [RTC\\_CNTL\\_FOSC\\_FORCE\\_PU](#) is set to 1, the fast RC oscillator is always-on;
  - \* Otherwise, the fast RC oscillator is off when chip enters sleep.

### 9.3.1 Power Management Unit (PMU)

ESP8684's power management unit controls the power supply to different power domains. The main components of the power management unit include:

- RTC main state machine: generates power gating, clock gating, and reset signals.
- Power controllers: power up and power down different power domains, according to the power gating signals from the main state machine.
- Sleep / wakeup controllers: send sleep or wakeup requests to the RTC main state machine.
- Clock controller: selects and powers up/down clock sources.
- Protection Timer: controls the transition interval between main state machine states.

In ESP8684's power management unit, the sleep / wakeup controllers send sleep or wakeup requests to the RTC main state machine, which then generates power gating, clock gating, and reset signals. Then, the power controller and clock controller power up and power down different power domains and clock sources, according to the signals generated by the RTC main state machine, so that the chip enters or exits the low-power modes. The main workflow is shown in [Figure 9.3-2](#).

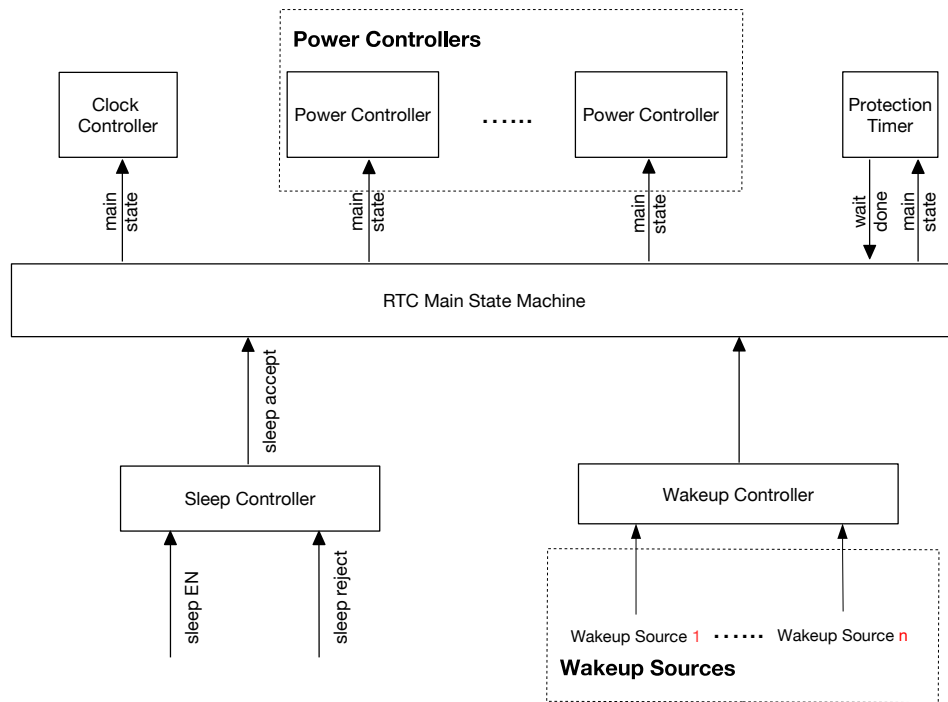


Figure 9.3-2. Power Management Unit Workflow

**Note:**

1. Each power domain has its own power controller. For a complete list of all the available power controllers controlling different power domains, please refer to Section 9.5.1.
2. For a complete list of all the available wakeup sources, please refer to Table 9.5-2.

### 9.3.2 Low-Power Clocks

In general, ESP8684 powers down its External Main Crystal Clock (XTAL\_CLK) and PLL Clock (PLL\_CLK) to reduce power consumption when working in low-power modes. During this time, the chip's low-power clocks remain on to provide clocks to low power domains, such as the power management unit.



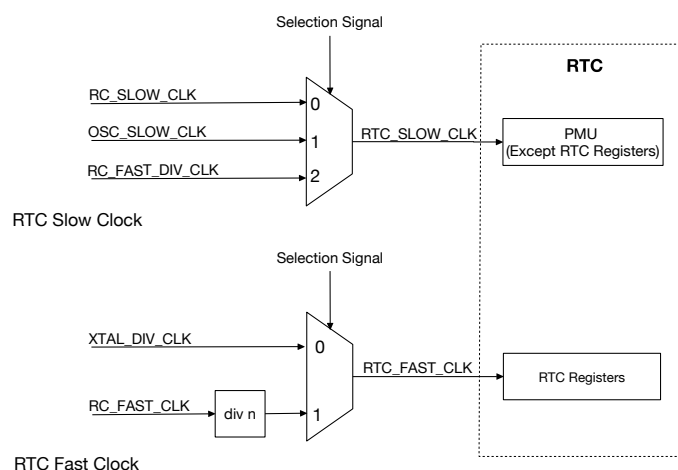


Figure 9.3-3. RTC\_SLOW\_CLK and RTC\_FAST\_CLK

Table 9.3-1. Low-power Clocks

Clock Type	Clock Source	Selection Signal	Power Domain
RTC_SLOW_CLK	OSC_SLOW_CLK	<a href="#">RTC_CNTL_ANA_CLK_RTC_SEL</a>	Power Management System (except RTC registers)
	RC_FAST_DIV_CLK		
	RC_SLOW_CLK (default)		
RTC_FAST_CLK	RC_FAST_CLK divided by n (default)	<a href="#">RTC_CNTL_FAST_CLK_RTC_SEL</a>	RTC Registers
	XTAL_DIV_CLK		

For more detailed description about clocks, please refer to [6 Reset and Clock](#).

### 9.3.3 Timers

ESP8684's low-power management uses RTC timer. The readable 48-bit RTC timer is a real-time counter (using RTC slow clock) that can be configured to log the time when one of the following events happens. For details, see [Table 9.3-2](#).

Table 9.3-2. The Triggering Conditions for the RTC Timer

Enabling Options	Descriptions
<a href="#">RTC_CNTL_TIMER_XTL_OFF</a>	RTC main state machine powers down or XTAL_CLK powers up.
<a href="#">RTC_CNTL_TIMER_SYS_STALL</a>	CPU enters or exits the stall state. This is to ensure the SYS_TIMER is continuous in time.
<a href="#">RTC_CNTL_TIMER_SYS_RST</a>	Resetting digital system completes.
<a href="#">RTC_CNTL_TIME_UPDATE</a>	Register <a href="#">RTC_CNTL_TIME_UPDATE</a> is configured by CPU (i.e. users).

The RTC timer updates two groups of registers upon any new trigger. The first group logs the time of the current trigger, and the other logs the previous trigger. Detailed information about these two register groups is shown below:

- Register group 0: logs the status of RTC timer at the current trigger.
  - [RTC\\_CNTL\\_TIME\\_HIGH0\\_REG](#)
  - [RTC\\_CNTL\\_TIME\\_LOW0\\_REG](#)
- Register group 1: logs the status of RTC timer at the previous trigger.
  - [RTC\\_CNTL\\_TIME\\_HIGH1\\_REG](#)
  - [RTC\\_CNTL\\_TIME\\_LOW1\\_REG](#)

On a new trigger, information on previous trigger is moved from register group 0 to register group 1 (and the original trigger logged in register group 1 is overwritten), and this new trigger is logged in register group 0. Therefore, only the last two triggers can be logged at any time.

It should be noted that any reset / sleep other than power-up reset will not stop or reset the RTC timer.

Also, the RTC timer can be used as a wakeup source. For details, see Section [9.5.3](#).

### 9.3.4 Voltage Regulators

ESP8684 has two regulators to maintain a constant power supply voltage to different power domains:

- Digital system voltage regulator for digital power domains;
- Low-power voltage regulator for RTC power domains.

**Note:**

For more detailed description about power domains, please refer to Section [9.5.1](#).

#### 9.3.4.1 Digital System Voltage Regulator

ESP8684's built-in digital system voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for digital power domains. This regulator is controlled by the [xpd\\_dig\\_reg](#) signal. For details, see description of Figure [9.3-1](#). For the architecture of the ESP8684 digital system voltage regulator, see Figure [9.3-4](#).

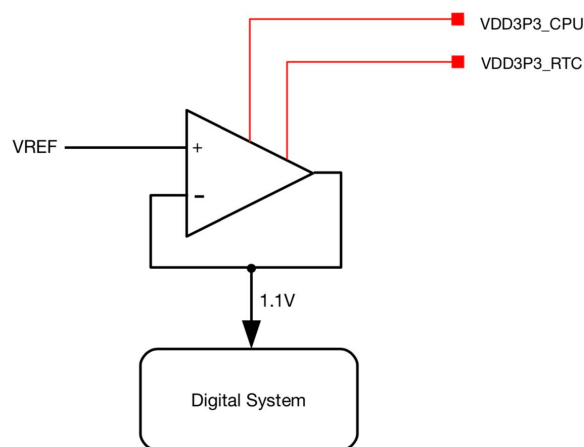


Figure 9.3-4. Digital System Regulator

### 9.3.4.2 Low-power Voltage Regulator

ESP8684's built-in low-power voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for RTC power domains. Note when the pin `CHIP_EN` is at a high level, the low-power voltage regulator cannot be turned off, but only switching between normal-work mode and Deep-sleep mode.

For the architecture of the ESP8684 low-power voltage regulator, see Figure 9.3-5.

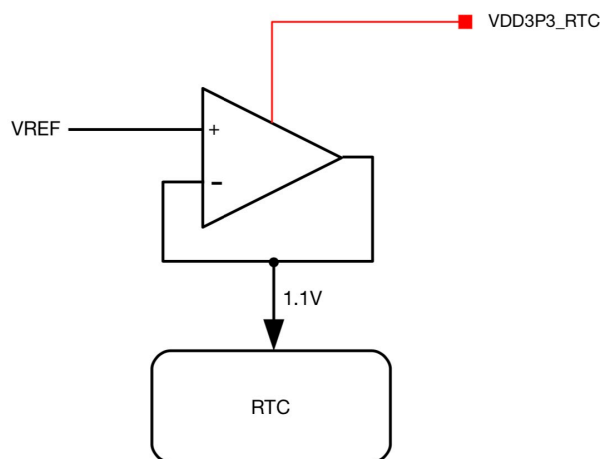


Figure 9.3-5. Low-power voltage regulator

## 9.4 Brownout Detector

The brownout detector checks the voltage of pins `VDDA`, `VDDA3P3`, `VDD3P3_RTC` and `VDD3P3_CPU`. If the voltage of these pins drops below the predefined threshold (2.7 V by default), 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 system to save and transfer important data.

The brownout detector has ultra-low power consumption and remains enabled whenever the chip is powered up. For the architecture of the ESP8684 brownout detector, see Figure 9.4-1.

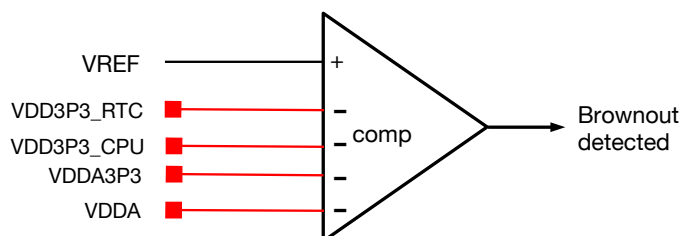


Figure 9.4-1. Brownout detector

`RTC_CNTL_BROWN_OUT_DET` indicates the output level of brownout detector. This register is low level by default, and outputs high level when the voltage on any of monitored pins drops below the predefined threshold.

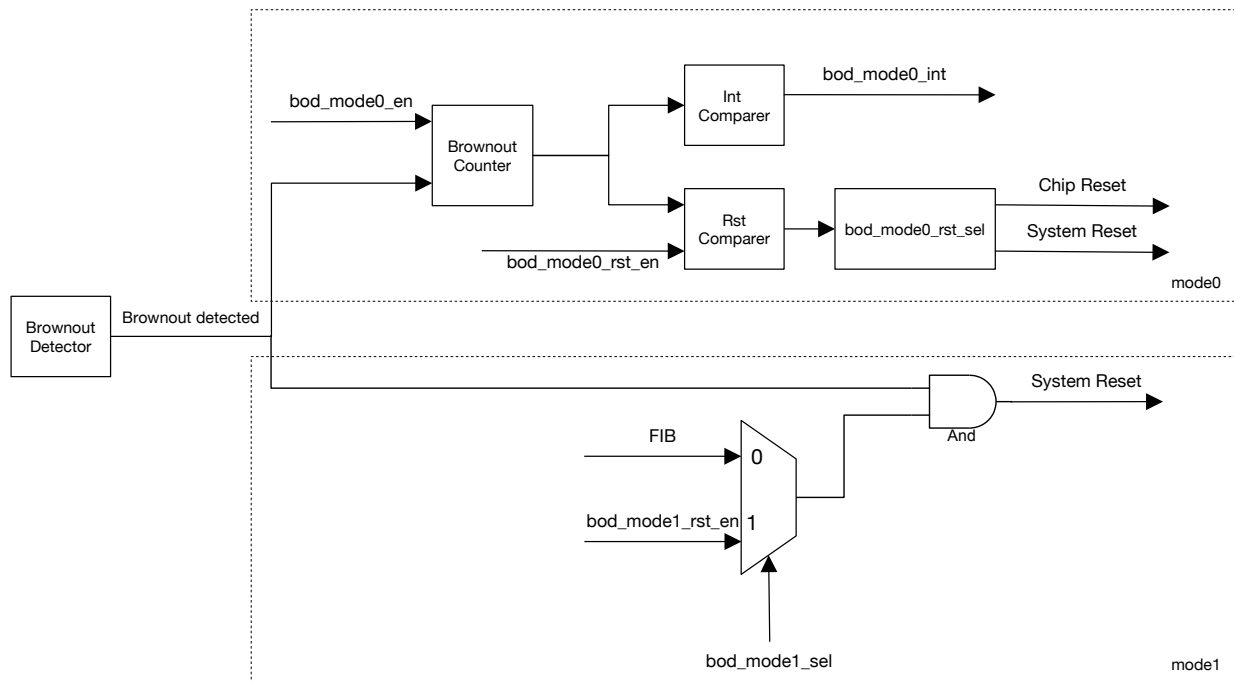


Figure 9.4-2. Brownout handling

As shown in the Figure 9.4-2, the brownout detector can handle the detected brownout signal in one of these two methods described below according to user configuration:

- mode0: triggers an interrupt when the brownout counter counts to the threshold pre-defined in the interrupt comparator (configured with [RTC\\_CNTL\\_BROWN\\_OUT\\_INT\\_WAIT](#))
  - Additionally, when `bod_mode0_rst_en` ([RTC\\_CNTL\\_BROWN\\_OUT\\_RST\\_ENA](#)) is enabled, the brownout detector also resets the chip when the brownout counter counts to the threshold pre-defined in the reset comparator (configured with [RTC\\_CNTL\\_BROWN\\_OUT\\_RST\\_WAIT](#)) based on the `rst_sel` (configured with [RTC\\_CNTL\\_BROWN\\_OUT\\_RST\\_SEL](#)):
    - \* 0: resets the chip
    - \* 1: resets the system

For more information regarding chip reset and system reset, please refer to Chapter 6 [Reset and Clock](#).

- mode1: resets the system directly.

To choose how the brownout detector handles the detected brownout signal:

- mode0: set the `bod_mode0_en` signal (configured with [RTC\\_CNTL\\_BROWN\\_OUT\\_ENA](#)).
- mode1: `bod_mode1_sel`
  - 0: set the `bod_mode1_rst_en` signal (configured with [RTC\\_CNTL\\_BROWN\\_OUT\\_ANA\\_RST\\_EN](#))
  - 1: FIB bus
- Note that mode1 prevails mode0 when both options are enabled at the same time

## 9.5 Power Modes Management

### 9.5.1 Power Domains

ESP8684 has 6 power domains in three power domain categories:

- RTC
  - Power management unit (PMU), including RTC timer and always-on registers
- Digital
  - Digital including digital core and Wireless digital circuit
- Analog
  - RC\_FAST\_CLK
  - XTAL\_CLK
  - PLL\_CLK
  - RF Circuits

### 9.5.2 Pre-defined Power Modes

As mentioned earlier, ESP8684 has four power modes, which are predefined configurations that power up different combinations of power domains. For details, please refer to Table 9.5-1.

**Table 9.5-1. Predefined Power Modes**

Power Mode	Power Domain					
	PMU	Digital	RC_FAST_CLK	XTAL_CLK	PLL_CLK	RF Circuits
Active	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	OFF
Light-sleep	ON	ON	OFF	OFF	OFF	OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF

By default, ESP8684 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 different low-power modes (including Modem-sleep, Light-sleep, and Deep-sleep) to save power. From Active to Deep-sleep, the number of available functionalities<sup>1</sup> and power consumption<sup>2</sup> decreases and wakeup latency increases. Also, the supported wakeup sources for different power modes are different<sup>3</sup>. Users can choose a power mode based on their requirements of functionality, power consumption, wakeup latency, and available wakeup sources.

**Note:**

1. For details, please refer to Table 9.5-1.
2. For details on power consumption, please refer to the Current Consumption Characteristics in [ESP8684 Datasheet](#).
3. For details on the supported wakeup sources, please refer to Section 9.5.3.

### 9.5.3 Wakeup Sources

The ESP8684 supports various wakeup sources, which could wake up the CPU in different sleep modes. The wakeup source is determined by `RTC_CNTL_WAKEUP_ENA` as shown in Table 9.5-2.

Table 9.5-2. Wakeup Source

WAKEUP_ENA	Wakeup Source	Light-sleep	Deep-sleep
0x4	GPIO <sup>1</sup>	Y	Y
0x8	RTC Timer	Y	Y
0x20	Wi-Fi <sup>2</sup>	Y	-
0x40	UART0 <sup>3</sup>	Y	-
0x80	UART1 <sup>3</sup>	Y	-
0x400	Bluetooth	Y	-

<sup>1</sup> In Deep-sleep mode, only the RTC GPIOs (not regular GPIOs) can work as a wakeup source.

<sup>2</sup> To wake up the chip with a Wi-Fi source, the chip switches between the Active, Modem-sleep, and Light-sleep modes. The CPU and RF modules are woken up at predetermined intervals to keep Wi-Fi connections active.

<sup>3</sup> A wakeup is triggered when the number of RX pulses received exceeds the setting in the threshold register `UART_SLEEP_CONF_REG`. For details, please refer to Chapter 19 *UART Controller (UART)*.

### 9.5.4 Reject Sleep

ESP8684 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.

Table 9.5-3. Reject Source

REJECT_ENA	Reject Source
0x4	GPIO
0x8	RTC Timer
0x20	Wi-Fi
0x400	Bluetooth

Users can configure the reject to sleep option according to Table 9.5-3 via the following registers.

- Configure the [RTC\\_CNTL\\_SLEEP\\_REJECT\\_ENA](#) 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\\_SLP\\_REJECT\\_CAUSE\\_REG](#) to check the reason for rejecting to sleep.

## 9.6 Register Summary

The addresses in this section are relative to low-power management base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Control / Configuration Registers</b>			
<a href="#">RTC_CNTL_OPTIONS0_REG</a>	Configures the power options of crystal and PLL clocks, and initiates reset by software	0x0000	varies
<a href="#">RTC_CNTL_SLP_TIMER0_REG</a>	RTC timer threshold register 0	0x0004	R/W
<a href="#">RTC_CNTL_SLP_TIMER1_REG</a>	RTC timer threshold register 1	0x0008	R/W
<a href="#">RTC_CNTL_TIME_UPDATE_REG</a>	RTC timer update control register	0x000C	R/W
<a href="#">RTC_CNTL_TIME_LOW0_REG</a>	Represents the lower 32 bits of RTC timer 0	0x0010	R/W
<a href="#">RTC_CNTL_TIME_HIGH0_REG</a>	Represents the higher 16 bits of RTC timer 0	0x0014	R/W
<a href="#">RTC_CNTL_STATE0_REG</a>	Configures the sleep / reject / wakeup state	0x0018	R/W
<a href="#">RTC_CNTL_TIMER1_REG</a>	Configures CPU stall options	0x001C	R/W
<a href="#">RTC_CNTL_TIMER2_REG</a>	Configures RTC_SLOW_CLK and touch controller	0x0020	R/W
<a href="#">RTC_CNTL_ANA_CONF_REG</a>	Configures the power options for I2C and PLLA	0x002C	R/W
<a href="#">RTC_CNTL_WAKEUP_STATE_REG</a>	Wakeup bitmap enabling register	0x0034	R/W
<a href="#">RTC_CNTL_STORE0_REG</a>	Reservation register 0	0x0048	R/W
<a href="#">RTC_CNTL_STORE1_REG</a>	Reservation register 1	0x004C	R/W
<a href="#">RTC_CNTL_STORE2_REG</a>	Reservation register 2	0x0050	R/W
<a href="#">RTC_CNTL_STORE3_REG</a>	Reservation register 3	0x0054	R/W
<a href="#">RTC_CNTL_EXT_WAKEUP_CONF_REG</a>	GPIO wakeup configuration register	0x005C	R/W
<a href="#">RTC_CNTL_SLP_REJECT_CONF_REG</a>	Configures sleep / reject options	0x0060	R/W
<a href="#">RTC_CNTL_CLK_CONF_REG</a>	RTC timer configuration register	0x0068	R/W
<a href="#">RTC_CNTL_REG</a>	RTC configuration register	0x0074	R/W
<a href="#">RTC_CNTL_PWC_REG</a>	RTC power configuration register	0x0078	R/W
<a href="#">RTC_CNTL_DIG_PWC_REG</a>	Digital system power configuration register	0x007C	R/W
<a href="#">RTC_CNTL_DIG_ISO_REG</a>	Digital system isolation configuration register	0x0080	R/W
<a href="#">RTC_CNTL_WDTCONFIG0_REG</a>	RTC watchdog configuration register	0x0084	R/W
<a href="#">RTC_CNTL_WDTCONFIG1_REG</a>	Configures the hold time of RTC watchdog in stage 0	0x0088	R/W
<a href="#">RTC_CNTL_WDTCONFIG2_REG</a>	Configures the hold time of RTC watchdog in stage 1	0x008C	R/W



Name	Description	Address	Access
<a href="#">RTC_CNTL_WDTCONFIG3_REG</a>	Configures the hold time of RTC watchdog in stage 2	0x0090	R/W
<a href="#">RTC_CNTL_WDTCONFIG4_REG</a>	Configures the hold time of RTC watchdog in stage 3	0x0094	R/W
<a href="#">RTC_CNTL_WDTFEED_REG</a>	RTC watchdog SW feed configuration register	0x0098	R/W
<a href="#">RTC_CNTL_WDTWPROTECT_REG</a>	RTC watchdog write protection configuration register	0x009C	R/W
<a href="#">RTC_CNTL_SWD_CONF_REG</a>	Super watchdog configuration register	0x00A0	R/W
<a href="#">RTC_CNTL_SWD_WPROTECT_REG</a>	Super watchdog write protection configuration register	0x00A4	R/W
<a href="#">RTC_CNTL_SW_CPU_STALL_REG</a>	CPU stall configuration register	0x00A8	R/W
<a href="#">RTC_CNTL_STORE4_REG</a>	Reservation register 4	0x00AC	R/W
<a href="#">RTC_CNTL_STORE5_REG</a>	Reservation register 5	0x00B0	R/W
<a href="#">RTC_CNTL_STORE6_REG</a>	Reservation register 6	0x00B4	R/W
<a href="#">RTC_CNTL_STORE7_REG</a>	Reservation register 7	0x00B8	R/W
<a href="#">RTC_CNTL_PAD_HOLD_REG</a>	Configures the hold options for RTC GPIOs	0x00C4	R/W
<a href="#">RTC_CNTL_DIG_PAD_HOLD_REG</a>	Configures the hold options for digital GPIOs	0x00C8	R/W
<a href="#">RTC_CNTL_BROWN_OUT_REG</a>	Brownout configuration register	0x00CC	R/W
<a href="#">RTC_CNTL_TIME_LOW1_REG</a>	Represents the lower 32 bits of RTC timer 1	0x00D0	R/W
<a href="#">RTC_CNTL_TIME_HIGH1_REG</a>	Represents the higher 16 bits of RTC timer 1	0x00D4	R/W
<a href="#">RTC_CNTL_USB_CONF_REG</a>	IO_MUX configuration register	0x00D8	R/W
<a href="#">RTC_CNTL_SLP_REJECT_CAUSE_REG</a>	Represents the reject-to-sleep cause	0x00DC	R/W
<a href="#">RTC_CNTL_OPTION1_REG</a>	RTC option register	0x00E0	R/W
<a href="#">RTC_CNTL_SLP_WAKEUP_CAUSE_REG</a>	Represents the sleep-to-wakeup cause.	0x00E4	R/W
<a href="#">RTC_CNTL_CNTL_GPIO_WAKEUP_REG</a>	GPIO wakeup configuration register	0x00FC	R/W
<a href="#">RTC_CNTL_CNTL_SENSOR_CTRL_REG</a>	SAR ADC control register	0x0108	R/W
<a href="#">RTC_CNTL_FIB_SEL_REG</a>	Brownout detector configuration register	0x00F8	R/W
<b>Status Registers</b>			
<a href="#">RTC_CNTL_RESET_STATE_REG</a>	Represents the CPU reset source	0x0030	R/W
<a href="#">RTC_CNTL_LOW_POWER_ST_REG</a>	Represents the RTC state	0x00BC	R/W
<b>Interrupt Registers</b>			
<a href="#">RTC_CNTL_INT_ENA_RTC_REG</a>	RTC interrupt enabling register	0x0038	R/W
<a href="#">RTC_CNTL_INT_RAW_RTC_REG</a>	RTC interrupt raw register	0x003C	R/W
<a href="#">RTC_CNTL_INT_ST_RTC_REG</a>	RTC interrupt state register	0x0040	R/W
<a href="#">RTC_CNTL_INT_CLR_RTC_REG</a>	RTC interrupt clear register	0x0044	R/W
<a href="#">RTC_CNTL_INT_ENA_RTC_W1TS_REG</a>	RTC interrupt enabling register (W1TS)	0x00EC	R/W
<a href="#">RTC_CNTL_INT_ENA_RTC_W1TC_REG</a>	RTC interrupt clear register (W1TC)	0x00F0	R/W

## 9.7 Registers

The addresses in this section are relative to low-power management base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

### Register 9.1. RTC\_CNTL\_OPTIONS0\_REG (0x0000)

[illegible]

**RTC\_CNTL\_SW\_STALL\_PROCPU\_C0** Write 0x2 to stall the CPU by SW. Valid only when **RTC\_CNTL\_SW\_STALL\_PROCPU\_C1** is configured to 0x21. (R/W)

**RTC\_CNTL\_SW\_PROCPU\_RST** Write 1 to reset the CPU by SW. (WO)

**RTC\_CNTL\_BB\_I2C\_FORCE\_PD** Write 1 to FPD BB\_I2C. (R/W)

**RTC\_CNTL\_BB\_I2C\_FORCE\_PU** Write 1 to FPU BB\_I2C. (R/W)

**RTC\_CNTL\_BBPLL\_I2C\_FORCE\_PD** Write 1 to FPD BB\_PLL\_I2C. (R/W)

**RTC\_CNTL\_BBPLL\_I2C\_FORCE\_PU** Write 1 to FPU BB\_PLL\_I2C. (R/W)

**RTC\_CNTL\_BBPLL\_FORCE\_PD** Write 1 to FPD BB\_PLL. (R/W)

**RTC\_CNTL\_BBPLL\_FORCE\_PU** Write 1 to FPU BB\_PLL. (R/W)

**RTC\_CNTL\_XTL\_FORCE\_PD** Write 1 to FPD the XTAL\_CLK. (R/W)

**RTC CNTL XTL FORCE PU** Write 1 to FPU the XTAL CLK. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_RST** Write 1 to force reset the digital system in deep-sleep. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_NORST** Write 1 to disable force reset to the digital system in deep-sleep. (R/W)

**RTC\_CNTL\_SW\_SYS\_RST** Write 1 to reset the digital power category via SW. (WO)

**Register 9.2. RTC\_CNTL\_SLP\_TIMER0\_REG (0x0004)**

RTC_CNTL_SLP_VAL_LO																															
31																															0
0x000000																															
Reset																															

**RTC\_CNTL\_SLP\_VAL\_LO** Configures the lower 32 bits of the trigger threshold for the RTC timer.  
(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																0	0x00																															Reset

**RTC\_CNTL\_SLP\_VAL\_HI** Configures the higher 16 bits of the trigger threshold for the RTC timer.  
(R/W)

**RTC\_CNTL\_MAIN\_TIMER\_ALARM\_EN** Write 1 to enable the timer alarm. (R/W)

**Register 9.4. RTC\_CNTL\_TIME\_UPDATE\_REG (0x000C)**

RTC_CNTL_TIME_UPDATE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
RTC_CNTL_TIMER_SYS_RST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
RTC_CNTL_TIMER_XTL_OFF																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
RTC_CNTL_TIMER_SYS_STALL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
(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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

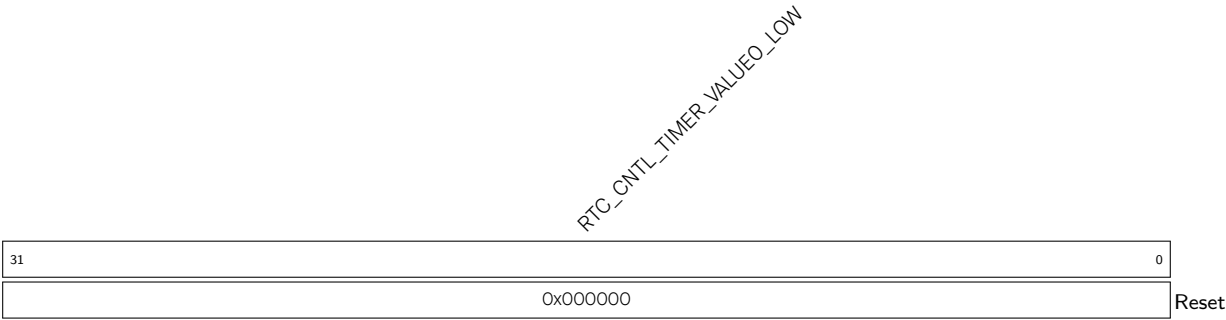
**RTC\_CNTL\_TIMER\_SYS\_STALL** Write 1 to enable recording digital system stall time. (R/W)

**RTC\_CNTL\_TIMER\_XTL\_OFF** Write 1 to enable recording XTAL\_CLK OFF time. (R/W)

**RTC\_CNTL\_TIMER\_SYS\_RST** Write 1 to enable recording digital system reset time. (R/W)

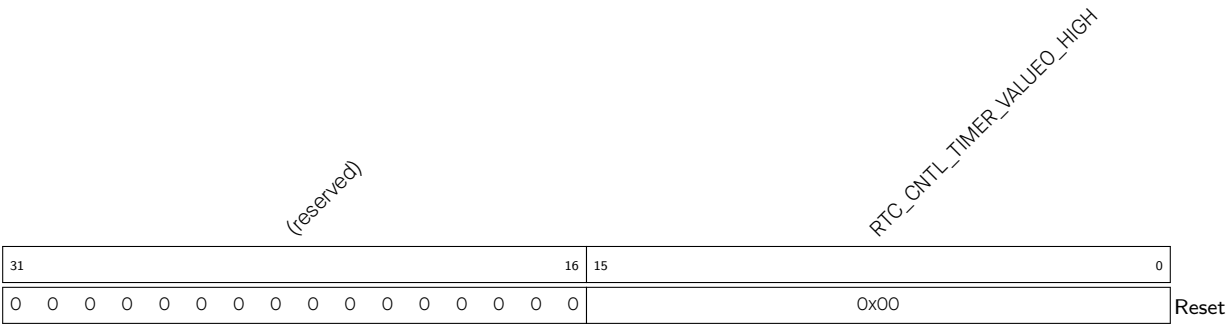
**RTC\_CNTL\_TIME\_UPDATE** Write 1 to update register with RTC timer. (R/W)

Register 9.5. RTC\_CNTL\_TIME\_LOW0\_REG (0x0010)



`RTC_CNTL_TIMER_VALUE0_LOW` Represents the lower 32 bits of RTC timer 0. (R/W)

Register 9.6. RTC\_CNTL\_TIME\_HIGH0\_REG (0x0014)



`RTC_CNTL_TIMER_VALUE0_HIGH` Represents the higher 16 bits of RTC timer 0. (R/W)

**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_APB2RTC_BRIDGE_SEL				(reserved)				RTC_CNTL_SLP_REJECT_CAUSE_CLR				RTC_CNTL_SW_CPU_INT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31	30	29	28	27						23	22	21																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									

**RTC\_CNTL\_SW\_CPU\_INT** Write 1 to send a SW RTC interrupt to CPU. (R/W)

**RTC\_CNTL\_SLP\_REJECT\_CAUSE\_CLR** Write 1 to clear the RTC reject-to-sleep cause. (R/W)

**RTC\_CNTL\_APB2RTC\_BRIDGE\_SEL** Configures the APB to RTC option.

0x0: APB to RTC using sync

0x1: APB to RTC using bridge

(R/W)

**RTC\_CNTL\_SDIO\_ACTIVE\_IND** Represents the SDIO is active. (R/W)

**RTC\_CNTL\_SLP\_WAKEUP** Represents wakeup events. (R/W)

**RTC\_CNTL\_SLP\_REJECT** Represents reject-to-sleep events. (R/W)

**RTC\_CNTL\_SLEEP\_EN** Write 1 to send the chip to sleep. (R/W)

**Register 9.8. RTC\_CNTL\_TIMER1\_REG (0x001C)**

RTC_CNTL_PLL_BUF_WAIT				RTC_CNTL_XTL_BUF_WAIT				RTC_CNTL_FOSC_WAIT				RTC_CNTL_CPU_STALL_WAIT				RTC_CNTL_CPU_STALL_EN																			
31				24				23				14				13				6				5				1				0			
40				80				0x10				1				1				Reset															

**RTC\_CNTL\_CPU\_STALL\_EN** Write 1 to enable CPU stalling. (R/W)

**RTC\_CNTL\_CPU\_STALL\_WAIT** Configures CPU stall wait cycles using RTC\_FAST\_CLK. (R/W)

**RTC\_CNTL\_FOSC\_WAIT** Configures the RC\_FAST\_CLK wait cycles using RTC\_SLOW\_CLK. (R/W)

**RTC\_CNTL\_XTL\_BUF\_WAIT** Configures XTAL\_CLK wait cycles using RTC\_SLOW\_CLK. (R/W)

**RTC\_CNTL\_PLL\_BUF\_WAIT** Configures the PLL\_CLK wait cycles using RTC\_SLOW\_CLK. (R/W)

**Register 9.9. RTC\_CNTL\_TIMER2\_REG (0x0020)**

RTC_CNTL_MN_TIME_FOSC_OFF																								(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																								24	23																								0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0x1																								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**RTC\_CNTL\_MIN\_TIME\_FOSC\_OFF** Configures the minimal cycle for RC\_FAST\_CLK (using RTC\_SLOW\_CLK) when powered down. (R/W)

**Register 9.10. RTC\_CNTL\_ANA\_CONF\_REG (0x002C)**

(reserved)										RTC_CNTL_SAR_I2C_PU										(reserved)																						
31										23										22	21	20	19	18	17																	0
0 0 0 0 0 0 0 0 0 0										1	0	0	0	1	0 0																	Reset										

**RTC\_CNTL\_I2C\_RESET\_POR\_FORCE\_PD** Write 1 to force not bypass I2C power-on reset. (R/W)

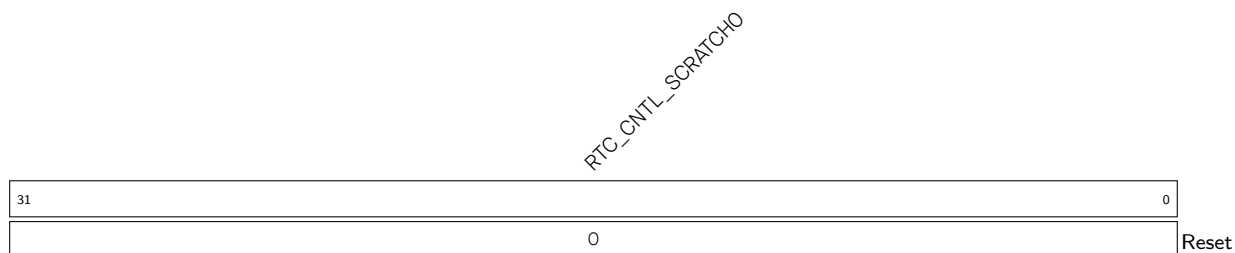
**RTC\_CNTL\_I2C\_RESET\_POR\_FORCE\_PU** Write 1 to force bypass I2C power-on reset. (R/W)

**RTC\_CNTL\_SAR\_I2C\_PU** Write 1 to FPU the SAR\_I2C. (R/W)

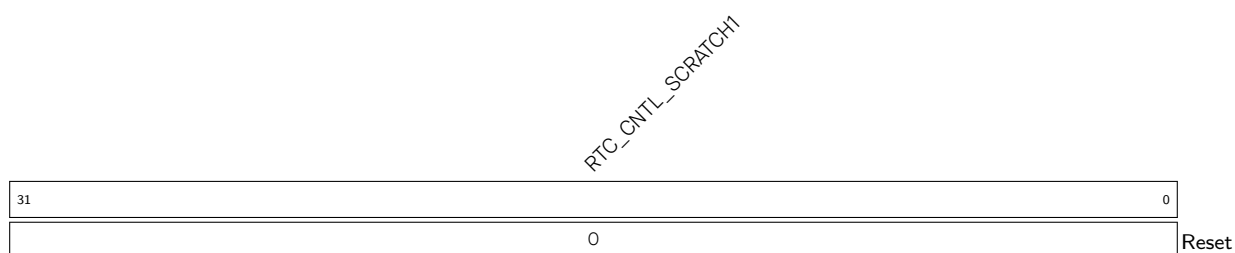
**Register 9.11. RTC\_CNTL\_WAKEUP\_STATE\_REG (0x0034)**

RTC_CNTL_WAKEUP_ENA																(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31																15	14	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

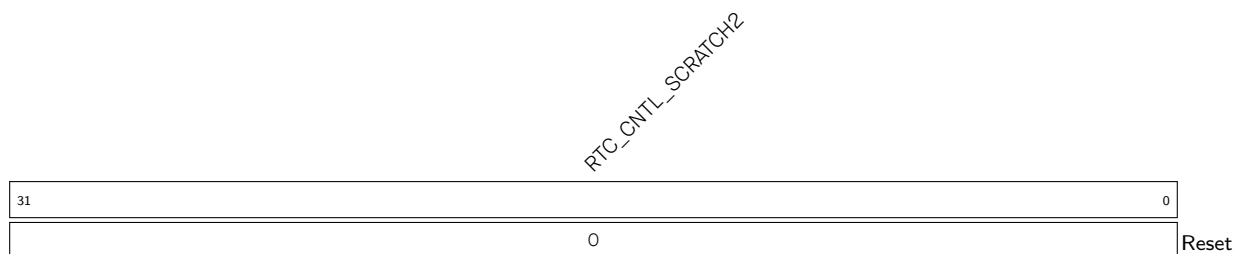
**RTC\_CNTL\_WAKEUP\_ENA** Configures the wakeup source. For details, please refer to Table 9.5-2. (R/W)

**Register 9.12. RTC\_CNTL\_STORE0\_REG (0x0048)**

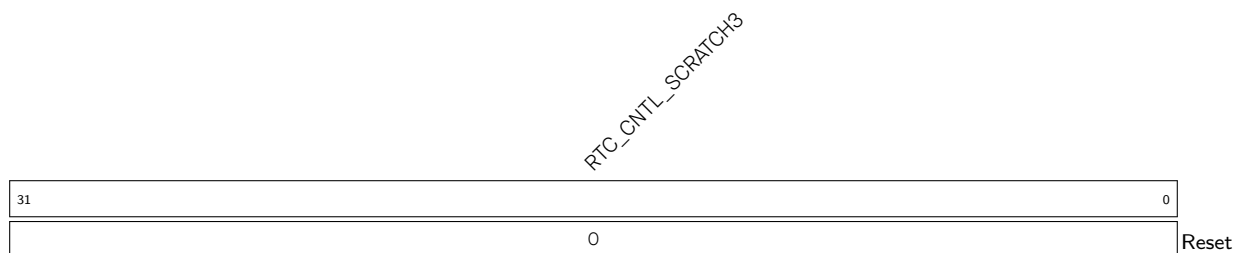
**RTC\_CNTL\_SCRATCH0** Reservation register 0. (R/W)

**Register 9.13. RTC\_CNTL\_STORE1\_REG (0x004C)**

**RTC\_CNTL\_SCRATCH1** Reservation register 1. (R/W)

**Register 9.14. RTC\_CNTL\_STORE2\_REG (0x0050)**

**RTC\_CNTL\_SCRATCH2** Reservation register 2. (R/W)

**Register 9.15. RTC\_CNTL\_STORE3\_REG (0x0054)**

**RTC\_CNTL\_SCRATCH3** Reservation register 3. (R/W)

Register 9.16. RTC\_CNTL\_EXT\_WAKEUP\_CONF\_REG (0x005C)

[illegible]

**RTC\_CNTL\_GPIO\_WAKEUP\_FILTER** Write 1 to enable the GPIO wakeup event filter. (R/W)

### Register 9.17. RTC\_CNTL\_SLP\_REJECT\_CONF\_REG (0x0060)

RTC_CNTL_DEEP_SLP_REJECT_EN		RTC_CNTL_LIGHT_SLP_REJECT_EN		RTC_CNTL_SLEEP_REJECT_ENA		(reserved)												
31	30	29				12	11											0
0	0	0				0 0 0 0 0 0 0 0 0 0 0 0 0 0											Reset	

**RTC\_CNTL\_SLEEP\_REJECT\_ENA** Write 1 to enable reject-to-sleep. (R/W)

**RTC\_CNTL\_LIGHT\_SLP\_REJECT\_EN** Write 1 to enable reject-to-light-sleep. (R/W)

**RTC\_CNTL\_DEEP\_SLP\_REJECT\_EN** Write 1 to enable reject-to-deep-sleep. (R/W)



**Register 9.18. RTC CNTL CLK CONF REG (0x0068)**

RTC_CNTL_ANA_CLK_RTC_SEL																										RTC_CNTL_FAST_CLK_RTC_SEL										RTC_CNTL_XTAL_GLOBAL_FORCE_NOGATING										RTC_CNTL_XTAL_GLOBAL_FORCE_GATING										RTC_CNTL_FOSC_FORCE_PU										RTC_CNTL_FOSC_FORCE_PD										RTC_CNTL_FOSC_DFREQ										RTC_CNTL_FOSC_DIV_SEL										RTC_CNTL_FOSC_FORCE_NOGATING										RTC_CNTL_XTAL_FORCE_NOGATING										(reserved)										RTC_CNTL_DIG_CLK8M_EN										(reserved)										(reserved)										RTC_CNTL_ENB_FOSC_DIV										RTC_CNTL_ENB_FOSC										RTC_CNTL_FOSC_DIV										RTC_CNTL_FOSC_DIV_SEL_VLD										RTC_CNTL_EFUSE_CLK_FORCE_NOGATING										(reserved)									
31	30	29	28	27	26	25	24											17	16	15	14	12		11	10	9	8	7	6	5	4		3	2	1	0																																																																																																																																																																																			
0	0	1	0	0	0	172										0	0	3		0	0	1	0	0	0	0	1	1	0	0	0	0	Reset																																																																																																																																																																																						

**RTC\_CNTL\_EFUSE\_CLK\_FORCE\_GATING** Write 1 to FPU the eFuse clock gating. (R/W)

**RTC\_CNTL\_EFUSE\_CLK\_FORCE\_NOGATING** Write 1 to FPD the eFuse clock gating. (R/W)

**RTC\_CNTL\_FOSC\_DIV\_SEL\_VLD** Write 1 to synchronize [RTC\\_CNTL\\_FOSC\\_DIV\\_SEL](#). Note that you have to invalidate the bus before modifying the frequency divider, then validate the new divider clock. (R/W)

**RTC\_CNTL\_FOSC\_DIV** Configures the RC\_FAST\_DIV\_CLK divider.

0x0: divided by 128

0x1: divided by 256

0x2: divided by 512

0x3: divided by 1024

(R/W)

**RTC\_CNTL\_ENB\_FOSC** Write 1 to disable RC\_FAST\_CLK and RC\_FAST\_DIV\_CLK. (R/W)

**RTC\_CNTL\_ENB\_FOSC\_DIV** Configures the RC\_FAST\_CLK divider.

0x0: RC\_FAST\_CLK divided by 256

0x1: RC FAST CLK

(R/W)

**RTC\_CNTL\_DIG\_FOSC\_EN** Write 1 to enable RC\_FAST\_CLK for the digital system. (R/W)

**RTC\_CNTL\_FOSC\_DIV\_SEL** Represents the RC\_FAST\_CLK divider, which is **RTC\_CNTL\_FOSC\_DIV\_SEL + 1**. (R/W)

Continued on the next page...

**Register 9.18. RTC\_CNTL\_CLK\_CONF\_REG (0x0068)**

Continued from the previous page...

**RTC\_CNTL\_XTAL\_FORCE\_NOGATING** Write 1 to force no gating to XTAL\_CLK during sleep. (R/W)

**RTC\_CNTL\_FOSC\_FORCE\_NOGATING** Write 1 to disable force gating to XTAL\_CLK during sleep.  
(R/W)

**RTC\_CNTL\_FOSC\_DFREQ** Configures the RC\_FAST\_CLK frequency. (R/W)

**RTC\_CNTL\_FOSC\_FORCE\_PD** Write 1 to FPD RC\_FAST\_CLK. (R/W)

**RTC\_CNTL\_FOSC\_FORCE\_PU** Write 1 to FPU RC\_FAST\_CLK. (R/W)

**RTC\_CNTL\_XTAL\_GLOBAL\_FORCE\_GATING** Write 1 to force enable XTAL\_CLK clock gating. (R/W)

**RTC\_CNTL\_XTAL\_GLOBAL\_FORCE\_NOGATING** Write 1 to force bypass the XTAL\_CLK clock gating.  
(R/W)

**RTC\_CNTL\_FAST\_CLK\_RTC\_SEL** Configures the RTC\_FAST\_CLK.

0x0: XTAL\_DIV\_CLK

0x1: FOSC\_DIV

(R/W)

**RTC\_CNTL\_ANA\_CLK\_RTC\_SEL** Configures the RTC\_SLOW\_CLK.

0x0: RC\_SLOW\_CLK

0x1: OSC\_SLOW\_CLK

0x2: RC\_FAST\_DIV\_CLK

0x3: Reserved

(R/W)

### Register 9.19. RTC\_CNTL\_REG (0x0074)

RTC_CNTL_REGULATOR_FORCE_PU																																RTC_CNTL_REGULATOR_FORCE_PD																																(reserved)																																RTC_CNTL_SCK_DCAP																																(reserved)																																RTC_CNTL_DIG_REG_CAL_EN																																(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31	30	29																												22	21	14																												13	8																												7	6	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

**RTC\_CNTL\_DIG\_REG\_CAL\_EN** Write 1 to enable digital regulator calibration by SW. (R/W)

**RTC\_CNTL\_SCK\_DCAP** Configures the RC\_SLOW\_CLK frequency. (R/W)

**RTC\_CNTL\_REGULATOR\_FORCE\_PD** Write 1 to FPD the low-power voltage regulator, which means decreasing its voltage to 0.8 V or lower. (R/W)

**RTC\_CNTL\_REGULATOR\_FORCE\_PU** Write 1 to FPU the low-power voltage regulator, which means increasing its voltage to higher than 0.8 V. (R/W)

### Register 9.20. RTC\_CNTL\_PWC\_REG (0x0078)

Diagram illustrating the structure of the `RTC_CNTL_FORCE_HOLD` register. The register is 32 bits wide, divided into three sections:

- Bits 31-22: (reserved)
- Bits 21-20: `RTC_CNTL_PAD_FORCE_HOLD`
- Bits 19-0: (reserved)

The register is labeled "Reset" on the right.

**RTC\_CNTL\_PAD\_FORCE\_HOLD** Write 1 to force RTC pad into hold state. (R/W)

### Register 9.21. RTC\_CNTL\_DIG\_PWC\_REG (0x007C)

[illegible]

**RTC\_CNTL\_VDD\_SPI\_PWR\_DRV** Configures the vdd\_spi's drive intensity. (R/W)

**RTC\_CNTL\_VDD\_SPI\_PWR\_FORCE** Write 1 to allow software to configure vdd\_spi's drive intensity.  
(R/W)

**RTC\_CNTL\_VDD\_SPI\_PD\_EN** Write 1 to power down VDD\_SPI in sleep. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_PD** Write 1 to FPD the digital system. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_PU** Write 1 to FPU the digital system. (R/W)

**RTC\_CNTL\_DG\_WRAP\_PD\_EN** Write 1 to enable FPD digital system in sleep. (R/W)

Register 9.22. RTC\_CNTL\_DIG\_ISO\_REG (0x0080)

RTC_CNTL_DG_WRAP_FORCE_NOISO RTC_CNTL_DG_WRAP_FORCE_ISO (reserved)																															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_CLR_DG_PAD_AUTOHOLD_EN RTC_CNTL_CLR_DG_PAD_AUTOHOLD (reserved)																														
31	30	29														16	15	14	13	12	11	10	9	8	0																																				
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset																									

**RTC\_CNTL\_DG\_PAD\_AUTOHOLD** Indicates the auto-hold status of the digital GPIOs. (R/W)

**RTC\_CNTL\_CLR\_DG\_PAD\_AUTOHOLD** Write 1 to clear the auto-hold enabler for the digital GPIOs.  
(R/W)

**RTC\_CNTL\_DG\_PAD\_AUTOHOLD\_EN** Write 1 to allow the digital GPIOs to enter the auto-hold status.  
(R/W)

**RTC\_CNTL\_DG\_PAD\_FORCE\_NOISO** Write 1 to disable the force isolation of the digital GPIOs. (R/W)

**RTC\_CNTL\_DG\_PAD\_FORCE\_ISO** Write 1 to force isolation of the digital GPIOs. (R/W)

**RTC\_CNTL\_DG\_PAD\_FORCE\_UNHOLD** Write 1 the force unhold the digital GPIOs. (R/W)

**RTC\_CNTL\_DG\_PAD\_FORCE\_HOLD** Write 1 the force hold the digital GPIOs. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_ISO** Write 1 to force isolation of the digital system. (R/W)

**RTC\_CNTL\_DG\_WRAP\_FORCE\_NOISO** Write 1 to disable the force isolation of the digital system.  
(R/W)

**Register 9.23. RTC\_CNTL\_WDTCONFIG0\_REG (0x0084)**

RTC_CNTL_WDT_EN		RTC_CNTL_WDT_STG0		RTC_CNTL_WDT_STG1		RTC_CNTL_WDT_STG2		RTC_CNTL_WDT_STG3		RTC_CNTL_WDT_CPU_RESET_LENGTH		RTC_CNTL_WDT_SYS_RESET_LENGTH		RTC_CNTL_WDT_FLASHBOOT_MOD_EN		RTC_CNTL_WDT_PROCPU_RESET_EN		(reserved)		RTC_CNTL_WDT_PAUSE_IN_SLP		(reserved)												
31	30	28	27	25	24	22	21	19	18	16	15	13	12	11	10	9	8																	0
0	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x1	0x1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

**RTC\_CNTL\_WDT\_PAUSE\_IN\_SLP** Write 1 to pause the watchdog in sleep. (R/W)

**RTC\_CNTL\_WDT\_PROCPU\_RESET\_EN** Write 1 to enable RTC WDT to reset CPU. (R/W)

**RTC\_CNTL\_WDT\_FLASHBOOT\_MOD\_EN** Write 1 to enable watchdog when the chip boots from flash. (R/W)

**RTC\_CNTL\_WDT\_SYS\_RESET\_LENGTH** Configures the length of the digital system reset counter. (R/W)

**RTC\_CNTL\_WDT\_CPU\_RESET\_LENGTH** Configures the length of the CPU reset counter. (R/W)

**RTC\_CNTL\_WDT\_STG3** Configures the timeout action for RTC watchdog timer at stage 3.

0x1: triggers an interrupt

0x2: resets the CPU core

0x3: resets the digital system excluding RTC

0x4: resets the digital system including RTC

(R/W)

**RTC\_CNTL\_WDT\_STG2** Configures the timeout action for RTC watchdog timer at stage 2.

0x1: triggers an interrupt

0x2: resets the CPU core

0x3: resets the digital system excluding RTC

0x4: resets the digital system including RTC

(R/W)

Continued on the next page...

**Register 9.23. RTC\_CNTL\_WDTCONFIG0\_REG (0x0084)**

Continued from the previous page...

**RTC\_CNTL\_WDT\_STG1** Configures the timeout action for RTC watchdog timer at stage 1.

0x1: triggers an interrupt

0x2: resets the CPU core

0x3: resets the digital system excluding RTC

0x4: resets the digital system including RTC

(R/W)

**RTC\_CNTL\_WDT\_STG0** Configures the timeout action for RTC watchdog timer at stage 0.

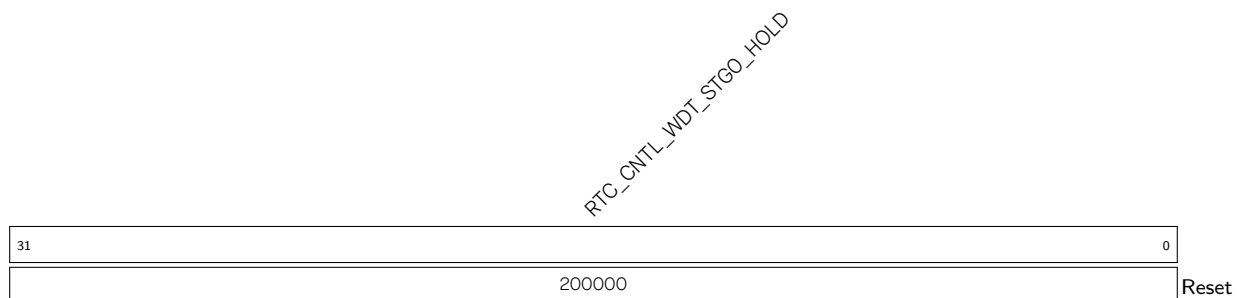
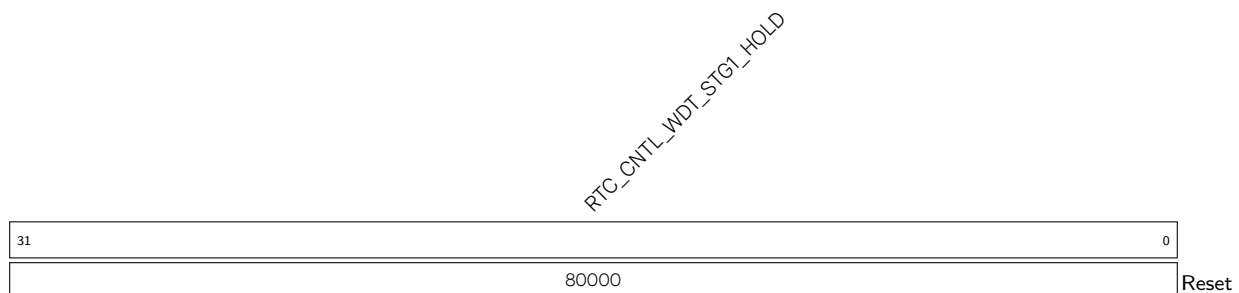
0x1: triggers an interrupt

0x2: resets the CPU core

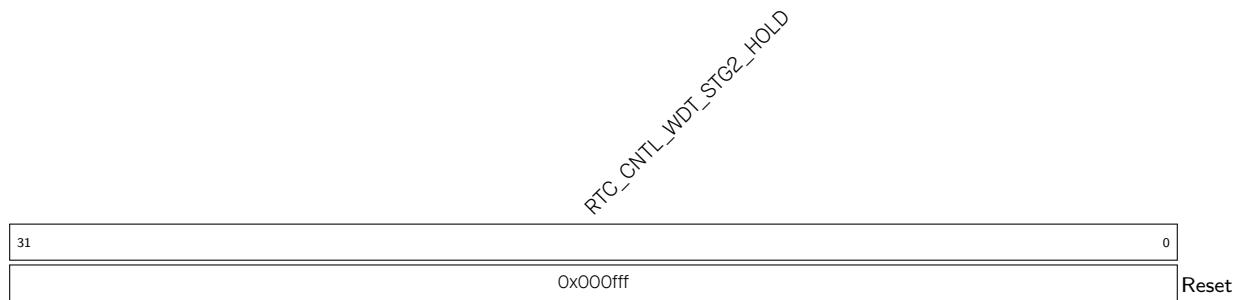
0x3: resets the digital system excluding RTC

0x4: resets the digital system including RTC

(R/W)

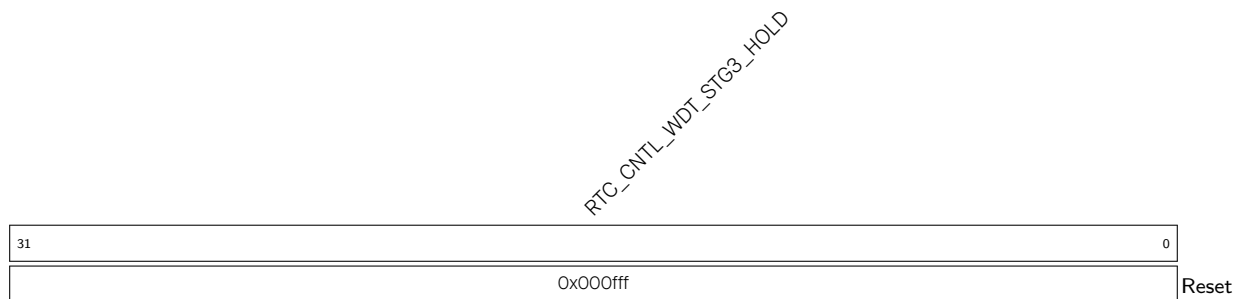
**RTC\_CNTL\_WDT\_EN** Write 1 to enable the RTC watchdog. (R/W)**Register 9.24. RTC\_CNTL\_WDTCONFIG1\_REG (0x0088)****RTC\_CNTL\_WDT\_STG0\_HOLD** Configures the hold time of RTC watchdog in stage 0. (R/W)**Register 9.25. RTC\_CNTL\_WDTCONFIG2\_REG (0x008C)****RTC\_CNTL\_WDT\_STG1\_HOLD** Configures the hold time of RTC watchdog in stage 1. (R/W)

### Register 9.26. RTC\_CNTL\_WDTCFIG3\_REG (0x0090)



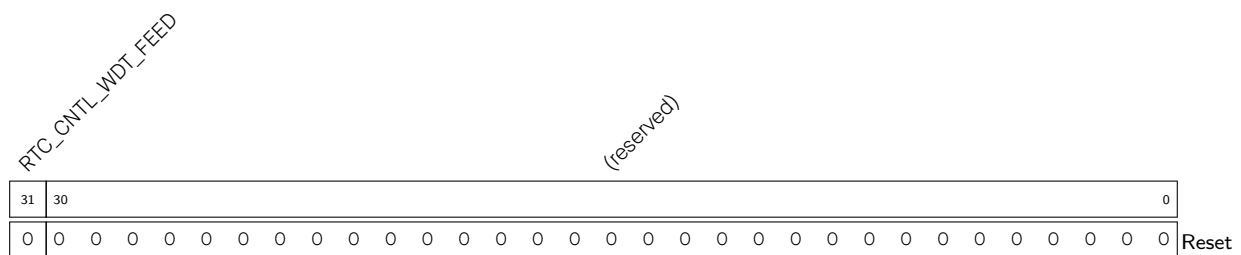
**RTC\_CNTL\_WDT\_STG2\_HOLD** Configures the hold time of RTC watchdog in stage 2. (R/W)

### Register 9.27. RTC\_CNTL\_WDTCONFIG4\_REG (0x0094)



**RTC\_CNTL\_WDT\_STG3\_HOLD** Configures the hold time of RTC watchdog in stage 3. (R/W)

### Register 9.28. RTC\_CNTL\_WDTFEED\_REG (0x0098)



**RTC\_CNTL\_WDT\_FEED** Write 1 to feed the RTC watchdog. (R/W)



**Register 9.29. RTC\_CNTL\_WDTWPROTECT\_REG (0x009C)**

RTC_CNTL_WDT_WKEY																															
31																															0
0x000000																															
Reset																															

**RTC\_CNTL\_WDT\_WKEY** If the register contains a different value than 0x50d83aa1, write protection for the RTC watchdog (RWDT) is enabled. (R/W)

**Register 9.30. RTC\_CNTL\_SWD\_CONF\_REG (0x00A0)**

RTC_CNTL_SWD_AUTO_FEED_EN				RTC_CNTL_SWD_SIGNAL_WIDTH													(reserved)													RTC_CNTL_SWD_FEED_INT		RTC_CNTL_SWD_RESET_FLAG																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
RTC_CNTL_SWD_DISABLE				RTC_CNTL_SWD_RST_FLAG_CLR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
31	30	29	28	27														18	17	16												2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
0	0	0	0	300													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**RTC\_CNTL\_SWD\_RESET\_FLAG** Represents the super watchdog reset flag. (R/W)

**RTC\_CNTL\_SWD\_FEED\_INT** Represents super watchdog will be fed via SW. (R/W)

**RTC\_CNTL\_SWD\_BYPASS\_RST** Write 1 to bypass super watchdog reset. (R/W)

**RTC\_CNTL\_SWD\_SIGNAL\_WIDTH** Configures the signal width sent to the super watchdog. (R/W)

**RTC\_CNTL\_SWD\_RST\_FLAG\_CLR** Write 1 to reset the super watchdog reset flag. (R/W)

**RTC\_CNTL\_SWD\_FEED** Write 1 to feed the super watchdog via SW. (R/W)

**RTC\_CNTL\_SWD\_DISABLE** Write 1 to disable super watchdog. (R/W)

**RTC\_CNTL\_SWD\_AUTO\_FEED\_EN** Write 1 to enable automatic watchdog feeding upon interrupt. (R/W)

**Register 9.31. RTC\_CNTL\_SWD\_WPROTECT\_REG (0x00A4)**

RTC_CNTL_SWD_WKEY	
31	0
0x000000	
Reset	

**RTC\_CNTL\_SWD\_WKEY** Configures the write protection key of the super watchdog. (R/W)

**Register 9.32. RTC\_CNTL\_SW\_CPU\_STALL\_REG (0x00A8)**

RTC_CNTL_SW_STALL_PROCPU_C1	
(reserved)	
31	0
26	25
0	0 0
Reset	

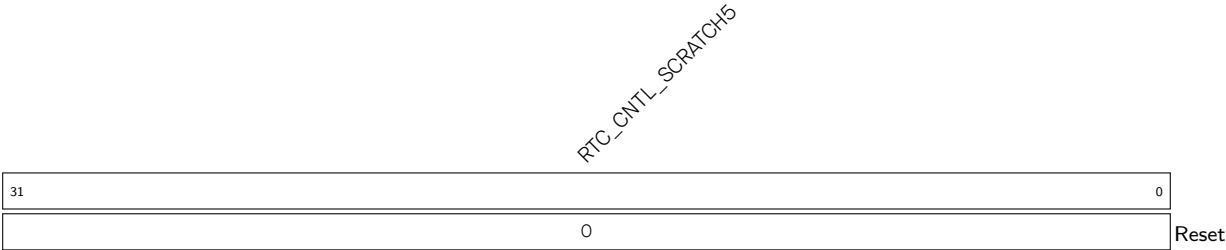
**RTC\_CNTL\_SW\_STALL\_PROCPU\_C1** Write 0x21 to stall the CPU by SW. Valid only when [RTC\\_CNTL\\_SW\\_STALL\\_PROCPU\\_CO](#) is configured to 0x2. (R/W)

**Register 9.33. RTC\_CNTL\_STORE4\_REG (0x00AC)**

RTC_CNTL_SCRATCH4	
31	0
0	
Reset	

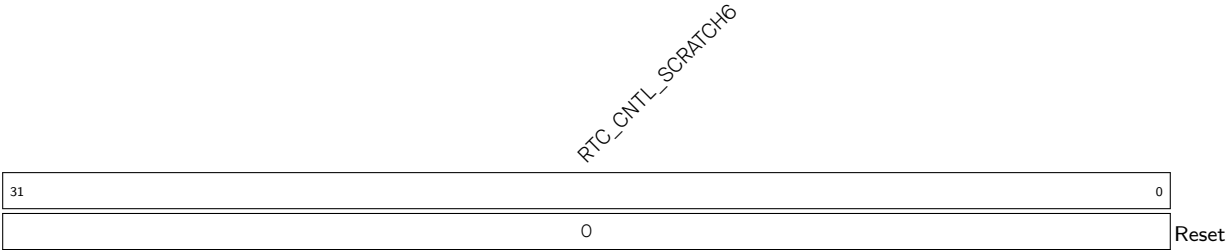
**RTC\_CNTL\_SCRATCH4** Reservation register 4. (R/W)

Register 9.34. RTC\_CNTL\_STORE5\_REG (0x00B0)



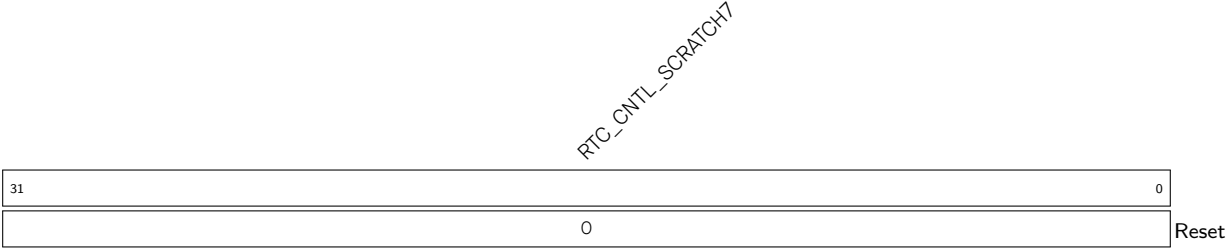
RTC\_CNTL\_SCRATCH5 Reservation register 5. (R/W)

Register 9.35. RTC\_CNTL\_STORE6\_REG (0x00B4)



RTC\_CNTL\_SCRATCH6 Reservation register 6. (R/W)

Register 9.36. RTC\_CNTL\_STORE7\_REG (0x00B8)



RTC\_CNTL\_SCRATCH7 Reservation register 7. (R/W)

### Register 9.37. RTC\_CNTL\_LOW\_POWER\_ST\_REG (0x00BC)

**RTC\_CNTL\_RDY\_FOR\_WAKEUP** 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_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).

### Register 9.38. RTC CNTL\_PAD\_HOLD\_REG (0x00C4)

[illegible]

**RTC\_CNTL\_GPIO\_PIN0\_HOLD** Sets the GPIO 0 to the holding state. (R/W)

**RTC\_CNTL\_GPIO\_PIN1\_HOLD** Sets the GPIO 1 to the holding state. (R/W)

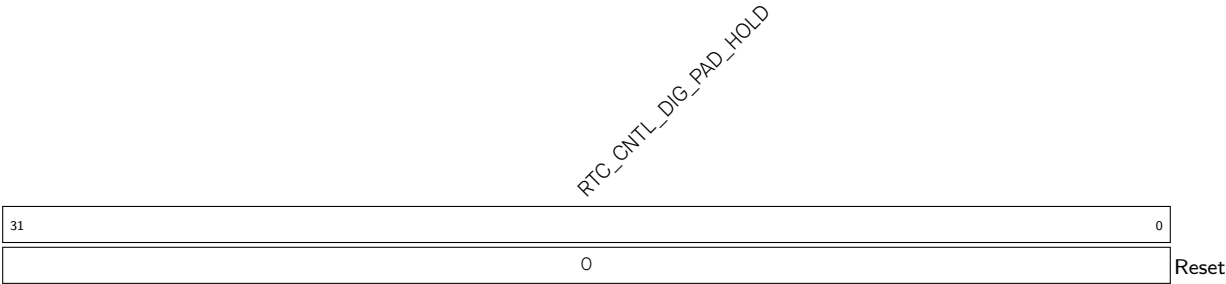
**RTC\_CNTL\_GPIO\_PIN2\_HOLD** Sets the GPIO 2 to the holding state. (R/W)

**RTC\_CNTL\_GPIO\_PIN3\_HOLD** Sets the GPIO 3 to the holding state. (R/W)

<b>RTC_CNTL_GPIO_PIN4_HOLD</b>	Sets the GPIO 4 to the holding state. (R/W)
--------------------------------	---

**RTC\_CNTL\_GPIO\_PIN5\_HOLD** Sets the GPIO 5 to the holding state. (R/W)

Register 9.39. RTC\_CNTL\_DIG\_PAD\_HOLD\_REG (0x00C8)



**RTC\_CNTL\_DIG\_PAD\_HOLD** Set GPIO 6 to GPIO 20 to the holding state. (See bitmap to locate any GPIO). (R/W)

**Register 9.40. RTC\_CNTL\_BROWN\_OUT\_REG (0x00CC)**

RTC_CNTL_BROWN_OUT_DET						RTC_CNTL_BROWN_OUT_RST_WAIT						RTC_CNTL_BROWN_OUT_PD_RF_ENA						RTC_CNTL_BROWN_OUT_INT_WAIT				(reserved)	
RTC_CNTL_BROWN_OUT_ENA						RTC_CNTL_BROWN_OUT_RST_ENA						RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA											
RTC_CNTL_BROWN_OUT_CNT_CLR																							
RTC_CNTL_BROWN_OUT_ANA_RST_SEL																							
RTC_CNTL_BROWN_OUT_RST_ENA																							
31	30	29	28	27	26	25					16	15	14	13					4	3			0
0	1	0	0	0	0	0x3ff					0	0	0x1					0	0	0	0	Reset	

**RTC\_CNTL\_BROWN\_OUT\_INT\_WAIT** Configures the waiting cycles before sending an interrupt. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_CLOSE\_FLASH\_ENA** Write 1 to enable PD the flash when a brownout happens. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_PD\_RF\_ENA** Write 1 to enable PD the RF circuits when a brownout happens. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_RST\_WAIT** Configures the waiting cycles before the reset after a brownout. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_RST\_ENA** Write 1 to reset brown-out. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_RST\_SEL** Configures the reset type when a brownout happens in mode0.  
 0x0: system reset  
 0x1: chip reset  
 (R/W)

**RTC\_CNTL\_BROWN\_OUT\_ANA\_RST\_EN** Write 1 to enable brownout detection mode1. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_CNT\_CLR** Write 1 to clear the brownout counter. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_ENA** Write 1 to enable brownout detection mode0. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_DET** Represents the status of the brownout signal. (R/W)

Register 9.41. RTC\_CNTL\_TIME\_LOW1\_REG (0x00D0)

RTC_CNTL_TIMER_VALUE1_LOW																															
31																															0
0x000000																															
Reset																															

**RTC\_CNTL\_TIMER\_VALUE1\_LOW** Represents the lower 32 bits of RTC timer 1. (R/W)

Register 9.42. RTC\_CNTL\_TIME\_HIGH1\_REG (0x00D4)

(reserved)																RTC_CNTL_TIMER_VALUE1_HIGH																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x00																Reset															

**RTC\_CNTL\_TIMER\_VALUE1\_HIGH** Represents the higher 16 bits of RTC timer 1. (R/W)

Register 9.43. RTC\_CNTL\_USB\_CONF\_REG (0x00D8)

(reserved)																	RTC_CNTL_IO_MUX_RESET_DISABLE												
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
Reset																													

**RTC\_CNTL\_IO\_MUX\_RESET\_DISABLE** Write 1 to disable io\_mux reset. (R/W)

Register 9.44. RTC\_CNTL\_SLP\_REJECT\_CAUSE\_REG (0x00DC)

(reserved)																RTC_CNTL_REJECT_CAUSE															
31																	17													0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															0

**RTC\_CNTL\_REJECT\_CAUSE** Represents the reject-to-sleep cause. (R/W)

Register 9.45. RTC\_CNTL\_OPTION1\_REG (0x00E0)

(reserved)																																RTC_CNTL_FORCE_DOWNLOAD_BOOT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
31																															1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**RTC\_CNTL\_FORCE\_DOWNLOAD\_BOOT** Write 1 to force the chip to boot from the download mode.  
(R/W)

Register 9.46. RTC\_CNTL\_SLP\_WAKEUP\_CAUSE\_REG (0x00E4)

(reserved)																RTC_CNTL_WAKEUP_CAUSE																			
31																	17	16																	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																0	Reset			

**RTC\_CNTL\_WAKEUP\_CAUSE** Represents the wakeup cause. (R/W)



Register 9.47. RTC\_CNTL\_CNTL\_GPIO\_WAKEUP\_REG (0x00FC)

RTC_CNTL_GPIO_PIN0_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN1_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN2_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN3_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN4_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN5_WAKEUP_ENABLE		RTC_CNTL_GPIO_PIN0_INT_TYPE		RTC_CNTL_GPIO_PIN1_INT_TYPE		RTC_CNTL_GPIO_PIN2_INT_TYPE		RTC_CNTL_GPIO_PIN3_INT_TYPE		RTC_CNTL_GPIO_PIN4_INT_TYPE		RTC_CNTL_GPIO_PIN5_INT_TYPE		RTC_CNTL_GPIO_PIN_CLK_GATE		RTC_CNTL_GPIO_WAKEUP_STATUS_CLR		RTC_CNTL_GPIO_WAKEUP_STATUS	
31	30	29	28	27	26	25	23		22	20		19	17		16	14		13	11		10	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		Reset			

Reset

**RTC\_CNTL\_GPIO\_WAKEUP\_STATUS** Write 1 to set the RTC GPIO wakeup flag. (R/W)**RTC\_CNTL\_GPIO\_WAKEUP\_STATUS\_CLR** Write 1 to clear the RTC GPIO flag. (R/W)**RTC\_CNTL\_GPIO\_PIN\_CLK\_GATE** Write 1 to enable the RTC GPIO clock gating. (R/W)**RTC\_CNTL\_GPIO\_PIN5\_INT\_TYPE** Configures GPIO 5 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PIN4\_INT\_TYPE** Configures GPIO 4 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PIN3\_INT\_TYPE** Configures GPIO 3 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PIN2\_INT\_TYPE** Configures GPIO 2 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PIN1\_INT\_TYPE** Configures GPIO 1 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PINO\_INT\_TYPE** Configures GPIO 0 wakeup type. (R/W)**RTC\_CNTL\_GPIO\_PIN5\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 5. (R/W)**RTC\_CNTL\_GPIO\_PIN4\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 4. (R/W)**RTC\_CNTL\_GPIO\_PIN3\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 3. (R/W)**RTC\_CNTL\_GPIO\_PIN2\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 2. (R/W)**RTC\_CNTL\_GPIO\_PIN1\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 1. (R/W)**RTC\_CNTL\_GPIO\_PINO\_WAKEUP\_ENABLE** Write 1 to enable wakeup from RTC GPIO 0. (R/W)

Register 9.48. RTC\_CNTL\_CNTL\_SENSOR\_CTRL\_REG (0x0108)

Diagram of the RTC\_CNTL\_FORCE\_XPD\_SAR register. The register is 32 bits wide, with bits 31 and 30 labeled 'RTC\_CNTL\_FORCE\_XPD\_SAR' and bit 0 labeled 'Reset'. The register is divided into three sections: bits 31-30 are labeled 'RTC\_CNTL\_FORCE\_XPD\_SAR', bits 29-0 are labeled '(reserved)', and bit 0 is labeled 'Reset'.

**RTC\_CNTL\_FORCE\_XPD\_SAR** Set this field to FPU SAR ADC. (R/W)

### Register 9.49. RTC\_CNTL\_RESET\_STATE\_REG (0x0030)

(reserved)																RTC_CNTL_DRESET_MASK_PROCPU RTC_CNTL_ODD_HALT_ON_RESET_PROCPU										(reserved)				(reserved)				(reserved)				RTC_CNTL_RESET_CAUSE											
31																21				20	19	18				14				13	12	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				1	0 0 0 0 0 0 0 0				0																						
																																Reset																	

**RTC\_CNTL\_RESET\_CAUSE\_PROCPU** Represents the CPU reset cause. (R/W)

**RTC\_CNTL\_OCD\_HALT\_ON\_RESET\_PROCPU** Write 1 to send CPU into halt state upon CPU reset.  
(R/W)

**RTC\_CNTL\_DRESET\_MASK\_PROCPU** Write 1 to bybass D-reset. (R/W)

### Register 9.50. RTC\_CNTL\_INT\_ENA\_RTC\_REG (0x0038)

[illegible]

**RTC\_CNTL\_SLP\_WAKEUP\_INT\_ENA** Write 1 to enable interrupt when chip wakes up from sleep.  
(R/W)

**RTC\_CNTL\_SLP\_REJECT\_INT\_ENA** Write 1 to enable interrupt when chip rejects to go to sleep.  
(R/W)

**RTC\_CNTL\_WDT\_INT\_ENA** Write 1 to enable RTC WDT interrupt. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_ENA** Write 1 to enable the brownout interrupt. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_ENA** Write 1 to enable the RTC timer interrupt. (R/W)

**RTC\_CNTL\_SWD\_INT\_ENA** Write 1 to enable the super watchdog interrupt. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_ENA** Write 1 to enable interrupt upon the ending of a BB\_PLL call.  
(R/W)

**Register 9.51. RTC\_CNTL\_INT\_RAW\_RTC\_REG (0x003C)**

[illegible]

**RTC\_CNTL\_SLP\_WAKEUP\_INT\_RAW** Represents the raw interrupt triggered when the chip wakes up from sleep. (R/W)

<b>RTC_CNTL_SLP_REJECT_INT_RAW</b>	Represents the raw interrupt triggered when the chip rejects to go to sleep. (R/W)
------------------------------------	--

**RTC\_CNTL\_WDT\_INT\_RAW** Represents the raw watchdog interrupt. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_RAW** Represents the raw brownout interrupt. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_RAW** Represents the raw RTC main timer interrupt. (R/W)

**RTC\_CNTL\_SWD\_INT\_RAW** Represents the raw super watchdog interrupt. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_RAW** Represents the raw interrupt upon the ending of a BB\_PLL call.  
(R/W)

### Register 9.52. RTC\_CNTL\_INT\_ST\_RTC\_REG (0x0040)

[illegible]

<b>RTC_CNTL_SLP_WAKEUP_INT_ST</b>	Represents the status of the interrupt triggered when the chip wakes up from sleep. (R/W)
-----------------------------------	---

<b>RTC_CNTL_SLP_REJECT_INT_ST</b>	Represents the status of the interrupt triggered when the chip rejects to go to sleep. (R/W)
-----------------------------------	--

**RTC\_CNTL\_WDT\_INT\_ST** Represents the status of the RTC watchdog interrupt. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_ST** Represents the status of the brownout interrupt. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_ST** Represents the status of the RTC main timer interrupt. (R/W)

**RTC\_CNTL\_SWD\_INT\_ST** Represents the status of the super watchdog interrupt. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_ST** Represents the status of the interrupt triggered upon the ending of a BB\_PLL call. (R/W)

**Register 9.53. RTC\_CNTL\_INT\_CLR\_RTC\_REG (0x0044)**

[illegible]

**RTC\_CNTL\_SLP\_WAKEUP\_INT\_CLR** Write 1 to clear the interrupt triggered when the chip wakes up from sleep. (R/W)

**RTC\_CNTL\_SLP\_REJECT\_INT\_CLR** Write 1 to clear the interrupt triggered when the chip rejects to go to sleep. (R/W)

**RTC\_CNTL\_WDT\_INT\_CLR** Write 1 to clear the RTC watchdog interrupt. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_CLR** Write 1 to clear the brownout interrupt. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_CLR** Write 1 to clear the RTC main timer interrupt. (R/W)

**RTC\_CNTL\_SWD\_INT\_CLR** Write 1 to clear the super watch dog interrupt state. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_CLR** Write 1 to clear the interrupt triggered upon the ending of a BB\_PLL call. (R/W)

Register 9.54. RTC\_CNTL\_INT\_ENA\_RTC\_W1TS\_REG (0x00EC)

[illegible]

**RTC\_CNTL\_SLP\_WAKEUP\_INT\_ENA\_W1TS** Write 1 to enable interrupt when the chip wakes up from sleep. If the value 1 is written to this bit, the **RTC\_CNTL\_SLP\_WAKEUP\_INT\_ENA** field will be set to 1. (R/W)

**RTC\_CNTL\_SLP\_REJECT\_INT\_ENA\_WITS** Write 1 to enable interrupt when the chip rejects to go to sleep. If the value 1 is written to this bit, the [RTC\\_CNTL\\_SLP\\_REJECT\\_INT\\_ENA](#) field will be set to 1. (R/W)

**RTC\_CNTL\_WDT\_INT\_ENA\_W1TS** Write 1 to enable the RTC watchdog interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_WDT\_INT\_ENA** field will be set to 1. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_ENA\_W1TS** Write 1 to enable the brownout interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_BROWN\_OUT\_INT\_ENA** field will be set to 1. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_ENA\_W1TS** Write 1 to enable the RTC main timer interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_MAIN\_TIMER\_INT\_ENA** field will be set to 1. (R/W)

**RTC\_CNTL\_SWD\_INT\_ENA\_W1TS** Write 1 to enable the super watchdog interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_SWD\_INT\_ENA** field will be set to 1. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_ENA\_W1TS** Write 1 to enable interrupt upon the ending of a BB\_PLL call. If the value 1 is written to this bit, the **RTC\_CNTL\_BBPLL\_CAL\_INT\_ENA** field will be set to 1. (R/W)

Register 9.55. RTC\_CNTL\_INT\_ENA\_RTC\_W1TC\_REG (0x00F0)

[illegible]

**RTC\_CNTL\_SLP\_WAKEUP\_INT\_ENA\_W1TC** Write 1 to disable interrupt when the chip wakes up from sleep. If the value 1 is written to this bit, the **RTC\_CNTL\_SLP\_WAKEUP\_INT\_CLR** field will be cleared. (R/W)

**RTC\_CNTL\_SLP\_REJECT\_INT\_ENA\_WITC** Write 1 to disable interrupt when the chip rejects to go to sleep. If the value 1 is written to this bit, the [RTC\\_CNTL\\_SLP\\_REJECT\\_INT\\_CLR](#) field will be cleared. (R/W)

**RTC\_CNTL\_WDT\_INT\_ENA\_W1TC** Write 1 to disable the RTC watchdog interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_WDT\_INT\_CLR** field will be cleared. (R/W)

**RTC\_CNTL\_BROWN\_OUT\_INT\_ENA\_W1TC** Write 1 to disable the brownout interrupt. If the value 1 is written to this bit, the [RTC\\_CNTL\\_BROWN\\_OUT\\_INT\\_CLR](#) field will be cleared. (R/W)

**RTC\_CNTL\_MAIN\_TIMER\_INT\_ENA\_W1TC** Write 1 to disable the RTC timer interrupt. If the value 1 is written to this bit, the **RTC\_CNTL\_MAIN\_TIMER\_INT\_CLR** field will be cleared. (R/W)

**RTC\_CNTL\_SWD\_INT\_ENA\_W1TC** Write 1 to disable the super watchdog interrupt enable bit. If the value 1 is written to this bit, the **RTC\_CNTL\_SWD\_INT\_CLR** field will be cleared. (R/W)

**RTC\_CNTL\_BBPLL\_CAL\_INT\_ENA\_W1TC** Write 1 to disable the interrupt upon the ending of a BB\_PLL call. If the value 1 is written to this bit, the **RTC\_CNTL\_BBPLL\_CAL\_INT\_CLR** field will be cleared. (R/W)

### Register 9.56. RTC\_CNTL\_FIB\_SEL\_REG (0x00F8)

[illegible]

**RTC\_CNTL\_RTC\_FIB\_SEL** Configures the brownout detector. (R/W)



## Chapter 10

# System Timer (SYSTIMER)

## 10.1 Overview

ESP8684 provides a 52-bit timer, which can be used to generate tick interrupts for the operating system, or be used as a general timer to generate periodic interrupts or one-time interrupts.

The timer consists of two counters UNIT0 and UNIT1. The count values can be monitored by three comparators COMP0, COMP1, and COMP2. See the timer block diagram on Figure 10.1-1.

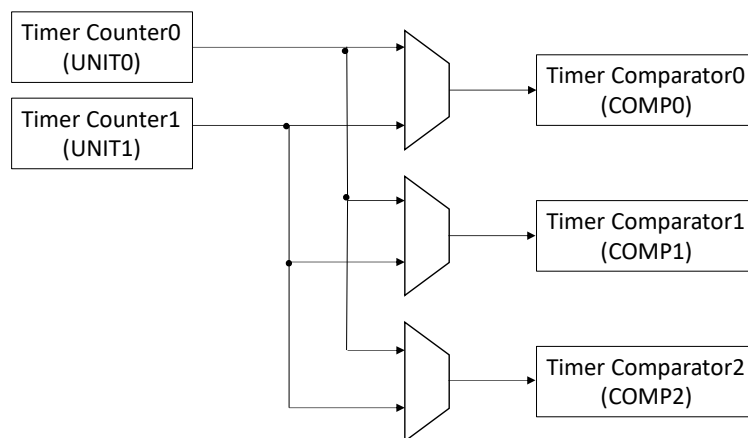


Figure 10.1-1. System Timer Structure

## 10.2 Features

The system timer has the following features:

- Two 52-bit counters and three 52-bit comparators
- Software accessing registers clocked by APB\_CLK
- Use CNT\_CLK for counting
- 52-bit alarm values ( $t$ ) and 26-bit alarm periods ( $\delta t$ )
- Two modes to generate alarms:
  - Target mode: only a one-time alarm is generated based on the alarm value ( $t$ )
  - Period mode: periodic alarms are generated based on the alarm period ( $\delta t$ )
- Three comparators generating three independent interrupts based on configured alarm value ( $t$ ) or alarm period ( $\delta t$ )

- Software configuring the reference count value. For example, the system timer is able to load back the sleep time recorded by RTC timer via software after Light-sleep
- Able to stall or continue running when CPU stalls or enters on-chip-debugging mode

## 10.3 Clock Source Selection

The counters and comparators are driven using XTAL\_CLK. After scaled by a fractional divider, a  $f_{XTAL\_CLK}/3$  clock is generated in one count cycle and a  $f_{XTAL\_CLK}/2$  clock in another count cycle. The average clock frequency is  $f_{XTAL\_CLK}/2.5$ , i.e. the CNT\_CLK in Figure 10.4-1.

Software operation such as configuring registers is clocked by APB\_CLK. For more information about APB\_CLK, see Chapter 6 [Reset and Clock](#).

The following two bits of system registers are also used to control the system timer:

- SYSTEM\_SYSTIMER\_CLK\_EN in register SYSTEM\_PERIP\_CLK\_ENO\_REG: enable APB\_CLK signal to system timer.
- SYSTEM\_SYSTIMER\_RST in register SYSTEM\_PERIP\_RST\_ENO\_REG: reset system timer.

Note that if the timer is reset, its registers will be restored to their default values. For more information, please refer to Table Peripheral Clock Gating and Reset in Chapter 13 [System Registers \(SYSTEM\)](#).

## 10.4 Functional Description

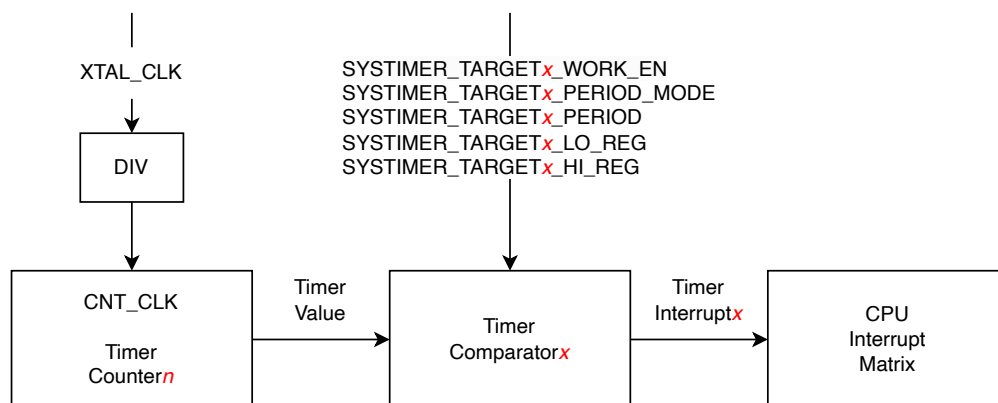


Figure 10.4-1. System Timer Alarm Generation

Figure 10.4-1 shows the procedure to generate alarm in system timer. In this process, one timer counter and one timer comparator are used. An alarm interrupt will be generated accordingly based on the comparison result in comparator.

### 10.4.1 Counter

The system timer has two 52-bit timer counters, shown as UNIT $n$  ( $n = 0$  or  $1$ ). Their counting clock source is CNT\_CLK. Whether UNIT $n$  works or not is controlled by two bits in register [SYSTIMER\\_CONF\\_REG](#):

- `SYSTIMER_TIMER_UNIT $n$ _WORK_EN`: set this bit to enable the counter UNIT $n$  in system timer.
- `SYSTIMER_TIMER_UNIT $n$ _COREO_STALL_EN`: if this bit is set, the counter UNIT $n$  stops when CPU is stalled. The counter continues its counting after the CPU resumes.

The configuration of the two bits to control the counter UNIT $n$  is shown below, assuming that CPU is stalled.

**Table 10.4-1. UNIT $n$  Configuration Bits**

<code>SYSTIMER_TIMER_UNIT<math>n</math>_WORK_EN</code>	<code>SYSTIMER_TIMER_UNIT<math>n</math>_COREO_STALL_EN</code>	Counter UNIT $n$
0	x <sup>*</sup>	Not at work
1	1	Stop counting, but will continue its counting after the CPU resumes.
1	0	Keep counting

<sup>\*</sup> x: Don't-care.

When the counter UNIT $n$  is at work, the count value is incremented on each counting cycle. When the counter UNIT $n$  is stopped, the count value stops increasing and keeps unchanged.

The lower 32 and higher 20 bits of initial count value are loaded from the registers `SYSTIMER_TIMER_UNIT $n$ _LOAD_LO` and `SYSTIMER_TIMER_UNIT $n$ _LOAD_HI`. Writing 1 to the bit `SYSTIMER_TIMER_UNIT $n$ _LOAD` will trigger a reload event, and the current count value will be changed immediately. If UNIT $n$  is at work, the counter will continue to count up from the new reloaded value.

Writing 1 to `SYSTIMER_TIMER_UNIT $n$ _UPDATE` will trigger an update event. The lower 32 and higher 20 bits of current count value will be locked into the registers `SYSTIMER_TIMER_UNIT $n$ _VALUE_LO` and `SYSTIMER_TIMER_UNIT $n$ _VALUE_HI`, and then `SYSTIMER_TIMER_UNIT $n$ _VALUE_VALID` is asserted. Before the next update event, the values of `SYSTIMER_TIMER_UNIT $n$ _VALUE_LO` and `SYSTIMER_TIMER_UNIT $n$ _VALUE_HI` remain unchanged.

## 10.4.2 Comparator and Alarm

The system timer has three 52-bit comparators, shown as COMP $x$  ( $x = 0, 1$ , or  $2$ ). The comparators can generate independent interrupts based on different alarm values ( $t$ ) or alarm periods ( $\delta t$ ).

Configure `SYSTIMER_TARGET $x$ _PERIOD_MODE` to choose from the two alarm modes for each COMP $x$ :

- 1: period mode
- 0: target mode

In period mode, the alarm period ( $\delta t$ ) is provided by the register `SYSTIMER_TARGET $x$ _PERIOD`. Assuming that current count value is  $t_1$ , when it reaches  $(t_1 + \delta t)$ , an alarm interrupt will be generated. Another alarm interrupt also will be generated when the count value reaches  $(t_1 + 2 * \delta t)$ . By such way, periodic alarms are generated.

In target mode, the lower 32 bits and higher 20 bits of the alarm value ( $t$ ) are provided by `SYSTIMER_TIMER_TARGET $x$ _LO` and `SYSTIMER_TIMER_TARGET $x$ _HI`. Assuming that current count value is  $t_2$  ( $t_2 \leq t$ ), an alarm interrupt will be generated when the count value reaches the alarm value ( $t$ ). Unlike in period mode, only one alarm

interrupt is generated in target mode.

`SYSTIMER_TARGETx_TIMER_UNIT_SEL` is used to choose the count value from which timer counter to be compared for alarm:

- 1: use the count value from UNIT<sub>1</sub>
- 0: use the count value from UNIT<sub>0</sub>

Finally, set `SYSTIMER_TARGETx_WORK_EN` and `COMPx` starts to compare the count value with the alarm value (t) in target mode or with the alarm period ( $t_1 + n \cdot \delta t$ ) in period mode.

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value +  $n \cdot \text{alarm period } \delta t$  ( $n = 1, 2, 3, \dots$ ) in period mode. But if the alarm value (t) set in registers is less than current count value, i.e. the target has already passed, or current count value is larger than the target value (t) within a range ( $0 \sim 2^{51} - 1$ ), an alarm interrupt also is generated immediately. The relationship between current count value  $t_c$ , the alarm value  $t_t$  and alarm trigger point is shown below.

Table 10.4-2. Trigger Point

Relationship Between $t_c$ and $t_t$	Trigger Point
$t_c - t_t \leq 0$	$t_c = t_t$ , an alarm is triggered.
$0 \leq t_c - t_t < 2^{51} - 1$ ( $t_c < 2^{51}$ and $t_t < 2^{51}$ , or $t_c \geq 2^{51}$ and $t_t \geq 2^{51}$ )	An alarm is triggered immediately.
$t_c - t_t \geq 2^{51} - 1$	$t_c$ overflows after counting to its maximum value 52'hffffffffffff, and then starts counting up from 0. When its value reaches $t_t$ , an alarm is triggered.

### 10.4.3 Synchronization Operation

The clock (APB\_CLK) used in software operation is not the same one as the timer counters and comparators working on CNT\_CLK. Synchronization is needed for some configuration registers. A complete synchronization action takes two steps:

1. Software writes suitable values to configuration fields, see the first column in Table 10.4-3.
2. Software writes 1 to corresponding bits to start synchronization, see the second column in Table 10.4-3.

Table 10.4-3. Synchronization Operation

Configuration Fields	Synchronization Enable Bit
<code>SYSTIMER_TIMER_UNIT<sub>n</sub>_LOAD_LO</code> <code>SYSTIMER_TIMER_UNIT<sub>n</sub>_LOAD_HI</code>	<code>SYSTIMER_TIMER_UNIT<sub>n</sub>_LOAD</code>
<code>SYSTIMER_TARGET<sub>x</sub>_PERIOD</code> <code>SYSTIMER_TIMER_TARGET<sub>x</sub>_HI</code> <code>SYSTIMER_TIMER_TARGET<sub>x</sub>_LO</code>	<code>SYSTIMER_TIMER_COMP<sub>x</sub>_LOAD</code>

### 10.4.4 Interrupt

Each comparator has one level-type alarm interrupt, named as SYSTIMER\_TARGET $x$ \_INT. Interrupts signal is asserted high when the comparator starts to alarm. Until the interrupt is cleared by software, it remains high. To enable interrupts, set the bit [SYSTIMER\\_TARGET \$x\$ \\_INT\\_ENA](#).

## 10.5 Programming Procedure

When configuring COMP $x$  and UNIT $n$ , please ensure the corresponding COMP and UNIT are at work.

### 10.5.1 Read Current Count Value

1. Set [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_UPDATE](#) to update the current count value into [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_HI](#) and [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_LO](#).
2. Poll the reading of [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_VALID](#), till it's 1, which means user now can read the count value from [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_HI](#) and [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_LO](#).
3. Read the lower 32 bits and higher 20 bits from [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_LO](#) and [SYSTIMER\\_TIMER\\_UNIT \$n\$ \\_VALUE\\_HI](#).

### 10.5.2 Configure One-Time Alarm in Target Mode

1. Set [SYSTIMER\\_TARGET \$x\$ \\_TIMER\\_UNIT\\_SEL](#) to select the counter (UNIT $0$  or UNIT $1$ ) used for COMP $x$ .
2. Read current count value, see Section 10.5.1. This value will be used to calculate the alarm value (t) in Step 4.
3. Clear [SYSTIMER\\_TARGET \$x\$ \\_PERIOD\\_MODE](#) to enable target mode.
4. Set an alarm value (t), and fill its lower 32 bits to [SYSTIMER\\_TIMER\\_TARGET \$x\$ \\_LO](#), and the higher 20 bits to [SYSTIMER\\_TIMER\\_TARGET \$x\$ \\_HI](#).
5. Set [SYSTIMER\\_TIMER\\_COMP \$x\$ \\_LOAD](#) to synchronize the alarm value (t) to COMP $x$ , i.e. load the alarm value (t) to the COMP $x$ .
6. Set [SYSTIMER\\_TARGET \$x\$ \\_WORK\\_EN](#) to enable the selected COMP $x$ . COMP $x$  starts comparing the count value with the alarm value (t).
7. Set [SYSTIMER\\_TARGET \$x\$ \\_INT\\_ENA](#) to enable timer interrupt. When Unit $n$  counts to the alarm value (t), a SYSTIMER\_TARGET $x$ \_INT interrupt is triggered.

### 10.5.3 Configure Periodic Alarms in Period Mode

1. Set [SYSTIMER\\_TARGET \$x\$ \\_TIMER\\_UNIT\\_SEL](#) to select the counter (UNIT $0$  or UNIT $1$ ) used for COMP $x$ .
2. Set an alarm period ( $\delta t$ ), and fill it to [SYSTIMER\\_TARGET \$x\$ \\_PERIOD](#).
3. Set [SYSTIMER\\_TIMER\\_COMP \$x\$ \\_LOAD](#) to synchronize the alarm period ( $\delta t$ ) to COMP $x$ , i.e. load the alarm period ( $\delta t$ ) to COMP $x$ .
4. Clear and then set [SYSTIMER\\_TARGET \$x\$ \\_PERIOD\\_MODE](#) to configure COMP $x$  into period mode.

- Set `SYSTIMER_TARGET $x$ _WORK_EN` to enable the selected COMP $x$ . COMP $x$  starts comparing the count value with the sum of start value +  $n \cdot \delta t$  ( $n = 1, 2, 3 \dots$ ).
- Set `SYSTIMER_TARGET $x$ _INT_ENA` to enable timer interrupt. A SYSTIMER\_TARGET $x$ \_INT interrupt is triggered when Unit $n$  counts to start value +  $n \cdot \delta t$  ( $n = 1, 2, 3 \dots$ ) set in Step 2.

### 10.5.4 Update After Light-sleep

- Configure the RTC timer before the chip goes to Light-sleep, to record the exact sleep time. For more information, see Chapter 9 *Low-power Management (RTC\_CNTL)*.
- Read the sleep time from the RTC timer when the chip is woken up from Light-sleep.
- Read current count value of system timer, see Section 10.5.1.
- Convert the time value recorded by the RTC timer from the clock cycles based on RTC\_SLOW\_CLK to that based on CNT\_CLK. For example, if the frequency of RTC\_SLOW\_CLK is 32 KHz, the recorded RTC timer value should be converted by multiplying by 500.
- Add the converted RTC value to the current count value of the system timer:
  - Fill the new value into `SYSTIMER_TIMER_UNIT $n$ _LOAD_LO` (low 32 bits) and `SYSTIMER_TIMER_UNIT $n$ _LOAD_HI` (high 20 bits).
  - Set `SYSTIMER_TIMER_UNIT $n$ _LOAD` to load new timer value into system timer. In such a way, the system timer is updated.

## 10.6 Register Summary

The addresses in this section are relative to system timer base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Clock Control Register</b>			
<code>SYSTIMER_CONF_REG</code>	Configure system timer clock	0x0000	R/W
<b>UNIT0 Control and Configuration Registers</b>			
<code>SYSTIMER_UNIT0_OP_REG</code>	Read UNIT0 value to registers	0x0004	varies
<code>SYSTIMER_UNIT0_LOAD_HI_REG</code>	High 20 bits to be loaded to UNIT0	0x000C	R/W
<code>SYSTIMER_UNIT0_LOAD_LO_REG</code>	Low 32 bits to be loaded to UNIT0	0x0010	R/W
<code>SYSTIMER_UNIT0_VALUE_HI_REG</code>	UNIT0 value, high 20 bits	0x0040	RO
<code>SYSTIMER_UNIT0_VALUE_LO_REG</code>	UNIT0 value, low 32 bits	0x0044	RO
<code>SYSTIMER_UNIT0_LOAD_REG</code>	UNIT0 synchronization register	0x005C	WT
<b>UNIT1 Control and Configuration Registers</b>			
<code>SYSTIMER_UNIT1_OP_REG</code>	Read UNIT1 value to registers	0x0008	varies
<code>SYSTIMER_UNIT1_LOAD_HI_REG</code>	High 20 bits to be loaded to UNIT1	0x0014	R/W
<code>SYSTIMER_UNIT1_LOAD_LO_REG</code>	Low 32 bits to be loaded to UNIT1	0x0018	R/W
<code>SYSTIMER_UNIT1_VALUE_HI_REG</code>	UNIT1 value, high 20 bits	0x0048	RO
<code>SYSTIMER_UNIT1_VALUE_LO_REG</code>	UNIT1 value, low 32 bits	0x004C	RO

Name	Description	Address	Access
<a href="#">SYSTIMER_UNIT1_LOAD_REG</a>	UNIT1 synchronization register	0x0060	WT
<b>Comparator0 Control and Configuration Registers</b>			
<a href="#">SYSTIMER_TARGET0_HI_REG</a>	Alarm value to be loaded to COMP0, high 20 bits	0x001C	R/W
<a href="#">SYSTIMER_TARGET0_LO_REG</a>	Alarm value to be loaded to COMP0, low 32 bits	0x0020	R/W
<a href="#">SYSTIMER_TARGET0_CONF_REG</a>	Configure COMP0 alarm mode	0x0034	R/W
<a href="#">SYSTIMER_COMP0_LOAD_REG</a>	COMP0 synchronization register	0x0050	WT
<b>Comparator1 Control and Configuration Registers</b>			
<a href="#">SYSTIMER_TARGET1_HI_REG</a>	Alarm value to be loaded to COMP1, high 20 bits	0x0024	R/W
<a href="#">SYSTIMER_TARGET1_LO_REG</a>	Alarm value to be loaded to COMP1, low 32 bits	0x0028	R/W
<a href="#">SYSTIMER_TARGET1_CONF_REG</a>	Configure COMP1 alarm mode	0x0038	R/W
<a href="#">SYSTIMER_COMP1_LOAD_REG</a>	COMP1 synchronization register	0x0054	WT
<b>Comparator2 Control and Configuration Registers</b>			
<a href="#">SYSTIMER_TARGET2_HI_REG</a>	Alarm value to be loaded to COMP2, high 20 bits	0x002C	R/W
<a href="#">SYSTIMER_TARGET2_LO_REG</a>	Alarm value to be loaded to COMP2, low 32 bits	0x0030	R/W
<a href="#">SYSTIMER_TARGET2_CONF_REG</a>	Configure COMP2 alarm mode	0x003C	R/W
<a href="#">SYSTIMER_COMP2_LOAD_REG</a>	COMP2 synchronization register	0x0058	WT
<b>Interrupt Registers</b>			
<a href="#">SYSTIMER_INT_ENA_REG</a>	Interrupt enable register of system timer	0x0064	R/W
<a href="#">SYSTIMER_INT_RAW_REG</a>	Interrupt raw register of system timer	0x0068	R/WTC/SS
<a href="#">SYSTIMER_INT_CLR_REG</a>	Interrupt clear register of system timer	0x006C	WT
<a href="#">SYSTIMER_INT_ST_REG</a>	Interrupt status register of system timer	0x0070	RO
<b>Version Register</b>			
<a href="#">SYSTIMER_DATE_REG</a>	Version control register	0x00FC	R/W

## 10.7 Registers

The addresses in this section are relative to system timer base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

### Register 10.1. SYSTIMER\_CONF\_REG (0x0000)

[illegible]

SYSTIMER TARGET2 WORK EN COMP2 work enable bit. (R/W)

**SYSTIMER\_TARGET1\_WORK\_EN** COMP1 work enable bit. (R/W)

**SYSTIMER\_TARGET0\_WORK\_EN** COMPO work enable bit. (R/W)

**SYSTIMER\_TIMER\_UNIT1\_CORE0\_STALL\_EN** UNIT1 is stalled when CPU stalled. (R/W)

**SYSTIMER TIMER UNIT0 CORE0 STALL EN** UNIT0 is stalled when CPU stalled. (R/W)

**SYSTIMER\_TIMER\_UNIT1\_WORK\_EN** UNIT1 work enable bit. (R/W)

**SYSTIMER\_TIMER\_UNIT0\_WORK\_EN** UNIT0 work enable bit. (R/W)

**SYSTIMER\_CLK\_EN** Register clock gating. 1: Register clock is always enabled for read and write operations. 0: Only enable needed clock for register read or write operations. (R/W)



### Register 10.2. SYSTIMER\_UNIT0\_OP\_REG (0x0004)

Diagram of the SYSTEM\_TIMER\_UNIT\_VALUE\_VALID register. The register is 32 bits wide. Bits 31, 30, and 29 are labeled (reserved). Bit 28 is labeled SYSTEM\_TIMER\_UNIT\_UPDATE. Bits 27 through 0 are labeled (reserved). The register is shown with a value of 0 in all bits.

**SYSTIMER\_TIMER\_UNIT0\_VALUE\_VALID** Timer value is synchronized and valid. (R/SS/WTC)

**SYSTIMER\_TIMER\_UNITO\_UPDATE** Update timer UNITO, i.e. read the UNITO count value to SYSTIMER\_TIMER\_UNITO\_VALUE\_HI and SYSTIMER\_TIMER\_UNITO\_VALUE\_LO. (WT)

### Register 10.3. SYSTIMER\_UNIT0\_LOAD\_HI\_REG (0x000C)

Diagram illustrating the structure of the SYSTIMER\_TMR\_LOAD register (32 bits):

- Bits 31 to 20: (reserved)
- Bits 19 to 0: SYSTIMER\_TMR\_LOAD\_HI
- Bit 0: Reset

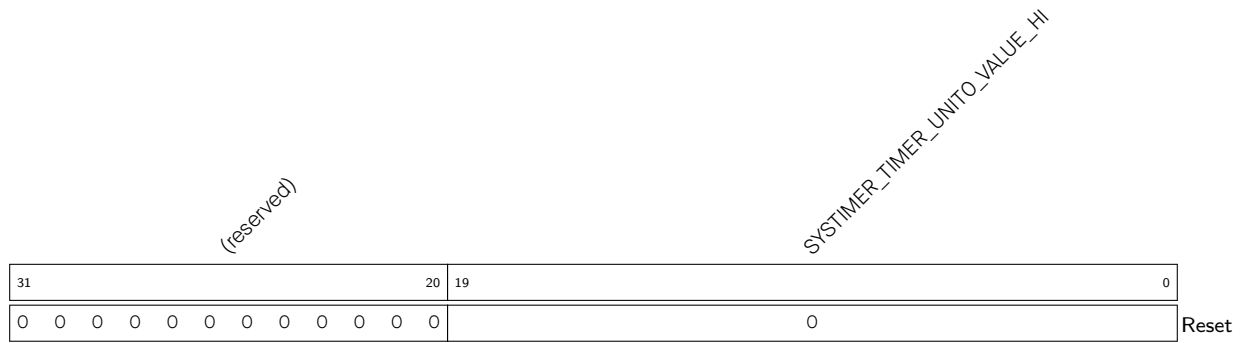
**SYSTIMER\_TIMER\_UNITO\_LOAD\_HI** The value to be loaded to UNITO, high 20 bits. (R/W)

#### Register 10.4. SYSTIMER\_UNIT0\_LOAD\_LO\_REG (0x0010)

Diagram of the SYSTIMER\_TMR\_LOAD register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. Bit 31 is labeled 'SYSTIMER\_TMR\_LOAD\_LOCK'. Bit 0 is labeled 'Reset'.

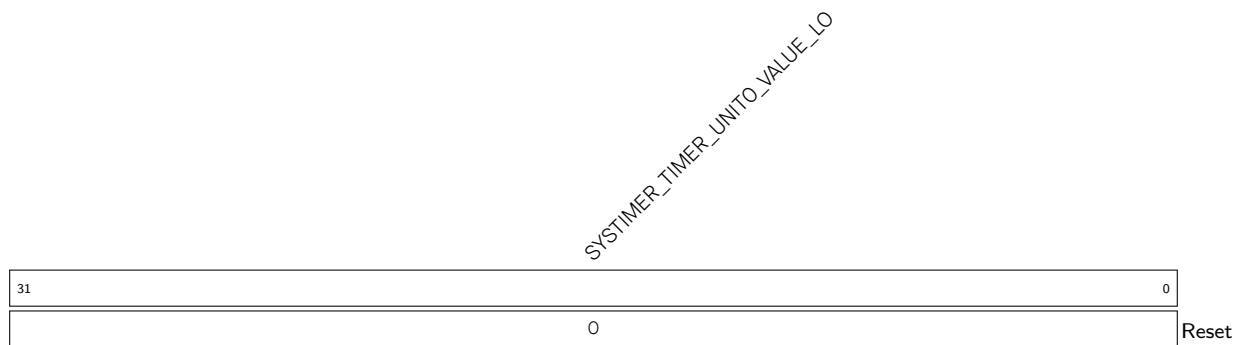
**SYSTIMER\_TIMER\_UNIT0\_LOAD\_LO** The value to be loaded to UNIT0, low 32 bits. (R/W)

Register 10.5. SYSTIMER\_UNITO\_VALUE\_HI\_REG (0x0040)



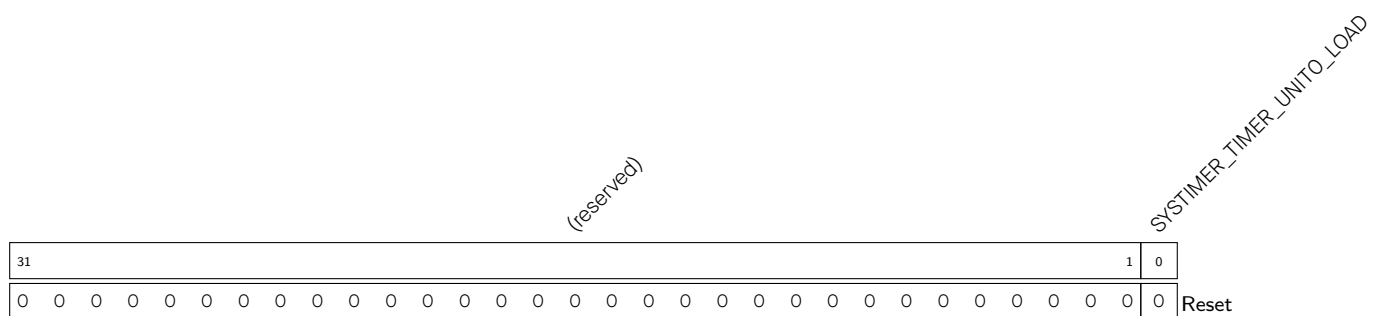
**SYSTIMER\_TIMER\_UNITO\_VALUE\_HI** UNITO read value, high 20 bits. (RO)

Register 10.6. SYSTIMER\_UNITO\_VALUE\_LO\_REG (0x0044)



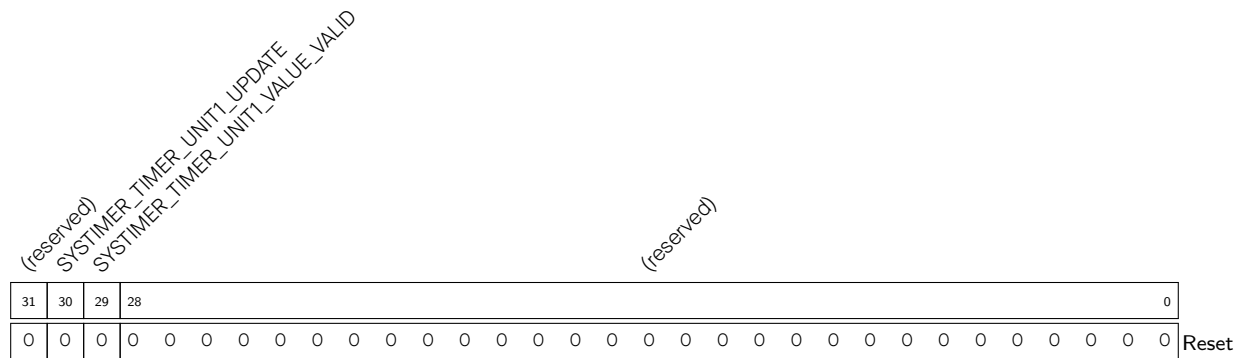
**SYSTIMER\_TIMER\_UNITO\_VALUE\_LO** UNITO read value, low 32 bits. (RO)

Register 10.7. SYSTIMER\_UNITO\_LOAD\_REG (0x005C)



**SYSTIMER\_TIMER\_UNITO\_LOAD** UNITO synchronization enable signal. Set this bit to reload the values of SYSTIMER\_TIMER\_UNITO\_LOAD\_HI and SYSTIMER\_TIMER\_UNITO\_LOAD\_LO to UNITO. (WT)

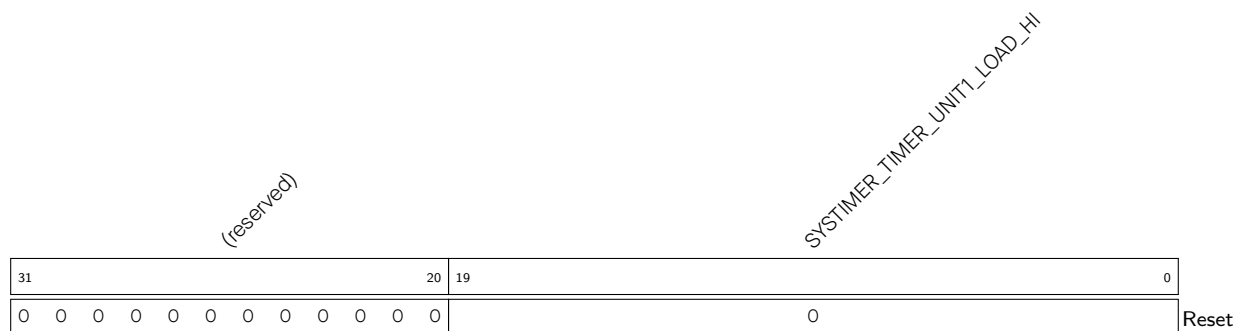
### Register 10.8. SYSTIMER\_UNIT1\_OP\_REG (0x0008)



**SYSTIMER\_TIMER\_UNIT1\_VALUE\_VALID** UNIT1 value is synchronized and valid. (R/SS/WTC)

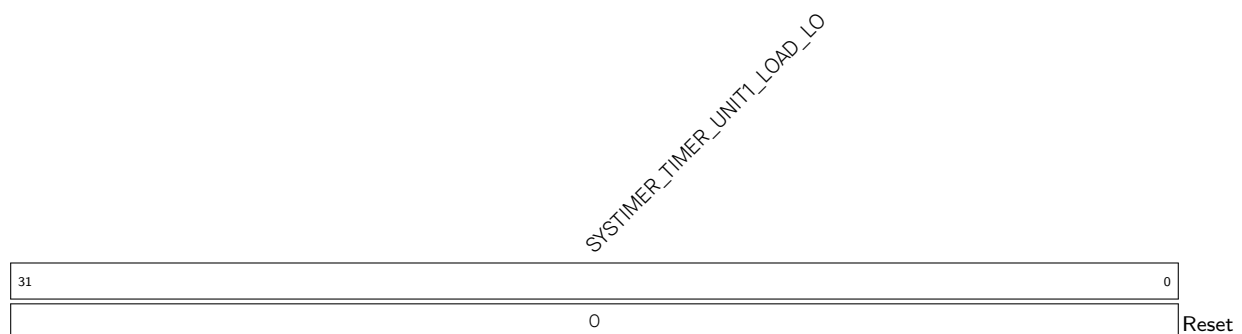
**SYSTIMER\_TIMER\_UNIT1\_UPDATE** Update timer UNIT1, i.e. read the UNIT1 count value to SYSTIMER\_TIMER\_UNIT1\_VALUE\_HI and SYSTIMER\_TIMER\_UNIT1\_VALUE\_LO. (WT)

### Register 10.9. SYSTIMER\_UNIT1\_LOAD\_HI\_REG (0x0014)



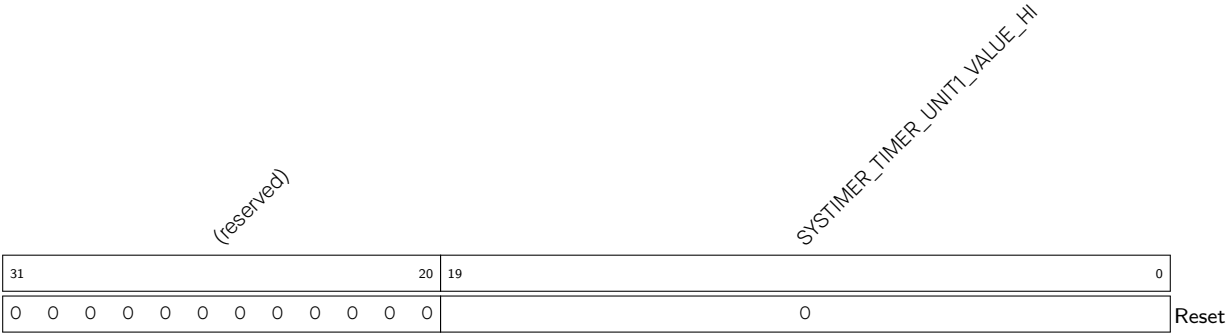
**SYSTIMER\_TIMER\_UNIT1\_LOAD\_HI** The value to be loaded to UNIT1, high 20 bits. (R/W)

### Register 10.10. SYSTIMER\_UNIT1\_LOAD\_LO\_REG (0x0018)



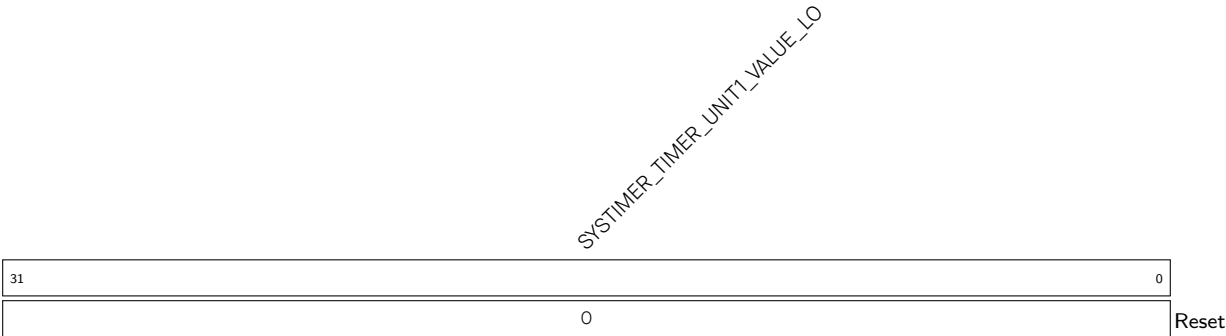
**SYSTIMER\_TIMER\_UNIT1\_LOAD\_LO** The value to be loaded to UNIT1, low 32 bits. (R/W)

Register 10.11. SYSTIMER\_UNIT1\_VALUE\_HI\_REG (0x0048)



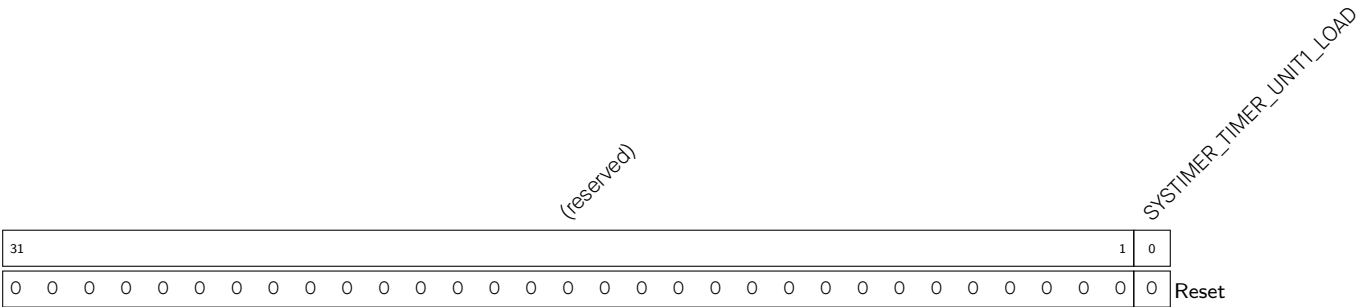
**SYSTIMER\_TIMER\_UNIT1\_VALUE\_HI** UNIT1 read value, high 20 bits. (RO)

Register 10.12. SYSTIMER\_UNIT1\_VALUE\_LO\_REG (0x004C)



**SYSTIMER\_TIMER\_UNIT1\_VALUE\_LO** UNIT1 read value, low 32 bits. (RO)

Register 10.13. SYSTIMER\_UNIT1\_LOAD\_REG (0x0060)



**SYSTIMER\_TIMER\_UNIT1\_LOAD** UNIT1 synchronization enable signal. Set this bit to reload the values of SYSTIMER\_TIMER\_UNIT1\_LOAD\_HI and SYSTIMER\_TIMER\_UNIT1\_LOAD\_LO to UNIT1. (WT)

Register 10.14. SYSTIMER\_TARGETO\_HI\_REG (0x001C)

(reserved)												SYSTIMER_TIMER_TARGETO_HI																				
31												20	19																			0
0	0	0	0	0	0	0	0	0	0	0	0	0																			Reset	

**SYSTIMER\_TIMER\_TARGETO\_HI** The alarm value to be loaded to COMPO, high 20 bits. (R/W)

Register 10.15. SYSTIMER\_TARGETO\_LO\_REG (0x0020)

SYSTIMER_TIMER_TARGETO_LO																															
31																															0
0																															Reset

**SYSTIMER\_TIMER\_TARGETO\_LO** The alarm value to be loaded to COMPO, low 32 bits. (R/W)

Register 10.16. SYSTIMER\_TARGETO\_CONF\_REG (0x0034)

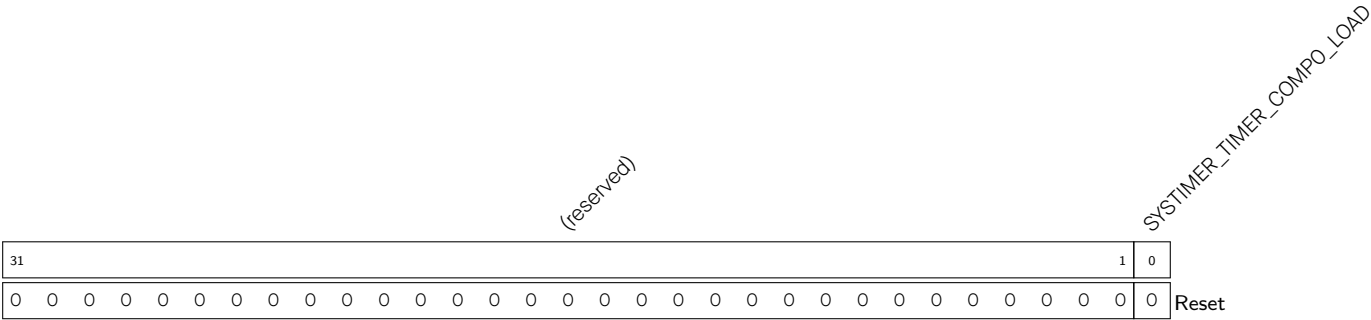
SYSTIMER_TARGETO_TIMER_UNIT_SEL										SYSTIMER_TARGETO_PERIOD_MODE										SYSTIMER_TARGETO_PERIOD									
(reserved)																													
31	30	29				26				25				0															
0	0	0	0	0	0	0x00000																Reset							

**SYSTIMER\_TARGETO\_PERIOD** COMPO alarm period. (R/W)

**SYSTIMER\_TARGETO\_PERIOD\_MODE** Set COMPO to period mode. (R/W)

**SYSTIMER\_TARGETO\_TIMER\_UNIT\_SEL** Select which unit to compare for COMPO. (R/W)

Register 10.17. SYSTIMER\_COMPO\_LOAD\_REG (0x0050)



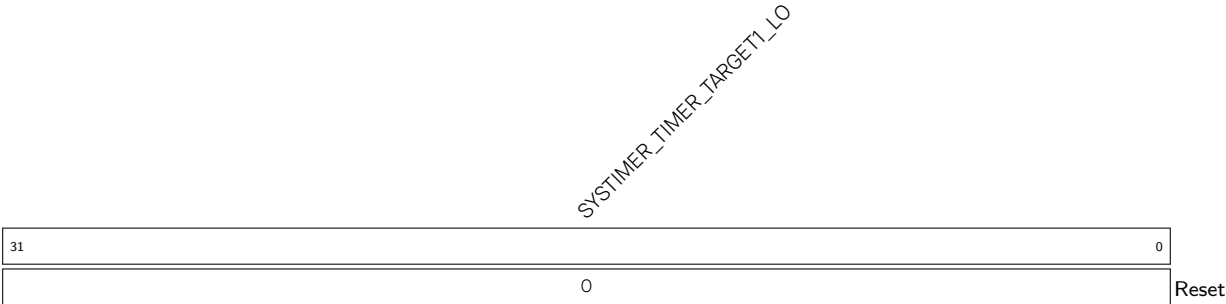
**SYSTIMER\_TIMER\_COMPO\_LOAD** COMPO synchronization enable signal. Set this bit to reload the alarm value/period to COMPO. (WT)

Register 10.18. SYSTIMER\_TARGET1\_HI\_REG (0x0024)



**SYSTIMER\_TIMER\_TARGET1\_HI** The alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 10.19. SYSTIMER\_TARGET1\_LO\_REG (0x0028)



**SYSTIMER\_TIMER\_TARGET1\_LO** The alarm value to be loaded to COMP1, low 32 bits. (R/W)

### Register 10.20. SYSTIMER\_TARGET1\_CONF\_REG (0x0038)

SYSTIMER\_TARGET1\_TIMER\_UNIT\_SEL

SYSTIMER\_TARGET1\_PERIOD\_MODE

(reserved)

SYSTIMER\_TARGET1\_PERIOD

0x000000

**SYSTIMER\_TARGET1\_PERIOD** COMP1 alarm period. (R/W)

**SYSTIMER\_TARGET1\_PERIOD\_MODE** Set COMP1 to period mode. (R/W)

**SYSTIMER\_TARGET1\_TIMER\_UNIT\_SEL** Select which unit to compare for COMP1. (R/W)

### Register 10.21. SYSTIMER\_COMP1\_LOAD\_REG (0x0054)

[illegible]

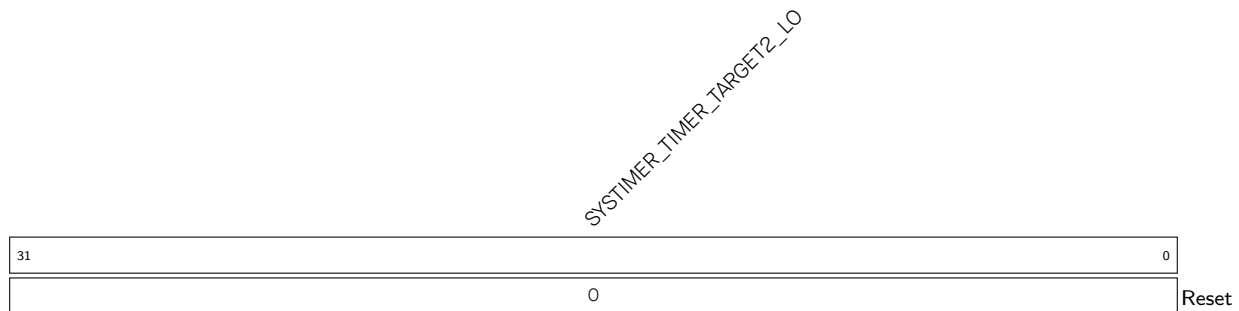
**SYSTIMER\_TIMER\_COMP1\_LOAD** COMP1 synchronization enable signal. Set this bit to reload the alarm value/period to COMP1. (WT)

### Register 10.22. SYSTIMER\_TARGET2\_HI\_REG (0x002C)

31	20	19	0
0 0 0 0 0 0 0 0 0 0 0 0 0			

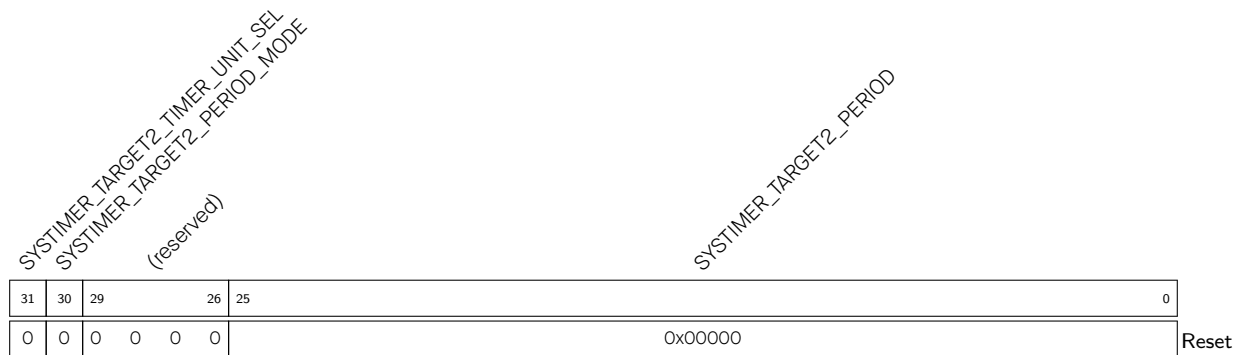
**SYSTIMER\_TIMER\_TARGET2\_HI** The alarm value to be loaded to COMP2, high 20 bits. (R/W)

### Register 10.23. SYSTIMER\_TARGET2\_LO\_REG (0x0030)



**SYSTIMER\_TIMER\_TARGET2\_LO** The alarm value to be loaded to COMP2, low 32 bits. (R/W)

### Register 10.24. SYSTIMER\_TARGET2\_CONF\_REG (0x003C)

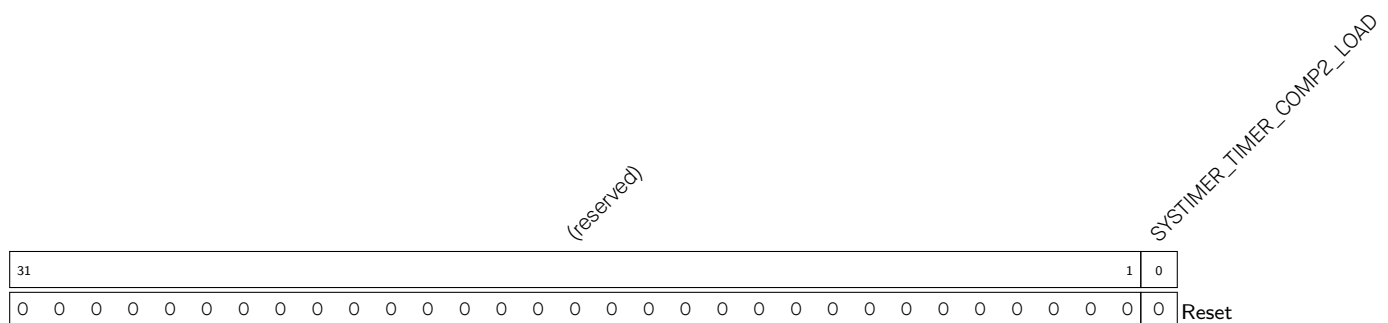


**SYSTIMER\_TARGET2\_PERIOD** COMP2 alarm period. (R/W)

**SYSTIMER\_TARGET2\_PERIOD\_MODE** Set COMP2 to period mode. (R/W)

**SYSTIMER\_TARGET2\_TIMER\_UNIT\_SEL** Select which unit to compare for COMP2. (R/W)

### Register 10.25. SYSTIMER\_COMP2\_LOAD\_REG (0x0058)



**SYSTIMER\_TIMER\_COMP2\_LOAD** COMP2 synchronization enable signal. Set this bit to reload the alarm value/period to COMP2. (WT)



### Register 10.26. SYSTIMER\_INT\_ENA\_REG (0x0064)

[illegible]

**SYSTIMER\_TARGET0\_INT\_ENA** SYSTIMER\_TARGET0\_INT enable bit. (R/W)

**SYSTIMER\_TARGET1\_INT\_ENA** SYSTIMER\_TARGET1\_INT enable bit. (R/W)

**SYSTIMER\_TARGET2\_INT\_ENA** SYSTIMER\_TARGET2\_INT enable bit. (R/W)

### Register 10.27. SYSTIMER\_INT\_RAW\_REG (0x0068)

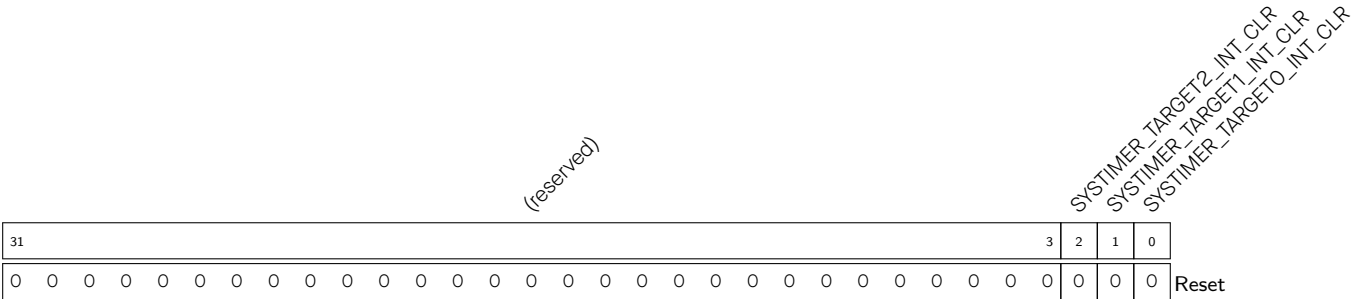
[illegible]

**SYSTIMER\_TARGET0\_INT\_RAW** SYSTIMER\_TARGET0\_INT raw bit. (R/WTC/SS)

**SYSTIMER\_TARGET1\_INT\_RAW** SYSTIMER\_TARGET1\_INT raw bit. (R/WTC/SS)

**SYSTIMER\_TARGET2\_INT\_RAW** SYSTIMER\_TARGET2\_INT raw bit. (R/WTC/SS)

Register 10.28. SYSTIMER\_INT\_CLR\_REG (0x006C)

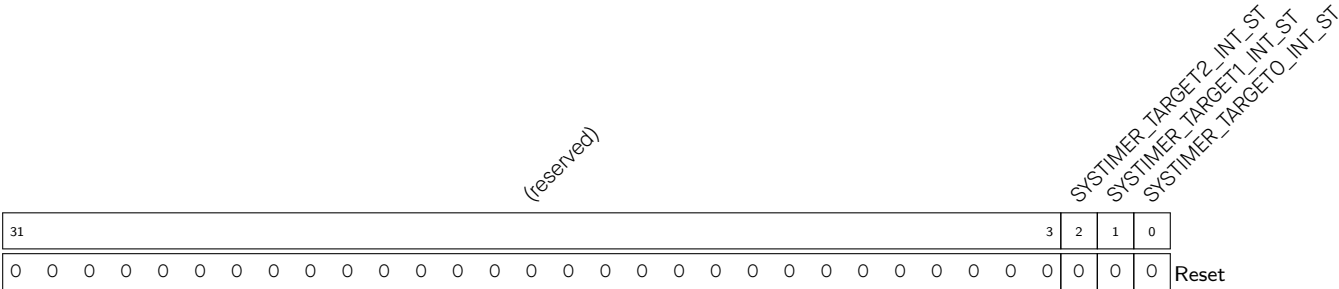


**SYSTIMER\_TARGET0\_INT\_CLR** SYSTIMER\_TARGET0\_INT clear bit. (WT)

**SYSTIMER\_TARGET1\_INT\_CLR** SYSTIMER\_TARGET1\_INT clear bit. (WT)

**SYSTIMER\_TARGET2\_INT\_CLR** SYSTIMER\_TARGET2\_INT clear bit. (WT)

Register 10.29. SYSTIMER\_INT\_ST\_REG (0x0070)

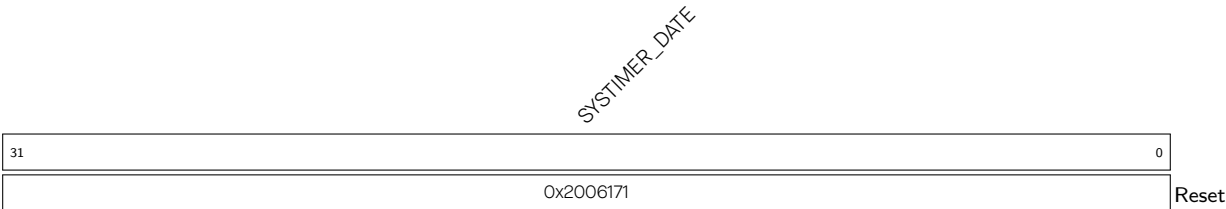


**SYSTIMER\_TARGET0\_INT\_ST** SYSTIMER\_TARGET0\_INT status bit. (RO)

**SYSTIMER\_TARGET1\_INT\_ST** SYSTIMER\_TARGET1\_INT status bit. (RO)

**SYSTIMER\_TARGET2\_INT\_ST** SYSTIMER\_TARGET2\_INT status bit. (RO)

Register 10.30. SYSTIMER\_DATE\_REG (0x00FC)



**SYSTIMER\_DATE** Version control register. (R/W)

## Chapter 11

### Timer Group (TIMG)

#### 11.1 Overview

General-purpose timers can be used to precisely time an interval, trigger an interrupt after a particular interval (periodically and aperiodically), or act as a hardware clock. As shown in Figure 11.1-1, the ESP8684 chip contains one timer group, namely timer group 0. The timer group consists of one general-purpose timer referred to as T0 and one Main System Watchdog Timer. The general-purpose timer is based on a 16-bit prescaler and a 54-bit auto-reload-capable up-down counter.

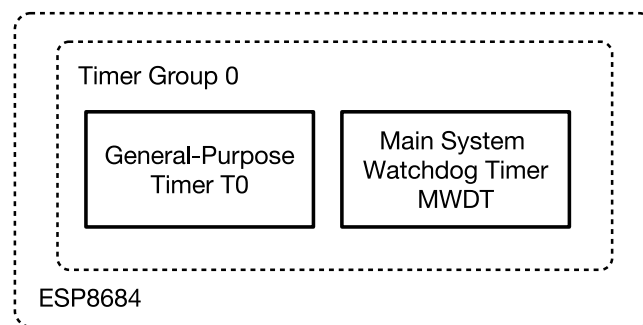


Figure 11.1-1. Timer Group Overview

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 12 [Watchdog Timers \(WDT\)](#). Therefore, the term ‘timer’ within this chapter refers to the general-purpose timer.

#### 11.2 Features

The timer’s features are summarized as follows:

- A 54-bit time-base counter programmable to incrementing or decrementing
- Two clock sources: 40 MHz PLL\_F40M\_CLK or XTAL\_CLK
- A 16-bit clock prescaler, from 2 to 65536
- Able to read real-time value of the time-base counter
- Able to halt and resume the time-base counter
- Programmable alarm generation
- Timer value reload — Auto-reload at alarm or software-controlled instant reload
- RTC slow clock RTC\_SLOW\_CLK frequency calculation
- Level interrupt generation

## 11.3 Functional Description

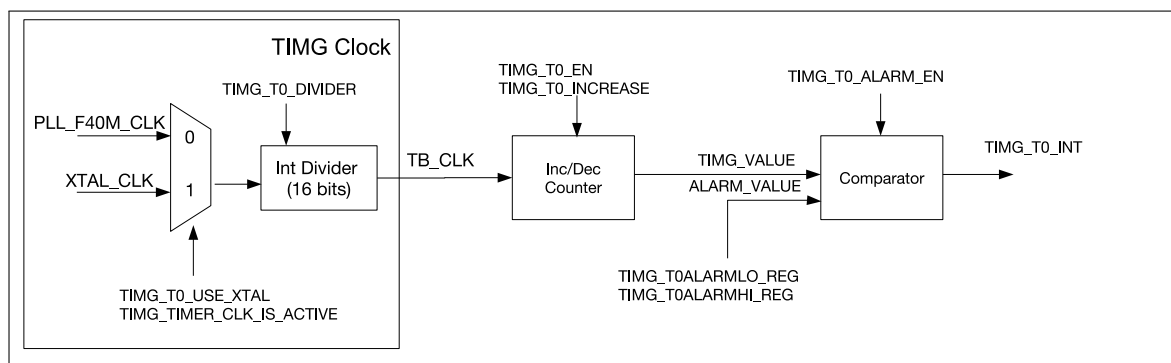


Figure 11.3-1. Timer Group Architecture

Figure 11.3-1 is a diagram of timer T0 in a timer group. T0 contains a clock selector, a 16-bit integer divider as a prescaler, a timer-based counter and a comparator for alarm generation.

### 11.3.1 16-bit Prescaler and Clock Selection

The timer can select between the PLL\_F40M\_CLK clock or external clock (XTAL\_CLK) as its clock source by setting the `TIMG_TO_USE_XTAL` field of the `TIMG_TOCONFIG_REG` register. Note that when the chip is in low-power mode and the clock source of CPU\_CLK is not PLL\_CLK (i.e. when `SYSTEM_SOC_CLK_SEL` is not 1, see details in Table 6.2-1 of Chapter 6 *Reset and Clock*), the timer can only select XTAL\_CLK.

The selected clock can be switched on by setting `TIMG_TIMER_CLK_IS_ACTIVE` field of the `TIMG_REGCLK_REG` register to 1 and switched off by setting it to 0. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB\_CLK) used by the time-base counter. When the `TIMG_TO_DIVIDER` field is configured as 2 ~ 65536, the divisor of the prescaler would be 2 ~ 65536. Note that programming value 0 to `TIMG_TO_DIVIDER` will result in the divisor being 65536. When the `TIMG_TO_DIVIDER` is set to 1, the actual divisor is 2 so the timer counter value represents the half of real time.

To modify the 16-bit prescaler, please first configure the `TIMG_TO_DIVIDER` field, and then set `TIMG_TO_DIVIDER_RST` to 1. Meanwhile, the timer must be disabled (i.e. `TIMG_TO_EN` should be cleared). Otherwise, the result can be unpredictable.

### 11.3.2 54-bit Time-base Counter

The 54-bit time-base counter is based on TB\_CLK and can be configured to increment or decrement via the `TIMG_TO_INCREASE` field. The time-base counter can be enabled or disabled by setting or clearing the `TIMG_TO_EN` field, respectively. When enabled, the time-base counter increments or decrements on each cycle of TB\_CLK. When disabled, the time-base counter is essentially frozen. Note that the `TIMG_TO_INCREASE` field can be changed no matter whether `TIMG_TO_EN` is set or not, and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the `TIMG_TOUUPDATE_REG`, the current value of the 54-bit timer starts to be latched into the `TIMG_TOLO_REG` and `TIMG_TOHI_REG` registers containing the lower 32-bits and higher 22-bits, respectively. When `TIMG_TOUUPDATE_REG` is cleared by

hardware, it indicates the latch operation has been completed and current timer value can be read from the [TIMG\\_TOLO\\_REG](#) and [TIMG\\_TOHI\\_REG](#) registers. [TIMG\\_TOLO\\_REG](#) and [TIMG\\_TOHI\\_REG](#) registers will remain unchanged for the CPU to read in its own time until [TIMG\\_TOUUPDATE\\_REG](#) is written to again.

### 11.3.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 11.3.4).

The 54-bit alarm value is configured using [TIMG\\_TOALARMLO\\_REG](#) and [TIMG\\_TOALARMHI\\_REG](#), which represent the lower 32-bits and higher 22-bits of the alarm value, respectively. However, the configured alarm value is ineffective until the alarm is enabled by setting the [TIMG\\_TO\\_ALARM\\_EN](#) field. To avoid alarm being enabled 'too late' (i.e. the timer value has already passed the alarm value when the alarm is enabled), the hardware will trigger the alarm immediately if the current timer value is:

- higher than the alarm value (within a defined range) when the up-down counter increments
- lower than the alarm value (within a defined range) when the up-down counter decrements

Table 11.3-1 and Table 11.3-2 show the relationship between the current value of the timer, the alarm value, and when an alarm is triggered. The current time value and the alarm value are defined as follows:

- $TIMG\_VALUE = \{TIMG\_TOHI\_REG, TIMG\_TOLO\_REG\}$
- $ALARM\_VALUE = \{TIMG\_TOALARMHI\_REG, TIMG\_TOALARMLO\_REG\}$

**Table 11.3-1. Alarm Generation When Up-Down Counter Increments**

Scenario	Range	Alarm
1	$ALARM\_VALUE - TIMG\_VALUE > 2^{53}$	Triggered
2	$0 < ALARM\_VALUE - TIMG\_VALUE \leq 2^{53}$	Triggered when the up-down counter counts $TIMG\_VALUE$ up to $ALARM\_VALUE$
3	$0 \leq TIMG\_VALUE - ALARM\_VALUE < 2^{53}$	Triggered
4	$TIMG\_VALUE - ALARM\_VALUE \geq 2^{53}$	Triggered when the up-down counter restarts counting up from 0 after reaching the timer's maximum value and counts $TIMG\_VALUE$ up to $ALARM\_VALUE$

**Table 11.3-2. Alarm Generation When Up-Down Counter Decrements**

Scenario	Range	Alarm
5	$TIMG\_VALUE - ALARM\_VALUE > 2^{53}$	Triggered
6	$0 < TIMG\_VALUE - ALARM\_VALUE \leq 2^{53}$	Triggered when the up-down counter counts $TIMG\_VALUE$ down to $ALARM\_VALUE$
7	$0 \leq ALARM\_VALUE - TIMG\_VALUE < 2^{53}$	Triggered
8	$ALARM\_VALUE - TIMG\_VALUE \geq 2^{53}$	Triggered when the up-down counter restarts counting down from the timer's maximum value after reaching the minimum value and counts $TIMG\_VALUE$ down to $ALARM\_VALUE$

When an alarm occurs, the [TIMG\\_TO\\_ALARM\\_EN](#) field is automatically cleared and no alarm will occur again until the [TIMG\\_TO\\_ALARM\\_EN](#) is set next time.

### 11.3.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the [TIMG\\_TO\\_LOAD\\_LO](#) and [TIMG\\_TO\\_LOAD\\_HI](#) fields that correspond to the lower 32-bits and higher 22-bits of the timer's new value, respectively. However, writing a reload value to [TIMG\\_TO\\_LOAD\\_LO](#) and [TIMG\\_TO\\_LOAD\\_HI](#) will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to [TIMG\\_TOLOAD\\_REG](#), which causes the timer's current value to be instantly reloaded. If [TIMG\\_TO\\_EN](#) is set, the timer will continue incrementing or decrementing from the new value. In this case if [TIMG\\_TO\\_ALARM\\_EN](#) is set, the timer will still trigger alarms in scenarios listed in Table 11.3-1 and 11.3-2. If [TIMG\\_TO\\_EN](#) is cleared, the timer will remain frozen at the new value until counting is re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs, thus allowing the timer to continue incrementing or decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the [TIMG\\_TO\\_AUTORELOAD](#) field should be set. If not enabled, the timer's value will continue to increment or decrement past the alarm value after an alarm.

### 11.3.5 RTC\_SLOW\_CLK Frequency Calculation

Using XTAL\_CLK as a reference, it is possible to calculate the frequency of clock sources for RTC\_SLOW\_CLK (i.e. RC\_SLOW\_CLK, RC\_FAST\_DIV\_CLK, and XTAL32K\_CLK) as follows:

1. Start periodic or one-shot frequency calculation (see Section 11.4.4 for details);
2. Once receiving the signal to start calculation, the counter of XTAL\_CLK and the counter of RTC\_SLOW\_CLK begin to work at the same time. When the counter of RTC\_SLOW\_CLK counts to CO, the two counters stop counting simultaneously;
3. Assume the value of XTAL\_CLK's counter is C1, and the frequency of RTC\_SLOW\_CLK would be calculated as:  $f_{rtc} = \frac{C0 \times f_{XTAL\_CLK}}{C1}$

### 11.3.6 Interrupts

Each timer has its own interrupt line that can be routed to the CPU, and thus each timer group has a total of two interrupt lines. Timers generate level interrupts that must be explicitly cleared by the CPU on each triggering.

Interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupts will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. To enable a timer's interrupt, the [TIMG\\_TO\\_INT\\_ENA](#) bit should be set.

The interrupts of each timer group are governed by a set of registers. Each timer within the group has a corresponding bit in each of these registers:

- [TIMG\\_TO\\_INT\\_RAW](#) : An alarm event sets it to 1. The bit will remain set until the timer's corresponding bit in [TIMG\\_TO\\_INT\\_CLR](#) is written.
- [TIMG\\_WDT\\_INT\\_RAW](#) : A stage time out will set the timer's bit to 1. The bit will remain set until the timer's corresponding bit in [TIMG\\_WDT\\_INT\\_CLR](#) is written.
- [TIMG\\_TO\\_INT\\_ST](#) : Reflects the status of each timer's interrupt and is generated by masking the bits of [TIMG\\_TO\\_INT\\_RAW](#) with [TIMG\\_TO\\_INT\\_ENA](#).
- [TIMG\\_WDT\\_INT\\_ST](#) : Reflects the status of each watchdog timer's interrupt and is generated by masking the bits of [TIMG\\_WDT\\_INT\\_RAW](#) with [TIMG\\_WDT\\_INT\\_ENA](#).
- [TIMG\\_TO\\_INT\\_ENA](#) : Used to enable or mask the interrupt status bits of timers within the group.
- [TIMG\\_WDT\\_INT\\_ENA](#) : Used to enable or mask the interrupt status bits of watchdog timer within the group.
- [TIMG\\_TO\\_INT\\_CLR](#) : Used to clear a timer's interrupt by setting its corresponding bit to 1. The timer's corresponding bit in [TIMG\\_TO\\_INT\\_RAW](#) and [TIMG\\_TO\\_INT\\_ST](#) will be cleared as a result. Note that a timer's interrupt must be cleared before the next interrupt occurs.
- [TIMG\\_WDT\\_INT\\_CLR](#) : Used to clear a timer's interrupt by setting its corresponding bit to 1. The watchdog timer's corresponding bit in [TIMG\\_WDT\\_INT\\_RAW](#) and [TIMG\\_WDT\\_INT\\_ST](#) will be cleared as a result. Note that a watchdog timer's interrupt must be cleared before the next interrupt occurs.

## 11.4 Configuration and Usage

### 11.4.1 Timer as a Simple Clock

1. Configure the time-base counter
  - Select clock source by setting or clearing [TIMG\\_TO\\_USE\\_XTAL](#) field. When CPU works in high performance mode, any value can be written to this field. When CPU works at low frequencies (i.e. when [SYSTEM\\_SOC\\_CLK\\_SEL](#) is not 1), this field must be set to 1.
  - Configure the 16-bit prescaler by setting [TIMG\\_TO\\_DIVIDER](#).
  - Configure the timer direction by setting or clearing [TIMG\\_TO\\_INCREASE](#).
  - Set the timer's starting value by writing the starting value to [TIMG\\_TO\\_LOAD\\_LO](#) and [TIMG\\_TO\\_LOAD\\_HI](#), then reloading it into the timer by writing any value to [TIMG\\_TOLOAD\\_REG](#).
2. Start the timer by setting [TIMG\\_TO\\_EN](#).
3. Get the timer's current value.
  - Write any value to [TIMG\\_TOUPDATE\\_REG](#) to latch the timer's current value.
  - Wait until [TIMG\\_TOUPDATE\\_REG](#) is cleared by hardware.
  - Read the latched timer value from [TIMG\\_TOLO\\_REG](#) and [TIMG\\_TOHI\\_REG](#).

### 11.4.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section [11.4.1](#).

2. Configure the alarm.
  - Configure the alarm value by setting [TIMG\\_TOALARMLO\\_REG](#) and [TIMG\\_TOALARMHI\\_REG](#).
  - Enable interrupt by setting [TIMG\\_TO\\_INT\\_ENA](#).
3. Disable auto reload by clearing [TIMG\\_TO\\_AUTORELOAD](#).
4. Start the alarm by setting [TIMG\\_TO\\_ALARM\\_EN](#).
5. Handle the alarm interrupt.
  - Clear the interrupt by setting the timer's corresponding bit in [TIMG\\_TO\\_INT\\_CLR](#).
  - Disable the timer by clearing [TIMG\\_TO\\_EN](#).

### 11.4.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 in Section [11.4.1](#).
2. Configure the alarm following step 2 in Section [11.4.2](#).
3. Enable auto reload by setting [TIMG\\_TO\\_AUTORELOAD](#) and configure the reload value via [TIMG\\_TO\\_LOAD\\_LO](#) and [TIMG\\_TO\\_LOAD\\_HI](#).
4. Start the alarm by setting [TIMG\\_TO\\_ALARM\\_EN](#).
5. Handle the alarm interrupt (repeat on each alarm iteration).
  - Clear the interrupt by setting the timer's corresponding bit in [TIMG\\_TO\\_INT\\_CLR](#).
  - If the next alarm requires a new alarm value and reload value (i.e. different alarm interval per iteration), then [TIMG\\_TOALARMLO\\_REG](#), [TIMG\\_TOALARMHI\\_REG](#), [TIMG\\_TO\\_LOAD\\_LO](#), and [TIMG\\_TO\\_LOAD\\_HI](#) should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
  - Re-enable the alarm by setting [TIMG\\_TO\\_ALARM\\_EN](#).
6. Stop the timer (on final alarm iteration).
  - Clear the interrupt by setting the timer's corresponding bit in [TIMG\\_TO\\_INT\\_CLR](#).
  - Disable the timer by clearing [TIMG\\_TO\\_EN](#).

### 11.4.4 RTC\_SLOW\_CLK Frequency Calculation

1. One-shot frequency calculation
  - Select the clock whose frequency is to be calculated (clock source of RTC\_SLOW\_CLK) via [TIMG\\_RTC\\_CALI\\_CLK\\_SEL](#), and configure the time of calculation via [TIMG\\_RTC\\_CALI\\_MAX](#).
  - Select one-shot frequency calculation by clearing [TIMG\\_RTC\\_CALI\\_START\\_CYCLING](#), and enable the two counters via [TIMG\\_RTC\\_CALI\\_START](#).
  - Once [TIMG\\_RTC\\_CALI\\_RDY](#) becomes 1, read [TIMG\\_RTC\\_CALI\\_VALUE](#) to get the value of XTAL\_CLK's counter, and calculate the frequency of RTC\_SLOW\_CLK according to the formula in Section [11.3.5](#).
2. Periodic frequency calculation



- Select the clock whose frequency is to be calculated (clock source of RTC\_SLOW\_CLK) via [TIMG\\_RTC\\_CALI\\_CLK\\_SEL](#), and configure the time of calculation via [TIMG\\_RTC\\_CALI\\_MAX](#).
- Select periodic frequency calculation by enabling [TIMG\\_RTC\\_CALI\\_START\\_CYCLING](#).
- When [TIMG\\_RTC\\_CALI\\_CYCLING\\_DATA\\_VLD](#) is 1, [TIMG\\_RTC\\_CALI\\_VALUE](#) is valid.

### 3. Timeout

If the counter of RTC\_SLOW\_CLK cannot finish counting in [TIMG\\_RTC\\_CALI\\_TIMEOUT\\_RST\\_CNT](#) cycles, [TIMG\\_RTC\\_CALI\\_TIMEOUT](#) will be set to indicate a timeout.

## 11.5 Register Summary

The addresses in this section are relative to **Timer Group** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>TO control and configuration registers</b>			
<a href="#">TIMG_TOCONFIG_REG</a>	Timer 0 configuration register	0x0000	varies
<a href="#">TIMG_TOLO_REG</a>	Timer 0 current value, low 32 bits	0x0004	RO
<a href="#">TIMG_TOHI_REG</a>	Timer 0 current value, high 22 bits	0x0008	RO
<a href="#">TIMG_TOUUPDATE_REG</a>	Write to copy current timer value to TIMG_TOLO_REG or TIMG_TOHI_REG	0x000C	R/W/SC
<a href="#">TIMG_TOALARMLO_REG</a>	Timer 0 alarm value, low 32 bits	0x0010	R/W
<a href="#">TIMG_TOALARMHI_REG</a>	Timer 0 alarm value, high bits	0x0014	R/W
<a href="#">TIMG_TOLOADLO_REG</a>	Timer 0 reload value, low 32 bits	0x0018	R/W
<a href="#">TIMG_TOLOADHI_REG</a>	Timer 0 reload value, high 22 bits	0x001C	R/W
<a href="#">TIMG_TOLOAD_REG</a>	Write to reload timer from TIMG_TOLOADLO_REG or TIMG_TOLOADHI_REG	0x0020	WT
<b>WDT control and configuration registers</b>			
<a href="#">TIMG_WDTCONFIG0_REG</a>	Watchdog timer configuration register	0x0048	varies
<a href="#">TIMG_WDTCONFIG1_REG</a>	Watchdog timer prescaler register	0x004C	varies
<a href="#">TIMG_WDTCONFIG2_REG</a>	Watchdog timer stage 0 timeout value	0x0050	R/W
<a href="#">TIMG_WDTCONFIG3_REG</a>	Watchdog timer stage 1 timeout value	0x0054	R/W
<a href="#">TIMG_WDTCONFIG4_REG</a>	Watchdog timer stage 2 timeout value	0x0058	R/W
<a href="#">TIMG_WDTCONFIG5_REG</a>	Watchdog timer stage 3 timeout value	0x005C	R/W
<a href="#">TIMG_WDTFEED_REG</a>	Write to feed the watchdog timer	0x0060	WT
<a href="#">TIMG_WDTWPROTECT_REG</a>	Watchdog write protect register	0x0064	R/W
<b>RTC frequency calculation control and configuration registers</b>			
<a href="#">TIMG_RTCCALICFG_REG</a>	RTC frequency calculation configuration register 0	0x0068	varies
<a href="#">TIMG_RTCCALICFG1_REG</a>	RTC frequency calculation configuration register 1	0x006C	RO
<a href="#">TIMG_RTCCALICFG2_REG</a>	RTC frequency calculation configuration register 2	0x0080	varies
<b>Interrupt registers</b>			
<a href="#">TIMG_INT_ENA_TIMERS_REG</a>	Interrupt enable bits	0x0070	R/W
<a href="#">TIMG_INT_RAW_TIMERS_REG</a>	Raw interrupt status	0x0074	R/SS/WTC
<a href="#">TIMG_INT_ST_TIMERS_REG</a>	Masked interrupt status	0x0078	RO
<a href="#">TIMG_INT_CLR_TIMERS_REG</a>	Interrupt clear bits	0x007C	WT
<b>Version register</b>			
<a href="#">TIMG_NTIMERS_DATE_REG</a>	Timer version control register	0x00F8	R/W
<b>Clock configuration registers</b>			

Name	Description	Address	Access
<a href="#">TIMG_REGCLK_REG</a>	Timer group clock gate register	0x00FC	R/W

## 11.6 Registers

The addresses in this section are relative to **Timer Group** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 11.1. TIMG\_TOCONFIG\_REG (0x0000)**

TIMG_TO_EN TIMG_TO_INCREASE TIMG_TO_AUTORELOAD				TIMG_TO_DIVIDER										TIMG_TO_DIVIDER_RST (reserved) TIMG_TO_ALARM_EN TIMG_TO_USE_XTAL (reserved)																	
31	30	29	28											13	12	11	10	9	8											0	
0	1	1	0x01											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

Reset

**TIMG\_TO\_USE\_XTAL** 1: Use XTAL\_CLK as the source clock of timer group. 0: Use PLL\_F40M\_CLK as the source clock of timer group. (R/W)

**TIMG\_TO\_ALARM\_EN** When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W/SC)

**TIMG\_TO\_DIVIDER\_RST** When set, timer 0's clock divider counter will be reset. (WT)

**TIMG\_TO\_DIVIDER** Timer 0 clock (TO\_clk) prescaler value. (R/W)

**TIMG\_TO\_AUTORELOAD** When set, timer 0 auto-reload at alarm is enabled. (R/W)

**TIMG\_TO\_INCREASE** When set, the timer 0 time-base counter will increment every clock tick. When cleared, the timer 0 time-base counter will decrement. (R/W)

**TIMG\_TO\_EN** When set, the timer 0 time-base counter is enabled. (R/W)

**Register 11.2. TIMG\_TOLO\_REG (0x0004)**

TIMG_TO_LO																															
31																															0
0x000000																															
Reset																															

Reset

**TIMG\_TO\_LO** After writing to TIMG\_TOUUPDATE\_REG, the low 32 bits of the time-base counter of timer 0 can be read here. (RO)

**Register 11.3. TIMG\_TOHI\_REG (0x0008)**

(reserved)										TIMG_TO_HI																															
31										22										21																					0
0 0 0 0 0 0 0 0 0 0										0x0000																					Reset										

**TIMG\_TO\_HI** After writing to TIMG\_TOUUPDATE\_REG, the high 22 bits of the time-base counter of timer 0 can be read here. (RO)

**Register 11.4. TIMG\_TOUUPDATE\_REG (0x000C)**

TIMG_TO_UPDATE																															(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	0	Reset																											

**TIMG\_TO\_UPDATE** After writing 0 or 1 to TIMG\_TOUUPDATE\_REG, the counter value is latched. (R/W/SC)

**Register 11.5. TIMG\_TOALARMLO\_REG (0x0010)**

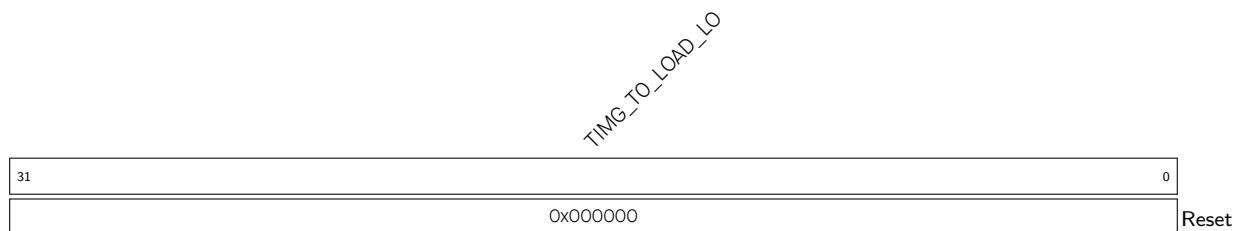
TIMG_TO_ALARM_LO																															
31																															0
0x000000																															
Reset																															

**TIMG\_TO\_ALARM\_LO** Timer 0 alarm trigger time-base counter value, low 32 bits. (R/W)

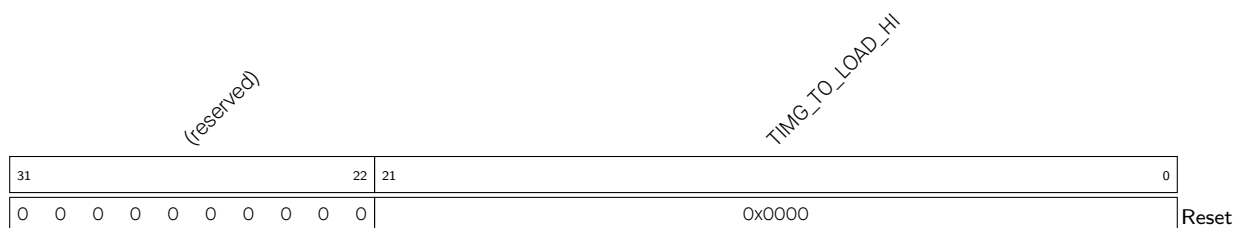
**Register 11.6. TIMG\_TOALARMHI\_REG (0x0014)**

(reserved)										TIMG_TO_ALARM_HI																						
31										22	21																					0
0	0	0	0	0	0	0	0	0	0	0x0000																					Reset	

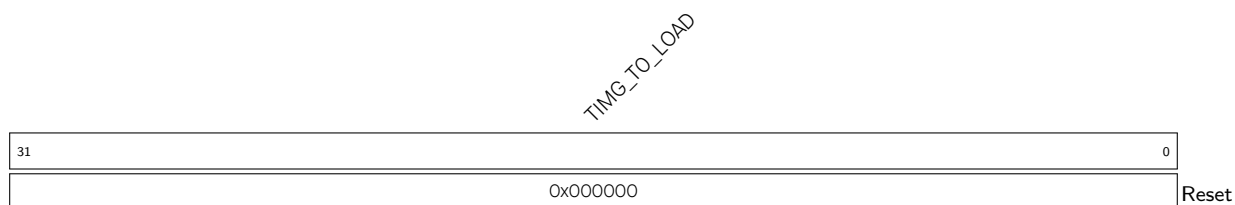
**TIMG\_TO\_ALARM\_HI** Timer 0 alarm trigger time-base counter value, high 22 bits. (R/W)

**Register 11.7. TIMG\_TOLOADLO\_REG (0x0018)**

**TIMG\_TO\_LOAD\_LO** Low 32 bits of the value that a reload will load onto timer 0 time-base counter.  
(R/W)

**Register 11.8. TIMG\_TOLOADHI\_REG (0x001C)**

**TIMG\_TO\_LOAD\_HI** High 22 bits of the value that a reload will load onto timer 0 time-base counter.  
(R/W)

**Register 11.9. TIMG\_TOLOAD\_REG (0x0020)**

**TIMG\_TO\_LOAD** Write any value to trigger a timer 0 time-base counter reload. (WT)

### Register 11.10. TIMG\_WDTCONFIG0\_REG (0x0048)

[illegible]

**TIMG\_WDT\_PROCPU\_RESET\_EN** WDT reset CPU enable. (R/W)

**TIMG\_WDT\_FLASHBOOT\_MOD\_EN** When set, Flash boot protection is enabled. (R/W)

**TIMG\_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\_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\_WDT\_USE\_XTAL** Chooses WDT clock. 0: PLL\_F40M\_CLK, 1: XTAL\_CLK. (R/W)

**TIMG\_WDT\_CONF\_UPDATE\_EN** Updates the WDT configuration registers. (WT)

**TIMG\_WDT\_STG3** Stage 3 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG\_WDT\_STG2** Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG\_WDT\_STG1** Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG\_WDT\_STGO** Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

**TIMG\_WDT\_EN** When set, MWDT is enabled. (R/W)

### Register 11.11. TIMG\_WDTCONFIG1\_REG (0x004C)

[illegible]

**TIMG\_WDT\_DIVCNT\_RST** When set, WDT 's clock divider counter will be reset. (WT)

**TIMG\_WDT\_CLK\_PRESCALE** MWDT clock prescaler value. MWDT clock period = MWDT's clock source period \* TIMG WDT CLK PRESCALE. (R/W)

**Register 11.12. TIMG\_WDTCONFIG2\_REG (0x0050)**

TIMG_WDT_STG0_HOLD	
31	0
26000000	
Reset	

**TIMG\_WDT\_STG0\_HOLD** Stage 0 timeout value, in MWDAT clock cycles. (R/W)

**Register 11.13. TIMG\_WDTCONFIG3\_REG (0x0054)**

TIMG_WDT_STG1_HOLD	
31	0
0x7ffffff	
Reset	

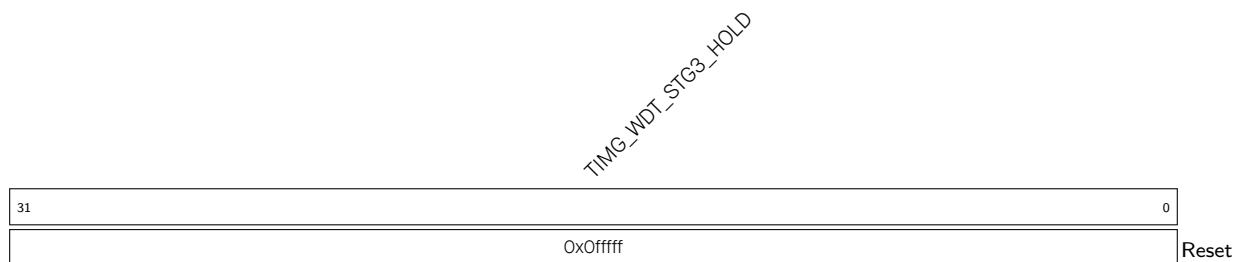
**TIMG\_WDT\_STG1\_HOLD** Stage 1 timeout value, in MWDAT clock cycles. (R/W)

**Register 11.14. TIMG\_WDTCONFIG4\_REG (0x0058)**

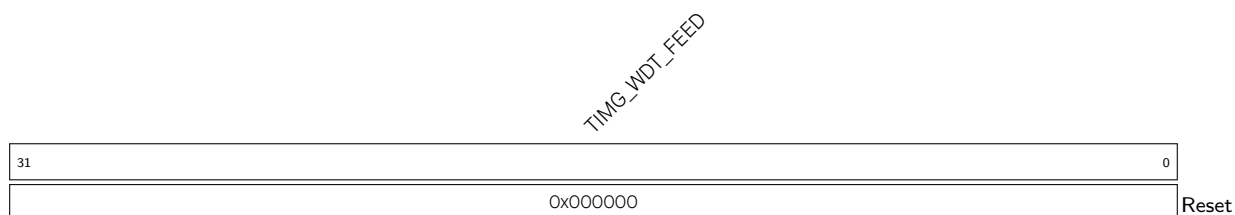
TIMG_WDT_STG2_HOLD	
31	0
0x0ffffff	
Reset	

**TIMG\_WDT\_STG2\_HOLD** Stage 2 timeout value, in MWDAT clock cycles. (R/W)

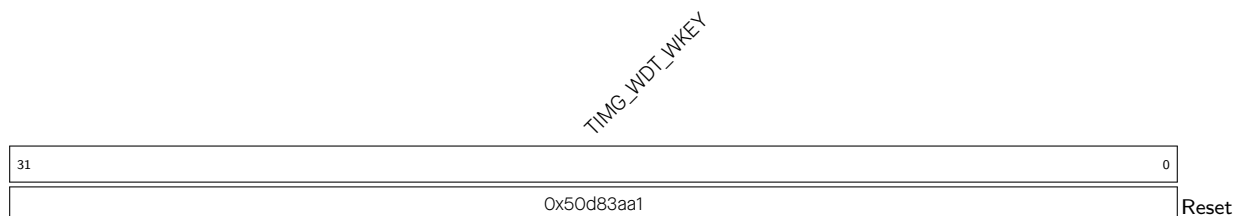


**Register 11.15. TIMG\_WDTCONFIG5\_REG (0x005C)**

**TIMG\_WDT\_STG3\_HOLD** Stage 3 timeout value, in MWDAT clock cycles. (R/W)

**Register 11.16. TIMG\_WDTFEED\_REG (0x0060)**

**TIMG\_WDT\_FEED** Write any value to feed the MWDAT. (WO) (WT)

**Register 11.17. TIMG\_WDTWPROTECT\_REG (0x0064)**

**TIMG\_WDT\_WKEY** If the register contains a different value than its reset value, write protection is enabled. (R/W)

### Register 11.18. TIMG\_RTCCALICFG\_REG (0x0068)

[illegible]

**TIME\_RTC\_CALI\_START\_CYCLING** 0: one-shot frequency calculation mode, 1: periodic frequency calculation mode. (R/W)

**TIMG\_RTC\_CALI\_CLK\_SEL** 0: RC\_SLOW\_CLK, 1: RC\_FAST\_DIV\_CLK, 2: XTAL32K\_CLK. (R/W)

**TIMG\_RTC\_CALI\_RDY** Marks the completion of one-shot frequency calculation. (RO)

**TIMG\_RTC\_CALI\_MAX** Configures the time to calculate the frequency of RTC slow clock RTC\_SLOW\_CLK. Measurement unit: RTC\_SLOW\_CLK cycle. (R/W)

**TIMG\_RTC\_CALI\_START** Set this bit to start one-shot frequency calculation. (R/W)

### Register 11.19. TIMG\_RTCCALICFG1\_REG (0x006C)

TIMG_RTC_CAL_VALUE										(reserved)		TIMG_RTC_CAL_CYCLING_DATA_VLD		
31							7	6	1			0		
0x000000							0	0	0	0	0	0	0	Reset

**TIMG\_RTC\_CALI\_CYCLING\_DATA\_VLD** Marks the completion of periodic frequency calculation.  
(RO)

**TIMG\_RTC\_CALI\_VALUE** When one-shot or periodic frequency calculation completes, read this value to calculate the frequency of RTC slow clock RTC\_SLOW\_CLK. Measurement unit: XTAL\_CLK cycle. (RO)

Register 11.20. TIMG\_RTCCALICFG2\_REG (0x0080)

																TIMG_RTC_CALL_TIMEOUT_THRES			
																TIMG_RTC_CALL_TIMEOUT_RST_CNT			
																(reserved)			
																TIMG_RTC_CALL_TIMEOUT			
31											7	6	3	2	1	0			
0x1ffffff												3		0	0	0	Reset		

**TIMG\_RTC\_CALI\_TIMEOUT** Indicates frequency calculation timeout. (RO)

**TIMG\_RTC\_CALI\_TIMEOUT\_RST\_CNT** Cycles to reset frequency calculation timeout. (R/W)

**TIMG\_RTC\_CALI\_TIMEOUT\_THRES** Threshold value for the frequency calculation timer. If the timer's value exceeds this threshold, a timeout is triggered. (R/W)

Register 11.21. TIMG\_INT\_ENA\_TIMERS\_REG (0x0070)

																(reserved)			
																TIMG_WDT_INT_ENA			
																TIMG_TO_INT_ENA			
31											2	1	0						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**TIMG\_TO\_INT\_ENA** The interrupt enable bit for the TIMG\_TO\_INT interrupt. (R/W)

**TIMG\_WDT\_INT\_ENA** The interrupt enable bit for the TIMG\_WDT\_INT interrupt. (R/W)

Register 11.22. TIMG\_INT\_RAW\_TIMERS\_REG (0x0074)

																(reserved)			
																TIMG_WDT_INT_RAW			
																TIMG_TO_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

**TIMG\_TO\_INT\_RAW** The raw interrupt status bit for the TIMG\_TO\_INT interrupt. (R/SS/WTC)

**TIMG\_WDT\_INT\_RAW** The raw interrupt status bit for the TIMG\_WDT\_INT interrupt. (R/SS/WTC)

Register 11.23. TIMG\_INT\_ST\_TIMERS\_REG (0x0078)

(reserved)																												TIMG_WDT_INT_ST TIMG_TO_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**TIMG\_TO\_INT\_ST** The masked interrupt status bit for the TIMG\_TO\_INT interrupt. (RO)

**TIMG\_WDT\_INT\_ST** The masked interrupt status bit for the TIMG\_WDT\_INT interrupt. (RO)

Register 11.24. TIMG\_INT\_CLR\_TIMERS\_REG (0x007C)

(reserved)																															TIMG_WDT_INT_CLR TIMG_TO_INT_CLR				
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	0	Reset

**TIMG\_TO\_INT\_CLR** Set this bit to clear the TIMG\_TO\_INT interrupt. (WT)

**TIMG\_WDT\_INT\_CLR** Set this bit to clear the TIMG\_WDT\_INT interrupt. (WT)

Register 11.25. TIMG\_NTIMERS\_DATE\_REG (0x00F8)

(reserved)				TIMG_NTIMGS_DATE																							31	28	27	0	
0	0	0	0	0x2006191																											Reset

**TIMG\_NTIMGS\_DATE** This is the version control register. (R/W)

Register 11.26. TIMG\_REGCLK\_REG (0x00FC)

TIMG_CLK_EN																															(reserved)																														
TIMG_TIMER_CLK_IS_ACTIVE																																																													
TIMG_WDT_CLK_IS_ACTIVE																																																													
31	30	29	28																																																								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	0	0	0	0	0	0	0	Reset																										

- TIMG\_WDT\_CLK\_IS\_ACTIVE Enable WDT's clock. (R/W)
- TIMG\_TIMER\_CLK\_IS\_ACTIVE Enable timer 0's clock. (R/W)
- TIMG\_CLK\_EN Register clock gate signal. 1: Registers can be read and written to by software. 0: Registers can not be read or written to by software. (R/W)

## Chapter 12

# Watchdog Timers (WDT)

## 12.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g. is stuck in a software loop or in overdue events) will fail to feed the watchdog thus trigger a watchdog timeout. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

As shown in Figure 12.1-1, ESP8684 contains two digital watchdog timers: one in the timer group in Chapter 11 [Timer Group \(TIMG\)](#) (called Main System Watchdog Timer, or MWDT) and one in the RTC Module (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages and each stage can be programmed to take one action upon timeout, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, CPU reset, and core reset, while RWDT supports four timeout actions: interrupt, CPU reset, core reset, and system reset (see details in Section 12.2.2.2 [Stages and Timeout Actions](#)). A timeout value can be set for each stage individually.

During the flash boot process, RWDT and the MWDT in timergroup 0 are enabled automatically in order to detect and recover from booting errors.

ESP8684 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

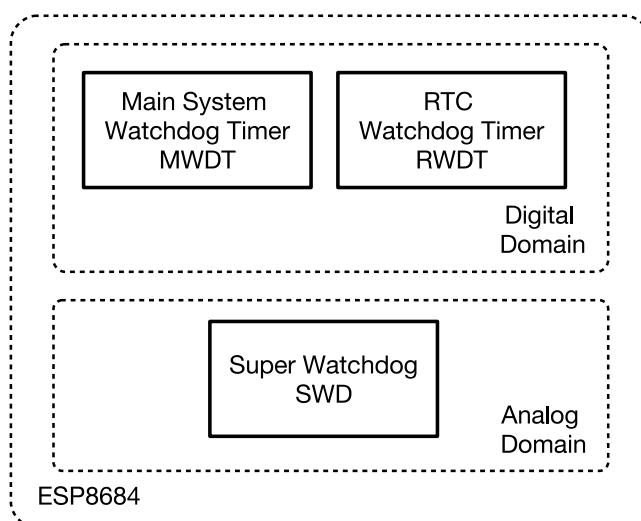


Figure 12.1-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer's, their register descriptions are provided in Chapter 11 [Timer Group \(TIMG\)](#) and Chapter 9 [Low-power Management](#)

(*RTC\_CNTL*).

## 12.2 Digital Watchdog Timers

### 12.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWD0: interrupt, CPU reset, core reset
  - RWDT: interrupt, CPU reset, core reset, system reset
- Flash boot protection at stage 0:
  - MWD0: core reset upon timeout
  - RWDT: system reset upon timeout
- Write protection that makes WDT register read only unless unlocked
- 32-bit timeout counter
- Clock source:
  - MWD0: 40 MHz PLL\_F40M\_CLK or XTAL\_CLK
  - RWDT: RTC\_SLOW\_CLK

### 12.2.2 Functional Description

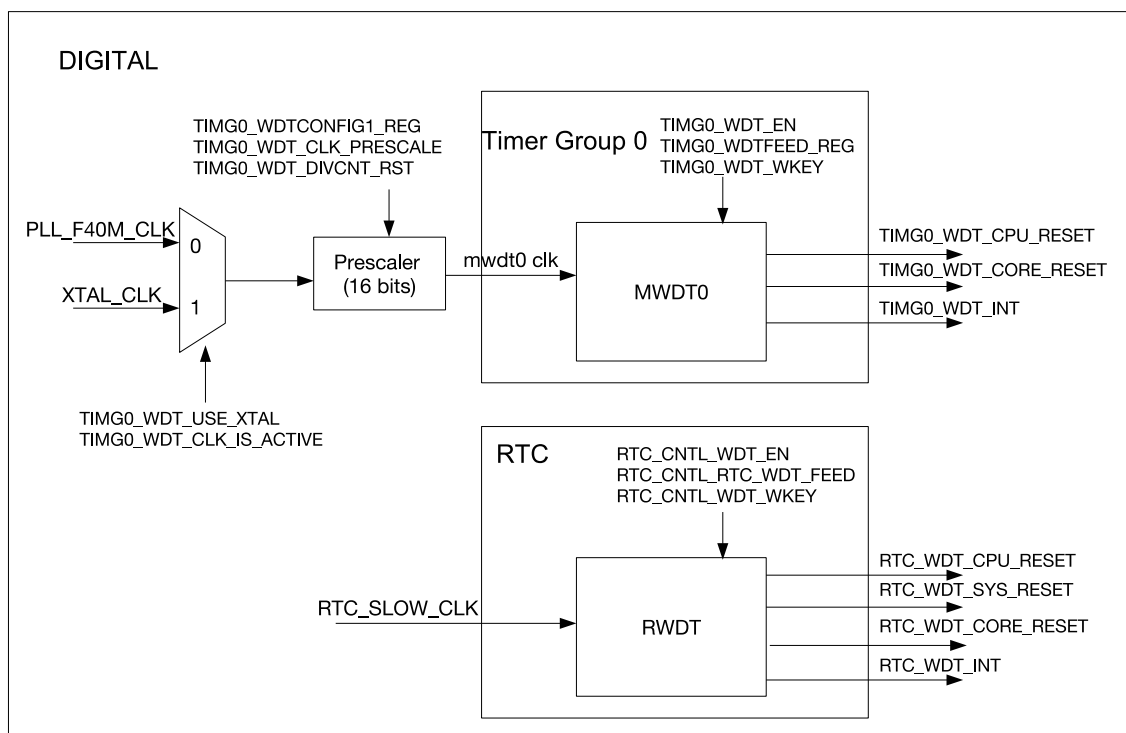


Figure 12.2-1. Digital Watchdog Timers in ESP8684



Figure 12.2-1 shows the two watchdog timers in ESP8684 digital systems.

### 12.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

MWDT can select between the PLL\_F40M\_CLK clock or external clock (XTAL\_CLK) as its clock source by setting the `TIMG_WDT_USE_XTAL` field of the `TIMG_WDTCONFIG0_REG` register. Note that when the chip is in low-power mode and the clock source of CPU\_CLK is not PLL\_CLK (i.e. when `SYSTEM_SOC_CLK_SEL` is not 1, see details in Table 6.2-1 of Chapter 6 *Reset and Clock*), MWDT can only select XTAL\_CLK. The selected clock is switched on by setting `TIMG_WDT_CLK_IS_ACTIVE` field of the `TIMG_REGCLK_REG` register to 1 and switched off by setting it to 0. Then the selected clock is divided by a 16-bit configurable prescaler. The 16-bit prescaler for MWDT is configured via the `TIMG_WDT_CLK_PRESCALE` field of `TIMG_WDTCONFIG1_REG`. When `TIMG_WDT_DIVCNT_RST` field is set, the prescaler is reset and it can be re-configured at once.

In contrast, the clock source of RWDT is derived directly from RTC\_SLOW\_CLK (see details in Chapter 6 *Reset and Clock*).

MWDT and RWDT are enabled by setting the `TIMG_WDT_EN` and `RTC_CNTL_WDT_EN` fields respectively. When enabled, the 32-bit counters of the watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e. timeout of the current stage). When this occurs, the current counter value is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to `TIMG_WDTFEED_REG` for MDWT and `RTC_CNTL_RTC_WDT_FEED` for RWDT.

### 12.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout action. When one stage times out, the timeout action is triggered, the counter value is reset to zero, and the next stage becomes active. MWDT/ RWDT provide four stages (called stages 0 to 3). The watchdog timers will progress through each stage in a loop (i.e. from stage 0 to 3, then back to stage 0).

Timeout values of each stage for MWDT are configured in `TIMG_WDTCONFIGi_REG` (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured using `RTC_CNTL_WDT_STGj_HOLD` field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT ( $T_{hold0}$ ) is determined by the combination of the

`EFUSE_WDT_DELAY_SEL` field of eFuse register `EFUSE_RD_REPEAT_DATA0_REG` and `RTC_CNTL_WDT_STG0_HOLD`. The relationship is as follows:

$$T_{hold0} = RTC\_CNTL\_WDT\_STG0\_HOLD \ll (EFUSE\_WDT\_DELAY\_SEL + 1)$$

where  $\ll$  is a left-shift operator.

Upon the timeout of each stage, one of the following timeout actions will be executed:

**Table 12.2-1. Timeout Actions**

Timeout Action	Description
Interrupt	Trigger an interrupt
CPU reset	Reset the CPU core
Core reset	Reset the main system (which includes MWDT, CPU, and all peripherals). The power management unit and RTC peripherals will not be reset
System reset	Reset the main system, power management unit and RTC peripherals (see details in Chapter 9 <i>Low-power Management (RTC_CNTL)</i> ). This action is only available in RWDT
Disabled	No effect on the system

For MWDT, the timeout action of all stages is configured in [TIMG\\_WDTCONFIG0\\_REG](#). Likewise for RWDT, the timeout action is configured in [RTC\\_CNTL\\_WDTCONFIG0\\_REG](#).

### 12.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g. due to a misplaced register write). Therefore, MWDT and RWDT incorporate a write protection mechanism that prevent the watchdogs from being disabled or tampered with due to an accidental write. The write protection mechanism is implemented using a write-key field for each timer ([TIMG\\_WDT\\_WKEY](#) for MWDT, [RTC\\_CNTL\\_WDT\\_WKEY](#) for RWDT). The value 0x50D83AA1 must be written to the watchdog timer's write-key field before any other register of the same watchdog timer can be changed. Any attempts to write to a watchdog timer's registers (other than the write-key field itself) whilst the write-key field's value is not 0x50D83AA1 will be ignored. The recommended procedure for accessing a watchdog timer is as follows:

1. Disable the write protection by writing the value 0x50D83AA1 to the timer's write-key field.
2. Make the required modification of the watchdog such as feeding or changing its configuration.
3. Re-enable write protection by writing any value other than 0x50D83AA1 to the timer's write-key field.

### 12.2.2.4 Flash Boot Protection

During flash booting process, MWDT as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon timeout, known as core reset. Likewise, stage 0 for RWDT is configured to system reset, which resets the main system and RTC when it times out. After booting, [TIMG\\_WDT\\_FLASHBOOT\\_MOD\\_EN](#) and [RTC\\_CNTL\\_WDT\\_FLASHBOOT\\_MOD\\_EN](#) should be cleared to stop the flash boot protection procedure for both MWDT and RWDT respectively. After this, MWDT and RWDT can be configured by software.

## 12.3 Super Watchdog

Super watchdog (SWD) is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required. SWD contains a watchdog circuit that needs to be fed for at least once during its timeout period, which is slightly less than one second. About 100 ms

before watchdog timeout, it will also send out a WD\_INTR signal as a request to remind the system to feed the watchdog.

If the system doesn't respond to SWD feed request and watchdog finally times out, SWD will generate a system level signal SWD\_RSTB to reset whole digital circuits on the chip.

The source of the clock for SWD is constant and can not be selected.

### 12.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD is about to time out
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

### 12.3.2 Super Watchdog Controller

#### 12.3.2.1 Structure

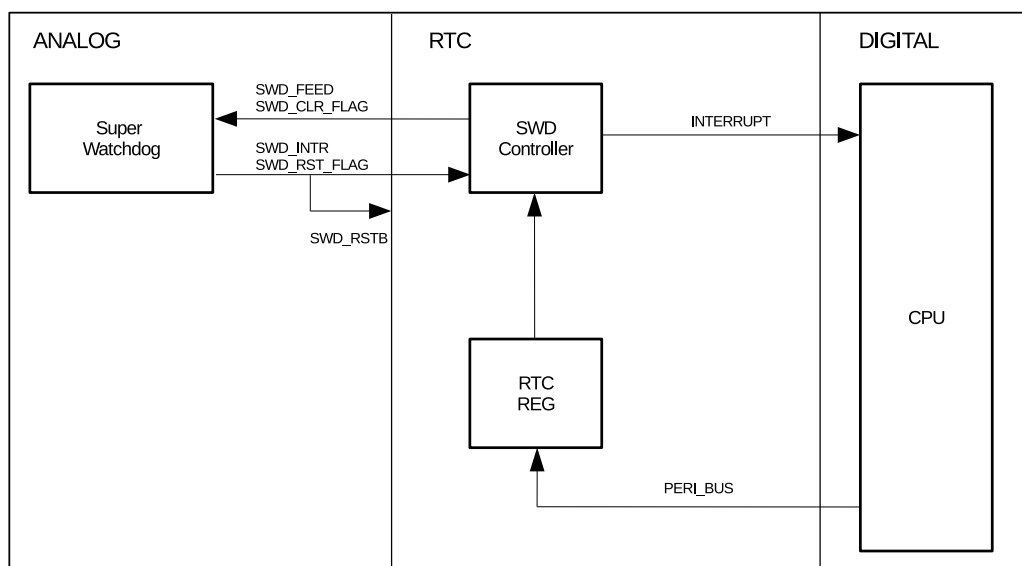


Figure 12.3-1. Super Watchdog Controller Structure

#### 12.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU.
- Main CPU can feed SWD directly by setting `RTC_CNTL_SWD_FEED`.

- When trying to feed SWD, CPU needs to disable SWD controller's write protection by writing 0x8F1D312A to [RTC\\_CNTL\\_SWD\\_WKEY](#). This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting [RTC\\_CNTL\\_SWD\\_AUTO\\_FEED\\_EN](#) to 1, SWD controller can also feed SWD itself without any interaction with CPU.

After reset:

- Check [RTC\\_CNTL\\_RESET\\_CAUSE\\_PROCPU\[5:0\]](#) for the cause of CPU reset.  
If [RTC\\_CNTL\\_RESET\\_CAUSE\\_PROCPU\[5:0\]](#) == 0x12, it indicates that the cause is SWD reset.
- Set [RTC\\_CNTL\\_SWD\\_RST\\_FLAG\\_CLR](#) to clear the SWD reset flag.

## 12.4 Interrupts

For watchdog timer interrupts, please refer to Section [11.3.6 Interrupts](#) in Chapter [11 Timer Group \(TIMG\)](#).

## 12.5 Registers

MWDT registers are part of the timer submodule and are described in Section [11.5 Register Summary](#) in Chapter [11 Timer Group \(TIMG\)](#). RWD and SWD registers are part of the RTC submodule and are described in Section [9.6 Register Summary](#) in Chapter [9 Low-power Management \(RTC\\_CNTL\)](#).

## Chapter 13

# System Registers (SYSTEM)

## 13.1 Overview

The ESP8684 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. Specifically, ESP8684 has various system configuration registers that can be used for the chip's clock management (clock gating), power management, and the configuration of peripherals and core-system modules. This chapter lists all these system registers and their functions.

## 13.2 Features

ESP8684 system registers can be used to control the following peripheral blocks and core modules:

- System and memory
- Clock
- Software interrupts
- Peripheral clock gating and reset

## 13.3 Function Description

### 13.3.1 System and Memory Registers

#### 13.3.1.1 Internal Memory

The following registers can be used to control ESP8684's internal memory:

- In register [SYSCON\\_CLKGATE\\_FORCE\\_ON\\_REG](#):
  - Setting different bits of the [SYSCON\\_ROM\\_CLKGATE\\_FORCE\\_ON](#) field forces on the clock gates of different blocks of Internal ROM 0 and Internal ROM 1.
  - Setting different bits of the [SYSCON\\_SRAM\\_CLKGATE\\_FORCE\\_ON](#) field forces on the clock gates of different blocks of Internal SRAM.
  - This means when the respective bits of this register are set to 1, the clock gate of the corresponding ROM or SRAM blocks will always be on. Otherwise, the clock gate will turn on automatically when the corresponding ROM or SRAM blocks are accessed and turn off automatically when the corresponding ROM or SRAM blocks are not accessed. Therefore, it's recommended to configure these bits to 0 to lower power consumption.

- In register [SYSCON\\_MEM\\_POWER\\_DOWN\\_REG](#):
  - Setting different bits of the [SYSCON\\_ROM\\_POWER\\_DOWN](#) field sends different blocks of Internal ROM 0 and Internal ROM 1 into retention state.
  - Setting different bits of the [SYSCON\\_SRAM\\_POWER\\_DOWN](#) field sends different blocks of Internal SRAM into retention state.
  - The “Retention” state is a low-power state of a memory block. In this state, the memory block still holds all the data stored but cannot be accessed, thus reducing the power consumption. Therefore, you can send a certain block of memory into the retention state to reduce power consumption if you know you are not going to use such memory block for some time.
- In register [SYSCON\\_MEM\\_POWER\\_UP\\_REG](#):
  - By default, all memory enters low-power state when the chip enters the Light-sleep mode.
  - Setting different bits of the [SYSCON\\_ROM\\_POWER\\_UP](#) field forces different blocks of Internal ROM 0 and Internal ROM 1 to work as normal (do not enter the retention state) when the chip enters Light-sleep.
  - Setting different bits of the [SYSCON\\_SRAM\\_POWER\\_UP](#) field forces different blocks of Internal SRAM to work as normal (do not enter the retention state) when the chip enters Light-sleep.

For detailed information about controlling different blocks using different controlling bits in the above-mentioned registers, please see Table 13.3-1 below.

**Table 13.3-1. Memory Controlling Bit**

Memory	Instruction Lowest Address	Instruction Highest Address	Data Lowest Address	Data Highest Address	Controlling Bit
ROM 0	0x4000_0000	0x4003_FFFF	-	-	Bit0
ROM 1	0x4004_0000	0x4007_FFFF	0x3FF0_0000	0x3FF3_FFFF	Bit1
	0x4008_0000	0x4008_FFFF	0x3FF4_0000	0x3FF4_FFFF	Bit2
SRAM Block 0	0x4037_C000	0x4037_FFFF	-	-	Bit0
SRAM Block 1	0x4038_0000	0x4038_FFFF	0x3FCA_0000	0x3FCA_FFFF	Bit1
SRAM Block 2	0x4039_0000	0x4039_FFFF	0x3FCB_0000	0x3FCB_FFFF	Bit2
SRAM Block 3	0x403A_0000	0x403B_FFFF	0x3FCC_0000	0x3FCD_FFFF	Bit3

For more information, please refer to Chapter 3 *System and Memory*.

### 13.3.1.2 External Memory

[SYSTEM\\_EXTERNAL\\_DEVICE\\_ENCRYPT\\_DECRYPT\\_CONTROL\\_REG](#) configures encryption and decryption options of the external memory. For details, please refer to Chapter 17 *External Memory Encryption and Decryption (XTS\_AES)*.

## 13.3.2 Clock Registers

The following registers are used to set clock sources and frequency. For more information, please refer to Chapter 6 *Reset and Clock*.

- [SYSTEM\\_CPU\\_PER\\_CONF\\_REG](#)

- [SYSTEM\\_SYSCLK\\_CONF\\_REG](#)

### 13.3.3 Interrupt Signal Registers

The following registers are used for generating the interrupt signals (software interrupt), which then can be routed to the CPU peripheral interrupts via the interrupt matrix. To be more specific, writing 1 to any of the following registers generates an interrupt signal. Writing 0 clears the interrupt signal. Therefore, these registers can be used by software to control interrupts. The following registers correspond to the interrupt source SW\_INTR\_0/1/2/3. For more information, please refer to Chapter 8 *Interrupt Matrix (INTMTRX)*.

- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_0\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_1\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_2\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_3\\_REG](#)

### 13.3.4 Peripheral Clock Gating and Reset Registers

The following registers are used for controlling the clock gating and reset of different peripherals. Details can be seen in Table 13.3-2.

- [SYSTEM\\_CACHE\\_CONTROL\\_REG](#)
- [SYSTEM\\_PERIP\\_CLK\\_ENO\\_REG](#)
- [SYSTEM\\_PERIP\\_RST\\_ENO\\_REG](#)
- [SYSTEM\\_PERIP\\_CLK\\_EN1\\_REG](#)
- [SYSTEM\\_PERIP\\_RST\\_EN1\\_REG](#)

ESP8684 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 and release the peripheral from reset state. For details, see the table below:

Table 13.3-2. Clock Gating and Reset Bits

Component	Clock Enabling Bit <sup>1</sup>	Reset Controlling Bit <sup>2, 3</sup>
<b>CACHE Control</b>	<a href="#">SYSTEM_CACHE_CONTROL_REG</a>	
DCACHE	<a href="#">SYSTEM_DCACHE_CLK_ON</a>	<a href="#">SYSTEM_DCACHE_RESET</a>
ICACHE	<a href="#">SYSTEM_ICACHE_CLK_ON</a>	<a href="#">SYSTEM_ICACHE_RESET</a>
<b>GDMA</b>	<a href="#">SYSTEM_GDMA_CTRL_REG</a>	
GDMA	<a href="#">SYSTEM_GDMA_CLK_ON</a>	<a href="#">SYSTEM_GDMA_RESET</a>
<b>CPU</b>	<a href="#">SYSTEM_CPU_PERI_CLK_EN_REG</a>	<a href="#">SYSTEM_CPU_PERI_RST_EN_REG</a>
DEBUG_ASSIST	<a href="#">SYSTEM_CLK_EN_ASSIST_DEBUG</a>	<a href="#">SYSTEM_RST_EN_ASSIST_DEBUG</a>
<b>Peripherals</b>	<a href="#">SYSTEM_PERIP_CLK_ENO_REG</a>	<a href="#">SYSTEM_PERIP_RST_ENO_REG</a>
SPIO / SPI1	<a href="#">SYSTEM_SPIO1_CLK_EN</a>	<a href="#">SYSTEM_SPIO1_RST</a>
UART0	<a href="#">SYSTEM_UART_CLK_EN</a>	<a href="#">SYSTEM_UART_RST</a>
UART1	<a href="#">SYSTEM_UART1_CLK_EN</a>	<a href="#">SYSTEM_UART1_RST</a>
SPI2	<a href="#">SYSTEM_SPI2_CLK_EN</a>	<a href="#">SYSTEM_SPI2_RST</a>

Cont'd on next page

Table 13.3-2 – cont'd from previous page

Component	Clock Enabling Bit <sup>1</sup>	Reset Controlling Bit <sup>2, 3</sup>
I2CO	<a href="#">SYSTEM_I2C_EXT0_CLK_EN</a>	<a href="#">SYSTEM_I2C_EXT0_RST</a>
LED PWM Controller	<a href="#">SYSTEM_LEDC_CLK_EN</a>	<a href="#">SYSTEM_LEDC_RST</a>
Timer Group0	<a href="#">SYSTEM_TIMERGROUP_CLK_EN</a>	<a href="#">SYSTEM_TIMERGROUP_RST</a>
UART MEM	<a href="#">SYSTEM_UART_MEM_CLK_EN</a> <sup>4</sup>	<a href="#">SYSTEM_UART_MEM_RST</a>
APB SARADC	<a href="#">SYSTEM_APB_SARADC_CLK_EN</a>	<a href="#">SYSTEM_APB_SARADC_RST</a>
System Timer	<a href="#">SYSTEM_SYSTIMER_CLK_EN</a>	<a href="#">SYSTEM_SYSTIMER_RST</a>
ADC Controller	<a href="#">SYSTEM_ADC2_ARB_CLK_EN</a>	<a href="#">SYSTEM_ADC2_ARB_RST</a>
<b>Accelerators</b>	<b><a href="#">SYSTEM_PERIP_CLK_EN1_REG</a></b>	<b><a href="#">SYSTEM_PERIP_RST_EN1_REG</a></b>
ECC Accelerator	<a href="#">SYSTEM_CRYPT0_ECC_CLK_EN</a>	<a href="#">SYSTEM_CRYPT0_ECC_RST</a>
SHA Accelerator	<a href="#">SYSTEM_CRYPT0_SHA_CLK_EN</a>	<a href="#">SYSTEM_CRYPT0_SHA_RST</a>
DMA	<a href="#">SYSTEM_DMA_CLK_EN</a>	<a href="#">SYSTEM_DMA_RST</a> <sup>5</sup>
TSENS	<a href="#">SYSTEM_TSENS_CLK_EN</a>	<a href="#">SYSTEM_TSENS_RST</a>

<sup>1</sup> Set the clock enabling bit to 1 to enable the clock, and to 0 to disable the clock.

<sup>2</sup> Set the reset controlling bit to 1 to reset a peripheral, and to 0 to disable the reset.

<sup>3</sup> Reset registers cannot be cleared by hardware. Therefore, SW reset clear is required after setting the reset registers.

<sup>4</sup> UART memory is shared by all UART peripherals, meaning having any active UART peripherals will prevent the UART memory from entering the clock-gated state.

<sup>5</sup> When DMA is required for peripheral communications, for example, SPI and SHA, DMA clock should also be enabled.



## 13.4 Register Summary

The addresses in this section are relative to the base address of System Registers provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Peripheral Clock Control Registers</b>			
<a href="#">SYSTEM_CPU_PERI_CLK_EN_REG</a>	CPU peripheral clock enable register	0x0000	Varies
<a href="#">SYSTEM_CPU_PERI_RST_EN_REG</a>	CPU peripheral reset register	0x0004	R/W
<a href="#">SYSTEM_PERIP_CLK_EN0_REG</a>	SYSTEM peripheral clock enable register 1	0x0010	R/W
<a href="#">SYSTEM_PERIP_CLK_EN1_REG</a>	SYSTEM peripheral clock enable register 1	0x0014	R/W
<a href="#">SYSTEM_PERIP_RST_EN0_REG</a>	SYSTEM peripheral reset register 0	0x0018	R/W
<a href="#">SYSTEM_PERIP_RST_EN1_REG</a>	SYSTEM peripheral reset register 1	0x001C	R/W
<a href="#">SYSTEM_GDMA_CTRL_REG</a>	GDMA clock control register	0x003C	R/W
<a href="#">SYSTEM_CACHE_CONTROL_REG</a>	Cache clock control register	0x0040	R/W
<b>Clock Configuration Registers</b>			
<a href="#">SYSTEM_CPU_PER_CONF_REG</a>	CPU clock configuration register	0x0008	Varies
<a href="#">SYSTEM_SYSClk_CONF_REG</a>	System clock configuration register	0x0058	R/W
<b>CPU Interrupt Control Registers</b>			
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_0_REG</a>	CPU interrupt control register 0	0x0028	R/W
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_1_REG</a>	CPU interrupt control register 1	0x002C	R/W
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_2_REG</a>	CPU interrupt control register 2	0x0030	R/W
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_3_REG</a>	CPU interrupt control register 3	0x0034	R/W
<b>System and Memory Control Registers</b>			
<a href="#">SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG</a>	External memory encryption and decryption control register	0x0044	R/W
<b>Clock Gate Control Registers</b>			
<a href="#">SYSTEM_CLOCK_GATE_REG</a>	Clock gate control register	0x0054	R/W
<b>Date Register</b>			
<a href="#">SYSTEM_DATE_REG</a>	Version register	0x0FFC	R/W

The addresses in this section are relative to the base address of APB Controller provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Configuration Register</b>			
<a href="#">SYSCON_CLKGATE_FORCE_ON_REG</a>	Internal memory clock gate enable register	0x00A4	R/W
<a href="#">SYSCON_MEM_POWER_DOWN_REG</a>	Internal memory control register	0x00A8	R/W
<a href="#">SYSCON_MEM_POWER_UP_REG</a>	Internal memory control register	0x00AC	R/W

## 13.5 Registers

The addresses in this section are relative to the base address of System Registers provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 13.1. SYSTEM\_CPU\_PERI\_CLK\_EN\_REG (0x0000)

(reserved)																SYSTEM_CLK_EN_ASSIST_DEBUG				(reserved)			
31																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

Reset

**SYSTEM\_CLK\_EN\_ASSIST\_DEBUG** Set this bit to enable ASSIST\_DEBUG clock. Please see Chapter 14 *Debug Assistant (ASSIST\_DEBUG)* for more information about ASSIST\_DEBUG. (R/W)

Register 13.2. SYSTEM\_CPU\_PERI\_RST\_EN\_REG (0x0004)

(reserved)																SYSTEM_RST_EN_ASSIST_DEBUG				(reserved)			
31																7	6	5					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

Reset

**SYSTEM\_RST\_EN\_ASSIST\_DEBUG** Set this bit to reset ASSIST\_DEBUG. Please see Chapter 14 *Debug Assistant (ASSIST\_DEBUG)* for more information about ASSIST\_DEBUG. (R/W)

(reserved)  
 SYSTEM\_ADC2\_APB\_CLK\_EN  
 SYSTEM\_TIMER1\_CLK\_EN  
 SYSTEM\_APB\_SARADC\_CLK\_EN  
 (reserved)  
 SYSTEM\_UART\_MEM\_CLK\_EN  
 (reserved)  
 (reserved)  
 SYSTEM\_TIMERGROUP\_CLK\_EN  
 (reserved)  
 SYSTEM\_LEDC\_CLK\_EN  
 (reserved)  
 SYSTEM\_I2C\_EXT0\_CLK\_EN  
 SYSTEM\_SPI2\_CLK\_EN  
 SYSTEM\_UART1\_CLK\_EN  
 (reserved)  
 SYSTEM\_UART\_CLK\_EN  
 (reserved)  
 SYSTEM\_SPI01\_CLK\_EN

Reset

**SYSTEM\_ADC2\_ARB\_CLK\_EN** Set this bit to enable ADC2\_ARB clock. (R/W)

(reserved)

SYSTEM\_TSENS\_CLK\_EN

(reserved)

SYSTEM\_DMA\_CLK\_EN

(reserved)

SYSTEM\_CRYPTO\_SHA\_CLK\_EN

SYSTEM\_CRYPTO\_ECC\_CLK\_EN

(reserved)

Reset

**SYSTEM TSENS CLK EN** Set this bit to enable TSENS clock. (R/W)

[illegible]

**SYSTEM\_ADC2\_ARB\_RST** Set this bit to reset ADC2\_ARB. (R/W)

[illegible]

**SYSTEM\_TSENS\_RST** Set this bit to reset TSENS. (R/W)

Register 13.7. SYSTEM\_GDMA\_CTRL\_REG (0x003C)

(reserved)																															SYSTEM_GDMA_RESET SYSTEM_GDMA_CLK_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	1	Reset

- SYSTEM\_GDMA\_CLK\_ON Set this bit to enable GDMA clock. (R/W)
- SYSTEM\_GDMA\_RESET Set this bit to reset GDMA. (R/W)

Register 13.8. SYSTEM\_CACHE\_CONTROL\_REG (0x0040)

(reserved)																												SYSTEM_DCACHE_RESET SYSTEM_DCACHE_CLK_ON SYSTEM_ICACHE_RESET SYSTEM_ICACHE_CLK_ON				
31																												4	3	2	1	0
0 0																												0	1	0	1	Reset

- SYSTEM\_ICACHE\_CLK\_ON Set this bit to enable i-cache clock. (R/W)
- SYSTEM\_ICACHE\_RESET Set this bit to reset i-cache. (R/W)
- SYSTEM\_DCACHE\_CLK\_ON Set this bit to enable d-cache clock. (R/W)
- SYSTEM\_DCACHE\_RESET Set this bit to reset d-cache. (R/W)

**Register 13.9. SYSTEM\_CPU\_PER\_CONF\_REG (0x0008)**

(reserved)																								SYSTEM_CPU_WAITI_DELAY_NUM				SYSTEM_CPU_WAIT_MODE_FORCE_ON				(reserved)				SYSTEM_CPUPERIOD_SEL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																								8	7	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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

**SYSTEM\_CPUPERIOD\_SEL** Set this field to select the CPU clock frequency. For details, please refer to Table 6.2-3 in Chapter 6 *Reset and Clock*. (R/W)

**SYSTEM\_CPU\_WAIT\_MODE\_FORCE\_ON** Set this bit to force on the clock gate of CPU wait mode. Usually, after executing the WFI (Wait-for-Interrupt) instruction, CPU enters the wait mode, during which the clock gate of CPU is turned off until any interrupts occur. In this way, power consumption is saved. However, if this bit is set, the clock gate of CPU is always on and will not be turned off by the WFI instruction. (R/W)

**SYSTEM\_CPU\_WAITI\_DELAY\_NUM** Set the number of delay cycles to turn off the CPU clock gate after the CPU enters the wait mode because of a WFI instruction. (R/W)

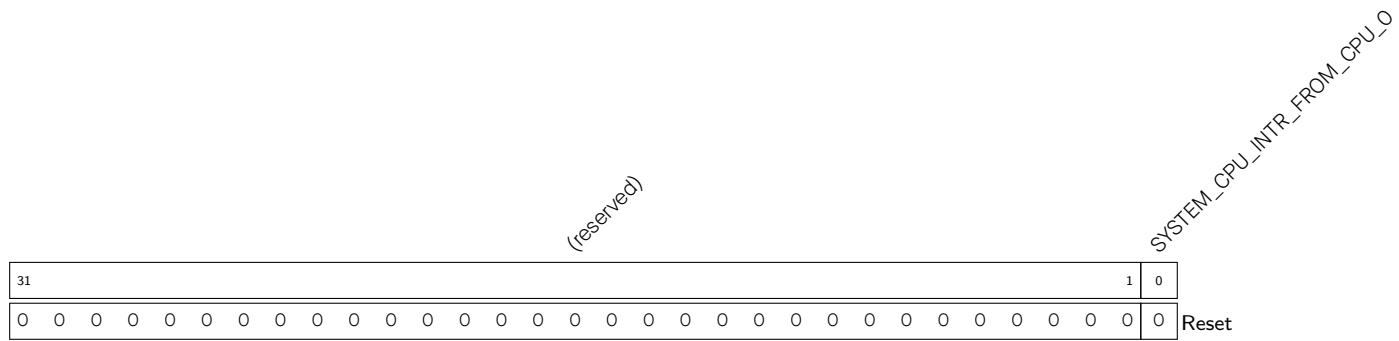
**Register 13.10. SYSTEM\_SYSClk\_CONF\_REG (0x0058)**

(reserved)												(reserved)				SYSTEM_SOC_CLK_SEL				SYSTEM_PRE_DIV_CNT											
31											19	18				12	11	10	9												0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x1											

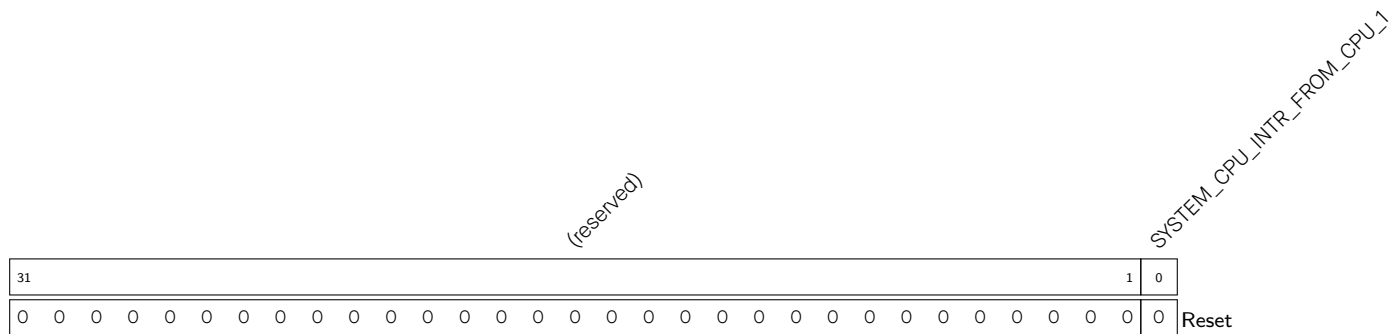
Reset

**SYSTEM\_PRE\_DIV\_CNT** This field is used to set the count of prescaler of XTAL\_CLK. For details, please refer to Table 6.2-2 in Chapter 6 *Reset and Clock*. (R/W)

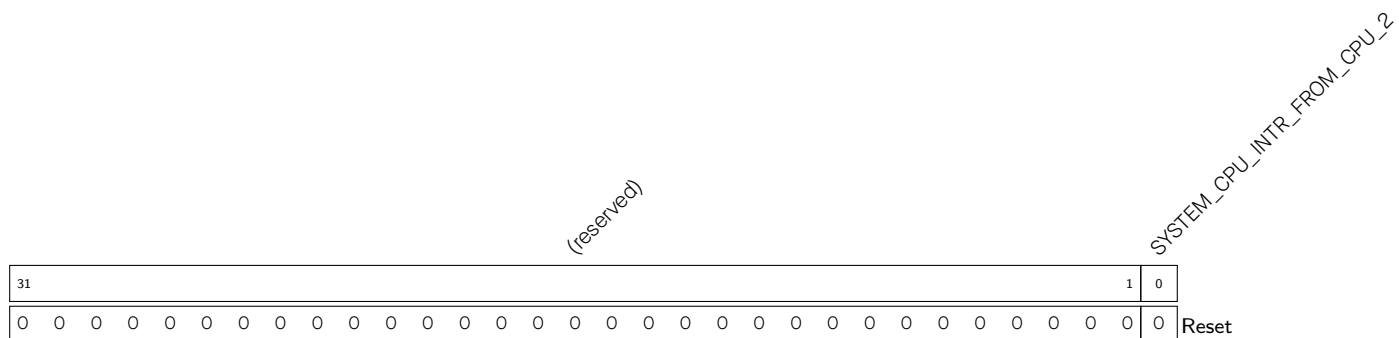
**SYSTEM\_SOC\_CLK\_SEL** This field is used to select SOC clock. For details, please refer to Table 6.2-3 in Chapter 6 *Reset and Clock*. (R/W)

**Register 13.11. SYSTEM\_CPU\_INTR\_FROM\_CPU\_0\_REG (0x0028)**

**SYSTEM\_CPU\_INTR\_FROM\_CPU\_0** Set this bit to generate CPU interrupt 0. This bit needs to be reset by software in the ISR process. (R/W)

**Register 13.12. SYSTEM\_CPU\_INTR\_FROM\_CPU\_1\_REG (0x002C)**

**SYSTEM\_CPU\_INTR\_FROM\_CPU\_1** Set this bit to generate CPU interrupt 1. This bit needs to be reset by software in the ISR process. (R/W)

**Register 13.13. SYSTEM\_CPU\_INTR\_FROM\_CPU\_2\_REG (0x0030)**

**SYSTEM\_CPU\_INTR\_FROM\_CPU\_2** Set this bit to generate CPU interrupt 2. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.14. SYSTEM\_CPU\_INTR\_FROM\_CPU\_3\_REG (0x0034)

(reserved)																																SYSTEM_CPU_INTR_FROM_CPU_3				
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	0	0	0	Reset

**SYSTEM\_CPU\_INTR\_FROM\_CPU\_3** Set this bit to generate CPU interrupt 3. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.15. SYSTEM\_EXTERNAL\_DEVICE\_ENCRYPT\_DECRYPT\_CONTROL\_REG (0x0044)

(reserved)																																SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
31																															4	3	2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SYSTEM\_ENABLE\_SPI\_MANUAL\_ENCRYPT** Set this bit to enable Manual Encryption under SPI Boot mode. (R/W)

**SYSTEM\_ENABLE\_DOWNLOAD\_DB\_ENCRYPT** Set this bit to enable Auto Encryption under Download Boot mode. (R/W)

**SYSTEM\_ENABLE\_DOWNLOAD\_GOCB\_DECRYPT** Set this bit to enable Auto Decryption under Download Boot mode. (R/W)

**SYSTEM\_ENABLE\_DOWNLOAD\_MANUAL\_ENCRYPT** Set this bit to enable Manual Encryption under Download Boot mode. (R/W)

Register 13.16. SYSTEM\_CLOCK\_GATE\_REG (0x0054)

(reserved)																																SYSTEM_CLK_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	0	0	1	Reset

**SYSTEM\_CLK\_EN** Set this bit to enable the system clock. (R/W)



**Register 13.17. SYSTEM\_DATE\_REG (0x0FFC)**

(reserved)				SYSTEM_DATE																												
31	28	27																														0
0	0	0	0	0x2108190																												Reset

**SYSTEM\_DATE** Version control register. (R/W)

The addresses in this section are relative to the base address of APB Controller provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 13.18. SYSCON\_CLKGATE\_FORCE\_ON\_REG (0x00A4)**

(reserved)																SYSCON_SRAM_CLKGATE_FORCE_ON										SYSCON_ROM_CLKGATE_FORCE_ON									
31																7	6	3			2	0													
0 0																0xf			7			Reset													

**SYSCON\_ROM\_CLKGATE\_FORCE\_ON** Set this field to configure the ROM clock gate to be always on; Set 0 to configure the clock gate to turn on automatically when ROM is accessed and turn off automatically when ROM is not accessed. (R/W)

**SYSCON\_SRAM\_CLKGATE\_FORCE\_ON** Set this field to configure the SRAM clock gate to be always on; Set 0 to configure the clock gate to turn on automatically when SRAM is accessed and turn off automatically when SRAM is not accessed. (R/W)

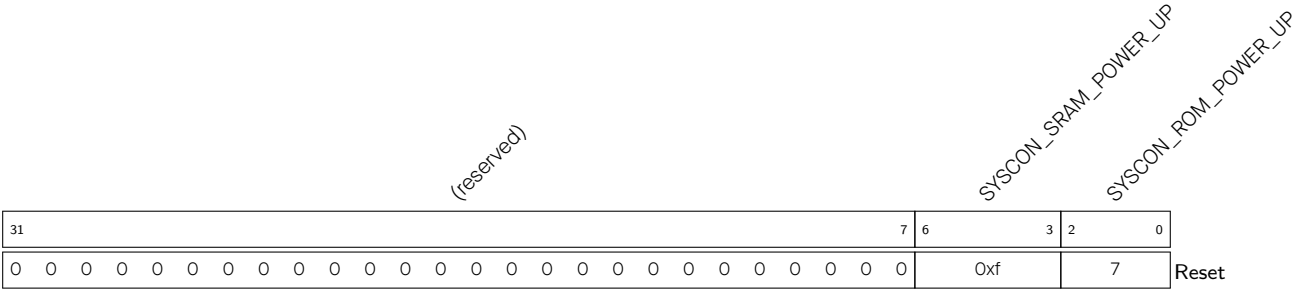
**Register 13.19. SYSCON\_MEM\_POWER\_DOWN\_REG (0x00A8)**

(reserved)																SYSCON_SRAM_POWER_DOWN		SYSCON_ROM_POWER_DOWN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31																7	6	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SYSCON\_ROM\_POWER\_DOWN** Set this field to send the internal ROM into retention state. (R/W)

**SYSCON\_SRAM\_POWER\_DOWN** Set this field to send the internal SRAM into retention state. (R/W)

Register 13.20. SYSCON\_MEM\_POWER\_UP\_REG (0x00AC)



- SYSCON\_ROM\_POWER\_UP** Set this field to force the internal ROM to work as normal (do not enter the retention state) when the chip enters light sleep. (R/W)
- SYSCON\_SRAM\_POWER\_UP** Set this field to force the internal SRAM to work as normal (do not enter the retention state) when the chip enters light sleep. (R/W)

## Chapter 14

### Debug Assistant (ASSIST\_DEBUG)

#### 14.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

#### 14.2 Features

The Debug Assistant module has the following features:

- Stack pointer (SP) monitoring
- Program counter (PC) logging before the CPU resets occurs
- CPU debugging status logging

#### 14.3 Functional Description

##### 14.3.1 SP Monitoring

The Debug Assistant module can monitor the SP so as to prevent stack overflow or erroneous push/pop. When the stack pointer exceeds the minimum or maximum thresholds, the Debug Assistant will record the PC's current value and generate an interrupt. Users can then read the recorded PC value to determine which instruction caused the out of bounds access. The minimum and maximum thresholds must be configured by software.

##### 14.3.2 PC Logging

In some cases, software developers want to know the PC at the last CPU reset. For instance, when the program is stuck and can only be reset, the developer may want to know where the program got stuck in order to debug. The Debug Assistant module can record the PC at the last CPU reset, which can be then read for software debugging.

##### 14.3.3 CPU Debugging Status Logging

The Debug Assistant module records the CPU debugging status by providing a set of read-only registers. Please refer to [1 ESP-RISC-V CPU](#) for more information.

## 14.4 Recommended Operation

### 14.4.1 SP Monitoring

SP bounds check monitoring:

- SP exceeds the upper bound address
- SP exceeds the lower bound address

The configuration process for SP monitoring is as follows:

1. Configure the monitored SP threshold with [ASSIST\\_DEBUG\\_CORE\\_O\\_SP\\_MIN\\_REG](#) and [ASSIST\\_DEBUG\\_CORE\\_O\\_SP\\_MAX\\_REG](#).
2. Configure interrupts.
  - Configure [ASSIST\\_DEBUG\\_CORE\\_O\\_INTR\\_EN\\_REG](#) to enable the interrupt of a monitoring mode.
  - Read [ASSIST\\_DEBUG\\_CORE\\_O\\_INTR\\_RAW\\_REG](#) to get the interrupt status of a monitoring mode.
  - Configure [ASSIST\\_DEBUG\\_CORE\\_O\\_INTR\\_CLR\\_REG](#) to clear their interrupts.
3. Configure [ASSIST\\_DEBUG\\_CORE\\_O\\_SP\\_MONITOR\\_EN\\_REG](#) to enable the monitoring mode(s). Various monitoring modes can be enabled at the same time.

Read [ASSIST\\_DEBUG\\_CORE\\_O\\_SP\\_PC](#) to get the PC value when an interrupt is triggered.

The interrupt of the Debug Assistant module corresponds to the interrupt source ASSIST\_DEBUG\_INTR of the interrupt matrix. For how to map the interrupt source to the CPU interrupt, please refer to the [8 Interrupt Matrix \(INTMTRX\)](#).

### 14.4.2 PC Logging Configuration Process

The CPU sends PC value to Debug Assistant. Only when [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_PDEBUGEN](#) is 1, the PC is valid, otherwise, it is always 0. Only when [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_RECORDEN](#) is 1, [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_PDEBUGPC\\_REG](#) samples the CPU's PC, otherwise, it keeps the original value.

The description of [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_EN\\_REG](#) and [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_PDEBUGPC\\_REG](#) can be found in section [14.8](#) and [14.9](#).

When the CPU resets, [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_EN\\_REG](#) will reset, while [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_PDEBUGPC\\_REG](#) will not. Therefore, the latter will keep the PC value at the CPU reset. [ASSIST\\_DEBUG\\_CORE\\_O\\_RCD\\_PDEBUGSP\\_REG](#) records the SP value at the reset.

## 14.5 Register Summary

The addresses in this section are relative to the Debug Assistant base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

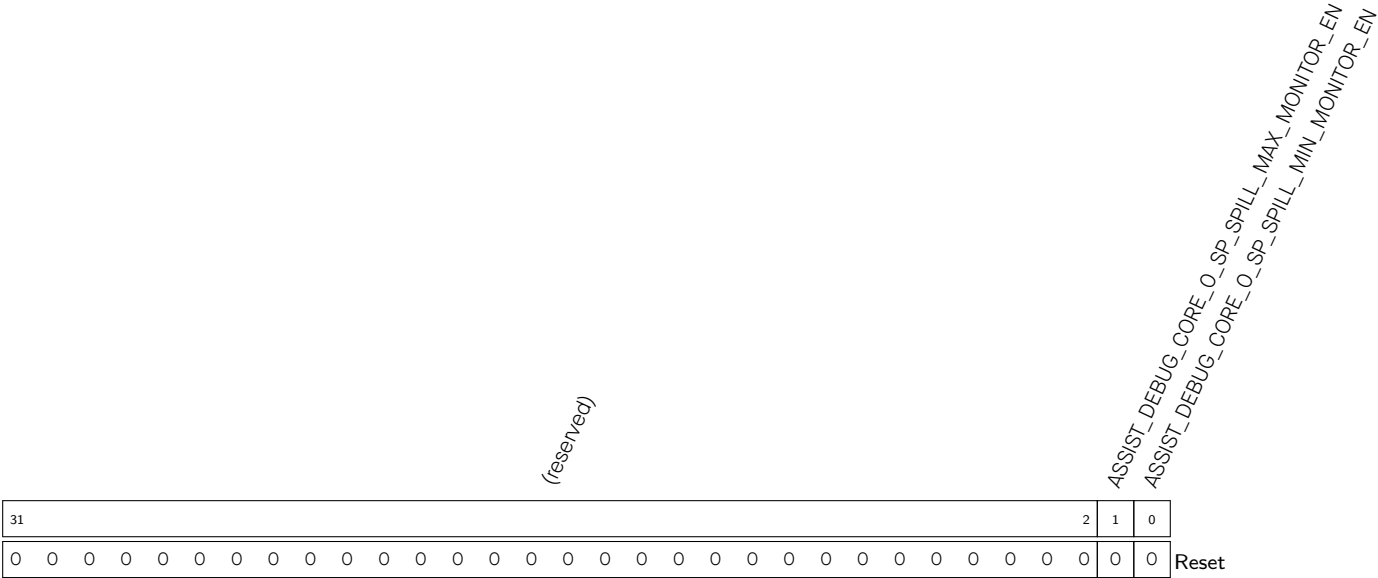
The abbreviations given in Column **Access** are explained in Section [Access Types for Registers](#).

Name	Description	Address	Access
<b>Monitor configuration registers</b>			
<a href="#">ASSIST_DEBUG_CORE_O_SP_MONITOR_EN_REG</a>	Configure monitoring modes	0x0000	R/W
<a href="#">ASSIST_DEBUG_CORE_O_SP_MIN_REG</a>	Configure stack min value	0x0010	R/W
<a href="#">ASSIST_DEBUG_CORE_O_SP_MAX_REG</a>	Configure stack max value	0x0014	R/W
<a href="#">ASSIST_DEBUG_CORE_O_SP_PC_REG</a>	Store PC value when an interrupt occurs	0x0018	RO
<b>Interrupt configuration registers</b>			
<a href="#">ASSIST_DEBUG_CORE_O_INTR_RAW_REG</a>	Store interrupt status of monitoring modes	0x0004	RO
<a href="#">ASSIST_DEBUG_CORE_O_INTR_EN_REG</a>	Enable interrupt of monitoring modes	0x0008	R/W
<a href="#">ASSIST_DEBUG_CORE_O_INTR_CLR_REG</a>	Clear interrupt of monitoring modes	0x000C	WT
<b>PC logging configuration register</b>			
<a href="#">ASSIST_DEBUG_CORE_O_RCD_EN_REG</a>	Enable PC logging	0x001C	R/W
<b>PC logging status registers</b>			
<a href="#">ASSIST_DEBUG_CORE_O_RCD_PDEBUGPC_REG</a>	Record PC value	0x0020	RO
<a href="#">ASSIST_DEBUG_CORE_O_RCD_PDEBUGSP_REG</a>	Record SP value	0x0024	RO
<b>CPU status registers</b>			
<a href="#">ASSIST_DEBUG_CORE_O_LASTPC_BEFORE_EXCEPTION_REG</a>	Store PC of the last command before CPU enters exception	0x0028	RO
<a href="#">ASSIST_DEBUG_CORE_O_DEBUG_MODE_REG</a>	Store CPU debug mode status	0x002C	RO
<b>Clock gate register</b>			
<a href="#">ASSIST_DEBUG_CLOCK_GATE_REG</a>	Clock gate register	0x0030	R/W
<b>Version register</b>			
<a href="#">ASSIST_DEBUG_DATE_REG</a>	Version control register	0x01FC	R/W

## 14.6 Registers

The addresses in this section are relative to the Debug Assistant base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

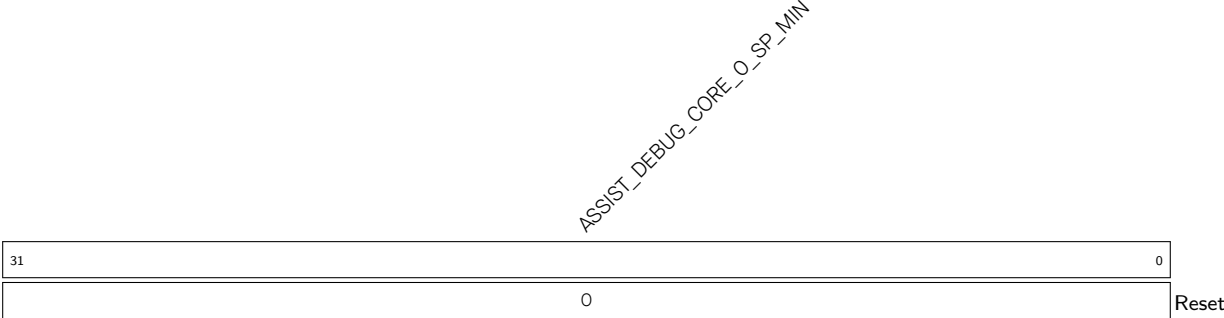
Register 14.1. ASSIST\_DEBUG\_CORE\_O\_SP\_MONITOR\_EN\_REG (0x0000)



**ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MIN\_MONITOR\_EN** Set 1 to enable SP underflow monitor.  
(R/W)

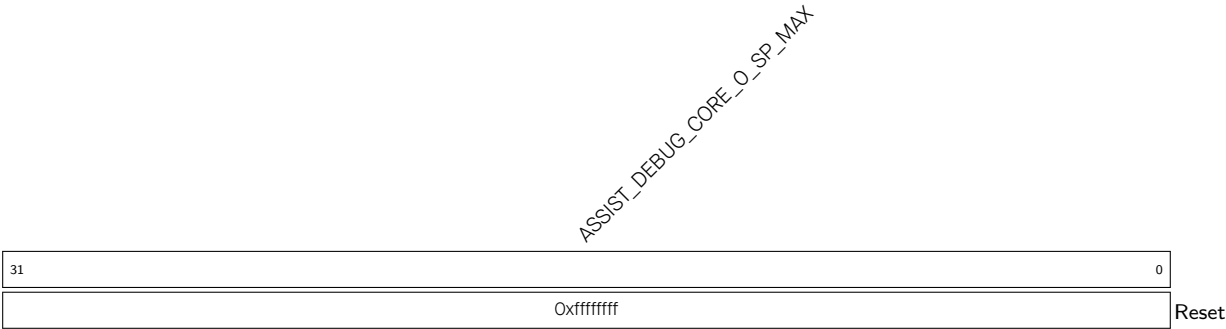
**ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MAX\_MONITOR\_EN** Set 1 to enable SP overflow monitor.  
(R/W)

Register 14.2. ASSIST\_DEBUG\_CORE\_O\_SP\_MIN\_REG (0x0010)



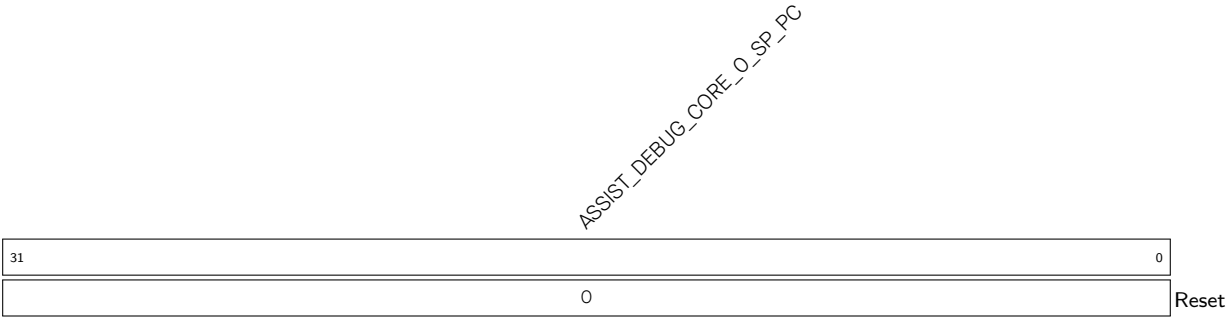
**ASSIST\_DEBUG\_CORE\_O\_SP\_MIN** Records the lower bound address of SP. (R/W)

Register 14.3. ASSIST\_DEBUG\_CORE\_O\_SP\_MAX\_REG (0x0014)



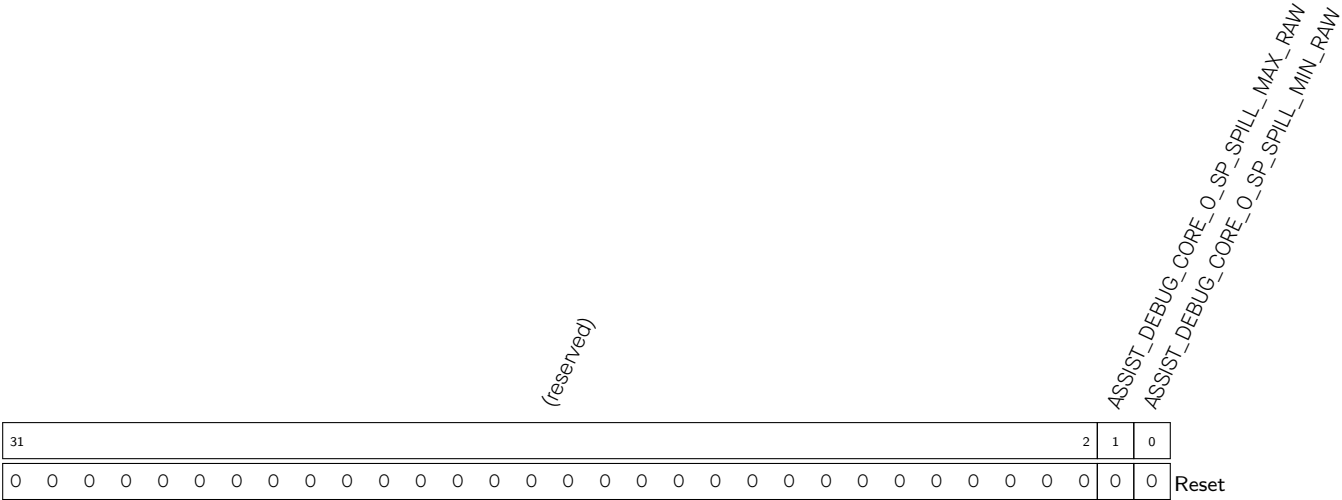
**ASSIST\_DEBUG\_CORE\_O\_SP\_MAX** Records the upper bound address of SP. (R/W)

Register 14.4. ASSIST\_DEBUG\_CORE\_O\_SP\_PC\_REG (0x0018)



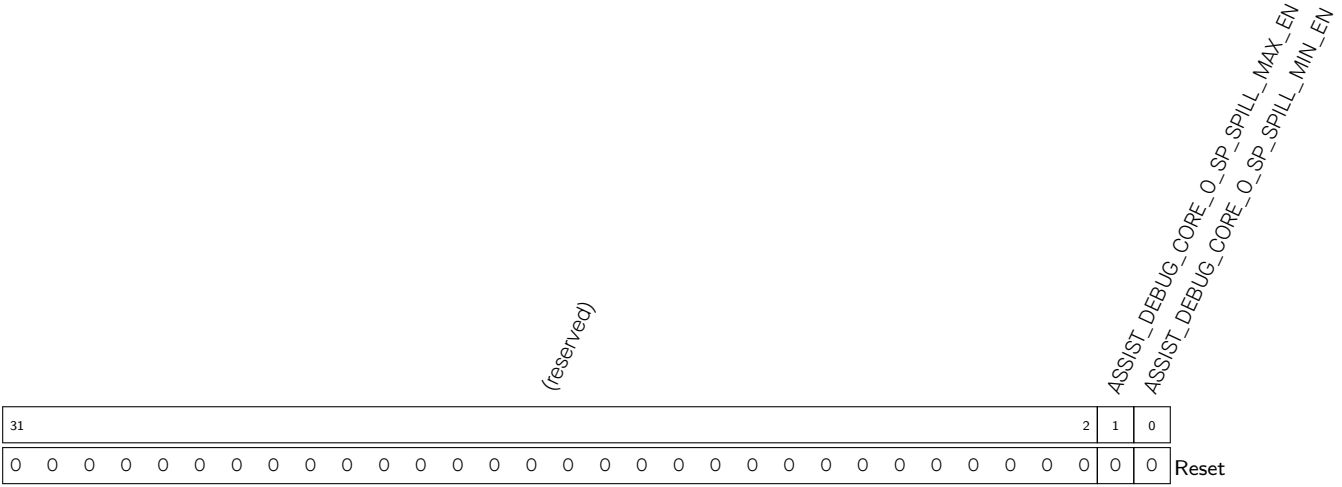
**ASSIST\_DEBUG\_CORE\_O\_SP\_PC** Records the PC value during stack monitoring. (RO)

Register 14.5. ASSIST\_DEBUG\_CORE\_O\_INTR\_RAW\_REG (0x0004)



- ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MIN\_RAW Stores the interrupt status of SP underflow monitoring. (RO)
- ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MAX\_RAW Stores the interrupt status of SP overflow monitoring. (RO)

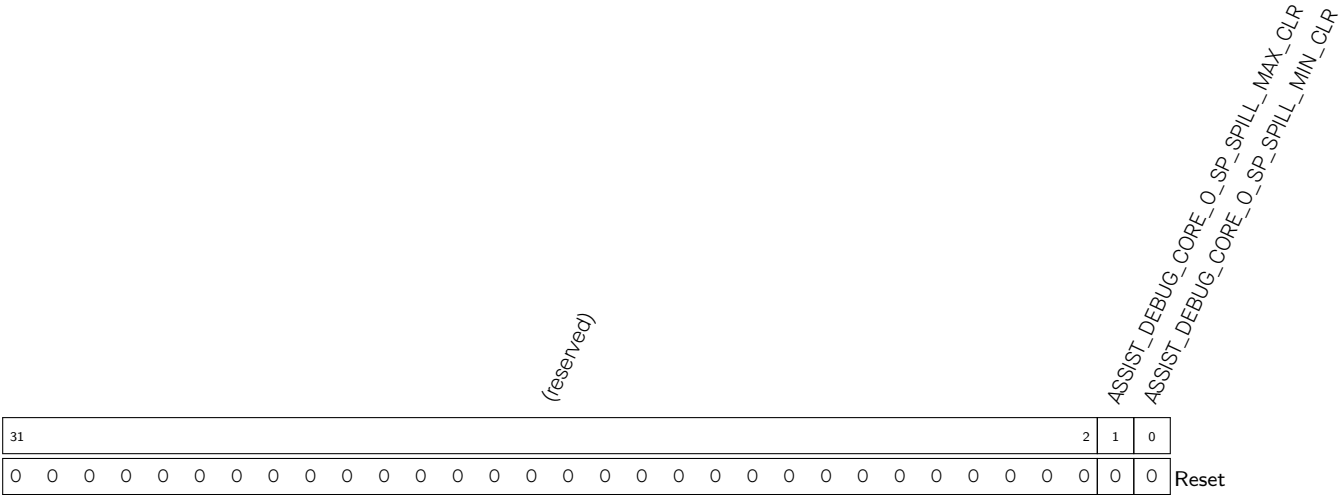
Register 14.6. ASSIST\_DEBUG\_CORE\_O\_INTR\_EN\_REG (0x0008)



- ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MIN\_EN SP underflow monitor interrupt enable bit, 1: interrupt enabled, 0: interrupt disabled. (R/W)
- ASSIST\_DEBUG\_CORE\_O\_SP\_SPILL\_MAX\_EN SP overflow monitor interrupt enable bit, 1: interrupt enabled, 0: interrupt disabled. (R/W)



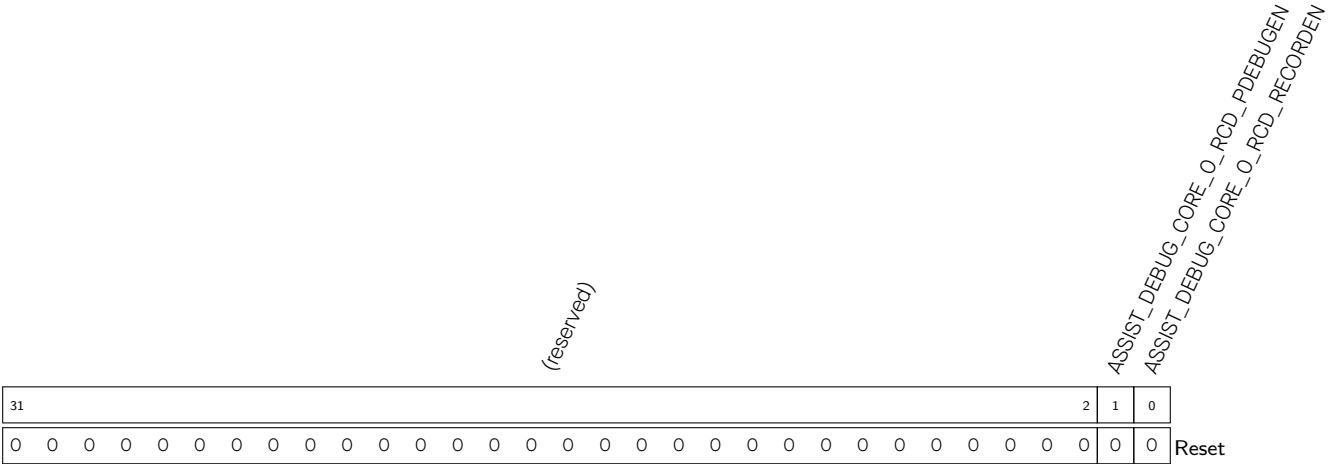
Register 14.7. ASSIST\_DEBUG\_CORE\_0\_INTR\_CLR\_REG (0x000C)



ASSIST\_DEBUG\_CORE\_0\_SP\_SPILL\_MIN\_CLR Set 1 to clear SP underflow monitor interrupt. (WT)

ASSIST\_DEBUG\_CORE\_0\_SP\_SPILL\_MAX\_CLR Set 1 to clear SP overflow monitor interrupt. (WT)

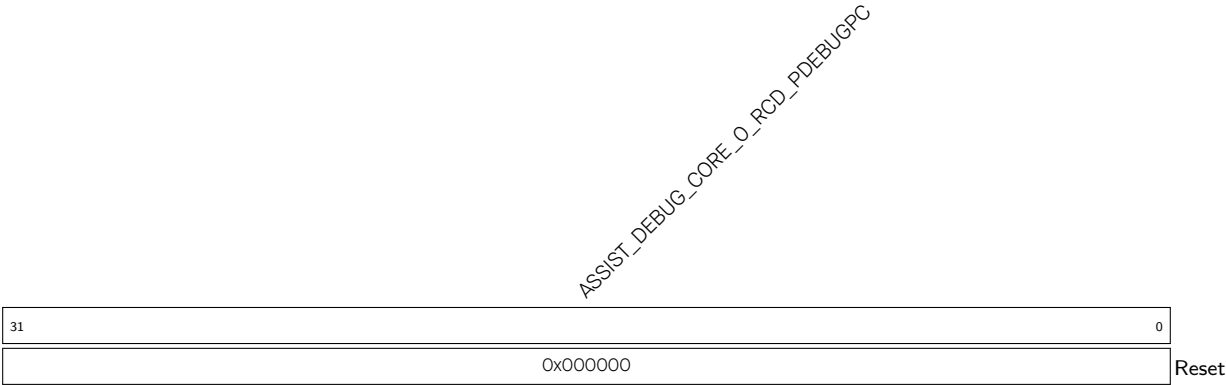
Register 14.8. ASSIST\_DEBUG\_CORE\_0\_RCD\_EN\_REG (0x001C)



ASSIST\_DEBUG\_CORE\_0\_RCD\_RECORDEN Set 1 to enable AS-  
SIST\_DEBUG\_CORE\_0\_RCD\_PDEBUGPC\_REG to record PC in real time. (R/W)

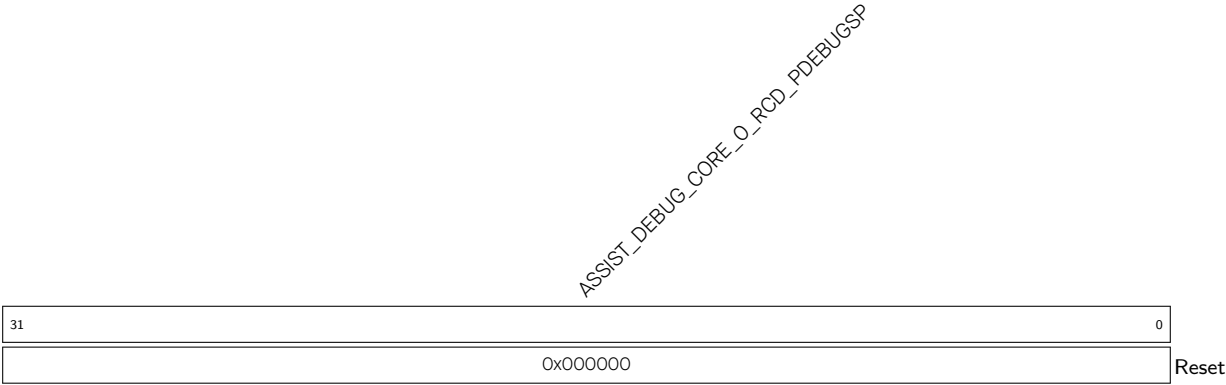
ASSIST\_DEBUG\_CORE\_0\_RCD\_PDEBUGEN Set 1 to enable CPU debug function. The CPU outputs  
PC only when this field is set to 1. (R/W)

Register 14.9. ASSIST\_DEBUG\_CORE\_O\_RCD\_PDEBUGPC\_REG (0x0020)



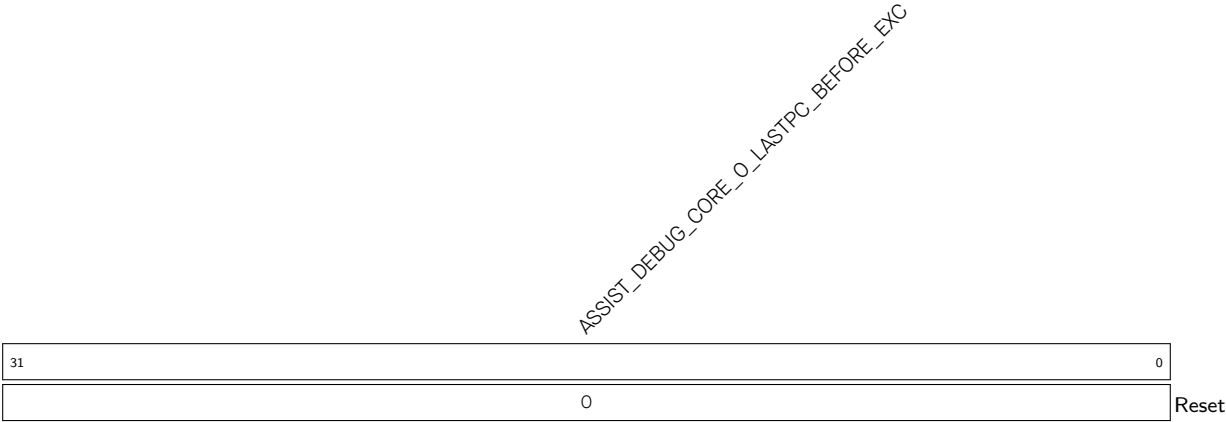
**ASSIST\_DEBUG\_CORE\_O\_RCD\_PDEBUGPC** Records the PC value at CPU reset. (RO)

Register 14.10. ASSIST\_DEBUG\_CORE\_O\_RCD\_PDEBUGSP\_REG (0x0024)



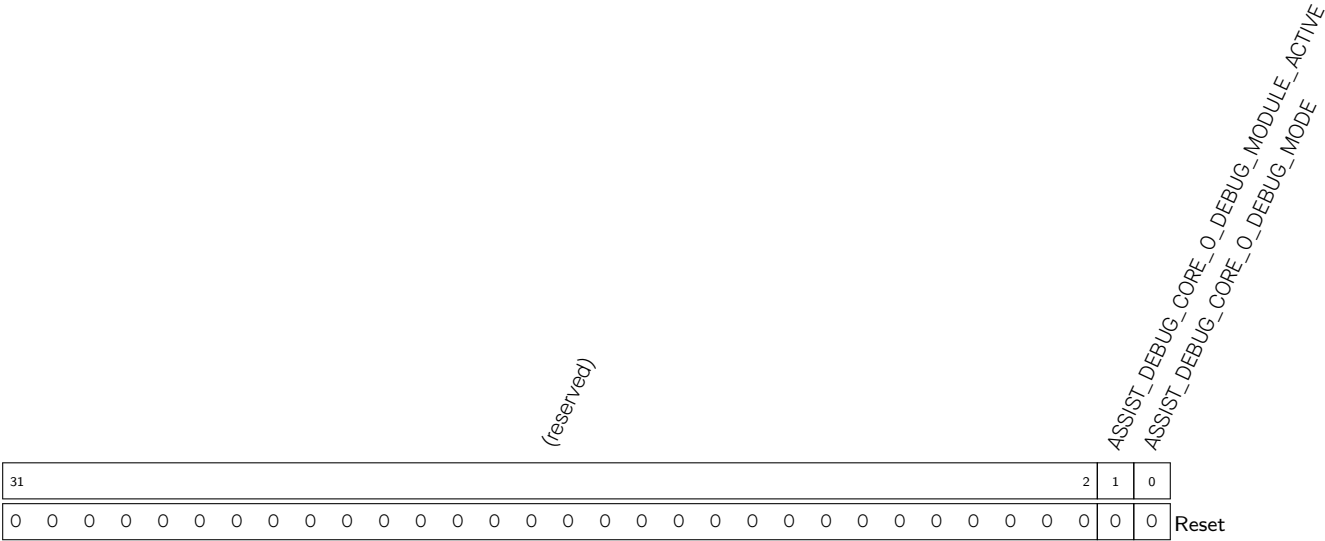
**ASSIST\_DEBUG\_CORE\_O\_RCD\_PDEBUGSP** Records SP. (RO)

Register 14.11. ASSIST\_DEBUG\_CORE\_O\_LASTPC\_BEFORE\_EXCEPTION\_REG (0x0028)



**ASSIST\_DEBUG\_CORE\_O\_LASTPC\_BEFORE\_EXC** Records the PC of the last instruction before the CPU enters exception. (RO)

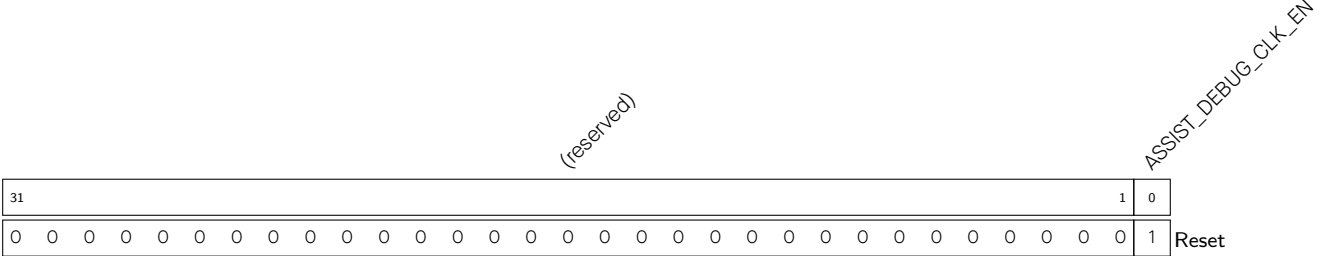
Register 14.12. ASSIST\_DEBUG\_CORE\_O\_DEBUG\_MODE\_REG (0x002C)



**ASSIST\_DEBUG\_CORE\_O\_DEBUG\_MODE** Indicates whether the RISC-V CPU is in debug mode. 1: in debug mode; 0: not in debug mode. (RO)

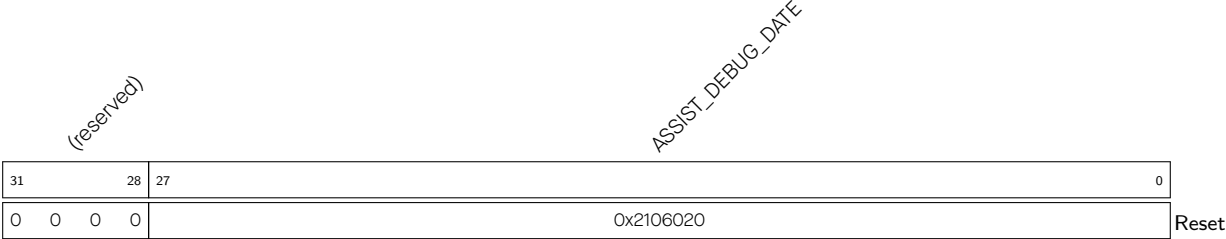
**ASSIST\_DEBUG\_CORE\_O\_DEBUG\_MODULE\_ACTIVE** Indicates the status of the RISC-V CPU debug module. 1: active status; 0: inactive status. (RO)

Register 14.13. ASSIST\_DEBUG\_CLOCK\_GATE\_REG (0x0030)



**ASSIST\_DEBUG\_CLK\_EN** Clock gate register. (R/W)

Register 14.14. ASSIST\_DEBUG\_DATE\_REG (0x01FC)



**ASSIST\_DEBUG\_DATE** Version control register. (R/W)

## Part IV

# Cryptography/Security Component

Dedicated to security features, this part explores cryptographic accelerators like SHA and ECC. It also covers external Memory Encryption and Decryption and random number generation, showcasing the SoC's capabilities in cryptography and secure data processing.

## Chapter 15

# ECC Hardware Accelerator (ECC)

## 15.1 Introduction

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves. ECC allows smaller keys compared to RSA cryptography while providing equivalent security.

ESP8684's ECC Accelerator can complete various calculation based on different elliptic curves, thus accelerating ECC algorithm and ECC-derived algorithms (such as ECDSA).

## 15.2 Features

ESP8684's ECC Accelerator supports:

- Two different elliptic curves, namely P-192 and P-256 defined in [FIPS 186-3](#)
- Seven working modes
- Interrupt upon completion of calculation

## 15.3 Terminology

To better illustrate the ECC accelerator, we will first introduce the terminology used in this chapter.

### 15.3.1 ECC Basics

#### 15.3.1.1 Elliptic Curve and Points on the Curves

The ECC algorithm is based on elliptic curves over prime fields, which can be represented as:

$$y^2 = x^3 + ax + b \bmod p$$

where,

- $p$  is a prime number.
- $a$  and  $b$  are two non-negative integers smaller than  $p$ .
- $(x, y)$  is a point on the curve satisfying the representation.

#### 15.3.1.2 Affine Coordinates and Jacobian Coordinates

An elliptic curve can be represented as below:

- In affine coordinates:

$$y^2 = x^3 + ax + b \bmod p$$

- In a Jacobian coordinates:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

To convert affine coordinates  $(x, y)$  to/from Jacobian coordinates  $(X, Y, Z)$ :

- From Affine to Jacobian coordinates

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- From Jacobian to affine coordinates

$$X = x$$

$$Y = y$$

$$Z = 1$$

## 15.3.2 ECC Definitions

### 15.3.2.1 Memory Blocks

ECC's memory blocks store input data and output data of the ECC operation.

**Table 15.3-1. ECC Accelerator Memory Blocks**

Memory	Size (byte)	Starting Address <sup>*</sup>	Ending Address <sup>*</sup>	Access
ECC_MULT_Mem_k	32	0x100	0x11F	R/W
ECC_MULT_Mem_Px	32	0x120	0x13F	R/W
ECC_MULT_Mem_Py	32	0x140	0x15F	R/W

<sup>\*</sup> Address offset relative to ECC accelerator base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

### 15.3.2.2 Data and Data Block

ESP8684's ECC operates on data of 256 bits. This data ( $D[255 : 0]$ ) can be divided into eight 32-bit data blocks  $D[n][31 : 0]$  ( $n = 0, 1, \dots, 7$ ). To be specific:

$$D[255 : 0] = D[7][31 : 0], D[6][31 : 0], D[5][31 : 0], D[4][31 : 0], D[3][31 : 0], D[2][31 : 0], D[1][31 : 0], D[0][31 : 0]$$

### 15.3.2.3 Write Data

Write data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be more specific, write data to an ECC memory block means write  $D[n][31 : 0]$  ( $n = 0, 1, \dots, 7$ ) to the "starting address of this ECC memory block  $+4 \times n$ " successively:

- write  $D[0]$  to "starting address"
- write  $D[1]$  to "starting address + 4"
- ...
- write  $D[7]$  to "starting address + 28"

**Note:**

When the key size of 192 bits is used, you need to append 0 before 192 bits of data and write 256 bits of data.

### 15.3.2.4 Read Data

Read data means reading data from the starting address and using this data as the output from the ECC algorithm. To be more specific, read data from an ECC memory block means read  $D[n][31:0]$  ( $n = 0, 1, \dots, 7$ ) from the “starting address of this ECC memory block +  $4 \times n$ ” successively:

- read  $D[0]$  from “starting address”
- read  $D[1]$  from “starting address + 4”
- ...
- read  $D[7]$  from “starting address + 28”

**Note:**

When the key size of 192 bits is used, only read 192 bit (6 blocks) of data.

### 15.3.2.5 Standard Calculation and Jacobian Calculation

ESP8684's ECC performs Base Point Calculation (including Base Point Verification and Base Point Multiplication) using the affine coordinates and Jacobian Calculation (including Jacobian Point Verification and Jacobian Point Multiplication) using the Jacobian coordinates.

## 15.4 Function Description

### 15.4.1 Key Size

ESP8684's ECC supports acceleration based on two key sizes (corresponding to two different elliptic curves). By configuring `ECC_MULT_KEY_LENGTH` field, users can choose desired key size. Details can be seen in Table 15.4-1 below.

Table 15.4-1. Choose ECC Accelerator Key Size

<code>ECC_MULT_KEY_LENGTH</code>	Elliptic Curves
1'b0	FIPS P-192
1'b1	FIPS P-256

<sup>1</sup> See definition of FIPS P-192 and P-256 in [FIPS 186-3](#).

### 15.4.2 Working Modes

ESP8684's ECC accelerator supports 7 working modes based on two elliptic curves described in the above section. By configuring `ECC_MULT_WORK_MODE` field, users can choose desired working mode. Details can be seen in Table 15.4-2.

Table 15.4-2. ECC Accelerator's Working Modes

ECC_MULT_WORK_MODE	Working Modes	ECC_MULT_WORK_MODE	Working Modes
3'd0	Point Multi Mode	3'd4	Jacobian Point Multi
3'd1	Division Mode	3'd5	<b>Reserved</b>
3'd2	Point Verif	3'd6	Jacobian Point Verif
3'd3	Point Verif + Multi	3'd7	Point Verification + Jacobian Multi

Detailed description about each working modes is provided in the following sections.

#### 15.4.2.1 Base Point Multiplication (Point Multi Mode)

Base Point Multiplication can be represented as:

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

where,

- Input:  $P_x$ ,  $P_y$ , and  $k$  are stored in [ECC\\_MULT\\_Mem\\_Px](#), [ECC\\_MULT\\_Mem\\_Py](#), and [ECC\\_MULT\\_Mem\\_k](#) respectively.
- Output:  $Q_x$  and  $Q_y$  are stored in [ECC\\_MULT\\_Mem\\_Px](#) and [ECC\\_MULT\\_Mem\\_Py](#) respectively.

#### 15.4.2.2 Finite Field Division (Division Mode)

Finite Field Division can be represented as:

$$\text{Result} = P_y \cdot k^{-1}$$

where,

- Input:  $P_y$  and  $k$  are stored in [ECC\\_MULT\\_Mem\\_Py](#) and [ECC\\_MULT\\_Mem\\_k](#).
- Output: Result is stored in [ECC\\_MULT\\_Mem\\_Py](#).

#### 15.4.2.3 Base Point Verification (Point Verif Mode)

Base Point Verification can be used to verify if a point  $(P_x, P_y)$  is on a selected elliptic curve.

- Input:  $P_x$  and  $P_y$  are stored in [ECC\\_MULT\\_Mem\\_Px](#) and [ECC\\_MULT\\_Mem\\_Py](#), respectively.
- Output: verification result is stored in [ECC\\_MULT\\_VERIFICATION\\_RESULT](#) field.

#### 15.4.2.4 Base Point Verification + Base Point Multiplication (Point Verif + Multi Mode)

In this working mode, ECC first verifies if Point  $(P_x, P_y)$  is on the selected elliptic curve or not. If yes, then perform the multiplication:

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

where,

- Input:  $P_x$ ,  $P_y$  and  $k$  are stored at [ECC\\_MULT\\_Mem\\_Px](#), [ECC\\_MULT\\_Mem\\_Py](#), and [ECC\\_MULT\\_Mem\\_k](#) respectively.



- Output:
  - verification result is stored in `ECC_MULT_VERIFICATION_RESULT` field.
  - $Q_x$  and  $Q_y$  are stored in `ECC_MULT_Mem_Px` and `ECC_MULT_Mem_Py` respectively.

#### 15.4.2.5 Jacobian Point Multiplication (Jacobian Point Multi Mode)

Jacobian Point Multiplication can be represented as:

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- $(Q_x, Q_y, Q_z)$  is a Jacobian point on the selected elliptic curve.
- 1 in the point's Jacobian coordinates is auto completed by hardware.
- Input:  $P_x$ ,  $P_y$  and  $k$  are stored in `ECC_MULT_Mem_Px`, `ECC_MULT_Mem_Py`, and `ECC_MULT_Mem_k` respectively.
- Output:  $Q_x$ ,  $Q_y$ , and  $Q_z$  are stored in `ECC_MULT_Mem_Px`, `ECC_MULT_Mem_Py`, and `ECC_MULT_Mem_k`, respectively.

#### 15.4.2.6 Jacobian Point Verification (Jacobian Point Verif Mode)

Jacobian Point Verification can be used to verify if a point  $(Q_x, Q_y, Q_z)$  is on a selected elliptic curve.

- $(Q_x, Q_y, Q_z)$  is the point in Jacobian Coordinates.
- Input:  $Q_x$ ,  $Q_y$ , and  $Q_z$  are stored in `ECC_MULT_Mem_Px`, `ECC_MULT_Mem_Py`, and `ECC_MULT_Mem_k`, respectively.
- Output: verification result is stored in `ECC_MULT_VERIFICATION_RESULT` field.

#### 15.4.2.7 Base Point Verification + Jacobian Point Multiplication (Point Verif + Jacobian Point Multi Mode)

In this working mode, ECC first verifies if Point  $(P_x, P_y)$  is on the selected elliptic curve or not. If yes, then perform the multiplication:

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- $(Q_x, Q_y, Q_z)$  is a Jacobian point on the selected elliptic curve.
- 1 in the point's Jacobian coordinates is auto completed by hardware.
- Input:  $P_x$ ,  $P_y$ , and  $k$  are stored in `ECC_MULT_Mem_Px`, `ECC_MULT_Mem_Py`, and `ECC_MULT_Mem_k`.
- Output:
  - verification result is stored in `ECC_MULT_VERIFICATION_RESULT` field.
  - $Q_x$ ,  $Q_y$ , and  $Q_z$  are stored in `ECC_MULT_Mem_Px`, `ECC_MULT_Mem_Py`, and `ECC_MULT_Mem_k`.

## 15.5 Clocks and Resets

ESP8684's ECC only has one clock module (crypto\_ecc\_clk) and one reset module (crypto\_ecc\_rst). Users should enable the ECC clock and disable the ECC reset before starting the ECC accelerator. For details on how to configure the ECC clock and reset, please refer to Chapter [6 Reset and Clock](#).

## 15.6 Interrupts

ESP8684's ECC accelerator can generate one interrupt signal ECC\_INTR and send it to [Interrupt Matrix](#).

**Note:**

Each interrupt signal is generated by any of its interrupts: any of its interrupt triggered can generate the interrupt signal.

ECC\_INTR has only one interrupt ECC\_MULT\_CALC\_DONE\_INT, which is triggered on the completion of an ECC computation.

This ECC interrupt ECC\_MULT\_CALC\_DONE\_INT is configured by the following registers:

- [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_RAW](#): stores the raw interrupt of [ECC\\_MULT\\_CALC\\_DONE\\_INT](#).
- [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_ST](#): indicates the status of the [ECC\\_MULT\\_CALC\\_DONE\\_INT](#) interrupt. This field is generated by enabling/disabling [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_RAW](#) field via [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_ENA](#).
- [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_ENA](#): enables/disables the [ECC\\_MULT\\_CALC\\_DONE\\_INT](#) interrupt.
- [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_CLR](#): set this bit to clear the [ECC\\_MULT\\_CALC\\_DONE\\_INT](#) interrupt status. By setting this bit to 1, fields [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_RAW](#) and [ECC\\_MULT\\_CALC\\_DONE\\_INT\\_ST](#) will be cleared.

## 15.7 Programming Procedures

The programming procedures for configuring ECC are described below:

1. Configure the ECC clock and reset.
2. Choose key size and working mode as described in Section [15.4](#).
3. Enable [ECC\\_MULT\\_CALC\\_DONE\\_INT](#) interrupt as described in Section [15.6](#).
4. Set [ECC\\_MULT\\_START](#) field to start ECC calculation.
5. Wait for the [ECC\\_MULT\\_CALC\\_DONE\\_INT](#) interrupt, which indicates the completion of the ECC calculation.
6. Check the result as described in Section [15.4](#).

## 15.8 Register Summary

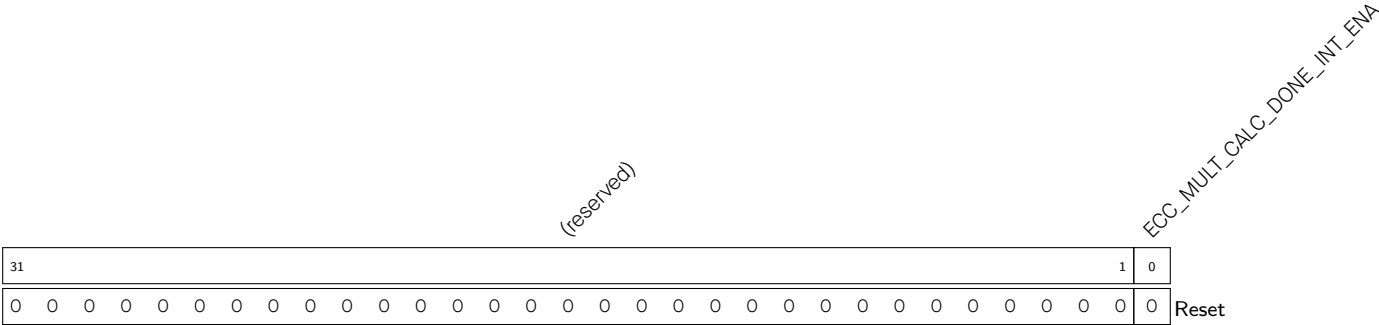
The addresses in this section are relative to ECC accelerator base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Interrupt Registers</b>			
<a href="#">ECC_MULT_INT_RAW_REG</a>	ECC raw interrupt status register	0x000C	RO/WTC/SS
<a href="#">ECC_MULT_INT_ST_REG</a>	ECC masked interrupt status register	0x0010	RO
<a href="#">ECC_MULT_INT_ENA_REG</a>	ECC interrupt enable register	0x0014	R/W
<a href="#">ECC_MULT_INT_CLR_REG</a>	ECC interrupt clear register	0x0018	WT
<b>Configuration Register</b>			
<a href="#">ECC_MULT_CONF_REG</a>	ECC configuration register	0x001C	varies
<b>Version Register</b>			
<a href="#">ECC_MULT_DATE_REG</a>	Version control register	0x00FC	R/W

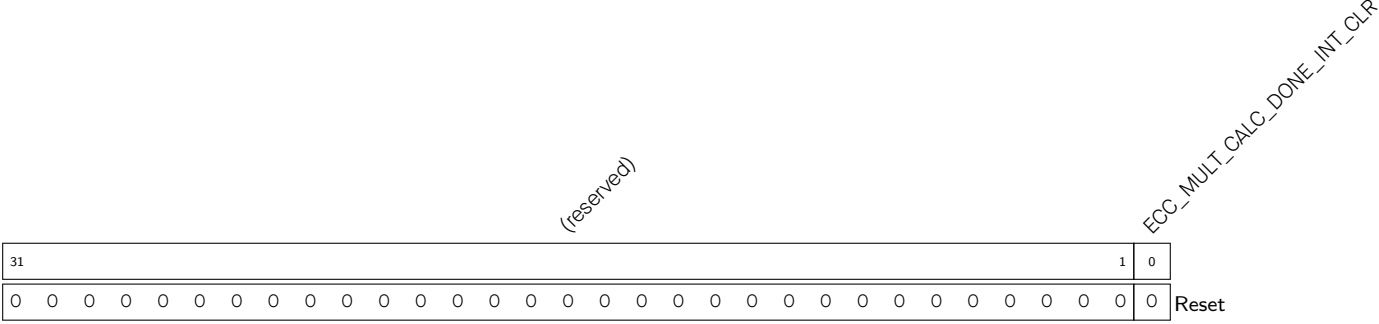


Register 15.3. ECC\_MULT\_INT\_ENA\_REG (0x0014)



ECC\_MULT\_CALC\_DONE\_INT\_ENA Write 1 to enable the ECC\_MULT\_CALC\_DONE\_INT interrupt.  
(R/W)

Register 15.4. ECC\_MULT\_INT\_CLR\_REG (0x0018)



ECC\_MULT\_CALC\_DONE\_INT\_CLR Write 1 to clear the ECC\_MULT\_CALC\_DONE\_INT interrupt. (WT)

### Register 15.5. ECC\_MULT\_CONF\_REG (0x001C)

[illegible]

**ECC\_MULT\_START** Write 1 to start calculation of ECC Accelerator. This bit will be self-cleared after the calculation is done. (R/W/SC)

**ECC\_MULT\_RESET** Write 1 to reset ECC Accelerator. (WT)

**ECC\_MULT\_KEY\_LENGTH** The key length mode bit of ECC Accelerator. 1'b0: P-192. 1'b1: P-256.  
(R/W)

**ECC\_MULT\_CLK\_EN** Write 1 to force on register clock gate. (R/W)

**ECC\_MULT\_WORK\_MODE** The work mode bits of ECC Accelerator. 3'd0: Point Multi Mode. 3'd1: Division mode. 3'd2: Point verification mode. 3'd3: Point Verif+Multi mode. 3'd4: Jacobian Point Multi Mode. 3'd5: Reserved. 3'd6: Jacobian Point Verification Mode. 3'd7: Point Verif + Jacobian Multi Mode. (R/W)

**ECC\_MULT\_VERIFICATION\_RESULT** The verification result bit of ECC Accelerator, only valid when calculation is done. (RO/SS)

### Register 15.6. ECC\_MULT\_DATE\_REG (0x00FC)

(reserved)				ECC_MULT_DATE																			
31	28	27																			0		
0	0	0	0	0x2012230																			Reset

**ECC\_MULT\_DATE** ECC Version control register. (R/W)

## Chapter 16

## SHA Accelerator (SHA)

### 16.1 Introduction

ESP8684 integrates an SHA accelerator, which is a hardware device that speeds up SHA algorithm significantly, compared to SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP8684 has two working modes, which are [Typical SHA](#) and [DMA-SHA](#).

### 16.2 Features

ESP8684 's SHA accelerator supports:

- Hash algorithms introduced in [FIPS PUB 180-4 Spec](#).
  - SHA-1
  - SHA-224
  - SHA-256
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleaved function
- Interrupt function when working in DMA-SHA working mode

### 16.3 Working Modes

The SHA accelerator integrated in ESP8684 has two working modes.

- [Typical SHA Working Mode](#): all the data is written and read via CPU directly.
- [DMA-SHA Working Mode](#): all the data is read via DMA. That is, users can configure the DMA controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

Users can start the SHA accelerator with different working modes by configuring registers [SHA\\_START\\_REG](#) and [SHA\\_DMA\\_START\\_REG](#). For details, please see Table [16.3-1](#).

**Table 16.3-1. SHA Accelerator Working Mode**

Working Mode	Configuration Method
<a href="#">Typical SHA</a>	Set <a href="#">SHA_START_REG</a> to 1

[DMA-SHA](#)Set [SHA\\_DMA\\_START\\_REG](#) to 1



Users can choose hash algorithms by configuring the [SHA\\_MODE\\_REG](#) register. For details, please see Table 16.3-2.

**Table 16.3-2. SHA Hash Algorithm Selection**

Hash Algorithm	<a href="#">SHA_MODE_REG</a> Configuration
SHA-1	0
SHA-224	1
SHA-256	2

## 16.4 Function Description

SHA accelerator can generate the message digest via two steps: [Preprocessing](#) and [Hash operation](#).

### 16.4.1 Preprocessing

Preprocessing consists of three steps: [padding the message](#), [parsing the message into message blocks](#) and [setting the initial hash value](#).

#### 16.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 bits. Thus, all the messages should be padded to a multiple of 512 bits before the hash task.

Suppose that the length of the message  $M$  is  $m$  bits. Then  $M$  shall be padded as introduced below:

1. First, append the bit “1” to the end of the message;
2. Second, append  $k$  bits of zeros, where  $k$  is the smallest, non-negative solution to the equation  $m + 1 + k \equiv 448 \bmod 512$ ;
3. Last, append the 64-bit block of value equal to the number  $m$  expressed using a binary representation.

For more details, please refer to Section “5.1 Padding the Message” in [FIPS PUB 180-4 Spec](#).

#### 16.4.1.2 Parsing the Message

The message and its padding must be parsed into  $N$  512-bit blocks,  $M^{(1)}$ ,  $M^{(2)}$ , ...,  $M^{(N)}$ . Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block  $i$  are denoted  $M_0^{(i)}$ , the next 32 bits are  $M_1^{(i)}$ , and so on up to  $M_{15}^{(i)}$ .

During the task, all the message blocks are written into the [SHA\\_M\\_n\\_REG](#):  $M_0^{(i)}$  is stored in [SHA\\_M\\_0\\_REG](#),  $M_1^{(i)}$  stored in [SHA\\_M\\_1\\_REG](#), ..., and  $M_{15}^{(i)}$  stored in [SHA\\_M\\_15\\_REG](#).

**Note:**

For more information about “message block”, please refer to Section “2.1 Glossary of Terms and Acronyms” in [FIPS PUB 180-4 Spec](#).

### 16.4.1.3 Setting the Initial Hash Value

Before hash task begins for any secure hash algorithms, the initial Hash value  $H(0)$  must be set based on different algorithms. However, the SHA accelerator uses the initial Hash values (constant C) stored in the hardware for hash tasks.

## 16.4.2 Hash Operation

After the preprocessing, the ESP8684 SHA accelerator starts to hash a message  $M$  and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP8684 SHA accelerator supports two working modes, which are [Typical SHA](#) and [DMA-SHA](#). The operation process for the SHA accelerator under two working modes is described in the following subsections.

### 16.4.2.1 Typical SHA Mode Process

Usually, the SHA accelerator will process all blocks of a message and produce a message digest before starting the computation of the next message digest.

However, ESP8684 SHA also supports optional “interleaved” message digest calculation. Users can insert new calculation each time the SHA accelerator completes a sequence of operations.

- In [Typical SHA](#) mode, this can be done after each individual message block.
- In [DMA-SHA](#) mode, this can be done after a full sequence of DMA operations is complete.

Specifically, users can read out the message digest from registers [SHA\\_H\\_n\\_REG](#) after completing part of a message digest calculation, and use the SHA accelerator for a different calculation. After the different calculation completes, users can restore the previous message digest to registers [SHA\\_H\\_n\\_REG](#), and resume the accelerator with the previously paused calculation.

#### Typical SHA Process

1. Select a hash algorithm.
  - Configure the [SHA\\_MODE\\_REG](#) register based on Table 16.3-2.
2. Process the current message block <sup>1</sup>.
  - Write the message block in registers [SHA\\_M\\_n\\_REG](#).
3. Start the SHA accelerator.
  - If this is the first time to execute this step, set the [SHA\\_START\\_REG](#) register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;
  - If this is not the first time to execute this step<sup>2</sup>, set the [SHA\\_CONTINUE\\_REG](#) register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the [SHA\\_H\\_n\\_REG](#) register to start calculation.
4. Check the progress of the current message block.
  - Poll register [SHA\\_BUSY\\_REG](#) until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the “idle” status <sup>3</sup>.

5. Decide if you have more message blocks to process:
  - If yes, please go back to Step 2.
  - Otherwise, please continue.
6. Obtain the message digest.
  - Read the message digest from registers [SHA\\_H\\_n\\_REG](#).

**Note:**

1. In this step, the software can also write the next message block (to be processed) in registers [SHA\\_M\\_n\\_REG](#), if any, while the hardware starts SHA calculation, to save time.
2. You are resuming the SHA accelerator with the previously paused calculation.
3. Here you can decide if you want to insert other calculations. If yes, please go to the [process for interleaved calculations](#) for details.

As mentioned above, ESP8684 SHA accelerator supports “interleaving” calculation under the **Typical SHA working mode**.

The process to implement interleaved calculation is described below.

1. Prepare to hand the SHA accelerator over for an interleaved calculation by storing the following data of the previous calculation.
  - The selected hash algorithm stored in the [SHA\\_MODE\\_REG](#) register.
  - The message digest stored in registers [SHA\\_H\\_n\\_REG](#).
2. Perform the interleaved calculation. For the detailed process of the interleaved calculation, please refer to [Typical SHA process](#) or [DMA-SHA process](#), depending on the working mode of your interleaved calculation.
3. Prepare to hand the SHA accelerator back to the previously paused calculation by restoring the following data of the previous calculation.
  - Write the previously stored hash algorithm back to register [SHA\\_MODE\\_REG](#).
  - Write the previously stored message digest back to registers [SHA\\_H\\_n\\_REG](#).
4. Write the next message block from the previous paused calculation in registers [SHA\\_M\\_n\\_REG](#), and set the [SHA\\_CONTINUE\\_REG](#) register to 1 to restart the SHA accelerator with the previously paused calculation.

#### 16.4.2.2 DMA-SHA Mode Process

ESP8684 SHA accelerator does not support “interleaving” message digest calculation at the level of individual message blocks when using DMA, which means you cannot insert new calculation before a complete DMA-SHA process (of one or more message blocks) completes. In this case, users who need interleaved operation are recommended to divide the message blocks and perform several DMA-SHA calculations, instead of trying to compute all the messages in one go.

Single DMA-SHA calculation supports up to 63 data blocks.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via DMA. Therefore, users are required to configure the DMA controller following the description in Chapter 2 [GDMA Controller \(GDMA\)](#).

### DMA-SHA process

1. Select a hash algorithm.
  - Select a hash algorithm by configuring the [SHA\\_MODE\\_REG](#) register. For details, please refer to Table 16.3-2.
2. Configure the [SHA\\_INT\\_ENA\\_REG](#) register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
  - Write the number of message blocks  $M$  to the [SHA\\_DMA\\_BLOCK\\_NUM\\_REG](#) register.
4. Start the DMA-SHA calculation.
  - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers [SHA\\_H\\_n\\_REG](#), then write 1 to register [SHA\\_DMA\\_CONTINUE\\_REG](#) to start SHA accelerator;
  - Otherwise, write 1 to register [SHA\\_DMA\\_START\\_REG](#) to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
  - The content of [SHA\\_BUSY\\_REG](#) register becomes 0, or
  - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the [SHA\\_INT\\_CLEAR\\_REG](#) register.
6. Obtain the message digest:
  - Read the message digest from registers [SHA\\_H\\_n\\_REG](#).

### 16.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers [SHA\\_H\\_n\\_REG](#) ( $n$ : 0~7). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 16.4-1 below:

Table 16.4-1. The Storage and Length of Message Digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage <sup>1</sup>
SHA-1	160	<a href="#">SHA_H_0_REG</a> ~ <a href="#">SHA_H_4_REG</a>
SHA-224	224	<a href="#">SHA_H_0_REG</a> ~ <a href="#">SHA_H_6_REG</a>
SHA-256	256	<a href="#">SHA_H_0_REG</a> ~ <a href="#">SHA_H_7_REG</a>

<sup>1</sup> The message digest is stored in registers from most significant bits to the least significant bits, with the first word stored in register [SHA\\_H\\_0\\_REG](#) and the second word stored in register [SHA\\_H\\_1\\_REG](#)... For details, please see subsection 16.4.1.2.

### 16.4.4 Interrupt

SHA accelerator supports interrupt on the completion of message digest calculation when working in the DMA-SHA mode. To enable this function, write 1 to register [SHA\\_INT\\_ENA\\_REG](#). Note that the interrupt should be cleared by software after use via setting the [SHA\\_INT\\_CLEAR\\_REG](#) register to 1.

## 16.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

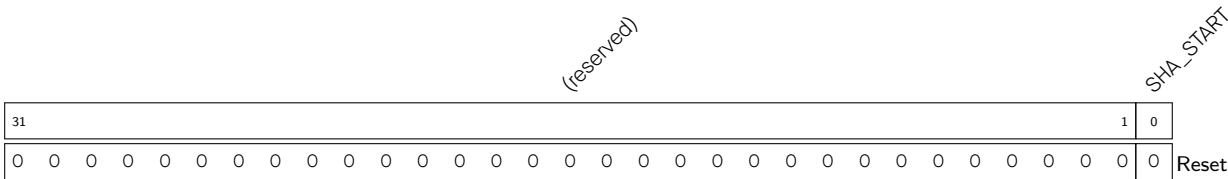
Name	Description	Address	Access
<b>Control/Status registers</b>			
<a href="#">SHA_CONTINUE_REG</a>	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
<a href="#">SHA_BUSY_REG</a>	Indicates if SHA Accelerator is busy or not	0x0018	RO
<a href="#">SHA_DMA_START_REG</a>	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
<a href="#">SHA_START_REG</a>	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
<a href="#">SHA_DMA_CONTINUE_REG</a>	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
<a href="#">SHA_INT_CLEAR_REG</a>	DMA-SHA interrupt clear register	0x0024	WO
<a href="#">SHA_INT_ENA_REG</a>	DMA-SHA interrupt enable register	0x0028	R/W
<b>Version Register</b>			
<a href="#">SHA_DATE_REG</a>	Version control register	0x002C	R/W
<b>Configuration Registers</b>			
<a href="#">SHA_MODE_REG</a>	Defines the algorithm of SHA accelerator	0x0000	R/W
<b>Data Registers</b>			
<a href="#">SHA_DMA_BLOCK_NUM_REG</a>	Block number register (only effective for DMA-SHA)	0x000C	R/W
<a href="#">SHA_H_0_REG</a>	Hash value	0x0040	R/W
<a href="#">SHA_H_1_REG</a>	Hash value	0x0044	R/W
<a href="#">SHA_H_2_REG</a>	Hash value	0x0048	R/W
<a href="#">SHA_H_3_REG</a>	Hash value	0x004C	R/W
<a href="#">SHA_H_4_REG</a>	Hash value	0x0050	R/W
<a href="#">SHA_H_5_REG</a>	Hash value	0x0054	R/W
<a href="#">SHA_H_6_REG</a>	Hash value	0x0058	R/W
<a href="#">SHA_H_7_REG</a>	Hash value	0x005C	R/W
<a href="#">SHA_M_0_REG</a>	Message	0x0080	R/W
<a href="#">SHA_M_1_REG</a>	Message	0x0084	R/W
<a href="#">SHA_M_2_REG</a>	Message	0x0088	R/W
<a href="#">SHA_M_3_REG</a>	Message	0x008C	R/W

Name	Description	Address	Access
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	0x00AC	R/W
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W

## 16.6 Registers

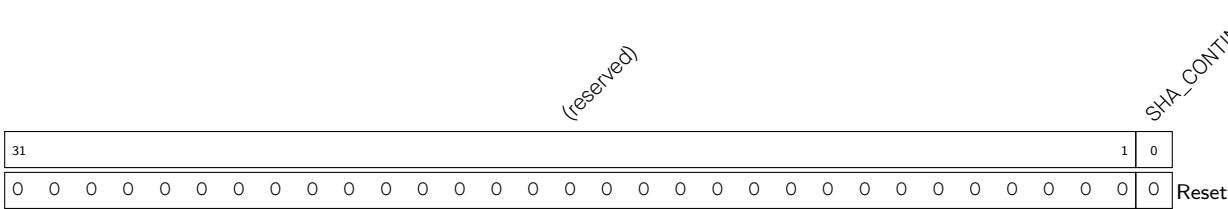
The addresses in this section are relative to the SHA accelerator base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 16.1. SHA\_START\_REG (0x0010)



**SHA\_START** Write 1 to start Typical SHA calculation. (WO)

Register 16.2. SHA\_CONTINUE\_REG (0x0014)



**SHA\_CONTINUE** Write 1 to continue Typical SHA calculation. (WO)

**Register 16.3. SHA\_BUSY\_REG (0x0018)**

(reserved)																															SHA_BUSY_STATE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

**SHA\_BUSY\_STATE** Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

**Register 16.4. SHA\_DMA\_START\_REG (0x001C)**

(reserved)																															SHA_DMA_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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SHA\_DMA\_START** Write 1 to start DMA-SHA calculation. (WO)

**Register 16.5. SHA\_DMA\_CONTINUE\_REG (0x0020)**

(reserved)																															SHA_DMA_CONTINUE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

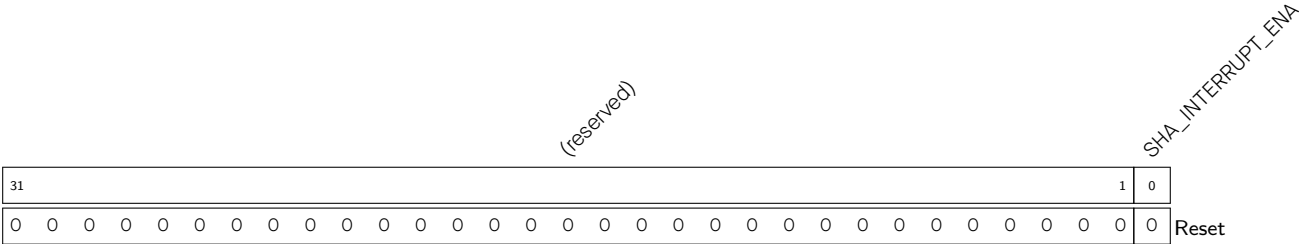
**SHA\_DMA\_CONTINUE** Write 1 to continue DMA-SHA calculation. (WO)

**Register 16.6. SHA\_INT\_CLEAR\_REG (0x0024)**

(reserved)																															SHA_CLEAR_INTERRUPT	
31																															1	0
0 0																															0	Reset

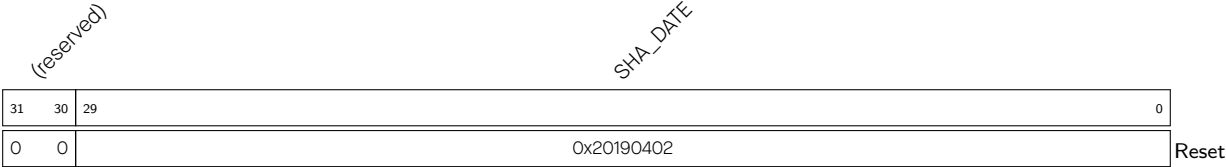
**SHA\_CLEAR\_INTERRUPT** Clears DMA-SHA interrupt. (WO)

Register 16.7. SHA\_INT\_ENA\_REG (0x0028)



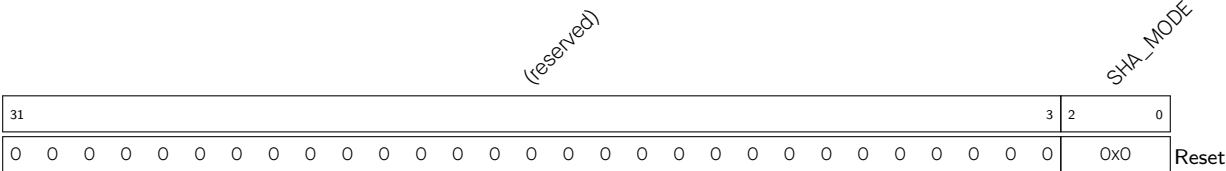
SHA\_INTERRUPT\_ENA Enables DMA-SHA interrupt. (R/W)

Register 16.8. SHA\_DATE\_REG (0x002C)



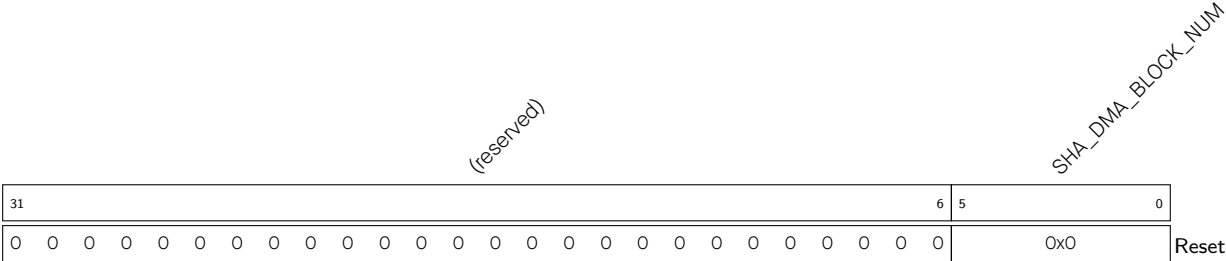
SHA\_DATE Version control register. (R/W)

Register 16.9. SHA\_MODE\_REG (0x0000)



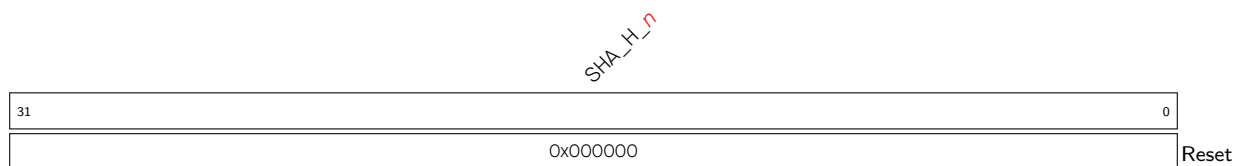
SHA\_MODE Defines the SHA algorithm. For details, please see Table 16.3-2. (R/W)

Register 16.10. SHA\_DMA\_BLOCK\_NUM\_REG (0x000C)

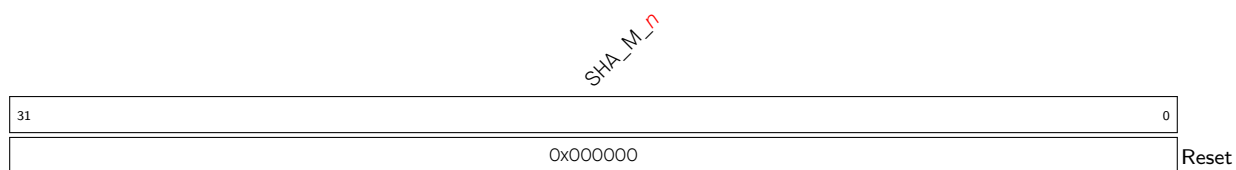


SHA\_DMA\_BLOCK\_NUM Defines the DMA-SHA block number. (R/W)



**Register 16.11. SHA\_H\_***n***\_REG (*n*: 0-7) (0x0040+4\**n*)**

**SHA\_H\_***n* Stores the *n*th 32-bit piece of the Hash value. (R/W)

**Register 16.12. SHA\_M\_***n***\_REG (*n*: 0-15) (0x0080+4\**n*)**

**SHA\_M\_***n* Stores the *n*th 32-bit piece of the message. (R/W)

## Chapter 17

# External Memory Encryption and Decryption (XTS\_AES)

## 17.1 Overview

The ESP8684 integrates an External Memory Encryption and Decryption module that complies with the XTS\_AES standard algorithm specified in [IEEE Std 1619-2007](#), providing security for users' application code and data stored in the external memory (flash). Users can store proprietary firmware and sensitive data (e.g., credentials for gaining access to a private network) to the external flash.

## 17.2 Features

This module supports the following features:

- General XTS\_AES algorithm, compliant with IEEE Std 1619-2007
- Software-triggered manual encryption
- High-speed auto decryption, without software's participation
- Encryption and decryption functions jointly determined by register configurations, eFuse parameters, and Boot modes

## 17.3 Module Structure

The External Memory Encryption and Decryption module consists of two blocks, namely the Manual Encryption block and Auto Decryption block. The module architecture is shown in Figure [17.3-1](#).

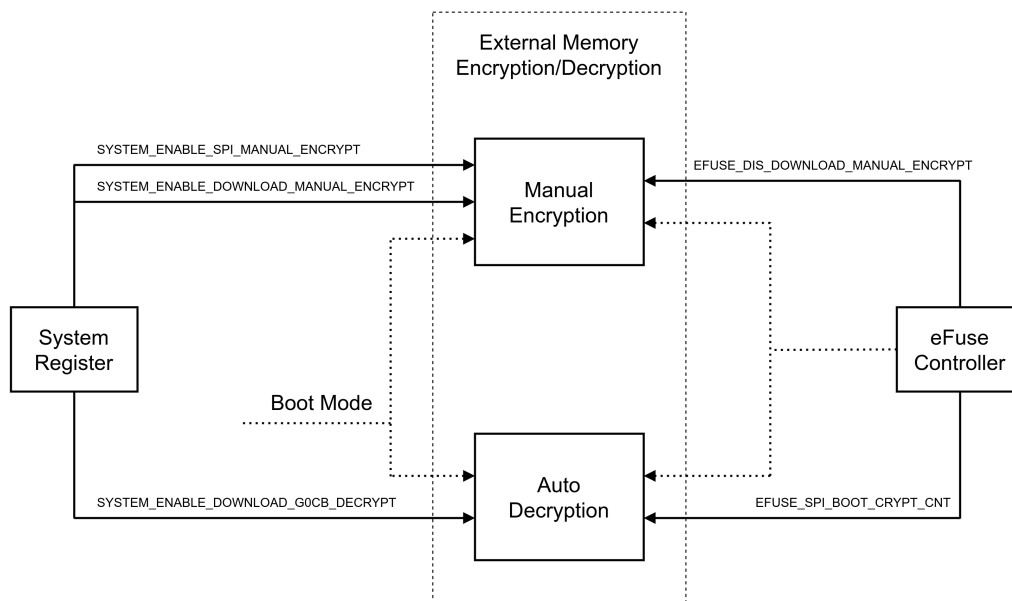


Figure 17.3-1. Architecture of the External Memory Encryption and Decryption

The Manual Encryption block can encrypt instructions/data which will then be written to the external flash as ciphertext via SPI1.

In the System Registers (SYSREG) peripheral (see Chapter 13 [System Registers \(SYSTEM\)](#)), the following four bits in register [SYSTEM\\_EXTERNAL\\_DEVICE\\_ENCRYPT\\_DECRYPT\\_CONTROL\\_REG](#) are relevant to the external memory encryption and decryption:

- [SYSTEM\\_ENABLE\\_DOWNLOAD\\_MANUAL\\_ENCRYPT](#)
- [SYSTEM\\_ENABLE\\_DOWNLOAD\\_GOCB\\_DECRYPT](#)
- [SYSTEM\\_ENABLE\\_DOWNLOAD\\_DB\\_ENCRYPT](#)
- [SYSTEM\\_ENABLE\\_SPI\\_MANUAL\\_ENCRYPT](#)

The XTS\_AES module also fetches two parameters from the peripheral eFuse Controller, which are: [EFUSE\\_DIS\\_DOWNLOAD\\_MANUAL\\_ENCRYPT](#) and [EFUSE\\_SPI\\_BOOT\\_ENCRYPT\\_DECRYPT\\_CNT](#). For detailed information, please see Chapter 4 [eFuse Controller \(eFuse\)](#).

## 17.4 Functional Description

### 17.4.1 XTS Algorithm

The manual encryption and auto decryption use the XTS algorithm. During implementation, the XTS algorithm is characterized by a "data unit" of 1024 bits, defined in the Section *XTS-AES encryption procedure* of [XTS-AES Tweakable Block Cipher](#) Standard. For more information about XTS-AES algorithm, please refer to [IEEE Std 1619-2007](#).

### 17.4.2 Key

The Manual Encryption block and Auto Decryption block share the same *Key* when implementing XTS algorithm. The *Key* is provided by the eFuse hardware and cannot be accessed by users.

The *Key* is 256-bit long. The value of the *Key* is determined by the eFuse parameters. For easier description, we define:

- *Key<sub>A</sub>*: the lowest 128-bit of BLOCK3 in eFuse.
- *Key<sub>B</sub>*: the highest 128-bit of BLOCK3 in eFuse.

There are two possibilities of how the *Key* is generated depending on the value of [EFUSE\\_XTS\\_KEY\\_LENGTH\\_256](#). In each case, the *Key* can be uniquely determined by *Key<sub>A</sub>* and *Key<sub>B</sub>* as shown in Table 17.4-1.

**Table 17.4-1. *Key* Generated Based on *Key<sub>A</sub>*, *Key<sub>B</sub>***

<a href="#">EFUSE_XTS_KEY_LENGTH_256</a>	<i>Key</i>	<i>Key</i> Length (bit)
1	{ <i>Key<sub>B</sub></i> , <i>Key<sub>A</sub></i> }	256
0	<i>SHA</i> – 256( <i>Key<sub>A</sub></i> ) <sup>1</sup>	256

<sup>1</sup> "SHA-256" indicates the SHA-256 algorithm, please refer to Chapter 16 [SHA Accelerator \(SHA\)](#).

### 17.4.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory (flash) where the first encrypted ciphertext is stored. The target memory space can be uniquely determined by two relevant parameters: target size and base address, whose definitions are listed below.

- Target size: the *size* of the target memory space, indicating the number of bytes encrypted in one encryption operation, which supports 16 or 32 bytes.
- Base address: the *base\_addr* of the target memory space. It is a 24-bit physical address, with range of 0x0000\_0000 ~ 0x00FF\_FFFF. It should be aligned to *size*, i.e., *base\_addr*%*size* == 0.

For example, if there are 16 bytes of instruction data need to be encrypted and written to address 0x130 ~ 0x13F in the external flash, then the target space is 0x130 ~ 0x13F, size is 16 (bytes), and base address is 0x130.

The encryption of any length (must be multiples of 16 bytes) of plaintext instruction/data can be completed separately in multiple operations, and each operation has its individual target memory space and the relevant parameters.

For Auto Decryption blocks, these parameters are automatically determined by hardware. For Manual Encryption blocks, these parameters should be configured by users.

**Note:**

The "tweak" defined in Section *Data units and tweaks* of [IEEE Std 1619-2007](#) is a 128-bit non-negative integer (*tweak*), which can be generated according to *tweak* = (*base\_addr* & 0x00FFF80). The lowest 7 bits and the highest 97 bits in *tweak* are always zero.

## 17.4.4 Data Writing

For Auto Decryption blocks, data writing is automatically applied in hardware. For Manual Encryption blocks, data writing should be applied by users. The Manual Encryption block has a register block which consists of 8 registers, i.e., `XTS_AES_PLAIN_n_REG` ( $n$ : 0 ~ 7), that are dedicated to data writing and can store up to 256 bits of plaintext at a time.

Actually, the Manual Encryption block does not care where the plaintext comes from, but only where the ciphertext will be stored. Because of the strict correspondence between plaintext and ciphertext, in order to better describe how the plaintext is stored in the register block, we assume that the plaintext is stored in the target memory space in the first place and replaced by ciphertext after encryption. Therefore, the following description no longer has the concept of “plaintext”, but uses “target memory space” instead. Please note that the plaintext can come from everywhere in actual use, but users should understand how the plaintext is stored in the register block.

**How mapping between target memory space and registers works:**

Assume a word in the target memory space is stored in *address*, define  $offset = address \% 32$ ,  $n = \frac{offset}{4}$ , then the word will be stored in register `XTS_AES_PLAIN_n_REG`.

For example, when the target size is 32, all registers in the register block will be used. The mapping between *offset* and registers is shown in Table 17.4-2.

**Table 17.4-2. Mapping Between Offsets and Registers**

<i>offset</i>	Register	<i>offset</i>	Register
0x00	<code>XTS_AES_PLAIN_0_REG</code>	0x10	<code>XTS_AES_PLAIN_4_REG</code>
0x04	<code>XTS_AES_PLAIN_1_REG</code>	0x14	<code>XTS_AES_PLAIN_5_REG</code>
0x08	<code>XTS_AES_PLAIN_2_REG</code>	0x18	<code>XTS_AES_PLAIN_6_REG</code>
0x0C	<code>XTS_AES_PLAIN_3_REG</code>	0x1C	<code>XTS_AES_PLAIN_7_REG</code>

## 17.4.5 Manual Encryption Block

The Manual Encryption block is a peripheral module. It is equipped with registers and can be accessed by the CPU directly. Registers embedded in this block, the System Registers (SYSREG) peripheral, eFuse parameters, and boot modes jointly configure and use this module. Please note that the Manual Encryption block can only encrypt for storage in external flash.

**Manual encryption is allowed only when the Manual Encryption block has operation permissions.** Whether the Manual Encryption block has operation permissions depends on:

- In SPI Boot mode

If bit `SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT` in register

`SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` is 1, the Manual Encryption block has operation permissions. Otherwise, it is not operational.

- In Download Boot mode

If bit `SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT` in register

`SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` is 1 and the eFuse parameter

EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT is 0, the Manual Encryption block has operation permissions. Otherwise, it is not operational.

**Note:**

- Even though the CPU can skip cache and get the encrypted instruction/data directly by reading the external memory, users can by no means access *Key*.

### 17.4.6 Auto Decryption Block

The Auto Decryption block is not a conventional peripheral, so it does not have any registers and cannot be accessed by the CPU directly. The System Registers (SYSREG) peripheral, eFuse parameters, and boot modes jointly configure and use this block.

**Auto decryption is allowed only when the Auto Decryption block has operation permissions.** Whether the Auto Decryption block has operation permissions depends on:

- In SPI Boot mode

If the first bit or the third bit in the eFuse parameter [EFUSE\\_SPI\\_BOOT\\_ENCRYPT\\_DECRYPT\\_CNT](#) (3 bits) is set to 1, then the Auto Decryption block has operation permissions. Otherwise, it is not operational.

- In Download Boot mode

If bit [SYSTEM\\_ENABLE\\_DOWNLOAD\\_GOCB\\_DECRYPT](#) in register [SYSTEM\\_EXTERNAL\\_DEVICE\\_ENCRYPT\\_DECRYPT\\_CONTROL\\_REG](#) is 1, the Auto Decryption block has operation permissions. Otherwise, it is not operational.

**Note:**

- When the Auto Decryption block has operation permissions, it will automatically decrypt the ciphertext if the CPU reads instructions/data from the external memory via cache to retrieve the instructions/data. The entire decryption process does not need software participation and is transparent to the cache. Users can by no means obtain the decryption *Key* during the process.
- When the Auto Decryption block has no operation permissions, it does not have any effect on the contents stored in the external memory, no matter if they are encrypted or not. Therefore, what the CPU reads via cache is the original information stored in the external memory.

## 17.5 Software Process

When the Manual Encryption block operates, software needs to be involved in the process. The steps are as follows:

1. Configure XTS\_AES:

- Set register [XTS\\_AES\\_PHYSICAL\\_ADDRESS\\_REG](#) to *base\_addr*.
- Set register [XTS\\_AES\\_LINESIZE\\_REG](#) to  $\frac{size}{32}$ .

For definitions of *base\_addr* and *size*, please refer to Section [17.4.3](#).

2. Write plaintext data to the registers block [XTS\\_AES\\_PLAIN\\_n\\_REG](#) ( $n$ : 0 ~ 7). For detailed information, please refer to Section [17.4.4](#).  
Please write data to registers according to your actual needs, and the unused ones could be set to arbitrary values.
3. Wait for Manual Encryption block to be idle. Poll register [XTS\\_AES\\_STATE\\_REG](#) until it reads 0 that indicates the Manual Encryption block is idle.
4. Trigger manual encryption by writing 1 to register [XTS\\_AES\\_TRIGGER\\_REG](#).
5. Wait for the encryption process completion. Poll register [XTS\\_AES\\_STATE\\_REG](#) until it reads 2.  
*Step 1 to 5 are the steps of encrypting plaintext instructions with the Manual Encryption block using the Key.*
6. Write 1 to register [XTS\\_AES\\_RELEASE\\_REG](#) to grant SPI1 the access to the encrypted ciphertext. Then, poll register [XTS\\_AES\\_STATE\\_REG](#) until it reads 3.
7. Call SPI1 to write the ciphertext in the external flash (see Chapter [20 SPI Controller \(SPI\)](#)).
8. Write 1 to register [XTS\\_AES\\_DESTROY\\_REG](#) to destroy the ciphertext. After this, the value of register [XTS\\_AES\\_STATE\\_REG](#) will become 0.

Repeat above steps according to the amount of plaintext instructions/data that need to be encrypted.

## 17.6 Register Summary

The addresses in this section are relative to External Memory Encryption and Decryption base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

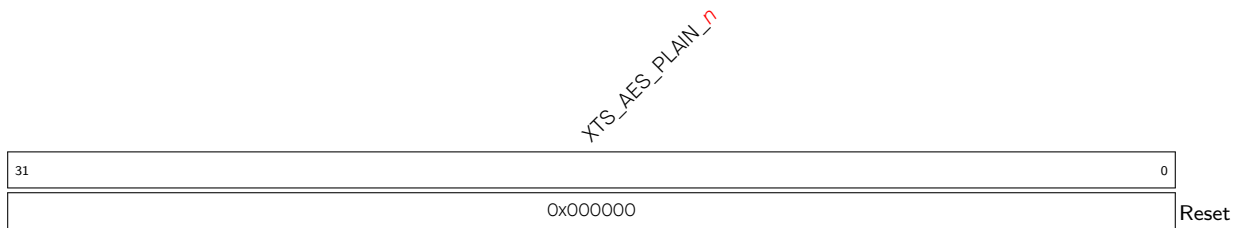
Name	Description	Address	Access
<b>Plaintext Register Heap</b>			
<a href="#">XTS_AES_PLAIN_0_REG</a>	Plaintext register 0	0x0000	R/W
<a href="#">XTS_AES_PLAIN_1_REG</a>	Plaintext register 1	0x0004	R/W
<a href="#">XTS_AES_PLAIN_2_REG</a>	Plaintext register 2	0x0008	R/W
<a href="#">XTS_AES_PLAIN_3_REG</a>	Plaintext register 3	0x000C	R/W
<a href="#">XTS_AES_PLAIN_4_REG</a>	Plaintext register 4	0x0010	R/W
<a href="#">XTS_AES_PLAIN_5_REG</a>	Plaintext register 5	0x0014	R/W
<a href="#">XTS_AES_PLAIN_6_REG</a>	Plaintext register 6	0x0018	R/W
<a href="#">XTS_AES_PLAIN_7_REG</a>	Plaintext register 7	0x001C	R/W
<b>Configuration Registers</b>			
<a href="#">XTS_AES_LINESIZE_REG</a>	Configures the size of target memory space	0x0040	R/W
<a href="#">XTS_AES_DESTINATION_REG</a>	Configures the type of the external memory	0x0044	R/W
<a href="#">XTS_AES_PHYSICAL_ADDRESS_REG</a>	Physical address	0x0048	R/W
<b>Control/Status Registers</b>			
<a href="#">XTS_AES_TRIGGER_REG</a>	Activates AES algorithm	0x004C	WO
<a href="#">XTS_AES_RELEASE_REG</a>	Release control	0x0050	WO
<a href="#">XTS_AES_DESTROY_REG</a>	Destroy control	0x0054	WO
<a href="#">XTS_AES_STATE_REG</a>	Status register	0x0058	RO
<b>Version Register</b>			
<a href="#">XTS_AES_DATE_REG</a>	Version control register	0x005C	RO



## 17.7 Registers

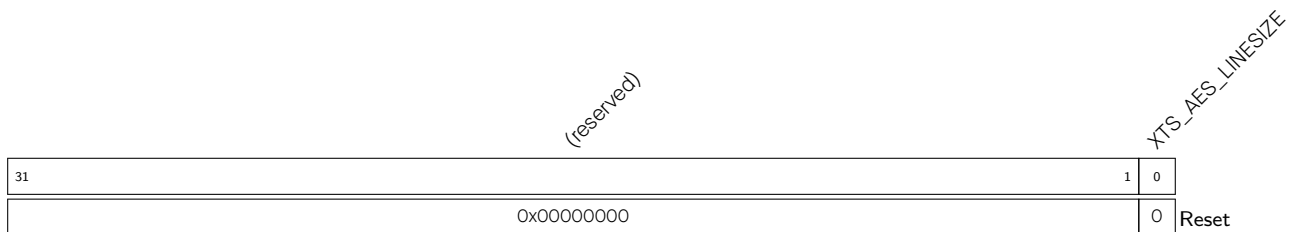
The addresses in this section are relative to External Memory Encryption and Decryption base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 17.1. XTS\_AES\_PLAIN\_*n*\_REG (*n*: 0-7) (0x0000+4\**n*)



XTS\_AES\_PLAIN\_*n* Stores the *n*th 32-bit piece of plaintext. (R/W)

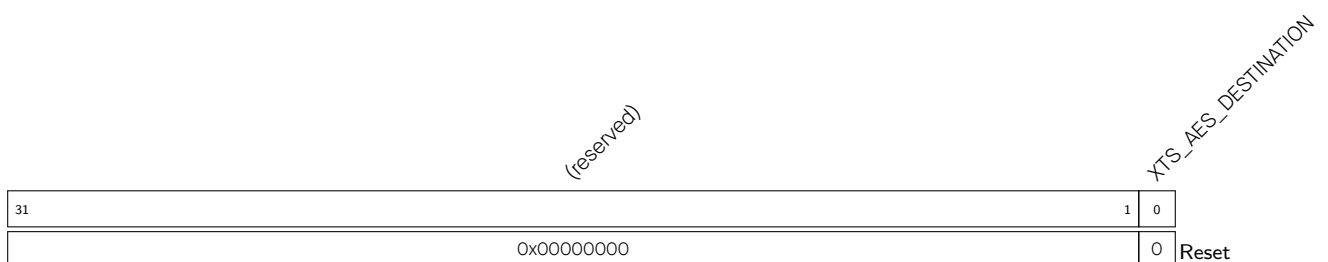
Register 17.2. XTS\_AES\_LINESIZE\_REG (0x0040)



XTS\_AES\_LINESIZE Configures the data size of one encryption operation. (R/W)

- 0: 16 bytes;
- 1: 32 bytes.

Register 17.3. XTS\_AES\_DESTINATION\_REG (0x0044)



XTS\_AES\_DESTINATION Configures the type of the external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Errors may occur if users write 1. (R/W)

- 0: flash;
- 1: external RAM.

#### Register 17.4. XTS\_AES\_PHYSICAL\_ADDRESS\_REG (0x0048)

31	30	29		0
0x0	0x00000000			Reset

**XTS\_AES\_PHYSICAL\_ADDRESS** Physical address. (Note that its value should be within the range of 0x0000\_0000 and 0x00FF\_FFFF). (R/W)

### Register 17.5. XTS\_AES\_TRIGGER\_REG (0x004C)

		(reserved)			XTS_AES_TRIGGER
31	1	0			
		0x00000000	x	Reset	

**XTS\_AES\_TRIGGER** Write 1 to trigger manual encryption. (WO)

### Register 17.6. XTS\_AES\_RELEASE\_REG (0x0050)

	(reserved)	XTS_AES_RELEASE
31	1	0
	0x00000000	x Reset

**XTS\_AES\_RELEASE** Write 1 to grant SPI1 access to the encrypted result. (WO)

**Register 17.7. XTS\_AES\_DESTROY\_REG (0x0054)**

(reserved)		XTS_AES_DESTROY	
31	1	0	
0x00000000		X	Reset

**XTS\_AES\_DESTROY** Write 1 to destroy encrypted result. (WO)

**Register 17.8. XTS\_AES\_STATE\_REG (0x0058)**

(reserved)		XTS_AES_STATE	
31	2	1	0
0x00000000		0x0	Reset

**XTS\_AES\_STATE** Indicates the status of the Manual Encryption block. (RO)

- 0x0 (XTS\_AES\_IDLE): idle;
- 0x1 (XTS\_AES\_BUSY): busy with encryption;
- 0x2 (XTS\_AES\_DONE): encryption is completed, but the encrypted result is not accessible to SPI;
- 0x3 (XTS\_AES\_RELEASE): encrypted result is accessible to SPI.

**Register 17.9. XTS\_AES\_DATE\_REG (0x005C)**

(reserved)		XTS_AES_DATE	
31	30	29	0
0	0	0x20200623	
		Reset	

**XTS\_AES\_DATE** Version control register. (R/W)

## Chapter 18

# Random Number Generator (RNG)

## 18.1 Introduction

The ESP8684 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

## 18.2 Features

The random number generator in ESP8684 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 bit seeds.
- Internal fast RC oscillator clock RC\_FAST\_CLK (typically about 17.5 MHz, and adjustable) is an **asynchronous clock** source and it increases the RNG entropy by introducing circuit metastability.

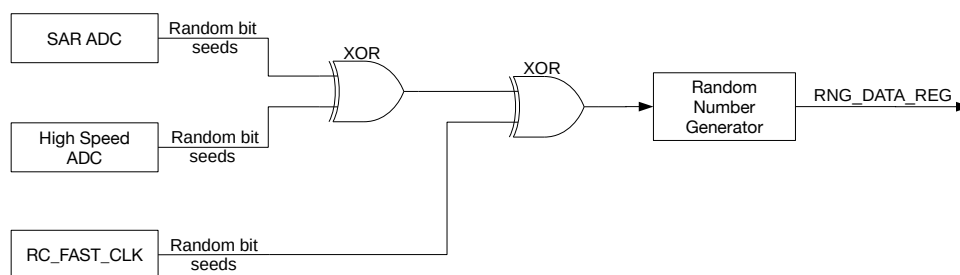


Figure 18.3-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 1-bit entropy in one ADC sampling cycle. Considering the maximum ADC sample rate is 128 KHz, it is thus advisable to read the [RNG\\_DATA\\_REG](#) register also at a maximum rate of 128 kHz.

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.

## 18.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC or high-speed ADC<sup>1</sup> 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 [23 On-Chip Sensor and Analog Signal Processing](#).
- High-speed ADC is enabled automatically when the wireless module is enabled.
- RC\_FAST\_CLK<sup>2</sup> is always enabled when the chip is on. Therefore, no need to enable this clock specifically.

### Note:

1. Note that, when the wireless 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.
2. RC\_FAST\_CLK increases the RNG entropy. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

When using the random number generator, read the [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.

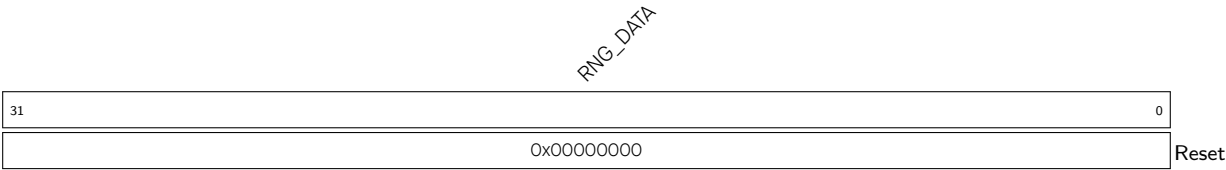
## 18.5 Register Summary

The abbreviations given in Column **Access** are explained in Section [Access Types for Registers](#).

Name	Description	Address	Access
<a href="#">RNG_DATA_REG</a>	Random number data	0x6002_60B0	RO

# 18.6 Register

Register 18.1. RNG\_DATA\_REG (0x6002\_60B0)



**RNG\_DATA** Random number source. (RO)

## Part V

# Connectivity Interface

This part addresses the connectivity aspects of the system, describing components related to various communication interfaces like I2C, SPI, UART, and LEDPWM.

## Chapter 19

### UART Controller (UART)

#### 19.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full-duplex mode. ESP8684 has two UART controllers compatible with various UART devices. They support Infrared Data Association (IrDA) and RS485 transmission.

Each of the two UART controllers has a group of registers that function identically. In this chapter, the two UART controllers are referred to as UART $n$ , in which  $n$  denotes 0 or 1.

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to the data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit(s) and parity bit.

A UART data frame usually begins with one start bit, followed by data bits, one parity bit (optional) and one or more stop bits. UART controllers on ESP8684 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control.

#### 19.2 Features

Each UART controller has the following features:

- Full-duplex asynchronous communication
- Configurable baud rate, up to 2.5 Mbaud
- Automatic baud rate detection of input signals
- Data frame format:
  - a START bit
  - data bits, ranging from 5 ~ 8
  - a parity bit
  - stop bits, whose length can be 1, 1.5, or 2 bits
- Special character AT\_CMD detection
- Supported protocols: RS485, IrDA
- UART as wake-up source



- Software and hardware flow control
- Three clock sources that can be divided:
  - 40 MHz PLL\_F40M\_CLK
  - internal fast RC oscillator RC\_FAST\_CLK
  - external crystal clock XTAL\_CLK
- 512 x 8-bit RAM shared by TX FIFOs and RX FIFOs of the two UART controllers

## 19.3 UART Architecture

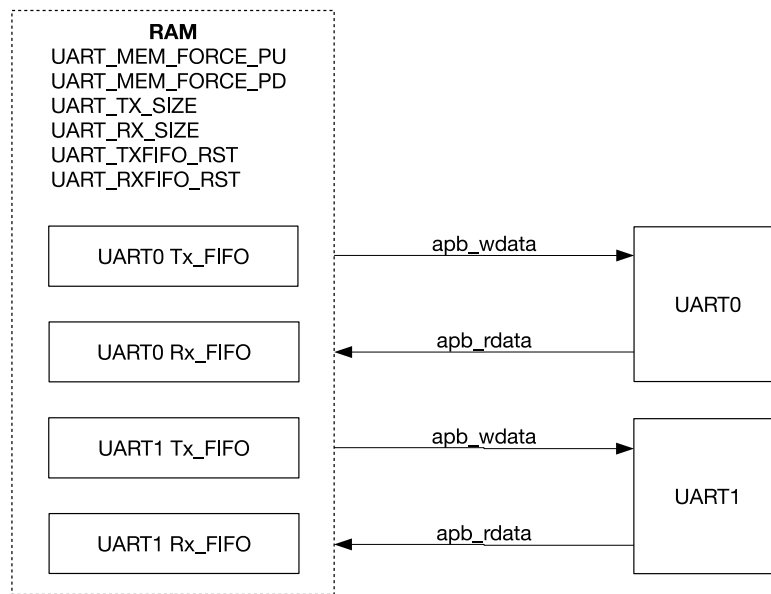


Figure 19.3-1. UART Architecture Overview

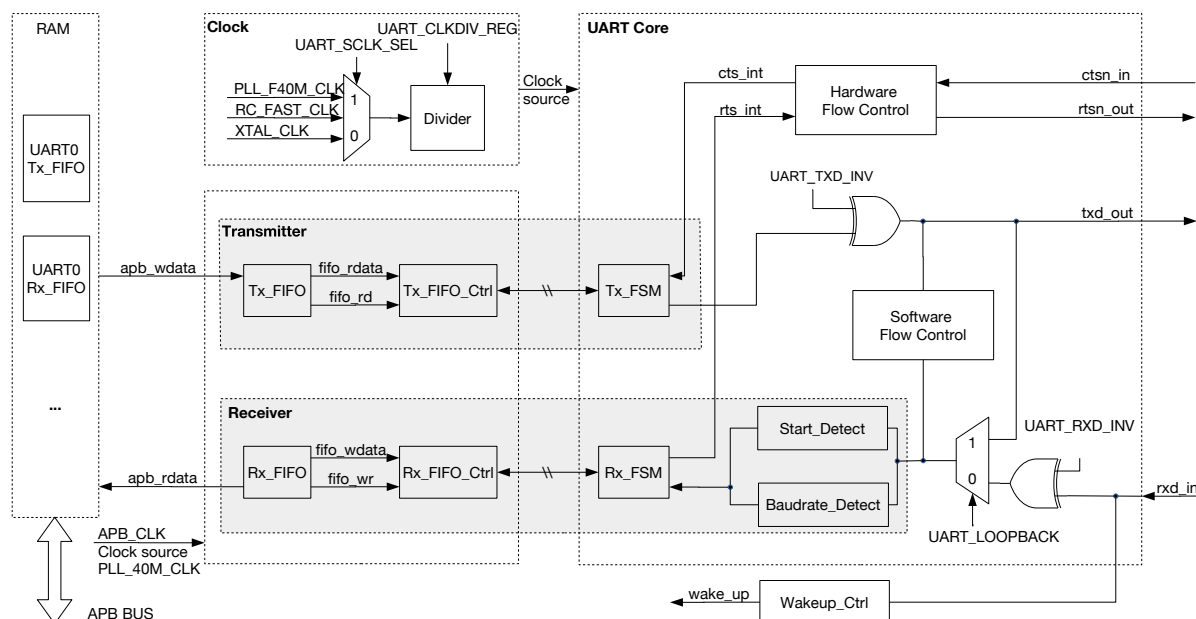


Figure 19.3-2. UART Architecture

Figure 19.3-2 shows the basic architecture of a UART controller. A UART controller works in two clock domains, namely APB\_CLK domain and Core Clock domain (the UART Core's clock domain).

APB\_CLK is derived from PLL\_F40M\_CLK.

The UART Core's clock is derived from the 40 MHz PLL\_F40M\_CLK, RC\_FAST\_CLK, or external crystal clock XTAL\_CLK (for details, please refer to Chapter 6 *Reset and Clock*), which are selected by configuring `UART_SCLK_SEL`. The selected clock source is divided by a divider to generate clock signals that drive the UART Core. The divisor is configured by `UART_CLKDIV_REG: UART_CLKDIV` for the integral part, and `UART_CLKDIV_FRAG` for the fractional part.

A UART controller is broken down into two parts according to functions: a transmitter and a receiver.

The transmitter contains a TX FIFO, which buffers data to be sent. Software can write data to Tx\_FIFO via the APB bus. Tx\_FIFO\_Ctrl controls writing and reading Tx\_FIFO. When Tx\_FIFO is not empty, Tx\_FSM reads data bits in the data frame via Tx\_FIFO\_Ctrl, and converts them into a bitstream. The levels of output signal txd\_out can be inverted by configuring the `UART_TXD_INV` field.

The receiver contains a RX FIFO, which buffers data to be processed. The levels of input signal rxd\_in can be inverted by configuring `UART_RXD_INV` field. Baudrate\_Detect measures the baud rate of input signal rxd\_in by detecting its minimum pulse width. Start\_Detect detects the start bit in a data frame. If the start bit is detected, Rx\_FSM stores data bits in the data frame into Rx\_FIFO by Rx\_FIFO\_Ctrl. Software can read data from Rx\_FIFO via the APB bus.

HW\_Flow\_Ctrl controls rxd\_in and txd\_out data flows by standard UART RTS and CTS flow control signals

(rtsn\_out and ctsn\_in). SW\_Flow\_Ctrl controls data flows by automatically adding special characters to outgoing data and detecting special characters in incoming data. When a UART controller is in Light-sleep mode (see Chapter 9 *Low-power Management (RTC\_CNTL)* for more details), Wakeup\_Ctrl counts up rising edges of rxd\_in. When the number is equal to or greater than (UART\_ACTIVE\_THRESHOLD + 3), a wake\_up signal is generated and sent to RTC, which then wakes up the ESP8684 chip.

## 19.4 Functional Description

### 19.4.1 Clock and Reset

Specific functional blocks of UART controllers are asynchronous. Their register configuration module, TX FIFO and RX FIFO are in APB\_CLK domain, while the UART Core that controls transmission and reception is in Core Clock domain. The three clock sources of the UART core, namely PLL\_F40M\_CLK, RC\_FAST\_CLK and external crystal clock XTAL\_CLK, are selected by configuring [UART\\_SCLK\\_SEL](#). The selected clock source is divided by a divider. This divider supports fractional frequency division: [UART\\_SCLK\\_DIV\\_NUM](#) field is the integral part, [UART\\_SCLK\\_DIV\\_B](#) field is the numerator of the fractional part, and [UART\\_SCLK\\_DIV\\_A](#) is the denominator of the fractional part. The divisor ranges from 1 to 256.

When the frequency of the UART Core's clock is higher than the frequency needed to generate the baud rate, the UART Core can be clocked at a lower frequency by the divider, in order to reduce power consumption. Usually, the UART Core's clock frequency is lower than the APB\_CLK's frequency, and can be divided by the largest divisor value when higher than the frequency needed to generate the baud rate. The frequency of the UART Core's clock can also be at most twice higher than the APB\_CLK. The clock for the UART transmitter and the UART receiver can be controlled independently. To enable the clock for the UART transmitter, [UART\\_TX\\_SCLK\\_EN](#) shall be set; to enable the clock for the UART receiver, [UART\\_RX\\_SCLK\\_EN](#) shall be set.

To ensure that the configured register values are synchronized from APB\_CLK domain to Core Clock domain, please follow procedures in Section [19.5](#).

To reset the whole UART, please:

- enable the clock for UART RAM by setting [SYSTEM\\_UART\\_MEM\\_CLK\\_EN](#) to 1;
- enable APB\_CLK for UART $n$  by setting [SYSTEM\\_UART \$n\$ \\_CLK\\_EN](#) to 1;
- clear [SYSTEM\\_UART \$n\$ \\_RST](#) to 0;
- write 1 to [UART\\_RST\\_CORE](#);
- write 1 to [SYSTEM\\_UART \$n\$ \\_RST](#);
- clear [SYSTEM\\_UART \$n\$ \\_RST](#) to 0;
- clear [UART\\_RST\\_CORE](#) to 0.

**Note:**

It is not recommended to reset the APB clock domain module ([SYSTEM\\_UART \$n\$ \\_RST](#)) or UART Core ([UART\\_RST\\_CORE](#)) only.

## 19.4.2 UART RAM

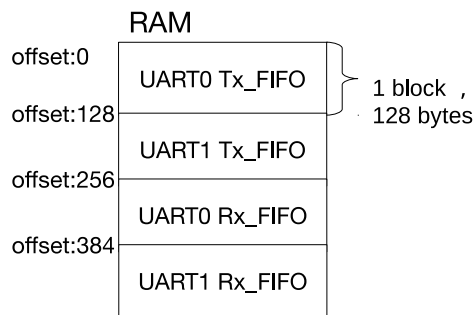


Figure 19.4-1. UART Controllers Sharing RAM

The two UART controllers on ESP8684 share  $512 \times 8$  bits of FIFO RAM. As Figure 19.4-1 illustrates, RAM is divided into 4 blocks, each having  $128 \times 8$  bits. Figure 19.4-1 shows how many RAM blocks are allocated to TX FIFOs and RX FIFOs of the two UART controllers by default. UART $n$  Tx\_FIFO can be expanded by configuring [UART\\_TX\\_SIZE](#), while UART $n$  Rx\_FIFO can be expanded by configuring [UART\\_RX\\_SIZE](#). Some limits are imposed:

- UART0 Tx\_FIFO can be increased up to 4 blocks (the whole RAM);
- UART1 Tx\_FIFO can be increased up to 3 blocks (from offset 128 to the end address);
- UART0 Rx\_FIFO can be increased up to 2 blocks (from offset 256 to the end address);
- UART1 Rx\_FIFO cannot be increased.

Please note that starting addresses of all FIFOs are fixed, so expanding one FIFO may take up the default space of other FIFOs. For example, by setting [UART\\_TX\\_SIZE](#) of UART0 to 2, the size of UART0 Tx\_FIFO is increased by 128 bytes (from offset 0 to offset 256). In this case, UART0 Tx\_FIFO takes up the default space for UART1 Tx\_FIFO, and UART1's transmitting function cannot be used as a result.

When neither of the two UART controllers is active, RAM could enter low-power mode by setting [UART\\_MEM\\_FORCE\\_PD](#).

UART0 Tx\_FIFO and UART1 Tx\_FIFO are reset by setting [UART\\_TXFIFO\\_RST](#). UART0 Rx\_FIFO and UART1 Rx\_FIFO are reset by setting [UART\\_RXFIFO\\_RST](#).

Data to be sent is written to TX FIFO via the APB bus, read automatically, and converted from a frame into a bitstream by hardware Tx\_FSM; data received is converted from a bitstream into a frame by hardware Rx\_FSM, written into RX FIFO, and then stored into RAM via the APB bus.

The empty signal threshold for Tx\_FIFO is configured by setting [UART\\_TXFIFO\\_EMPTY\\_THRHD](#). When data stored in Tx\_FIFO is less than [UART\\_TXFIFO\\_EMPTY\\_THRHD](#), a UART\_TXFIFO\_EMPTY\_INT interrupt is generated. The full signal threshold for Rx\_FIFO is configured by setting [UART\\_RXFIFO\\_FULL\\_THRHD](#). When data stored in Rx\_FIFO is greater than [UART\\_RXFIFO\\_FULL\\_THRHD](#), a UART\_RXFIFO\_FULL\_INT interrupt is generated. In addition, when Rx\_FIFO receives more data than its capacity, a UART\_RXFIFO\_OVF\_INT interrupt is generated.

UART $n$  can access FIFO via register [UART\\_FIFO\\_REG](#). Writing to [UART\\_RXFIFO\\_RD\\_BYTE](#) stores the data into the TX FIFO. As [UART\\_RXFIFO\\_RD\\_BYTE](#) is a read-only register field, the hardware does not actually perform a write operation on [UART\\_RXFIFO\\_RD\\_BYTE](#); instead, upon detecting a write request to this field's address, it

passes the corresponding write data to the TX FIFO via a separate bypass. Reading `UART_RXFIFO_RD_BYTE` retrieves the data from the RX FIFO.

### 19.4.3 Baud Rate Generation and Detection

#### 19.4.3.1 Baud Rate Generation

Before a UART controller sends or receives data, the baud rate should be configured by setting corresponding registers. The baud rate generator of a UART controller functions by dividing the input clock source. It can divide the clock source by a fractional amount. The divisor is configured by `UART_CLKDIV_REG`: `UART_CLKDIV` for the integral part, and `UART_CLKDIV_FRAG` for the fractional part. When using the 40 MHz input clock, the UART controller supports a maximum baud rate of 2.5 Mbaud.

The divisor of the baud rate divider is equal to

$$UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}$$

meaning that the final baud rate is equal to

$$\frac{INPUT\_FREQ}{UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}}$$

where `INPUT_FREQ` is the frequency of UART Core's source clock. For example, if `UART_CLKDIV` = 694 and `UART_CLKDIV_FRAG` = 7, then the divisor value is

$$694 + \frac{7}{16} = 694.4375$$

When `UART_CLKDIV_FRAG` is 0, the baud rate generator is an integer clock divider where an output pulse is generated every `UART_CLKDIV` input pulses.

When `UART_CLKDIV_FRAG` is not 0, the divider is fractional and the output baud rate clock pulses are not strictly uniform. As shown in Figure 19.4-2, for every 16 output pulses, the generator divides either  $(UART\_CLKDIV + 1)$  input pulses or `UART_CLKDIV` input pulses per output pulse. A total of `UART_CLKDIV_FRAG` output pulses are generated by dividing  $(UART\_CLKDIV + 1)$  input pulses, and the remaining  $(16 - UART\_CLKDIV\_FRAG)$  output pulses are generated by dividing `UART_CLKDIV` input pulses.

The output pulses are interleaved as shown in Figure 19.4-2 below, to make the output timing more uniform:

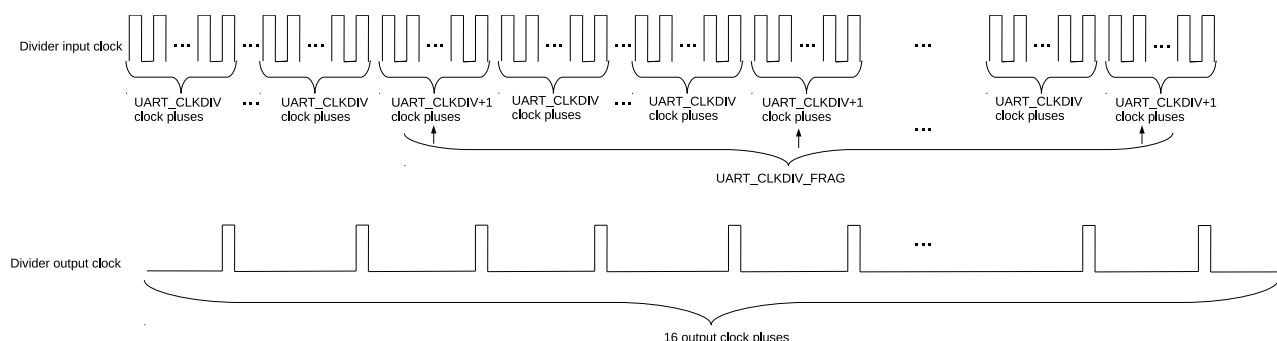


Figure 19.4-2. UART Controllers Division

To support IrDA (see Section 19.4.7 for details), the fractional clock divider for IrDA data transmission generates clock signals divided by  $16 \times UART\_CLKDIV\_REG$ . This divider works similarly as the one elaborated

above: it takes `UART_CLKDIV/16` as the integer value and the lowest four bits of `UART_CLKDIV` as the fractional value.

### 19.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting `UART_AUTOBAUD_EN`. The Baudrate\_Detect module shown in Figure 19.3-2 filters any noise whose pulse width is shorter than `UART_GLITCH_FILTER`.

Before communication starts, the transmitter could send random data to the receiver for baud rate detection. `UART_LOWPULSE_MIN_CNT` stores the minimum low pulse width, `UART_HIGHPULSE_MIN_CNT` stores the minimum high pulse width, `UART_POSEDGE_MIN_CNT` stores the minimum pulse width between two rising edges, and `UART_NEGEDGE_MIN_CNT` stores the minimum pulse width between two falling edges. These four fields are read by software to determine the transmitter's baud rate.

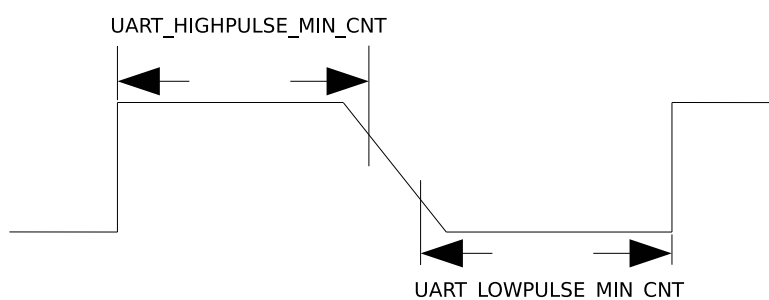


Figure 19.4-3. The Timing Diagram of Weak UART Signals Along Falling Edges

Baud rate  $B_{\text{uart}}$  can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or falling edges in a metastable state, which results in the inaccuracy of `UART_LOWPULSE_MIN_CNT` or `UART_HIGHPULSE_MIN_CNT`, use a weighted average of these two values to eliminate errors. In this case, the baud rate is calculated as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

where  $f_{\text{clk}}$  stands for clock frequency.

2. If UART signals are weak along falling edges as shown in Figure 19.4-3, which leads to an inaccurate average of `UART_LOWPULSE_MIN_CNT` and `UART_HIGHPULSE_MIN_CNT`, use `UART_POSEDGE_MIN_CNT` to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

3. If UART signals are weak along rising edges, use `UART_NEGEDGE_MIN_CNT` to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

### 19.4.4 UART Data Frame

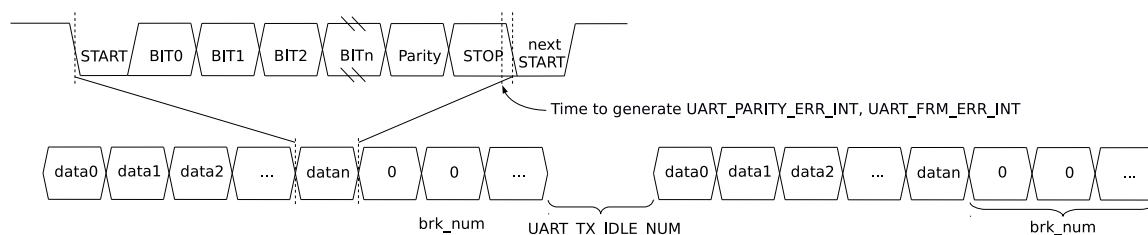


Figure 19.4-4. Structure of UART Data Frame

Figure 19.4-4 shows the basic structure of a data frame. A frame starts with one START bit, and ends with STOP bits which can be 1, 1.5, or 2 bits long, configured by `UART_STOP_BIT_NUM` (in RS485 mode turnaround delay may be added. See details in Section 19.4.6.2). The START bit is logical low, whereas STOP bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by `UART_BIT_NUM`. When `UART_PARITY_EN` is set, a parity bit is added after data bits. `UART_PARITY` is used to choose even parity or odd parity. When the receiver detects a parity bit error in the data received, a `UART_PARITY_ERR_INT` interrupt is generated, and the data received is still stored into RX FIFO. When the receiver detects a framing error (i.e. the sampled stop bit is not 1), a `UART_FRM_ERR_INT` interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx\_FIFO has been sent, a `UART_TX_DONE_INT` interrupt is generated. After this, if the `UART_TXD_BRK` bit is set, then the transmitter will enter the Break condition and send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by `UART_TX_BRK_NUM`. Once the transmitter has sent all NULL characters, a `UART_TX_BRK_DONE_INT` interrupt is generated. The minimum interval between data frames can be configured using `UART_TX_IDLE_NUM`. If the transmitter stays idle for `UART_TX_IDLE_NUM` or more time, a `UART_TX_BRK_IDLE_DONE_INT` interrupt is generated.

The receiver can also detect the Break conditions when the RX data line 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.

### 19.4.5 AT\_CMD Character Structure

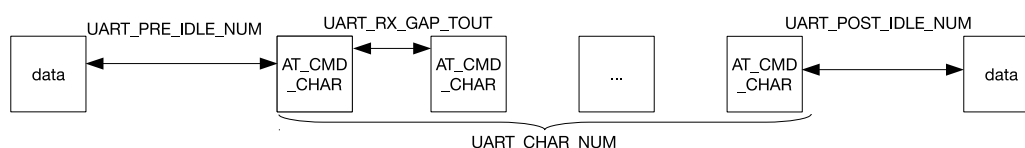


Figure 19.4-5. AT\_CMD Character Structure

Figure 19.4-5 is the structure of a special character AT\_CMD. If the receiver constantly receives AT\_CMD\_CHAR and the following conditions are met, a UART\_AT\_CMD\_CHAR\_DET\_INT interrupt is generated.

- The interval between the first AT\_CMD\_CHAR and the last non-AT\_CMD\_CHAR character is at least `UART_PRE_IDLE_NUM` cycles.
- The interval between two AT\_CMD\_CHAR characters is less than `UART_RX_GAP_TOUT` cycles.
- The number of AT\_CMD\_CHAR characters is equal to or greater than `UART_CHAR_NUM`.
- The interval between the last AT\_CMD\_CHAR character and next non-AT\_CMD\_CHAR character is at least `UART_POST_IDLE_NUM` cycles.

## 19.4.6 RS485

The two UART controllers support RS485 protocol. This protocol uses differential signals to transmit data, so it can communicate over longer distances at higher bit rates than RS232. RS485 has two-wire half-duplex mode and four-wire full-duplex mode. UART controllers support two-wire half-duplex transmission and bus snooping. In a two-wire RS485 multidrop network, there can be 32 slaves at most.

### 19.4.6.1 Driver Control

As shown in Figure 19.4-6, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion. An RS485 transceiver contains a driver (D) and a receiver (R). When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver (D). When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

The UART receiver converts differential signals to single-ended signals via the receiver (R). RE is the enable control signal for the receiver. When RE is 0, the receiver is enabled; when RE is 1, the receiver is disabled. If RE is configured as 0, the UART controller is allowed to snoop data on the bus, including the data sent by itself.

DE can be controlled by either software or hardware. To reduce the cost of software, in our design, DE is controlled by hardware (can still be controlled by software). As shown in Figure 19.4-6, DE is connected to `dtrn_out` of UART (please refer to Section 19.4.9.1 for more details).

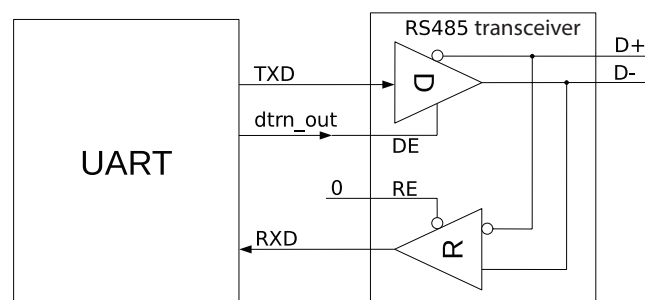


Figure 19.4-6. Driver Control Diagram in RS485 Mode

### 19.4.6.2 Turnaround Delay

By default, the two UART controllers work in receiver mode. When a UART controller is switched from transmitter mode to receiver mode, the RS485 protocol requires a turnaround delay of one cycle after the stop bit for stable data transmission. The UART transmitter supports adding a turnaround delay of one cycle before



the start bit or after the stop bit. When [UART\\_DLO\\_EN](#) is set, a turnaround delay of one cycle is added before the start bit; when [UART\\_DL1\\_EN](#) is set, a turnaround delay of one cycle is added after the stop bit.

### 19.4.6.3 Bus Snooping

In a two-wire multidrop network, UART controllers support bus snooping if RE of the external RS485 transceiver is 0. By default, a UART controller is not allowed to transmit and receive data simultaneously. If [UART\\_RS485TX\\_RX\\_EN](#) is set and the external RS485 transceiver is configured as in Figure 19.4-6, a UART controller may receive data in transmitter mode and snoop the bus. If [UART\\_RS485RXBY\\_TX\\_EN](#) is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop the data sent by themselves. In transmitter mode, when a UART controller detects a collision between the data sent and the data received, a [UART\\_RS485\\_CLASH\\_INT](#) is generated; when a UART controller detects a framing error, a [UART\\_RS485\\_FRM\\_ERR\\_INT](#) interrupt is generated; when a UART controller detects a polarity error, a [UART\\_RS485\\_PARITY\\_ERR\\_INT](#) is generated.

### 19.4.7 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol, and the link management protocol. The two UART controllers implement IrDA's physical layer. In IrDA encoding, a UART controller supports data rates up to 115.2 kbit/s (SIR, or serial infrared mode). As shown in Figure 19.4-7, the IrDA encoder converts a NRZ (non-return to zero code) signal to a RZI (return to zero inverted code) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits high level to indicate logic "0", and low levels to indicate logic "1". The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the encoder output polarity is the opposite of the decoder input polarity. If a low pulse is detected, it indicates that a start bit has been received.

When IrDA function is enabled, one bit is divided into 16 clock cycles. If the bit to be sent is zero, then the 9th, 10th and 11th clock cycle are high.

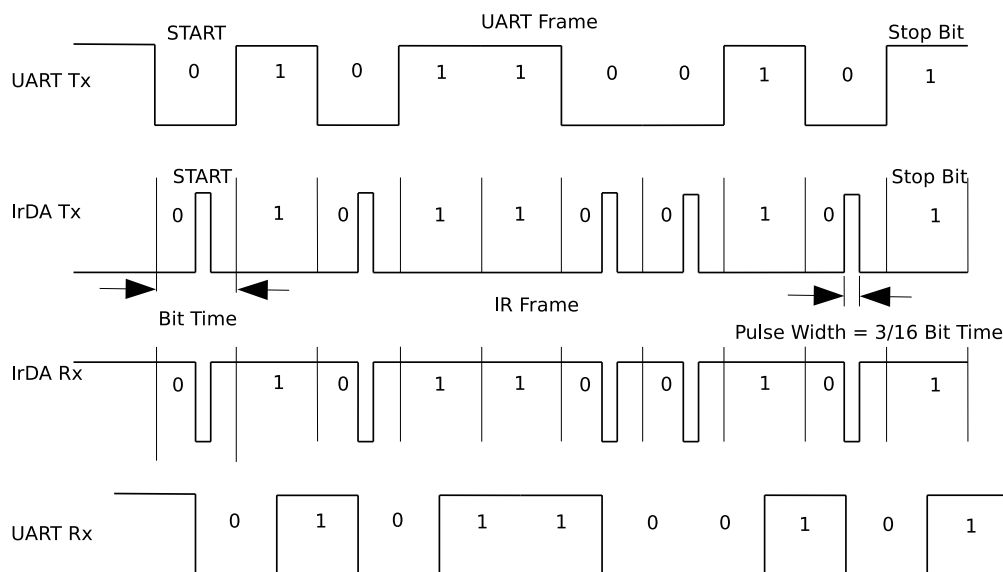


Figure 19.4-7. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in

Figure 19.4-8, IrDA function is enabled by setting `UART_IRDA_EN`. When `UART_IRDA_TX_EN` is set (high), the IrDA transceiver is enabled to send data and not allowed to receive data; when `UART_IRDA_TX_EN` is reset (low), the IrDA transceiver is enabled to receive data and not allowed to send data.

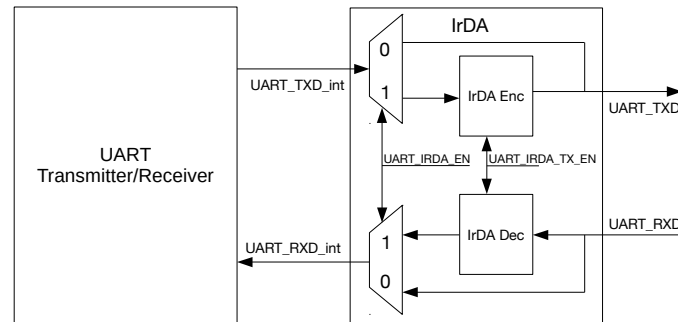


Figure 19.4-8. IrDA Encoding and Decoding Diagram

### 19.4.8 Wake-up

UART0 and UART1 can be set as wake-up source. When a UART controller is in Light-sleep mode, Wakeup\_Ctrl counts up the rising edges of rxd\_in. When the number of rising edges is equal to or greater than (`UART_ACTIVE_THRESHOLD + 3`), a wake\_up signal is generated and sent to RTC, which then wakes up ESP8684.

After the chip is woken up by UART, it is necessary to clear the wake\_up signal by transmitting data to UART in Active mode or resetting the whole UART, otherwise the number of rising edges required for the next wakeup will be reduced.

### 19.4.9 Flow Control

UART controllers have two ways to control data flow, namely hardware flow control and software flow control. Hardware flow control is achieved using output signal rtsn\_out and input signal dsrn\_in. Software flow control is achieved by inserting special characters in the data flow sent and detecting special characters in the data flow received.

### 19.4.9.1 Hardware Flow Control

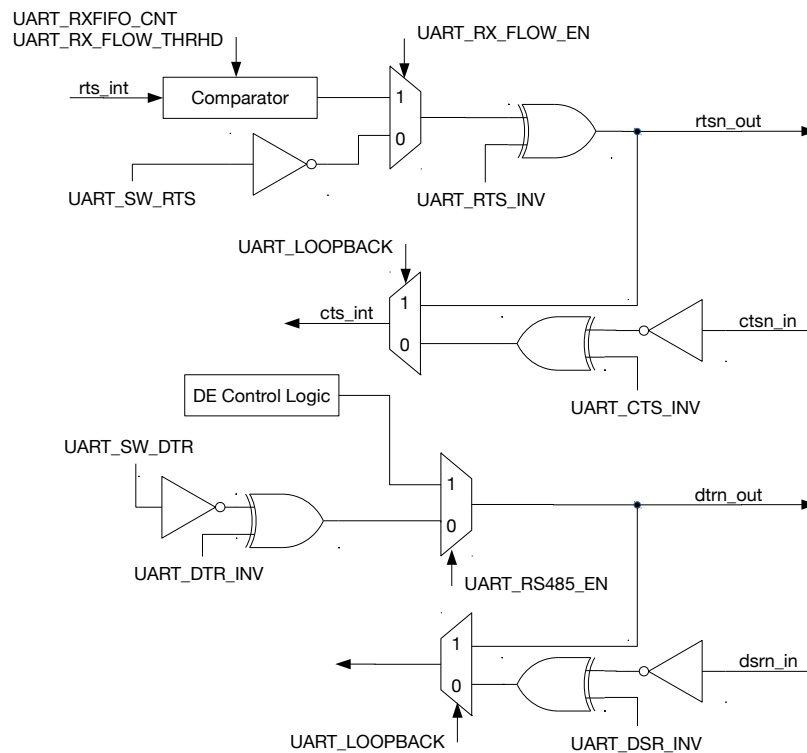
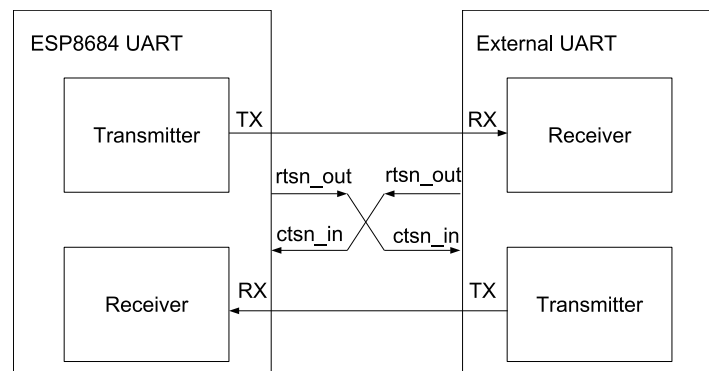


Figure 19.4-9. Hardware Flow Control Diagram

Figure 19.4-9 shows the hardware flow control of a UART controller. Hardware flow control uses output signal `rtsn_out` and input signal `dsrn_in`. Figure 19.4-10 illustrates how these signals are connected between UART on ESP8684 (hereinafter referred to as IUO) and the external UART (hereinafter referred to as EUO).

When `rtsn_out` of IUO is low, EUO is allowed to send data; when `rtsn_out` of IUO is high, EUO is notified to stop sending data until `rtsn_out` of IUO returns to low. The output signal `rtsn_out` can be controlled in two ways.

- Software control: Enter this mode by clearing `UART_RX_FLOW_EN` to 0. In this mode, the level of `rtsn_out` is changed by configuring `UART_SW_RTS`.
- Hardware control: Enter this mode by setting `UART_RX_FLOW_EN` to 1. In this mode, `rtsn_out` is pulled high when data in Rx\_FIFO exceeds `UART_RX_FLOW_THRHD`.



**Figure 19.4-10. Connection between Hardware Flow Control Signals**

When `ctsn_in` of IUO is low, IUO is allowed to send data; when `ctsn_in` is high, IUO is not allowed to send data. When IUO detects an edge change of `ctsn_in`, a `UART_CTS_CHG_INT` interrupt is generated.

If `dtrn_out` of IUO is high, it indicates that IUO is ready to transmit data. `dtrn_out` is generated by configuring the `UART_SW_DTR` field. When the IUO transmitter detects an edge change of `dsrn_in`, a `UART_DSR_CHG_INT` interrupt is generated. After this interrupt is detected, software can obtain the level of input signal `dsrn_in` by reading `UART_DSRN`. If `dsrn_in` is high, it indicates that EUO is ready to transmit data.

In a two-wire RS485 multidrop network enabled by setting `UART_RS485_EN`, `dtrn_out` is generated by hardware (i.e. DE control logic in Figure 19.4-10) and used for transmit/receive turnaround. When data transmission starts, `dtrn_out` is pulled high and the external driver is enabled; when data transmission completes, `dtrn_out` is pulled low and the external driver is disabled. Please note that when there is a turnaround delay of one cycle added after the stop bit, `dtrn_out` is pulled low after the delay.

UART loopback test is enabled by setting `UART_LOOPBACK`. In the test, UART output signal `txd_out` is connected to its input signal `rx_d_in`, `rtsn_out` is connected to `ctsn_in`, and `dtrn_out` is connected to `dsrn_out`. If the data sent matches the data received, it indicates that UART controllers are working properly.

### 19.4.9.2 Software Flow Control

Instead of `ctsn_in` and `rtsn_out` signals, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting `UART_SW_FLOW_CON_EN` to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in the data flow received, and generate a `UART_SW_XOFF_INT` or a `UART_SW_XON_INT` interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting `UART_FORCE_XOFF`, or to start sending data by setting `UART_FORCE_XON`.

Software determines whether to insert flow control characters according to the remaining room in RX FIFO. When `UART_SEND_XOFF` is set, the transmitter sends an XOFF character configured by `UART_XOFF_CHAR` after the current byte in transmission; when `UART_SEND_XON` is set, the transmitter sends an XON character configured by `UART_XON_CHAR` after the current byte in transmission. If the RX FIFO of a UART controller stores more data than `UART_XOFF_THRESHOLD`, `UART_SEND_XOFF` is set by hardware. As a result, the transmitter sends an XOFF character configured by `UART_XOFF_CHAR` after the current byte in transmission. If the RX FIFO of a UART controller stores less data than `UART_XON_THRESHOLD`, `UART_SEND_XON` is set by

hardware. As a result, the transmitter sends an XON character configured by [UART\\_XON\\_CHAR](#) after the current byte in transmission.

### 19.4.10 UART Interrupts

- `UART_AT_CMD_CHAR_DET_INT`: Triggered when the receiver detects an `AT_CMD` character.
- `UART_RS485_CLASH_INT`: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- `UART_RS485_FRM_ERR_INT`: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- `UART_RS485_PARITY_ERR_INT`: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- `UART_TX_DONE_INT`: Triggered when all data in the transmitter's TX FIFO has been sent.
- `UART_TX_BRK_IDLE_DONE_INT`: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- `UART_TX_BRK_DONE_INT`: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- `UART_GLITCH_DET_INT`: Triggered when the receiver detects a glitch in the middle of the start bit.
- `UART_SW_XOFF_INT`: Triggered when [UART\\_SW\\_FLOW\\_CON\\_EN](#) is set and the receiver receives a XOFF character.
- `UART_SW_XON_INT`: Triggered when [UART\\_SW\\_FLOW\\_CON\\_EN](#) is set and the receiver receives a XON character.
- `UART_RXFIFO_TOUT_INT`: Triggered when the receiver takes more time than [UART\\_RX\\_TOUT\\_THRHD](#) to receive one byte.
- `UART_BRK_DET_INT`: Triggered when the receiver detects a NULL character (i.e. logic 0 for one NULL character transmission) after stop bits.
- `UART_CTS_CHG_INT`: Triggered when the receiver detects an edge change of CTSn signals.
- `UART_DSR_CHG_INT`: Triggered when the receiver detects an edge change of DSRn signals.
- `UART_RXFIFO_OVF_INT`: Triggered when the receiver receives more data than the capacity of RX FIFO.
- `UART_FRM_ERR_INT`: Triggered when the receiver detects a framing error.
- `UART_PARITY_ERR_INT`: Triggered when the receiver detects a parity error.
- `UART_TXFIFO_EMPTY_INT`: Triggered when TX FIFO stores less data than what [UART\\_TXFIFO\\_EMPTY\\_THRHD](#) specifies.
- `UART_RXFIFO_FULL_INT`: Triggered when the receiver receives more data than what [UART\\_RXFIFO\\_FULL\\_THRHD](#) specifies.
- `UART_WAKEUP_INT`: Triggered when UART is woken up.

## 19.5 Programming Procedures

### 19.5.1 Register Type

All UART registers are in APB\_CLK domain. According to whether clock domain crossing and synchronization are required, UART registers that can be configured by software are classified into three types, namely immediate registers, synchronous registers, and static registers. Immediate registers are read in APB\_CLK domain, and take effect after configured via the APB bus. Synchronous registers are read in Core Clock domain, and take effect after synchronization. Static registers are also read in Core Clock domain, but would not change dynamically. Therefore, for static registers clock domain crossing is not required, and software can turn on and off the clock for the UART transmitter or receiver to ensure that the configuration sampled in Core Clock domain is correct.

#### 19.5.1.1 Synchronous Registers

Read in Core Clock domain, synchronous registers implement the clock domain crossing design to ensure that their values sampled in Core Clock domain are correct. These registers as listed in Table 19.5-1 are configured as follows:

- Enable register synchronization by clearing [UART\\_UPDATE\\_CTRL](#) to 0;
- Wait for [UART\\_REG\\_UPDATE](#) to become 0, which indicates the completion of last synchronization;
- Configure synchronous registers;
- Synchronize the configured values to Core Clock domain by writing 1 to [UART\\_REG\\_UPDATE](#).

Table 19.5-1. UART<sub>n</sub> Synchronous Registers

Register	Field
<a href="#">UART_CLKDIV_REG</a>	<a href="#">UART_CLKDIV_FRAG[3:0]</a>
	<a href="#">UART_CLKDIV[11:0]</a>
<a href="#">UART_CONFO_REG</a>	<a href="#">UART_AUTOBAUD_EN</a>
	<a href="#">UART_ERR_WR_MASK</a>
	<a href="#">UART_TXD_INV</a>
	<a href="#">UART_RXD_INV</a>
	<a href="#">UART_IRDA_EN</a>
	<a href="#">UART_TX_FLOW_EN</a>
	<a href="#">UART_LOOPBACK</a>
	<a href="#">UART_IRDA_RX_INV</a>
	<a href="#">UART_IRDA_TX_EN</a>
	<a href="#">UART_IRDA_WCTL</a>
	<a href="#">UART_IRDA_TX_EN</a>
	<a href="#">UART_IRDA_DPLX</a>
	<a href="#">UART_STOP_BIT_NUM</a>
	<a href="#">UART_BIT_NUM</a>
	<a href="#">UART_PARITY_EN</a>
	<a href="#">UART_PARITY</a>

Cont'd on next page

Table 19.5-1 – cont'd from previous page

Register	Field
UART_FLOW_CONF_REG	UART_SEND_XOFF
	UART_SEND_XON
	UART_FORCE_XOFF
	UART_FORCE_XON
	UART_XONOFF_DEL
	UART_SW_FLOW_CON_EN
UART_TXBRK_CONF_REG	UART_RS485_TX_DLY_NUM[3:0]
	UART_RS485_RX_DLY_NUM
	UART_RS485RXBY_TX_EN
	UART_RS485TX_RX_EN
	UART_DL1_EN
	UART_DLO_EN
	UART_RS485_EN

### 19.5.1.2 Static Registers

Static registers, though also read in Core Clock domain, would not change dynamically when UART controllers are at work, so they do not implement the clock domain crossing design. These registers must be configured when the UART transmitter or receiver is not at work. In this case, software can turn off the clock for the UART transmitter or receiver, so that static registers are not sampled in their metastable state. When software turns on the clock, the configured values are stable to be correctly sampled. Static registers as listed in Table 19.5-2 are configured as follows:

- Turn off the clock for the UART transmitter by clearing [UART\\_TX\\_SCLK\\_EN](#), or the clock for the UART receiver by clearing [UART\\_RX\\_SCLK\\_EN](#), depending on which one (transmitter or receiver) is not at work;
- Configure static registers;
- Turn on the clock for the UART transmitter by writing 1 to [UART\\_TX\\_SCLK\\_EN](#), or the clock for the UART receiver by writing 1 to [UART\\_RX\\_SCLK\\_EN](#).

Table 19.5-2. UART<sub>n</sub> Static Registers

Register	Field
UART_RX_FILT_REG	UART_GLITCH_FILT_EN
	UART_GLITCH_FILT[7:0]
UART_SLEEP_CONF_REG	UART_ACTIVE_THRESHOLD[9:0]
UART_SWFC_CONFO_REG	UART_XOFF_CHAR[7:0]
UART_SWFC_CONF1_REG	UART_XON_CHAR[7:0]
UART_IDLE_CONF_REG	UART_TX_IDLE_NUM[9:0]
UART_AT_CMD_PRECNT_REG	UART_PRE_IDLE_NUM[15:0]
UART_AT_CMD_POSTCNT_REG	UART_POST_IDLE_NUM[15:0]
UART_AT_CMD_GAPTOUT_REG	UART_RX_GAP_TOUT[15:0]
UART_AT_CMD_CHAR_REG	UART_CHAR_NUM[7:0]
	UART_AT_CMD_CHAR[7:0]

### 19.5.1.3 Immediate Registers

Except those listed in Table 19.5-1 and Table 19.5-2, registers that can be configured by software are immediate registers read in APB\_CLK domain, such as interrupt and FIFO configuration registers.

## 19.5.2 Detailed Steps

Figure 19.5-1 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.



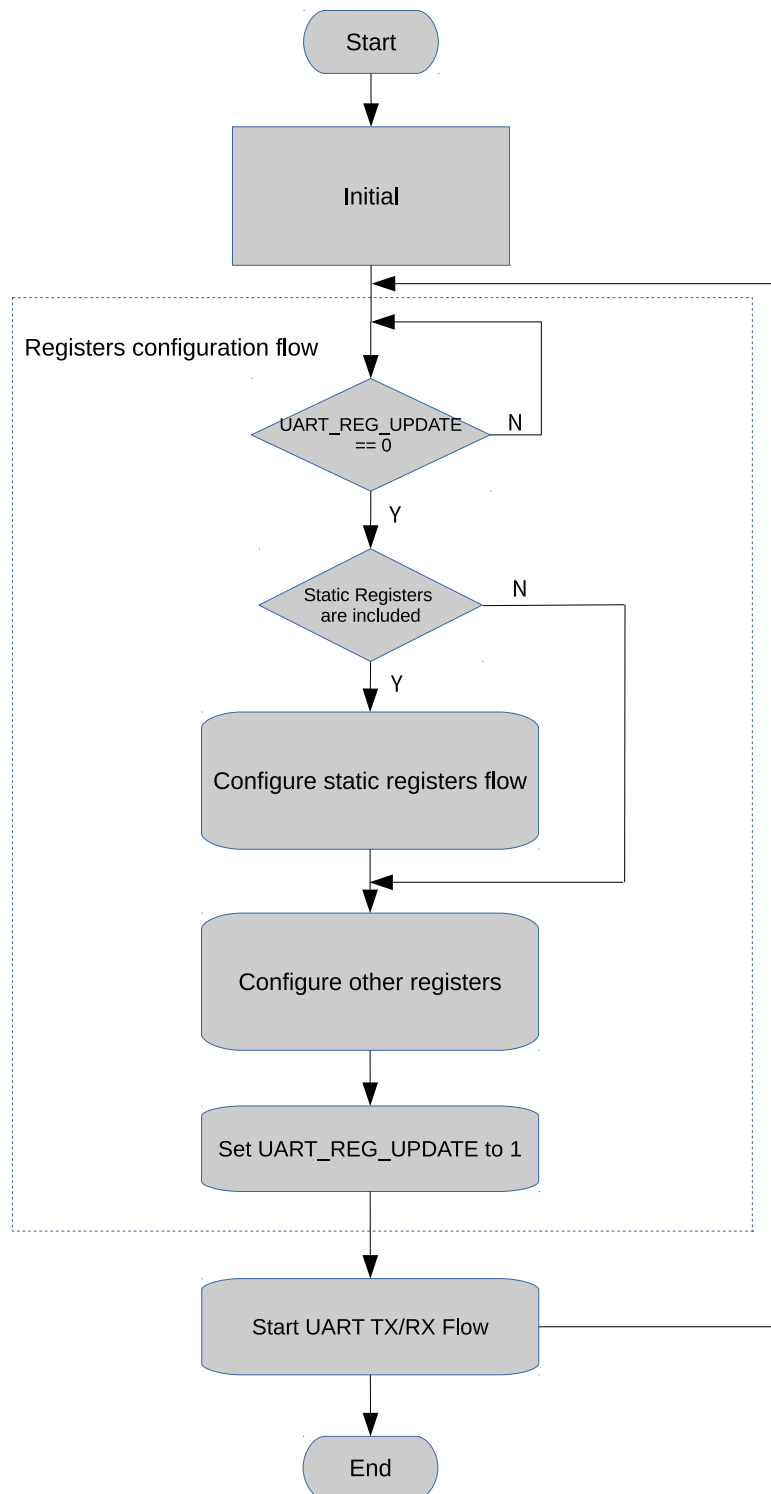


Figure 19.5-1. UART Programming Procedures

### 19.5.2.1 Initializing UART $n$

To initialize UART $n$ :

- enable the clock for UART RAM by setting `SYSTEM_UART_MEM_CLK_EN` to 1;
- enable APB\_CLK for UART $n$  by setting `SYSTEM_UART $n$ _CLK_EN` to 1;
- clear `SYSTEM_UART $n$ _RST`;

- write 1 to [UART\\_RST\\_CORE](#);
- write 1 to [SYSTEM\\_UART \$n\$ \\_RST](#);
- clear [SYSTEM\\_UART \$n\$ \\_RST](#);
- clear [UART\\_RST\\_CORE](#);
- enable register synchronization by clearing [UART\\_UPDATE\\_CTRL](#).

### 19.5.2.2 Configuring UART $n$ Communication

To configure UART $n$  communication:

- wait for [UART\\_REG\\_UPDATE](#) to become 0, which indicates the completion of the last synchronization;
- configure static registers (if any) following Section 19.5.1.2;
- select the clock source via [UART\\_SCLK\\_SEL](#);
- configure divisor of the divider via [UART\\_SCLK\\_DIV\\_NUM](#), [UART\\_SCLK\\_DIV\\_A](#), and [UART\\_SCLK\\_DIV\\_B](#);
- configure the baud rate for transmission via [UART\\_CLKDIV](#) and [UART\\_CLKDIV\\_FRAG](#);
- configure data length via [UART\\_BIT\\_NUM](#);
- configure odd or even parity check via [UART\\_PARITY\\_EN](#) and [UART\\_PARITY](#);
- optional steps depending on your application ...
- synchronize the configured values to the Core Clock domain by writing 1 to [UART\\_REG\\_UPDATE](#).

### 19.5.2.3 Enabling UART $n$

To enable UART $n$  transmitter:

- configure TX FIFO's empty threshold via [UART\\_TXFIFO\\_EMPTY\\_THRHD](#);
- disable [UART\\_TXFIFO\\_EMPTY\\_INT](#) interrupt by clearing [UART\\_TXFIFO\\_EMPTY\\_INT\\_ENA](#);
- write data to be sent to [UART\\_RXFIFO\\_RD\\_BYTE](#);
- clear [UART\\_TXFIFO\\_EMPTY\\_INT](#) interrupt by setting [UART\\_TXFIFO\\_EMPTY\\_INT\\_CLR](#);
- enable [UART\\_TXFIFO\\_EMPTY\\_INT](#) interrupt by setting [UART\\_TXFIFO\\_EMPTY\\_INT\\_ENA](#);
- detect [UART\\_TXFIFO\\_EMPTY\\_INT](#) and wait for the completion of data transmission.

To enable UART $n$  receiver:

- configure RX FIFO's full threshold via [UART\\_RXFIFO\\_FULL\\_THRHD](#);
- enable [UART\\_RXFIFO\\_FULL\\_INT](#) interrupt by setting [UART\\_RXFIFO\\_FULL\\_INT\\_ENA](#);
- detect [UART\\_TXFIFO\\_FULL\\_INT](#) and wait until the RX FIFO is full;
- read data from RX FIFO via [UART\\_RXFIFO\\_RD\\_BYTE](#), and obtain the number of bytes received in RX FIFO via [UART\\_RXFIFO\\_CNT](#).

## 19.6 Register Summary

The addresses in this section are relative to **UART Controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>FIFO Configuration</b>			
<a href="#">UART_FIFO_REG</a>	FIFO data register	0x0000	RO
<a href="#">UART_MEM_CONF_REG</a>	UART threshold and allocation configuration	0x0060	R/W
<b>UART Interrupt Register</b>			
<a href="#">UART_INT_RAW_REG</a>	Raw interrupt status	0x0004	R/WTC/SS
<a href="#">UART_INT_ST_REG</a>	Masked interrupt status	0x0008	RO
<a href="#">UART_INT_ENA_REG</a>	Interrupt enable bits	0x000C	R/W
<a href="#">UART_INT_CLR_REG</a>	Interrupt clear bits	0x0010	WT
<b>Configuration Register</b>			
<a href="#">UART_CLKDIV_REG</a>	Clock divider configuration	0x0014	R/W
<a href="#">UART_RX_FILT_REG</a>	RX filter configuration	0x0018	R/W
<a href="#">UART_CONFO_REG</a>	Configuration register 0	0x0020	R/W
<a href="#">UART_CONF1_REG</a>	Configuration register 1	0x0024	R/W
<a href="#">UART_FLOW_CONF_REG</a>	Software flow control configuration	0x0034	varies
<a href="#">UART_SLEEP_CONF_REG</a>	Sleep mode configuration	0x0038	R/W
<a href="#">UART_SWFC_CONFO_REG</a>	Software flow control character configuration register 0	0x003C	R/W
<a href="#">UART_SWFC_CONF1_REG</a>	Software flow control character configuration register 1	0x0040	R/W
<a href="#">UART_TXBRK_CONF_REG</a>	TX break character configuration	0x0044	R/W
<a href="#">UART_IDLE_CONF_REG</a>	Frame end idle time configuration	0x0048	R/W
<a href="#">UART_RS485_CONF_REG</a>	RS485 mode configuration	0x004C	R/W
<a href="#">UART_CLK_CONF_REG</a>	UART core clock configuration	0x0078	R/W
<b>Status Register</b>			
<a href="#">UART_STATUS_REG</a>	UART status register	0x001C	RO
<a href="#">UART_MEM_TX_STATUS_REG</a>	TX FIFO write and read offset address	0x0064	RO
<a href="#">UART_MEM_RX_STATUS_REG</a>	RX FIFO write and read offset address	0x0068	RO
<a href="#">UART_FSM_STATUS_REG</a>	UART transmitter and receiver status	0x006C	RO
<b>Autobaud Register</b>			
<a href="#">UART_LOWPULSE_REG</a>	Autobaud minimum low pulse duration register	0x0028	RO
<a href="#">UART_HIGHPULSE_REG</a>	Autobaud minimum high pulse duration register	0x002C	RO
<a href="#">UART_RXD_CNT_REG</a>	Autobaud edge change count register	0x0030	RO
<a href="#">UART_POSPULSE_REG</a>	Autobaud high pulse register	0x0070	RO
<a href="#">UART_NEGPULSE_REG</a>	Autobaud low pulse register	0x0074	RO
<b>AT Escape Sequence Selection Configuration</b>			

Name	Description	Address	Access
<a href="#">UART_AT_CMD_PRECNT_REG</a>	Pre-sequence timing configuration	0x0050	R/W
<a href="#">UART_AT_CMD_POSTCNT_REG</a>	Post-sequence timing configuration	0x0054	R/W
<a href="#">UART_AT_CMD_GAPTOU_REG</a>	Timeout configuration	0x0058	R/W
<a href="#">UART_AT_CMD_CHAR_REG</a>	AT escape sequence detection configuration	0x005C	R/W
<b>Version Register</b>			
<a href="#">UART_DATE_REG</a>	UART version control register	0x007C	R/W
<a href="#">UART_ID_REG</a>	UART ID register	0x0080	varies

## 19.7 Registers

The addresses in this section are relative to **UART Controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 19.1. UART\_FIFO\_REG (0x0000)**

(reserved)																								UART_RXFIFO_RD_BYTE									
31																								8	7	0							
0 0																								0								Reset	

**UART\_RXFIFO\_RD\_BYTE** UART<sub>n</sub> accesses FIFO via this field. (RO)

**Register 19.2. UART\_MEM\_CONF\_REG (0x0060)**

(reserved)																UART_MEM_FORCE_PU																UART_MEM_FORCE_PD																UART_RX_TOUT_THRHD																UART_RX_FLOW_THRHD																UART_TX_SIZE																UART_RX_SIZE																(reserved)															
31				28				27				26				25								16				15								7				6								4				3								1				0																																																															
0				0				0				0				0				0				0xa								0x0								0x1								1								0				Reset																																																																			

**UART\_RX\_SIZE** This field is used to configure the amount of RAM allocated for RX FIFO. The default number is 128 bytes. (R/W)

**UART\_TX\_SIZE** This field is used to configure the amount of RAM allocated for TX FIFO. The default number is 128 bytes. (R/W)

**UART\_RX\_FLOW\_THRHD** This field is used to configure the maximum amount of data bytes that can be received when hardware flow control works. (R/W)

**UART\_RX\_TOUT\_THRHD** This field is used to configure the threshold time that the receiver takes to receive one byte, in the unit of bit time (the time it takes to transfer one bit). The UART\_RXFIFO\_TOUT\_INT interrupt will be triggered when the receiver takes more time to receive one byte with UART\_RX\_TOUT\_EN set to 1. (R/W)

**UART\_MEM\_FORCE\_PD** Set this bit to force power down UART RAM. (R/W)

**UART\_MEM\_FORCE\_PU** Set this bit to force power up UART RAM. (R/W)

**Register 19.3. UART\_INT\_RAW\_REG (0x0004)**

(reserved)																															UART_WAKEUP_INT_RAW 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_GLITCH_DET_INT_RAW UART_SW_XON_INT_RAW UART_SW_XOFF_INT_RAW UART_RXFIFO_TOUT_INT_RAW UART_BRK_DET_INT_RAW UART_CTS_CHG_INT_RAW UART_DSR_CHG_INT_RAW UART_RXFIFO_EMPTY_INT_RAW UART_TXFIFO_FULL_INT_RAW																		
31																				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	0	0	0	1	0	Reset												

**UART\_RXFIFO\_FULL\_INT\_RAW** This interrupt raw bit turns to high level when the receiver receives more data than what UART\_RXFIFO\_FULL\_THRHD specifies. (R/WTC/SS)

**UART\_TXFIFO\_EMPTY\_INT\_RAW** This interrupt raw bit turns to high level when the amount of data in TX FIFO is less than what UART\_TXFIFO\_EMPTY\_THRHD specifies. (R/WTC/SS)

**UART\_PARITY\_ERR\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a parity error in the data. (R/WTC/SS)

**UART\_FRM\_ERR\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a framing error. (R/WTC/SS)

**UART\_RXFIFO\_OVF\_INT\_RAW** This interrupt raw bit turns to high level when the receiver receives more data than the capacity of RX FIFO. (R/WTC/SS)

**UART\_DSR\_CHG\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects the edge change of DSRn signal. (R/WTC/SS)

**UART\_CTS\_CHG\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects the edge change of CTSn signal. (R/WTC/SS)

**UART\_BRK\_DET\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a 0 after the stop bit. (R/WTC/SS)

**UART\_RXFIFO\_TOUT\_INT\_RAW** This interrupt raw bit turns to high level when the receiver takes more time than UART\_RX\_TOUT\_THRHD to receive a byte. (R/WTC/SS)

**UART\_SW\_XON\_INT\_RAW** This interrupt raw bit turns to high level when the receiver receives an XON character and UART\_SW\_FLOW\_CON\_EN is set to 1. (R/WTC/SS)

**UART\_SW\_XOFF\_INT\_RAW** This interrupt raw bit turns to high level when the receiver receives an XOFF character and UART\_SW\_FLOW\_CON\_EN is set to 1. (R/WTC/SS)

**UART\_GLITCH\_DET\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a glitch in the middle of a start bit. (R/WTC/SS)

Continued on the next page...

**Register 19.3. UART\_INT\_RAW\_REG (0x0004)**

Continued from the previous page...

**UART\_TX\_BRK\_DONE\_INT\_RAW** This interrupt raw bit turns to high level when the transmitter completes sending NULL characters, after all data in TX FIFO are sent. (R/WTC/SS)

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_RAW** This interrupt raw bit turns to high level when the transmitter has kept the shortest duration after sending the last data. (R/WTC/SS)

**UART\_TX\_DONE\_INT\_RAW** This interrupt raw bit turns to high level when the transmitter has sent out all data in FIFO. (R/WTC/SS)

**UART\_RS485\_PARITY\_ERR\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a parity error from the echo of the transmitter in RS485 mode. (R/WTC/SS)

**UART\_RS485\_FRM\_ERR\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects a framing error from the echo of the transmitter in RS485 mode. (R/WTC/SS)

**UART\_RS485\_CLASH\_INT\_RAW** This interrupt raw bit turns to high level when a collision is detected between the transmitter and the receiver in RS485 mode. (R/WTC/SS)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_RAW** This interrupt raw bit turns to high level when the receiver detects the configured UART\_AT\_CMD\_CHAR. (R/WTC/SS)

**UART\_WAKEUP\_INT\_RAW** This interrupt raw bit turns to high level when the input RXD edge changes more times than what (UART\_ACTIVE\_THRESHOLD + 3) specifies in Light-sleep mode. (R/WTC/SS)

**Register 19.4. UART\_INT\_ST\_REG (0x0008)**

(reserved)																UART_WAKEUP_INT_ST UART_AT_CMD_CHAR_DET_INT_ST UART_RS485_CLASH_INT_ST UART_RS485_FRM_ERR_INT_ST UART_TX_DONE_INT_ST UART_TX_BRK_IDLE_INT_ST UART_TX_BRK_DONE_INT_ST UART_GLITCH_DET_INT_ST UART_SW_XON_INT_ST UART_SW_XOFF_INT_ST UART_RXFIFO_TOUT_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																	
31													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	Reset				

**UART\_RXFIFO\_FULL\_INT\_ST** This is the status bit for the UART\_RXFIFO\_FULL\_INT interrupt when UART\_RXFIFO\_FULL\_INT\_ENA is set to 1. (RO)

**UART\_TXFIFO\_EMPTY\_INT\_ST** This is the status bit for the UART\_TXFIFO\_EMPTY\_INT when UART\_TXFIFO\_EMPTY\_INT\_ENA is set to 1. (RO)

**UART\_PARITY\_ERR\_INT\_ST** This is the status bit for the UART\_PARITY\_ERR\_INT interrupt when UART\_PARITY\_ERR\_INT\_ENA is set to 1. (RO)

**UART\_FRM\_ERR\_INT\_ST** This is the status bit for the UART\_FRM\_ERR\_INT interrupt when UART\_FRM\_ERR\_INT\_ENA is set to 1. (RO)

**UART\_RXFIFO\_OVF\_INT\_ST** This is the status bit for the UART\_RXFIFO\_OVF\_INT interrupt when UART\_RXFIFO\_OVF\_INT\_ENA is set to 1. (RO)

**UART\_DSR\_CHG\_INT\_ST** This is the status bit for the UART\_DSR\_CHG\_INT interrupt when UART\_DSR\_CHG\_INT\_ENA is set to 1. (RO)

**UART\_CTS\_CHG\_INT\_ST** This is the status bit for the UART\_CTS\_CHG\_INT interrupt when UART\_CTS\_CHG\_INT\_ENA is set to 1. (RO)

**UART\_BRK\_DET\_INT\_ST** This is the status bit for the UART\_BRK\_DET\_INT interrupt when UART\_BRK\_DET\_INT\_ENA is set to 1. (RO)

**UART\_RXFIFO\_TOUT\_INT\_ST** This is the status bit for the UART\_RXFIFO\_TOUT\_INT interrupt when UART\_RXFIFO\_TOUT\_INT\_ENA is set to 1. (RO)

**UART\_SW\_XON\_INT\_ST** This is the status bit for the UART\_SW\_XON\_INT interrupt when UART\_SW\_XON\_INT\_ENA is set to 1. (RO)

**UART\_SW\_XOFF\_INT\_ST** This is the status bit for the UART\_SW\_XOFF\_INT interrupt when UART\_SW\_XOFF\_INT\_ENA is set to 1. (RO)

**UART\_GLITCH\_DET\_INT\_ST** This is the status bit for the UART\_GLITCH\_DET\_INT interrupt when UART\_GLITCH\_DET\_INT\_ENA is set to 1. (RO)

**UART\_TX\_BRK\_DONE\_INT\_ST** This is the status bit for the UART\_TX\_BRK\_DONE\_INT interrupt when UART\_TX\_BRK\_DONE\_INT\_ENA is set to 1. (RO)

Continued on the next page...



**Register 19.4. UART\_INT\_ST\_REG (0x0008)**

Continued from the previous page...

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_ST** This is the status bit for the UART\_TX\_BRK\_IDLE\_DONE\_INT interrupt when UART\_TX\_BRK\_IDLE\_DONE\_INT\_ENA is set to 1. (RO)

**UART\_TX\_DONE\_INT\_ST** This is the status bit for the UART\_TX\_DONE\_INT interrupt when UART\_TX\_DONE\_INT\_ENA is set to 1. (RO)

**UART\_RS485\_PARITY\_ERR\_INT\_ST** This is the status bit for the UART\_RS485\_PARITY\_ERR\_INT interrupt when UART\_RS485\_PARITY\_INT\_ENA is set to 1. (RO)

**UART\_RS485\_FRM\_ERR\_INT\_ST** This is the status bit for the UART\_RS485\_FRM\_ERR\_INT interrupt when UART\_RS485\_FRM\_ERR\_INT\_ENA is set to 1. (RO)

**UART\_RS485\_CLASH\_INT\_ST** This is the status bit for the UART\_RS485\_CLASH\_INT interrupt when UART\_RS485\_CLASH\_INT\_ENA is set to 1. (RO)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_ST** This is the status bit for the UART\_AT\_CMD\_CHAR\_DET\_INT interrupt when UART\_AT\_CMD\_CHAR\_DET\_INT\_ENA is set to 1. (RO)

**UART\_WAKEUP\_INT\_ST** This is the status bit for the UART\_WAKEUP\_INT interrupt when UART\_WAKEUP\_INT\_ENA is set to 1. (RO)

**Register 19.5. UART\_INT\_ENA\_REG (0x000C)**

(reserved)																												UART_WAKEUP_INT_ENA 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_GLITCH_DET_INT_ENA UART_SW_XON_INT_ENA UART_SW_XOFF_DET_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																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31																								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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_FULL\_INT\_ENA** This is the enable bit for the UART\_RXFIFO\_FULL\_INT interrupt. (R/W)

**UART\_TXFIFO\_EMPTY\_INT\_ENA** This is the enable bit for the UART\_TXFIFO\_EMPTY\_INT interrupt. (R/W)

**UART\_PARITY\_ERR\_INT\_ENA** This is the enable bit for the UART\_PARITY\_ERR\_INT interrupt. (R/W)

**UART\_FRM\_ERR\_INT\_ENA** This is the enable bit for the UART\_FRM\_ERR\_INT interrupt. (R/W)

**UART\_RXFIFO\_OVF\_INT\_ENA** This is the enable bit for the UART\_RXFIFO\_OVF\_INT interrupt. (R/W)

**UART\_DSR\_CHG\_INT\_ENA** This is the enable bit for the UART\_DSR\_CHG\_INT interrupt. (R/W)

**UART\_CTS\_CHG\_INT\_ENA** This is the enable bit for the UART\_CTS\_CHG\_INT interrupt. (R/W)

**UART\_BRK\_DET\_INT\_ENA** This is the enable bit for the UART\_BRK\_DET\_INT interrupt. (R/W)

**UART\_RXFIFO\_TOUT\_INT\_ENA** This is the enable bit for the UART\_RXFIFO\_TOUT\_INT interrupt. (R/W)

**UART\_SW\_XON\_INT\_ENA** This is the enable bit for the UART\_SW\_XON\_INT interrupt. (R/W)

**UART\_SW\_XOFF\_INT\_ENA** This is the enable bit for the UART\_SW\_XOFF\_INT interrupt. (R/W)

**UART\_GLITCH\_DET\_INT\_ENA** This is the enable bit for the UART\_GLITCH\_DET\_INT interrupt. (R/W)

**UART\_TX\_BRK\_DONE\_INT\_ENA** This is the enable bit for the UART\_TX\_BRK\_DONE\_INT interrupt. (R/W)

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_ENA** This is the enable bit for the UART\_TX\_BRK\_IDLE\_DONE\_INT interrupt. (R/W)

**UART\_TX\_DONE\_INT\_ENA** This is the enable bit for the UART\_TX\_DONE\_INT interrupt. (R/W)

Continued on the next page...

**Register 19.5. UART\_INT\_ENA\_REG (0x000C)**

Continued from the previous page...

**UART\_RS485\_PARITY\_ERR\_INT\_ENA** This is the enable bit for the UART\_RS485\_PARITY\_ERR\_INT interrupt. (R/W)

**UART\_RS485\_FRM\_ERR\_INT\_ENA** This is the enable bit for the UART\_RS485\_PARITY\_ERR\_INT interrupt. (R/W)

**UART\_RS485\_CLASH\_INT\_ENA** This is the enable bit for the UART\_RS485\_CLASH\_INT interrupt. (R/W)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_ENA** This is the enable bit for the UART\_AT\_CMD\_CHAR\_DET\_INT interrupt. (R/W)

**UART\_WAKEUP\_INT\_ENA** This is the enable bit for the UART\_WAKEUP\_INT interrupt. (R/W)

**Register 19.6. UART\_INT\_CLR\_REG (0x0010)**

(reserved)																UART_WAKEUP_INT_CLR 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_GLITCH_DET_INT_CLR UART_SW_XOFF_DET_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												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	Reset		

**UART\_RXFIFO\_FULL\_INT\_CLR** Set this bit to clear the UART\_THE\_RXFIFO\_FULL\_INT interrupt. (WT)

**UART\_TXFIFO\_EMPTY\_INT\_CLR** Set this bit to clear the UART\_TXFIFO\_EMPTY\_INT interrupt. (WT)

**UART\_PARITY\_ERR\_INT\_CLR** Set this bit to clear the UART\_PARITY\_ERR\_INT interrupt. (WT)

**UART\_FRM\_ERR\_INT\_CLR** Set this bit to clear the UART\_FRM\_ERR\_INT interrupt. (WT)

**UART\_RXFIFO\_OVF\_INT\_CLR** Set this bit to clear the UART\_UART\_RXFIFO\_OVF\_INT interrupt. (WT)

**UART\_DSR\_CHG\_INT\_CLR** Set this bit to clear the UART\_DSR\_CHG\_INT interrupt. (WT)

**UART\_CTS\_CHG\_INT\_CLR** Set this bit to clear the UART\_CTS\_CHG\_INT interrupt. (WT)

**UART\_BRK\_DET\_INT\_CLR** Set this bit to clear the UART\_BRK\_DET\_INT interrupt. (WT)

**UART\_RXFIFO\_TOUT\_INT\_CLR** Set this bit to clear the UART\_RXFIFO\_TOUT\_INT interrupt. (WT)

**UART\_SW\_XON\_INT\_CLR** Set this bit to clear the UART\_SW\_XON\_INT interrupt. (WT)

**UART\_SW\_XOFF\_INT\_CLR** Set this bit to clear the UART\_SW\_XOFF\_INT interrupt. (WT)

**UART\_GLITCH\_DET\_INT\_CLR** Set this bit to clear the UART\_GLITCH\_DET\_INT interrupt. (WT)

**UART\_TX\_BRK\_DONE\_INT\_CLR** Set this bit to clear the UART\_TX\_BRK\_DONE\_INT interrupt. (WT)

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_CLR** Set this bit to clear the UART\_TX\_BRK\_IDLE\_DONE\_INT interrupt. (WT)

**UART\_TX\_DONE\_INT\_CLR** Set this bit to clear the UART\_TX\_DONE\_INT interrupt. (WT)

**UART\_RS485\_PARITY\_ERR\_INT\_CLR** Set this bit to clear the UART\_RS485\_PARITY\_ERR\_INT interrupt. (WT)

Continued on the next page...

### Register 19.6. UART\_INT\_CLR\_REG (0x0010)

Continued from the previous page...

**UART\_RS485\_FRM\_ERR\_INT\_CLR** Set this bit to clear the UART\_RS485\_FRM\_ERR\_INT interrupt.  
(WT)

**UART\_RS485\_CLASH\_INT\_CLR** Set this bit to clear the UART\_RS485\_CLASH\_INT interrupt. (WT)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_CLR** Set this bit to clear the UART\_AT\_CMD\_CHAR\_DET\_INT interrupt. (WT)

**UART\_WAKEUP\_INT\_CLR** Set this bit to clear the UART\_WAKEUP\_INT interrupt. (WT)

### Register 19.7. UART\_CLKDIV\_REG (0x0014)

(reserved)	UART_CLKDIV_FRAG	(reserved)	UART_CLKDIV					
31 <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	24	23	20	19	12	11	0	
0 0 0 0 0 0 0 0 0	0x0		0 0 0 0 0 0 0 0 0				0x2b6	Reset

**UART\_CLKDIV** The integral part of the frequency divisor. (R/W)

**UART\_CLKDIV\_FRAG** The fractional part of the frequency divisor. (R/W)

### Register 19.8. UART\_RX\_FILT\_REG (0x0018)

[illegible]

**UART\_GLITCH\_FILT** When input pulse width is lower than this value, the pulse is ignored. (R/W)

**UART\_GLITCH\_FILT\_EN** Set this bit to enable RX signal filter. (R/W)

### Register 19.9. UART\_CONFO\_REG (0x0020)

(reserved)		UART_MEM_CLK_EN		UART_AUTOBAUD_EN		UART_ERR_WR_MASK		UART_CLK_EN		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_LOOPBACK		UART_IRDA_RX_INV		UART_IRDA_TX_INV		UART_IRDA_WCTL		UART_IRDA_TX_EN		UART_TXD_DPLX		UART_SW_DTR		UART_STOP_BIT_NUM		UART_BIT_NUM		UART_PARITY_EN		UART_PARITY	
31	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	1	0	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\_PARITY** This bit is used to configure the parity check mode. (R/W)

**UART\_PARITY\_EN** Set this bit to enable UART parity check. (R/W)

**UART\_BIT\_NUM** This field is used to set the length of data. (R/W)

**UART\_STOP\_BIT\_NUM** This field is used to set the length of stop bit. (R/W)

**UART\_SW\_RTS** This bit is used to configure the software RTS signal which is used in software flow control. (R/W)

**UART\_SW\_DTR** This bit is used to configure the software DTR signal which is used in software flow control. (R/W)

**UART\_TXD\_BRK** Set this bit to enable the transmitter to send NULL characters when the process of sending data is done. (R/W)

**UART\_IRDA\_DPLX** Set this bit to enable IrDA loopback mode. (R/W)

**UART\_IRDA\_TX\_EN** This is the start enable bit for IrDA transmitter. (R/W)

**UART\_IRDA\_WCTL** 1: The IrDA transmitter's 11th bit is the same as 10th bit; 0: Set IrDA transmitter's 11th bit to 0. (R/W)

**UART\_IRDA\_TX\_INV** Set this bit to invert the level of IrDA transmitter. (R/W)

**UART\_IRDA\_RX\_INV** Set this bit to invert the level of IrDA receiver. (R/W)

**UART\_LOOPBACK** Set this bit to enable UART loopback test mode. (R/W)

**UART TX FLOW EN** Set this bit to enable flow control function for the transmitter. (R/W)

**UART\_IRDA\_EN** Set this bit to enable IrDA protocol. (R/W)

**UART\_RXFIFO\_RST** Set this bit to reset the UART RX FIFO. (R/W)

**UART\_TXFIFO\_RST** Set this bit to reset the UART TX FIFO. (R/W)

**UART\_RXD\_INV** Set this bit to invert the level value of UART RXD signal. (R/W)

**UART\_CTS\_INV** Set this bit to invert the level value of UART CTS signal. (R/W)

**UART\_DSR\_INV** Set this bit to invert the level value of UART DSR signal. (R/W)

Continued on the next page...

**Register 19.9. UART\_CONFO\_REG (0x0020)**

Continued from the previous page...

**UART\_TXD\_INV** Set this bit to invert the level value of UART TXD signal. (R/W)**UART\_RTS\_INV** Set this bit to invert the level value of UART RTS signal. (R/W)**UART\_DTR\_INV** Set this bit to invert the level value of UART DTR signal. (R/W)**UART\_CLK\_EN** 1: Force clock on for register; 0: Support clock only when application writes registers. (R/W)**UART\_ERR\_WR\_MASK** 1: The receiver stops storing data into FIFO when data is wrong; 0: The receiver stores the data even if the received data is wrong. (R/W)**UART\_AUTOBAUD\_EN** This is the enable bit for baud rate detection. (R/W)**UART\_MEM\_CLK\_EN** The signal to enable UART RAM clock gating. (R/W)**Register 19.10. UART\_CONF1\_REG (0x0024)**

(reserved)										UART_RX_TOUT_EN UART_RX_FLOW_EN UART_RX_TOUT_FLOW_DIS UART_DIS_RX_DAT_OVF				UART_TXFIFO_EMPTY_THRHD				UART_RXFIFO_FULL_THRHD					
31											22	21	20	19	18	17			9	8			0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x60				0x60				Reset	

**UART\_RXFIFO\_FULL\_THRHD** An UART\_RXFIFO\_FULL\_INT interrupt is generated when the receiver receives more data than the value of this field. (R/W)**UART\_TXFIFO\_EMPTY\_THRHD** An UART\_TXFIFO\_EMPTY\_INT interrupt is generated when the number of data bytes in TX FIFO is less than the value of this field. (R/W)**UART\_DIS\_RX\_DAT\_OVF** Disable UART RX data overflow detection. (R/W)**UART\_RX\_TOUT\_FLOW\_DIS** Set this bit to stop accumulating idle\_cnt when hardware flow control works. (R/W)**UART\_RX\_FLOW\_EN** This is the flow enable bit for UART receiver. (R/W)**UART\_RX\_TOUT\_EN** This is the enable bit for UART receiver's timeout function. (R/W)

### Register 19.11. UART\_FLOW\_CONF\_REG (0x0034)

[illegible]

**UART\_SW\_FLOW\_CON\_EN** Set this bit to enable software flow control. When UART receives flow control characters XON or XOFF, which can be configured by UART\_XON\_CHAR or UART\_XOFF\_CHAR respectively, UART\_SW\_XON\_INT or UART\_SW\_XOFF\_INT interrupts can be triggered if enabled. (R/W)

**UART\_XONOFF\_DEL** Set this bit to remove flow control characters from the received data. (R/W)

**UART\_FORCE\_XON** Set this bit to force the transmitter to send data. (R/W)

**UART\_FORCE\_XOFF** Set this bit to stop the transmitter from sending data. (R/W)

**UART\_SEND\_XON** Set this bit to send an XON character. This bit is cleared by hardware automatically.  
(R/W/SS/SC)

**UART\_SEND\_XOFF** Set this bit to send an XOFF character. This bit is cleared by hardware automatically. (R/W/SS/SC)

### Register 19.12. UART\_SLEEP\_CONF\_REG (0x0038)

(reserved)																							UART_ACTIVE_THRESHOLD																											
31																							10										9	0																
0 0																							0xf0																	Reset										

**UART\_ACTIVE\_THRESHOLD** UART is activated from Light-sleep mode when the input RXD edge changes more times than the value of this field plus 3. (R/W)



**Register 19.13. UART\_SWFC\_CONFO\_REG (0x003C)**

(reserved)																UART_XOFF_CHAR								UART_XOFF_THRESHOLD																																							
31																17																16								9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x13								0xe0								Reset																															

**UART\_XOFF\_THRESHOLD** When the number of data bytes in RX FIFO is more than the value of this field with UART\_SW\_FLOW\_CON\_EN set to 1, the transmitter sends an XOFF character. (R/W)

**UART\_XOFF\_CHAR** This field stores the XOFF flow control character. (R/W)

**Register 19.14. UART\_SWFC\_CONF1\_REG (0x0040)**

(reserved)																UART_XON_CHAR								UART_XON_THRESHOLD										
31																17	16								9	8								0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x11								0x0								Reset		

**UART\_XON\_THRESHOLD** When the number of data bytes in RX FIFO is less than the value of this field with UART\_SW\_FLOW\_CON\_EN set to 1, the transmitter sends an XON character. (R/W)

**UART\_XON\_CHAR** This field stores the XON flow control character. (R/W)

**Register 19.15. UART\_TXBRK\_CONF\_REG (0x0044)**

(reserved)																								UART_TX_BRK_NUM									
31																								8	7	0							
0 0																								Oxa								Reset	

**UART\_TX\_BRK\_NUM** This field is used to configure the number of 0 to be sent after the process of sending data is done. It is active when UART\_TXD\_BRK is set to 1. (R/W)

### Register 19.16. UART\_IDLE\_CONF\_REG (0x0048)

Register structure diagram for UART\_TX\_IDLE\_NUM and UART\_RX\_IDLE\_THRHD. The register is 32 bits wide, divided into three fields: (reserved) [31:20], UART\_TX\_IDLE\_NUM [19:10], and UART\_RX\_IDLE\_THRHD [9:0]. The UART\_TX\_IDLE\_NUM field is labeled with a value of 0x100, and the UART\_RX\_IDLE\_THRHD field is labeled with a value of 0x100. A 'Reset' label is at the bottom right.

**UART\_RX\_IDLE\_THRHD** A frame end signal is generated when the receiver takes more time to receive one byte data than the value of this field, in the unit of bit time (the time it takes to transfer one bit). (R/W)

**UART\_TX\_IDLE\_NUM** This field is used to configure the duration time between transfers, in the unit of bit time (the time it takes to transfer one bit). (R/W)

### Register 19.17. UART\_RS485\_CONF\_REG (0x004C)

[illegible]

**UART\_RS485\_EN** Set this bit to choose RS485 mode. (R/W)

**UART\_DLO\_EN** Configures whether or not to add a turnaround delay of 1 bit before the start bit.

0: Not add

1: Add

(R/W)

**UART\_DL1\_EN** Configures whether or not to add a turnaround delay of 1 bit after the stop bit.

0: Not add

1: Add

(R/W)

**UART\_RS485TX\_RX\_EN** Set this bit to enable the receiver could receive data when the transmitter is transmitting data in RS485 mode. (R/W)

**UART\_RS485RXBY\_TX\_EN** Set this bit to enable RS485 transmitter to send data when RS485 receiver line is busy. (R/W)

**UART\_RS485\_RX\_DLY\_NUM** This bit is used to delay the receiver's internal data signal. (R/W)

**UART RS485 TX DLY NUM** This field is used to delay the transmitter's internal data signal. (R/W)

**Register 19.18. UART\_CLK\_CONF\_REG (0x0078)**

(reserved)						UART_RX_SCLK_EN UART_TX_SCLK_EN UART_RST_CORE UART_SCLK_EN UART_SCLK_SEL					UART_SCLK_DIV_NUM					UART_SCLK_DIV_A					UART_SCLK_DIV_B												
31						26	25	24	23	22	21	20	19						12	11						6	5						0
0 0 0 0 0 0						1	1	0	1	3			0x1					0x0					0x0					Reset					

Reset

**UART\_SCLK\_DIV\_B** The denominator of the frequency divisor. (R/W)**UART\_SCLK\_DIV\_A** The numerator of the frequency divisor. (R/W)**UART\_SCLK\_DIV\_NUM** The integral part of the frequency divisor. (R/W)**UART\_SCLK\_SEL** Selects UART clock source. 1: PLL\_F40M\_CLK; 2: RC\_FAST\_CLK; 3: XTAL\_CLK. (R/W)**UART\_SCLK\_EN** Set this bit to enable UART TX/RX clock. (R/W)**UART\_RST\_CORE** Write 1 and then write 0 to this bit, to reset UART TX/RX. (R/W)**UART\_TX\_SCLK\_EN** Set this bit to enable UART TX clock. (R/W)**UART\_RX\_SCLK\_EN** Set this bit to enable UART RX clock. (R/W)**Register 19.19. UART\_STATUS\_REG (0x001C)**

UART_TXD UART_RTSN UART_DTRN (reserved)				UART_TXFIFO_CNT								UART_RXD UART_CTSN UART_DSRN (reserved)				UART_RXFIFO_CNT					
31	30	29	28	26	25					16	15	14	13	12	10	9					0
1	1	1	0	0	0	0					1	1	0	0	0	0	0				Reset

Reset

**UART\_RXFIFO\_CNT** Stores the number of valid data bytes in RX FIFO. (RO)**UART\_DSRN** This bit represents the level of the internal UART DSR signal. (RO)**UART\_CTSN** This bit represents the level of the internal UART CTS signal. (RO)**UART\_RXD** This bit represents the level of the internal UART RXD signal. (RO)**UART\_TXFIFO\_CNT** Stores the number of data bytes in TX FIFO. (RO)**UART\_DTRN** This bit represents the level of the internal UART DTR signal. (RO)**UART\_RTSN** This bit represents the level of the internal UART RTS signal. (RO)**UART\_TXD** This bit represents the level of the internal UART TXD signal. (RO)

### Register 19.20. UART\_MEM\_TX\_STATUS\_REG (0x0064)

(reserved)												UART_TX_RADDR												(reserved)		UART_APB_TX_WADDR										Reset
31 21 20												11 10 9												0												
0 0 0 0 0 0 0 0 0 0 0 0												0x0												0		0x0										

**UART\_APB\_TX\_WADDR** This field stores the offset address in TX FIFO when software writes TX FIFO via APB. (RO)

**UART\_TX\_RADDR** This field stores the offset address in TX FIFO when TX FSM reads data via Tx\_FIFO\_Ctrl. (RO)

### Register 19.21. UART\_MEM\_RX\_STATUS\_REG (0x0068)

(reserved)												UART_RX_WADDR												(reserved)			UART_APB_RX_RADDR																																			
31												20												11												10			9												0											
0 0 0 0 0 0 0 0 0 0 0 0												0x100												0			0x100												Reset																							

**UART\_APB\_RX\_RADDR** This field stores the offset address in RX FIFO when software reads data from RX FIFO via APB. UART0 is 0x100. UART1 is 0x180. (RO)

**UART\_RX\_WADDR** This field stores the offset address in RX FIFO when Rx\_FIFO\_Ctrl writes RX FIFO. UART0 is 0x100. UART1 is 0x180. (RO)

### Register 19.22. UART\_FSM\_STATUS\_REG (0x006C)

[illegible]

**UART\_ST\_URX\_OUT** This is the status field of the receiver. (RO)

**UART\_ST\_UTX\_OUT** This is the status field of the transmitter. (RO)

### Register 19.23. UART\_LOWPULSE\_REG (0x0028)

Diagram illustrating the UART\_LSR register structure. The register is 32 bits wide, divided into two 16-bit halves. The upper 16 bits are labeled "(reserved)". The lower 16 bits are labeled "UART\_LOWPUULSE\_MIN\_CNT". The register is also labeled "Reset".

**UART\_LOWPULSE\_MIN\_CNT** This field stores the value of the minimum duration time of the low level pulse, in the unit of APB\_CLK cycles. It is used in baud rate detection. (RO)

### Register 19.24. UART\_HIGHPULSE\_REG (0x002C)

(reserved)																UART_HIGHPULSE_MIN_CNT															
31																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0fff															
																Reset															

**UART\_HIGHPULSE\_MIN\_CNT** This field stores the value of the maximum duration time for the high level pulse, in the unit of APB\_CLK cycles. It is used in baud rate detection. (RO)

### Register 19.25. UART\_RXD\_CNT\_REG (0x0030)

(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										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										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\_RXD\_EDGE\_CNT** This field stores the count of RXD edge change. It is used in baud rate detection. (RO)

**Register 19.26. UART\_POSPULSE\_REG (0x0070)**

(reserved)																UART_POSEDGE_MIN_CNT																																															
31																12																11																0															
0 0																0xffff																Reset																															

**UART\_POSEDGE\_MIN\_CNT** This field stores the minimal input clock count between two positive edges. It is used in baud rate detection. (RO)

**Register 19.27. UART\_NEGPULSE\_REG (0x0074)**

(reserved)																								UART_NEGEDGE_MIN_CNT																							
31																								12								11								0							
0 0																								0fff								Reset															

**UART\_NEGEDGE\_MIN\_CNT** This field stores the minimal input clock count between two negative edges. It is used in baud rate detection. (RO)

**Register 19.28. UART\_AT\_CMD\_PRECNT\_REG (0x0050)**

(reserved)																UART_PRE_IDLE_NUM																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x901																Reset															

**UART\_PRE\_IDLE\_NUM** This field is used to configure the idle duration time before the first AT\_CMD is received by the receiver, in the unit of bit time (the time it takes to transfer one bit). (R/W)

### Register 19.29. UART\_AT\_CMD\_POSTCNT\_REG (0x0054)

(reserved)																UART_POST_IDLE_NUM																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x901																Reset															

**UART\_POST\_IDLE\_NUM** This field is used to configure the duration time between the last AT\_CMD and the next data byte, in the unit of bit time (the time it takes to transfer one bit). (R/W)

### Register 19.30. UART\_AT\_CMD\_GAPTOOUT\_REG (0x0058)

Diagram illustrating the structure of the 32-bit register:

- Bits 31 to 16: (reserved)
- Bits 15 to 0: UART\_RX\_GAP\_TOUT
- Bit 0: Reset (highlighted in red)

**UART\_RX\_GAP\_TOUT** This field is used to configure the duration time between the AT\_CMD characters, in the unit of bit time (the time it takes to transfer one bit). (R/W)

### Register 19.31. UART\_AT\_CMD\_CHAR\_REG (0x005C)

(reserved)																UART_CHAR_NUM								UART_AT_CMD_CHAR																	
31																15								7								0									
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x3								0x2b										Reset							

**UART\_AT\_CMD\_CHAR** This field is used to configure the content of AT\_CMD character. (R/W)

**UART\_CHAR\_NUM** This field is used to configure the number of continuous AT\_CMD characters received by the receiver. (R/W)

Register 19.32. UART\_DATE\_REG (0x007C)

UART_DATE	
31	0
0x2008270	
Reset	

**UART\_DATE** This is the version control register. (R/W)

Register 19.33. UART\_ID\_REG (0x0080)

UART_REG_UPDATE		UART_ID	
UART_UPDATE_CTRL			
31	30	29	0
0	1	0x000500	
Reset			

**UART\_ID** This field is used to configure the UART\_ID. (R/W)

**UART\_UPDATE\_CTRL** This bit is used to control register synchronization mode. This bit must be cleared before writing 1 to UART\_REG\_UPDATE to synchronize configured values to UART Core’s clock domain. (R/W)

**UART\_REG\_UPDATE** When this bit is set to 1 by software, registers are synchronized to UART Core’s clock domain. This bit is cleared by hardware after synchronization is done. (R/W/SC)



## Chapter 20

## SPI Controller (SPI)

### 20.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP8684 chip integrates three SPI controllers:

- SPI0,
- SPI1,
- and General Purpose SPI2 (GP-SPI2).

SPI0 and SPI1 controllers (MSPI) are primarily reserved for internal use to communicate with external flash and PSRAM memory. This chapter mainly focuses on the GP-SPI2 controller.

### 20.2 Glossary

To better illustrate the functions of GP-SPI2, the following terms are used in this chapter.

<b>Master Mode</b>	GP-SPI2 acts as an SPI master and initiates SPI transactions.
<b>Slave Mode</b>	GP-SPI2 acts as an SPI slave and exchanges data with its master when its CS is asserted.
<b>MISO</b>	Master in, slave out, data transmission from a slave to a master.
<b>MOSI</b>	Master out, slave in, data transmission from a master to a slave
<b>Transaction</b>	One instance of a master asserting a CS line, transferring data to and from a slave, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
<b>SPI Transfer</b>	The whole process of an SPI master exchanging data with a slave. One SPI transfer consists of one or more SPI transactions.
<b>Single Transfer</b>	An SPI transfer that consists of only one transaction.
<b>CPU-Controlled Transfer</b>	A data transfer that happens between CPU buffer <a href="#">SPI_WO_REG</a> ~ <a href="#">SPI_W15_REG</a> and SPI peripheral.
<b>DMA-Controlled Transfer</b>	A data transfer that happens between DMA and SPI peripheral, controlled by the DMA engine.
<b>Configurable Segmented Transfer</b>	A data transfer controlled by DMA in SPI master mode. Such transfer consists of multiple transactions (segments), and each transaction can be configured independently.
<b>Slave Segmented Transfer</b>	A data transfer controlled by DMA in SPI slave mode. Such transfer consists of multiple transactions (segments).

<b>Full-duplex</b>	The sending line and receiving line between the master and the slave are independent. Sending data and receiving data happen at the same time.
<b>Half-duplex</b>	Only one side, the master or the slave, sends data, and the other side receives data. Sending data and receiving data can not happen simultaneously on one side.
<b>4-line full-duplex</b>	4-line here means: clock line, CS line, and two data lines. The two data lines can be used to send or receive data simultaneously.
<b>4-line half-duplex</b>	4-line here means: clock line, CS line, and two data lines. The two data lines can not be used simultaneously.
<b>3-line half-duplex</b>	3-line here means: clock line, CS line, and one data line. The data line is used to transmit or receive data.
<b>1-bit SPI</b>	In one clock cycle, one bit can be transferred.
<b>(2-bit) Dual SPI</b>	In one clock cycle, two bits can be transferred.
<b>Dual Output Read</b>	A data mode of Dual SPI. In one clock cycle, one bit of a command, or one bit of an address, or two bits of data can be transferred.
<b>Dual I/O Read</b>	Another data mode of Dual SPI. In one clock cycle, one bit of a command, or two bits of an address, or two bits of data can be transferred.
<b>(4-bit) Quad SPI</b>	In one clock cycle, four bits can be transferred.
<b>Quad Output Read</b>	A data mode of Quad SPI. In one clock cycle, one bit of a command, or one bit of an address, or four bits of data can be transferred.
<b>Quad I/O Read</b>	Another data mode of Quad SPI. In one clock cycle, one bit of a command, or four bits of an address, or four bits of data can be transferred.
<b>QPI</b>	In one clock cycle, four bits of a command, or four bits of an address, or four bits of data can be transferred.

## 20.3 Features

Some of the key features of GP-SPI2 are:

- Master and slave modes
- Half- and full-duplex communications
- CPU- and DMA-controlled transfers
- Various data modes:
  - 1-bit SPI mode
  - 2-bit Dual SPI mode
  - 4-bit Quad SPI mode
  - QPI mode
- Configurable module clock frequency:

- Master: up to 40 MHz
- Slave: up to 40 MHz
- Configurable data length:
  - CPU-controlled transfer in master mode or in slave mode: 1 ~ 64 B
  - DMA-controlled single transfer in master mode: 1 ~ 32 KB
  - DMA-controlled configurable segmented transfer in master mode: data length is unlimited
  - DMA-controlled single transfer or segmented transfer in slave mode: data length is unlimited
- Configurable bit read/write order
- Independent interrupts for CPU-controlled transfer and DMA-controlled transfer
- Configurable clock polarity and phase
- Four SPI clock modes: mode 0 ~ mode 3
- Six CS lines in master mode: CS0 ~ CS5
- Able to communicate with SPI devices, such as a sensor, a screen controller, as well as a flash or RAM chip

## 20.4 Architectural Overview

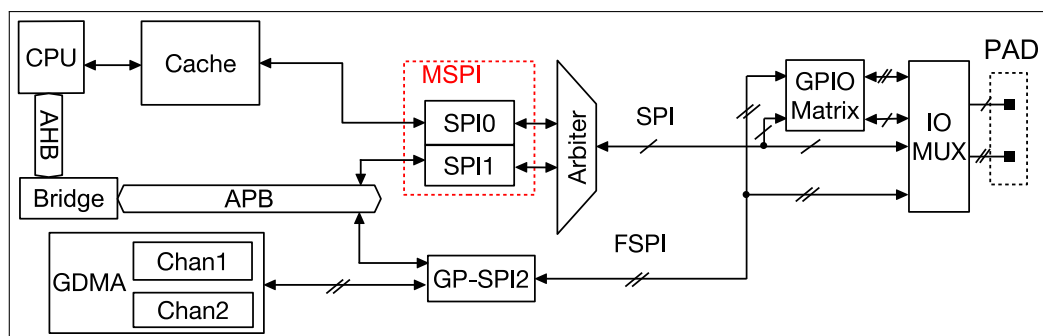


Figure 20.4-1. SPI Module Overview

Figure 20.4-1 shows an overview of SPI module. GP-SPI2 exchanges data with SPI devices by the following ways:

- CPU-controlled transfer: CPU ↔ GP-SPI2 ↔ SPI devices
- DMA-controlled transfer: GDMA ↔ GP-SPI2 ↔ SPI devices

The signals for GP-SPI2 are prefixed with “FSPI” (Fast SPI). FSPI bus signals are routed to GPIO pins via either GPIO matrix or IO MUX. For more information, see Chapter 5 [IO MUX and GPIO Matrix \(GPIO, IO MUX\)](#).

## 20.5 Functional Description

## 20.5.1 Data Modes

GP-SPI2 can be configured as either a master or a slave to communicate with other SPI devices in the following data modes, see Table 20.5-1.

**Table 20.5-1. Data Modes Supported by GP-SPI2**

Supported Mode		CMD State	Address State	Data State
1-bit SPI		1-bit	1-bit	1-bit
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit
	Dual I/O Read	1-bit	2-bit	2-bit
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit
	Quad I/O Read	1-bit	4-bit	4-bit
QPI		4-bit	4-bit	4-bit

For more information about the data modes used when GP-SPI2 works as a master or a slave, see Section 20.5.8 and Section 20.5.9, respectively.

## 20.5.2 Introduction to FSPI Bus Signals

Functional description of FSPI bus signals is shown in Table 20.5-2. Table 20.5-3 lists the signals used in various SPI modes.

**Table 20.5-2. Functional Description of FSPI Bus Signals**

FSPI Bus Signal	Function
FSPID	MOSI/SIO0 (serial data input and output, bit0)
FSPIQ	MISO/SIO1 (serial data input and output, bit1)
FSPIWP	SIO2 (serial data input and output, bit2)
FSPiHD	SIO3 (serial data input and output, bit3)
FSPICLK	Input and output clock in master/slave mode
FSPICSO	Input and output CS signal in master/slave mode
FSPICS1 ~ 5	Output CS signal in master mode

Table 20.5-3. Signals Used in Various SPI Modes

FSPI Signal	Master Mode						Slave Mode					
	FD <sup>1</sup>	1-bit SPI		2-bit Dual SPI	4-bit Quad SPI	QPI	FD	1-bit SPI		2-bit Dual SPI	4-bit Quad SPI	QPI
FSPICLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS1	Y	Y	Y	Y	Y	Y						
FSPICS2	Y	Y	Y	Y	Y	Y						
FSPICS3	Y	Y	Y	Y	Y	Y						
FSPICS4	Y	Y	Y	Y	Y	Y						
FSPICS5	Y	Y	Y	Y	Y	Y						
FSPID	Y	Y	(Y) <sup>3</sup>	Y <sup>4</sup>	Y <sup>5</sup>	Y	Y	Y	(Y) <sup>6</sup>	Y <sup>7</sup>	Y <sup>8</sup>	Y
FSPIQ	Y		(Y) <sup>3</sup>	Y <sup>4</sup>	Y <sup>5</sup>	Y	Y		(Y) <sup>6</sup>	Y <sup>7</sup>	Y <sup>8</sup>	Y
FSPIWP					Y <sup>5</sup>	Y					Y <sup>8</sup>	Y
FSPIHD					Y <sup>5</sup>	Y					Y <sup>8</sup>	Y

<sup>1</sup> FD: full-duplex<sup>2</sup> HD: half-duplex<sup>3</sup> Only one of the two signals is used at a time.<sup>4</sup> The two signals are used in parallel.<sup>5</sup> The four signals are used in parallel.<sup>6</sup> Only one of the two signals is used at a time.<sup>7</sup> The two signals are used in parallel.<sup>8</sup> The four signals are used in parallel.

### 20.5.3 Bit Read/Write Order Control

In master mode:

- The bit order of the command, address and data sent by the GP-SPI2 master is controlled by [SPI\\_WR\\_BIT\\_ORDER](#).
- The bit order of the data received by the master is controlled by [SPI\\_RD\\_BIT\\_ORDER](#).

In slave mode:

- The bit order of the data sent by the GP-SPI2 slave is controlled by [SPI\\_WR\\_BIT\\_ORDER](#).
- The bit order of the command, address and data received by the slave is controlled by [SPI\\_RD\\_BIT\\_ORDER](#).

Table [20.5-4](#) shows the function of [SPI\\_RD/WR\\_BIT\\_ORDER](#).

Table 20.5-4. Bit Order Control in GP-SPI2 Master and Slave Modes

Bit Mode	FSPI Bus Data	<a href="#">SPI_RD/WR_BIT_ORDER</a> = 0 (MSB)	<a href="#">SPI_RD/WR_BIT_ORDER</a> = 2 (MSB)	<a href="#">SPI_RD/WR_BIT_ORDER</a> = 1 (LSB)	<a href="#">SPI_RD/WR_BIT_ORDER</a> = 3 (LSB)
1-bit mode	FSPID or FSPIQ	B7→B6→B5→B4→B3→B2→B1→B0	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7	B0→B1→B2→B3→B4→B5→B6→B7
2-bit mode	FSPIQ	B7→B5→B3→B1	B6→B4→B2→B0	B1→B3→B5→B7	B0→B2→B4→B6
	FSPID	B6→B4→B2→B0	B7→B5→B3→B1	B0→B2→B4→B6	B1→B3→B5→B7
4-bit mode	FSPIHD	B7→B3	B4→B0	B3→B7	B0→B4
	FSPIWP	B6→B2	B5→B1	B2→B6	B1→B5
	FSPIQ	B5→B1	B6→B2	B1→B5	B2→B6
	FSPID	B4→B0	B7→B3	B0→B4	B3→B7

## 20.5.4 Transfer Modes

GP-SPI2 supports the following transfers when working as a master or a slave.

Table 20.5-5. Supported Transfers in Master and Slave Modes

Mode		CPU-Controlled Single Transfer	DMA-Controlled Single Transfer	DMA-Controlled Configurable Segmented Transfer	DMA-Controlled Slave Segmented Transfer
Master	Full-Duplex	Y	Y	Y	–
	Half-Duplex	Y	Y	Y	–
Slave	Full-Duplex	Y	Y	–	Y
	Half-Duplex	Y	Y	–	Y

The following sections provide detailed information about the transfer modes listed in the table above.

## 20.5.5 CPU-Controlled Data Transfer

GP-SPI2 provides 16 x 32-bit data buffers, i.e., `SPI_W0_REG` ~ `SPI_W15_REG`, see Figure 20.5-1.

CPU-controlled transfer indicates the transfer, in which the data to send is from GP-SPI2 data buffer and the received data is stored to GP-SPI2 data buffer. In such transfer, every single transaction needs to be triggered by the CPU, after its related registers are configured. For such reason, the CPU-controlled transfer is always single transfers (consisting of only one transaction). CPU-controlled mode supports full-duplex communication and half-duplex communication.

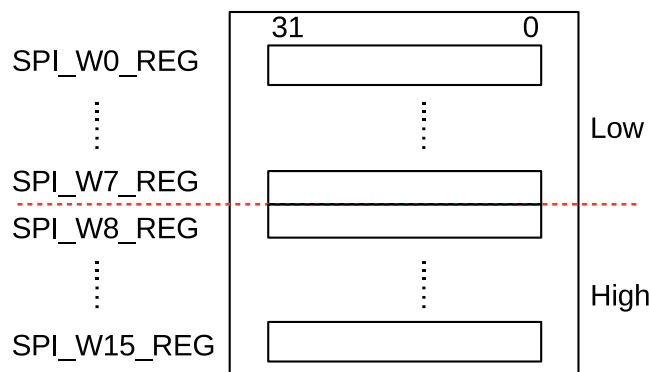


Figure 20.5-1. Data Buffer Used in CPU-Controlled Transfer

### 20.5.5.1 CPU-Controlled Master Mode

In a CPU-controlled master full-duplex or half-duplex transfer, the RX or TX data is saved to or sent from `SPI_W0_REG` ~ `SPI_W15_REG`. The bits `SPI_USR_MOSI_HIGHPART` and `SPI_USR_MISO_HIGHPART` control which buffers are used, see the list below.

- TX data
  - When `SPI_USR_MOSI_HIGHPART` is cleared, i.e. high part mode is disabled, TX data is read from `SPI_W0_REG` ~ `SPI_W15_REG` and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in `SPI_W0_REG` ~ `SPI_W15_REG` may be sent more than once.** Take each 256 bytes as a cycle:



- \* The first 64 bytes (Byte 0 ~ Byte 63) are read from [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 64 ~ Byte 255 are read from [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* Byte 256 ~ Byte 319 (the first 64 bytes in the another 256 bytes) are read from [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#) again, respectively, same as the behaviors described above.

**For instance:** to send 258 bytes (Byte 0 ~ Byte 257), the data is read from the registers as follows:

- \* The first 64 bytes (Byte 0 ~ Byte 63) are read from [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 64 ~ Byte 255 are read from [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* The other bytes (Byte 256 and Byte 257) are read from [SPI\\_WO\\_REG](#)[7:0] and [SPI\\_WO\\_REG](#)[15:8] again, respectively. The logic is:
  - The address to read data for Byte 256 is the result of  $(256 \% 64 = 0)$ , i.e., [SPI\\_WO\\_REG](#)[7:0].
  - The address to read data for Byte 257 is the result of  $(257 \% 64 = 1)$ , i.e., [SPI\\_WO\\_REG](#)[15:8].
- When [SPI\\_USR\\_MOSI\\_HIGHPART](#) is set, i.e. high part mode is enabled, TX data is read from [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#) and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 32, the data in [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#) may be sent more than once.** Take each 256 bytes as a cycle:
  - \* The first 32 bytes (Byte 0 ~ Byte 31) are read from [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
  - \* Byte 32 ~ Byte 255 are read from [SPI\\_W15\\_REG](#)[31:24] repeatedly.
  - \* Byte 256 ~ Byte 287 (the first 32 bytes in the another 256 bytes) are read from [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#) again, respectively, same as the behaviors described above.

**For instance:** to send 258 bytes (Byte 0 ~ Byte 257), the data is read from the registers as follows:

- \* The first 32 bytes (Byte 0 ~ Byte 31) are read from [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 32 ~ Byte 255 are read from [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* The other bytes (Byte 256 and Byte 257) are read from [SPI\\_W8\\_REG](#)[7:0] and [SPI\\_W8\\_REG](#)[15:8] again, respectively. The logic is:
  - The address to read data for Byte 256 is the result of  $(256 \% 32 = 0)$ , i.e., [SPI\\_W8\\_REG](#)[7:0].
  - The address to read data for Byte 257 is the result of  $(257 \% 32 = 1)$ , i.e., [SPI\\_W8\\_REG](#)[15:8].

#### • RX data

- When [SPI\\_USR\\_MISO\\_HIGHPART](#) is cleared, i.e. high part mode is disabled, RX data is saved to [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#) may be overwritten.** Take each 256 bytes as a cycle:
  - \* The first 64 bytes (Byte 0 ~ Byte 63) are saved to [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
  - \* Byte 64 ~ Byte 255 are saved to [SPI\\_W15\\_REG](#)[31:24] repeatedly.

- \* Byte 255 ~ Byte 319 (the first 64 bytes in the another 256 bytes) are saved to [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#) again, respectively, same as the behaviors described above.

**For instance:** to receive 258 bytes (Byte 0 ~ Byte 257), the data is saved to the registers as follows:

- \* The first 64 bytes (Byte 0 ~ Byte 63) are saved to [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 64 ~ Byte 255 are saved to [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* The other bytes (Byte 256 and Byte 257) are saved to [SPI\\_WO\\_REG](#)[7:0] and [SPI\\_WO\\_REG](#)[15:8] again, respectively. The logic is:
  - The address to save Byte 256 is the result of  $(256 \% 64 = 0)$ , i.e., [SPI\\_WO\\_REG](#)[7:0].
  - The address to save Byte 257 is the result of  $(257 \% 64 = 1)$ , i.e., [SPI\\_WO\\_REG](#)[15:8].
- When [SPI\\_USR\\_MISO\\_HIGHPART](#) is set, i.e. high part mode is enabled, the RX data is saved to [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#), and the data address is incremented by 1 on each byte transferred.

**If the data byte length is larger than 32, the content of [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#) may be overwritten.** Take each 256 bytes as a cycle:

- \* Byte 0 ~ Byte 31 are saved to [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 32 ~ Byte 255 are saved to [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* Byte 256 ~ Byte 287 (the first 32 bytes in the another 256 bytes) are saved to [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#) again, respectively.

**For instance:** to receive 258 bytes (Byte 0 ~ Byte 257), the data is saved to the registers as follows:

- \* The first 32 bytes (Byte 0 ~ Byte 31) are saved to [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#), respectively.
- \* Byte 32 ~ Byte 255 are saved to [SPI\\_W15\\_REG](#)[31:24] repeatedly.
- \* The other bytes (Byte 256 and Byte 257) are saved to [SPI\\_W8\\_REG](#)[7:0] and [SPI\\_W8\\_REG](#)[15:8] again, respectively. The logic is:
  - The address to save Byte 256 is the result of  $(256 \% 32 = 0)$ , i.e., [SPI\\_W8\\_REG](#)[7:0].
  - The address to save Byte 257 is the result of  $(257 \% 32 = 1)$ , i.e., [SPI\\_W8\\_REG](#)[15:8].

**Note:**

- TX/RX data address mentioned above both are byte-addressable.
    - If high part mode is disabled, Address 0 stands for [SPI\\_WO\\_REG](#)[7:0], and Address 1 for [SPI\\_WO\\_REG](#)[15:8], and so on.
    - If high part mode is enabled, Address 0 stands for [SPI\\_W8\\_REG](#)[7:0], and Address 1 for [SPI\\_W8\\_REG](#)[15:8], and so on.
- The largest address points to [SPI\\_W15\\_REG](#)[31:24].
- To avoid any possible error in TX/RX data, such as TX data being sent more than once or RX data being overwritten, please make sure the registers are configured correctly.

### 20.5.5.2 CPU-Controlled Slave Mode

In a CPU-controlled slave full-duplex or half-duplex transfer, the RX data or TX data is saved to or sent from [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#), which are byte-addressable.

- In full-duplex communication, the address of [SPI\\_WO\\_REG ~ SPI\\_W15\\_REG](#) starts from 0 and is incremented by 1 on each byte transferred. If the data address is larger than 63, the data in [SPI\\_WO\\_REG ~ SPI\\_W15\\_REG](#) will be overwritten, same as the behaviors described in the master mode when high part mode is disabled.
- In half-duplex communication, the ADDR value in [transmission format](#) is the start address of the RX or TX data, corresponding to the registers [SPI\\_WO\\_REG ~ SPI\\_W15\\_REG](#). The RX or TX address is incremented by 1 on each byte transferred. If the address is larger than 63 (the highest byte address, i.e. [SPI\\_W15\\_REG\[31:24\]](#)), the data in [SPI\\_W8\\_REG ~ SPI\\_W15\\_REG](#) will be overwritten, same as the behaviors described in the master mode when high part mode is enabled.

According to your applications, the registers [SPI\\_WO\\_REG ~ SPI\\_W15\\_REG](#) can be used as:

- data buffers only
- data buffers and status buffers
- status buffers only

## 20.5.6 DMA-Controlled Data Transfer

DMA-controlled transfer refers to the transfer, in which the GDMA RX module receives data and the GDMA TX module sends data. This transfer is supported both in master mode and in slave mode.

A DMA-controlled transfer can be

- a single transfer, consisting of only one transaction. GP-SPI2 supports this transfer both in master and slave modes.
- a configurable segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only in master mode. For more information, see Section [20.5.8.5](#).
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only in slave mode. For more information, see Section [20.5.9.3](#).

A DMA-controlled transfer only needs to be triggered once by CPU. When such a transfer is triggered, data is transferred by the GDMA engine from or to the DMA-linked memory, without CPU operation.

DMA-controlled mode supports full-duplex communication, half-duplex communication and functions described in Section [20.5.8](#) and Section [20.5.9](#). Meanwhile, the GDMA RX module is independent from the GDMA TX module, which means that there are four kinds of full-duplex communications:

- Data is received in DMA-controlled mode and sent in DMA-controlled mode.
- Data is received in DMA-controlled mode but sent in CPU-controlled mode.
- Data is received in CPU-controlled mode but sent in DMA-controlled mode.
- Data is received in CPU-controlled mode and sent in CPU-controlled mode.

### 20.5.6.1 GDMA Configuration

- Select a GDMA channel, and configure a GDMA TX/RX descriptor, see Chapter [2 GDMA Controller \(GDMA\)](#).

- Set the bit [GDMA\\_INLINK\\_START\\_CHn](#) or [GDMA\\_OUTLINK\\_START\\_CHn](#) to start GDMA RX engine and TX engine, respectively.
- Before all the GDMA TX buffer is used or the GDMA TX engine is reset, if [GDMA\\_OUTLINK\\_RESTART\\_CHn](#) is set, a new TX buffer will be added to the end of the last TX buffer in use.
- GDMA RX buffer is linked in the same way as the GDMA TX buffer, by setting [GDMA\\_INLINK\\_START\\_CHn](#) or [GDMA\\_INLINK\\_RESTART\\_CHn](#).
- The TX and RX data lengths are determined by the configured GDMA TX and RX buffer respectively, both of which are 0 ~ 32 KB.
- Initialize GDMA inlink and outlink before GDMA starts. The bits [SPI\\_DMA\\_RX\\_ENA](#) and [SPI\\_DMA\\_TX\\_ENA](#) in register [SPI\\_DMA\\_CONF\\_REG](#) should be set, otherwise the read/write data will be stored to/sent from the registers [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#).

In master mode, if [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT\\_ENA](#) is set, then the interrupt [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#) will be triggered when one single transfer or one configurable segmented transfer is finished.

In slave mode, if [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT\\_ENA](#) is set, then the interrupt [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#) will be triggered when one of the following conditions are met.

**Table 20.5-6. Interrupt Trigger Condition on GP-SPI2 Data Transfer in Slave Mode**

Transfer Type	Control Bit <sup>1</sup>	Control Bit <sup>2</sup>	Condition
Slave Single Transfer	0	0	A single transfer is done.
	1	0	A single transfer is done. Or the length of the received data is equal to ( <a href="#">SPI_MS_DATA_BITLEN</a> + 1)
Slave Segmented Transfer	0	1	( <a href="#">CMD7</a> or <a href="#">End_SEG_TRANS</a> ) is received correctly.
	1	1	( <a href="#">CMD7</a> or <a href="#">End_SEG_TRANS</a> ) is received correctly. Or the length of the received data is equal to ( <a href="#">SPI_MS_DATA_BITLEN</a> + 1)

<sup>1</sup> [SPI\\_RX\\_EOF\\_EN](#)

<sup>2</sup> [SPI\\_DMA\\_SLV\\_SEG\\_TRANS\\_EN](#)

### 20.5.6.2 GDMA TX/RX Buffer Length Control

It is recommended that the length of configured GDMA TX/RX buffer is equal to the length of real transferred data.

- If the length of configured GDMA TX buffer is shorter than that of real transferred data, the extra data will be the same as the last transferred data. [SPI\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) and [GDMA\\_OUT\\_EOF\\_CHn\\_INT](#) are triggered.
- If the length of configured GDMA TX buffer is longer than that of real transferred data, the TX buffer is not fully used, and the remaining buffer will be used for following transaction even if a new TX buffer is linked later. Please keep it in mind. Or save the unused data and reset DMA.
- If the length of configured GDMA RX buffer is shorter than that of real transferred data, the extra data will be lost. The interrupts [SPI\\_INFIFO\\_FULL\\_ERR\\_INT](#) and [SPI\\_TRANS\\_DONE\\_INT](#) are triggered. But [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#) interrupt is not generated.

- If the length of configured GDMA RX buffer is longer than that of real transferred data, the RX buffer is not fully used, and the remaining buffer is discarded. In the following transaction, a new linked buffer will be used directly.

## 20.5.7 Data Flow Control in GP-SPI2 Master and Slave Modes

CPU-controlled and DMA-controlled transfers are supported in GP-SPI2 master and slave modes.

CPU-controlled transfer means that data is transferred between registers `SPI_W0_REG ~ SPI_W15_REG` and the SPI device. DMA-controlled transfer means that data is transferred between the configured GDMA TX/RX buffer and the SPI device. To select between the two transfer modes, configure `SPI_DMA_RX_ENA` and `SPI_DMA_TX_ENA` before the transfer starts.

### 20.5.7.1 GP-SPI2 Functional Blocks

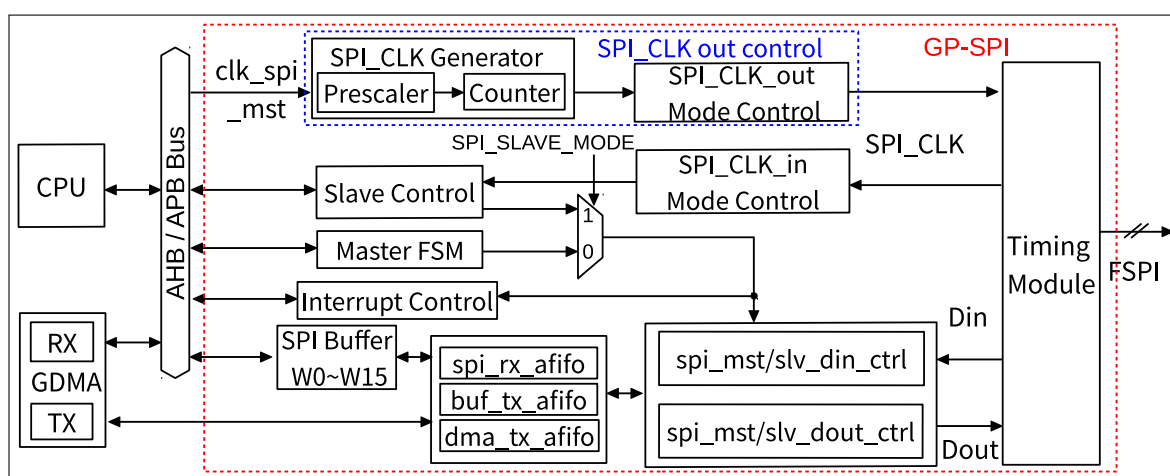


Figure 20.5-2. GP-SPI2 Block Diagram

Figure 20.5-2 shows the main functional blocks in GP-SPI2, including:

- **Master FSM:** all the features, supported in GP-SPI2 master mode, are controlled by this state machine together with register configuration.
- **SPI Buffer:** `SPI_W0_REG ~ SPI_W15_REG`, see Figure 20.5-1. The data transferred in CPU-controlled mode is prepared in this buffer.
- **Timing Module:** capture data on FSPI bus.
- `spi_mst/slv_din_ctrl` and `spi_mst/slv_dout_ctrl`: convert the TX/RX data into bytes.
- `spi_rx_afifo`: store the received data.
- `buf_tx_afifo`: store the data to send.
- `dma_tx_afifo`: store the data from GDMA.
- `clk_spi_mst`: this clock is the module clock of GP-SPI2 and derived from `PLL_CLK`. It is used in GP-SPI2 master mode, to generate `SPI_CLK` signal for data transfer and for slaves.
- **SPI\_CLK Generator:** generate `SPI_CLK` by dividing `clk_spi_mst`. The divider is determined by `SPI_CLKCNT_N` and `SPI_CLKDIV_PRE`, see Section 20.7.

- SPI\_CLK\_out Mode Control: output the SPI\_CLK signal for data transfer and for slaves.
- SPI\_CLK\_in Mode Control: capture the SPI\_CLK signal from SPI master when GP-SPI2 works as a slave.

### 20.5.7.2 Data Flow Control in Master Mode

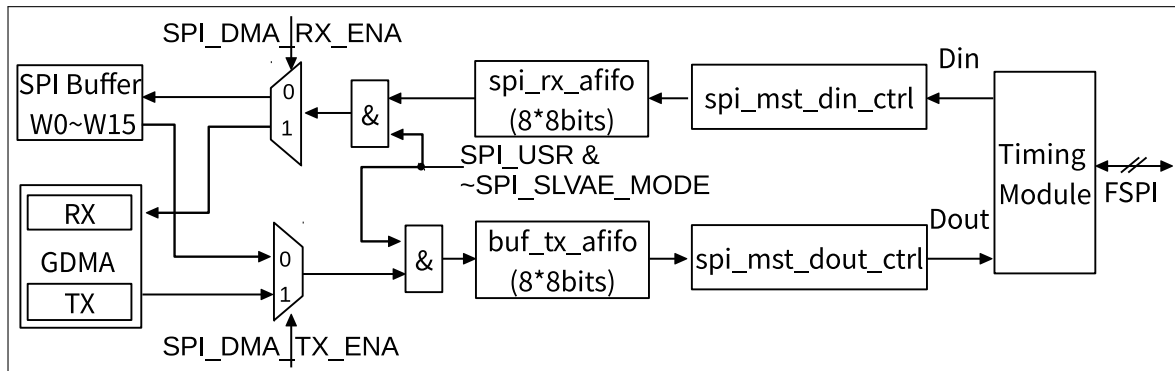


Figure 20.5-3. Data Flow Control in GP-SPI2 Master Mode

Figure 20.5-3 shows the data flow of GP-SPI2 in master mode. Its control logic is as follows:

- RX data: data in FSPI bus is captured by Timing Module, converted in units of bytes by spi\_mst\_din\_ctrl module, then buffered in spi\_rx\_afifo, and finally stored in corresponding addresses according to the transfer modes.
  - CPU-controlled transfer: the data is stored to registers [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#).
  - DMA-controlled transfer: the data is stored to GDMA RX buffer.
- TX data: the TX data is from corresponding addresses according to transfer modes and is saved to buf\_tx\_afifo.
  - CPU-controlled transfer: TX data is from [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#).
  - DMA-controlled transfer: TX data is from GDMA TX buffer.

The data in buf\_tx\_afifo is sent out to Timing Module in 1/2/4-bit modes, controlled by GP-SPI2 state machine. The Timing Module can be used for timing compensation. For more information, see [Section 20.8](#).

### 20.5.7.3 Data Flow Control in Slave Mode

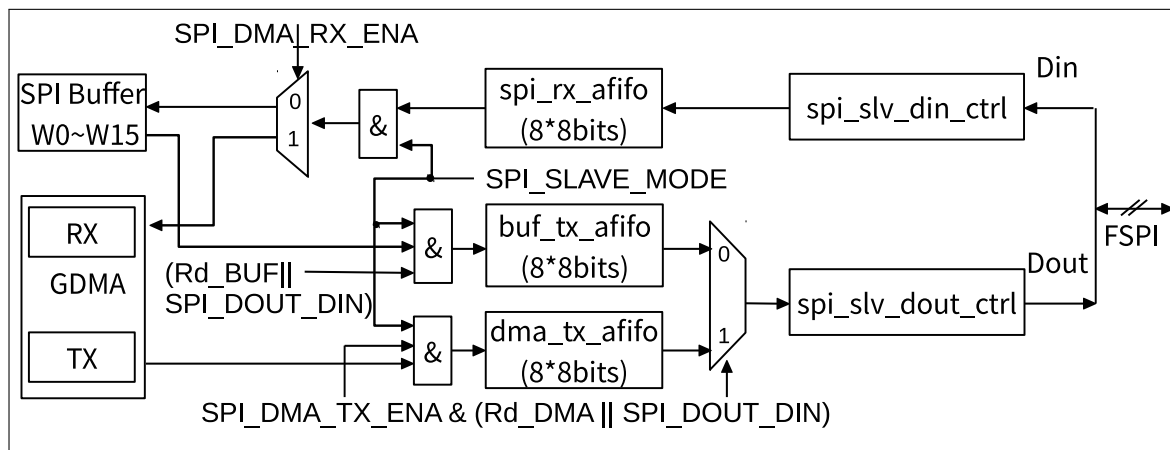


Figure 20.5-4. Data Flow Control in GP-SPI2 Slave Mode

Figure 20.5-4 shows the data flow in GP-SPI2 slave mode. Its control logic is as follows:

- In CPU/DMA-controlled full-duplex/half-duplex modes, when an external SPI master starts the SPI transfer, data on the FSPI bus is captured, converted into unit of bytes by the `spi_slv_din_ctrl` module, and then is stored in `spi_rx_afifo`.
  - In CPU-controlled full-duplex transfer, the received data in `spi_rx_afifo` will be later stored into registers `SPI_W0_REG` ~ `SPI_W15_REG`, successively.
  - In half-duplex `Wr_BUF` transfer, when the value of address (`SLV_ADDR[7:0]`) is received, the received data in `spi_rx_afifo` will be stored in the related address of registers `SPI_W0_REG` ~ `SPI_W15_REG`
  - In DMA-controlled full-duplex transfer or in half-duplex `Wr_DMA` transfer, the received data in `spi_rx_afifo` will be stored in the configured GDMA RX buffer.
- In CPU-controlled full-/half-duplex transfer, the data to send is stored in `buf_tx_afifo`. In DMA-controlled full-/half-duplex transfer, the data to send is stored in `dma_tx_afifo`. Therefore, `Rd_BUF` transaction controlled by CPU and `Rd_DMA` transaction controlled by DMA can be done in one slave segmented transfer. TX data comes from corresponding addresses according the transfer modes.
  - In CPU-controlled full-duplex transfer, when `SPI_SLAVE_MODE` and `SPI_DOUTDIN` are set and `SPI_DMA_TX_ENA` is cleared, the data in `SPI_W0_REG` ~ `SPI_W15_REG` will be stored into `buf_tx_afifo`;
  - In CPU-controlled half-duplex transfer, when `SPI_SLAVE_MODE` is set, `SPI_DOUTDIN` is cleared, `Rd_BUF` command and `SLV_ADDR[7:0]` are received, the data started from the related address of `SPI_W0_REG` ~ `SPI_W15_REG` will be stored into `buf_tx_afifo`;
  - In DMA-controlled full-duplex transfer, when `SPI_SLAVE_MODE`, `SPI_DOUTDIN` and `SPI_DMA_TX_ENA` are set, the data in the configured GDMA TX buffer will be stored into `dma_tx_afifo`;
  - In DMA-controlled half-duplex transfer, when `SPI_SLAVE_MODE` is set, `SPI_DOUTDIN` is cleared, and `Rd_DMA` command is received, the data in the configured GDMA TX buffer will be stored into `dma_tx_afifo`.

The data in `buf_tx_afifo` or `dma_tx_afifo` is sent out by `splv_dout_ctrl` module in 1/2/4-bit modes.

## 20.5.8 GP-SPI2 Works as a Master

GP-SPI2 can be configured as a SPI master by clearing the bit `SPI_SLAVE_MODE` in `SPI_SLAVE_REG`. In this operation mode, GP-SPI2 provides clock signal (the divided clock from GP-SPI2 module clock) and six CS lines (CS0 ~ CS5).

### Note:

- The length of transferred data must be an integral multiple of byte (8 bits), otherwise the extra bits will be lost. The extra bits here means the result of total data bits mod 8.
- To transfer bits that is not an integral multiple of byte (8 bits), consider implementing it in CMD state or ADDR state.

### 20.5.8.1 State Machine

When GP-SPI2 works as a master, the state machine controls its various states during data transfer, including configuration (CONF), preparation (PREP), command (CMD), address (ADDR), dummy (DUMMY), data out (DOUT), and data in (DIN) states. GP-SPI2 is mainly used to access 1/2/4-bit SPI devices, such as flash and external RAM, thus the naming of GP-SPI2 states keeps consistent with the sequence naming of flash and external RAM. The meaning of each state is described as follows and Figure 20.5-5 shows the workflow of GP-SPI2 state machine.

1. IDLE: GP-SPI2 is not active or is in slave mode.
2. CONF: only used in DMA-controlled [configurable segmented transfer](#). Set `SPI_USR` and `SPI_USR_CONF` to enable this state. If this state is not enabled, it means the current transfer is a single transfer.
3. PREP: prepare an SPI transaction and control SPI CS setup time. Set `SPI_USR` and `SPI_CS_SETUP` to enable this state.
4. CMD: send command sequence. Set `SPI_USR` and `SPI_USR_COMMAND` to enable this state.
5. ADDR: send address sequence. Set `SPI_USR` and `SPI_USR_ADDR` to enable this state.
6. DUMMY (wait cycle): send dummy sequence. Set `SPI_USR` and `SPI_USR_DUMMY` to enable this state.
7. DATA: transfer data.
  - DOUT: send data sequence. Set `SPI_USR` and `SPI_USR_MOSI` to enable this state.
  - DIN: receive data sequence. Set `SPI_USR` and `SPI_USR_MISO` to enable this state.
8. DONE: control SPI CS hold time. Set `SPI_USR` to enable this state.



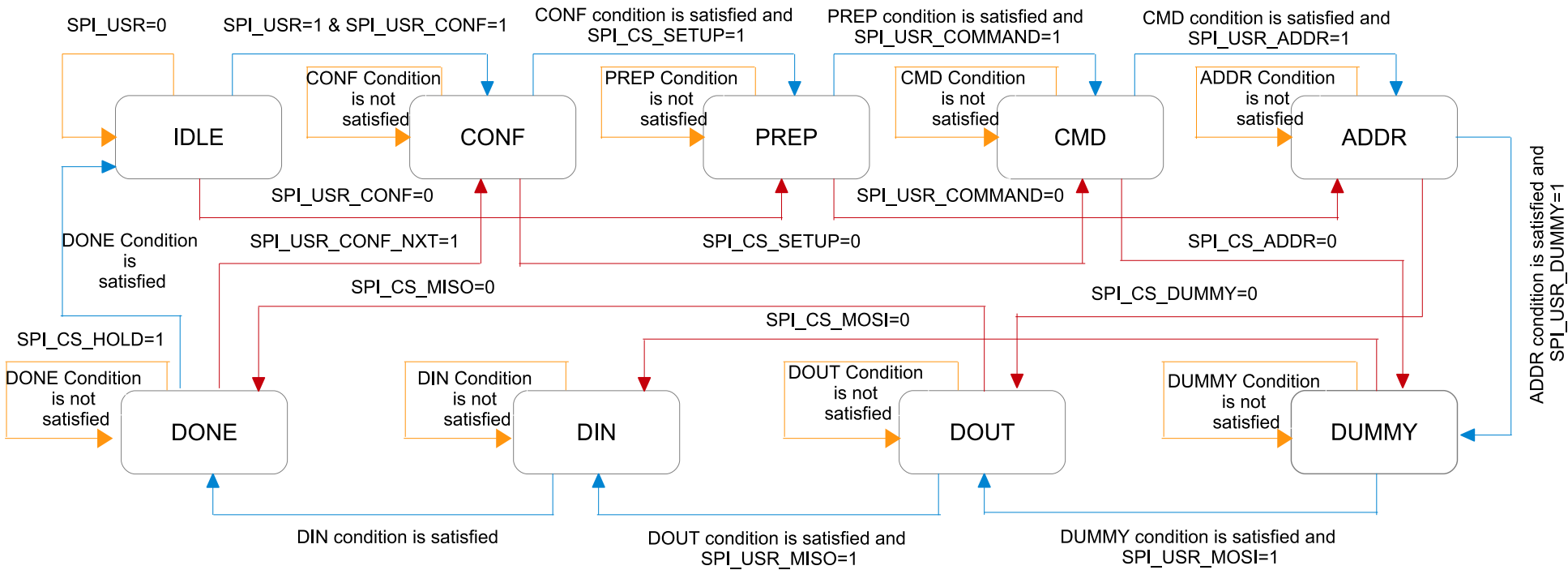


Figure 20.5-5. GP-SPI2 State Machine in Master Mode

Legend to state flow:

- —: indicates corresponding state condition is not satisfied; repeats current state.
- —: corresponding registers are set and conditions are satisfied; goes to next state.
- —: state registers are not set; skips one or more following states, depending on the registers of the following states are set or not.

Explanation to the conditions listed in the figure above:

- CONF condition: `gpc[17:0] >= SPI_CONF_BITLEN[17:0]`
- PREP condition: `gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]`
- CMD condition: `gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]`
- ADDR condition: `gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]`
- DUMMY condition: `gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]`
- DOUT condition: `gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]`
- DIN condition: `gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]`
- DONE condition: `(gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)`

A counter (`gpc[17:0]`) is used in the state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently. The cycle length of each state can also be configured independently.

### 20.5.8.2 Register Configuration for State and Bit Mode Control

#### Introduction

The registers, related to GP-SPI2 state control, are listed in Table 20.5-7. Users can enable QPI mode for GP-SPI2 by setting the bit `SPI_QPI_MODE` in register `SPI_USER_REG`.

Table 20.5-7. Registers Used for State Control in 1/2/4-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus	Control Registers for 4-bit Mode FSPI Bus
CMD	<code>SPI_USR_COMMAND_VALUE</code> <code>SPI_USR_COMMAND_BITLEN</code> <code>SPI_USR_COMMAND</code>	<code>SPI_USR_COMMAND_VALUE</code> <code>SPI_USR_COMMAND_BITLEN</code> <code>SPI_FCMD_DUAL</code> <code>SPI_USR_COMMAND</code>	<code>SPI_USR_COMMAND_VALUE</code> <code>SPI_USR_COMMAND_BITLEN</code> <code>SPI_FCMD_QUAD</code> <code>SPI_USR_COMMAND</code>
ADDR	<code>SPI_USR_ADDR_VALUE</code> <code>SPI_USR_ADDR_BITLEN</code> <code>SPI_USR_ADDR</code>	<code>SPI_USR_ADDR_VALUE</code> <code>SPI_USR_ADDR_BITLEN</code> <code>SPI_USR_ADDR</code> <code>SPI_FADDR_DUAL</code>	<code>SPI_USR_ADDR_VALUE</code> <code>SPI_USR_ADDR_BITLEN</code> <code>SPI_USR_ADDR</code> <code>SPI_FADDR_QUAD</code>
DUMMY	<code>SPI_USR_DUMMY_CYCLELEN</code> <code>SPI_USR_DUMMY</code>	<code>SPI_USR_DUMMY_CYCLELEN</code> <code>SPI_USR_DUMMY</code>	<code>SPI_USR_DUMMY_CYCLELEN</code> <code>SPI_USR_DUMMY</code>
DIN	<code>SPI_USR_MISO</code> <code>SPI_MS_DATA_BITLEN</code>	<code>SPI_USR_MISO</code> <code>SPI_MS_DATA_BITLEN</code> <code>SPI_FREAD_DUAL</code>	<code>SPI_USR_MISO</code> <code>SPI_MS_DATA_BITLEN</code> <code>SPI_FREAD_QUAD</code>

Table 20.5-7. Registers Used for State Control in 1/2/4-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus	Control Registers for 4-bit Mode FSPI Bus
DOUT	<a href="#">SPI_USR_MOSI</a> <a href="#">SPI_MS_DATA_BITLEN</a>	<a href="#">SPI_USR_MOSI</a> <a href="#">SPI_MS_DATA_BITLEN</a> <a href="#">SPI_FWRITE_DUAL</a>	<a href="#">SPI_USR_MOSI</a> <a href="#">SPI_MS_DATA_BITLEN</a> <a href="#">SPI_FWRITE_QUAD</a>

As shown in Table 20.5-7, the registers in each cell should be configured to set the FSPI bus to corresponding bit mode, i.e. the mode shown in the table header, at a specific state (corresponding to the first column).

### Configuration

For instance, when GP-SPI2 reads data, and

- CMD is in 1-bit mode
- ADDR is in 2-bit mode
- DUMMY is 8 clock cycles
- DIN is in 4-bit mode

The register configuration can be as follows:

#### 1. Configure CMD state related registers.

- Configure the required command value in [SPI\\_USR\\_COMMAND\\_VALUE](#).
- Configure command bit length in [SPI\\_USR\\_COMMAND\\_BITLEN](#). [SPI\\_USR\\_COMMAND\\_BITLEN](#) = expected bit length - 1.
- Set [SPI\\_USR\\_COMMAND](#).
- Clear [SPI\\_FCMD\\_DUAL](#) and [SPI\\_FCMD\\_QUAD](#).

#### 2. Configure ADDR state related registers.

- Configure the required address value in [SPI\\_USR\\_ADDR\\_VALUE](#).
- Configure address bit length in [SPI\\_USR\\_ADDR\\_BITLEN](#). [SPI\\_USR\\_ADDR\\_BITLEN](#) = expected bit length - 1.
- Set [SPI\\_USR\\_ADDR](#) and [SPI\\_FADDR\\_DUAL](#).
- Clear [SPI\\_FADDR\\_QUAD](#).

#### 3. Configure DUMMY state related registers.

- Configure DUMMY cycles in [SPI\\_USR\\_DUMMY\\_CYCLELEN](#). [SPI\\_USR\\_DUMMY\\_CYCLELEN](#) = expected clock cycles - 1.
- Set [SPI\\_USR\\_DUMMY](#).

#### 4. Configure DIN state related registers.

- Configure read data bit length in [SPI\\_MS\\_DATA\\_BITLEN](#). [SPI\\_MS\\_DATA\\_BITLEN](#) = bit length expected - 1.

- Set [SPI\\_FREAD\\_QUAD](#) and [SPI\\_USR\\_MISO](#).
- Clear [SPI\\_FREAD\\_DUAL](#).
- Configure GDMA in DMA-controlled mode. In CPU controlled mode, no action is needed.

5. Clear [SPI\\_USR\\_MOSI](#).

6. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#), and [SPI\\_RX\\_AFIFO\\_RST](#) to reset these buffers.

7. Set [SPI\\_USR](#) to start GP-SPI2 transfer.

When writing data (DOUT state), [SPI\\_USR\\_MOSI](#) should be configured instead, while [SPI\\_USR\\_MISO](#) should be cleared. The output data bit length is the value of [SPI\\_MS\\_DATA\\_BITLEN](#) + 1. Output data should be configured in GP-SPI2 data buffer ([SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#)) in CPU-controlled mode, or GDMA TX buffer in DMA-controlled mode. The data byte order is incremented from LSB (byte 0) to MSB.

Pay special attention to the command value in [SPI\\_USR\\_COMMAND\\_VALUE](#) and to address value in [SPI\\_USR\\_ADDR\\_VALUE](#).

The configuration of command value is as follows:

**Table 20.5-8. Sending Sequence of Command Value**

<a href="#">COMMAND_BITLEN</a> <sup>1</sup>	<a href="#">COMMAND_VALUE</a> <sup>2</sup>	<a href="#">BIT_ORDER</a> <sup>3</sup>	Sending Sequence of Command Value
0 - 7	[7:0]	1	<a href="#">COMMAND_VALUE</a> [ <a href="#">COMMAND_BITLEN</a> :0] is sent first.
		0	<a href="#">COMMAND_VALUE</a> [7:7 - <a href="#">COMMAND_BITLEN</a> ] is sent first.
8 - 15	[15:0]	1	<a href="#">COMMAND_VALUE</a> [7:0] is sent first, and then <a href="#">COMMAND_VALUE</a> [ <a href="#">COMMAND_BITLEN</a> :8] is sent.
		0	<a href="#">COMMAND_VALUE</a> [7:0] is sent first, and then <a href="#">COMMAND_VALUE</a> [15:15 - <a href="#">COMMAND_BITLEN</a> ] is sent.

<sup>1</sup> [SPI\\_USR\\_COMMAND\\_BITLEN](#): this field is used to configure the bit length of the command.

<sup>2</sup> [SPI\\_USR\\_COMMAND\\_VALUE](#): command value is written into this field. For which part of this field is used, see the table above.

<sup>3</sup> [SPI\\_WR\\_BIT\\_ORDER](#): 0: LSB first; 1: MSB first.

The configuration of address value is as follows:

**Table 20.5-9. Sending Sequence of Address Value**

<a href="#">ADDR_BITLEN</a> <sup>1</sup>	<a href="#">ADDR_VALUE</a> <sup>2</sup>	<a href="#">BIT_ORDER</a> <sup>3</sup>	Sending Sequence of Address Value
0 - 7	[31:24]	1	<a href="#">ADDR_VALUE</a> [ <a href="#">ADDR_BITLEN</a> + 24:24] is sent first.
		0	<a href="#">ADDR_VALUE</a> [31:31 - <a href="#">ADDR_BITLEN</a> ] is sent first.
8 - 15	[31:16]	1	<a href="#">ADDR_VALUE</a> [31:24] is sent first, and then <a href="#">ADDR_VALUE</a> [ <a href="#">ADDR_BITLEN</a> + 8:16] is sent.
		0	<a href="#">ADDR_VALUE</a> [31:24] is sent first, and then <a href="#">ADDR_VALUE</a> [23:31 - <a href="#">ADDR_BITLEN</a> ] is sent.

16 - 23	[31:8]	1	<a href="#">ADDR_VALUE[31:16]</a> is sent first, and then <a href="#">ADDR_VALUE[ADDR_BITLEN - 8:8]</a> is sent.
		0	<a href="#">ADDR_VALUE[31:16]</a> is sent first, and then <a href="#">ADDR_VALUE[15:31 - ADDR_BITLEN]</a> is sent.
24 - 31	[31:0]	1	<a href="#">ADDR_VALUE[31:8]</a> is sent first, and then <a href="#">ADDR_VALUE[ADDR_BITLEN - 24:0]</a> is sent.
		0	<a href="#">ADDR_VALUE[31:8]</a> is sent first, and then <a href="#">ADDR_VALUE[7:31 - ADDR_BITLEN]</a> is sent.

<sup>1</sup> [SPI\\_USR\\_ADDR\\_BITLEN](#): this field is used to configure the bit length of the address.

<sup>2</sup> [SPI\\_USR\\_ADDR\\_VALUE](#): address value is written into this field. For which part of this field is used, see the table above.

<sup>3</sup> [SPI\\_WR\\_BIT\\_ORDER](#): 0: LSB first; 1: MSB first.

### 20.5.8.3 Full-Duplex Communication (1-bit Mode Only)

#### Introduction

GP-SPI2 supports SPI full-duplex communication. In this mode, SPI master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode via MOSI (FSPID, sending) and MISO (FSPIQ, receiving) at the same time. To enable this communication mode, set the bit [SPI\\_DOUTDIN](#) in register [SPI\\_USER\\_REG](#). Figure 20.5-6 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

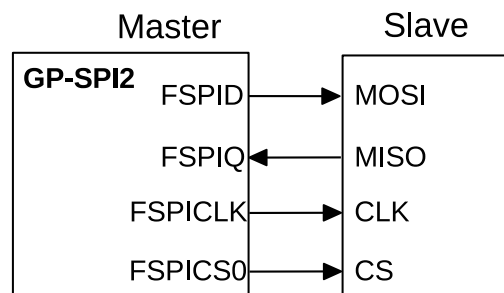


Figure 20.5-6. Full-Duplex Communication Between GP-SPI2 Master and a Slave

In full-duplex communication, the behavior of states CMD, ADDR, DUMMY, DOUT and DIN are configurable. Usually, the states CMD, ADDR and DUMMY are not used in this communication. The bit length of transferred data is configured in [SPI\\_MS\\_DATA\\_BITLEN](#). The actual bit length used in communication equals to ([SPI\\_MS\\_DATA\\_BITLEN](#) + 1).

#### Configuration

To start a data transfer, follow the steps below:

- Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
- Configure AHB clock (AHB\_CLK), APB clock (APB\_CLK, see Chapter 6 [Reset and Clock](#)), and module clock (clk\_spi\_mst) for the GP-SPI2 module.
- Set [SPI\\_DOUTDIN](#) and clear [SPI\\_SLAVE\\_MODE](#), to enable full-duplex communication in master mode.
- Configure GP-SPI2 registers listed in Table 20.5-7.

- Configure SPI CS setup time and hold time according to Section 20.6.
- Set the property of FSPICLK according to Section 20.7.
- Prepare data according to the selected transfer mode:
  - In CPU-controlled MOSI mode, prepare data in registers [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#).
  - In DMA-controlled mode,
    - \* configure [SPI\\_DMA\\_TX\\_ENA/SPI\\_DMA\\_RX\\_ENA](#)
    - \* configure GDMA TX/RX link
    - \* start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
- Configure interrupts and wait for SPI slave to get ready for transfer.
- Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#), and [SPI\\_RX\\_AFIFO\\_RST](#) to reset these buffers.
- Set [SPI\\_USR](#) in register [SPI\\_CMD\\_REG](#) to start the transfer and wait for the configured interrupts.

#### 20.5.8.4 Half-Duplex Communication (1/2/4-bit Mode)

##### Introduction

In this mode, GP-SPI2 provides CLK and CS signals. Only one side (SPI master or slave) can send data at a time, while the other side receives the data. To enable this communication mode, clear the bit [SPI\\_DOUTDIN](#) in register [SPI\\_USER\\_REG](#). The standard format of SPI half-duplex communication is CMD + [ADDR +] [DUMMY +] [DOUT or DIN]. The states ADDR, DUMMY, DOUT, and DIN are optional, and can be disabled or enabled independently.

As described in Section 20.5.8.2, the properties of GP-SPI2 states: CMD, ADDR, DUMMY, DOUT and DIN, such as cycle length, value, and parallel bus bit mode, can be set independently. For the register configuration, see Table 20.5-7.

The detailed properties of half-duplex GP-SPI2 are as follows:

1. CMD: 0 ~ 16 bits, master output, slave input.
2. ADDR: 0 ~ 32 bits, master output, slave input.
3. DUMMY: 0 ~ 256 FSPICLK cycles, master output, slave input.
4. DOUT: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master output, slave input.
5. DIN: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master input, slave output.

##### Configuration

The register configuration is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB\_CLK), APB clock (APB\_CLK), and module clock (clk\_spi\_mst) for the GP-SPI2 module.
3. Clear [SPI\\_DOUTDIN](#) and [SPI\\_SLAVE\\_MODE](#), to enable half-duplex communication in master mode.

4. Configure GP-SPI2 registers listed in Table 20.5-7.
5. Configure SPI CS setup time and hold time according to Section 20.6.
6. Set the property of FSPICLK according to Section 20.7.
7. Prepare data according to the selected transfer mode:
  - In CPU-controlled MOSI mode, prepare data in registers [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#).
  - In DMA-controlled mode,
    - configure [SPI\\_DMA\\_TX\\_ENA](#)/[SPI\\_DMA\\_RX\\_ENA](#)
    - configure GDMA TX/RX link
    - start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
8. Configure interrupts and wait for SPI slave to get ready for transfer.
9. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#), and [SPI\\_RX\\_AFIFO\\_RST](#) to reset these buffers.
10. Set [SPI\\_USR](#) in register [SPI\\_CMD\\_REG](#) to start the transfer and wait for the configured interrupts.

### Application Example

The following example shows how GP-SPI2 accesses flash and external RAM in master half-duplex mode.

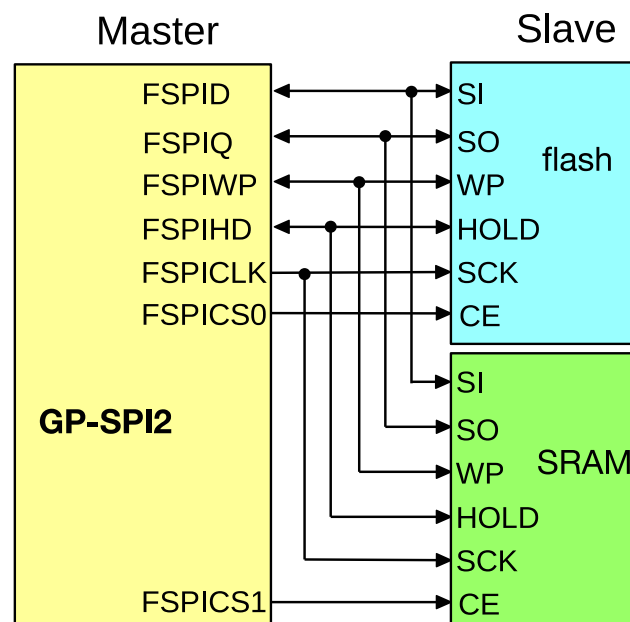


Figure 20.5-7. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

Figure 20.5-8 indicates GP-SPI2 Quad I/O Read sequence according to standard flash specification. Other GP-SPI2 command sequences are implemented in accordance with the requirements of SPI slaves.

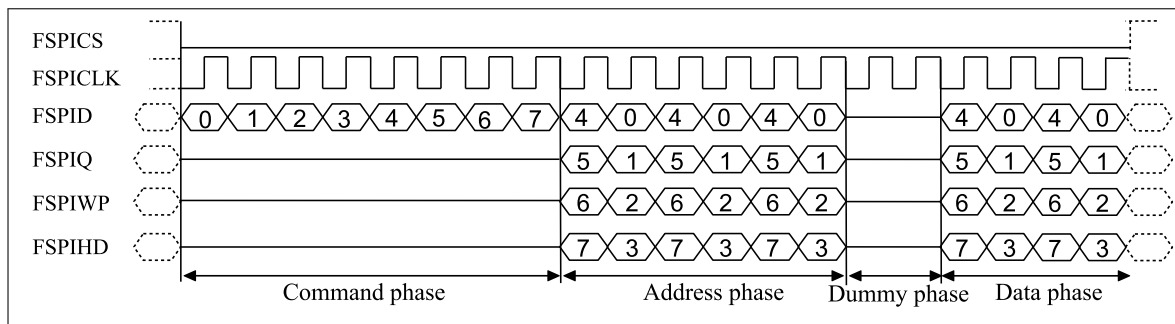


Figure 20.5-8. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

### 20.5.8.5 DMA-Controlled Configurable Segmented Transfer

#### Note:

Note that there is no separate section on how to configure a single transfer in master mode, since the CONF state of a configurable segmented transfer can be skipped to implement a single transfer.

#### Introduction

When GP-SPI2 works as a master, it provides a feature named: configurable segmented transfer controlled by DMA.

A DMA-controlled transfer in master mode can be

- a single transfer, consisting of only one transaction;
- or a configurable segmented transfer, consisting of several transactions (segments).

In a configurable segmented transfer, the registers of each single transaction (segment) are configurable. This feature enables GP-SPI2 to do as many transactions (segments) as configured after such transfer is triggered once by the CPU. Figure 20.5-9 shows how this feature works.

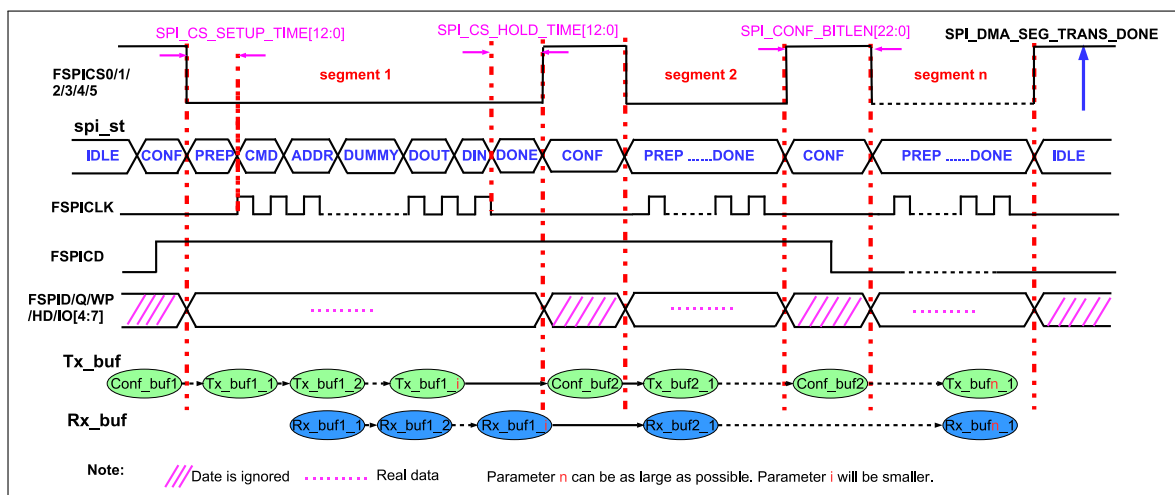


Figure 20.5-9. Configurable Segmented Transfer in DMA-Controlled Master Mode

As shown in Figure 20.5-9, the registers for one transaction (segment *n*) can be reconfigured by GP-SPI2 hardware according to the content in its Conf\_buf*n* during a CONF state, before this segment starts.



It's recommended to provide separate GDMA CONF links and CONF buffers (Conf\_buf*i* in Figure 20.5-9) for each CONF state. A GDMA TX link is used to connect all the CONF buffers and TX data buffers (Tx\_buf*i* in Figure 20.5-9) into a chain. Hence, the behavior of the FSPI bus in each segment can be controlled independently.

For example, in a configurable segment transfer, its segment*i*, segment*j*, and segment*k* can be configured to full-duplex, half-duplex MISO, and half-duplex MOSI, respectively. *i*, *j*, and *k* represent different segment numbers.

Meanwhile, the state of GP-SPI2, the data length and cycle length of the FSPI bus, and the behavior of the GDMA, can be configured independently for each segment. When this whole DMA-controlled transfer (consisting of several segments) has finished, a GP-SPI2 interrupt, [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#), is triggered.

### Configuration

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB\_CLK), APB clock (APB\_CLK), and module clock (clk\_spi\_mst) for GP-SPI2 module.
3. Clear [SPI\\_DOUTDIN](#) and [SPI\\_SLAVE\\_MODE](#), to enable half-duplex communication in master mode.
4. Configure GP-SPI2 registers listed in Table 20.5-7.
5. Configure SPI CS setup time and hold time according to Section 20.6.
6. Set the property of FSPICLK according to Section 20.7.
7. Prepare descriptors for GDMA CONF buffer and TX data (optional) for each segment. Chain the descriptors of CONF buffer and TX buffers of several segments into one linked list.
8. Similarly, prepare descriptors for RX buffers for each segment and chain them into one linked list.
9. Configure all the needed CONF buffers, TX buffers and RX buffers, respectively for each segment before this DMA-controlled transfer begins.
10. Point [GDMA\\_OUTLINK\\_ADDR\\_CH\*n\*](#) to the head address of the CONF and TX buffer descriptor linked list, and then set [GDMA\\_OUTLINK\\_START\\_CH\*n\*](#) to start the TX GDMA.
11. Clear the bit [SPI\\_RX\\_EOF\\_EN](#) in register [SPI\\_DMA\\_CONF\\_REG](#). Point [GDMA\\_INLINK\\_ADDR\\_CH\*n\*](#) to the head address of the RX buffer descriptor linked list, and then set [GDMA\\_INLINK\\_START\\_CH\*n\*](#) to start the RX GDMA.
12. Set [SPI\\_USR\\_CONF](#) to enable CONF state.
13. Set [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT\\_ENA](#) to enable the [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. Configure other interrupts if needed according to Section 20.9.
14. Wait for all the slaves to get ready for transfer.
15. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#) and [SPI\\_RX\\_AFIFO\\_RST](#), to reset these buffers.
16. Set [SPI\\_USR](#) to start this DMA-controlled transfer.
17. Wait for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt, which means this transfer has finished and the data has been stored into corresponding memory.

### Configuration of CONF Buffer and Magic Value

In a configurable segmented transfer, only registers which will change from the last transaction (segment) need to be re-configured to new values in CONF state. The configuration of other registers can be skipped (i.e. kept the same) to save time and chip resources.

The first word in GDMA CONF buffer<sup>*i*</sup>, called SPI\_BIT\_MAP\_WORD, defines whether each GP-SPI2 register is to be updated or not in segment<sup>*i*</sup>. The relation of SPI\_BIT\_MAP\_WORD and GP-SPI2 registers to update can be seen in Bitmap (BM) Table, Table 20.5-10. If a bit in the BM table is set to 1, its corresponding register value will be updated in this segment. Otherwise, if some registers should be kept from being changed, the related bits should be set to 0.

Table 20.5-10. BM Table for CONF State

BM Bit	Register Name	BM Bit	Register Name
0	<a href="#">SPI_ADDR_REG</a>	7	<a href="#">SPI_MISC_REG</a>
1	<a href="#">SPI_CTRL_REG</a>	8	reserved
2	<a href="#">SPI_CLOCK_REG</a>	9	reserved
3	<a href="#">SPI_USER_REG</a>	10	reserved
4	<a href="#">SPI_USER1_REG</a>	11	<a href="#">SPI_DMA_CONF_REG</a>
5	<a href="#">SPI_USER2_REG</a>	12	<a href="#">SPI_DMA_INT_ENA_REG</a>
6	<a href="#">SPI_MS_DLEN_REG</a>	13	<a href="#">SPI_DMA_INT_CLR_REG</a>

Then new values of all the registers to be modified should be placed right after SPI\_BIT\_MAP\_WORD, in consecutive words in the CONF buffer.

To ensure the correctness of the content in each CONF buffer, the value in SPI\_BIT\_MAP\_WORD[31:28] is used as “magic value”, and will be compared with [SPI\\_DMA\\_SEG\\_MAGIC\\_VALUE](#) in register [SPI\\_SLAVE\\_REG](#). The value of [SPI\\_DMA\\_SEG\\_MAGIC\\_VALUE](#) should be configured before this DMA-controlled transfer starts, and can not be changed during these segments.

- If SPI\_BIT\_MAP\_WORD[31:28] == [SPI\\_DMA\\_SEG\\_MAGIC\\_VALUE](#), this DMA-controlled transfer continues normally; the interrupt [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) is triggered at the end of this DMA-controlled transfer.
- If SPI\_BIT\_MAP\_WORD[31:28] != [SPI\\_DMA\\_SEG\\_MAGIC\\_VALUE](#), GP-SPI2 state (spi\_st) goes back to IDLE and the transfer is ended immediately. The interrupt [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) is still triggered, with [SPI\\_SEG\\_MAGIC\\_ERR\\_INT\\_RAW](#) bit set to 1.

### CONF Buffer Configuration Example

Table 20.5-11 and Table 20.5-12 provide an example to show how to configure a CONF buffer for a transaction (segment <sup>*i*</sup>) in which [SPI\\_ADDR\\_REG](#), [SPI\\_CTRL\\_REG](#), [SPI\\_CLOCK\\_REG](#), [SPI\\_USER\\_REG](#), [SPI\\_USER1\\_REG](#) need to be updated.

Table 20.5-11. An Example of CONF buffer<sup>*i*</sup> in Segment<sup>*i*</sup>

CONF buffer <sup><i>i</i></sup>	Note
---------------------------------	------

SPI_BIT_MAP_WORD	The first word in this buffer. Its value is 0xA000001F in this example when the <a href="#">SPI_DMA_SEG_MAGIC_VALUE</a> is set to 0xA. As shown in Table 20.5-12, bits 0, 1, 2, 3, and 4 are set, indicating the following registers will be updated.
<a href="#">SPI_ADDR_REG</a>	The second word, stores the new value to <a href="#">SPI_ADDR_REG</a> .
<a href="#">SPI_CTRL_REG</a>	The third word, stores the new value to <a href="#">SPI_CTRL_REG</a> .
<a href="#">SPI_CLOCK_REG</a>	The fourth word, stores the new value to <a href="#">SPI_CLOCK_REG</a> .
<a href="#">SPI_USER_REG</a>	The fifth word, stores the new value to <a href="#">SPI_USER_REG</a> .
<a href="#">SPI_USER1_REG</a>	The sixth word, stores the new value to <a href="#">SPI_USER1_REG</a> .

Table 20.5-12. BM Bit Value v.s. Register to Be Updated in This Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
0	1	<a href="#">SPI_ADDR_REG</a>	7	0	<a href="#">SPI_MISC_REG</a>
1	1	<a href="#">SPI_CTRL_REG</a>	8	0	reserved
2	1	<a href="#">SPI_CLOCK_REG</a>	9	0	reserved
3	1	<a href="#">SPI_USER_REG</a>	10	0	reserved
4	1	<a href="#">SPI_USER1_REG</a>	11	0	<a href="#">SPI_DMA_CONF_REG</a>
5	0	<a href="#">SPI_USER2_REG</a>	12	0	<a href="#">SPI_DMA_INT_ENA_REG</a>
6	0	<a href="#">SPI_MS_DLEN_REG</a>	13	0	<a href="#">SPI_DMA_INT_CLR_REG</a>

**Notes:**

In a DMA-controlled configurable segmented transfer, please pay special attention to the following bits:

- [SPI\\_USR\\_CONF](#): set [SPI\\_USR\\_CONF](#) before [SPI\\_USR](#) is set, to enable this transfer.
- [SPI\\_USR\\_CONF\\_NXT](#): if segment *i* is not the final transaction of this whole DMA-controlled transfer, its [SPI\\_USR\\_CONF\\_NXT](#) bit should be set to 1.
- [SPI\\_CONF\\_BITLEN](#): GP-SPI2 CS setup time and hold time are programmable independently in each segment, see Section 20.6 for detailed configuration. The CS high time in each segment is about:

$$(\text{SPI\_CONF\_BITLEN} + 5) \times T_{\text{AHB\_CLK}}$$

The CS high time in CONF state can be set from 125 *ns* to 6.5536 *ms* when  $f_{\text{AHB\_CLK}}$  is 40 MHz.

([SPI\\_CONF\\_BITLEN](#) + 5) will overflow from (0x40000 - [SPI\\_CONF\\_BITLEN](#) - 5) if [SPI\\_CONF\\_BITLEN](#) is larger than 0x3FFFA.

### 20.5.9 GP-SPI2 Works as a Slave

GP-SPI2 can be used as a slave to communicate with an SPI master. As a slave, GP-SPI2 supports 1-bit SPI, 2-bit dual SPI, 4-bit quad SPI, and QPI modes, with specific communication formats. To enable this mode, set [SPI\\_SLAVE\\_MODE](#) in register [SPI\\_SLAVE\\_REG](#).

The CS signal must be held low during the transmission, and its falling/rising edges indicate the start/end of a single or segmented transmission. The length of transferred data must be in unit of bytes, otherwise the extra bits will be lost. The extra bits here means the result of total bits % 8.

### 20.5.9.1 Communication Formats

In GP-SPI2 slave mode, SPI full-duplex and half-duplex communications are available. To select from the two communications, configure [SPI\\_DOUTDIN](#) in register [SPI\\_USER\\_REG](#).

Full-duplex communication means that input data and output data are transmitted simultaneously throughout the entire transaction. All bits are treated as input or output data, which means no command, address or dummy states are expected. The interrupt [SPI\\_TRANS\\_DONE\\_INT](#) is triggered once the transaction ends.

In half-duplex communication, the format is CMD+ADDR+DUMMY+DATA (DIN or DOUT).

- “DIN” means that an SPI master reads data from GP-SPI2.
- “DOUT” means that an SPI master writes data to GP-SPI2.

The detailed properties of each state are as follows:

1. CMD:

- Indicate the function of SPI slave;
- One byte from master to slave;
- Only the values in Table [20.5-13](#) and Table [20.5-14](#) are valid;
- Can be sent in 1-bit SPI mode or 4-bit QPI mode.

2. ADDR:

- The address for [Wr\\_BUF](#) and [Rd\\_BUF](#) commands in CPU-controlled transfer, or placeholder bits in other transfers and can be defined by application;
- One byte from master to slave;
- Can be sent in 1-bit, 2-bit or 4-bit modes (according to the command).

3. DUMMY:

- Its value is meaningless. SPI slave prepares data in this state;
- Bit mode of FSPI bus is also meaningless here;
- Last for eight SPI\_CLK cycles.

4. DIN or DOUT:

- Data length can be 0 ~ 64 B in CPU-controlled mode and unlimited in DMA-controlled mode;
- Can be sent in 1-bit, 2-bit or 4-bit modes according to the CMD value.

**Note:**

The states of ADDR and DUMMY can never be skipped in any half-duplex communications.

When a half-duplex transaction is complete, the transferred CMD and ADDR values are latched into [SPI\\_SLV\\_LAST\\_COMMAND](#) and [SPI\\_SLV\\_LAST\\_ADDR](#) respectively. The [SPI\\_SLV\\_CMD\\_ERR\\_INT\\_RAW](#) will be set if the transferred CMD value is not supported by GP-SPI2 slave mode. The [SPI\\_SLV\\_CMD\\_ERR\\_INT\\_RAW](#) can only be cleared by software.

### 20.5.9.2 Supported CMD Values in Half-Duplex Communication

In half-duplex communication, the defined values of CMD determine the transfer types. Unsupported CMD values are disregarded, meanwhile the related transfer is ignored and [SPI\\_SLV\\_CMD\\_ERR\\_INT\\_RAW](#) is set. The transfer format is CMD (8 bits) + ADDR (8 bits) + DUMMY (8 SPI\_CLK cycles) + DATA (unit in bytes). The detailed description of CMD[3:0] is as follows:

- 0x1 (Wr\_BUF): CPU-controlled write mode. Master sends data and GP-SPI2 receives data. The data is stored in the related address of [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#).
- 0x2 (Rd\_BUF): CPU-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from the related address of [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#).
- 0x3 (Wr\_DMA): DMA-controlled write mode. Master sends data and GP-SPI2 receives data. The data is stored in GP-SPI2 GDMA RX buffer.
- 0x4 (Rd\_DMA): DMA-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from GP-SPI2 GDMA TX buffer.
- 0x7 (CMD7): used to generate an [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. It can also generate a GDMA\_IN\_SUC\_EOF\_CH<sub>n</sub>\_INT interrupt in a slave segmented transfer when GDMA RX link is used. But it will not end GP-SPI2's slave segmented transfer.
- 0x8 (CMD8): only used to generate an [SPI\\_SLV\\_CMD8\\_INT](#) interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0x9 (CMD9): only used to generate an [SPI\\_SLV\\_CMD9\\_INT](#) interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0xA (CMDA): only used to generate an [SPI\\_SLV\\_CMDA\\_INT](#) interrupt, which will not end GP-SPI2's slave segmented transfer.

The detailed function of CMD7, CMD8, CMD9, and CMDA commands is reserved for user definition. These commands can be used as handshake signals, as passwords of some specific functions, as triggers of some user defined actions, and so on.

1/2/4-bit modes in states of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY state is always in 1-bit mode and lasts for eight SPI\_CLK cycles. The definition of CMD[7:4] is as follows:

- 0x0: CMD, ADDR, and DATA states all are in 1-bit mode.
- 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
- 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
- 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
- 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode. Or in QPI mode.

In addition, if the value of CMD[7:0] is 0x05, 0xA5, 0x06, or 0xDD, DUMMY and DATA states are skipped. The definition of CMD[7:0] is as follows:

- 0x05 (End\_SEG\_TRANS): master sends 0x05 command to end slave segmented transfer in SPI mode.
- 0xA5 (End\_SEG\_TRANS): master sends 0xA5 command to end slave segmented transfer in QPI mode.

- 0x06 (En\_QPI): GP-SPI2 enters QPI mode when receiving the 0x06 command and the bit [SPI\\_QPI\\_MODE](#) in register [SPI\\_USER\\_REG](#) is set.
- 0xDD (Ex\_QPI): GP-SPI2 exits QPI mode when receiving the 0xDD command and the bit [SPI\\_QPI\\_MODE](#) is cleared.

All the CMD values supported by GP-SPI2 are listed in Table 20.5-13 and Table 20.5-14. Note that DUMMY state is always in 1-bit mode and lasts for eight SPI\_CLK cycles.

**Table 20.5-13. Supported CMD Values in SPI Mode**

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0x01	1-bit mode	1-bit mode	1-bit mode
	0x11	1-bit mode	1-bit mode	2-bit mode
	0x21	1-bit mode	1-bit mode	4-bit mode
	0x51	1-bit mode	2-bit mode	2-bit mode
	0xA1	1-bit mode	4-bit mode	4-bit mode
Rd_BUF	0x02	1-bit mode	1-bit mode	1-bit mode
	0x12	1-bit mode	1-bit mode	2-bit mode
	0x22	1-bit mode	1-bit mode	4-bit mode
	0x52	1-bit mode	2-bit mode	2-bit mode
	0xA2	1-bit mode	4-bit mode	4-bit mode
Wr_DMA	0x03	1-bit mode	1-bit mode	1-bit mode
	0x13	1-bit mode	1-bit mode	2-bit mode
	0x23	1-bit mode	1-bit mode	4-bit mode
	0x53	1-bit mode	2-bit mode	2-bit mode
	0xA3	1-bit mode	4-bit mode	4-bit mode
Rd_DMA	0x04	1-bit mode	1-bit mode	1-bit mode
	0x14	1-bit mode	1-bit mode	2-bit mode
	0x24	1-bit mode	1-bit mode	4-bit mode
	0x54	1-bit mode	2-bit mode	2-bit mode
	0xA4	1-bit mode	4-bit mode	4-bit mode
CMD7	0x07	1-bit mode	1-bit mode	-
	0x17	1-bit mode	1-bit mode	-
	0x27	1-bit mode	1-bit mode	-
	0x57	1-bit mode	2-bit mode	-
	0xA7	1-bit mode	4-bit mode	-
CMD8	0x08	1-bit mode	1-bit mode	-
	0x18	1-bit mode	1-bit mode	-
	0x28	1-bit mode	1-bit mode	-
	0x58	1-bit mode	2-bit mode	-
	0xA8	1-bit mode	4-bit mode	-
CMD9	0x09	1-bit mode	1-bit mode	-
	0x19	1-bit mode	1-bit mode	-
	0x29	1-bit mode	1-bit mode	-
	0x59	1-bit mode	2-bit mode	-
	0xA9	1-bit mode	4-bit mode	-
	0x0A	1-bit mode	1-bit mode	-

Table 20.5-13. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
	0x1A	1-bit mode	1-bit mode	-
	0x2A	1-bit mode	1-bit mode	-
	0x5A	1-bit mode	2-bit mode	-
	0xAA	1-bit mode	4-bit mode	-
End_SEG_TRANS	0x05	1-bit mode	-	-
En_QPI	0x06	1-bit mode	-	-

Table 20.5-14. Supported CMD Values in QPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0xA1	4-bit mode	4-bit mode	4-bit mode
Rd_BUF	0xA2	4-bit mode	4-bit mode	4-bit mode
Wr_DMA	0xA3	4-bit mode	4-bit mode	4-bit mode
Rd_DMA	0xA4	4-bit mode	4-bit mode	4-bit mode
CMD7	0xA7	4-bit mode	4-bit mode	-
CMD8	0xA8	4-bit mode	4-bit mode	-
CMD9	0xA9	4-bit mode	4-bit mode	-
CMDA	0xAA	4-bit mode	4-bit mode	-
End_SEG_TRANS	0xA5	4-bit mode	4-bit mode	-
Ex_QPI	0xDD	4-bit mode	4-bit mode	-

Master sends 0x06 CMD (En\_QPI) to set GP-SPI2 slave to QPI mode and all the states of supported transfer will be in 4-bit mode afterwards. If 0xDD CMD (Ex\_QPI) is received, GP-SPI2 slave will be back to SPI mode.

Other transfer types than described in Table 20.5-13 and Table 20.5-14 are ignored. If the transferred data is not in unit of byte, GP-SPI2 will send or receive the data in unit of byte, but the extra bits (the result of total bits mod 8) will be lost. But if the CS low time is longer than 2 APB clock (APB\_CLK) cycles,

[SPI\\_TRANS\\_DONE\\_INT](#) will be triggered. For more information on interrupts triggered at the end of transmissions, please refer to Section 20.9.

### 20.5.9.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI2 works as a slave, it supports full-duplex and half-duplex communications controlled by DMA and by CPU. DMA-controlled transfer can be a single transfer, or a slave segmented transfer consisting of several transactions (segments). The CPU-controlled transfer can only be one single transfer, since each CPU-controlled transaction needs to be triggered by CPU.

In a slave segmented transfer, all transfer types listed in Table 20.5-13 and Table 20.5-14 are supported in a single transaction (segment). It means that CPU-controlled transaction and DMA-controlled transaction can be mixed in one slave segmented transfer.

It is recommended that in a slave segmented transfer:

- CPU-controlled transaction is used for handshake communication and short data transfers.

- DMA-controlled transaction is used for large data transfers.

#### 20.5.9.4 Configuration of Slave Single Transfer

In slave mode, GP-SPI2 supports CPU/DMA-controlled full-duplex/half-duplex single transfers. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB\_CLK) and APB clock (APB\_CLK).
3. Set the bit [SPI\\_SLAVE\\_MODE](#), to enable slave mode.
4. Configure [SPI\\_DOUTDIN](#):
  - 1: enable full-duplex communication.
  - 0: enable half-duplex communication.
5. Prepare data:
  - if CPU-controlled transfer mode is selected and GP-SPI2 is used to send data, then prepare data in registers [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#).
  - if DMA-controlled transfer mode is selected,
    - configure [SPI\\_DMA\\_TX\\_ENA](#)/[SPI\\_DMA\\_RX\\_ENA](#) and [SPI\\_RX\\_EOF\\_EN](#).
    - configure GDMA TX/RX link.
    - start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
6. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#), and [SPI\\_RX\\_AFIFO\\_RST](#) to reset these buffers.
7. Clear [SPI\\_DMA\\_SLV\\_SEG\\_TRANS\\_EN](#) in register [SPI\\_DMA\\_CONF\\_REG](#) to enable slave single transfer mode.
8. Set [SPI\\_TRANS\\_DONE\\_INT\\_ENA](#) in [SPI\\_DMA\\_INT\\_ENA\\_REG](#) and wait for the interrupt [SPI\\_TRANS\\_DONE\\_INT](#). In DMA-controlled mode, it is recommended to wait for the interrupt [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#) when GDMA RX buffer is used, which means that data has been stored in the related memory. Other interrupts described in Section 20.9 are optional.

#### 20.5.9.5 Configuration of Slave Segmented Transfer in Half-Duplex

GDMA must be used in this mode. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB\_CLK) and APB clock (APB\_CLK).
3. Set [SPI\\_SLAVE\\_MODE](#) to enable slave mode.
4. Clear [SPI\\_DOUTDIN](#) to enable half-duplex communication.
5. Prepare data in registers [SPI\\_W0\\_REG](#) ~ [SPI\\_W15\\_REG](#), if needed.
6. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#) and [SPI\\_RX\\_AFIFO\\_RST](#) to reset these buffers.
7. Set bits [SPI\\_DMA\\_RX\\_ENA](#) and [SPI\\_DMA\\_TX\\_ENA](#). Clear the bit [SPI\\_RX\\_EOF\\_EN](#). Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 20.5.6 and Section 20.5.7.



8. Set [SPI\\_DMA\\_SLV\\_SEG\\_TRANS\\_EN](#) in [SPI\\_DMA\\_CONF\\_REG](#) to enable slave segmented transfer.
9. Set [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT\\_ENA](#) in [SPI\\_DMA\\_INT\\_ENA\\_REG](#) and wait for the interrupt [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#), which means that the segmented transfer has finished and data has been put into the related memory. Other interrupts described in Section 20.9 are optional.

When End\_SEG\_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI2, this slave segmented transfer is ended and the interrupt [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) is triggered.

#### 20.5.9.6 Configuration of Slave Segmented Transfer in Full-Duplex

GDMA must be used in this mode. In such transfer, the data is transferred from and to the GDMA buffer. The interrupt [GDMA\\_IN\\_SUC\\_EOF\\_CHn](#) \_INT is triggered when the transfer ends. The configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure AHB clock (AHB\_CLK) and APB clock (APB\_CLK).
3. Set [SPI\\_SLAVE\\_MODE](#) and [SPI\\_DOUTDIN](#), to enable full-duplex communication in slave mode.
4. Set [SPI\\_DMA\\_AFIFO\\_RST](#), [SPI\\_BUF\\_AFIFO\\_RST](#), and [SPI\\_RX\\_AFIFO\\_RST](#), to reset these buffers.
5. Set [SPI\\_DMA\\_TX\\_ENA](#)/[SPI\\_DMA\\_RX\\_ENA](#). Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 20.5.6 and Section 20.5.7.
6. Set the bit [SPI\\_RX\\_EOF\\_EN](#) in register [SPI\\_DMA\\_CONF\\_REG](#). Configure [SPI\\_MS\\_DATA\\_BITLEN](#)[17:0] in register [SPI\\_MS\\_DLEN\\_REG](#) to the byte length of the received DMA data.
7. Set [SPI\\_DMA\\_SLV\\_SEG\\_TRANS\\_EN](#) in [SPI\\_DMA\\_CONF\\_REG](#) to enable slave segmented transfer mode.
8. Set [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT\\_ENA](#) and wait for the interrupt [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#).

## 20.6 CS Setup Time and Hold Time Control

SPI bus CS ([SPI\\_CS](#)) setup time and hold time are very important to meet the timing requirements of various SPI devices (e.g. flash or PSRAM).

CS setup time is the time between the CS falling edge and the first latch edge of SPI bus CLK ([SPI\\_CLK](#)). The first latch edge for mode 0 and mode 3 is rising edge, and falling edge for mode 1 and mode 2.

CS hold time is the time between the last latch edge of [SPI\\_CLK](#) and the CS rising edge.

In slave mode, the CS setup time and hold time should be longer than  $0.5 \times T_{SPI\_CLK}$ , otherwise the SPI transfer may be incorrect.  $T_{SPI\_CLK}$  is one cycle of [SPI\\_CLK](#).

In master mode, set the CS setup time by specifying [SPI\\_CS\\_SETUP](#) in [SPI\\_USER\\_REG](#) and [SPI\\_CS\\_SETUP\\_TIME](#) in [SPI\\_USER1\\_REG](#):

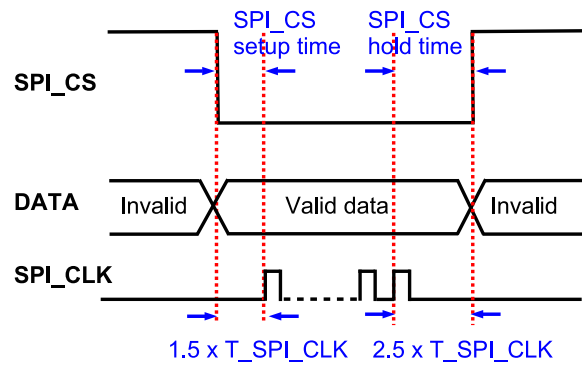
- If [SPI\\_CS\\_SETUP](#) is cleared, the SPI CS setup time is  $0.5 \times T_{SPI\_CLK}$ .
- If [SPI\\_CS\\_SETUP](#) is set, the SPI CS setup time is  $(\text{SPI\_CS\_SETUP\_TIME} + 1.5) \times T_{SPI\_CLK}$ .

Set the CS hold time by specifying [SPI\\_CS\\_HOLD](#) in [SPI\\_USER\\_REG](#) and [SPI\\_CS\\_HOLD\\_TIME](#) in [SPI\\_USER1\\_REG](#):

- If [SPI\\_CS\\_HOLD](#) is cleared, the SPI CS hold time is  $0.5 \times T_{SPI\_CLK}$ ;

- If `SPI_CS_HOLD` is set, the SPI CS hold time is  $(\text{SPI\_CS\_HOLD\_TIME} + 1.5) \times T_{\text{SPI\_CLK}}$ .

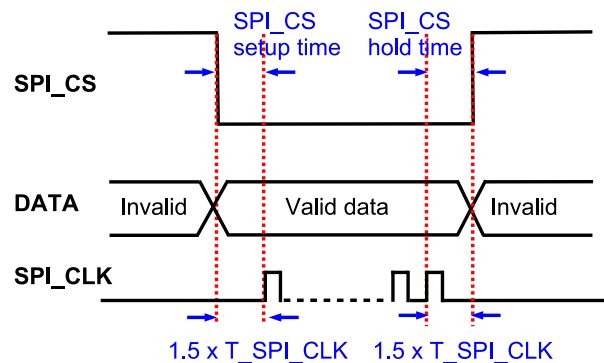
Figure 20.6-1 and Figure 20.6-2 show the recommended CS timing and register configuration to access external RAM and flash.



Register Configurations:

`SPI_CS_SETUP = 1`; `SPI_CS_SETUP_TIME = 0`;  
`SPI_CS_HOLD = 1`; `SPI_CS_HOLD_TIME = 1`.

Figure 20.6-1. Recommended CS Timing and Settings When Accessing External RAM



Register Configurations:

`SPI_CS_SETUP = 1`; `SPI_CS_SETUP_TIME = 0`;  
`SPI_CS_HOLD = 1`; `SPI_CS_HOLD_TIME = 0`.

Figure 20.6-2. Recommended CS Timing and Settings When Accessing Flash

## 20.7 GP-SPI2 Clock Control

GP-SPI2 has the following clocks:

- `clk_spi_mst`: module clock of GP-SPI2, derived from `PLL_CLK`. Used in GP-SPI2 master mode, to generate `SPI_CLK` signal for data transfer and for slaves.
- `SPI_CLK`: output clock in master mode.
- `AHB_CLK/APB_CLK`: clock for register configuration.

In master mode, the maximum output clock frequency of GP-SPI2 is  $f_{\text{clk\_spi\_mst}}$ . To have slower frequencies, the output clock frequency can be divided as follows:

$$f_{\text{SPI\_CLK}} = \frac{f_{\text{clk\_spi\_mst}}}{(\text{SPI\_CLKCNT\_N} + 1)(\text{SPI\_CLKDIV\_PRE} + 1)}$$

The divider is configured by `SPI_CLKCNT_N` and `SPI_CLKDIV_PRE` in register `SPI_CLOCK_REG`. When the bit `SPI_CLK_EQU_SYSCLK` in register `SPI_CLOCK_REG` is set to 1, the output clock frequency of GP-SPI2 will be  $f_{\text{clk\_spi\_mst}}$ . For other integral clock divisions, `SPI_CLK_EQU_SYSCLK` should be set to 0.

In slave mode, the supported input clock frequency ( $f_{\text{SPI\_CLK}}$ ) of GP-SPI2 is:

$$f_{\text{SPI\_CLK}} \leq 40\text{MHz}$$

## 20.7.1 Clock Phase and Polarity

SPI protocol has four clock modes, modes 0 ~ 3, see Figure 20.7-1 and Figure 20.7-2 (excerpted from SPI protocol):

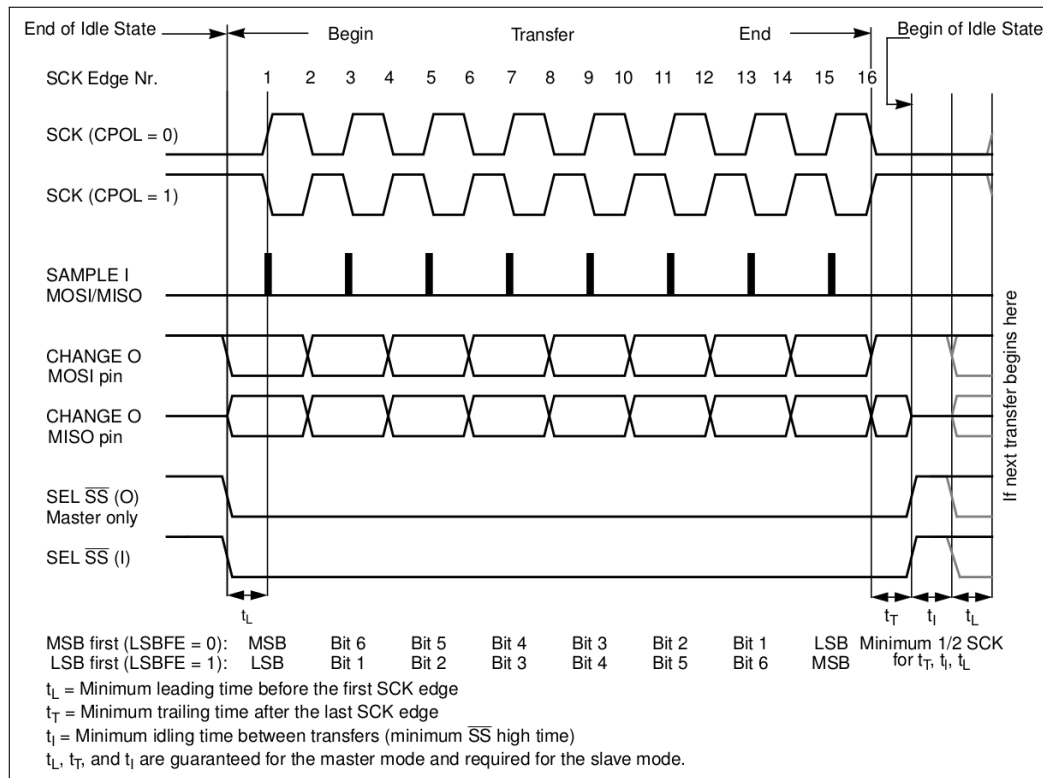


Figure 20.7-1. SPI Clock Mode 0 or 2

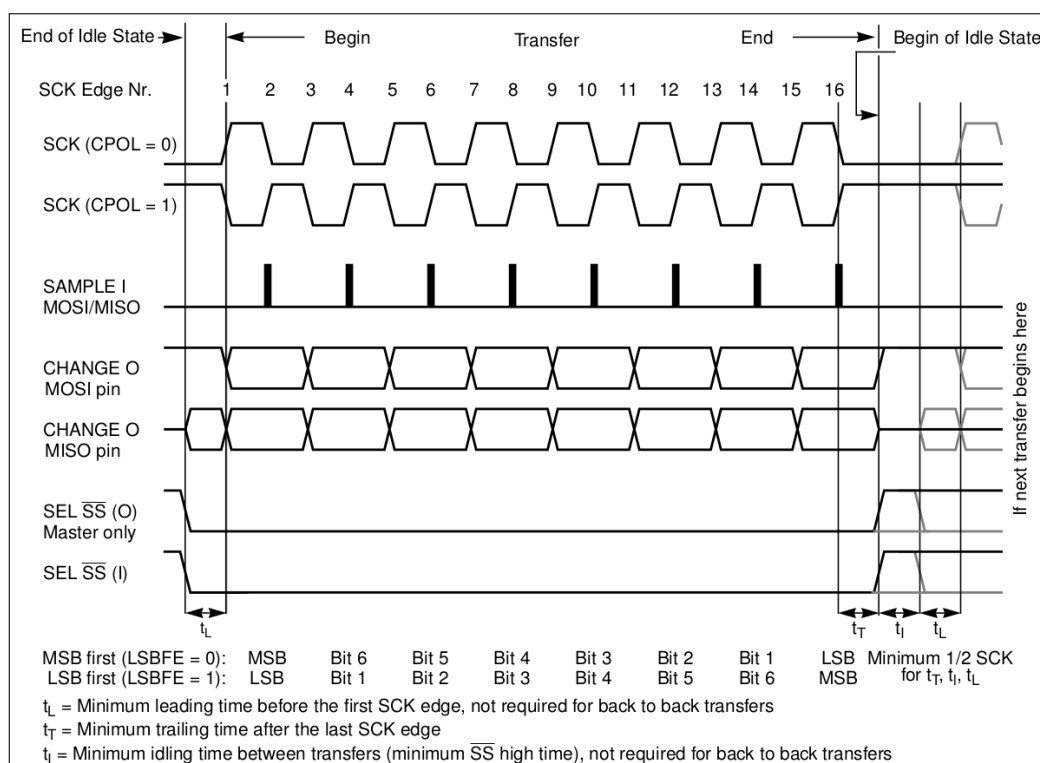


Figure 20.7-2. SPI Clock Mode 1 or 3

1. Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge. The first data is shifted out before the first negative edge of SCK.
2. Mode 1: CPOL = 0, CPHA = 1; SCK is 0 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge.
3. Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge. The first data is shifted out before the first positive edge of SCK.
4. Mode 3: CPOL = 1, CPHA = 1; SCK is 1 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge.

## 20.7.2 Clock Control in Master Mode

The four clock modes 0 ~ 3 are supported in GP-SPI2 master mode. The polarity and phase of GP-SPI2 clock are controlled by the bit `SPI_CLK_IDLE_EDGE` in register `SPI_MISC_REG` and the bit `SPI_CLK_OUT_EDGE` in register `SPI_USER_REG`. The register configuration for SPI clock modes 0 ~ 3 is provided in Table 20.7-1, and can be changed according to the path delay in the application.

Table 20.7-1. Clock Phase and Polarity Configuration in Master Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
<code>SPI_CLK_IDLE_EDGE</code>	0	0	1	1
<code>SPI_CLK_OUT_EDGE</code>	0	1	1	0

[SPI\\_CLK\\_MODE](#) is used to select the number of rising edges of SPI\_CLK, when SPI\_CS raises high, to be 0, 1, 2 or SPI\_CLK always on.

**Note:**

When [SPI\\_CLK\\_MODE](#) is configured to 1 or 2, the bit [SPI\\_CS\\_HOLD](#) must be set and the value of [SPI\\_CS\\_HOLD\\_TIME](#) should be larger than 1.

### 20.7.3 Clock Control in Slave Mode

GP-SPI2 slave mode also supports clock modes 0 ~ 3. The polarity and phase are configured by the bits [SPI\\_TSCK\\_I\\_EDGE](#) and [SPI\\_RSCK\\_I\\_EDGE](#) in register [SPI\\_USER\\_REG](#). The output edge of data is controlled by [SPI\\_CLK\\_MODE\\_13](#) in register [SPI\\_SLAVE\\_REG](#). The detailed register configuration is shown in Table 20.7-2:

Table 20.7-2. Clock Phase and Polarity Configuration in Slave Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
<a href="#">SPI_TSCK_I_EDGE</a>	0	1	1	0
<a href="#">SPI_RSCK_I_EDGE</a>	0	1	1	0
<a href="#">SPI_CLK_MODE_13</a>	0	1	0	1

## 20.8 GP-SPI2 Timing Compensation

The I/O lines are mapped via GPIO matrix or IO MUX. But there is no timing adjustment in IO MUX. The input data and output data can be delayed for 1 or 2 APB\_CLK cycles at the rising or falling edge in GPIO matrix. For detailed register configuration, see Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

In GP-SPI2 slave mode, if the bit [SPI\\_RSCK\\_DATA\\_OUT](#) in register [SPI\\_SLAVE\\_REG](#) is set to 1, the output data is sent at latch edge, which is half an SPI clock cycle earlier. This can be used for slave mode timing compensation.

## 20.9 Interrupts

### Interrupt Summary

GP-SPI2 provides an SPI interface interrupt SPI\_INT. When an SPI transfer ends, an interrupt is generated in GP-SPI2.

- [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#): triggered when GDMA RX FIFO length is shorter than the real transferred data length.
- [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#): triggered when GDMA TX FIFO length is shorter than the real transferred data length.
- [SPI\\_SLV\\_EX\\_QPI\\_INT](#): triggered when Ex\_QPI is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- [SPI\\_SLV\\_EN\\_QPI\\_INT](#): triggered when En\_QPI is received correctly in GP-SPI2 slave mode and the SPI transfer ends.

- SPI\_SLV\_CMD7\_INT: triggered when CMD7 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI\_SLV\_CMD8\_INT: triggered when CMD8 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI\_SLV\_CMD9\_INT: triggered when CMD9 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI\_SLV\_CMDA\_INT: triggered when CMDA is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI\_SLV\_RD\_DMA\_DONE\_INT: triggered at the end of Rd\_DMA transfer in slave mode.
- SPI\_SLV\_WR\_DMA\_DONE\_INT: triggered at the end of Wr\_DMA transfer in slave mode.
- SPI\_SLV\_RD\_BUF\_DONE\_INT: triggered at the end of Rd\_BUF transfer in slave mode.
- SPI\_SLV\_WR\_BUF\_DONE\_INT: triggered at the end of Wr\_BUF transfer in slave mode.
- SPI\_TRANS\_DONE\_INT: triggered at the end of SPI bus transfer in both master and slave modes.
- SPI\_DMA\_SEG\_TRANS\_DONE\_INT: triggered at the end of End\_SEG\_TRANS transfer in GP-SPI2 slave segmented transfer mode or at the end of configurable segmented transfer in master mode.
- SPI\_SEG\_MAGIC\_ERR\_INT: triggered when a Magic error occurs in CONF buffer during configurable segmented transfer in master mode.
- SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT: triggered by RX AFIFO write-full error in GP-SPI2 master mode.
- SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT: triggered by TX AFIFO read-empty error in GP-SPI2 master mode.
- SPI\_SLV\_CMD\_ERR\_INT: triggered when a received command value is not supported in GP-SPI2 slave mode.
- SPI\_APP2\_INT: used and triggered by software. Only used for user defined function.
- SPI\_APP1\_INT: used and triggered by software. Only used for user defined function.

### Interrupts Used in Master and Slave Modes

Table 20.9-1 and Table 20.9-2 show the interrupts used in GP-SPI2 master and slave modes. Set the interrupt enable bit SPI\_\*\_INT\_ENA in [SPI\\_DMA\\_INT\\_ENA\\_REG](#) and wait for the SPI\_INT interrupt. When the transfer ends, the related interrupt is triggered and should be cleared by software before the next transfer.

**Table 20.9-1. GP-SPI2 Master Mode Interrupts**

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	<a href="#">GDMA_IN_SUC_EOF_CH<sub>n</sub>_INT</a> <sup>1</sup>
		CPU	<a href="#">SPI_TRANS_DONE_INT</a> <sup>2</sup>
	Half-duplex MOSI Mode	DMA	<a href="#">SPI_TRANS_DONE_INT</a>
		CPU	<a href="#">SPI_TRANS_DONE_INT</a>
	Half-duplex MISO Mode	DMA	<a href="#">GDMA_IN_SUC_EOF_CH<sub>n</sub>_INT</a>
		CPU	<a href="#">SPI_TRANS_DONE_INT</a>

Continued on the next page

Table 20.9-1 – Continued from the previous page

Transfer Type	Communication Mode	Controlled by	Interrupt
Configurable Segmented Transfer	Full-duplex	DMA	<a href="#">SPI_DMA_SEG_TRANS_DONE_INT</a> <sup>3</sup>
		CPU	Not supported
	Half-duplex MOSI Mode	DMA	<a href="#">SPI_DMA_SEG_TRANS_DONE_INT</a>
		CPU	Not supported
	Half-duplex MISO	DMA	<a href="#">SPI_DMA_SEG_TRANS_DONE_INT</a>
		CPU	Not supported

<sup>1</sup> If [GDMA\\_IN\\_SUC\\_EOF\\_CHn\\_INT](#) is triggered, it means all the RX data of GP-SPI2 has been stored in the RX buffer, and the TX data has been transferred to the slave.

<sup>2</sup> [SPI\\_TRANS\\_DONE\\_INT](#) is triggered when CS is high, which indicates that master has completed the data exchange in [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#) with slave in this mode.

<sup>3</sup> If [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) is triggered, it means that the whole configurable segmented transfer (consisting of several segments) has finished, i.e. the RX data has been stored in the RX buffer completely and all the TX data has been sent out.

Table 20.9-2. GP-SPI2 Slave Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	<a href="#">GDMA_IN_SUC_EOF_CHn_INT</a> <sup>1</sup>
		CPU	<a href="#">SPI_TRANS_DONE_INT</a> <sup>2</sup>
	Half-duplex MOSI Mode	DMA (Wr_DMA)	<a href="#">GDMA_IN_SUC_EOF_CHn_INT</a> <sup>3</sup>
		CPU (Wr_BUF)	<a href="#">SPI_TRANS_DONE_INT</a> <sup>4</sup>
	Half-duplex MISO Mode	DMA (Rd_DMA)	<a href="#">SPI_TRANS_DONE_INT</a> <sup>5</sup>
		CPU (Rd_BUF)	<a href="#">SPI_TRANS_DONE_INT</a> <sup>6</sup>
Slave Segmented Transfer	Full-duplex	DMA	<a href="#">GDMA_IN_SUC_EOF_CHn_INT</a> <sup>7</sup>
		CPU	Not supported <sup>8</sup>
	Half-duplex MOSI Mode	DMA (Wr_DMA)	<a href="#">SPI_DMA_SEG_TRANS_DONE_INT</a> <sup>9</sup>
		CPU (Wr_BUF)	Not supported <sup>10</sup>
	Half-duplex MISO Mode	DMA (Rd_DMA)	<a href="#">SPI_DMA_SEG_TRANS_DONE_INT</a> <sup>11</sup>
		CPU (Rd_BUF)	Not supported <sup>12</sup>

- <sup>1</sup> If [GDMA\\_IN\\_SUC\\_EOF\\_CH \$n\$ \\_INT](#) is triggered, it means all the RX data has been stored in the RX buffer, and the TX data has been sent to the slave.
- <sup>2</sup> [SPI\\_TRANS\\_DONE\\_INT](#) is triggered when CS is high, which indicates that master has completed the data exchange in [SPI\\_WO\\_REG](#) ~ [SPI\\_W15\\_REG](#) with slave in this mode.
- <sup>3</sup> [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) just means that the transmission on the SPI bus is done, but can not ensure that all the push data has been stored in the RX buffer. For this reason, [GDMA\\_IN\\_SUC\\_EOF\\_CH \$n\$ \\_INT](#) is recommended.
- <sup>4</sup> Or wait for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#).
- <sup>5</sup> Or wait for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#).
- <sup>6</sup> Or wait for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#).
- <sup>7</sup> Slave should set the total read data byte length in [SPI\\_MS\\_DATA\\_BITLEN](#) before the transfer begins. Set [SPI\\_RX\\_EOF\\_EN](#) to 1 before the end of the interrupt program.
- <sup>8</sup> Master and slave should define a method to end the segmented transfer, such as via GPIO interrupt.
- <sup>9</sup> Master sends End\_SEG\_TRAN to end the segmented transfer or slave sets the total read data byte length in [SPI\\_MS\\_DATA\\_BITLEN](#) and waits for [GDMA\\_IN\\_SUC\\_EOF\\_CH \$n\$ \\_INT](#).
- <sup>10</sup> Half-duplex Wr\_BUF single transfer can be used in a slave segmented transfer.
- <sup>11</sup> Master sends End\_SEG\_TRAN to end the segmented transfer.
- <sup>12</sup> Half-duplex Rd\_BUF single transfer can be used in a slave segmented transfer.

## 20.10 Register Summary

The addresses in this section are relative to SPI base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>User-defined control registers</b>			
<a href="#">SPI_CMD_REG</a>	Command control register	0x0000	varies
<a href="#">SPI_ADDR_REG</a>	Address value register	0x0004	R/W
<a href="#">SPI_USER_REG</a>	SPI USER control register	0x0010	varies
<a href="#">SPI_USER1_REG</a>	SPI USER control register 1	0x0014	R/W
<a href="#">SPI_USER2_REG</a>	SPI USER control register 2	0x0018	R/W
<b>Control and configuration registers</b>			
<a href="#">SPI_CTRL_REG</a>	SPI control register	0x0008	R/W
<a href="#">SPI_MS_DLEN_REG</a>	SPI data bit length control register	0x001C	R/W
<a href="#">SPI_MISC_REG</a>	SPI MISC register	0x0020	R/W
<a href="#">SPI_DMA_CONF_REG</a>	SPI DMA control register	0x0030	varies
<a href="#">SPI_SLAVE_REG</a>	SPI slave control register	0x00E0	varies
<a href="#">SPI_SLAVE1_REG</a>	SPI slave control register 1	0x00E4	R/W/SS
<b>Clock control registers</b>			
<a href="#">SPI_CLOCK_REG</a>	SPI clock control register	0x000C	R/W
<a href="#">SPI_CLK_GATE_REG</a>	SPI module clock and register clock control	0x00E8	R/W
<b>Interrupt registers</b>			



Name	Description	Address	Access
<a href="#">SPI_DMA_INT_ENA_REG</a>	SPI DMA interrupt enable register	0x0034	R/W
<a href="#">SPI_DMA_INT_CLR_REG</a>	SPI DMA interrupt clear register	0x0038	WT
<a href="#">SPI_DMA_INT_RAW_REG</a>	SPI DMA interrupt raw register	0x003C	varies
<a href="#">SPI_DMA_INT_ST_REG</a>	SPI DMA interrupt status register	0x0040	RO
<a href="#">SPI_DMA_INT_SET_REG</a>	SPI DMA interrupt software set register	0x0044	RO
<b>CPU-controlled data buffer</b>			
<a href="#">SPI_W0_REG</a>	SPI CPU-controlled buffer 0	0x0098	R/W/SS
<a href="#">SPI_W1_REG</a>	SPI CPU-controlled buffer 1	0x009C	R/W/SS
<a href="#">SPI_W2_REG</a>	SPI CPU-controlled buffer 2	0x00A0	R/W/SS
<a href="#">SPI_W3_REG</a>	SPI CPU-controlled buffer 3	0x00A4	R/W/SS
<a href="#">SPI_W4_REG</a>	SPI CPU-controlled buffer 4	0x00A8	R/W/SS
<a href="#">SPI_W5_REG</a>	SPI CPU-controlled buffer 5	0x00AC	R/W/SS
<a href="#">SPI_W6_REG</a>	SPI CPU-controlled buffer 6	0x00B0	R/W/SS
<a href="#">SPI_W7_REG</a>	SPI CPU-controlled buffer 7	0x00B4	R/W/SS
<a href="#">SPI_W8_REG</a>	SPI CPU-controlled buffer 8	0x00B8	R/W/SS
<a href="#">SPI_W9_REG</a>	SPI CPU-controlled buffer 9	0x00BC	R/W/SS
<a href="#">SPI_W10_REG</a>	SPI CPU-controlled buffer 10	0x00C0	R/W/SS
<a href="#">SPI_W11_REG</a>	SPI CPU-controlled buffer 11	0x00C4	R/W/SS
<a href="#">SPI_W12_REG</a>	SPI CPU-controlled buffer 12	0x00C8	R/W/SS
<a href="#">SPI_W13_REG</a>	SPI CPU-controlled buffer 13	0x00CC	R/W/SS
<a href="#">SPI_W14_REG</a>	SPI CPU-controlled buffer 14	0x00D0	R/W/SS
<a href="#">SPI_W15_REG</a>	SPI CPU-controlled buffer 15	0x00D4	R/W/SS
<b>Version register</b>			
<a href="#">SPI_DATE_REG</a>	Version control register	0x00F0	R/W

## 20.11 Registers

The addresses in this section are relative to SPI base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

Register 20.1. SPI\_CMD\_REG (0x0000)

(reserved)							SPI_USR SPI_UPDATE		(reserved)							SPI_CONF_BITLEN																	
31							25	24	23	22							18	17															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															Reset			

**SPI\_CONF\_BITLEN** Define the SPI CLK cycles of SPI CONF state. Can be configured in CONF state. (R/W)

**SPI\_UPDATE** Set this bit to synchronize SPI registers from APB clock domain into SPI module clock domain. This bit is only used in SPI master mode. (WT)

**SPI\_USR** User-defined command enable. An SPI operation will be triggered when the bit is set. The bit will be cleared once the operation done. 1: enable; 0: disable. Can not be changed by CONF\_buf. (R/W/SC)

Register 20.2. SPI\_ADDR\_REG (0x0004)

SPI_USR_ADDR_VALUE																																
31																															0	
																															0	Reset

**SPI\_USR\_ADDR\_VALUE** Address to slave. Can be configured in CONF state. (R/W)

**Register 20.3. SPI\_USER\_REG (0x0010)**

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 (reserved) SPI_USR_CONF_NXT (reserved) SPI_FWRITE_QUAD SPI_FWRITE_DUAL (reserved) SPI_CLK_OUT_EDGE SPI_RSCK_I_EDGE SPI_CS_SETUP SPI_CS_HOLD (reserved) SPI_QPI_MODE (reserved) SPI_DOUTDIN																		
31	30	29	28	27	26	25	24	23								18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	Reset

**SPI\_DOUTDIN** Set the bit to enable full duplex communication. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

**SPI\_QPI\_MODE** 1: Enable QPI mode. 0: Disable QPI mode. This configuration is applicable when the SPI controller works as master or slave. Can be configured in CONF state. (R/W/SS/SC)

**SPI\_TSCK\_I\_EDGE** In slave mode, this bit can be used to change the polarity of TSCK. 0: TSCK = SPI\_CLK\_I. 1: TSCK = !SPI\_CLK\_I. (R/W)

**SPI\_CS\_HOLD** Keep SPI CS low when SPI is in DONE state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

**SPI\_CS\_SETUP** Enable SPI CS when SPI is in prepare (PREP) state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

**SPI\_RSCK\_I\_EDGE** In slave mode, this bit can be used to change the polarity of RSCK. 0: RSCK = !SPI\_CLK\_I. 1: RSCK = SPI\_CLK\_I. (R/W)

**SPI\_CLK\_OUT\_EDGE** This bit together with [SPI\\_CLK\\_IDLE\\_EDGE](#) is used to control SPI clock mode. Can be configured in CONF state. For more information, see Section [20.7.2](#). (R/W)

**SPI\_FWRITE\_DUAL** In write operations, read-data phase is in 2-bit mode. Can be configured in CONF state. (R/W)

**SPI\_FWRITE\_QUAD** In write operations, read-data phase is in 4-bit mode. Can be configured in CONF state. (R/W)

**SPI\_USR\_CONF\_NXT** Enable the CONF state for the next transaction (segment) in a configurable segmented transfer. Can be configured in CONF state. (R/W)

- If this bit is set, it means this configurable segmented transfer will continue its next transaction (segment).
- If this bit is cleared, it means this transfer will end after the current transaction (segment) is finished. Or this is not a configurable segmented transfer.

**SPI\_SIO** Set the bit to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

**SPI\_USR\_MISO\_HIGHPART** In read-data phase, only access to high-part of the buffers: [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#). 1: enable; 0: disable. Can be configured in CONF state. (R/W)

Continued on the next page...

**Register 20.3. SPI\_USER\_REG (0x0010)**

Continued from the previous page...

**SPI\_USR\_MOSI\_HIGHPART** In write-data phase, only access to high-part of the buffers: [SPI\\_W8\\_REG](#) ~ [SPI\\_W15\\_REG](#). 1: enable; 0: disable. Can be configured in CONF state. (R/W)

**SPI\_USR\_DUMMY\_IDLE** If this bit is set, SPI clock is disabled in DUMMY state. Can be configured in CONF state. (R/W)

**SPI\_USR\_MOSI** Set this bit to enable the write-data (DOUT) state of an operation. Can be configured in CONF state. (R/W)

**SPI\_USR\_MISO** Set this bit to enable the read-data (DIN) state of an operation. Can be configured in CONF state. (R/W)

**SPI\_USR\_DUMMY** Set this bit to enable the DUMMY state of an operation. Can be configured in CONF state. (R/W)

**SPI\_USR\_ADDR** Set this bit to enable the address (ADDR) state of an operation. Can be configured in CONF state. (R/W)

**SPI\_USR\_COMMAND** Set this bit to enable the command (CMD) state of an operation. Can be configured in CONF state. (R/W)

### Register 20.4. SPI\_USER1\_REG (0x0014)

SPI_USR_ADDR_BITLEN								SPI_CS_HOLD_TIME								SPI_CS_SETUP_TIME				SPI_MST_WFULL_ERR_END_EN								(reserved)								SPI_USR_DUMMY_CYCLELEN							
31	27	26	22	21	17	16	15	8	7	0																																	
23								0x1								0				1	0	0	0	0	0	0	0	0	0	7								Reset					

**SPI\_USR\_DUMMY\_CYCLELEN** The length of DUMMY state, in unit of SPI\_CLK cycles. This value is (the expected cycle number - 1). Can be configured in CONF state. (R/W)

**SPI\_MST\_WFULL\_ERR\_END\_EN** 1: SPI transfer is ended when SPI RX AFIFO wfull error occurs in GP-SPI2 master full-/half-duplex modes. 0: SPI transfer is not ended when SPI RX AFIFO wfull error occurs in GP-SPI2 master full-/half-duplex modes. (R/W)

**SPI\_CS\_SETUP\_TIME** The length of prepare (PREP) state, in unit of SPI\_CLK cycles. This value is equal to the expected cycles -1. This field is used together with **SPI\_CS\_SETUP**. Can be configured in CONF state. (R/W)

**SPI\_CS\_HOLD\_TIME** Delay cycles of CS pin, in units of SPI\_CLK cycles. This field is used together with [SPI\\_CS\\_HOLD](#). Can be configured in CONF state. (R/W)

**SPI\_USR\_ADDR\_BITLEN** The bit length in address state. This value is (expected bit number - 1).  
Can be configured in CONF state. (R/W)

### Register 20.5. SPI\_USER2\_REG (0x0018)

[illegible]

**SPI\_USR\_COMMAND\_VALUE** The value of command. Can be configured in CONF state. (R/W)

**SPI\_MST\_EMPTY\_ERR\_END\_EN** 1: SPI transfer is ended when SPI TX AFIFO read empty error occurs in GP-SPI2 master full-/half-duplex modes. 0: SPI transfer is not ended when SPI TX AFIFO read empty error occurs in GP-SPI2 master full-/half-duplex modes. (R/W)

**SPI\_USR\_COMMAND\_BITLEN** The bit length of command state. This value is (expected bit number - 1). Can be configured in CONF state. (R/W)



Register 20.7. SPI\_MS\_DLEN\_REG (0x001C)

(reserved)														SPI_MS_DATA_BITLEN																	
3118														170																	
000000000000000														0																	Reset

**SPI\_MS\_DATA\_BITLEN** The value of this field is the configured SPI transmission data bit length in master mode DMA controlled transfer or CPU controlled transfer. The value is also the configured bit length in slave mode DMA RX controlled transfer. The register value shall be (bit\_num - 1). Can be configured in CONF state. (R/W)

### Register 20.8. SPI\_MISC\_REG (0x0020)

(reserved)								$\overline{\text{SPI\_CS\_KEEP\_ACTIVE}}$								(reserved)								$\overline{\text{SPI\_OK\_IDLE\_EDGE}}$																$\text{SPI\_SLAVE\_CS\_POL}$								(reserved)								$\text{SPI\_MASTER\_CS\_POL}$								$\text{SPI\_CK\_DIS}$								$\text{SPI\_CS5\_DIS}$								$\text{SPI\_CS4\_DIS}$								$\text{SPI\_CS3\_DIS}$								$\text{SPI\_CS2\_DIS}$								$\text{SPI\_CS1\_DIS}$								$\text{SPI\_CS0\_DIS}$							
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	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	1	1	1	1	1	1	0																																																																																						

**SPI\_CS0\_DIS** SPI CS0 pin enable bit. 1: disable CS0, 0: SPI\_CS0 signal is from/to CS0 pin. Can be configured in CONF state. (R/W)

**SPI\_CS1\_DIS** SPI CS1 pin enable bit. 1: disable CS1, 0: SPI\_CS1 signal is from/to CS1 pin. Can be configured in CONF state. (R/W)

**SPI\_CS2\_DIS** SPI CS2 pin enable bit. 1: disable CS2, 0: SPI\_CS2 signal is from/to CS2 pin. Can be configured in CONF state. (R/W)

**SPI\_CS3\_DIS** SPI CS3 pin enable bit. 1: disable CS3, 0: SPI\_CS3 signal is from/to CS3 pin. Can be configured in CONF state. (R/W)

**SPI\_CS4\_DIS** SPI CS4 pin enable bit. 1: disable CS4, 0: SPI\_CS4 signal is from/to CS4 pin. Can be configured in CONF state. (R/W)

**SPI\_CS5\_DIS** SPI CS5 pin enable bit. 1: disable CS5, 0: SPI\_CS5 signal is from/to CS5 pin. Can be configured in CONF state. (R/W)

**SPI\_CK\_DIS** 1: disable SPI\_CLK output. 0: enable SPI\_CLK output. Can be configured in CONF state. (R/W)

**SPI\_MASTER\_CS\_POL** SPI\_MASTER\_CS\_POL[*i*] configures the polarity of SPI CS*i* (*i* is from 0 ~ 5) line in master mode. 0: CS*i* is low active. 1: CS*i* is high active. Can be configured in CONF state. (R/W)

**SPI\_SLAVE\_CS\_POL** Configure SPI slave input CS polarity. 1: invert. 0: not change. Can be configured in CONF state. (R/W)

**SPI\_CK\_IDLE\_EDGE** 1: SPI\_CLK line is high when GP-SPI2 is in idle. 0: SPI\_CLK line is low when GP-SPI2 is in idle. Can be configured in CONF state. (R/W)

**SPI\_CS\_KEEP\_ACTIVE** SPI CS line keeps low when the bit is set. Can be configured in CONF state.  
(R/W)



Register 20.9. SPI\_DMA\_CONF\_REG (0x0030)

SPI_DMA_AFIFO_RST					SPI_DMA_RX_ENA					(reserved)					SPI_RX_EOF_EN					SPI_SLV_TX_SEG_TRANS_CLR_EN					(reserved)					SPI_DMA_INFIFO_FULL				
SPI_BUF_AFIFO_RST															SPI_SLV_RX_SEG_TRANS_CLR_EN										SPI_DMA_OUTFIFO_EMPTY									
SPI_RX_AFIFO_RST																																		
SPI_DMA_TX_ENA																																		
																				</														

Reset

**SPI\_DMA\_OUTFIFO\_EMPTY** Records the status of DMA TX FIFO. 1: DMA TX FIFO is not ready for sending data. 0: DMA TX FIFO is ready for sending data. (RO)

**SPI\_DMA\_INFIFO\_FULL** Records the status of DMA RX FIFO. 1: DMA RX FIFO is not ready for receiving data. 0: DMA RX FIFO is ready for receiving data. (RO)

**SPI\_DMA\_SLV\_SEG\_TRANS\_EN** 1: enable DMA-controlled segmented transfer in slave half-duplex mode. 0: disable. (R/W)

**SPI\_SLV\_RX\_SEG\_TRANS\_CLR\_EN** In slave segmented transfer, if the size of the DMA RX buffer is smaller than the size of the received data, 1: the data in all the following Wr\_DMA transactions will not be received; 0: the data in this Wr\_DMA transaction will not be received, but in the following transactions, (R/W)

- if the size of DMA RX buffer is not 0, the data in following Wr\_DMA transactions will be received.
- if the size of DMA RX buffer is 0, the data in following Wr\_DMA transactions will not be received.

**SPI\_SLV\_TX\_SEG\_TRANS\_CLR\_EN** In slave segmented transfer, if the size of the DMA TX buffer is smaller than the size of the transmitted data, (R/W)

- 1: the data in the following transactions will not be updated, i.e. the old data is transmitted repeatedly.
- 0: the data in this transaction will not be updated. But in the following transactions,
  - if new data is filled in DMA TX FIFO, new data will be transmitted.
  - if no new data is filled in DMA TX FIFO, no new data will be transmitted.

**SPI\_RX\_EOF\_EN** 1: In a DAM-controlled transfer, if the bit number of transferred data is equal to (SPI\_MS\_DATA\_BITLEN + 1), then GDMA\_IN\_SUC\_EOF\_CH<sub>n</sub>\_INT\_RAW will be set by hardware. 0: GDMA\_IN\_SUC\_EOF\_CH<sub>n</sub>\_INT\_RAW is set by SPI\_TRANS\_DONE\_INT event in a single transfer, or by an SPI\_DMA\_SEG\_TRANS\_DONE\_INT event in a segmented transfer. (R/W)

**SPI\_DMA\_RX\_ENA** Set this bit to enable SPI DMA controlled receive data mode. (R/W)

**SPI\_DMA\_TX\_ENA** Set this bit to enable SPI DMA controlled send data mode. (R/W)

Continued on the next page...

**Register 20.9. SPI\_DMA\_CONF\_REG (0x0030)**

Continued from the previous page...

**SPI\_RX\_AFIFO\_RST** Set this bit to reset spi\_rx\_afifo as shown in Figure 20.5-3 and in Figure 20.5-4. spi\_rx\_afifo is used to receive data in SPI master and slave transfer. (WT)

**SPI\_BUF\_AFIFO\_RST** Set this bit to reset buf\_tx\_afifo as shown in Figure 20.5-3 and in Figure 20.5-4. buf\_tx\_afifo is used to send data out in CPU-controlled master and slave transfer. (WT)

**SPI\_DMA\_AFIFO\_RST** Set this bit to reset dma\_tx\_afifo as shown in Figure 20.5-3 and in Figure 20.5-4. dma\_tx\_afifo is used to send data out in DMA-controlled slave transfer. (WT)

### Register 20.10. SPI\_SLAVE\_REG (0x00E0)

[illegible]

**SPI\_CLK\_MODE** SPI clock mode control bits. Can be configured in CONF state. (R/W)

- 0: SPI clock is off when CS becomes inactive.
- 1: SPI clock is delayed one cycle after CS becomes inactive.
- 2: SPI clock is delayed two cycles after CS becomes inactive.
- 3: SPI clock is always on.

**SPI\_CLK\_MODE\_13** Configure clock mode. (R/W)

- 1: support SPI clock mode 1 or 3. See Table 20.7-2.
- 0: support SPI clock mode 0 or 2. See Table 20.7-2.

**SPI\_RSCK\_DATA\_OUT** Save half a cycle when TSCK is the same as RSCK. 1: output data at RSCK rising edge. 0: output data at TSCK rising edge. (R/W)

**SPI\_SLV\_RDDMA\_BITLEN\_EN** If this bit is set, **SPI\_SLV\_DATA\_BITLEN** is used to store the data bit length of Rd\_DMA transfer (R/W)

**SPI\_SLV\_WRDMA\_BITLEN\_EN** If this bit is set, **SPI\_SLV\_DATA\_BITLEN** is used to store the data bit length of Wr\_DMA transfer. (R/W)

**SPI\_SLV\_RDBUF\_BITLEN\_EN** If this bit is set, **SPI\_SLV\_DATA\_BITLEN** is used to store data bit length of Rd\_BUF transfer. (R/W)

**SPI\_SLV\_WRBUF\_BITLEN\_EN** If this bit is set, **SPI\_SLV\_DATA\_BITLEN** is used to store data bit length of Wr\_BUF transfer. (R/W)

**SPI\_DMA\_SEG\_MAGIC\_VALUE** Configure the magic value of BM table in DMA-controlled configurable segmented transfer. (R/W)

**SPI\_SLAVE\_MODE** Set SPI work mode. 1: slave mode. 0: master mode. (R/W)

**SPI\_SOFT\_RESET** Software reset enable bit. If this bit is set, the SPI clock line, CS line, and data line are reset. Can be configured in CONF state. (WT)

**SPI\_USR\_CONF** 1: enable the CONF state of current DMA-controlled configurable segmented transfer, which means a configurable segmented transfer is started. 0: the current transfer is not a configurable segmented transfer. (R/W)

Register 20.11. SPI\_SLAVE1\_REG (0x00E4)

SPI_SLV_LAST_ADDR									
SPI_SLV_LAST_COMMAND									
SPI_SLV_DATA_BITLEN									
31	26	25	18	17	0				
0		0		0					
Reset									

**SPI\_SLV\_DATA\_BITLEN** Configure the transferred data bit length in SPI slave full-/half-duplex modes. (R/W/SS)

**SPI\_SLV\_LAST\_COMMAND** In slave mode, it is the value of command. (R/W/SS)

**SPI\_SLV\_LAST\_ADDR** In slave mode, it is the value of address. (R/W/SS)

Register 20.12. SPI\_CLOCK\_REG (0x000C)

SPI_CLK_EQU_SYSCLK									
(reserved)									
SPI_CLKDIV_PRE									
SPI_CLKCNT_N									
SPI_CLKCNT_H									
SPI_CLKCNT_L									
31	30	22	21	18	17	12	11	6	5
1	0	0	0	0	0	0	0	0	0
0									
0x3									
0x1									
0x3									
Reset									

**SPI\_CLKCNT\_L** In master mode, this field must be equal to SPI\_CLKCNT\_N. In slave mode, it must be 0. Can be configured in CONF state. (R/W)

**SPI\_CLKCNT\_H** In master mode, this field is used to configure the duty cycle of SPI\_CLK (high level). It's recommended to configure this value to  $\text{floor}((\text{SPI\_CLKCNT\_N} + 1)/2 - 1)$ .  $\text{floor}()$  here is to round a number down, e.g.,  $\text{floor}(2.2) = 2$ . In slave mode, it must be 0. Can be configured in CONF state. (R/W)

**SPI\_CLKCNT\_N** In master mode, this is the divider of SPI\_CLK. So SPI\_CLK frequency is  $f_{\text{clk\_spi\_mst}}/(\text{SPI\_CLKDIV\_PRE} + 1)/(\text{SPI\_CLKCNT\_N} + 1)$ . Can be configured in CONF state. (R/W)

**SPI\_CLKDIV\_PRE** In master mode, this is the pre-divider of SPI\_CLK. Can be configured in CONF state. (R/W)

**SPI\_CLK\_EQU\_SYSCLK** In master mode, 1: SPI\_CLK is equal to clk\_spi\_mst. 0: SPI\_CLK is divided from clk\_spi\_mst. Can be configured in CONF state. (R/W)

Register 20.13. SPI\_CLK\_GATE\_REG (0x00E8)

(reserved)																																SPI_MST_CLK_SEL SPI_MST_CLK_ACTIVE SPI_CLK_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- SPI\_CLK\_EN Set this bit to enable clock gate. (R/W)
- SPI\_MST\_CLK\_ACTIVE Set this bit to power on the SPI module clock. (R/W)
- SPI\_MST\_CLK\_SEL This bit is used to select SPI module clock source in master mode. 1: PLL\_F80M\_CLK. 0: XTAL\_CLK. (R/W)

Register 20.14. SPI\_DMA\_INT\_ENA\_REG (0x0034)

(reserved)												SPI_APP1_INT_ENA SPI_APP2_INT_ENA SPI_MST_TX_AFIFO_EMPTY_ERR_INT_ENA SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA SPI_SLV_CMD_ERR_INT_ENA (reserved) SPI_SEG_MAGIC_ERR_INT_ENA SPI_DMA_SEG_TRANS_DONE_INT_ENA SPI_TRANS_DONE_INT_ENA SPI_SLV_WR_BUF_DONE_INT_ENA SPI_SLV_RD_BUF_DONE_INT_ENA SPI_SLV_WR_DMA_DONE_INT_ENA SPI_SLV_RD_DMA_DONE_INT_ENA SPI_SLV_CMD9_INT_ENA SPI_SLV_CMD8_INT_ENA SPI_SLV_CMD7_INT_ENA SPI_SLV_EX_QPI_INT_ENA SPI_SLV_EN_QPI_INT_ENA SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA SPI_DMA_INFIFO_FULL_ERR_INT_ENA																					
31												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	Reset		

**SPI\_DMA\_INFIFO\_FULL\_ERR\_INT\_ENA** The enable bit for [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#) interrupt. (R/W)

**SPI\_DMA\_OUTFIFO\_EMPTY\_ERR\_INT\_ENA** The enable bit for [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_EX\_QPI\_INT\_ENA** The enable bit for [SPI\\_SLV\\_EX\\_QPI\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_EN\_QPI\_INT\_ENA** The enable bit for [SPI\\_SLV\\_EN\\_QPI\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_CMD7\_INT\_ENA** The enable bit for [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_CMD8\_INT\_ENA** The enable bit for [SPI\\_SLV\\_CMD8\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_CMD9\_INT\_ENA** The enable bit for [SPI\\_SLV\\_CMD9\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_CMDA\_INT\_ENA** The enable bit for [SPI\\_SLV\\_CMDA\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_RD\_DMA\_DONE\_INT\_ENA** The enable bit for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_WR\_DMA\_DONE\_INT\_ENA** The enable bit for [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_RD\_BUF\_DONE\_INT\_ENA** The enable bit for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_SLV\_WR\_BUF\_DONE\_INT\_ENA** The enable bit for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_TRANS\_DONE\_INT\_ENA** The enable bit for [SPI\\_TRANS\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_DMA\_SEG\_TRANS\_DONE\_INT\_ENA** The enable bit for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. (R/W)

**SPI\_SEG\_MAGIC\_ERR\_INT\_ENA** The enable bit for [SPI\\_SEG\\_MAGIC\\_ERR\\_INT](#) interrupt. (R/W)

Continued on the next page...

**Register 20.14. SPI\_DMA\_INT\_ENA\_REG (0x0034)**

Continued from the previous page...

**SPI\_SLV\_CMD\_ERR\_INT\_ENA** The enable bit for [SPI\\_SLV\\_CMD\\_ERR\\_INT](#) interrupt. (R/W)

**SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT\_ENA** The enable bit for [SPI\\_MST\\_RX\\_AFIFO\\_WFULL\\_ERR\\_INT](#) interrupt. (R/W)

**SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT\_ENA** The enable bit for [SPI\\_MST\\_TX\\_AFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (R/W)

**SPI\_APP2\_INT\_ENA** The enable bit for [SPI\\_APP2\\_INT](#) interrupt. (R/W)

**SPI\_APP1\_INT\_ENA** The enable bit for [SPI\\_APP1\\_INT](#) interrupt. (R/W)

## Register 20.15. SPI\_DMA\_INT\_CLR\_REG (0x0038)

(reserved)												SPI_APP1_INT_CLR SPI_APP2_INT_CLR SPI_MST_TX_INT_CLR SPI_MST_RX_AFIFO_EMPTY_ERR_INT_CLR SPI_SLV_CMD_ERR_INT_CLR (reserved) SPI_SEG_MAGIC_ERR_INT_CLR SPI_DMA_SEG_TRANS_DONE_INT_CLR SPI_TRANS_DONE_INT_CLR SPI_SLV_WR_BUF_DONE_INT_CLR SPI_SLV_RD_BUF_DONE_INT_CLR SPI_SLV_WR_DMA_DONE_INT_CLR SPI_SLV_RD_DMA_DONE_INT_CLR SPI_SLV_CMD7_INT_CLR SPI_SLV_CMD8_INT_CLR SPI_SLV_CMD9_INT_CLR SPI_SLV_CMDA_INT_CLR SPI_SLV_EN_QPI_INT_CLR SPI_SLV_EX_QPI_INT_CLR SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR SPI_DMA_INFIFO_FULL_ERR_INT_CLR																				
31											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	Reset	

**SPI\_DMA\_INFIFO\_FULL\_ERR\_INT\_CLR** The clear bit for [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_DMA\_OUTFIFO\_EMPTY\_ERR\_INT\_CLR** The clear bit for [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_SLV\_EX\_QPI\_INT\_CLR** The clear bit for [SPI\\_SLV\\_EX\\_QPI\\_INT](#) interrupt. (WT)

**SPI\_SLV\_EN\_QPI\_INT\_CLR** The clear bit for [SPI\\_SLV\\_EN\\_QPI\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD7\_INT\_CLR** The clear bit for [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD8\_INT\_CLR** The clear bit for [SPI\\_SLV\\_CMD8\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD9\_INT\_CLR** The clear bit for [SPI\\_SLV\\_CMD9\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMDA\_INT\_CLR** The clear bit for [SPI\\_SLV\\_CMDA\\_INT](#) interrupt. (WT)

**SPI\_SLV\_RD\_DMA\_DONE\_INT\_CLR** The clear bit for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_WR\_DMA\_DONE\_INT\_CLR** The clear bit for [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_RD\_BUF\_DONE\_INT\_CLR** The clear bit for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_WR\_BUF\_DONE\_INT\_CLR** The clear bit for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_TRANS\_DONE\_INT\_CLR** The clear bit for [SPI\\_TRANS\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_DMA\_SEG\_TRANS\_DONE\_INT\_CLR** The clear bit for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SEG\_MAGIC\_ERR\_INT\_CLR** The clear bit for [SPI\\_SEG\\_MAGIC\\_ERR\\_INT](#) interrupt. (WT)

Continued on the next page...



**Register 20.15. SPI\_DMA\_INT\_CLR\_REG (0x0038)**

Continued from the previous page...

**SPI\_SLV\_CMD\_ERR\_INT\_CLR** The clear bit for [SPI\\_SLV\\_CMD\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT\_CLR** The clear bit for [SPI\\_MST\\_RX\\_AFIFO\\_WFULL\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT\_CLR** The clear bit for [SPI\\_MST\\_TX\\_AFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_APP2\_INT\_CLR** The clear bit for [SPI\\_APP2\\_INT](#) interrupt. (WT)

**SPI\_APP1\_INT\_CLR** The clear bit for [SPI\\_APP1\\_INT](#) interrupt. (WT)

Register 20.16. SPI\_DMA\_INT\_RAW\_REG (0x003C)

(reserved)												SPI_APP1_INT_RAW SPI_APP2_INT_RAW SPI_MST_TX_AFIFO_EMPTY_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_SLV_CMD_ERR_INT_RAW (reserved) SPI_SEG_MAGIC_ERR_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD7_INT_RAW SPI_SLV_EN_QPI_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT_RAW																				
31											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	Reset	

**SPI\_DMA\_INFIFO\_FULL\_ERR\_INT\_RAW** The raw bit for [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_DMA\_OUTFIFO\_EMPTY\_ERR\_INT\_RAW** The raw bit for [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_EX\_QPI\_INT\_RAW** The raw bit for [SPI\\_SLV\\_EX\\_QPI\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_EN\_QPI\_INT\_RAW** The raw bit for [SPI\\_SLV\\_EN\\_QPI\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_CMD7\_INT\_RAW** The raw bit for [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_CMD8\_INT\_RAW** The raw bit for [SPI\\_SLV\\_CMD8\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_CMD9\_INT\_RAW** The raw bit for [SPI\\_SLV\\_CMD9\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_CMDA\_INT\_RAW** The raw bit for [SPI\\_SLV\\_CMDA\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_RD\_DMA\_DONE\_INT\_RAW** The raw bit for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_WR\_DMA\_DONE\_INT\_RAW** The raw bit for [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_RD\_BUF\_DONE\_INT\_RAW** The raw bit for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_WR\_BUF\_DONE\_INT\_RAW** The raw bit for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_TRANS\_DONE\_INT\_RAW** The raw bit for [SPI\\_TRANS\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

Continued on the next page...

**Register 20.16. SPI\_DMA\_INT\_RAW\_REG (0x003C)**

Continued from the previous page...

**SPI\_DMA\_SEG\_TRANS\_DONE\_INT\_RAW** The raw bit for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SEG\_MAGIC\_ERR\_INT\_RAW** The raw bit for [SPI\\_SEG\\_MAGIC\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_SLV\_CMD\_ERR\_INT\_RAW** The raw bit for [SPI\\_SLV\\_CMD\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT\_RAW** The raw bit for [SPI\\_MST\\_RX\\_AFIFO\\_WFULL\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT\_RAW** The raw bit for [SPI\\_MST\\_TX\\_AFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (R/W/WTC/SS)

**SPI\_APP2\_INT\_RAW** The raw bit for [SPI\\_APP2\\_INT](#) interrupt. The value is only controlled by the application. (R/W/WTC)

**SPI\_APP1\_INT\_RAW** The raw bit for [SPI\\_APP1\\_INT](#) interrupt. The value is only controlled by the application. (R/W/WTC)

Register 20.17. SPI\_DMA\_INT\_ST\_REG (0x0040)

(reserved)												SPI_APP1_INT_ST SPI_APP2_INT_ST SPI_MST_TX_AFIFO_EMPTY_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_SLV_CMD_ERR_INT_ST (reserved) SPI_SEG_MAGIC_ERR_INT_ST SPI_DMA_SEG_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_WR_DMA_DONE_INT_ST SPI_SLV_RD_DMA_DONE_INT_ST SPI_SLV_CMD9_INT_ST SPI_SLV_CMD8_INT_ST SPI_SLV_CMD7_INT_ST SPI_SLV_EN_QPI_INT_ST SPI_SLV_EX_QPI_INT_ST SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST SPI_DMA_INFIFO_FULL_ERR_INT_ST											
31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SPI\_DMA\_INFIFO\_FULL\_ERR\_INT\_ST** The status bit for [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#) interrupt. (RO)

**SPI\_DMA\_OUTFIFO\_EMPTY\_ERR\_INT\_ST** The status bit for [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (RO)

**SPI\_SLV\_EX\_QPI\_INT\_ST** The status bit for [SPI\\_SLV\\_EX\\_QPI\\_INT](#) interrupt. (RO)

**SPI\_SLV\_EN\_QPI\_INT\_ST** The status bit for [SPI\\_SLV\\_EN\\_QPI\\_INT](#) interrupt. (RO)

**SPI\_SLV\_CMD7\_INT\_ST** The status bit for [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. (RO)

**SPI\_SLV\_CMD8\_INT\_ST** The status bit for [SPI\\_SLV\\_CMD8\\_INT](#) interrupt. (RO)

**SPI\_SLV\_CMD9\_INT\_ST** The status bit for [SPI\\_SLV\\_CMD9\\_INT](#) interrupt. (RO)

**SPI\_SLV\_CMDA\_INT\_ST** The status bit for [SPI\\_SLV\\_CMDA\\_INT](#) interrupt. (RO)

**SPI\_SLV\_RD\_DMA\_DONE\_INT\_ST** The status bit for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_SLV\_WR\_DMA\_DONE\_INT\_ST** The status bit for [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_SLV\_RD\_BUF\_DONE\_INT\_ST** The status bit for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_SLV\_WR\_BUF\_DONE\_INT\_ST** The status bit for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_TRANS\_DONE\_INT\_ST** The status bit for [SPI\\_TRANS\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_DMA\_SEG\_TRANS\_DONE\_INT\_ST** The status bit for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. (RO)

**SPI\_SEG\_MAGIC\_ERR\_INT\_ST** The status bit for [SPI\\_SEG\\_MAGIC\\_ERR\\_INT](#) interrupt. (RO)

**SPI\_SLV\_CMD\_ERR\_INT\_ST** The status bit for [SPI\\_SLV\\_CMD\\_ERR\\_INT](#) interrupt. (RO)

Continued on the next page...

## Register 20.17. SPI\_DMA\_INT\_ST\_REG (0x0040)

Continued from the previous page...

**SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT\_ST** The status bit for [SPI\\_MST\\_RX\\_AFIFO\\_WFULL\\_ERR\\_INT](#) interrupt. (RO)

**SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT\_ST** The status bit for [SPI\\_MST\\_TX\\_AFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (RO)

**SPI\_APP2\_INT\_ST** The status bit for [SPI\\_APP2\\_INT](#) interrupt. (RO)

**SPI\_APP1\_INT\_ST** The status bit for [SPI\\_APP1\\_INT](#) interrupt. (RO)

## Register 20.18. SPI\_DMA\_INT\_SET\_REG (0x0044)

(reserved)												<a href="#">SPI_APP1_INT_SET</a> <a href="#">SPI_APP2_INT_SET</a> <a href="#">SPI_MST_TX_AFIFO_EMPTY_ERR_INT_SET</a> <a href="#">SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET</a> <a href="#">SPI_SLV_CMD_ERR_INT_SET</a> (reserved) <a href="#">SPI_SEG_MAGIC_ERR_INT_SET</a> <a href="#">SPI_DMA_SEG_TRANS_DONE_INT_SET</a> <a href="#">SPI_TRANS_DONE_INT_SET</a> <a href="#">SPI_SLV_WR_BUF_DONE_INT_SET</a> <a href="#">SPI_SLV_RD_BUF_DONE_INT_SET</a> <a href="#">SPI_SLV_RD_DMA_DONE_INT_SET</a> <a href="#">SPI_SLV_CMD9_INT_SET</a> <a href="#">SPI_SLV_CMD8_INT_SET</a> <a href="#">SPI_SLV_EN_QPI_INT_SET</a> <a href="#">SPI_SLV_EX_QPI_INT_SET</a> <a href="#">SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET</a> <a href="#">SPI_DMA_INFIFO_FULL_ERR_INT_SET</a>											
31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**SPI\_DMA\_INFIFO\_FULL\_ERR\_INT\_SET** The software set bit for [SPI\\_DMA\\_INFIFO\\_FULL\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_DMA\_OUTFIFO\_EMPTY\_ERR\_INT\_SET** The software set bit for [SPI\\_DMA\\_OUTFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_SLV\_EX\_QPI\_INT\_SET** The software set bit for [SPI\\_SLV\\_EX\\_QPI\\_INT](#) interrupt. (WT)

**SPI\_SLV\_EN\_QPI\_INT\_SET** The software set bit for [SPI\\_SLV\\_EN\\_QPI\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD7\_INT\_SET** The software set bit for [SPI\\_SLV\\_CMD7\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD8\_INT\_SET** The software set bit for [SPI\\_SLV\\_CMD8\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMD9\_INT\_SET** The software set bit for [SPI\\_SLV\\_CMD9\\_INT](#) interrupt. (WT)

**SPI\_SLV\_CMDA\_INT\_SET** The software set bit for [SPI\\_SLV\\_CMDA\\_INT](#) interrupt. (WT)

Continued on the next page...

**Register 20.18. SPI\_DMA\_INT\_SET\_REG (0x0044)**

Continued from the previous page...

**SPI\_SLV\_RD\_DMA\_DONE\_INT\_SET** The software set bit for [SPI\\_SLV\\_RD\\_DMA\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_WR\_DMA\_DONE\_INT\_SET** The software set bit for [SPI\\_SLV\\_WR\\_DMA\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_RD\_BUF\_DONE\_INT\_SET** The software set bit for [SPI\\_SLV\\_RD\\_BUF\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SLV\_WR\_BUF\_DONE\_INT\_SET** The software set bit for [SPI\\_SLV\\_WR\\_BUF\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_TRANS\_DONE\_INT\_SET** The software set bit for [SPI\\_TRANS\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_DMA\_SEG\_TRANS\_DONE\_INT\_SET** The software set bit for [SPI\\_DMA\\_SEG\\_TRANS\\_DONE\\_INT](#) interrupt. (WT)

**SPI\_SEG\_MAGIC\_ERR\_INT\_SET** The software set bit for [SPI\\_SEG\\_MAGIC\\_ERR\\_INT](#) interrupt. (WT)

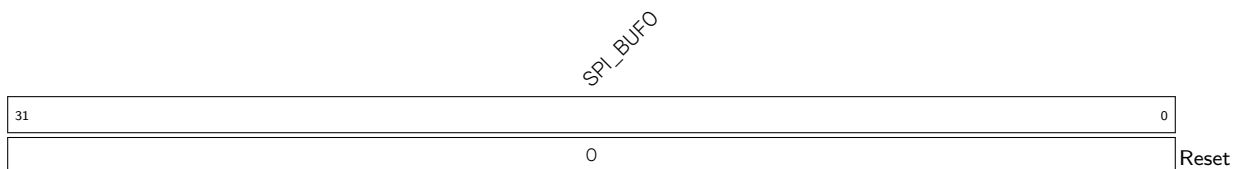
**SPI\_SLV\_CMD\_ERR\_INT\_SET** The software set bit for [SPI\\_SLV\\_CMD\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_MST\_RX\_AFIFO\_WFULL\_ERR\_INT\_SET** The software set bit for [SPI\\_MST\\_RX\\_AFIFO\\_WFULL\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_MST\_TX\_AFIFO\_EMPTY\_ERR\_INT\_SET** The software set bit for [SPI\\_MST\\_TX\\_AFIFO\\_EMPTY\\_ERR\\_INT](#) interrupt. (WT)

**SPI\_APP2\_INT\_SET** The software set bit for [SPI\\_APP2\\_INT](#) interrupt. (WT)

**SPI\_APP1\_INT\_SET** The software set bit for [SPI\\_APP1\\_INT](#) interrupt. (WT)

**Register 20.19. SPI\_WO\_REG (0x0098)**

**SPI\_BUFO** 32-bit data buffer 0. (R/W/SS)

## Register 20.20. SPI\_W1\_REG (0x009C)

SPI_BUF1	
31	0
0	
Reset	

**SPI\_BUF1** 32-bit data buffer 1. (R/W/SS)

## Register 20.21. SPI\_W2\_REG (0x00A0)

SPI_BUF2	
31	0
0	
Reset	

**SPI\_BUF2** 32-bit data buffer 2. (R/W/SS)

## Register 20.22. SPI\_W3\_REG (0x00A4)

SPI_BUF3	
31	0
0	
Reset	

**SPI\_BUF3** 32-bit data buffer 3. (R/W/SS)

## Register 20.23. SPI\_W4\_REG (0x00A8)

SPI_BUF4	
31	0
0	
Reset	

**SPI\_BUF4** 32-bit data buffer 4. (R/W/SS)

SPI\_BUF5

SPI\_BUF6

SPI\_BUF7

SPI\_BUF8



## SPI\_BUF9

**SPI\_BUF9** 32-bit data buffer 9. (R/W/SS)

## SPI\_BUF10

**SPI\_BUF10** 32-bit data buffer 10. (R/W/SS)

## SPI\_BUF11

**SPI\_BUF11** 32-bit data buffer 11. (R/W/SS)

## SPI\_BUF12

**SPI\_BUF12** 32-bit data buffer 12. (R/W/SS)

Register 20.32. SPI\_W13\_REG (0x00CC)

SPI_BUF13	
31	0
0	
Reset	

SPI\_BUF13 32-bit data buffer 13. (R/W/SS)

Register 20.33. SPI\_W14\_REG (0x00D0)

SPI_BUF14	
31	0
0	
Reset	

SPI\_BUF14 32-bit data buffer 14. (R/W/SS)

Register 20.34. SPI\_W15\_REG (0x00D4)

SPI_BUF15	
31	0
0	
Reset	

SPI\_BUF15 32-bit data buffer 15. (R/W/SS)

Register 20.35. SPI\_DATE\_REG (0x00F0)

(reserved)		SPI_DATE	
31	28	27	0
0	0	0	0
		0x2106070	
		Reset	

SPI\_DATE Version control register. (R/W)

## Chapter 21

### I2C Master Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP8684 to communicate with multiple external devices. These external devices can share one bus.

ESP8684 provides one I2C controller operating in master mode.

#### 21.1 Overview

I2C is a two-wire bus, consisting of a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I2C bus can be connected to a single or multiple master devices and a single or multiple slave devices. However, only one master device can access a slave at a time via the bus.

The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high, and sending nine clock pulses via SCL. The first eight pulses are used to transmit a 7-bit address followed by a read/write ( $R/\overline{W}$ ) bit. If the address of an I2C slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master can send data to the slave according to the  $R/\overline{W}$  bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once finishing communication, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send a RSTART condition, a slave address and a  $R/\overline{W}$  bit before changing its operation. The RSTART condition is used to change the transfer direction and the mode of the devices (master mode or slave mode).

#### 21.2 Features

The I2C master controller has the following features:

- Master mode only
- Communication between multiple masters
- Standard mode (100 Kbit/s)
- Fast mode (400 Kbit/s)
- 7-bit and 10-bit slave addressing
- Continuous data transfer achieved by pulling SCL low
- Programmable digital noise filtering
- Double addressing mode, which uses slave address and slave memory or register address

## 21.3 I2C Architecture

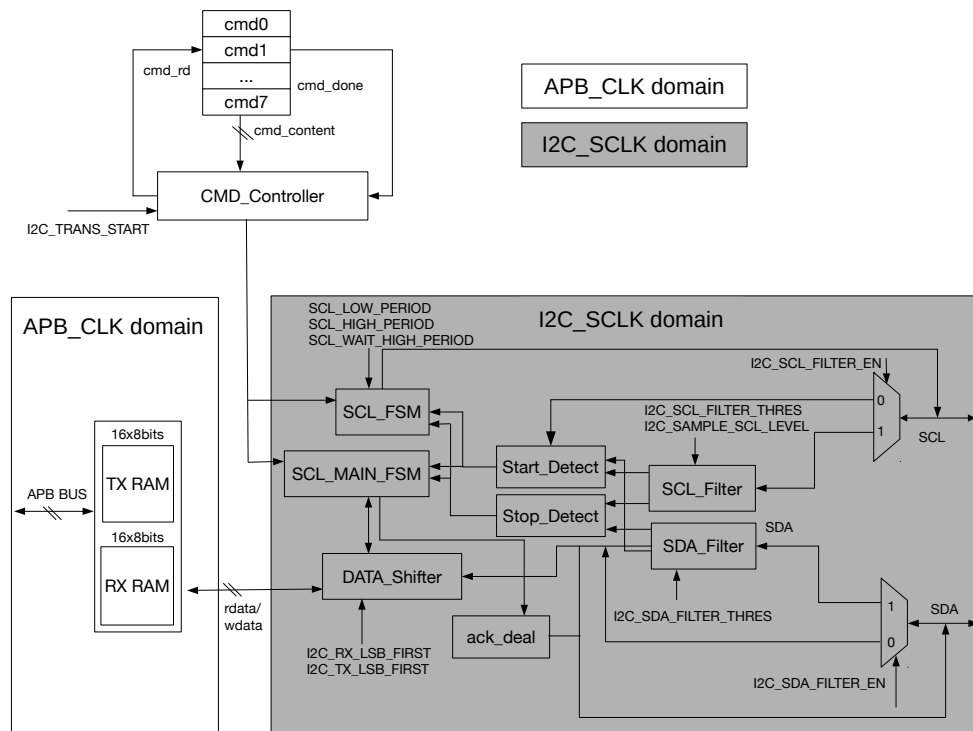


Figure 21.3-1. I2C Master Architecture

Figure 21.3-1 shows the architecture of an I2C master. The I2C master controller has the following main parts:

- transmit and receive memory (TX/RX RAM)
- command controller (CMD\_Controller)
- SCL clock controller (SCL\_FSM)
- SDA data controller (SCL\_MAIN\_FSM)
- serial/parallel data converter (DATA\_Shifter)
- filter for SCL (SCL\_Filter)
- filter for SDA (SDA\_Filter)
- ACK bit controller (ACK\_deal)

Besides, the I2C master controller also has a clock module which generates I2C clocks, and a synchronization module which synchronizes the APB bus and the I2C master controller.

The clock module is used to select clock sources, turn on and off clocks, and divide clocks. SCL\_Filter and SDA\_Filter remove noises on SCL input signals and SDA input signals respectively. The synchronization module synchronizes signal transfer between different clock domains.

Figure 21.3-2 and Figure 21.3-3 are the timing diagram and corresponding parameters of the I2C protocol. SCL\_FSM generates the SCL clock timing sequence conforming to the I2C protocol.

SCL\_MAIN\_FSM controls the execution of I2C commands and the sequence of the SDA line. Also, it controls the ACK\_deal module to generate ACK bit or detect the level of ACK bit on SDA line. CMD\_Controller is used for an I2C master to generate (R)START, STOP, WRITE, READ and END commands. TX RAM and RX RAM store data to be transmitted and data received respectively. DATA\_Shifter shifts data between serial and parallel form.

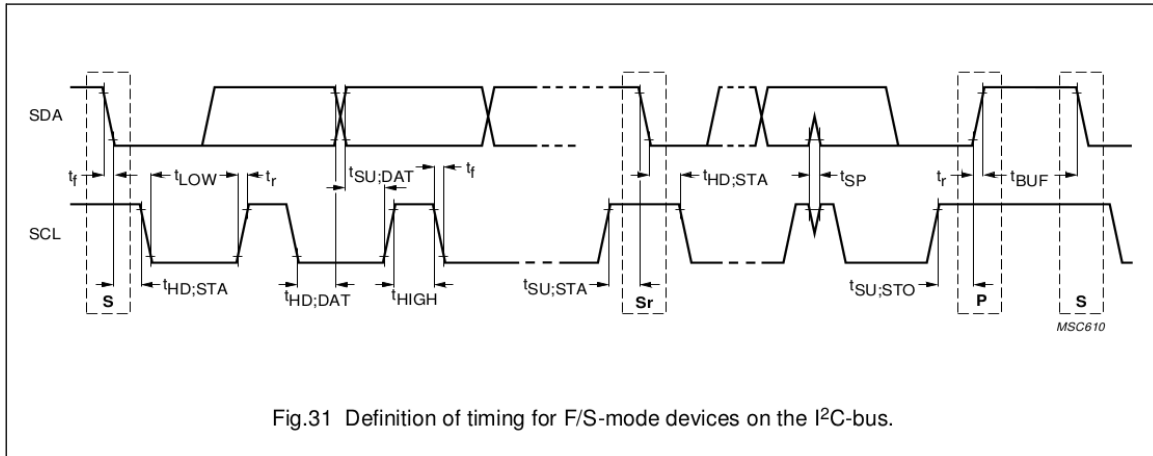


Figure 21.3-2. I2C Protocol Timing (Cited from Fig. 31 in [The I2C-bus specification](#) Version 2.1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	$f_{SCL}$	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	—	0.6	—	$\mu s$
LOW period of the SCL clock	$t_{LOW}$	4.7	—	1.3	—	$\mu s$
HIGH period of the SCL clock	$t_{HIGH}$	4.0	—	0.6	—	$\mu s$
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	—	0.6	—	$\mu s$
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I <sup>2</sup> C-bus devices	$t_{HD;DAT}$	5.0 0 <sup>(2)</sup>	— 3.45 <sup>(3)</sup>	— 0 <sup>(2)</sup>	— 0.9 <sup>(3)</sup>	$\mu s$ $\mu s$
Data set-up time	$t_{SU;DAT}$	250	—	100 <sup>(4)</sup>	—	ns
Rise time of both SDA and SCL signals	$t_r$	—	1000	$20 + 0.1C_b^{(5)}$	300	ns
Fall time of both SDA and SCL signals	$t_f$	—	300	$20 + 0.1C_b^{(5)}$	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	—	0.6	—	$\mu s$
Bus free time between a STOP and START condition	$t_{BUF}$	4.7	—	1.3	—	$\mu s$

Figure 21.3-3. I2C Timing Parameters (Cited from Table 5 in [The I2C-bus specification](#) Version 2.1)

## 21.4 Functional Description

Note that operations may differ between the I2C master controller in ESP8684 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

### 21.4.1 Clock Configuration

Registers, TX RAM, and RX RAM are configured and accessed in the APB\_CLK clock domain. The main logic of the I2C master controller, including SCL\_FSM, SCL\_MAIN\_FSM, SCL\_FILTER, SDA\_FILTER, and DATA\_SHIFTER, are in the I2C\_SCLK clock domain.

You can choose the clock source for I2C\_SCLK from XTAL\_CLK or RC\_FAST\_CLK via [I2C\\_SCLK\\_SEL](#). When [I2C\\_SCLK\\_SEL](#) is cleared, the clock source is XTAL\_CLK. When [I2C\\_SCLK\\_SEL](#) is set, the clock source is RC\_FAST\_CLK. The clock source is enabled by configuring [I2C\\_SCLK\\_ACTIVE](#) as high level, which passes through a fractional divider to generate I2C\_SCLK according to the following equation:

$$Divisor = I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

XTAL\_CLK supports both 26MHz and 40MHz crystal frequencies. The frequency of RC\_FAST\_CLK is 17.5 MHz. Limited by timing parameters, the derived clock I2C\_SCLK should operate at a frequency 20 times larger than SCL's frequency.

### 21.4.2 SCL and SDA Noise Filtering

SCL\_Filter and SDA\_Filter modules are identical and are used to filter signal noises on SCL and SDA, respectively. These filters can be enabled or disabled by configuring [I2C\\_SCL\\_FILTER\\_EN](#) and [I2C\\_SDA\\_FILTER\\_EN](#).

Take SCL\_Filter as an example. When enabled, SCL\_Filter samples input signals on the SCL line continuously. These input signals are valid only if they remain unchanged for consecutive [I2C\\_SCL\\_FILTER\\_THRES](#) I2C\_SCLK clock cycles. Given that only valid input signals can pass through the filter, SCL\_Filter can remove glitches whose pulse width is shorter than [I2C\\_SCL\\_FILTER\\_THRES](#) I2C\_SCLK clock cycles, while SDA\_Filter can remove glitches whose pulse width is shorter than [I2C\\_SDA\\_FILTER\\_THRES](#) I2C\_SCLK clock cycles.

### 21.4.3 Generating SCL Pulses in Idle State

Usually when the I2C bus is idle, the SCL line is held high. The I2C master controller in ESP8684 can be programmed to generate SCL pulses in idle state. If the [I2C\\_SCL\\_RST\\_SLV\\_EN](#) bit is set, hardware will send [I2C\\_SCL\\_RST\\_SLV\\_NUM](#) SCL pulses. When software reads 0 in [I2C\\_SCL\\_RST\\_SLV\\_EN](#) (this bit is cleared automatically by hardware), set [I2C\\_CONF\\_UPGATE](#) to stop this function.

### 21.4.4 Synchronization

I2C registers are configured in APB\_CLK domain, whereas the I2C master controller is configured in asynchronous I2C\_SCLK domain. Therefore, before being used by the I2C master controller, register values should be synchronized by first writing configuration registers and then writing 1 to [I2C\\_CONF\\_UPGATE](#). Registers that need synchronization are listed in Table 21.4-1.

Table 21.4-1. I2C Registers that Need Synchronization

Register	Parameter	Address
<a href="#">I2C_CTR_REG</a>	<a href="#">I2C_SLV_TX_AUTO_START_EN</a>	0x0004
	<a href="#">I2C_SDA_FORCE_OUT</a>	

	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

### 21.4.5 Open-Drain Output

SCL and SDA output drivers must be configured as open drain. There are two ways to achieve this:

1. Set `I2C_SCL_FORCE_OUT` and `I2C_SDA_FORCE_OUT`, and configure `GPIO_PINn_PAD_DRIVER` for corresponding SCL and SDA pads as open-drain.
2. Clear `I2C_SCL_FORCE_OUT` and `I2C_SDA_FORCE_OUT`.

Because these lines are configured as open-drain, the low-to-high transition time of each line is longer, determined together by the pull-up resistor and the line capacitance. The output duty cycle of I2C is limited by the SDA and SCL line's pull-up speed, mainly SCL's speed.

In addition, when `I2C_SCL_FORCE_OUT` and `I2C_SCL_PD_EN` are set to 1, SCL can be forced low; when `I2C_SDA_FORCE_OUT` and `I2C_SDA_PD_EN` are set to 1, SDA can be forced low.

## 21.4.6 Timing Parameter Configuration

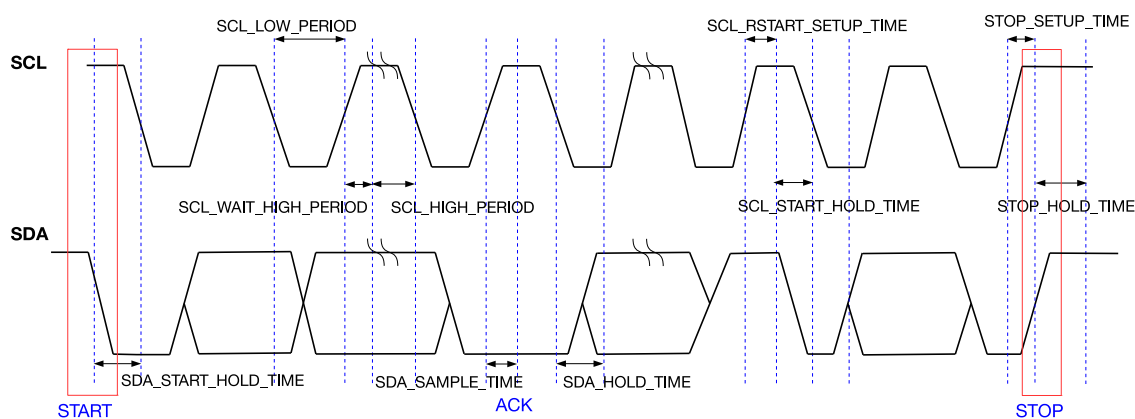


Figure 21.4-1. I2C Timing Diagram

Figure 21.4-1 shows the timing diagram of an I2C master. This figure also specifies registers used to configure the START bit, STOP bit, data hold time, data sample time, waiting time on the rising SCL edge, etc. Timing parameters are calculated as follows in I2C\_SCLK clock cycles:

1.  $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2.  $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3.  $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4.  $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5.  $t_r = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6.  $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7.  $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8.  $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9.  $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

Timing registers are:

1. **I2C\_SCL\_START\_HOLD\_TIME**: Specifies the interval between pulling SDA low and pulling SCL low when the master generates a START condition. This interval is  $(I2C\_SCL\_START\_HOLD\_TIME + 1)$  in I2C\_SCLK cycles.
2. **I2C\_SCL\_LOW\_PERIOD**: Specifies the low period of SCL. This period lasts  $(I2C\_SCL\_LOW\_PERIOD + 1)$  in I2C\_SCLK cycles. However, it could be extended when SCL is pulled low by peripheral devices or by an END command executed by the I2C master controller, or when the clock is stretched.
3. **I2C\_SCL\_WAIT\_HIGH\_PERIOD**: Specifies time for SCL to go high in I2C\_SCLK cycles. Please make sure that SCL could be pulled high within this time period. Otherwise, the high period of SCL may be incorrect.
4. **I2C\_SCL\_HIGH\_PERIOD**: Specifies the high period of SCL in I2C\_SCLK cycles. When SCL goes high



within (`I2C_SCL_WAIT_HIGH_PERIOD` + 1) in `I2C_SCLK` cycles, its frequency is:

$$f_{scl} = \frac{f_{I2C\_SCLK}}{I2C\_SCL\_LOW\_PERIOD + I2C\_SCL\_HIGH\_PERIOD + I2C\_SCL\_WAIT\_HIGH\_PERIOD + 3}$$

5. `I2C_SDA_SAMPLE_TIME`: Specifies the interval between the rising edge of SCL and the level sampling time of SDA. It is advised to set a value in the middle of SCL's high period, so as to correctly sample the level of SCL.
6. `I2C_SDA_HOLD_TIME`: Specifies the interval between changing the SDA output level and the falling edge of SCL.

Timing parameters limits corresponding register configuration.

1.  $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2.  $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. `I2C_SDA_HOLD_TIME` + `I2C_SCL_START_HOLD_TIME` > `SDA_FILTER_THRES` + 3
4. `I2C_SCL_WAIT_HIGH_PERIOD` < `I2C_SDA_SAMPLE_TIME` < `I2C_SCL_HIGH_PERIOD`
5. `I2C_SDA_SAMPLE_TIME` < `I2C_SCL_WAIT_HIGH_PERIOD` + `I2C_SCL_START_HOLD_TIME` + `I2C_SCL_RSTART_SETUP_TIME`

### 21.4.7 Timeout Control

The I2C master controller has three types of timeout control, namely timeout control for `SCL_FSM`, for `SCL_MAIN_FSM`, and for the SCL line. The first two are always enabled, while enabling the third is configurable.

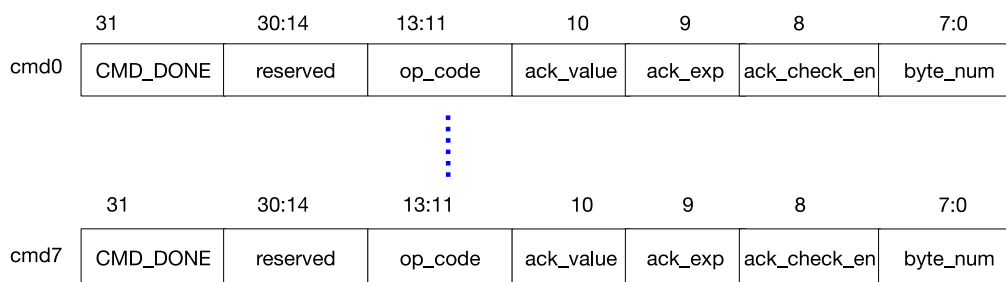
When `SCL_FSM` remains unchanged for more than  $2^{I2C\_SCL\_ST\_TO\_I2C}$  clock cycles, an `I2C_SCL_ST_TO_INT` interrupt is triggered, and then `SCL_FSM` goes to idle state. The value of `I2C_SCL_ST_TO_I2C` should be less than or equal to 22, which means `SCL_FSM` could remain unchanged for  $2^{22}$  `I2C_SCLK` clock cycles at most before the interrupt is generated.

When `SCL_MAIN_FSM` remains unchanged for more than  $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C}$  clock cycles, an `I2C_SCL_MAIN_ST_TO_INT` interrupt is triggered, and then `SCL_MAIN_FSM` goes to idle state. The value of `I2C_SCL_MAIN_ST_TO_I2C` should be less than or equal to 22, which means `SCL_MAIN_FSM` could remain unchanged for  $2^{22}$  `I2C_SCLK` clock cycles at most before the interrupt is generated.

Timeout control for SCL is enabled by setting `I2C_TIME_OUT_EN`. When the level of SCL remains unchanged for more than  $2^{I2C\_TIME\_OUT\_VALUE}$  clock cycles, an `I2C_TIME_OUT_INT` interrupt is triggered, and then the I2C bus goes to idle state.

### 21.4.8 Command Configuration

The `CMD_Controller` of the I2C master reads commands from 8 sequential command registers and controls `SCL_FSM` and `SCL_MAIN_FSM` accordingly.



**Figure 21.4-2. Structure of I2C Command Registers**

Figure 21.4-2 illustrates the structure of command registers. Fields of command registers are:

1. **CMD\_DONE**: Indicates that a command has been executed. After each command has been executed, the CMD\_DONE bit in the corresponding command register is set to 1 by hardware. By reading this bit, software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. **op\_code**: Indicates the command. The I2C master controller supports five commands:
  - **RSTART**: op\_code = 6. The I2C master controller sends a START bit or a RSTART bit defined by the I2C protocol.
  - **WRITE**: op\_code = 1. The I2C master controller sends a slave address, a register address (only in double addressing mode) and data to the slave.
  - **READ**: op\_code = 3. The I2C master controller reads data from the slave.
  - **STOP**: op\_code = 2. The I2C master controller sends a STOP bit defined by the I2C protocol. This code also indicates that the command sequence has been executed, and the CMD\_Controller stops reading commands. After restarted by software, the CMD\_Controller resumes reading commands from command register 0.
  - **END**: op\_code = 4. The I2C master controller pulls the SCL line low and suspends I2C communication. This code also indicates that the command sequence has completed, and the CMD\_Controller stops executing commands. Once software refreshes data in command registers and the RAM, the CMD\_Controller can be restarted to execute commands from command register 0 again.
3. **ack\_value**: Used to configure the level of the ACK bit sent by the I2C master controller during a read operation. This bit is ignored in RSTART, STOP, END and WRITE conditions.
4. **ack\_exp**: Used to configure the level of the ACK bit expected by the I2C master controller during a write operation. This bit is ignored during RSTART, STOP, END and READ conditions.
5. **ack\_check\_en**: Used to enable the I2C master controller during a write operation to check whether the ACK level sent by the slave matches ack\_exp in the command. If this bit is set and the level received does not match ack\_exp in the WRITE command, the master will generate an I2C\_NACK\_INT interrupt and a STOP condition for data transfer. If this bit is cleared, the controller will not check the ACK level sent by the slave. This bit is ignored during RSTART, STOP, END and READ conditions.
6. **byte\_num**: Specifies the length of data (in bytes) to be read or written. Can range from 1 to 255 bytes. This bit is ignored during RSTART, STOP and END conditions.

Each command sequence is executed starting from command register 0 and terminated by a STOP or an END. Therefore, there must be a STOP or an END command in one command sequence.

A complete data transfer on the I2C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, each one separated by an END command. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I2C communication.

### 21.4.9 TX/RX RAM Data Storage

Both TX RAM and RX RAM are  $16 \times 8$  bits, and can be accessed in FIFO or non-FIFO mode. If [I2C\\_NONFIFO\\_EN](#) bit is cleared, both RAMs are accessed in FIFO mode; if [I2C\\_NONFIFO\\_EN](#) bit is set, both RAMs are accessed in non-FIFO mode.

TX RAM stores data that the I2C master controller needs to send. During communication, when the I2C master controller needs to send data (except acknowledgement bits), it reads data from TX RAM and sends them sequentially via SDA. All data must be stored in TX RAM in the order they will be sent to slaves. The data stored in TX RAM include slave addresses, read/write bits, register addresses (only in double addressing mode) and data to be sent.

TX RAM can be read and written by the CPU. The CPU writes to TX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU writes to TX RAM via the fixed address [I2C\\_DATA\\_REG](#), with addresses for writing in TX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields ([I2C Base Address](#) + 0x100) ~ ([I2C Base Address](#) + 0x17C). Each byte in TX RAM occupies an entire word in the address space. Therefore, the address of the first byte is ([I2C Base Address](#) + 0x100), the second byte is ([I2C Base Address](#) + 0x104), the third byte is ([I2C Base Address](#) + 0x108), and so on. The CPU can only read TX RAM via direct addresses. Addresses for reading TX RAM are the same with addresses for writing TX RAM.

RX RAM stores data the I2C master controller receives during communication. Values of RX RAM can be read by software after I2C communication completes.

RX RAM can only be read by the CPU. The CPU reads RX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU reads RX RAM via the fixed address [I2C\\_DATA\\_REG](#), with addresses for reading RX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields ([I2C Base Address](#) + 0x180) ~ ([I2C Base Address](#) + 0x1FC). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is [I2C Base Address](#) + 0x180, the second byte is [I2C Base Address](#) + 0x184, the third byte is [I2C Base Address](#) + 0x188 and so on.

In FIFO mode, TX RAM of a master may wrap around to send data larger than the FIFO depth (for ESP8684 the depth is 16 bytes). Set [I2C\\_FIFO\\_PRT\\_EN](#). If the size of data to be sent is smaller than [I2C\\_TXFIFO\\_WM\\_THRHD](#), an I2C\_TXFIFO\_WM\_INT interrupt is generated. After receiving the interrupt, software continues writing to [I2C\\_DATA\\_REG](#). Please ensure that software writes to or refreshes TX RAM before the master sends data, otherwise it may result in unpredictable consequences.

In FIFO mode, RX RAM of a slave may also wrap around to receive data larger than the FIFO depth (for ESP8684 the depth is 16 bytes). Set [I2C\\_FIFO\\_PRT\\_EN](#) and clear [I2C\\_RX\\_FULL\\_ACK\\_LEVEL](#). If data already received (to be overwritten) is larger than [I2C\\_RXFIFO\\_WM\\_THRHD](#), an I2C\_RXFIFO\_WM\_INT interrupt is generated. After receiving the interrupt, software continues reading from [I2C\\_DATA\\_REG](#).

### 21.4.10 Data Conversion

DATA\_Shifter is used for serial/parallel conversion, converting byte data in TX RAM to an outgoing serial bitstream or an incoming serial bitstream to byte data in RX RAM. [I2C\\_RX\\_LSB\\_FIRST](#) and [I2C\\_TX\\_LSB\\_FIRST](#) can be used to select LSB- or MSB-first storage and transmission of data.

### 21.4.11 Addressing Mode

Besides 7-bit addressing, the ESP8684 I2C also supports 10-bit addressing and double addressing. 10-bit addressing can be mixed with 7-bit addressing.

Define the slave address as SLV\_ADDR. In 7-bit addressing mode, the slave address is SLV\_ADDR[6:0]; in 10-bit addressing mode, the slave address is SLV\_ADDR[9:0].

In 7-bit addressing mode, the master only needs to send one byte of address, which comprises SLV\_ADDR[6:0] and a  $R/\overline{W}$  bit. In 7-bit addressing mode, there is a special case called general call addressing (broadcast). When the master sends the general call address (0x00) and the  $R/\overline{W}$  bit is 0, slaves that support general call addressing respond to the master regardless of their own address.

In 10-bit addressing mode, the master needs to send two bytes of address. The first byte is slave\_addr\_first\_7bits followed by a  $R/\overline{W}$  bit, and slave\_addr\_first\_7bits should be configured as (0x78 | SLV\_ADDR[9:8]). The second byte is slave\_addr\_second\_byte, which should be configured as SLV\_ADDR[7:0]. Since a 10-bit slave address has one more byte than a 7-bit address, byte\_num of the WRITE command and the number of bytes in the RAM increase by one.

Some I2C slaves support double addressing, where the first address is the address of an I2C slave, and the second one is the slave's memory address. ESP8684 I2C also supports double addressing.

### 21.4.12 Starting of the I2C Master Controller

To start the I2C master controller, after configuring the controller to master mode ([I2C\\_MS\\_MODE](#)) and command registers, write 1 to [I2C\\_TRANS\\_START](#) in order that the master starts to parse and execute command sequences. The master always executes a command sequence starting from command register 0 to a STOP or an END at the end. To execute another command sequence starting from command register 0, refresh commands by writing 1 again to [I2C\\_TRANS\\_START](#).

## 21.5 Programming Example

This section provides programming examples for typical communication scenarios. ESP8684 has one I2C master controller. For the convenience of description, the I2C master in subsequent figures is ESP8684's I2C master controller, and the I2C slave are controllers compliant with [The I2C-bus specification](#) Version 2.1 and have corresponding functions. I2C master is referred to as I2C<sub>master</sub>, and I2C slave is referred to as I2C<sub>slave</sub>.

### 21.5.1 I2C<sub>master</sub> Writes to I2C<sub>slave</sub> with a 7-bit Address in One Command Sequence

### 21.5.1.1 Introduction

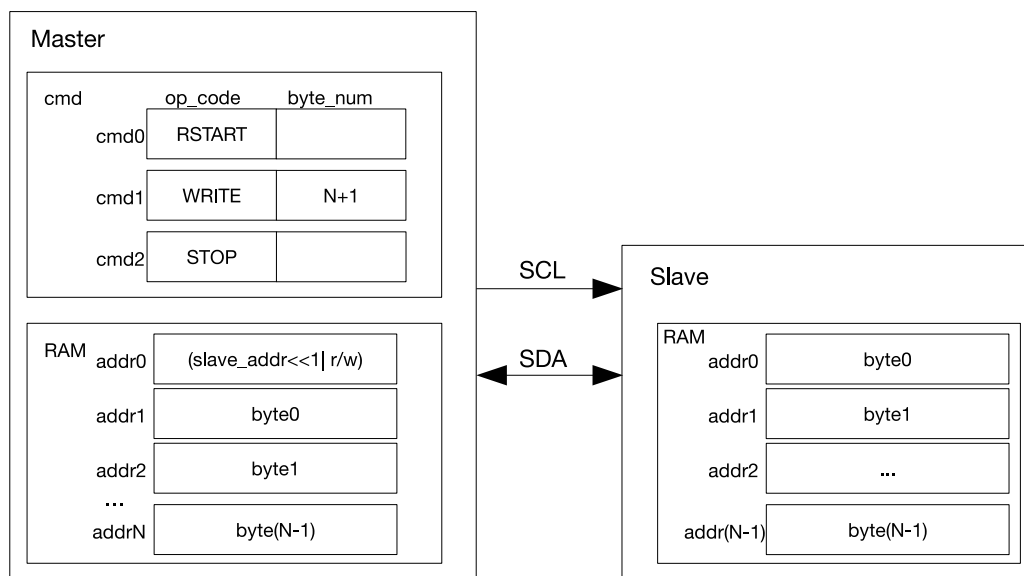


Figure 21.5-1. I2C<sub>master</sub> Writing to I2C<sub>slave</sub> with a 7-bit Address

Figure 21.5-1 shows how I2C<sub>master</sub> writes N bytes of data to I2C<sub>slave</sub>'s registers and RAM using 7-bit addressing. As shown in figure 21.5-1, the first byte in the RAM of I2C<sub>master</sub> is a 7-bit I2C<sub>slave</sub> address followed by a  $R/\overline{W}$  bit. When the  $R/\overline{W}$  bit is 0, it indicates a WRITE operation. The remaining bytes are used to store data ready for transfer. The cmd box contains related command sequences.

After the command sequence is configured and data in RAM is ready, I2C<sub>master</sub> enables the controller and initiates data transfer by setting the [I2C\\_TRANS\\_START](#) bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL being used by other masters or slaves.
2. Execute a RSTART command and send a START bit.
3. Execute a WRITE command by taking N+1 bytes from the RAM in order and send them to I2C<sub>slave</sub> in the same order. The first byte is the address of I2C<sub>slave</sub>.
4. Send a STOP. Once the I2C<sub>master</sub> transfers a STOP bit, an I2C\_TRANS\_COMPLETE\_INT interrupt is generated.

### 21.5.1.2 Configuration Example

1. Configure the timing parameter registers of I2C<sub>master</sub> and I2C<sub>slave</sub> according to Section 21.4.6. Adjust the timing of I2C<sub>slave</sub> according to its manual.
2. Set [I2C\\_MS\\_MODE](#) to 1.
3. Write 1 to [I2C\\_CONF\\_UPGATE](#) to synchronize registers.
4. Configure command registers of I2C<sub>master</sub>.

Command register	op_code	ack_value	ack_exp	ack_check_er	byte_num
<a href="#">I2C_COMMAND0</a>	RSTART	—	—	—	—
<a href="#">I2C_COMMAND1</a>	WRITE	ack_value	ack_exp	1	N+1

I2C_COMMAND2	STOP	—	—	—	—
--------------	------	---	---	---	---

- Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub> in either FIFO mode or non-FIFO mode according to Section 21.4.9.
- Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[7:0].
- Write 1 to I2C\_TRANS\_START to start transfer, and enable I2C<sub>slave</sub>.
- I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as a matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
- I2C<sub>master</sub> sends data, and checks ACK value or not according to ack\_check\_en.
- If data to be sent (N) is larger than the depth of TX FIFO, TX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
- After data transfer completes, I2C<sub>master</sub> executes the STOP command, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.2 I2C<sub>master</sub> Writes to I2C<sub>slave</sub> with a 10-bit Address in One Command Sequence

### 21.5.2.1 Introduction

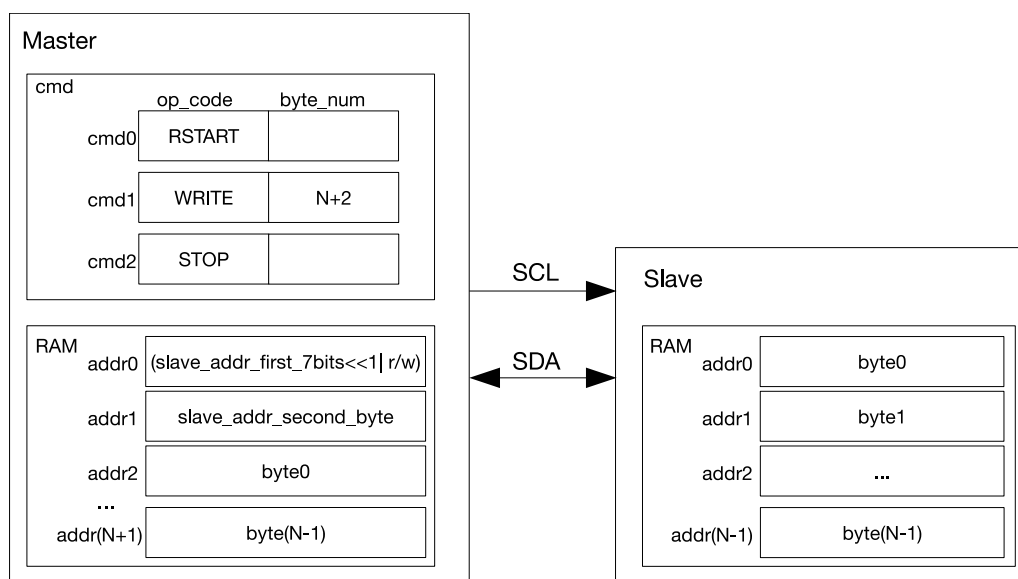


Figure 21.5-2. I2C<sub>master</sub> Writing to a Slave with a 10-bit Address

Figure 21.5-2 shows how I2C<sub>master</sub> writes N bytes of data using 10-bit addressing to an I2C slave. The configuration and transfer process is similar to what is described in 21.5.1, except that a 10-bit I2C<sub>slave</sub> address is formed from two bytes. Since a 10-bit I2C<sub>slave</sub> address has one more byte than a 7-bit I2C<sub>slave</sub> address, byte\_num and length of data in TX RAM increase by 1 accordingly.

### 21.5.2.2 Configuration Example

1. Set I2C\_MS\_MODE to 1.
2. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
3. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2	STOP	—	—	—	—

4. Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[9:0].
5. Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub>. The first byte of I2C<sub>slave</sub> address comprises ((0x78 | I2C\_SLAVE\_ADDR[9:8])«1) and a  $R/\overline{W}$  bit. The second byte of I2C<sub>slave</sub> address is I2C\_SLAVE\_ADDR[7:0]. These two bytes are followed by data to be sent in FIFO or non-FIFO mode.
6. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
7. Write 1 to I2C\_TRANS\_START to start transfer, and enable I2C<sub>slave</sub>.
8. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
9. I2C<sub>master</sub> sends data, and checks ACK value or not according to ack\_check\_en.
10. If data to be sent is larger than the depth of TX FIFO, TX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
11. After data transfer completes, I2C<sub>master</sub> executes the STOP command, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.3 I2C<sub>master</sub> Writes to I2C<sub>slave</sub> with Two 7-bit Addresses in One Command Sequence

### 21.5.3.1 Introduction

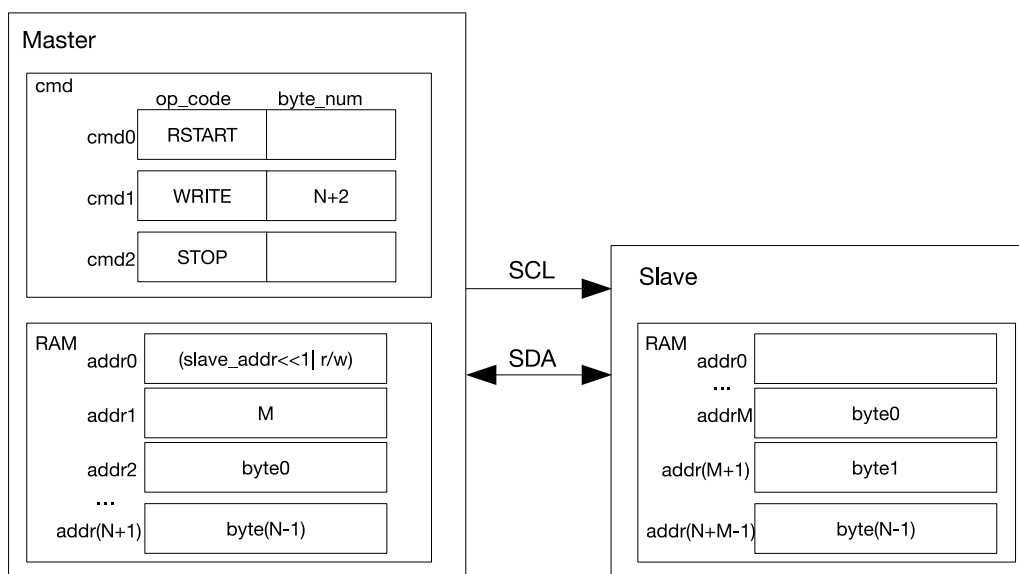


Figure 21.5-3. I2C<sub>master</sub> Writing to I2C<sub>slave</sub> with Two 7-bit Addresses

Figure 21.5-3 shows how I2C<sub>master</sub> writes N bytes of data to I2C<sub>slave</sub>'s registers or RAM using 7-bit double addressing. The configuration and transfer process is similar to what is described in Section 21.5.1, except that in 7-bit double addressing mode I2C<sub>master</sub> sends two 7-bit addresses. The first address is the address of an I2C slave, and the second one is I2C<sub>slave</sub>'s memory address (i.e. addrM in Figure 21.5-3 on the right). When using double addressing, RAM must be accessed in non-FIFO mode. The I2C slave put received byte0 ~ byte(N-1) into its registers or RAM in an order starting from addrM.

### 21.5.3.2 Configuration Example

1. Choose an I2C<sub>slave</sub> that supports double addressing mode and enable this mode.
2. Set `I2C_MS_MODE` to 1.
3. Write 1 to `I2C_CONF_UPGATE` to synchronize registers.
4. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
<code>I2C_COMMAND0</code>	RSTART	—	—	—	—
<code>I2C_COMMAND1</code>	WRITE	ack_value	ack_exp	1	N+2
<code>I2C_COMMAND2</code>	STOP	—	—	—	—

5. Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub> in FIFO or non-FIFO mode.
6. Set the address of I2C<sub>slave</sub> via `I2C_SLAVE_ADDR[7:0]`.
7. Write 1 to `I2C_CONF_UPGATE` to synchronize registers.



8. Write 1 to [I2C\\_TRANS\\_START](#) to start transfer, and enable  $I2C_{slave}$ .
9.  $I2C_{slave}$  compares the slave address sent by  $I2C_{master}$  with its own address. When `ack_check_en` in  $I2C_{master}$ 's WRITE command is 1,  $I2C_{master}$  checks ACK value each time it sends a byte. When `ack_check_en` is 0,  $I2C_{master}$  does not check ACK value and take  $I2C_{slave}$  as matching slave by default.
  - Match: If the received ACK value matches `ack_exp` (the expected ACK value),  $I2C_{master}$  continues data transfer.
  - Not match: If the received ACK value does not match `ack_exp`,  $I2C_{master}$  generates an `I2C_NACK_INT` interrupt and stops data transfer.
10.  $I2C_{slave}$  receives the RX RAM address sent by  $I2C_{master}$  and adds the offset.
11.  $I2C_{master}$  sends data, and checks ACK value or not according to `ack_check_en`.
12. If data to be sent is larger than the depth of TX FIFO, TX RAM of  $I2C_{master}$  may wrap around in FIFO mode. For details, please refer to Section [21.4.9](#).
13. After data transfer completes,  $I2C_{master}$  executes the STOP command, and generates an `I2C_TRANS_COMPLETE_INT` interrupt.

## 21.5.4 I2C<sub>master</sub> Writes to I2C<sub>slave</sub> with a 7-bit Address in Multiple Command Sequences

### 21.5.4.1 Introduction

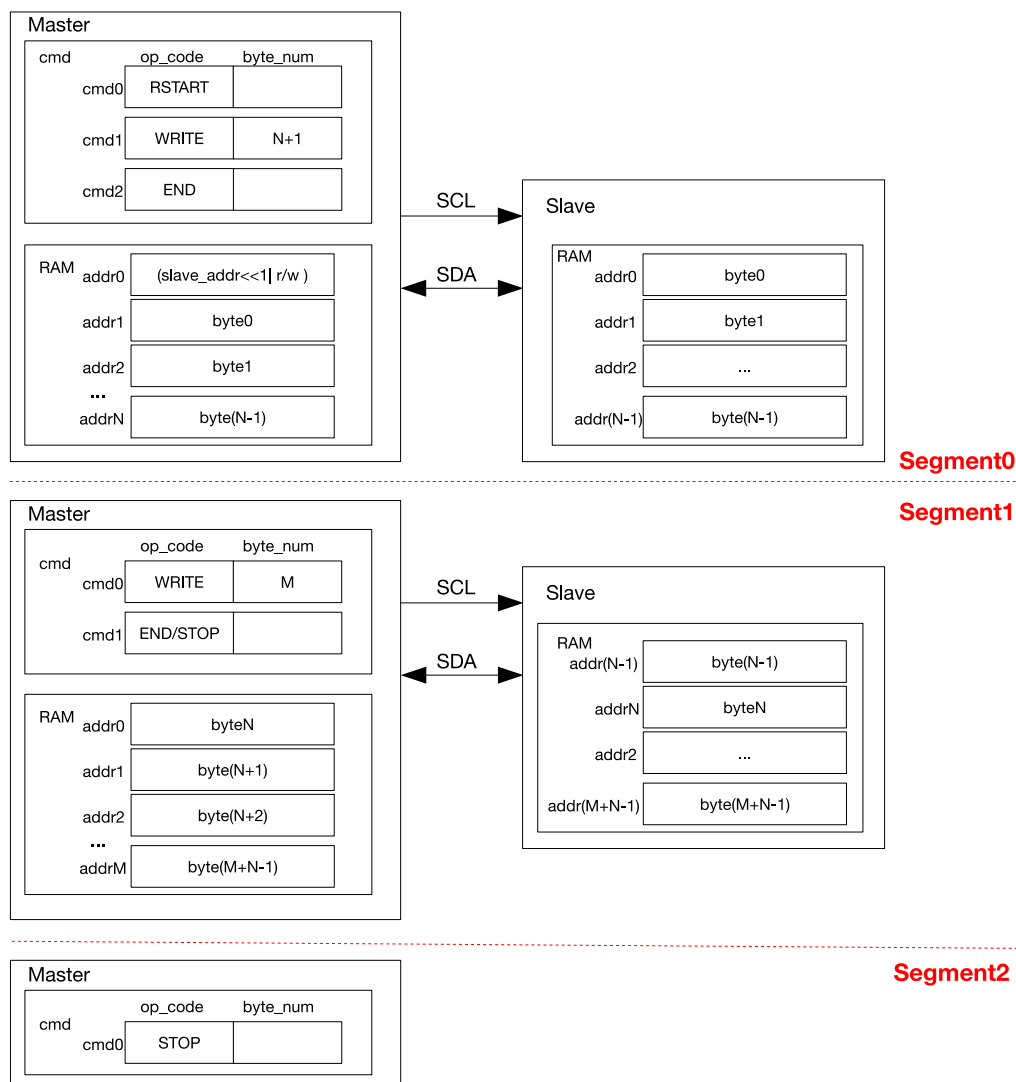


Figure 21.5-4. I2C<sub>master</sub> Writing to I2C<sub>slave</sub> with a 7-bit Address in Multiple Sequences

Given that the I2C RAM holds only 16 bytes, when data are too large to be processed even by the wrapped RAM, it is advised to transmit them in multiple command sequences by adding an END command at the end of every sequence. When the controller executes this END command to pull SCL low, software refreshes command sequence registers and RAM for next the transfer.

Figure 21.5-4 shows how I2C<sub>master</sub> writes to an I2C slave in two or three segments as an example. For the first segment, the CMD\_Controller registers are configured as shown in Segment0. Once data in I2C<sub>master</sub>'s RAM is ready and I2C\_TRANS\_START is set, I2C<sub>master</sub> initiates data transfer. After executing the END command, I2C<sub>master</sub> turns off the SCL clock and pulls SCL low to reserve the bus. Meanwhile, the controller generates an I2C\_END\_DETECT\_INT interrupt.

For the second segment, after detecting the I2C\_END\_DETECT\_INT interrupt, software refreshes the

CMD\_Controller registers, reloads the RAM and clears this interrupt, as shown in Segment1. If cmd1 in the second segment is a STOP, then data is transmitted to I2C<sub>slave</sub> in two segments. I2C<sub>master</sub> resumes data transfer after I2C\_TRANS\_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C\_END\_DETECT\_INT is detected, the CMD\_Controller registers of I2C<sub>master</sub> are configured as shown in Segment2. Once I2C\_TRANS\_START is set, I2C<sub>master</sub> generates a STOP bit and terminates the transfer.

Note that other I2C<sub>master</sub> devices will not transact on the bus between two segments. The bus is only released after a STOP signal is sent. The I2C master controller can be reset by setting I2C\_FSM\_RST field at any time. This field will later be cleared automatically by hardware.

### 21.5.4.2 Configuration Example

1. Set I2C\_MS\_MODE to 1.
2. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
3. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2	END	—	—	—	—

4. Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub> in either FIFO mode or non-FIFO mode according to Section 21.4.9.
5. Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[7:0].
6. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
7. Write 1 to I2C\_TRANS\_START to start transfer, and enable I2C<sub>slave</sub>.
8. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
9. I2C<sub>master</sub> sends data, and checks ACK value or not according to ack\_check\_en.
10. After the I2C\_END\_DETECT\_INT interrupt is generated, set I2C\_END\_DETECT\_INT\_CLR to 1 to clear this interrupt.
11. Update I2C<sub>master</sub>'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1	END/STOP	—	—	—	—

12. Write M bytes of data to be sent to TX RAM of I2C<sub>master</sub> in FIFO or non-FIFO mode.
13. Write 1 to [I2C\\_TRANS\\_START](#) bit to start transfer and repeat step 9.
14. If the command is a STOP, I2C stops transfer and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.
15. If the command is an END, repeat step 10.
16. Update I2C<sub>master</sub>'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
<a href="#">I2C_COMMAND1</a>	STOP	—	—	—	—

17. Write 1 to [I2C\\_TRANS\\_START](#) bit to start transfer.
18. I2C<sub>master</sub> executes the STOP command and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.5 I2C<sub>master</sub> Reads I2C<sub>slave</sub> with a 7-bit Address in One Command Sequence

### 21.5.5.1 Introduction

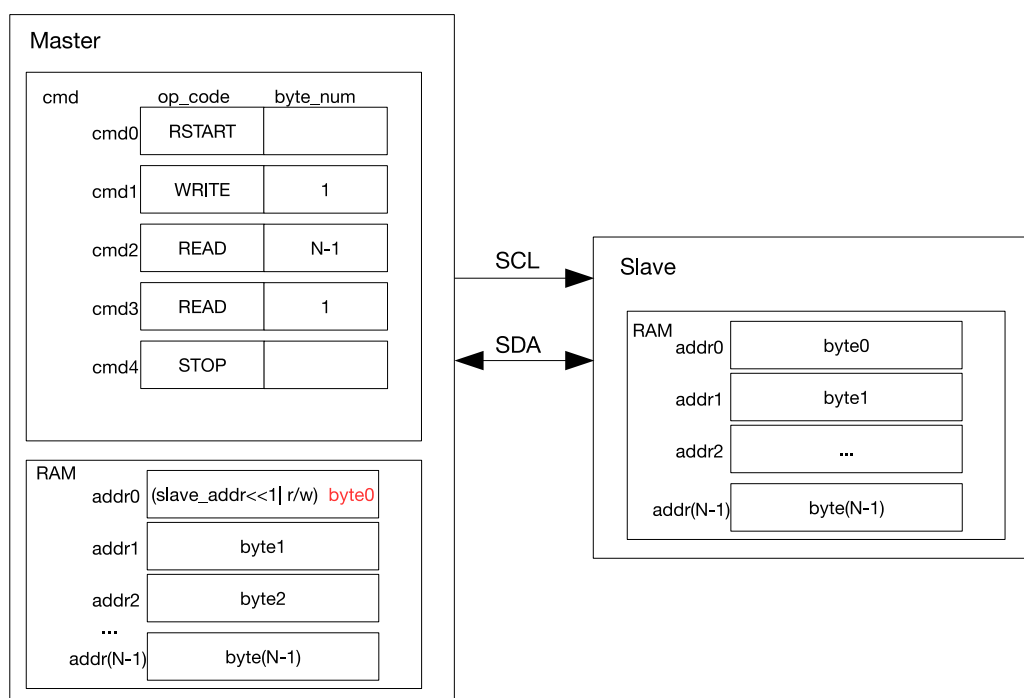


Figure 21.5-5. I2C<sub>master</sub> Reading I2C<sub>slave</sub> with a 7-bit Address

Figure 21.5-5 shows how I2C<sub>master</sub> reads N bytes of data from an I2C slave's registers or RAM using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed, I2C<sub>master</sub> sends I2C<sub>slave</sub> address. The byte sent comprises a 7-bit I2C<sub>slave</sub> address and a  $R/\overline{W}$  bit. When the  $R/\overline{W}$  bit is 1, it indicates a READ operation. If the address of an I2C slave matches the sent address, this matching slave starts sending

data to I2C<sub>master</sub>. I2C<sub>master</sub> generates acknowledgements according to ack\_value defined in the READ command upon receiving a byte.

As illustrated in Figure 21.5-5, I2C<sub>master</sub> executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd3. This configuration may be changed as required. I2C<sub>master</sub> writes received data into the controller RAM from addr0, whose original content (a I2C<sub>slave</sub> address and a  $R/\overline{W}$  bit) is overwritten by byte0 marked red in Figure 21.5-5.

### 21.5.5.2 Configuration Example

1. Set I2C\_MS\_MODE to 1.
2. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
3. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	1
I2C_COMMAND2	READ	0	0	1	N-1
I2C_COMMAND3	READ	1	0	1	1
I2C_COMMAND4	STOP	—	—	—	—

4. Write I2C<sub>slave</sub> address to TX RAM of I2C<sub>master</sub> in either FIFO mode or non-FIFO mode according to Section 21.4.9.
5. Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[7:0].
6. Write 1 to I2C\_CONF\_UPGATE to synchronize registers.
7. Write 1 to I2C\_TRANS\_START bit to start transfer, and enable I2C<sub>slave</sub>.
8. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
9. I2C<sub>slave</sub> sends data, and I2C<sub>master</sub> sends ACK value according to ack\_check\_en in the READ command.
10. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
11. After I2C<sub>master</sub> has received the last byte of data, set ack\_value to 1. I2C<sub>slave</sub> will stop transfer once receiving the I2C\_NACK\_INT interrupt.
12. After data transfer completes, I2C<sub>master</sub> executes the STOP command, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.6 I2C<sub>master</sub> Reads I2C<sub>slave</sub> with a 10-bit Address in One Command Sequence

### 21.5.6.1 Introduction

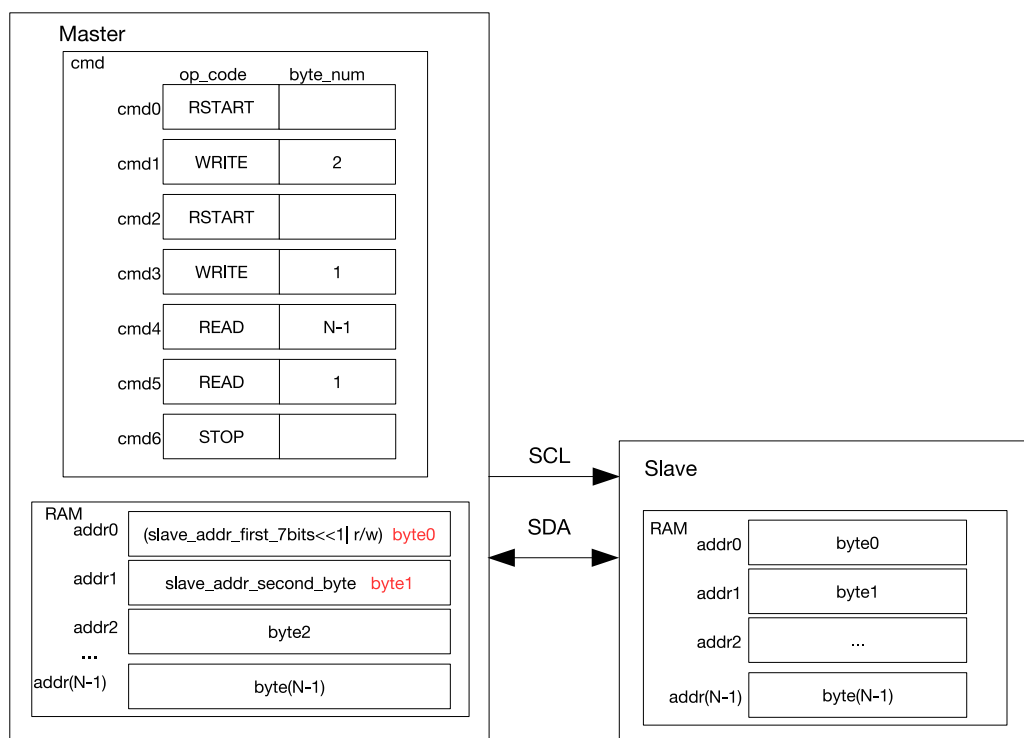


Figure 21.5-6. I2C<sub>master</sub> Reading I2C<sub>slave</sub> with a 10-bit Address

Figure 21.5-6 shows how I2C<sub>master</sub> reads data from an I2C slave's registers or RAM using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the I2C<sub>master</sub> is formed from two bytes, and correspondingly TX RAM of this master stores a 10-bit address of two bytes. The  $R/\overline{W}$  bit in the first byte is 0, which indicates a WRITE operation. After a RSTART condition, I2C<sub>master</sub> sends the first byte of address again to read data from I2C<sub>slave</sub>, but the  $R/\overline{W}$  bit is 1, which indicates a READ operation. The two address bytes can be configured as described in Section 21.5.2.

### 21.5.6.2 Configuration Example

1. Set `I2C_MS_MODE` to 1.
2. Write 1 to `I2C_CONF_UPGATE` to synchronize registers.
3. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND0</code>	RSTART	—	—	—	—
<code>I2C_COMMAND1</code>	WRITE	0	0	1	2
<code>I2C_COMMAND2</code>	RSTART	—	—	—	—
<code>I2C_COMMAND3</code>	WRITE	0	0	1	1
<code>I2C_COMMAND4</code>	READ	0	0	1	N-1

<a href="#">I2C_COMMAND5</a>	READ	1	0	1	1
<a href="#">I2C_COMMAND6</a>	STOP	—	—	—	—

4. Set the 10-bit address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[9:0].
5. Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub> in either FIFO or non-FIFO mode. The first byte of address comprises ((0x78 | I2C\_SLAVE\_ADDR[9:8])«1) and a  $R/\overline{W}$  bit, which is 1 and indicates a WRITE operation. The second byte of address is I2C\_SLAVE\_ADDR[7:0]. The third byte is ((0x78 | I2C\_SLAVE\_ADDR[9:8])«1) and a  $R/\overline{W}$  bit, which is 1 and indicates a READ operation.
6. Write 1 to [I2C\\_CONF\\_UPGATE](#) to synchronize registers.
7. Write 1 to [I2C\\_TRANS\\_START](#) to start transfer, and enable I2C<sub>slave</sub>.
8. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
9. I2C<sub>master</sub> sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C\_SLAVE\_ADDR[9:8])«1) and a  $R/\overline{W}$  bit that indicates READ.
10. I2C<sub>slave</sub> repeats step 8. If its address matches the address sent by I2C<sub>master</sub>, I2C<sub>slave</sub> proceed on to the next steps.
11. I2C<sub>slave</sub> sends data, and I2C<sub>master</sub> sends ACK value according to ack\_check\_en in the READ command.
12. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section [21.4.9](#).
13. After I2C<sub>master</sub> has received the last byte of data, set ack\_value to 1. I2C<sub>slave</sub> will stop transfer once receiving the I2C\_NACK\_INT interrupt.
14. After data transfer completes, I2C<sub>master</sub> executes the STOP command, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.7 I2C<sub>master</sub> Reads I2C<sub>slave</sub> with Two 7-bit Addresses in One Command Sequence

### 21.5.7.1 Introduction

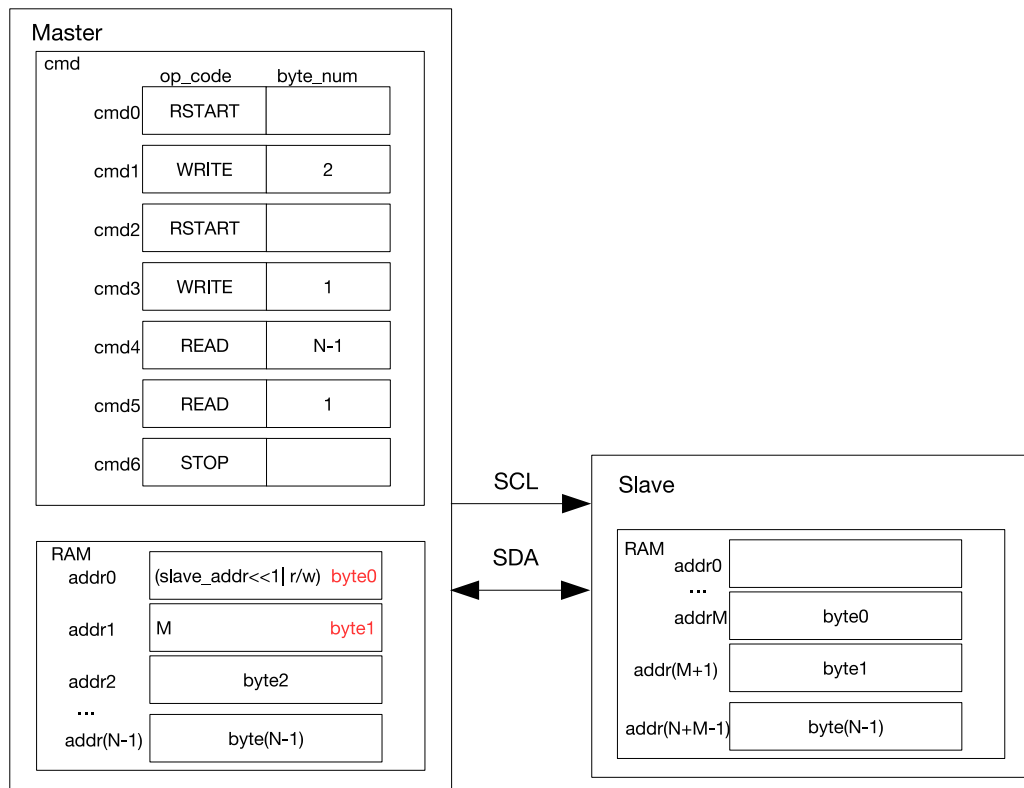


Figure 21.5-7. I2C<sub>master</sub> Reading N Bytes of Data from addrM of I2C<sub>slave</sub> with a 7-bit Address

Figure 21.5-7 shows how I2C<sub>master</sub> reads data from specified addresses in I2C slave's registers or RAM. I2C<sub>master</sub> sends two bytes of addresses: the first byte is a 7-bit I2C<sub>slave</sub> address followed by a  $R/\overline{W}$  bit, which is 0 and indicates a WRITE; the second byte is I2C<sub>slave</sub>'s memory address addrM. After a RSTART condition, I2C<sub>master</sub> sends the first byte of address again, but the  $R/\overline{W}$  bit is 1 which indicates a READ. Then, I2C<sub>master</sub> reads data starting from addrM.

### 21.5.7.2 Configuration Example

1. Set `I2C_MS_MODE` to 1.
2. Write 1 to `I2C_CONF_UPGATE` to synchronize registers.
3. Choose an I2C<sub>slave</sub> that supports double addressing mode and enable this mode.
4. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND0</code>	RSTART	—	—	—	—
<code>I2C_COMMAND1</code>	WRITE	0	0	1	2
<code>I2C_COMMAND2</code>	RSTART	—	—	—	—
<code>I2C_COMMAND3</code>	WRITE	0	0	1	1



<a href="#">I2C_COMMAND4</a>	READ	0	0	1	N-1
<a href="#">I2C_COMMAND5</a>	READ	1	0	1	1
<a href="#">I2C_COMMAND6</a>	STOP	—	—	—	—

5. Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[7:0].
6. Write I2C<sub>slave</sub> address and data to be sent to TX RAM of I2C<sub>master</sub> in either FIFO or non-FIFO mode according to Section [21.4.9](#). The first byte of address comprises ( I2C\_SLAVE\_ADDR[6:0])«1) and a  $R/\overline{W}$  bit, which is 0 and indicates a WRITE. The second byte of address is memory address M of I2C<sub>slave</sub>. The third byte is ( I2C\_SLAVE\_ADDR[6:0])«1) and a  $R/\overline{W}$  bit, which is 1 and indicates a READ.
7. Write 1 to [I2C\\_CONF\\_UPGATE](#) to synchronize registers.
8. Write 1 to [I2C\\_TRANS\\_START](#) to start transfer, and enable I2C<sub>slave</sub>.
9. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
10. I2C<sub>slave</sub> receives memory address sent by I2C<sub>master</sub> and adds the offset.
11. I2C<sub>master</sub> sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C\_SLAVE\_ADDR[9:8])«1) and a R bit.
12. I2C<sub>slave</sub> repeats step 9. If its address matches the address sent by I2C<sub>master</sub>, I2C<sub>slave</sub> proceed on to the next steps.
13. I2C<sub>slave</sub> sends data, and I2C<sub>master</sub> sends ACK value according to ack\_check\_en in the READ command.
14. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section [21.4.9](#).
15. After I2C<sub>master</sub> has received the last byte of data, set ack\_value to 1. I2C<sub>slave</sub> will stop transfer once receiving the I2C\_NACK\_INT interrupt.
16. After data transfer completes, I2C<sub>master</sub> executes the STOP command, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.5.8 I2C<sub>master</sub> Reads I2C<sub>slave</sub> with a 7-bit Address in Multiple Command Sequences

### 21.5.8.1 Introduction

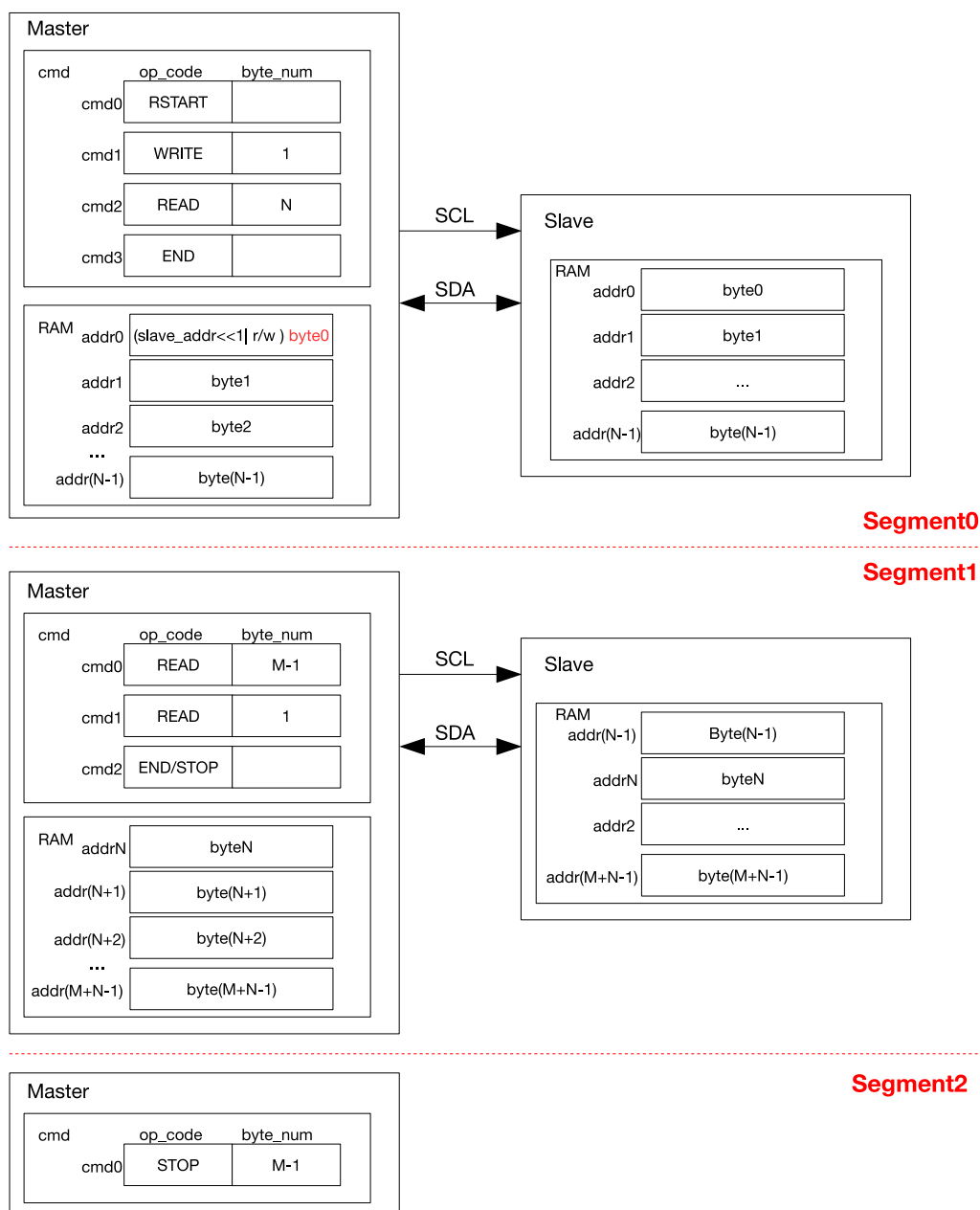


Figure 21.5-8. I2C<sub>master</sub> Reading I2C<sub>slave</sub> with a 7-bit Address in Segments

Figure 21.5-8 shows how I2C<sub>master</sub> reads (N+M) bytes of data from an I2C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. The procedures for Segment0 is similar to Figure 21.5-5, except that the last command is an END.
2. Prepare the data that I2C<sub>slave</sub> will send, and set [I2C\\_TRANS\\_START](#) (master) to start data transfer. After executing the END command, I2C<sub>master</sub> refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C\_END\_DETECT\_INT interrupt. If cmd2 in Segment1 is a

STOP, then data is read from I2C<sub>slave</sub> in two segments. I2C<sub>master</sub> resumes data transfer by setting [I2C\\_TRANS\\_START](#) and terminates the transfer by sending a STOP bit.

3. If cmd2 in Segment1 is an END, then data is read from I2C<sub>slave</sub> in three segments. After the second data transfer finishes and an I2C\_END\_DETECT\_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once [I2C\\_TRANS\\_START](#)(master) is set, I2C<sub>master</sub> terminates the transfer by sending a STOP bit.

### 21.5.8.2 Configuration Example

1. Set [I2C\\_MS\\_MODE](#) to 1.
2. Write 1 to [I2C\\_CONF\\_UPGATE](#) to synchronize registers.
3. Configure command registers of I2C<sub>master</sub>.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
<a href="#">I2C_COMMAND0</a>	RSTART	—	—	—	—
<a href="#">I2C_COMMAND1</a>	WRITE	0	0	1	1
<a href="#">I2C_COMMAND2</a>	READ	0	0	1	N
<a href="#">I2C_COMMAND3</a>	END	—	—	—	—

4. Write I2C<sub>slave</sub> address to TX RAM of I2C<sub>master</sub> in FIFO or non-FIFO mode.
5. Set the address of I2C<sub>slave</sub> via I2C\_SLAVE\_ADDR[7:0].
6. Write 1 to [I2C\\_CONF\\_UPGATE](#) to synchronize registers.
7. Write 1 to [I2C\\_TRANS\\_START](#) to start transfer, and enable I2C<sub>slave</sub>.
8. I2C<sub>slave</sub> compares the slave address sent by I2C<sub>master</sub> with its own address. When ack\_check\_en in I2C<sub>master</sub>'s WRITE command is 1, I2C<sub>master</sub> checks ACK value each time it sends a byte. When ack\_check\_en is 0, I2C<sub>master</sub> does not check ACK value and take I2C<sub>slave</sub> as matching slave by default.
  - Match: If the received ACK value matches ack\_exp (the expected ACK value), I2C<sub>master</sub> continues data transfer.
  - Not match: If the received ACK value does not match ack\_exp, I2C<sub>master</sub> generates an I2C\_NACK\_INT interrupt and stops data transfer.
9. I2C<sub>slave</sub> sends data, and I2C<sub>master</sub> sends ACK value according to ack\_check\_en in the READ command.
10. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of I2C<sub>master</sub> may wrap around in FIFO mode. For details, please refer to Section [21.4.9](#).
11. Once finishing reading data in the first READ command, I2C<sub>master</sub> executes the END command and triggers an I2C\_END\_DETECT\_INT interrupt, which is cleared by setting [I2C\\_END\\_DETECT\\_INT\\_CLR](#) to 1.
12. Update I2C<sub>master</sub>'s command registers using one of the following two methods:

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
<a href="#">I2C_COMMAND0</a>	READ	ack_value	ack_exp	1	M
<a href="#">I2C_COMMAND1</a>	END	—	—	—	—

Or

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
<a href="#">I2C_COMMAND0</a>	READ	0	0	1	M-1
<a href="#">I2C_COMMAND1</a>	READ	1	0	1	1
<a href="#">I2C_COMMAND2</a>	STOP	—	—	—	—

13. Prepare data that I2C<sub>slave</sub> will send.
14. Write 1 to [I2C\\_TRANS\\_START](#) bit to start transfer and repeat step 9.
15. If the last command is a STOP, then set ack\_value to 1 after I2C<sub>master</sub> has received the last byte of data. I2C<sub>slave</sub> stops transfer upon the I2C\_NACK\_INT interrupt. I2C<sub>master</sub> executes the STOP command to stop transfer and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.
16. If the last command is an END, then repeat step 11 and proceed on to the next steps.
17. Update I2C<sub>master</sub>'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_er	byte_num
<a href="#">I2C_COMMAND1</a>	STOP	—	—	—	—

18. Write 1 to [I2C\\_TRANS\\_START](#) bit to start transfer.
19. I2C<sub>master</sub> executes the STOP command to stop transfer, and generates an I2C\_TRANS\_COMPLETE\_INT interrupt.

## 21.6 Interrupts

- I2C\_DET\_START\_INT: Triggered when the master or the slave detects a START bit.
- I2C\_SCL\_MAIN\_ST\_TO\_INT: Triggered when the main state machine SCL\_MAIN\_FSM remains unchanged for over [I2C\\_SCL\\_MAIN\\_ST\\_TO\\_I2C\[23:0\]](#) clock cycles.
- I2C\_SCL\_ST\_TO\_INT: Triggered when the state machine SCL\_FSM remains unchanged for over [I2C\\_SCL\\_ST\\_TO\\_I2C\[23:0\]](#) clock cycles.
- I2C\_RXFIFO\_UDF\_INT: Triggered when the I2C master controller reads RX FIFO via the APB bus, but RX FIFO is empty.
- I2C\_TXFIFO\_OVF\_INT: Triggered when the I2C master controller writes TX FIFO via the APB bus, but TX FIFO is full.
- I2C\_NACK\_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C\_TRANS\_START\_INT: Triggered when the I2C master controller sends a START bit.
- I2C\_TIME\_OUT\_INT: Triggered when SCL stays high or low for more than  $2^{\text{I2C\_TIME\_OUT\_VALUE}}$  clock cycles during data transfer.
- I2C\_TRANS\_COMPLETE\_INT: Triggered when the I2C master controller detects a STOP bit.

- I2C\_MST\_TXFIFO\_UDF\_INT: Triggered when TX FIFO of the master underflows.
- I2C\_ARBITRATION\_LOST\_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.
- I2C\_BYTE\_TRANS\_DONE\_INT: Triggered when the I2C master controller sends or receives a byte.
- I2C\_END\_DETECT\_INT: Triggered when op\_code of the master indicates an END command and an END condition is detected.
- I2C\_RXFIFO\_OVF\_INT: Triggered when RX FIFO of the I2C master controller overflows.
- I2C\_TXFIFO\_WM\_INT: I2C TX FIFO watermark interrupt. Triggered when [I2C\\_FIFO\\_PRT\\_EN](#) is 1 and the pointers of TX FIFO are less than [I2C\\_TXFIFO\\_WM\\_THRHD](#)[4:0].
- I2C\_RXFIFO\_WM\_INT: I2C RX FIFO watermark interrupt. Triggered when [I2C\\_FIFO\\_PRT\\_EN](#) is 1 and the pointers of RX FIFO are greater than [I2C\\_RXFIFO\\_WM\\_THRHD](#)[4:0].

## 21.7 Register Summary

The addresses in this section are relative to **I2C Master controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Timing registers</b>			
<a href="#">I2C_SCL_LOW_PERIOD_REG</a>	Configures the low level width of SCL	0x0000	R/W
<a href="#">I2C_SDA_HOLD_REG</a>	Configures the hold time after a falling SCL edge	0x0030	R/W
<a href="#">I2C_SDA_SAMPLE_REG</a>	Configures the sample time after a rising SCL edge	0x0034	R/W
<a href="#">I2C_SCL_HIGH_PERIOD_REG</a>	Configures the high level width of SCL	0x0038	R/W
<a href="#">I2C_SCL_START_HOLD_REG</a>	Configures the delay between the SDA and SCL falling edge for a START condition	0x0040	R/W
<a href="#">I2C_SCL_RSTART_SETUP_REG</a>	Configures the delay between the rising edge of SCL and the falling edge of SDA	0x0044	R/W
<a href="#">I2C_SCL_STOP_HOLD_REG</a>	Configures the delay after the SCL clock edge for a STOP condition	0x0048	R/W
<a href="#">I2C_SCL_STOP_SETUP_REG</a>	Configures the delay between the SDA and SCL rising edge for a STOP condition	0x004C	R/W
<a href="#">I2C_SCL_ST_TIME_OUT_REG</a>	SCL status timeout register	0x0078	R/W
<a href="#">I2C_SCL_MAIN_ST_TIME_OUT_REG</a>	SCL main status timeout register	0x007C	R/W
<b>Configuration registers</b>			
<a href="#">I2C_CTR_REG</a>	Transmission configuration register	0x0004	varies
<a href="#">I2C_TO_REG</a>	Timeout control register	0x000C	R/W
<a href="#">I2C_FIFO_CONF_REG</a>	FIFO configuration register	0x0018	R/W
<a href="#">I2C_FILTER_CFG_REG</a>	SCL and SDA filter configuration register	0x0050	R/W
<a href="#">I2C_CLK_CONF_REG</a>	I2C clock configuration register	0x0054	R/W
<a href="#">I2C_SCL_SP_CONF_REG</a>	Power configuration register	0x0080	varies
<b>Status registers</b>			
<a href="#">I2C_SR_REG</a>	Describes I2C work status	0x0008	RO
<a href="#">I2C_FIFO_ST_REG</a>	FIFO status register	0x0014	RO
<a href="#">I2C_DATA_REG</a>	Read/write FIFO register	0x001C	R/W
<b>Interrupt registers</b>			
<a href="#">I2C_INT_RAW_REG</a>	Raw interrupt status	0x0020	R/SS/WTC
<a href="#">I2C_INT_CLR_REG</a>	Interrupt clear bits	0x0024	WT
<a href="#">I2C_INT_ENA_REG</a>	Interrupt enable bits	0x0028	R/W
<a href="#">I2C_INT_STATUS_REG</a>	Status of captured I2C communication events	0x002C	RO
<b>Command registers</b>			
<a href="#">I2C_COMDO_REG</a>	I2C command register 0	0x0058	varies
<a href="#">I2C_COMD1_REG</a>	I2C command register 1	0x005C	varies
<a href="#">I2C_COMD2_REG</a>	I2C command register 2	0x0060	varies

Name	Description	Address	Access
<a href="#">I2C_COMD3_REG</a>	I2C command register 3	0x0064	varies
<a href="#">I2C_COMD4_REG</a>	I2C command register 4	0x0068	varies
<a href="#">I2C_COMD5_REG</a>	I2C command register 5	0x006C	varies
<a href="#">I2C_COMD6_REG</a>	I2C command register 6	0x0070	varies
<a href="#">I2C_COMD7_REG</a>	I2C command register 7	0x0074	varies
<b>Address registers</b>			
<a href="#">I2C_TXFIFO_START_ADDR_REG</a>	I2C TX FIFO base address register	0x0100	HRO
<a href="#">I2C_RXFIFO_START_ADDR_REG</a>	I2C RX FIFO base address register	0x0180	HRO
<b>Version register</b>			
<a href="#">I2C_DATE_REG</a>	Version control register	0x00F8	R/W

## 21.8 Registers

The addresses in this section are relative to **I2C Master controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 21.1. I2C\_SCL\_LOW\_PERIOD\_REG (0x0000)**

(reserved)																I2C_SCL_LOW_PERIOD																	
31																9	8	0															
0 0																0																Reset	

**I2C\_SCL\_LOW\_PERIOD** This field is used to configure how long SCL remains low, in I2C module clock cycles. (R/W)

**Register 21.2. I2C\_SDA\_HOLD\_REG (0x0030)**

(reserved)																I2C_SDA_HOLD_TIME																	
31																9	8	0															
0 0																0																Reset	

**I2C\_SDA\_HOLD\_TIME** This field is used to configure the time to hold the data after the falling edge of SCL, in I2C module clock cycles. (R/W)

**Register 21.3. I2C\_SDA\_SAMPLE\_REG (0x0034)**

(reserved)																I2C_SDA_SAMPLE_TIME																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
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																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0															

**I2C\_SDA\_SAMPLE\_TIME** This field is used to configure how long SDA is sampled, in I2C module clock cycles. (R/W)



**Register 21.4. I2C\_SCL\_HIGH\_PERIOD\_REG (0x0038)**

(reserved)																I2C_SCL_WAIT_HIGH_PERIOD								I2C_SCL_HIGH_PERIOD																															
31																16								15								9								8								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 field is used to configure how long SCL remains high, in I2C module clock cycles. (R/W)

**I2C\_SCL\_WAIT\_HIGH\_PERIOD** This field is used to configure the SCL\_FSM's waiting period for SCL high level, in I2C module clock cycles. (R/W)

**Register 21.5. I2C\_SCL\_START\_HOLD\_REG (0x0040)**

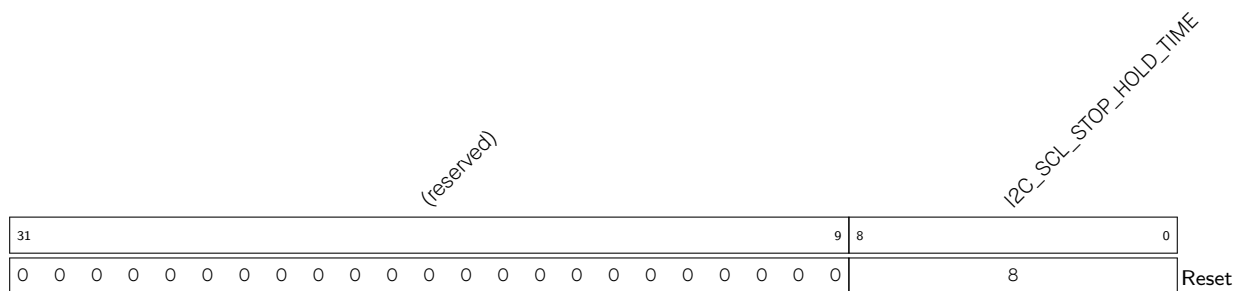
(reserved)																								I2C_SCL_START_HOLD_TIME									
31																								9	8	0							
0 0																								8								Reset	

**I2C\_SCL\_START\_HOLD\_TIME** This field is used to configure the time between the falling edge of SDA and the falling edge of SCL for a START condition, in I2C module clock cycles. (R/W)

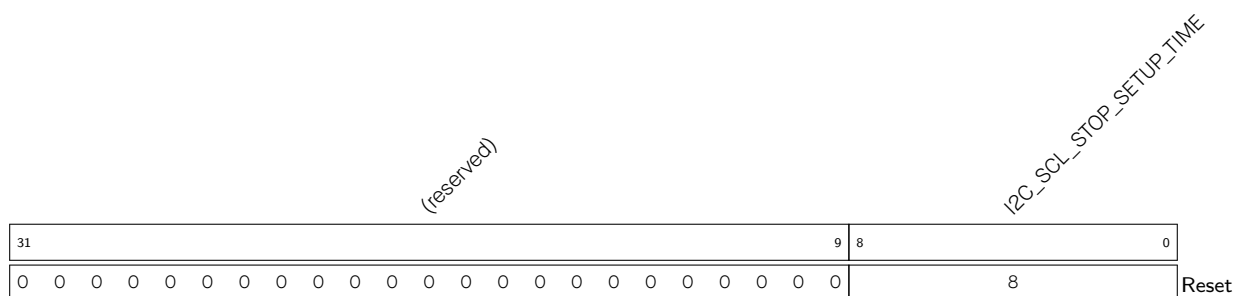
**Register 21.6. I2C\_SCL\_RSTART\_SETUP\_REG (0x0044)**

(reserved)																								I2C_SCL_RSTART_SETUP_TIME									
31																								9	8	0							
0 0																								8								Reset	

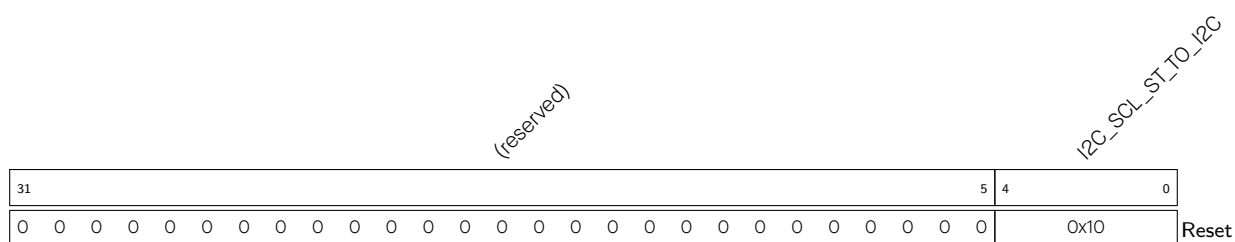
**I2C\_SCL\_RSTART\_SETUP\_TIME** This field is used to configure the time between the rising edge of SCL and the falling edge of SDA for a RSTART condition, in I2C module clock cycles. (R/W)

**Register 21.7. I2C\_SCL\_STOP\_HOLD\_REG (0x0048)**

**I2C\_SCL\_STOP\_HOLD\_TIME** This field is used to configure the delay after the STOP condition, in I2C module clock cycles. (R/W)

**Register 21.8. I2C\_SCL\_STOP\_SETUP\_REG (0x004C)**

**I2C\_SCL\_STOP\_SETUP\_TIME** This field is used to configure the time between the rising edge of SCL and the rising edge of SDA, in I2C module clock cycles. (R/W)

**Register 21.9. I2C\_SCL\_ST\_TIME\_OUT\_REG (0x0078)**

**I2C\_SCL\_ST\_TO\_I2C** The maximum time that SCL\_FSM remains unchanged. It should be no more than 23. (R/W)

I2C\_SCL\_MAIN\_ST\_TO\_I2C

(reserved)

**I2C\_SCL\_MAIN\_ST\_TO\_I2C** The maximum time that SCL\_MAIN\_FSM remains unchanged. It should be no more than 23. (R/W)

### Register 21.11. I2C\_CTR\_REG (0x0004)

[illegible]

**I2C\_SDA\_FORCE\_OUT** Configures the SDA output mode.

0: Open drain output

1: Direct output

(R/W)

**I2C\_SCL\_FORCE\_OUT** Configures the SCL output mode.

0: Open drain output

1: Direct output

(R/W)

**I2C\_SAMPLE\_SCL\_LEVEL** This bit is used to select the sampling mode. 0: samples SDA data on the SCL high level; 1: samples SDA data on the SCL low level. (R/W)

**I2C\_RX\_FULL\_ACK\_LEVEL** This bit is used to configure the ACK value that need to be sent by master when I2C\_RXFIFO\_CNT has reached the threshold. (R/W)

**I2C\_MS\_MODE** Set this bit to configure the I2C master controller as an I2C Master. Clear this bit to make the I2C master controller non-operational. (R/W)

**I2C\_TRANS\_START** Set this bit to start sending the data in TX FIFO. (WT)

**I2C\_TX\_LSB\_FIRST** This bit is used to control the order to send data. 0: sends data from the most significant bit; 1: sends data from the least significant bit. (R/W)

**I2C\_RX\_LSB\_FIRST** This bit is used to control the order to receive data. 0: receives data from the most significant bit; 1: receives data from the least significant bit. (R/W)

**I2C\_CLK\_EN** Reserved. (R/W)

**I2C ARBITRATION EN** This is the enable bit for I2C bus arbitration function. (R/W)

**I2C\_FSM\_RST** This bit is used to reset the SCL\_FSM. (WT)

**I2C\_CONF\_UPGATE** Synchronization bit. (WT)

**I2C SLV TX AUTO START EN** This is the enable bit for slave to send data automatically. (R/W)

**Register 21.12. I2C\_TO\_REG (0x000C)**

(reserved)																										I2C_TIME_OUT_EN		I2C_TIME_OUT_VALUE		
31																										6	5	4	0	
0 0																										0	0x10		Reset	

**I2C\_TIME\_OUT\_VALUE** This field is used to configure the timeout value for receiving a data bit in I2C\_SCLK clock cycles. The configured timeout value equals  $2^{I2C\_TIME\_OUT\_VALUE}$  clock cycles. (R/W)

**I2C\_TIME\_OUT\_EN** This is the enable bit for timeout control. (R/W)

**Register 21.13. I2C\_FIFO\_CONF\_REG (0x0018)**

(reserved)																I2C_FIFO_PRT_EN		I2C_TX_FIFO_RST		I2C_RX_FIFO_RST		(reserved)		I2C_NONFIFO_EN		(reserved)		I2C_TXFIFO_WM_THRHD		(reserved)		I2C_RXFIFO_WM_THRHD	
31																15	14	13	12	11	10	9	8	5		4	3	0					
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																1	0	0	0	0	0	0x2		0	0x6		Reset						

**I2C\_RXFIFO\_WM\_THRHD** The watermark threshold of RX FIFO in non-FIFO mode. When I2C\_FIFO\_PRT\_EN is 1 and RX FIFO counter is bigger than I2C\_RXFIFO\_WM\_THRHD[3:0], I2C\_RXFIFO\_WM\_INT\_RAW bit is valid. (R/W)

**I2C\_TXFIFO\_WM\_THRHD** The watermark threshold of TX FIFO in non-FIFO mode. When I2C\_FIFO\_PRT\_EN is 1 and TX FIFO counter is smaller than I2C\_TXFIFO\_WM\_THRHD[3:0], I2C\_TXFIFO\_WM\_INT\_RAW bit is valid. (R/W)

**I2C\_NONFIFO\_EN** Set this bit to enable APB non-FIFO mode. (R/W)

**I2C\_RX\_FIFO\_RST** Set this bit to reset RX FIFO. (R/W)

**I2C\_TX\_FIFO\_RST** Set this bit to reset TX FIFO. (R/W)

**I2C\_FIFO\_PRT\_EN** The control enable bit of FIFO pointer in non-FIFO mode. This bit controls the valid bits and TX/RX FIFO overflow, underflow, full and empty interrupts. (R/W)

**Register 21.14. I2C\_FILTER\_CFG\_REG (0x0050)**

(reserved)																				I2C_SDA_FILTER_EN		I2C_SCL_FILTER_EN		I2C_SDA_FILTER_THRES		I2C_SCL_FILTER_THRES		
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	1	1	0	0	Reset				

Reset

**I2C\_SCL\_FILTER\_THRES** When a pulse on the SCL input has smaller width than the value of this field in I2C module clock cycles, the I2C master controller ignores that pulse. (R/W)

**I2C\_SDA\_FILTER\_THRES** When a pulse on the SDA input has smaller width than the value of this field in I2C module clock cycles, the I2C master controller ignores that pulse. (R/W)

**I2C\_SCL\_FILTER\_EN** This is the filter enable bit for SCL. (R/W)

**I2C\_SDA\_FILTER\_EN** This is the filter enable bit for SDA. (R/W)

**Register 21.15. I2C\_CLK\_CONF\_REG (0x0054)**

(reserved)										I2C_SCLK_ACTIVE I2C_SCLK_SEL		I2C_SCLK_DIV_B		I2C_SCLK_DIV_A		I2C_SCLK_DIV_NUM			
31											22	21	20	19	14	13	8	7	0
0	0	0	0	0	0	0	0	0	0	1	0	0		0		0		Reset	

Reset

**I2C\_SCLK\_DIV\_NUM** The integral part of the divisor. (R/W)

**I2C\_SCLK\_DIV\_A** The numerator of the divisor's fractional part. (R/W)

**I2C\_SCLK\_DIV\_B** The denominator of the divisor's fractional part. (R/W)

**I2C\_SCLK\_SEL** The clock selection bit for the I2C master controller. 0: XTAL\_CLK; 1: RC\_FAST\_CLK. (R/W)

**I2C\_SCLK\_ACTIVE** The clock switch bit for the I2C master controller. (R/W)

**Register 21.16. I2C\_SCL\_SP\_CONF\_REG (0x0080)**

(reserved)																								I2C_SDA_PD_EN		I2C_SCL_PD_EN		I2C_SCL_RST_SLV_NUM				I2C_SCL_RST_SLV_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
31																								8	7	6	5	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_RST\_SLV\_EN** When the master is idle, set this bit to send out SCL pulses. The number of pulses equals to I2C\_SCL\_RST\_SLV\_NUM[4:0]. (R/W/SC)

**I2C\_SCL\_RST\_SLV\_NUM** Configures the pulses of SCL generated. Valid when I2C\_SCL\_RST\_SLV\_EN is 1. (R/W)

**I2C\_SCL\_PD\_EN** The power down enable bit for the I2C output SCL line. 0: Not power down; 1: Power down. Set I2C\_SCL\_FORCE\_OUT and I2C\_SCL\_PD\_EN to 1 to stretch SCL low. (R/W)

**I2C\_SDA\_PD\_EN** The power down enable bit for the I2C output SDA line. 0: Not power down; 1: Power down. Set I2C\_SDA\_FORCE\_OUT and I2C\_SDA\_PD\_EN to 1 to stretch SDA low. (R/W)

**Register 21.17. I2C\_SR\_REG (0x0008)**

<div>(reserved)</div>																																<div>I2C_SCL_STATE_LAST</div>				<div>(reserved)</div>			<div>I2C_SCL_MAIN_STATE_LAST</div>				<div>(reserved)</div>			<div>I2C_TXFIFO_CNT</div>				<div>(reserved)</div>			<div>I2C_RXFIFO_CNT</div>				<div>(reserved)</div>			<div>I2C_BUS_BUSY</div>		<div>I2C_ARB_LOST</div>		<div>(reserved)</div>		<div>I2C_RESP_REC</div>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
31	30	28	27	26	24	23	22	18	17	13	12	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_RESP\_REC** The received ACK value. 0: ACK; 1: NACK. (RO)

**I2C\_ARB\_LOST** When the I2C master controller loses control of the SCL line, this bit changes to 1. (RO)

**I2C\_BUS\_BUSY** 0: The I2C bus is in idle state; 1: The I2C bus is busy transferring data. (RO)

**I2C\_RXFIFO\_CNT** This field represents the number of data bytes to be sent. (RO)

**I2C\_TXFIFO\_CNT** This field stores the number of data bytes received in RAM. (RO)

**I2C\_SCL\_MAIN\_STATE\_LAST** This field indicates the status of the state machine. 0: idle; 1: address shift; 2: ACK address; 3: receive data; 4: transmit data; 5: send ACK; 6: wait for ACK. (RO)

**I2C\_SCL\_STATE\_LAST** This field indicates the status of the state machine used to produce SCL. 0: idle; 1: start; 2: falling edge; 3: low; 4: rising edge; 5: high; 6: stop. (RO)

Register 21.18. I2C\_FIFO\_ST\_REG (0x0014)

(reserved)																I2C_TXFIFO_WADDR								(reserved)								I2C_TXFIFO_RADDR								(reserved)								I2C_RXFIFO_WADDR								(reserved)								I2C_RXFIFO_RADDR							
31																19				18				15				14				13				10				9				8				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				Reset																							

**I2C\_RXFIFO\_RADDR** This is the offset address of the APB reading from RX FIFO. (RO)

**I2C\_RXFIFO\_WADDR** This is the offset address of the I2C master controller receiving data and writing to RX FIFO. (RO)

**I2C\_TXFIFO\_RADDR** This is the offset address of the I2C master controller reading from TX FIFO.  
(RO)

**I2C\_TXFIFO\_WADDR** This is the offset address of APB bus writing to TX FIFO. (RO)

### Register 21.19. I2C\_DATA\_REG (0x001C)

[illegible]

**I2C\_FIFO\_RDATA** This field is used to read data from RX FIFO, or write data to TX FIFO. (R/W)



## Register 21.20. I2C\_INT\_RAW\_REG (0x0020)

(reserved)																I2C_DET_START_INT_RAW I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_ST_TO_INT_RAW I2C_RXFIFO_UDF_INT_RAW I2C_TXFIFO_UDF_INT_RAW I2C_NACK_INT_RAW I2C_TRANS_START_INT_RAW I2C_TRANS_OUT_INT_RAW I2C_TRANS_COMPLETE_INT_RAW I2C_ARBTRATION_LOST_INT_RAW I2C_BYTE_TRANS_DONE_INT_RAW I2C_END_DETECT_INT_RAW I2C_RXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT_RAW																	
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0									

**I2C\_RXFIFO\_WM\_INT\_RAW** The raw interrupt bit for the I2C\_RXFIFO\_WM\_INT interrupt. (R/SS/WTC)

**I2C\_TXFIFO\_WM\_INT\_RAW** The raw interrupt bit for the I2C\_TXFIFO\_WM\_INT interrupt. (R/SS/WTC)

**I2C\_RXFIFO\_OVF\_INT\_RAW** The raw interrupt bit for the I2C\_RXFIFO\_OVF\_INT interrupt. (R/SS/WTC)

**I2C\_END\_DETECT\_INT\_RAW** The raw interrupt bit for the I2C\_END\_DETECT\_INT interrupt. (R/SS/WTC)

**I2C\_BYTE\_TRANS\_DONE\_INT\_RAW** The raw interrupt bit for the I2C\_BYTE\_TRANS\_DONE\_INT interrupt. (R/SS/WTC)

**I2C\_ARBTRATION\_LOST\_INT\_RAW** The raw interrupt bit for the I2C\_ARBTRATION\_LOST\_INT interrupt. (R/SS/WTC)

**I2C\_MST\_TXFIFO\_UDF\_INT\_RAW** The raw interrupt bit for the I2C\_MST\_TXFIFO\_UDF\_INT interrupt. (R/SS/WTC)

**I2C\_TRANS\_COMPLETE\_INT\_RAW** The raw interrupt bit for the I2C\_TRANS\_COMPLETE\_INT interrupt. (R/SS/WTC)

**I2C\_TIME\_OUT\_INT\_RAW** The raw interrupt bit for the I2C\_TIME\_OUT\_INT interrupt. (R/SS/WTC)

**I2C\_TRANS\_START\_INT\_RAW** The raw interrupt bit for the I2C\_TRANS\_START\_INT interrupt. (R/SS/WTC)

**I2C\_NACK\_INT\_RAW** The raw interrupt bit for the I2C\_NACK\_INT interrupt. (R/SS/WTC)

**I2C\_TXFIFO\_OVF\_INT\_RAW** The raw interrupt bit for the I2C\_TXFIFO\_OVF\_INT interrupt. (R/SS/WTC)

**I2C\_RXFIFO\_UDF\_INT\_RAW** The raw interrupt bit for the I2C\_RXFIFO\_UDF\_INT interrupt. (R/SS/WTC)

Continued on the next page...

**Register 21.20. I2C\_INT\_RAW\_REG (0x0020)**

Continued from the previous page...

**I2C\_SCL\_ST\_TO\_INT\_RAW** The raw interrupt bit for the I2C\_SCL\_ST\_TO\_INT interrupt. (R/SS/WTC)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_RAW** The raw interrupt bit for the I2C\_SCL\_MAIN\_ST\_TO\_INT interrupt.  
(R/SS/WTC)

**I2C\_DET\_START\_INT\_RAW** The raw interrupt bit for the I2C\_DET\_START\_INT interrupt. (R/SS/WTC)

## Register 21.21. I2C\_INT\_CLR\_REG (0x0024)

(reserved)																I2C_DET_START_INT_CLR I2C_SCL_MAIN_ST_TO_INT_CLR I2C_SCL_ST_TO_INT_CLR I2C_RXFIFO_UDF_INT_CLR I2C_TXFIFO_OVF_INT_CLR I2C_NACK_INT_CLR I2C_TRANS_START_INT_CLR I2C_TIME_OUT_INT_CLR I2C_TRANS_COMPLETE_INT_CLR I2C_MST_TXFIFO_UDF_INT_CLR I2C_ARBITRATION_LOST_INT_CLR I2C_BYTE_TRANS_DONE_INT_CLR I2C_END_DETECT_INT_CLR I2C_RXFIFO_OVF_INT_CLR I2C_RXFIFO_WM_INT_CLR																
31																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	Reset							

**I2C\_RXFIFO\_WM\_INT\_CLR** Set this bit to clear the I2C\_RXFIFO\_WM\_INT interrupt. (WT)

**I2C\_TXFIFO\_WM\_INT\_CLR** Set this bit to clear the I2C\_TXFIFO\_WM\_INT interrupt. (WT)

**I2C\_RXFIFO\_OVF\_INT\_CLR** Set this bit to clear the I2C\_RXFIFO\_OVF\_INT interrupt. (WT)

**I2C\_END\_DETECT\_INT\_CLR** Set this bit to clear the I2C\_END\_DETECT\_INT interrupt. (WT)

**I2C\_BYTE\_TRANS\_DONE\_INT\_CLR** Set this bit to clear the I2C\_BYTE\_TRANS\_DONE\_INT interrupt. (WT)

**I2C\_ARBITRATION\_LOST\_INT\_CLR** Set this bit to clear the I2C\_ARBITRATION\_LOST\_INT interrupt. (WT)

**I2C\_MST\_TXFIFO\_UDF\_INT\_CLR** Set this bit to clear the I2C\_MST\_TXFIFO\_UDF\_INT interrupt. (WT)

**I2C\_TRANS\_COMPLETE\_INT\_CLR** Set this bit to clear the I2C\_TRANS\_COMPLETE\_INT interrupt. (WT)

**I2C\_TIME\_OUT\_INT\_CLR** Set this bit to clear the I2C\_TIME\_OUT\_INT interrupt. (WT)

**I2C\_TRANS\_START\_INT\_CLR** Set this bit to clear the I2C\_TRANS\_START\_INT interrupt. (WT)

**I2C\_NACK\_INT\_CLR** Set this bit to clear the I2C\_NACK\_INT interrupt. (WT)

**I2C\_TXFIFO\_OVF\_INT\_CLR** Set this bit to clear the I2C\_TXFIFO\_OVF\_INT interrupt. (WT)

**I2C\_RXFIFO\_UDF\_INT\_CLR** Set this bit to clear the I2C\_RXFIFO\_UDF\_INT interrupt. (WT)

**I2C\_SCL\_ST\_TO\_INT\_CLR** Set this bit to clear the I2C\_SCL\_ST\_TO\_INT interrupt. (WT)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_CLR** Set this bit to clear the I2C\_SCL\_MAIN\_ST\_TO\_INT interrupt. (WT)

**I2C\_DET\_START\_INT\_CLR** Set this bit to clear the I2C\_DET\_START\_INT interrupt. (WT)

## Register 21.22. I2C\_INT\_ENA\_REG (0x0028)

(reserved)																I2C_DET_START_INT_ENA I2C_SCL_MAIN_ST_TO_INT_ENA I2C_SCL_ST_TO_INT_ENA I2C_RXFIFO_UDF_INT_ENA I2C_RXFIFO_OVF_INT_ENA I2C_NACK_INT_ENA I2C_TRANS_START_INT_ENA I2C_TIME_OUT_INT_ENA I2C_TRANS_COMPLETE_INT_ENA I2C_MST_TXFIFO_UDF_INT_ENA I2C_ARBITRATION_LOST_INT_ENA I2C_BYTE_TRANS_DONE_INT_ENA I2C_END_DETECT_INT_ENA I2C_RXFIFO_OVF_INT_ENA I2C_RXFIFO_WM_INT_ENA																
31																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	Reset							

**I2C\_RXFIFO\_WM\_INT\_ENA** The interrupt enable bit for the I2C\_RXFIFO\_WM\_INT interrupt. (R/W)

**I2C\_TXFIFO\_WM\_INT\_ENA** The interrupt enable bit for the I2C\_TXFIFO\_WM\_INT interrupt. (R/W)

**I2C\_RXFIFO\_OVF\_INT\_ENA** The interrupt enable bit for the I2C\_RXFIFO\_OVF\_INT interrupt. (R/W)

**I2C\_END\_DETECT\_INT\_ENA** The interrupt enable bit for the I2C\_END\_DETECT\_INT interrupt. (R/W)

**I2C\_BYTE\_TRANS\_DONE\_INT\_ENA** The interrupt enable bit for the I2C\_BYTE\_TRANS\_DONE\_INT interrupt. (R/W)

**I2C\_ARBITRATION\_LOST\_INT\_ENA** The interrupt enable bit for the I2C\_ARBITRATION\_LOST\_INT interrupt. (R/W)

**I2C\_MST\_TXFIFO\_UDF\_INT\_ENA** The interrupt enable bit for the I2C\_MST\_TXFIFO\_UDF\_INT interrupt. (R/W)

**I2C\_TRANS\_COMPLETE\_INT\_ENA** The interrupt enable bit for the I2C\_TRANS\_COMPLETE\_INT interrupt. (R/W)

**I2C\_TIME\_OUT\_INT\_ENA** The interrupt enable bit for the I2C\_TIME\_OUT\_INT interrupt. (R/W)

**I2C\_TRANS\_START\_INT\_ENA** The interrupt enable bit for the I2C\_TRANS\_START\_INT interrupt. (R/W)

**I2C\_NACK\_INT\_ENA** The interrupt enable bit for the I2C\_NACK\_INT interrupt. (R/W)

**I2C\_TXFIFO\_OVF\_INT\_ENA** The interrupt enable bit for the I2C\_TXFIFO\_OVF\_INT interrupt. (R/W)

**I2C\_RXFIFO\_UDF\_INT\_ENA** The interrupt enable bit for the I2C\_RXFIFO\_UDF\_INT interrupt. (R/W)

**I2C\_SCL\_ST\_TO\_INT\_ENA** The interrupt enable bit for the I2C\_SCL\_ST\_TO\_INT interrupt. (R/W)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_ENA** The interrupt enable bit for the I2C\_SCL\_MAIN\_ST\_TO\_INT interrupt. (R/W)

**I2C\_DET\_START\_INT\_ENA** The interrupt enable bit for the I2C\_DET\_START\_INT interrupt. (R/W)

Register 21.23. I2C\_INT\_STATUS\_REG (0x002C)

(reserved)																I2C_DET_START_INT_ST I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_ST_TO_INT_ST I2C_RXFIFO_UDF_INT_ST I2C_TXFIFO_UDF_INT_ST I2C_NACK_INT_ST I2C_TRANS_START_INT_ST I2C_TIME_OUT_INT_ST I2C_TRANS_COMPLETE_INT_ST I2C_MST_TXFIFO_UDF_INT_ST I2C_ARBTRATION_LOST_INT_ST I2C_BYTE_TRANS_DONE_INT_ST I2C_END_DETECT_INT_ST I2C_RXFIFO_OVF_INT_ST I2C_TXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT_ST																
31																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	Reset	

**I2C\_RXFIFO\_WM\_INT\_ST** The masked interrupt status bit for the I2C\_RXFIFO\_WM\_INT interrupt. (RO)

**I2C\_TXFIFO\_WM\_INT\_ST** The masked interrupt status bit for the I2C\_TXFIFO\_WM\_INT interrupt. (RO)

**I2C\_RXFIFO\_OVF\_INT\_ST** The masked interrupt status bit for the I2C\_RXFIFO\_OVF\_INT interrupt. (RO)

**I2C\_END\_DETECT\_INT\_ST** The masked interrupt status bit for the I2C\_END\_DETECT\_INT interrupt. (RO)

**I2C\_BYTE\_TRANS\_DONE\_INT\_ST** The masked interrupt status bit for the I2C\_BYTE\_TRANS\_DONE\_INT interrupt. (RO)

**I2C\_ARBTRATION\_LOST\_INT\_ST** The masked interrupt status bit for the I2C\_ARBTRATION\_LOST\_INT interrupt. (RO)

**I2C\_MST\_TXFIFO\_UDF\_INT\_ST** The masked interrupt status bit for the I2C\_MST\_TXFIFO\_UDF\_INT interrupt. (RO)

**I2C\_TRANS\_COMPLETE\_INT\_ST** The masked interrupt status bit for the I2C\_TRANS\_COMPLETE\_INT interrupt. (RO)

**I2C\_TIME\_OUT\_INT\_ST** The masked interrupt status bit for the I2C\_TIME\_OUT\_INT interrupt. (RO)

**I2C\_TRANS\_START\_INT\_ST** The masked interrupt status bit for the I2C\_TRANS\_START\_INT interrupt. (RO)

**I2C\_NACK\_INT\_ST** The masked interrupt status bit for the I2C\_NACK\_INT interrupt. (RO)

**I2C\_TXFIFO\_OVF\_INT\_ST** The masked interrupt status bit for the I2C\_TXFIFO\_OVF\_INT interrupt. (RO)

**I2C\_RXFIFO\_UDF\_INT\_ST** The masked interrupt status bit for the I2C\_RXFIFO\_UDF\_INT interrupt. (RO)

Continued on the next page...

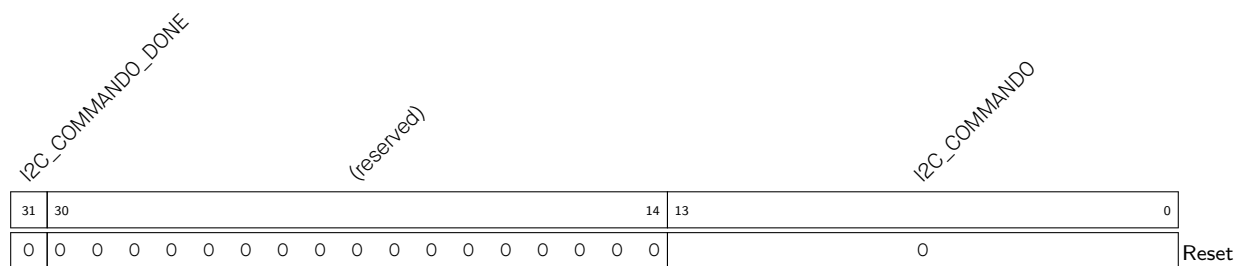
**Register 21.23. I2C\_INT\_STATUS\_REG (0x002C)**

Continued from the previous page...

**I2C\_SCL\_ST\_TO\_INT\_ST** The masked interrupt status bit for the I2C\_SCL\_ST\_TO\_INT interrupt. (RO)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_ST** The masked interrupt status bit for the I2C\_SCL\_MAIN\_ST\_TO\_INT interrupt. (RO)

**I2C\_DET\_START\_INT\_ST** The masked interrupt status bit for the I2C\_DET\_START\_INT interrupt. (RO)

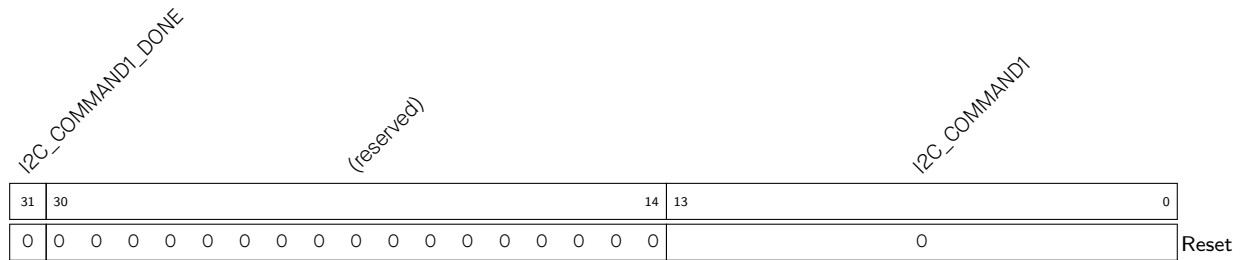
**Register 21.24. I2C\_COMDO\_REG (0x0058)**

**I2C\_COMMANDO** This is the content of command register 0. It consists of three parts:

- op\_code is the command. 1: WRITE; 2: STOP; 3: READ; 4: END; 6: RSTART.
- Byte\_num represents the number of bytes that need to be sent or received.
- ack\_check\_en, ack\_exp and ack are used to control the ACK bit. For more information, see Section [21.4.8](#).

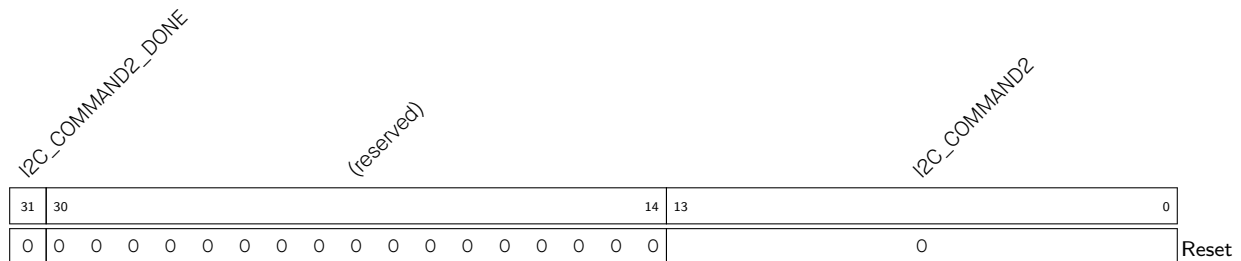
(R/W)

**I2C\_COMMANDO\_DONE** When command 0 has been executed, this bit changes to high level. (R/W/SS)

**Register 21.25. I2C\_COMD1\_REG (0x005C)**

**I2C\_COMMAND1** This is the content of command register 1. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND1\_DONE** When command 1 has been executed, this bit changes to high level. (R/W/SS)

**Register 21.26. I2C\_COMD2\_REG (0x0060)**

**I2C\_COMMAND2** This is the content of command register 2. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND2\_DONE** When command 2 has been executed, this bit changes to high Level. (R/W/SS)

**Register 21.27. I2C\_COMD3\_REG (0x0064)**

**I2C\_COMMAND3** This is the content of command register 3. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND3\_DONE** When command 3 has been executed, this bit changes to high level. (R/W/SS)

### Register 21.28. I2C\_COMMD4\_REG (0x0068)

I2C_COMMAND4_DONE																(reserved)																I2C_COMMAND4															
31	30														14	13														0																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														Reset																

**I2C\_COMMAND4** This is the content of command register 4. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND4\_DONE** When command 4 has been executed, this bit changes to high level.  
(R/W/SS)

### Register 21.29. I2C\_COMMD5\_REG (0x006C)

Diagram illustrating the structure of the I2C\_COMMAND5 register:

- Bit 31: I2C\_COMMAND5\_DONE
- Bits 14-31: (reserved)
- Bits 0-13: I2C\_COMMAND5

31	30	14	13	0
0	0	0	0	0

**I2C\_COMMAND5** This is the content of command register 5. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND5\_DONE** When command 5 has been executed, this bit changes to high level.  
(R/W/SS)

### Register 21.30. I2C\_COMMD6\_REG (0x0070)

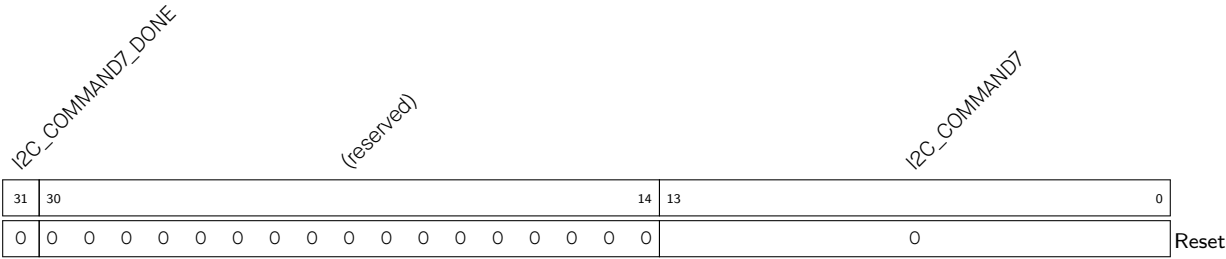
31		30																	14		13																	0	
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																	0	Reset	

**I2C\_COMMAND6** This is the content of command register 6. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND6\_DONE** When command 6 has been executed, this bit changes to high level.  
(R/W/SS)



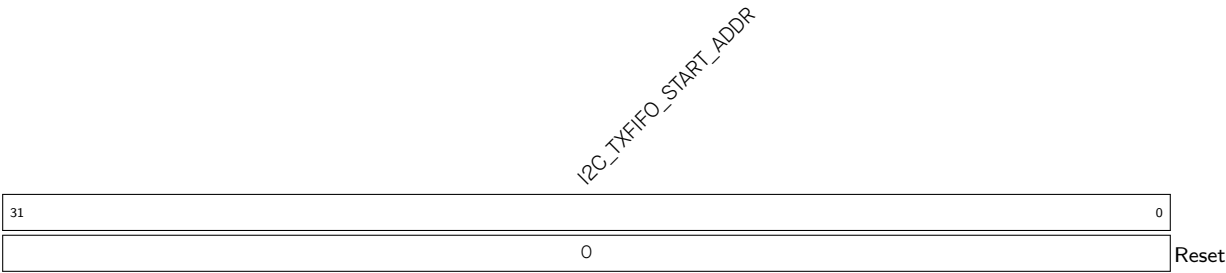
Register 21.31. I2C\_COMD7\_REG (0x0074)



**I2C\_COMMAND7** This is the content of command register 7. It is the same as that of [I2C\\_COMMAND0](#). (R/W)

**I2C\_COMMAND7\_DONE** When command 7 has been executed, this bit changes to high level. (R/W/SS)

Register 21.32. I2C\_TXFIFO\_START\_ADDR\_REG (0x0100)



**I2C\_TXFIFO\_START\_ADDR** This is the start address of I2C TX FIFO. (HRO)

Register 21.33. I2C\_RXFIFO\_START\_ADDR\_REG (0x0180)



**I2C\_RXFIFO\_START\_ADDR** This is the start address of I2C RX FIFO. (HRO)

Register 21.34. I2C\_DATE\_REG (0x00F8)

I2C_DATE	
31	0
0x20070201	
Reset	

**I2C\_DATE** This is the version control register. (R/W)

## Chapter 22

### LED PWM Controller (LEDC)

#### 22.1 Overview

The LED PWM Controller is a peripheral designed to generate PWM signals for LED control. It has specialized features such as automatic duty cycle fading. However, the LED PWM Controller can also be used to generate PWM signals for other purposes.

#### 22.2 Features

The LED PWM Controller has the following features:

- Six independent PWM generators (i.e. six channels)
- Maximum PWM duty cycle resolution: 14 bits
- Adjustable phase and duty cycle of PWM signal output
- PWM duty cycle dithering
- Automatic duty cycle fading — gradual increase/decrease of a PWM's duty cycle without interference from the processor. An interrupt will be generated upon fade completion
- PWM signal output in low-power mode (Light-sleep mode)
- Three clock sources that can be divided:
  - PLL\_F60M\_CLK
  - RC\_FAST\_CLK
  - XTAL\_CLK
- Four independent timers that support fractional division

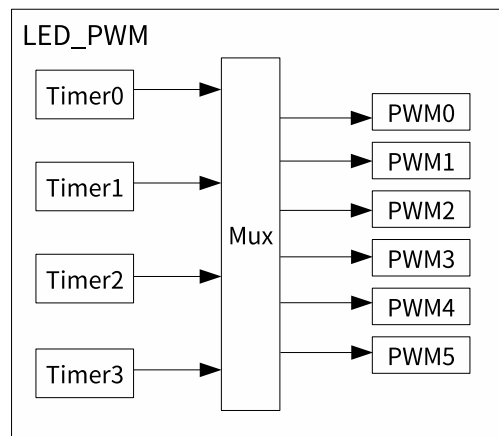


Figure 22.2-1. LED PWM Architecture

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer $x$  (where  $x$  ranges from 0 to 3). Likewise, the six PWM generators are also identical in features and operation, and thus are collectively referred to as PWM $n$  (where  $n$  ranges from 0 to 5).

## 22.3 Functional Description

### 22.3.1 Architecture

Figure 22.2-1 shows the architecture of the LED PWM Controller.

The four timers can be independently configured (i.e. each has a configurable clock divider, and counter overflow value) and each internally maintains a timebase counter (i.e. a counter that counts on cycles of a reference clock). Each PWM generator selects one of the timers and uses the timer's counter value as a reference to generate its PWM signal.

Figure 22.3-1 illustrates the main functional blocks of the timer and the PWM generator.

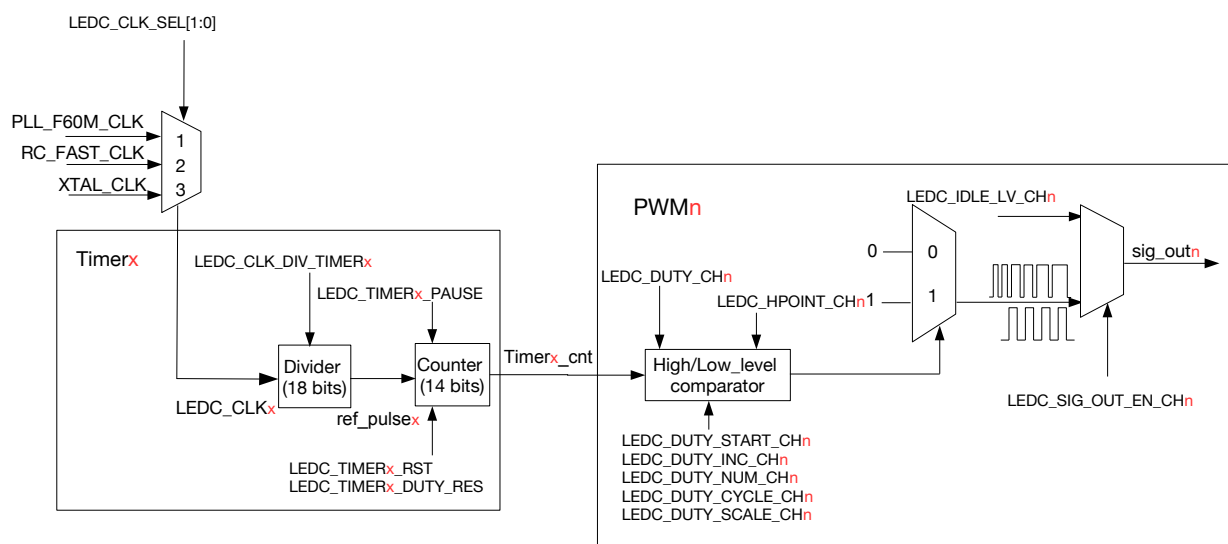


Figure 22.3-1. LED PWM Generator Diagram

### 22.3.2 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 22.3-1, this clock signal used by the timebase counter is named ref\_pulse $x$ . All timers use the same clock source LEDC\_CLK $x$ , which is then passed through a clock divider to generate ref\_pulse $x$  for the counter.

#### 22.3.2.1 Clock Source

LED PWM registers configured by software are clocked by APB\_CLK. For more information about APB\_CLK, see Chapter 6 *Reset and Clock*. To use the LED PWM peripheral, the APB\_CLK signal to the LED PWM has to be enabled. The APB\_CLK signal to LED PWM can be enabled by setting the `SYSTEM_LEDC_CLK_EN` field in the register `SYSTEM_PERIP_CLK_ENO_REG` and be reset via software by setting the `SYSTEM_LEDC_RST` field

in the register `SYSTEM_PERIP_RST_ENO_REG`. For more information, please refer to Table 13.3-1 in Chapter 13 *System Registers (SYSTEM)*.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: PLL\_F60M\_CLK, RC\_FAST\_CLK and XTAL\_CLK (see Chapter 6 *Reset and Clock* for more details about each clock signal). The procedure for selecting a clock source signal for LEDC\_CLK<sub>x</sub> is described below:

- PLL\_F60M\_CLK: Set `LEDC_CLK_SEL[1:0]` to 1
- RC\_FAST\_CLK: Set `LEDC_CLK_SEL[1:0]` to 2
- XTAL\_CLK: Set `LEDC_CLK_SEL[1:0]` to 3

The LEDC\_CLK<sub>x</sub> signal will then be passed through the clock divider.

### 22.3.2.2 Clock Divider Configuration

The LEDC\_CLK<sub>x</sub> signal is passed through a clock divider to generate the ref\_pulse<sub>x</sub> signal for the counter. The frequency of ref\_pulse<sub>x</sub> is equal to the frequency of LEDC\_CLK<sub>x</sub> divided by the divisor LEDC\_CLK\_DIV (see Figure 22.3-1).

The divisor LEDC\_CLK\_DIV is a fractional value. Thus, it can be a non-integer divisor for finer granularity of available frequencies. LEDC\_CLK\_DIV is configured according to the following equation.

$$LEDC\_CLK\_DIV = A + \frac{B}{256}$$

- A corresponds to the most significant 10 bits of `LEDC_CLK_DIV_TIMERx` (i.e. `LEDC_TIMERx_CONF_REG[21:12]`)
- The fractional part B corresponds to the least significant 8 bits of `LEDC_CLK_DIV_TIMERx` (i.e. `LEDC_TIMERx_CONF_REG[11:4]`)

When the fractional part B is zero, LEDC\_CLK\_DIV is equivalent to an integer divisor (i.e. an integer prescaler). In other words, a ref\_pulse<sub>x</sub> clock pulse is generated after every A number of LEDC\_CLK<sub>x</sub> clock pulses.

However, when B is nonzero, LEDC\_CLK\_DIV becomes a non-integer divisor. The clock divider implements non-integer frequency division by alternating between A and (A+1) LEDC\_CLK<sub>x</sub> clock pulses per ref\_pulse<sub>x</sub> clock pulse. This will result in the average frequency of ref\_pulse<sub>x</sub> clock pulse being the desired frequency (i.e. the non-integer divided frequency). For every 256 ref\_pulse<sub>x</sub> clock pulses:

- a number of B ref\_pulse<sub>x</sub> clock pulses will have duration of (A+1) LEDC\_CLK<sub>x</sub> clock pulses
- a number of (256-B) ref\_pulse<sub>x</sub> clock pulses will have duration of A LEDC\_CLK<sub>x</sub> clock pulses
- the ref\_pulse<sub>x</sub> clock pulses with duration of (A+1) pulses are evenly distributed amongst those with duration of A pulses

Figure 22.3-2 illustrates the relation between LEDC\_CLK<sub>x</sub> clock pulses and ref\_pulse<sub>x</sub> clock pulses when dividing by a non-integer LEDC\_CLK\_DIV.

To change the timer's clock divisor at runtime, first configure the `LEDC_CLK_DIV_TIMERx` field, and then set the `LEDC_TIMERx_PARA_UP` field to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. The `LEDC_TIMERx_PARA_UP` field will be automatically cleared by hardware.

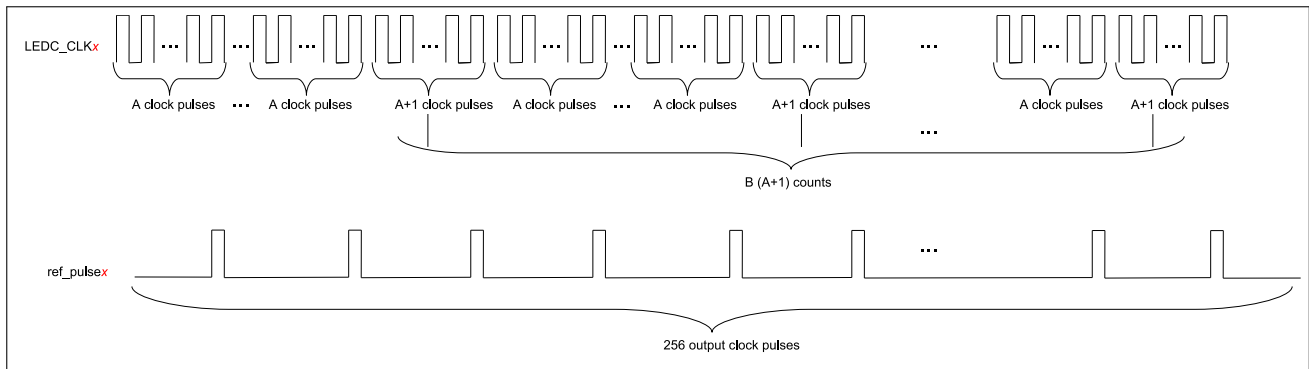


Figure 22.3-2. Frequency Division When LEDC\_CLK\_DIV is a Non-Integer Value

### 22.3.2.3 14-bit Counter

Each timer contains a 14-bit timebase counter that uses `ref_pulsex` as its reference clock (see Figure 22.3-1). The `LEDC_TIMERx_DUTY_RES` field configures the overflow value of this 14-bit counter. Hence, the maximum resolution of the PWM duty cycle is 14 bits. The counter counts up to  $(2^{\text{LEDC\_TIMERx\_DUTY\_RES}} - 1)$ , overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software.

The counter can trigger `LEDC_TIMERx_OVF_INT` interrupt (generated automatically by hardware without configuration) every time the counter overflows. It can also be configured to trigger `LEDC_OVF_CNT_CHn_INT` interrupt after the counter overflows  $(\text{LEDC\_OVF\_NUM\_CHn} + 1)$  times. To configure `LEDC_OVF_CNT_CHn_INT` interrupt, please:

1. Configure `LEDC_TIMER_SEL_CHn` as the counter for the PWM generator
2. Enable the counter by setting `LEDC_OVF_CNT_EN_CHn`
3. Set `LEDC_OVF_NUM_CHn` to the number of counter overflows to generate an interrupt, minus 1
4. Enable the overflow interrupt by setting `LEDC_OVF_CNT_CHn_INT_ENA`
5. Set `LEDC_TIMERx_DUTY_RES` to enable the timer and wait for a `LEDC_OVF_CNT_CHn_INT` interrupt

Referring to Figure 22.3-1, the frequency of a PWM generator output signal (`sig_outn`) is dependent on the frequency of the timer's clock source `LEDC_CLKx`, the clock divisor `LEDC_CLK_DIV`, and the duty resolution (counter width) `LEDC_TIMERx_DUTY_RES`:

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLKx}}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMERx\_DUTY\_RES}}}$$

Based on the formula above, the desired duty resolution can be calculated as follows:

$$\text{LEDC\_TIMERx\_DUTY\_RES} = \log_2 \left( \frac{f_{\text{LEDC\_CLKx}}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}} \right)$$

Table 22.3-1 lists the commonly-used frequencies and their corresponding resolutions.

**Table 22.3-1. Commonly-used Frequencies and Resolutions**

LEDC_CLK <sub>x</sub>	PWM Frequency	Highest Resolution (bit) <sup>1</sup>	Lowest Resolution (bit) <sup>2</sup>
PLL_F60M_CLK (60 MHz)	1 kHz	14	6
PLL_F60M_CLK (60 MHz)	5 kHz	13	4
PLL_F60M_CLK (60 MHz)	10 kHz	12	3
XTAL_CLK (40 MHz)	1 kHz	14	6
XTAL_CLK (40 MHz)	4 kHz	13	4
RC_FAST_CLK (17.5 MHz)	1 kHz	14	5
RC_FAST_CLK (17.5 MHz)	1.75 kHz	13	4

<sup>1</sup> The highest resolution is calculated when the clock divisor LEDC\_CLK\_DIV is 1 and rounded down. If the highest resolution calculated by the formula is higher than the counter's width 14 bits, then the highest resolution should be 14 bits.

<sup>2</sup> The lowest resolution is calculated when the clock divisor LEDC\_CLK\_DIV 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.

To change the overflow value at runtime, first set the `LEDC_TIMERx_DUTY_RES` field, and then set the `LEDC_TIMERx_PARA_UP` field. This will cause the newly configured values to take effect upon the next overflow of the counter. If `LEDC_OVF_CNT_EN_CHn` field is reconfigured, `LEDC_PARA_UP_CHn` should be set to apply the new configuration. In summary, these configuration values need to be updated by setting `LEDC_TIMERx_PARA_UP` or `LEDC_PARA_UP_CHn`. `LEDC_TIMERx_PARA_UP` and `LEDC_PARA_UP_CHn` will be automatically cleared by hardware.

### 22.3.3 PWM Generators

To generate a PWM signal, a PWM generator (PWM<sub>n</sub>) selects a timer (Timer<sub>x</sub>). Each PWM generator can be configured separately by setting `LEDC_TIMER_SEL_CHn` to use one of four timers to generate the PWM output.

As shown in Figure 22.3-1, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 14-bit counter value (Timer<sub>x</sub>\_cnt) to two trigger values Hpoint<sub>n</sub> and Lpoint<sub>n</sub>. When the timer's counter value is equal to Hpoint<sub>n</sub> or Lpoint<sub>n</sub>, the PWM signal is high or low, respectively, as described below:

- If Timer<sub>x</sub>\_cnt == Hpoint<sub>n</sub>, sig\_out<sub>n</sub> is 1.
- If Timer<sub>x</sub>\_cnt == Lpoint<sub>n</sub>, sig\_out<sub>n</sub> is 0.

Figure 22.3-3 illustrates how Hpoint<sub>n</sub> or Lpoint<sub>n</sub> are used to generate a fixed duty cycle PWM output signal.

For a particular PWM generator (PWM<sub>n</sub>), its Hpoint<sub>n</sub> is sampled from the `LEDC_HPOINT_CHn` field each time the selected timer's counter overflows. Likewise, Lpoint<sub>n</sub> is also sampled on every counter overflow and is calculated from the sum of the `LEDC_DUTY_CHn[18:4]` and `LEDC_HPOINT_CHn` fields. By setting Hpoint<sub>n</sub> and



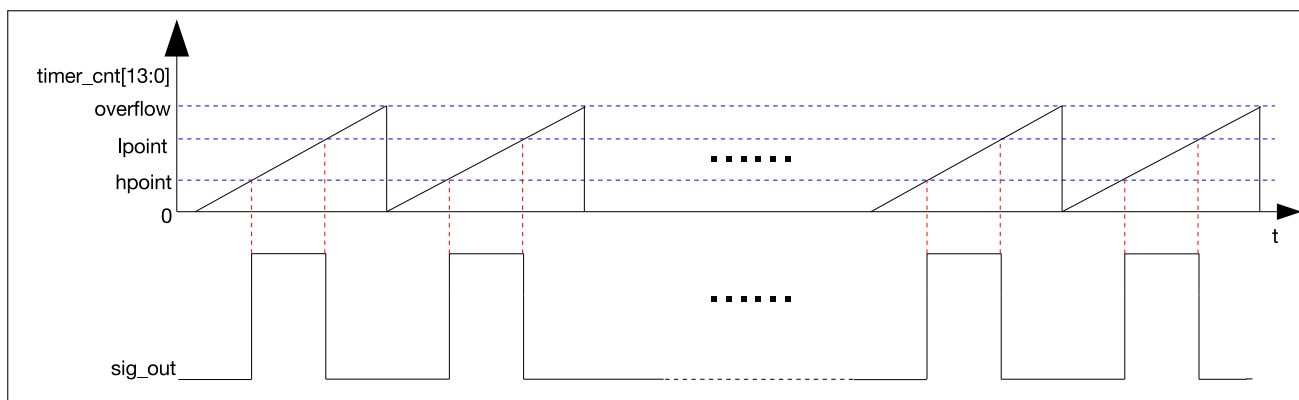


Figure 22.3-3. LED\_PWM Output Signal Diagram

Lpoint $n$  via the LEDC\_HPOINT\_CH $n$  and LEDC\_DUTY\_CH $n$ [18:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig\_out $n$ ) is enabled by setting LEDC\_SIG\_OUT\_EN\_CH $n$ . When LEDC\_SIG\_OUT\_EN\_CH $n$  is cleared, PWM signal output is disabled, and the output signal (sig\_out $n$ ) will output a constant level as specified by LEDC\_IDLE\_LV\_CH $n$ .

The bits LEDC\_DUTY\_CH $n$ [3:0] are used to dither the duty cycles of the PWM output signal (sig\_out $n$ ) by periodically altering the duty cycle of sig\_out $n$ . When LEDC\_DUTY\_CH $n$ [3:0] is set to a non-zero value, then for every 16 cycles of sig\_out $n$ , LEDC\_DUTY\_CH $n$ [3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC\_DUTY\_CH $n$ [3:0]) cycles. For instance, if LEDC\_DUTY\_CH $n$ [18:4] is set to 10 and LEDC\_DUTY\_CH $n$ [3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC\_TIMER\_SEL\_CH $n$ , LEDC\_HPOINT\_CH $n$ , LEDC\_DUTY\_CH $n$ [18:4] and LEDC\_SIG\_OUT\_EN\_CH $n$  are reconfigured, LEDC\_PARA\_UP\_CH $n$  must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC\_PARA\_UP\_CH $n$  field will be automatically cleared by hardware.

### 22.3.4 Duty Cycle Fading

The PWM generators can fade the duty cycle of a PWM output signal (i.e. gradually change the duty cycle from one value to another). If Duty Cycle Fading is enabled, the value of Lpoint $n$  will be incremented/decremented after a fixed number of counter overflows has occurred. Figure 22.3-4 illustrates Duty Cycle Fading.

Duty Cycle Fading is configured using the following register fields:

- LEDC\_DUTY\_CH $n$  is used to set the initial value of Lpoint $n$ .
- LEDC\_DUTY\_START\_CH $n$  will enable/disable duty cycle fading when set/cleared.
- LEDC\_DUTY\_CYCLE\_CH $n$  sets the number of counter overflow cycles for every Lpoint $n$  increment/decrement. In other words, Lpoint $n$  will be incremented/decremented after LEDC\_DUTY\_CYCLE\_CH $n$  counter overflows.
- LEDC\_DUTY\_INC\_CH $n$  configures whether Lpoint $n$  is incremented/decremented if set/cleared.

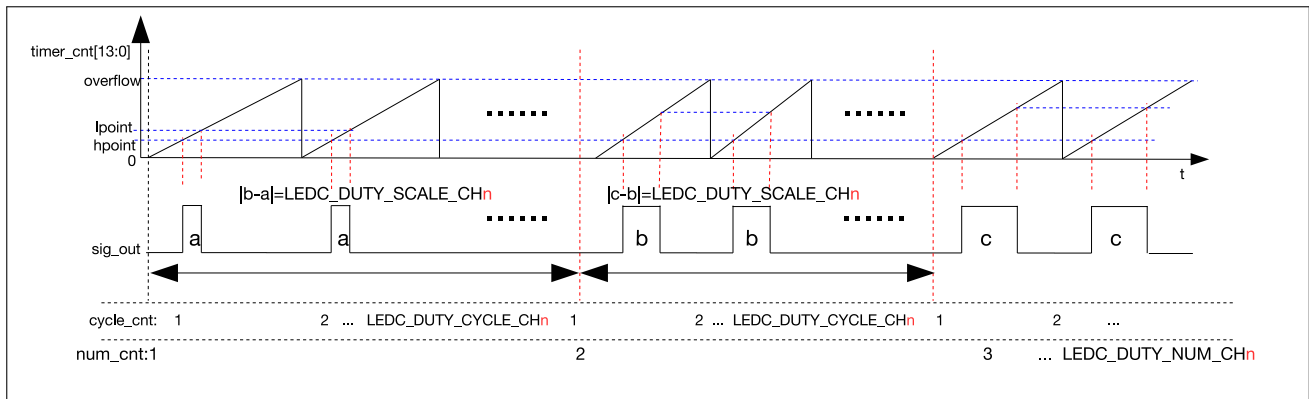


Figure 22.3-4. Output Signal Diagram of Fading Duty Cycle

- `LEDC_DUTY_SCALE_CHn` sets the amount that `Lpointn` is incremented/decremented.
- `LEDC_DUTY_NUM_CHn` sets the maximum number of increments/decrements before duty cycle fading stops.

If the fields `LEDC_DUTY_CHn`, `LEDC_DUTY_START_CHn`, `LEDC_DUTY_CYCLE_CHn`, `LEDC_DUTY_INC_CHn`, `LEDC_DUTY_SCALE_CHn`, and `LEDC_DUTY_NUM_CHn` are reconfigured, `LEDC_PARA_UP_CHn` must be set to apply the new configuration. After this field is set, the values for duty cycle fading will take effect at once. `LEDC_PARA_UP_CHn` field will be automatically cleared by hardware.

### 22.3.5 Interrupts

- `LEDC_OVF_CNT_CHn_INT`: Triggered when the timer counter overflows for  $(\text{LEDC\_OVF\_NUM\_CHn} + 1)$  times and the register `LEDC_OVF_CNT_EN_CHn` is set to 1.
- `LEDC_DUTY_CHNG_END_CHn_INT`: Triggered when a fade on an LED PWM generator has finished.
- `LEDC_TIMERx_OVF_INT`: Triggered when an LED PWM timer has reached its maximum counter value.

## 22.4 Register Summary

The addresses in this section are relative to **LED PWM Controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Configuration Register</b>			
<a href="#">LEDC_CHO_CONFO_REG</a>	Configuration register 0 for channel 0	0x0000	varies
<a href="#">LEDC_CHO_CONF1_REG</a>	Configuration register 1 for channel 0	0x000C	varies
<a href="#">LEDC_CH1_CONFO_REG</a>	Configuration register 0 for channel 1	0x0014	varies
<a href="#">LEDC_CH1_CONF1_REG</a>	Configuration register 1 for channel 1	0x0020	varies
<a href="#">LEDC_CH2_CONFO_REG</a>	Configuration register 0 for channel 2	0x0028	varies
<a href="#">LEDC_CH2_CONF1_REG</a>	Configuration register 1 for channel 2	0x0034	varies
<a href="#">LEDC_CH3_CONFO_REG</a>	Configuration register 0 for channel 3	0x003C	varies
<a href="#">LEDC_CH3_CONF1_REG</a>	Configuration register 1 for channel 3	0x0048	varies
<a href="#">LEDC_CH4_CONFO_REG</a>	Configuration register 0 for channel 4	0x0050	varies
<a href="#">LEDC_CH4_CONF1_REG</a>	Configuration register 1 for channel 4	0x005C	varies
<a href="#">LEDC_CH5_CONFO_REG</a>	Configuration register 0 for channel 5	0x0064	varies
<a href="#">LEDC_CH5_CONF1_REG</a>	Configuration register 1 for channel 5	0x0070	varies
<a href="#">LEDC_CONF_REG</a>	Global ledc configuration register	0x00D0	R/W
<b>Hpoint Register</b>			
<a href="#">LEDC_CHO_HPOINT_REG</a>	High point register for channel 0	0x0004	R/W
<a href="#">LEDC_CH1_HPOINT_REG</a>	High point register for channel 1	0x0018	R/W
<a href="#">LEDC_CH2_HPOINT_REG</a>	High point register for channel 2	0x002C	R/W
<a href="#">LEDC_CH3_HPOINT_REG</a>	High point register for channel 3	0x0040	R/W
<a href="#">LEDC_CH4_HPOINT_REG</a>	High point register for channel 4	0x0054	R/W
<a href="#">LEDC_CH5_HPOINT_REG</a>	High point register for channel 5	0x0068	R/W
<b>Duty Cycle Register</b>			
<a href="#">LEDC_CHO_DUTY_REG</a>	Initial duty cycle for channel 0	0x0008	R/W
<a href="#">LEDC_CHO_DUTY_R_REG</a>	Current duty cycle for channel 0	0x0010	RO
<a href="#">LEDC_CH1_DUTY_REG</a>	Initial duty cycle for channel 1	0x001C	R/W
<a href="#">LEDC_CH1_DUTY_R_REG</a>	Current duty cycle for channel 1	0x0024	RO
<a href="#">LEDC_CH2_DUTY_REG</a>	Initial duty cycle for channel 2	0x0030	R/W
<a href="#">LEDC_CH2_DUTY_R_REG</a>	Current duty cycle for channel 2	0x0038	RO
<a href="#">LEDC_CH3_DUTY_REG</a>	Initial duty cycle for channel 3	0x0044	R/W
<a href="#">LEDC_CH3_DUTY_R_REG</a>	Current duty cycle for channel 3	0x004C	RO
<a href="#">LEDC_CH4_DUTY_REG</a>	Initial duty cycle for channel 4	0x0058	R/W
<a href="#">LEDC_CH4_DUTY_R_REG</a>	Current duty cycle for channel 4	0x0060	RO
<a href="#">LEDC_CH5_DUTY_REG</a>	Initial duty cycle for channel 5	0x006C	R/W
<a href="#">LEDC_CH5_DUTY_R_REG</a>	Current duty cycle for channel 5	0x0074	RO
<b>Timer Register</b>			
<a href="#">LEDC_TIMER0_CONF_REG</a>	Timer 0 configuration	0x00A0	varies
<a href="#">LEDC_TIMER0_VALUE_REG</a>	Timer 0 current counter value	0x00A4	RO

Name	Description	Address	Access
<a href="#">LEDC_TIMER1_CONF_REG</a>	Timer 1 configuration	0x00A8	varies
<a href="#">LEDC_TIMER1_VALUE_REG</a>	Timer 1 current counter value	0x00AC	RO
<a href="#">LEDC_TIMER2_CONF_REG</a>	Timer 2 configuration	0x00B0	varies
<a href="#">LEDC_TIMER2_VALUE_REG</a>	Timer 2 current counter value	0x00B4	RO
<a href="#">LEDC_TIMER3_CONF_REG</a>	Timer 3 configuration	0x00B8	varies
<a href="#">LEDC_TIMER3_VALUE_REG</a>	Timer 3 current counter value	0x00BC	RO
<b>Interrupt Register</b>			
<a href="#">LEDC_INT_RAW_REG</a>	Raw interrupt status	0x00C0	R/WTC/SS
<a href="#">LEDC_INT_ST_REG</a>	Masked interrupt status	0x00C4	RO
<a href="#">LEDC_INT_ENA_REG</a>	Interrupt enable bits	0x00C8	R/W
<a href="#">LEDC_INT_CLR_REG</a>	Interrupt clear bits	0x00CC	WT
<b>Version Register</b>			
<a href="#">LEDC_DATE_REG</a>	Version control register	0x00FC	R/W

## 22.5 Registers

The addresses in this section are relative to **LED PWM Controller** base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 22.1. LEDC\_CH $n$ \_CONF0\_REG ( $n$ : 0-5) (0x0000+0x14\* $n$ )**

(reserved)																LEDC_OVF_CNT_RESET_CH <sup>n</sup> LEDC_OVF_CNT_EN_CH <sup>n</sup>				LEDC_OVF_NUM_CH <sup>n</sup>				LEDC_PARA_UP_CH <sup>n</sup> LEDC_IDLE_LV_CH <sup>n</sup> LEDC_SIG_OUT_EN_CH <sup>n</sup> LEDC_TIMER_SEL_CH <sup>n</sup>				
31															17	16	15	14					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	

Reset

**LEDC\_TIMER\_SEL\_CH $n$**  This field is used to select one of timers for channel  $n$ .

0: select Timer 0; 1: select Timer 1; 2: select Timer 2; 3: select Timer 3 (R/W)

**LEDC\_SIG\_OUT\_EN\_CH $n$**  Set this bit to enable signal output on channel  $n$ . (R/W)

**LEDC\_IDLE\_LV\_CH $n$**  This bit is used to control the output value when channel  $n$  is inactive (when LEDC\_SIG\_OUT\_EN\_CH $n$  is 0). (R/W)

**LEDC\_PARA\_UP\_CH $n$**  This bit is used to update the listed fields for channel  $n$ , and will be automatically cleared by hardware. (WT)

- LEDC\_HPOINT\_CH $n$
- LEDC\_DUTY\_START\_CH $n$
- LEDC\_SIG\_OUT\_EN\_CH $n$
- LEDC\_TIMER\_SEL\_CH $n$
- LEDC\_DUTY\_NUM\_CH $n$
- LEDC\_DUTY\_CYCLE\_CH $n$
- LEDC\_DUTY\_SCALE\_CH $n$
- LEDC\_DUTY\_INC\_CH $n$
- LEDC\_OVF\_CNT\_EN\_CH $n$

**LEDC\_OVF\_NUM\_CH $n$**  This field is used to configure the number of counter overflows to generate an interrupt minus 1. The LEDC\_OVF\_CNT\_CH $n$ \_INT interrupt will be triggered when channel  $n$  overflows for (LEDC\_OVF\_NUM\_CH $n$  + 1) times. (R/W)

Continued on the next page...

**Register 22.1. LEDC\_CH $n$ \_CONF0\_REG ( $n$ : 0-5) (0x0000+0x14\* $n$ )**

Continued from the previous page...

**LEDC\_OVF\_CNT\_EN\_CH $n$**  This bit is used to enable the counter that counts the number of times when the timer selected by channel  $n$  overflows. (R/W)

**LEDC\_OVF\_CNT\_RESET\_CH $n$**  Set this bit to reset the timer-overflow counter of channel  $n$ . (WT)

**Register 22.2. LEDC\_CH $n$ \_CONF1\_REG ( $n$ : 0-5) (0x000C+0x14\* $n$ )**

LEDC_DUTY_START_CH $n$ LEDC_DUTY_INC_CH $n$		LEDC_DUTY_NUM_CH $n$		LEDC_DUTY_CYCLE_CH $n$		LEDC_DUTY_SCALE_CH $n$	
31	30	29	20	19	10	9	0
0	1	0x0	0x0	0x0	0x0	0x0	Reset

**LEDC\_DUTY\_SCALE\_CH $n$**  This field configures the step size of the duty cycle change during fading. (R/W)

**LEDC\_DUTY\_CYCLE\_CH $n$**  The duty will change every LEDC\_DUTY\_CYCLE\_CH $n$  on channel  $n$ . (R/W)

**LEDC\_DUTY\_NUM\_CH $n$**  This field sets the maximum number of increments/decrements before duty cycle fading stops. (R/W)

**LEDC\_DUTY\_INC\_CH $n$**  This bit determines whether the duty cycle of the output signal on channel  $n$  increases or decreases. 1: Increase; 0: Decrease. (R/W)

**LEDC\_DUTY\_START\_CH $n$**  If this bit is set to 1, other configured fields in LEDC\_CH $n$ \_CONF1\_REG will take effect upon the next timer overflow.(R/W/SC)

### Register 22.3. LEDC\_CONF\_REG (0x00D0)

[illegible]

**LEDC\_CLK\_SEL** This field is used to select the common clock source for all the 4 timers.

1: PLL\_60M\_CLK; 2: RC\_FAST\_CLK; 3: XTAL\_CLK. (R/W)

**LEDC\_CLK\_EN** This bit is used to control the clock.

1: Force clock on for register. 0: Support clock only when application writes registers. (R/W)

Register 22.4. LEDC\_CH $n$ \_HPOINT\_REG ( $n$ : 0-5) (0x0004+0x14\* $n$ )

(reserved)																LEDC_HPOINT_CH <sup>n</sup>																																															
31																14																13																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x00																Reset																															

**LEDC\_HPOINT\_CH<sub>n</sub>** The output value changes to high when the selected timer for this channel has reached the value specified by this field. (R/W)

Register 22.5. LEDC\_CH $n$ \_DUTY\_REG ( $n$ : 0-5) (0x0008+0x14\* $n$ )

Register layout for LEDC\_DUTY\_CHn:

31	19	18	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0			
0x000			

Reset

**LEDC\_DUTY\_CH<sub>n</sub>** This field is used to change the output duty by controlling the Lpoint. The output value turns to low when the selected timer for this channel has reached the Lpoint. (R/W)

Register 22.6. LEDC\_CH $n$ \_DUTY\_R\_REG ( $n$ : 0-5) (0x0010+0x14\* $n$ )

(reserved)														LEDC_DUTY_R_CH <sup>n</sup>																															
31														19														18																	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0														0x000																	Reset														

**LEDC\_DUTY\_R\_CH $n$**  This field stores the current duty cycle of the output signal on channel  $n$ . (RO)

Register 22.7. LEDC\_TIMER $x$ \_CONF\_REG ( $x$ : 0-3) (0x00A0+0x8\* $x$ )

(reserved)							LEDC_TIMER <del>x</del> _PARA_UP					(reserved)					LEDC_TIMER <del>x</del> _RST					LEDC_TIMER <del>x</del> _PAUSE					LEDC_CLK_DIV_TIMER <del>x</del>										LEDC_TIMER <del>x</del> _DUTY_RES			
31							26	25	24	23	22	21											4	3				0												
0	0	0	0	0	0	0	0	0	1	0	0x000												0x0			Reset														

**LEDC\_TIMER $x$ \_DUTY\_RES** This field is used to control the range of the counter in timer  $x$ . (R/W)

**LEDC\_CLK\_DIV\_TIMER $x$**  This field is used to configure the divisor for the divider in timer  $x$ . The least significant eight bits represent the fractional part. (R/W)

**LEDC\_TIMER $x$ \_PAUSE** This bit is used to suspend the counter in timer  $x$ . (R/W)

**LEDC\_TIMER $x$ \_RST** This bit is used to reset timer  $x$ . The counter will show 0 after reset. (R/W)

**LEDC\_TIMER $x$ \_PARA\_UP** Set this bit to update LEDC\_CLK\_DIV\_TIMER $x$  and LEDC\_TIMER $x$ \_DUTY\_RES. (WT)

Register 22.8. LEDC\_TIMER $x$ \_VALUE\_REG ( $x$ : 0-3) (0x00A4+0x8\* $x$ )

(reserved)														LEDC_TIMER <sub>x</sub> CNT																															
31														14														13																	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														0																	Reset														

**LEDC\_TIMER $x$ \_CNT** This field stores the current counter value of timer  $x$ . (RO)



Register 22.9. LEDC\_INT\_RAW\_REG (0x00C0)

(reserved)																LEDC_OVF_CNT_CH5_INT_RAW LEDC_OVF_CNT_CH4_INT_RAW LEDC_OVF_CNT_CH3_INT_RAW LEDC_OVF_CNT_CH2_INT_RAW LEDC_OVF_CNT_CH1_INT_RAW LEDC_DUTY_CHNG_END_CH5_INT_RAW LEDC_DUTY_CHNG_END_CH4_INT_RAW LEDC_DUTY_CHNG_END_CH3_INT_RAW LEDC_DUTY_CHNG_END_CH2_INT_RAW LEDC_DUTY_CHNG_END_CH1_INT_RAW LEDC_TIMER3_OVF_INT_RAW LEDC_TIMER2_OVF_INT_RAW LEDC_TIMER1_OVF_INT_RAW LEDC_TIMER0_OVF_INT_RAW																
31																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	Reset							

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_RAW** The raw interrupt status of LEDC\_TIMER<sub>x</sub>\_OVF\_INT. (R/WTC/SS)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_RAW** The raw interrupt status of LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT. (R/WTC/SS)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_RAW** The raw interrupt status of LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT. (R/WTC/SS)

Register 22.10. LEDC\_INT\_ST\_REG (0x00C4)

<div>(reserved)</div>																																<div>LEDC_OVF_CNT_CH5_INT_ST LEDC_OVF_CNT_CH4_INT_ST LEDC_OVF_CNT_CH3_INT_ST LEDC_OVF_CNT_CH2_INT_ST LEDC_OVF_CNT_CH1_INT_ST LEDC_DUTY_CHNG_END_CH5_INT_ST LEDC_DUTY_CHNG_END_CH4_INT_ST LEDC_DUTY_CHNG_END_CH3_INT_ST LEDC_DUTY_CHNG_END_CH2_INT_ST LEDC_DUTY_CHNG_END_CH1_INT_ST LEDC_TIMER3_OVF_INT_ST LEDC_TIMER2_OVF_INT_ST LEDC_TIMER1_OVF_INT_ST LEDC_TIMER0_OVF_INT_ST</div>																
31																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	Reset

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_ST** This is the masked interrupt status bit for the LEDC\_TIMER<sub>x</sub>\_OVF\_INT interrupt when LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_ENA is set to 1. (RO)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_ST** This is the masked interrupt status bit for the LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT interrupt when LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_ENA is set to 1. (RO)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_ST** This is the masked interrupt status bit for the LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT interrupt when LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_ENA is set to 1. (RO)

Register 22.11. LEDC\_INT\_ENA\_REG (0x00C8)

(reserved)																LEDC_OVF_CNT_CH5_INT_ENA LEDC_OVF_CNT_CH4_INT_ENA LEDC_OVF_CNT_CH3_INT_ENA LEDC_OVF_CNT_CH2_INT_ENA LEDC_OVF_CNT_CH1_INT_ENA LEDC_DUTY_CHNG_END_CH5_INT_ENA LEDC_DUTY_CHNG_END_CH4_INT_ENA LEDC_DUTY_CHNG_END_CH3_INT_ENA LEDC_DUTY_CHNG_END_CH2_INT_ENA LEDC_TIMER3_OVF_INT_ENA LEDC_TIMER2_OVF_INT_ENA LEDC_TIMER1_OVF_INT_ENA LEDC_TIMER0_OVF_INT_ENA																
31																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	Reset						

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_ENA** The interrupt enable bit for the LEDC\_TIMER<sub>x</sub>\_OVF\_INT interrupt. (R/W)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_ENA** The interrupt enable bit for the LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT interrupt. (R/W)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_ENA** The interrupt enable bit for the LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT interrupt. (R/W)

Register 22.12. LEDC\_INT\_CLR\_REG (0x00CC)

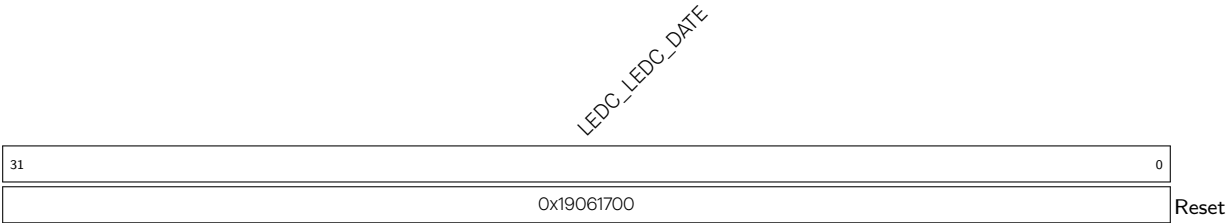
(reserved)																																LEDC_OVF_CNT_CH5_INT_CLR LEDC_OVF_CNT_CH4_INT_CLR LEDC_OVF_CNT_CH3_INT_CLR LEDC_OVF_CNT_CH2_INT_CLR LEDC_OVF_CNT_CH1_INT_CLR LEDC_DUTY_CHNG_END_CH0_INT_CLR LEDC_DUTY_CHNG_END_CH5_INT_CLR LEDC_DUTY_CHNG_END_CH4_INT_CLR LEDC_DUTY_CHNG_END_CH3_INT_CLR LEDC_DUTY_CHNG_END_CH2_INT_CLR LEDC_TIMER3_OVF_INT_CLR LEDC_TIMER2_OVF_INT_CLR LEDC_TIMER1_OVF_INT_CLR LEDC_TIMER0_OVF_INT_CLR																	
31																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	Reset

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_CLR** Set this bit to clear the LEDC\_TIMER<sub>x</sub>\_OVF\_INT interrupt. (WT)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_CLR** Set this bit to clear the LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT interrupt. (WT)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_CLR** Set this bit to clear the LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT interrupt. (WT)

Register 22.13. LEDC\_DATE\_REG (0x00FC)



**LEDC\_LEDC\_DATE** This is the version control register. (R/W)

## Part VI

# Analog Signal Processing

This part describes components related to analog-to-digital conversion, on-chip sensors, and features such as temperature sensing, demonstrating the system's capabilities in handling analog signals.

## Chapter 23

# On-Chip Sensor and Analog Signal Processing

## 23.1 Overview

ESP8684 provides the following analog signal processing peripheral and on-chip sensor:

- One 12-bit Successive Approximation ADC (SAR ADC) for measuring analog signals from five channels.
- One temperature sensor for measuring the internal temperature of the ESP8684 chip.

## 23.2 SAR ADC

### 23.2.1 Overview

ESP8684 integrates one 12-bit SAR ADC, which is able to measure analog signals from up to five pins. The SAR ADC is managed by DIG ADC controller, which drives [Digital\\_Reader](#) to sample channel voltages of SAR ADC. This controller supports multi-channel scanning and threshold monitoring.

### 23.2.2 Features

SAR ADC has the following features:

- One ADC Reader module ([Digital\\_Reader](#)) to read sampling results
- 12-bit sampling resolution
- Able to sample the analog voltages from up to five pins
- One DIG ADC controller
  - Provides separate control modules for one-time sampling and multi-channel scanning
  - Supports one-time sampling and multi-channel scanning working simultaneously
  - User-defined scanning sequence in multi-channel scanning mode
  - Provides two filters with configurable filter coefficient
  - Supports threshold monitoring

### 23.2.3 Functional Description

The major components of SAR ADC and their interconnections are shown in Figure [23.2-1](#).

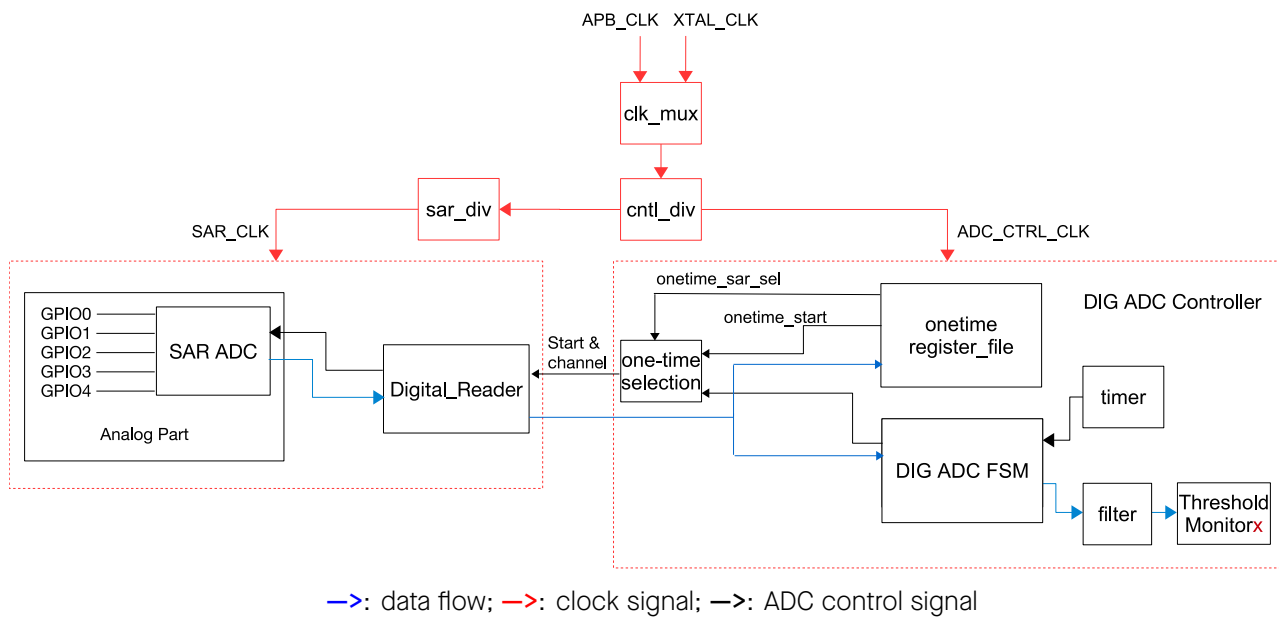


Figure 23.2-1. SAR ADC Function Overview

As shown in Figure 23.2-1, the SAR ADC module consists of the following components:

- SAR ADC: measures voltages from up to five channels.
- Clock management: selects clock sources and their dividers:
  - Clock sources: can be APB\_CLK or XTAL\_CLK.
  - Divided Clocks:
    - \* SAR\_CLK: operating clock for SAR ADC and Digital\_Reader. Note that the divider (sar\_div) of SAR\_CLK must be no less than 2.
    - \* ADC\_CTRL\_CLK: operating clock for DIG ADC FSM.
- Digital\_Reader (driven by DIG ADC FSM): reads data from SAR ADC.
- DIG ADC FSM: generates the signals required throughout the ADC sampling process.
- Threshold monitor<sub>x</sub>: threshold monitor 1 and threshold monitor 2. The monitor<sub>x</sub> will trigger an interrupt when the sampled value is greater than the pre-set high threshold or less than the pre-set low threshold.

The following sections describe the individual components in details.

### 23.2.3.1 Input Signals

In order to sample an analog signal, the SAR ADC must first select the analog pin to measure via an internal multiplexer. A summary of all the analog signals that may be sent to the SAR ADC module for processing are presented in Table 23.2-1.

Table 23.2-1. SAR ADC Input Signals

Signal	Channel
GPIO0	0
GPIO1	1
GPIO2	2
GPIO3	3
GPIO4	4

### 23.2.3.2 ADC Conversion and Attenuation

When the SAR ADC converts an analog voltage, the resolution (12-bit) of the conversion spans voltage range from 0 mV to  $V_{ref}$ .  $V_{ref}$  is the SAR ADC's internal reference voltage (1100 mV by design). The output value of the conversion (data) is mapped to analog voltage  $V_{data}$  using the following formula:

$$V_{data} = \frac{V_{ref}}{4095} \times data$$

In order to convert voltages larger than  $V_{ref}$ , input signals can be attenuated before being input into the SAR ADCs. The attenuation can be configured to 0 dB, 2.5 dB, 6 dB, and 10 dB.

### 23.2.3.3 DIG ADC Controller

The clock of the DIG ADC controller is quite fast, thus the sample rate is high. For more information, see Section ADC Characteristics in [ESP8684 Series Datasheet](#).

This controller supports:

- up to 12-bit sampling resolution
- one-time sampling triggered by software
- multi-channel scanning triggered by the timer

The configuration of a one-time sampling triggered by the software is as follows:

- Set [APB\\_SARADC1\\_ONETIME\\_SAMPLE](#) to enable the one-time sampling function of the SAR ADC.
- Configure [APB\\_SARADC\\_ONETIME\\_CHANNEL](#) to select one channel to sample.
- Configure [APB\\_SARADC\\_ONETIME\\_ATTEN](#) to set attenuation.
- Configure [APB\\_SARADC\\_ONETIME\\_START](#) to start this one-time sampling.
- On completion of sampling, [APB\\_SARADC\\_ADC1\\_DONE\\_INT\\_RAW](#) interrupt is generated. Software can use this interrupt to initiate reading of the sampled values from [APB\\_SARADC\\_ADC1\\_DATA](#).

If the timer-triggered multi-channel scanning is selected, follow the configuration below. Note that in this mode, the scan sequence is performed according to the configuration entered into pattern table.

- Configure [APB\\_SARADC\\_TIMER\\_TARGET](#) to set the trigger target for DIG ADC timer. When the timer counting reaches two times of the pre-configured cycle number, a sampling operation is triggered. For the working clock of the timer, see Section 23.2.3.4.
- Configure [APB\\_SARADC\\_TIMER\\_EN](#) to enable the timer.

- When the timer times out, DIG ADC FSM starts sampling according to the pattern table;
- An interrupt is triggered once the scanning is completed. The software needs to read the sampled data from corresponding registers, otherwise, the sampled data will be directly discarded after passing through the threshold monitor.

### 23.2.3.4 DIG ADC Clock

Two clocks can be used as the working clock of DIG ADC controller, depending on the configuration of `APB_SARADC_CLK_SEL`:

- 1: Select the clock (`ADC_CTRL_CLK`) divided from `XTAL_CLK`.
- 0: Select `APB_CLK`.

If `ADC_CTRL_CLK` is selected, users can configure the divider by `APB_SARADC_CLKM_DIV_NUM`.

Note that due to speed limits of SAR ADC, the operating clock of Digital\_Reader and SAR ADC is `SAR_CLK`, the frequency of which affects the sampling precision. The lower the frequency, the higher the precision. `SAR_CLK` is divided from `ADC_CTRL_CLK`. The divider coefficient is configured by `APB_SARADC_SAR_CLK_DIV`.

The ADC needs 25 `SAR_CLK` clock cycles per sample, so the maximum sampling rate is limited by the `SAR_CLK` frequency. For more information about clocks, see Chapter 6 *Reset and Clock*.

### 23.2.3.5 DIG ADC FSM

#### Overview

Figure 23.2-2 shows the diagram of DIG ADC FSM.

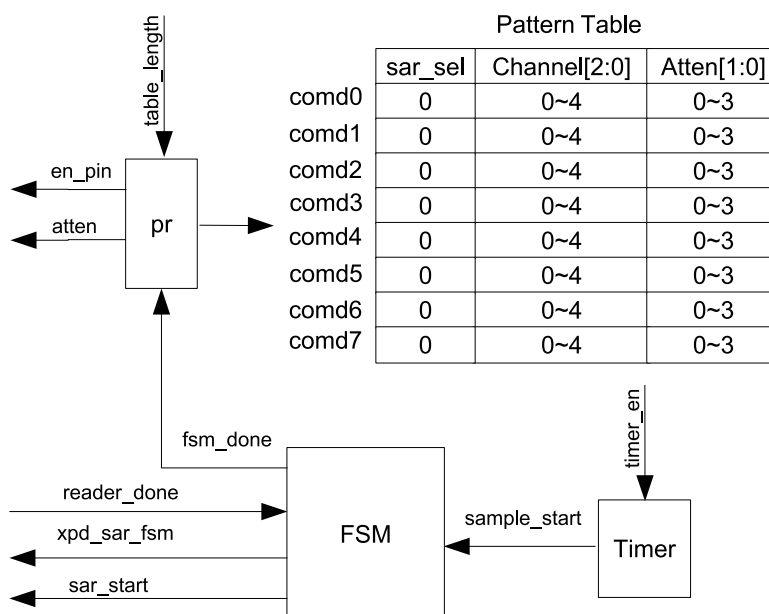


Figure 23.2-2. Diagram of DIG ADC FSM

Wherein:

- Timer: a dedicated timer for DIG ADC controller, to generate a `sample_start` signal.



- pr: the pointer to pattern table entries. FSM sends out corresponding signals based on the configuration of the pattern table entry that the pointer points to.

The execution process is as follows:

- Configure [APB\\_SARADC\\_TIMER\\_EN](#) to enable the DIG ADC timer. The timeout event of this timer triggers a sample\_start signal. This signal drives the FSM module to start sampling.
- When the FSM module receives the sample\_start signal, it starts the following operations:
  - Power up SAR ADC.
  - Select SAR ADC as the working ADC, configure the ADC channel and attenuation, based on the pattern table entry that the current pr points to.
  - According to the configuration information, output the corresponding en\_pad and atten signals to the analog side.
  - Initiate the sar\_start signal and start sampling.
- When the FSM receives the reader\_done signal from ADC Reader (Digital\_Reader), it will
  - stop sampling,
  - the data is discarded after passing through the filter and the threshold monitor, see Figure 23.2-1),
  - update the pattern table pointer (pr) and wait for the next sampling. Note that if the pointer (pr) is smaller than [APB\\_SARADC\\_SAR\\_PATT\\_LEN](#) (table\_length), then pr = pr + 1, otherwise, pr is cleared.

### Pattern Table

There is one pattern table in the controller, consisting of the [APB\\_SARADC\\_SAR\\_PATT\\_TAB1\\_REG](#) and [APB\\_SARADC\\_SAR\\_PATT\\_TAB2\\_REG](#) registers, see Figure 23.2-3 and Figure 23.2-4:

(reserved)								cmd0						cmd1						cmd2						cmd3					
31		24	23		18	17		12	11		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

cmd x represents pattern table entries. x here is the index, 0 ~ 3.

Figure 23.2-3. APB\_SARADC\_SAR\_PATT\_TAB1\_REG and Pattern Table Entry 0 - Entry 3

(reserved)								cmd4						cmd5						cmd6						cmd7					
31		24	23		18	17		12	11		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

cmd x represents pattern table entries. x here is the index, 4 ~ 7.

Figure 23.2-4. APB\_SARADC\_SAR\_PATT\_TAB2\_REG and Pattern Table Entry 4 - Entry 7

Each register consists of four 6-bit pattern table entries. Each entry is composed of three fields that contain working ADC, ADC channel and attenuation information, as shown in Table 23.2-5.

sar_sel		ch_sel		atten	
5	4	2	1	0	
x	xx		x	x	

Figure 23.2-5. Pattern Table Entry

**atten** Attenuation. 0: 0 dB; 1: 2.5 dB; 2: 6 dB; 3: 10 dB.

**ch\_sel** ADC channel, see Table 23.2-1.

**sar\_sel** Working ADC. 0: SAR ARC. ESP8684 provides only one SAR ADC, therefore, this value is always 0.

### Configuration of multi-channel scanning

In this example, two channels are selected for multi-channel scanning:

- Channel 2, with the attenuation of 10 dB
- Channel 0, with the attenuation of 2.5 dB

The detailed configuration is as follows:

- Configure the first pattern table entry (cmd0):

sar_sel		ch_sel		atten	
5	4	2	1	0	
0	2		3		

Figure 23.2-6. cmd0 Configuration

**atten** write the value of 3 to this field, to set the attenuation to 10 dB.

**ch\_sel** write the value of 2 to this field, to select channel 2 (see Table 23.2-1).

**sar\_sel** Configure this bit to 0.

- Configure the second pattern table entry (cmd1):

sar_sel		ch_sel		atten	
5	4	2	1	0	
0	0		1		

Figure 23.2-7. cmd1 Configuration

**atten** write the value of 1 to this field, to set the attenuation to 2.5 dB.

**ch\_sel** write the value of 0 to this field, to select channel 0 (see Table 23.2-1).

**sar\_sel** Configure this bit to 0.

- Configure `APB_SARADC_SAR_PATT_LEN` to 1, i.e., set pattern table length to (this value + 1 = 2). Then pattern table entries cmd0 and cmd1 will be used.
- Enable the timer, then DIG ADC controller starts scanning the two channels in cycles, as configured in the pattern table entries.

### 23.2.3.6 ADC Filters

The DIG ADC controller provides two filters for automatic filtering of sampled ADC data. Both filters can be configured to any channel of the SAR ADC and then filter the sampled data for the target channel. The filter's formula is shown below:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$ : the filtered data value.
- $data_{in}$ : the sampled data value from the ADC.
- $data_{prev}$ : the last filtered data value.
- $k$ : the filter coefficient.

The filters are configured as follows:

- Configure `APB_SARADC_FILTER_CHANNELx` to select the ADC channel for filter  $x$ ;
- Configure `APB_SARADC_FILTER_FACTORx` to set the coefficient for filter  $x$ ;

Note that  $x$  is used here as the placeholder of filter index. 0: filter 0; 1: filter 1.

### 23.2.3.7 Threshold Monitoring

DIG ADC controller contains two threshold monitors that can be configured to monitor on any channel of the SAR ADC. A high threshold interrupt is triggered when the ADC sample value is larger than the pre-configured high threshold, and a low threshold interrupt is triggered if the sample value is lower than the pre-configured low threshold.

The configuration of threshold monitoring is as follows:

- Set `APB_SARADC_THRESx_EN` to enable threshold monitor  $x$ .
- Configure `APB_SARADC_THRESx_LOW` to set a low threshold;
- Configure `APB_SARADC_THRESx_HIGH` to set a high threshold;
- Configure `APB_SARADC_THRESx_CHANNEL` to select the channel to monitor.

Note that  $x$  is used here as the placeholder of monitor index. 0: monitor 0; 1: monitor 1.

## 23.3 Temperature Sensor

### 23.3.1 Overview

ESP8684 provides a temperature sensor to monitor temperature changes inside the chip in real time.

### 23.3.2 Features

The temperature sensor has the following features:

- Supports software triggering and, once triggered, the data can be read continuously
- Configurable temperature offset based on the environment, to improve the accuracy

- Adjustable measurement range

### 23.3.3 Functional Description

The temperature sensor can be started by software as follows:

- Set `APB_SARADC_TSENS_PU` to power up the temperature sensor;
- Wait for `APB_SARADC_TSENS_XPD_WAIT` clock cycles till the reset of temperature sensor is released, the sensor starts measuring the temperature;
- If this is the first time to start the temperature sensor, wait the sensor to get started up (about 100  $\mu$ s). Then, the temperature data can be read continuously from `APB_SARADC_TSENS_OUT`.

The actual temperature ( $^{\circ}\text{C}$ ) can be obtained by converting the output of temperature sensor via the following formula:

$$T(^{\circ}\text{C}) = 0.4386 * VALUE - 27.88 * offset - 20.52$$

VALUE in the formula is the output of the temperature sensor, and the offset is determined by the temperature offset. The temperature offset varies in different actual environment (the temperature range). For details, refer to Table 23.3-1.

Table 23.3-1. Temperature Offset

Measurement Range ( $^{\circ}\text{C}$ )	Temperature Offset ( $^{\circ}\text{C}$ )
50 ~ 125	-2
20 ~ 100	-1
-10 ~ 80	0
-30 ~ 50	1
-40 ~ 20	2

## 23.4 Interrupts

- `APB_SARADC_ADC1_DONE_INT`: triggered when SAR ADC completes one data conversion.
- `APB_SARADC_THRESx_HIGH_INT`: triggered when the sampling value is higher than the high threshold of monitor `x`.
- `APB_SARADC_THRESx_LOW_INT`: triggered when the sampling value is lower than the low threshold of monitor `x`.

## 23.5 Register Summary

The addresses in this section are relative to the ADC controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

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

Name	Description	Address	Access
<b>Configuration registers</b>			
<a href="#">APB_SARADC_CTRL_REG</a>	Configuration register for SAR ADC FSM	0x0000	R/W
<a href="#">APB_SARADC_CTRL2_REG</a>	Configuration register for SAR ADC FSM sampling	0x0004	R/W
<a href="#">APB_SARADC_FILTER_CTRL1_REG</a>	Configuration register 1 for filter	0x0008	R/W
<a href="#">APB_SARADC_SAR_PATT_TAB1_REG</a>	Pattern table register 1	0x0018	R/W
<a href="#">APB_SARADC_SAR_PATT_TAB2_REG</a>	Pattern table register 2	0x001C	R/W
<a href="#">APB_SARADC_ONETIME_SAMPLE_REG</a>	Configuration register for one-time sampling	0x0020	R/W
<a href="#">APB_SARADC_FILTER_CTRL0_REG</a>	Configuration register 0 for filter	0x0028	R/W
<a href="#">APB_SARADC_1_DATA_STATUS_REG</a>	SAR ADC sampling data register	0x002C	RO
<a href="#">APB_SARADC_THRES0_CTRL_REG</a>	Sampling threshold control register 0	0x0034	R/W
<a href="#">APB_SARADC_THRES1_CTRL_REG</a>	Sampling threshold control register 1	0x0038	R/W
<a href="#">APB_SARADC_THRES_CTRL_REG</a>	Sampling threshold enable register	0x003C	R/W
<a href="#">APB_SARADC_INT_ENA_REG</a>	Enable register of SAR ADC interrupts	0x0040	R/W
<a href="#">APB_SARADC_INT_RAW_REG</a>	Raw register of SAR ADC interrupts	0x0044	RO
<a href="#">APB_SARADC_INT_ST_REG</a>	State register of SAR ADC interrupts	0x0048	RO
<a href="#">APB_SARADC_INT_CLR_REG</a>	Clear register of SAR ADC interrupts	0x004C	WO
<a href="#">APB_SARADC_DMA_CONF_REG</a>	DMA configuration register for SAR ADC	0x0050	R/W
<a href="#">APB_SARADC_APB_ADC_CLKM_CONF_REG</a>	SAR ADC clock control register	0x0054	R/W
<a href="#">APB_SARADC_APB_TSENS_CTRL_REG</a>	Temperature sensor control register 1	0x0058	varies
<a href="#">APB_SARADC_APB_TSENS_CTRL2_REG</a>	Temperature sensor control register 2	0x005C	R/W
<b>Version register</b>			
<a href="#">APB_SARADC_APB_CTRL_DATE_REG</a>	Version control register	0x03FC	R/W

## 23.6 Register

The addresses in this section are relative to the ADC controller base address provided in Table 3.3-3 in Chapter 3 *System and Memory*.

**Register 23.1. APB\_SARADC\_CTRL\_REG (0x0000)**

(reserved)		(reserved)		APB_SARADC_XPD_SAR_FORCE		(reserved)		APB_SARADC_SAR_PATT_P_CLEAR		(reserved)		APB_SARADC_SAR_PATT_LEN		APB_SARADC_SAR_CLK_DIV		APB_SARADC_SAR_CLK_GATED		(reserved)		APB_SARADC_START APB_SARADC_START_FORCE	
31	30	29	28	27	26	24	23	22	18	17	15	14	7		6	5	2		1	0	
1		0		0		0 0 0		0 0 0 0 0		7		4		1		0 0 0 0		0		0 Reset	

**APB\_SARADC\_START\_FORCE** 0: select FSM to start SAR ADC. 1: select software to start SAR ADC.  
(R/W)

**APB\_SARADC\_START** Write 1 here to start the SAR ADC by software. Valid only when [APB\\_SARADC\\_START\\_FORCE](#) = 1. (R/W)

**APB\_SARADC\_SAR\_CLK\_GATED** 0: SAR ADC clock is always on. 1: SAR ADC clock is turned off when SAR ADC is in idle. (R/W)

**APB\_SARADC\_SAR\_CLK\_DIV** SAR ADC clock divider. This value should be no less than 2. (R/W)

**APB\_SARADC\_SAR\_PATT\_LEN** Configure how many pattern table entries will be used. If this field is set to 1, then pattern table entries (cmd0) and (cmd1) will be used. (R/W)

**APB\_SARADC\_SAR\_PATT\_P\_CLEAR** Clear the pointer of pattern table entry for DIG ADC controller.  
(R/W)

**APB\_SARADC\_XPD\_SAR\_FORCE** Force select XPD SAR. (R/W)

Register 23.2. APB\_SARADC\_CTRL2\_REG (0x0004)

(reserved)								APB_SARADC_TIMER_EN								APB_SARADC_TIMER_TARGET								(reserved)				APB_SARADC_SAR1_INV				APB_SARADC_MAX_MEAS_NUM								APB_SARADC_MEAS_NUM_LIMIT																																																																																							
31								25								24								23								12								11								10								9								8								1								0																																															
0								0								0								0								0								0								0								0								0								0								10								0								0								255								0								Reset							

**APB\_SARADC\_MEAS\_NUM\_LIMIT** Enable the limitation of SAR ADC maximum conversion times.

Valid only when the timer is used to control SAR ADC. (R/W)

**APB\_SARADC\_MAX\_MEAS\_NUM** The SAR ADC maximum conversion times. (R/W)

**APB\_SARADC\_SAR1\_INV** Write 1 here to invert the data of SAR ADC. (R/W)

**APB\_SARADC\_TIMER\_TARGET** Set SAR ADC timer target. (R/W)

**APB\_SARADC\_TIMER\_EN** Enable SAR ADC timer trigger. (R/W)

Register 23.3. APB\_SARADC\_FILTER\_CTRL1\_REG (0x0008)

APB_SARADC_FILTER_FACTOR0										APB_SARADC_FILTER_FACTOR1										(reserved)																				
31	29	28	26	25																											0									
0		0		0 0																												0	Reset							

**APB\_SARADC\_FILTER\_FACTOR1** The filter coefficient for SAR ADC filter 1. (R/W)

**APB\_SARADC\_FILTER\_FACTOR0** The filter coefficient for SAR ADC filter 0. (R/W)

**Register 23.4. APB\_SARADC\_SAR\_PATT\_TAB1\_REG (0x0018)**

(reserved)								APB_SARADC_SAR_PATT_TAB1																							
31								24	0																						
0	0	0	0	0	0	0	0	0xffffffff																							Reset

**APB\_SARADC\_SAR\_PATT\_TAB1** Pattern table entries 0 ~ 3 (each entry is six bits). (R/W)

**Register 23.5. APB\_SARADC\_SAR\_PATT\_TAB2\_REG (0x001C)**

(reserved)								APB_SARADC_SAR_PATT_TAB2																						
31								24	23																			0		
0	0	0	0	0	0	0	0	0xffffffff																						Reset

**APB\_SARADC\_SAR\_PATT\_TAB2** Pattern table entries 4 ~ 7 (each entry is six bits). (R/W)

**Register 23.6. APB\_SARADC\_ONETIME\_SAMPLE\_REG (0x0020)**

APB_SARADC1_ONETIME_SAMPLE (reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
APB_SARADC_ONETIME_START																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
APB_SARADC_ONETIME_CHANNEL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
APB_SARADC_ONETIME_ATTEN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31	30	29	28				25	24	23	22																				0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**APB\_SARADC\_ONETIME\_ATTEN** Configure the attenuation for a one-time sampling. (R/W)

**APB\_SARADC\_ONETIME\_CHANNEL** Configure the channel for a one-time sampling. (R/W)

**APB\_SARADC\_ONETIME\_START** Start SAR ADC one-time sampling. (R/W)

**APB\_SARADC1\_ONETIME\_SAMPLE** Enable SAR ADC one-time sampling. (R/W)



### Register 23.7. APB\_SARADC\_FILTER\_CTRL0\_REG (0x0028)

[illegible]

**APB\_SARADC\_FILTER\_CHANNEL1** The filter channel for SAR ADC filter 1. (R/W)

**APB\_SARADC\_FILTER\_CHANNELO** The filter channel for SAR ADC filter 0. (R/W)

**APB\_SARADC\_FILTER\_RESET** Reset SAR ADC filter. (R/W)

### Register 23.8. APB\_SARADC\_1\_DATA\_STATUS\_REG (0x002C)

Diagram of the SARADC1\_DATA register. The register is 32 bits wide, divided into a 16-bit upper half and a 16-bit lower half. The upper half is labeled '(reserved)' and the lower half is labeled 'APB\_SARADC1\_DATA'. The register is reset to 0.

31	17	16	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
0			

Reset

**APB\_SARADC\_ADC1\_DATA** SAR ADC conversion data. (RO)

### Register 23.9. APB\_SARADC\_THRES0\_CTRL\_REG (0x0034)

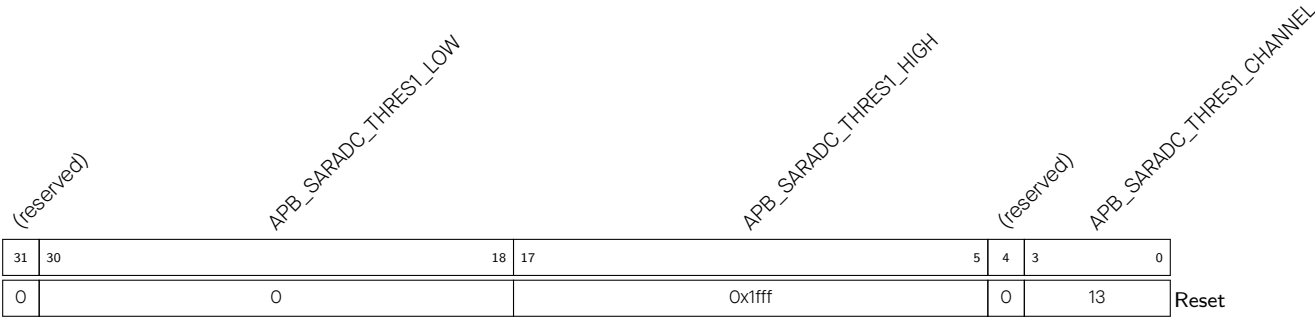
		(reserved)																APB_SARADC_THRES0_LOW																																APB_SARADC_THRES0_HIGH																																(reserved)																APB_SARADC_THRES0_CHANNEL															
31		30																18																17																5																4		3		0																																											
0		0																																0x1fff																0		13		Reset																																																											

**APB\_SARADC\_THRES0\_CHANNEL** The channel for SAR ADC monitor 0. (R/W)

**APB\_SARADC\_THRES0\_HIGH** The high threshold for SAR ADC monitor 0. (R/W)

**APB\_SARADC\_THRES0\_LOW** The low threshold for SAR ADC monitor 0. (R/W)

Register 23.10. APB\_SARADC\_THRES1\_CTRL\_REG (0x0038)

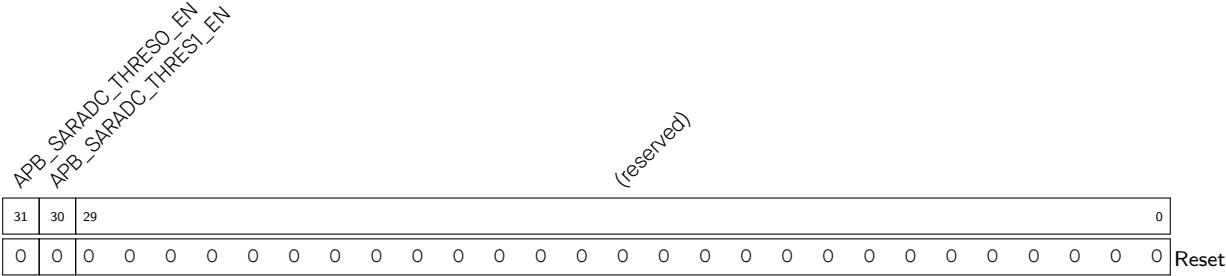


**APB\_SARADC\_THRES1\_CHANNEL** The channel for SAR ADC monitor 1. (R/W)

**APB\_SARADC\_THRES1\_HIGH** The high threshold for SAR ADC monitor 1. (R/W)

**APB\_SARADC\_THRES1\_LOW** The low threshold for SAR ADC monitor 1. (R/W)

Register 23.11. APB\_SARADC\_THRES\_CTRL\_REG (0x003C)



**APB\_SARADC\_THRES1\_EN** Enable threshold monitor 1. (R/W)

**APB\_SARADC\_THRES0\_EN** Enable threshold monitor 0. (R/W)

### Register 23.12. APB\_SARADC\_INT\_ENA\_REG (0x0040)

Diagram of the SARADC\_THRES1 register structure:

- Bit 31: APB\_SARADC\_ADC1\_DONE\_INT\_ENA
- Bit 30: (reserved)
- Bit 29: APB\_SARADC\_THRES0\_HIGH\_INT\_ENA
- Bit 28: APB\_SARADC\_THRES1\_HIGH\_INT\_ENA
- Bit 27: APB\_SARADC\_THRES0\_LOW\_INT\_ENA
- Bit 26: APB\_SARADC\_THRES1\_LOW\_INT\_ENA
- Bits 25-31: (reserved)
- Bit 0: Reset

**APB\_SARADC\_THRES1\_LOW\_INT\_ENA** Enable bit of **APB\_SARADC\_THRES1\_LOW\_INT** interrupt.  
(R/W)

**APB\_SARADC\_THRESO\_LOW\_INT\_ENA** Enable bit of [APB\\_SARADC\\_THRESO\\_LOW\\_INT](#) interrupt.  
(R/W)

**APB\_SARADC\_THRES1\_HIGH\_INT\_ENA** Enable bit of [APB\\_SARADC\\_THRES1\\_HIGH\\_INT](#) interrupt.  
(R/W)

**APB\_SARADC\_THRESO\_HIGH\_INT\_ENA** Enable bit of **APB\_SARADC\_THRESO\_HIGH\_INT** interrupt.  
(R/W)

**APB\_SARADC\_ADC1\_DONE\_INT\_ENA** Enable bit of **APB\_SARADC\_ADC1\_DONE\_INT** interrupt.  
(R/W)

Register 23.13. APB\_SARADC\_INT\_RAW\_REG (0x0044)

APB_SARADC_ADC1_DONE_INT_RAW (reserved)						(reserved)																														0
31	30	29	28	27	26	25																														0
0	0	0	0	0	0	0	0	0	0	0	0	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

**APB\_SARADC\_THRES1\_LOW\_INT\_RAW** Raw bit of [APB\\_SARADC\\_THRES1\\_LOW\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES0\_LOW\_INT\_RAW** Raw bit of [APB\\_SARADC\\_THRES0\\_LOW\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES1\_HIGH\_INT\_RAW** Raw bit of [APB\\_SARADC\\_THRES1\\_HIGH\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES0\_HIGH\_INT\_RAW** Raw bit of [APB\\_SARADC\\_THRES0\\_HIGH\\_INT](#) interrupt. (RO)

**APB\_SARADC\_ADC1\_DONE\_INT\_RAW** Raw bit of [APB\\_SARADC\\_ADC1\\_DONE\\_INT](#) interrupt. (RO)

Register 23.14. APB\_SARADC\_INT\_ST\_REG (0x0048)

APB_SARADC_ADC1_DONE_INT_ST (reserved)																															APB_SARADC_THRES0_HIGH_INT_ST (reserved)																															APB_SARADC_THRES0_LOW_INT_ST (reserved)																															APB_SARADC_THRES1_HIGH_INT_ST (reserved)																															APB_SARADC_THRES1_LOW_INT_ST (reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31	30	29	28	27	26	25																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		

**APB\_SARADC\_THRES1\_LOW\_INT\_ST** Status of [APB\\_SARADC\\_THRES1\\_LOW\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES0\_LOW\_INT\_ST** Status of [APB\\_SARADC\\_THRES0\\_LOW\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES1\_HIGH\_INT\_ST** Status of [APB\\_SARADC\\_THRES1\\_HIGH\\_INT](#) interrupt. (RO)

**APB\_SARADC\_THRES0\_HIGH\_INT\_ST** Status of [APB\\_SARADC\\_THRES0\\_HIGH\\_INT](#) interrupt. (RO)

**APB\_SARADC\_ADC1\_DONE\_INT\_ST** Status of [APB\\_SARADC\\_ADC1\\_DONE\\_INT](#) interrupt. (RO)

### Register 23.15. APB\_SARADC\_INT\_CLR\_REG (0x004C)

Diagram illustrating the structure of the APB\_SARADC\_THRES0 register. The register is 32 bits wide, with bits 31 down to 25 labeled as (reserved). Bit 24 is labeled APB\_SARADC\_THRES0\_HIGH\_INT\_CLR. Bit 23 is labeled APB\_SARADC\_THRES0\_LOW\_INT\_CLR. Bit 22 is labeled APB\_SARADC\_THRES1\_HIGH\_INT\_CLR. Bit 21 is labeled APB\_SARADC\_THRES1\_LOW\_INT\_CLR. Bit 20 is labeled APB\_SARADC\_DONE\_INT\_CLR. Bit 19 is labeled (reserved). The remaining bits (18 down to 0) are labeled (reserved). A Reset signal is shown at the bottom right.

**APB\_SARADC\_THRES1\_LOW\_INT\_CLR** Clear bit of [APB\\_SARADC\\_THRES1\\_LOW\\_INT](#) interrupt. (WO)

**APB\_SARADC\_THRESO\_LOW\_INT\_CLR** Clear bit of **APB\_SARADC\_THRESO\_LOW\_INT** interrupt.  
(WO)

**APB\_SARADC\_THRES1\_HIGH\_INT\_CLR** Clear bit of **APB\_SARADC\_THRES1\_HIGH\_INT** interrupt.  
(WO)

**APB\_SARADC\_THRESO\_HIGH\_INT\_CLR** Clear bit of [APB\\_SARADC\\_THRESO\\_HIGH\\_INT](#) interrupt.  
(WO)

**APB\_SARADC\_ADC1\_DONE\_INT\_CLR** Clear bit of [APB\\_SARADC\\_ADC1\\_DONE\\_INT](#) interrupt. (WO)

### Register 23.16. APB\_SARADC\_DMA\_CONF\_REG (0x0050)

31		30		29																16		15										0
0		0		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																255										Reset		

APB\_SARADC\_APB\_ADC\_TRANS

APB\_SARADC\_APB\_ADC\_RESET\_FSM

(reserved)

(reserved)

**APB\_SARADC\_APB\_ADC\_RESET\_FSM** Reset DIG ADC controller status. (R/W)

**APB\_SARADC\_APB\_ADC\_TRANS** ESP8684 does not support DMA. This bit must be set to 0. Data is discarded after passing through the filter and the threshold monitor. (R/W)

Register 23.17. APB\_SARADC\_APB\_ADC\_CLKM\_CONF\_REG (0x0054)

(reserved)										APB_SARADC_CLK_SEL										(reserved)										APB_SARADC_CLKM_DIV_A										APB_SARADC_CLKM_DIV_B										APB_SARADC_CLKM_DIV_NUM										
31										23										22	21	20	19	14										13	8										7	0														
0 0 0 0 0 0 0 0 0 0										0										0	0x0										0x0										4										Reset									

**APB\_SARADC\_CLKM\_DIV\_NUM** The integer part of ADC clock divider. Divider value = APB\_SARADC\_CLKM\_DIV\_NUM + APB\_SARADC\_CLKM\_DIV\_B/APB\_SARADC\_CLKM\_DIV\_A. (R/W)

**APB\_SARADC\_CLKM\_DIV\_B** The numerator value of fractional clock divider. (R/W)

**APB\_SARADC\_CLKM\_DIV\_A** The denominator value of fractional clock divider. (R/W)

**APB\_SARADC\_CLK\_SEL** 0: Use APB\_CLK as clock source, 1: use divided-down XTAL\_CLK as clock source. (R/W)

Register 23.18. APB\_SARADC\_APB\_TSSENS\_CTRL\_REG (0x0058)

(reserved)										APB_SARADC_TSSENS_PU										APB_SARADC_TSSENS_CLK_DIV										APB_SARADC_TSSENS_IN_INV										(reserved)										APB_SARADC_TSSENS_OUT																																																	
31										23										22										21										14										13										12										8										7										0									
0 0 0 0 0 0 0 0 0 0										0										6										0										0 0 0 0 0 0 0										0x0										Reset																																							

**APB\_SARADC\_TSSENS\_OUT** Temperature sensor data out. (RO)

**APB\_SARADC\_TSSENS\_IN\_INV** Invert temperature sensor input value. (R/W)

**APB\_SARADC\_TSSENS\_CLK\_DIV** Temperature sensor clock divider. (R/W)

**APB\_SARADC\_TSSENS\_PU** Temperature sensor power up. (R/W)

### Register 23.19. APB\_SARADC\_APB\_TSENS\_CTRL2\_REG (0x005C)

(reserved)																APB_SARADC_TSENS_CLK_SEL		(reserved)																APB_SARADC_TSENS_XPD_WAIT															
31																16		15	14	13	12	11	0																										
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	1	0	0	0 x 2																													

Reset

**APB\_SARADC\_TSENS\_XPD\_WAIT** The wait time before temperature sensor is powered up. (R/W)

**APB\_SARADC\_TSENS\_CLK\_SEL** Choose working clock for temperature sensor. 0: RC\_FAST\_CLK.  
1: XTAL\_CLK. (R/W)

### Register 23.20. APB\_SARADC\_APB\_CTRL\_DATE\_REG (0x03FC)

Diagram of the APB\_SARADC\_DATE register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. The value 0x02107210 is shown in the center. A label 'Reset' is at the bottom right.

**APB\_SARADC\_DATE** Version register. (R/W)

# Part VII

## Appendix

This part contains the following information starting from the next page:

- [Related Documentation and Resources](#)
- [Glossary](#)
- [Programming Reserved Register Field](#)
- [Interrupt Configuration Registers](#)
- [Revision History](#)



## Related Documentation and Resources

### Related Documentation

- [ESP8684 Series Datasheet](#) – Specifications of the ESP8684 hardware.
- [ESP8684 Hardware Design Guidelines](#) – Guidelines on how to integrate the ESP8684 into your hardware product.
- [ESP8684 Series SoC Errata](#) – Descriptions of known errors in ESP8684 series of SoCs.
- *Certificates*  
<https://espressif.com/en/support/documents/certificates>
- *ESP8684 Product/Process Change Notifications (PCN)*  
<https://espressif.com/en/support/documents/pcns?keys=ESP8684>
- *Documentation Updates and Update Notification Subscription*  
<https://espressif.com/en/support/download/documents>

### Developer Zone

- [ESP-IDF Programming Guide for ESP8684](#) – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.  
<https://github.com/espressif>
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.  
<https://esp32.com/>
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.  
<https://blog.espressif.com/>
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.  
<https://espressif.com/en/support/download/sdks-demos>

### Products

- *ESP8684 Series SoCs* – Browse through all ESP8684 SoCs.  
<https://espressif.com/en/products/socs?id=ESP8684>
- *ESP8684 Series Modules* – Browse through all ESP8684-based modules.  
<https://espressif.com/en/products/modules?id=ESP8684>
- *ESP8684 Series DevKits* – Browse through all ESP8684-based devkits.  
<https://espressif.com/en/products/devkits?id=ESP8684>
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.  
<https://products.espressif.com/#/product-selector?language=en>

### Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.  
<https://espressif.com/en/contact-us/sales-questions>

## Glossary

### Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED Control PWM (Pulse Width Modulation)
MCPWM	Motor Control PWM (Pulse Width Modulation)
PCNT	Pulse Count Controller
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

### Abbreviations Related to Registers

REG	<b>Register.</b>
SYSREG	<b>System registers</b> are a group of registers that control system reset, memory, clocks, software interrupts, power management, clock gating, etc.
ISO	<b>Isolation.</b> 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	<b>Non-maskable interrupt</b> 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     | • R/W/SS     | • R/WS/SS/SC  |
| • WO     | • R/W/SS/SC  | • R/SS/WTC    |
| • WT     | • R/WC/SS    | • R/SC/WTC    |
| • R/W    | • R/WC/SC    | • R/SS/SC/WTC |
| • R/W1   | • R/WC/SS/SC | • RF/WF       |
| • WL     | • R/WS/SC    | • R/SS/RC     |
| • R/W/SC | • R/WS/SS    | • 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 23.21. Register X (Address)

(reserved)												Field_C				(reserved)												Field_B		Field_A			
31											20	19					16	15											2	1	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	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 23.6-4.

- **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 23.6-4. 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
2025-06-25	v1.3	<p>Updated the following chapters:</p> <ul style="list-style-type: none"> <li>Chapter <a href="#">2 GDMA Controller (GDMA)</a>: Added descriptions for the GDMA_OUTFIFO_OVF_CHO_INT, GDMA_OUTFIFO_UDF_CHO_INT, GDMA_INFIFO_OVF_CHO_INT, and GDMA_INFIFO_UDF_CHO_INT interrupts</li> <li>Chapter <a href="#">15 ECC Hardware Accelerator (ECC)</a>: Updated register field prefix from ECC to ECC_MULT</li> <li>Chapter <a href="#">22 LED PWM Controller (LEDC)</a>: Updated the lowest resolution in Table <a href="#">22.3-1</a></li> <li>Chapter <a href="#">19 UART Controller (UART)</a>: Updated descriptions about clearing the wake_up signal; add clarification about writing <a href="#">UART_RXFIFO_RD_BYTE</a></li> <li>Chapter <a href="#">21 I2C Master Controller (I2C)</a>, <a href="#">10 System Timer (SYSTIMER)</a>: Updated the crystal frequencies supported by XTAL_CLK</li> <li>Chapter <a href="#">9 Low-power Management (RTC_CNTL)</a>: Updated the description of predefined low-power modes</li> <li>Chapter <a href="#">23 On-Chip Sensor and Analog Signal Processing</a>: Corrected “-0.5” to “+0.5” in the ADC filter formula</li> </ul>
2024-02-19	v1.2	<p>Updated the font to Maison Neue.</p> <p>Updated the following chapters:</p> <ul style="list-style-type: none"> <li>Chapter <a href="#">2 GDMA Controller (GDMA)</a>: Updated the descriptions of suc_eof and the EOF flag</li> <li>Chapter <a href="#">19 UART Controller (UART)</a>: Updated the number of rising edges required to generate the wake_up signal</li> <li>Chapter <a href="#">21 I2C Master Controller (I2C)</a>: Updated I2C timeout configuration and the corresponding description of the <a href="#">I2C_TIME_OUT_VALUE</a> field</li> <li>Chapter <a href="#">9 Low-power Management (RTC_CNTL)</a>: Updated the description of register <a href="#">RTC_CNTL_WDT_WKEY</a></li> </ul>

Cont'd on next page



## Cont'd from previous page

Date	Version	Release notes
2023-10-27	v1.1	<p>Added Section <a href="#">Programming Reserved Register Field</a> and Section <a href="#">Interrupt Configuration Registers</a></p> <p>Updated the following chapters:</p> <ul style="list-style-type: none"> <li>Chapter <a href="#">8 Interrupt Matrix (INTMTRX)</a>: Updated register prefix APB_CNTL to SYSCON</li> <li>Chapter <a href="#">11 Timer Group (TIMG)</a>: Updated the description of <a href="#">TIMG_WDT_CLK_PRESCALE</a></li> <li>Chapter <a href="#">20 SPI Controller (SPI)</a>: Updated clock information</li> <li>Chapter <a href="#">5 IO MUX and GPIO Matrix (GPIO, IO MUX)</a>: Updated the description in Section 5.9</li> <li>Chapter <a href="#">8 Interrupt Matrix (INTMTRX)</a>: Deleted the INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_NMI_MAP_REG register and related information</li> <li>Chapter <a href="#">5 IO MUX and GPIO Matrix (GPIO, IO MUX)</a>: Deleted the GPIO_PCPU_NMI_INT_REG register and related information</li> <li>Chapter <a href="#">7 Chip Boot Control</a>: Added SPI Download Boot mode and renamed Download Boot mode to Joint Download mode in Section <a href="#">7.3.2</a></li> <li>Chapter <a href="#">7 Chip Boot Control</a>: Provided more details about how eFuse controls chip boot mode</li> <li>Chapter <a href="#">21 I2C Master Controller (I2C)</a>: Updated the descriptions of <a href="#">I2C_COMDO_REG</a>, <a href="#">I2C_SDA_FORCE_OUT</a> and <a href="#">I2C_SCL_FORCE_OUT</a></li> <li>Chapter <a href="#">13 System Registers (SYSTEM)</a>: Updated the description of <a href="#">SYSTEM_SYSCLOCK_CONF_REG</a></li> </ul>
2023-05-20	v1.0	<p>Updated the following chapters:</p> <ul style="list-style-type: none"> <li>Chapter <a href="#">2 GDMA Controller (GDMA)</a>: Updated the descriptions of the <a href="#">GDMA_IN_SUC_EOF_CHO_INT</a> interrupt and the <a href="#">GDMA_INLINK_DSCR_ADDR_CHO</a> field</li> <li>Updated the procedures to read the timer's value in Chapter <a href="#">11 Timer Group (TIMG)</a></li> <li>Removed ULP-RISC-V references from Chapter <a href="#">12 Watchdog Timers (WDT)</a></li> <li>Added descriptions about the break condition, and updated the maximum length of stop bits and related descriptions in Chapter <a href="#">19 UART Controller (UART)</a></li> <li>Added the formula to calculate duty cycle resolution and Table <a href="#">Commonly-used Frequencies and Resolutions</a> in Chapter <a href="#">22 LED PWM Controller (LEDC)</a></li> <li>Chapter <a href="#">23 On-Chip Sensor and Analog Signal Processing</a>: Updated the pattern table indexes in Figure <a href="#">23.2-3</a> and Figure <a href="#">23.2-4</a></li> <li>Chapter <a href="#">3 System and Memory</a>: Updated the description to cache</li> </ul>

Cont'd on next page

## Cont'd from previous page

Date	Version	Release notes
2022-10-27	v0.3	<p>Added the following chapters:</p> <ul style="list-style-type: none"><li>• <a href="#">2 GDMA Controller (GDMA)</a></li><li>• <a href="#">9 Low-power Management (RTC_CNTL)</a></li><li>• <a href="#">20 SPI Controller (SPI)</a></li><li>• <a href="#">23 On-Chip Sensor and Analog Signal Processing</a></li></ul> <p>Updated the following chapters:</p> <ul style="list-style-type: none"><li>• <a href="#">18 Random Number Generator (RNG)</a></li><li>• <a href="#">14 Debug Assistant (ASSIST_DEBUG)</a></li></ul>
2022-07-14	v0.2	<p>Added the following chapters:</p> <ul style="list-style-type: none"><li>• <a href="#">4 eFuse Controller (eFuse)</a></li><li>• <a href="#">15 ECC Hardware Accelerator (ECC)</a></li></ul> <p>Updated the following chapters:</p> <ul style="list-style-type: none"><li>• <a href="#">1 ESP-RISC-V CPU</a></li><li>• <a href="#">5 IO MUX and GPIO Matrix (GPIO, IO MUX)</a></li><li>• <a href="#">6 Reset and Clock</a></li></ul>
2022-05-18	v0.1	Preliminary release



## Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

ALL THIRD PARTY'S INFORMATION IN THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES TO ITS AUTHENTICITY AND ACCURACY.

NO WARRANTY IS PROVIDED TO THIS DOCUMENT FOR ITS MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, NOR DOES ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2025 Espressif Systems (Shanghai) Co., Ltd. All rights reserved.

[www.espressif.com](http://www.espressif.com)