ESP32-S2

Technical Reference Manual

Preliminary V0.4

Espressif Systems

About This Manual

The **ESP32-S2 Technical Reference Manual** is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32-S2 memory and peripherals.

For pin definition, electrical characteristics and package information, please see *ESP32-S2 Datasheet*.

Document Updates

Please always refer to the latest version on https://www.espressif.com/en/support/download/documents.

Revision History

For any changes to this document over time, please refer to the last page.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Disclaimer and Copyright Notice

Contents

List of Tables

List of Figures

1. System and Memory

1.1 Overview

The ESP32-S2 is a single-core system with one Harvard Architecture Xtensa® LX7 CPU. All internal memory, external memory, and peripherals are located on the CPU buses.

1.2 Features

- **Address Space**
  - 4 GB (32 bits wide) address space in total accessed from the data bus and instruction bus
  - 464 KB internal memory address space accessed from the instruction bus
  - 400 KB internal memory address space accessed from the data bus
  - 1.77 MB peripheral address space
  - 7.5 MB external memory virtual address space accessed from the instruction bus
  - 14.5 MB external memory virtual address space accessed from the data bus
  - 320 KB internal DMA address space
  - 10.5 MB external DMA address space
- **Internal Memory**
  - 128 KB Internal ROM
  - 320 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- **External Memory**
  - Supports up to 1 GB external SPI flash
  - Supports up to 1 GB external SPI RAM
- **DMA**
  - 9 DMA-supported modules / peripherals

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

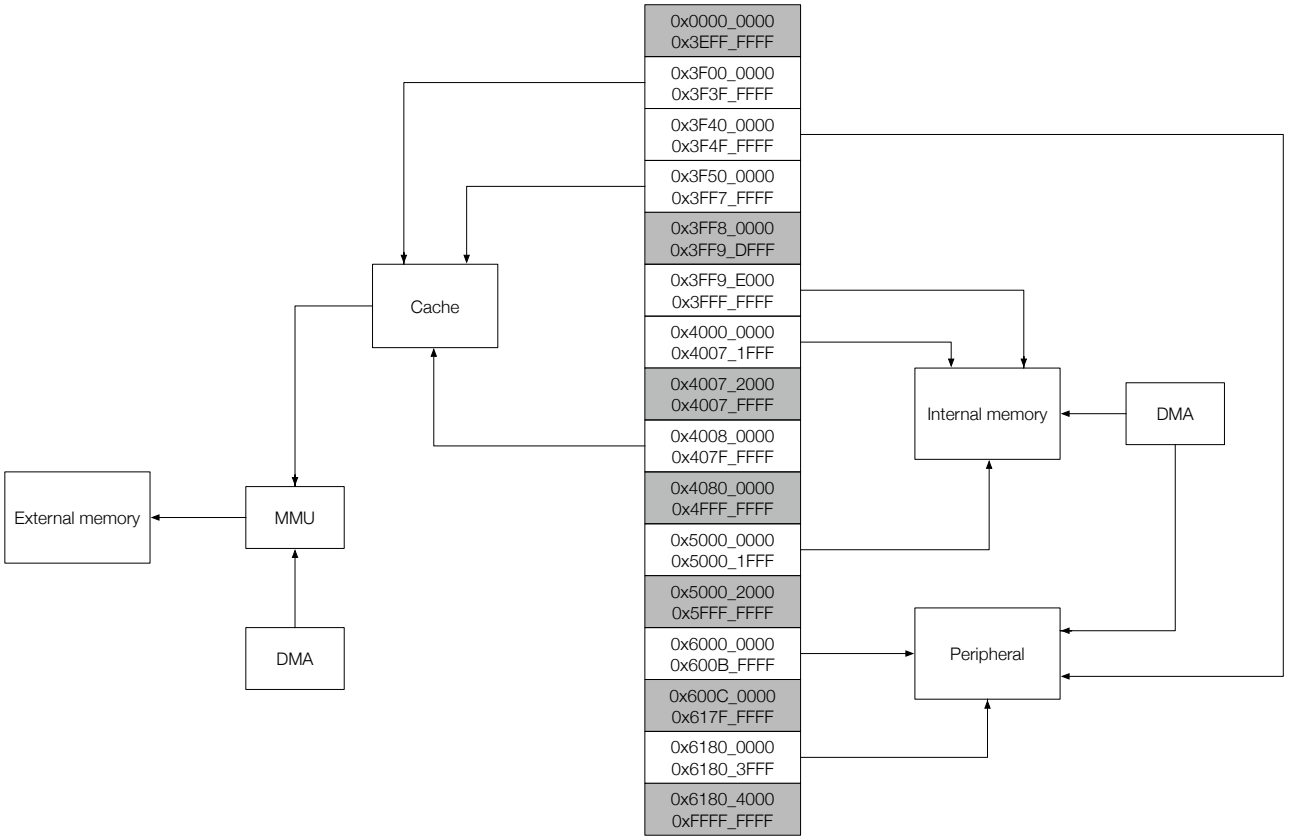


Figure 1-1. System Structure and Address Mapping

Note:

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

1.3 Functional Description

1.3.1 Address Mapping

The Harvard Architecture Xtensa® LX7 CPU can address 4 GB (32 bits wide) memory space.

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

Both data bus and instruction bus are little-endian. The CPU can access data via the data bus in a byte-, half-word-, or word-aligned matter. The CPU can also access data via the instruction bus, but only in a word-aligned manner; non-word-aligned access will cause a CPU exception.

The CPU can:

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

- access modules / peripherals via data bus.

Table 1 lists the address ranges on the data bus and instruction bus and their corresponding target memory.

Some internal and external memory can be accessed via both data bus and instruction bus. In such cases, the same memory is available to the CPU at two address ranges.

Table 1: Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3EFF_FFFF		Reserved
Data bus	0x3F00_0000	0x3F3F_FFFF	4 MB	External memory
	0x3F40_0000	0x3F4F_FFFF	1 MB	Peripherals
	0x3F50_0000	0x3FF7_FFFF	10.5 MB	External memory
	0x3FF8_0000	0x3FF9_DFFF		Reserved
Data bus	0x3FF9_E000	0x3FFF_FFFF	392 KB	Internal memory
Instruction bus	0x4000_0000	0x4007_1FFF	456 KB	Internal memory
	0x4007_2000	0x4007_FFFF		Reserved
Instruction bus	0x4008_0000	0x407F_FFFF	7.5 MB	External memory
	0x4080_0000	0x4FFF_FFFF		Reserved
Data / Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	Internal memory
	0x5000_2000	0x5FFF_FFFF		Reserved
Data / Instruction bus	0x6000_0000	0x600B_FFFF	768 KB	Peripherals
	0x600C_0000	0x617F_FFFF		Reserved
Data / Instruction bus	0x6180_0000	0x6180_3FFF	16 KB	Peripherals
	0x6180_4000	0xFFFF_FFFF		Reserved

1.3.2 Internal Memory

The internal memory consists of four segments: Internal ROM (128 KB), Internal SRAM (320 KB), RTC FAST Memory (8 KB), and RTC SLOW Memory (8 KB).

The Internal ROM is broken down into two parts: Internal ROM 0 (64 KB) and Internal ROM 1 (64 KB).

The Internal SRAM is broken down into two parts: Internal SRAM 0 (32 KB) and Internal SRAM 1 (288 KB).

RTC FAST Memory and RTC SLOW Memory are both implemented as SRAM.

Table 2 lists all types of internal memory and their address ranges on the data bus and instruction bus.

Table 2: Internal Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Data bus	0x3FF9_E000	0x3FF9_FFFF	8 KB	RTC FAST Memory	YES
	0x3FFA_0000	0x3FFA_FFFF	64 KB	Internal ROM 1	NO
	0x3FFB_0000	0x3FFB_7FFF	32 KB	Internal SRAM 0	YES
	0x3FFB_8000	0x3FFF_FFFF	288 KB	Internal SRAM 1	YES

Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Instruction bus	0x4000_0000	0x4000_FFFF	64 KB	Internal ROM 0	NO
	0x4001_0000	0x4001_FFFF	64 KB	Internal ROM 1	NO
	0x4002_0000	0x4002_7FFF	32 KB	Internal SRAM 0	YES
	0x4002_8000	0x4006_FFFF	288 KB	Internal SRAM 1	YES
	0x4007_0000	0x4007_1FFF	8 KB	RTC FAST Memory	YES
Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Data / Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	YES

Note:

"YES" in the "Permission Control" column indicates that a permission is required for memory access. Permission Control registers can be used to limit Instruction or Data bus access to individual regions of these memory types.

1.3.2.1 Internal ROM 0

Internal ROM 0 is a 64-KB, read-only memory space, addressed by the CPU on the instruction bus via range(s) described in Table 2.

1.3.2.2 Internal ROM 1

Internal ROM 1 is a 64-KB, read-only memory space, addressed by the CPU on the data or instruction bus via range(s) described in Table 2.

The two address ranges access Internal ROM 1 in the same order, so, for example, addresses 0x3FFA_0000 and 0x4001_0000 access the same word, 0x3FFA_0004 and 0x4001_0004 access the same word, 0x3FFA_0008 and 0x4001_0008 access the same word, etc.

1.3.2.3 Internal SRAM 0

Internal SRAM 0 is a 32-KB, read-and-write memory space, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 2.

Hardware can be configured to use 8 KB, 16 KB, 24 KB, or the entire 32 KB space in this memory to cache external memory. The space used as cache cannot be accessed by the CPU, while the remaining space can still be accessed by the CPU.

1.3.2.4 Internal SRAM 1

Internal SRAM 1 is a 288-KB, read-and-write memory space, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 2.

Internal SRAM 1 comprises eighteen 16-KB (sub)memory blocks. One block can be used as Trace Memory, in which case this block's address range cannot be accessed by the CPU.

1.3.2.5 RTC FAST Memory

RTC FAST Memory is an 8-KB, read-and-write SRAM, addressed by the CPU on the data or instruction bus, in the same order, via range(s) described in Table 2.

1.3.2.6 RTC SLOW Memory

RTC SLOW Memory is an 8-KB, read-and-write SRAM, addressed by the CPU via range(s) shared by the data bus and the instruction bus, as described in Table 2.

RTC SLOW Memory can also be used as a peripheral addressable to the CPU via either 0x3F42_1000 ~ 0x3F42_2FFF or 0x6002_1000 ~ 0x6002_2FFF on the data bus.

1.3.3 External Memory

ESP32-S2 supports multiple QSPI/OSPI flash and RAM chips. It also supports hardware encryption/decryption based on XTS-AES to protect user programs and data in the flash and external RAM.

1.3.3.1 External Memory Address Mapping

The CPU accesses the external flash and RAM via the cache. According to the MMU settings, the cache maps the CPU's address to the external physical memory address. Due to this address mapping, the ESP32-S2 can address up to 1 GB external flash and 1 GB external RAM.

Using the cache, ESP32-S2 can support the following address space mappings at the same time.

- Up to 7.5 MB instruction bus address space can be mapped into the external flash or RAM as individual 64 KB blocks, via the instruction cache (ICache). Byte (8-bit), half-word (16-bit) and word (32-bit) reads are supported.
- Up to 4 MB read-only data bus address space can be mapped into the external flash or RAM as individual 64 KB blocks, via ICache. Byte (8-bit), half-word (16-bit) and word (32-bit) reads are supported.
- Up to 10.5 MB data bus address space can be mapped into the external RAM as individual 64 KB blocks, via DCache. Byte (8-bit), half-word (16-bit) or word (32-bit) reads and writes are supported. Blocks from this 10.5 MB space can also be mapped into the external flash or RAM, for read operations only.

Table 3 lists the mapping between the cache and the corresponding address ranges on the data bus and instruction bus.

Table 3: External Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Permission Control
	Low Address	High Address			
Data bus	0x3F00_0000	0x3F3F_FFFF	4 MB	ICache	YES
Data bus	0x3F50_0000	0x3FF7_FFFF	10.5 MB	DCache	YES
Instruction bus	0x4008_0000	0x407F_FFFF	7.5 MB	ICache	YES

Note:

"YES" in the "Permission Control" column indicates that a permission is required for memory access. Permission Control registers can be used to limit Instruction or Data bus access to individual regions of these memory types.

1.3.3.2 Cache

As shown in Figure 1-2, the caches on ESP32-S2 are separated and allow prompt response upon simultaneous requests from the data bus and instruction bus. Some internal memory space can be used as cache (see Section 1.3.2.3). When a cache miss occurs, the cache controller will initiate a request to the external memory. When ICache and DCache simultaneously initiate a request, the arbiter determines which gets the access to the external memory first. The cache size of ICache and DCache can be configured to 8 KB and 16 KB, respectively, while their block size can be configured to 16 bytes and 32 bytes, respectively.

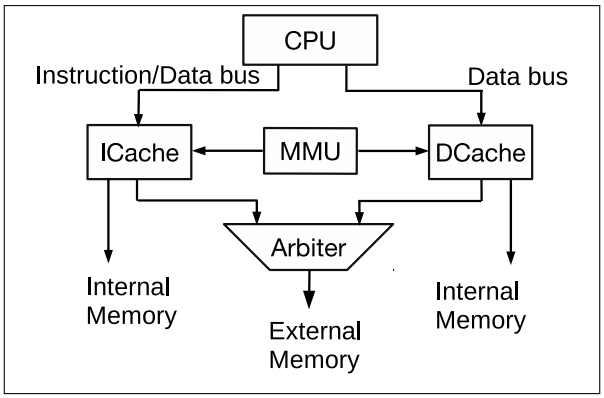


Figure 1-2. Cache Structure

1.3.3.3 Cache Operations

ESP32-S2 caches support the following operations:

1. **Invalidate**: The cache clears the valid bit of a tag. The CPU needs to access the external memory in order to read/write the data. There are two types of invalidation operations: manual invalidation and automatic invalidation. Manual invalidation performs only on data in the specified area in the cache, while automatic invalidation performs on all data in the cache. Both ICache and DCache have this function.
2. **Clean**: The cache clears the dirty bit of the tag and retains the valid bit. The CPU can then read/write the data directly from the cache. Only DCache has this function.
3. **Write-back**: The cache clears the dirty block flag of the tag and retains the valid bit. It also forces the data in the corresponding address to be written back to the external memory. The CPU can then read/write the data directly from the cache. Only DCache has this function.
4. **Preload**: To preload a cache is to load instructions and data into the cache in advance. The minimum unit of a preloading is one block. There are two types of preloading: manual preloading and automatic preloading. Manual preloading means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Automatic preloading means the hardware prefetches a piece of continuous data according to the current hit / miss address (depending on configuration).
5. **Lock / Unlock**: 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. When manual lock is enabled, the cache checks the data that has been filled into the cache memory and locks the data that falls in the specified area. The data in the locked area is always stored in the cache and will not be replaced. But when all the ways within the cache are locked, the cache will replace data, as if the ways were not locked. Unlocking is the reverse of locking. The manual invalidation, clean, and write-back operations are only available after unlocking.

1.3.4 DMA

With DMA, the ESP32-S2 can perform data transfers between:

- modules / peripherals and internal memory;
- different types of internal memory;
- modules / peripherals and external memory;
- internal and external memory.

DMA uses the same addressing as the data bus to read and write Internal SRAM 0 and Internal SRAM 1. Specifically, DMA uses address range 0x3FFB_0000 ~ 0x3FFB_7FFF to access Internal SRAM 0, and 0x3FFB_8000 ~ 0x3FFF_FFFF to access Internal SRAM 1. Note that DMA cannot access the internal memory occupied by the cache.

In addition, DMA addresses the external RAM from 0x3F50_0000 ~ 0x3FF7_FFFF, the same used by the CPU to access DCache. When DCache and DMA access the external memory simultaneously, data consistency is required.

Nine modules / peripherals on the ESP32-S2 support DMA, as shown in Table 4. With DMA, some of them can only access internal memory, some can access both internal and external memory.

Table 4: Peripherals with DMA Support

UART0	UART1
SPI2	SPI3
I2S0	
ADC Controller	
Copy DMA	
AES Accelerator	SHA Accelerator

1.3.5 Modules / Peripherals

The CPU can access modules / peripherals via address range 0x3F40_0000 ~ 0x3F4F_FFFF on the data bus, or via 0x6000_0000 ~ 0x600B_FFFF and 0x6180_0000 ~ 0x6180_3FFF shared by the data bus and instruction bus.

1.3.5.1 Naming Conventions for Peripheral Buses

There are two peripheral buses defined as follows:

- **PeriBus1**: Which refers to the address range 0x3F40_0000 ~ 0x3F4F_FFFF on the bus. 0x3F40_0000 is the base address.
- **PeriBus2**: Which refers to the address ranges 0x6000_0000 ~ 0x600B_FFFF and 0x6180_0000 ~ 0x6180_3FFF on the bus. 0x6000_0000 is the base address.

All references to "PeriBus1" and "PeriBus2" in this document indicate the corresponding address range(s).

1.3.5.2 Differences Between PeriBus1 and PeriBus2

The CPU can access modules / peripherals more efficiently through PeriBus1 than through PeriBus2. However, PeriBus1 features speculative reads, which means it cannot guarantee that each read is valid. Therefore, the CPU has to use PeriBus2 to access some special registers, for example, FIFO registers.

In addition, PeriBus1 will upset the order of r/w operations on the bus to improve performance, which may cause programs that have strict requirements on the r/w order to crash. In such cases, please add volatile before the program statement, or use PeriBus2 instead.

1.3.5.3 Module / Peripheral Address Mapping

Table 5 lists all the modules / peripherals and their respective address ranges. Note that addresses in column "Boundary Address" are offsets relative to the base address, instead of absolute addresses. The absolute addresses are the addition of bus base address and the corresponding offsets.

Table 5: Module / Peripheral Address Mapping

Target	Boundary Address		Size	Notes
	Low Address	High Address		
UART0	0x0000_0000	0x0000_0FFF	4 KB	1, 2, 3
Reserved	0x0000_1000	0x0000_1FFF		
SPI1	0x0000_2000	0x0000_2FFF	4 KB	1, 2
SPI0	0x0000_3000	0x0000_3FFF	4 KB	1, 2
GPIO	0x0000_4000	0x0000_4FFF	4 KB	1, 2
Reserved	0x0000_5000	0x0000_6FFF		
TIMER	0x0000_7000	0x0000_7FFF	4 KB	1, 2
RTC	0x0000_8000	0x0000_8FFF	4 KB	1, 2
IO MUX	0x0000_9000	0x0000_9FFF	4 KB	1, 2
Reserved	0x0000_A000	0x0000_EFFF		
I2S0	0x0000_F000	0x0000_FFFF	4 KB	1, 2, 3
UART1	0x0001_0000	0x0001_0FFF	4 KB	1, 2, 3
Reserved	0x0001_1000	0x0001_2FFF		
I2C0	0x0001_3000	0x0001_3FFF	4 KB	1, 2, 3
UHCI0	0x0001_4000	0x0001_4FFF	4 KB	1, 2
Reserved	0x0001_5000	0x0001_5FFF		
RMT	0x0001_6000	0x0001_6FFF	4 KB	1, 2, 3
PCNT	0x0001_7000	0x0001_7FFF	4 KB	1, 2
Reserved	0x0001_8000	0x0001_8FFF		
LED PWM Controller	0x0001_9000	0x0001_9FFF	4 KB	1, 2
eFuse Controller	0x0001_A000	0x0001_AFFF	4 KB	1, 2
Reserved	0x0001_B000	0x0001_EFFF		
Timer Group 0	0x0001_F000	0x0001_FFFF	4 KB	1, 2
Timer Group 1	0x0002_0000	0x0002_0FFF	4 KB	1, 2
RTC SLOW Memory	0x0002_1000	0x0002_2FFF	8 KB	1, 2, 3
System Timer	0x0002_3000	0x0002_3FFF	4 KB	1, 2
SPI2	0x0002_4000	0x0002_4FFF	4 KB	1, 2

Target	Boundary Address		Size	Notes
	Low Address	High Address		
SPI3	0x0002_5000	0x0002_5FFF	4 KB	1, 2
APB Controller	0x0002_6000	0x0002_6FFF	4 KB	1, 2
I2C1	0x0002_7000	0x0002_7FFF	4 KB	1, 2, 3
Reserved	0x0002_8000	0x0002_AFFF		
TWAI Controller	0x0002_B000	0x0002_BFFF	4 KB	1, 2
Reserved	0x0002_C000	0x0003_8FFF		
USB OTG	0x0003_9000	0x0003_9FFF	4 KB	1, 2, 3, 4
AES Accelerator	0x0003_A000	0x0003_AFFF	4 KB	1, 2
SHA Accelerator	0x0003_B000	0x0003_BFFF	4 KB	1, 2
RSA Accelerator	0x0003_C000	0x0003_CFFF	4 KB	1, 2
Digital Signature	0x0003_D000	0x0003_DFFF	4 KB	1, 2
HMAC	0x0003_E000	0x0003_EFFF	4 KB	1, 2
Crypto DMA	0x0003_F000	0x0003_FFFF	4 KB	1, 2
Reserved	0x0004_4000	0x000C_DFFF		
ADC Controller	0x0004_0000	0x0004_0FFF	4 KB	1, 2
Reserved	0x0004_1000	0x0007_FFFF		
USB OTG	0x0008_0000	0x000B_FFFF	256 KB	1, 2, 3, 4
System Registers	0x000C_0000	0x000C_0FFF	4 KB	1
Sensitive Register	0x000C_1000	0x000C_1FFF	4 KB	1
Interrupt Matrix	0x000C_2000	0x000C_2FFF	4 KB	1
Copy DMA	0x000C_3000	0x000C_3FFF	4 KB	1
Reserved	0x000C_4000	0x000C_EFFF		
Dedicated GPIO	0x000C_F000	0x000C_FFFF	4 KB	1
Reserved	0x000D_1000	0x000F_FFFF		
Configure Cache	0x0180_0000	0x0180_3FFF	16 KB	2

Note:

1. This module / peripheral can be accessed from PeriBus1.
2. This module / peripheral can be accessed from PeriBus2.
3. Some special addresses in this module / peripheral are not accessible from PeriBus1 (see Section 1.3.5.4).
4. The address space in this module / peripheral is not continuous.

1.3.5.4 Addresses with Restricted Access from PeriBus1

As mentioned in Section 1.3.5.2, PeriBus1 features speculative reads, which means it is forbidden to read FIFO registers. Table 6 below lists the address (range) with restricted access from PeriBus1.

There are four reserved user-defined registers that can be configured as needed to add more addresses with restricted access.

Table 6: Addresses with Restricted Access

Peripherals	Addresses with Restricted Access
UART0	0x3F40_0000
UART1	0x3F41_0000
I2S0	0x3F40_F004
RMT	0x3F41_6000 ~ 0x3F41_600F
I2C0	0x3F41_301C
I2C1	0x3F42_701C
USB OTG	0x3F48_0020, 0x3F48_1000 ~ 0x3F49_0FFF

2. Reset and Clock

2.1 Reset

2.1.1 Overview

ESP32-S2 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 2-1 shows the scopes of affected subsystems when different types of reset occur.

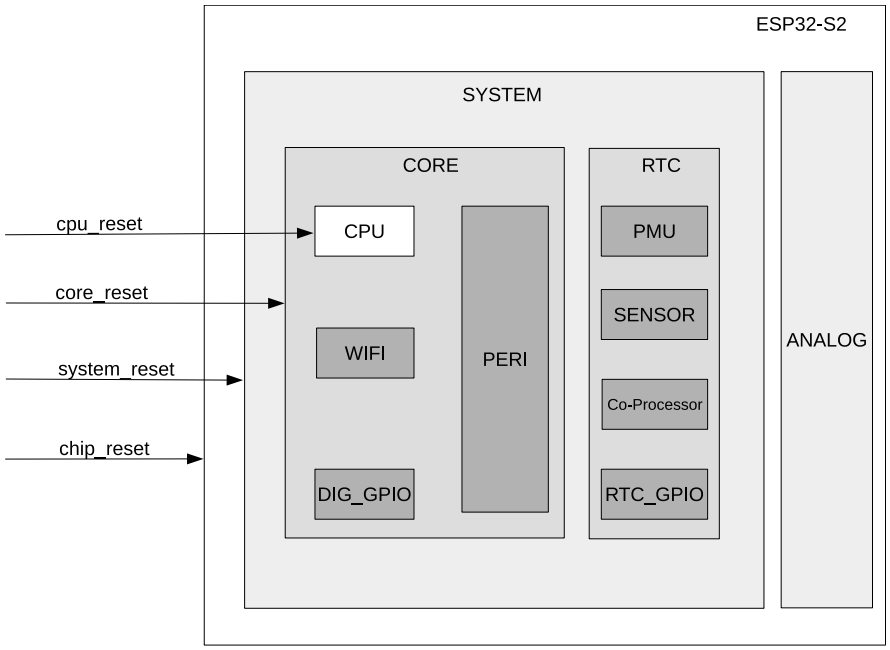


Figure 2-1. System Reset

- CPU Reset: Only resets CPU core. Once such reset is released, programs will be executed from CPU reset vector.
- Core Reset: Resets the whole digital system except RTC, including CPU, peripherals, Wi-Fi, and digital GPIOs.
- System Reset: Resets the whole digital system, including RTC.
- Chip Reset: Resets the whole chip.

2.1.2 Reset Source

CPU will be reset immediately when any of the reset above occurs. Users can get reset source codes by reading register RTC_CNTL_RESET_CAUSE_PROCPU after the reset is released.

Table 7 lists different reset sources and the types of reset they trigger.

Table 7: Reset Source

Code	Source	Reset Type	Comments
0x01	Chip reset	Chip Reset	See the note below
0x0F	Brown-out system reset	System Reset	Triggered by brown-out detector
0x10	RWDT system reset	System Reset	See Chapter 13 *Watchdog Timers*
0x13	GLITCH reset	System Reset	-
0x03	Software system reset	Core Reset	Triggered by configuring RTC_CNTL_SW_SYS_RST
0x05	Deep-sleep reset	Core Reset	See Chapter 28 *Low-power Management*
0x07	MWDT0 global reset	Core Reset	See Chapter 13 *Watchdog Timers*
0x08	MWDT1 global reset	Core Reset	See Chapter 13 *Watchdog Timers*
0x09	RWDT core reset	Core Reset	See Chapter 13 *Watchdog Timers*
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 13 *Watchdog Timers*
0x0C	Software CPU reset	CPU Reset	Triggered by configuring RTC_CNTL_SW_PROCPU_RST
0x0D	RWDT CPU reset	CPU Reset	See Chapter 13 *Watchdog Timers*
0x11	MWDT1 CPU reset	CPU Reset	See Chapter 13 *Watchdog Timers*

Note:

- Chip Reset can be triggered by the following three sources:
  - Triggered by chip power-on;
  - Triggered by brown-out detector;
  - Triggered by Super Watchdog (SWD).
- Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on register configuration. For more information, please see Chapter 28 *Low-power Management*.

2.2 Clock

2.2.1 Overview

ESP32-S2 provides multiple clock sources, which allow CPU, peripherals and RTC to work at different frequencies, thus providing more flexibility in meeting the requirements of various application scenarios. Figure 2-2 shows the system clock structure.

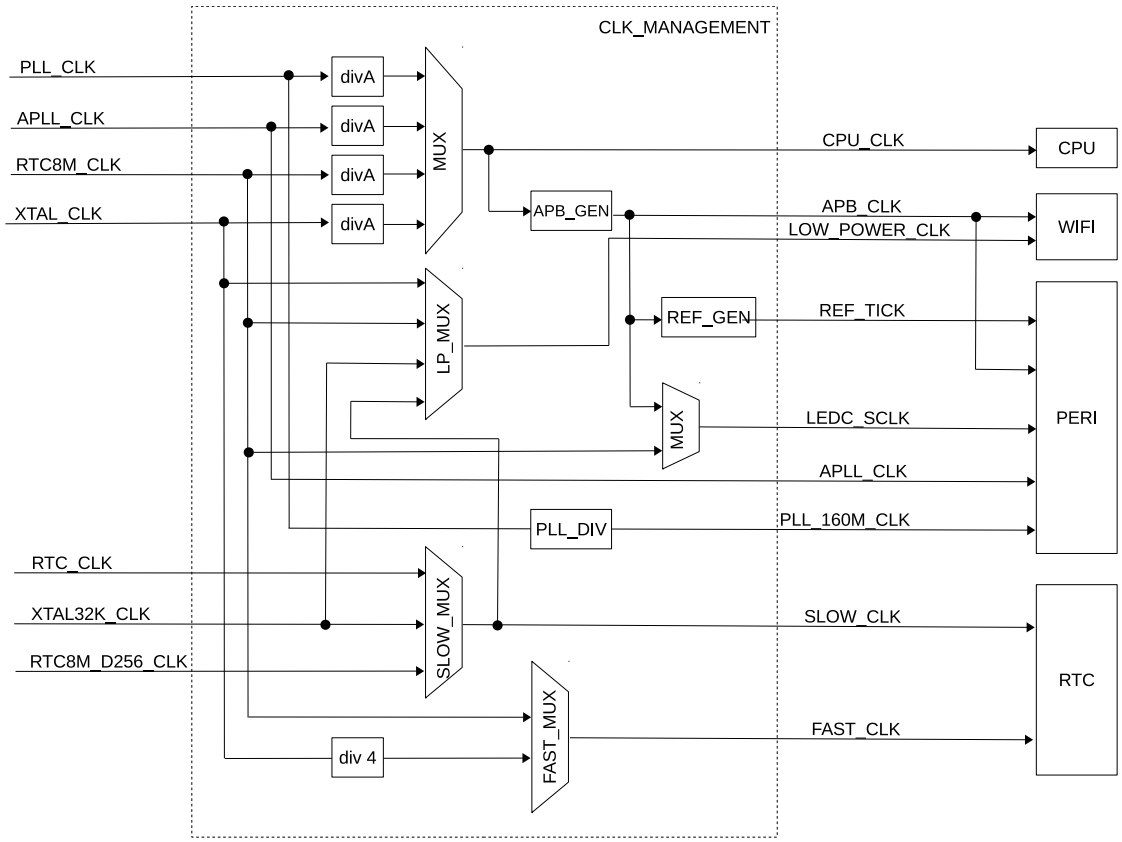


Figure 2-2. System Clock

2.2.2 Clock Source

ESP32-S2 uses external crystal, internal PLL, or internal oscillator working as clock sources to generate different kinds of clocks, which can be classified in three types depending on their clock speed.

- High speed clock for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_CLK (320 MHz or 480 MHz): internal PLL clock
  - XTAL_CLK (40 MHz): external crystal clock
- Slow speed clock for low-power devices, such as power management unit and low-power peripherals
  - XTAL32K_CLK (32 kHz): external crystal clock
  - RTC8M_CLK (8 MHz by default): internal oscillator with adjustable frequency
  - RTC8M_D256_CLK (31.250 kHz by default): internal clock derived from RTC8M_CLK divided by 256
  - RTC_CLK (90 kHz by default): internal oscillator with adjustable frequency
- Audio clock for audio-related devices
  - APLL_CLK (16 MHz ~ 128 MHz): internal Audio PLL clock

2.2.3 CPU Clock

As Figure 2-2 shows, CPU_CLK is the master clock for CPU and it can be as high as 240 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, APLL_CLK, RTC8M_CLK or XTAL_CLK as CPU_CLK clock source by configuring register SYSTEM_SOC_CLK_SEL, see Table 8 and Table 9.

Table 8: CPU_CLK Source

SYSTEM_SOC_CLK_SEL Value	Clock Source
0	XTAL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

Table 9: CPU_CLK Selection

Clock Source	SEL_0*	SEL_1*	SEL_2*	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 (480 MHz)	1	1	0	CPU_CLK = PLL_CLK / 6 CPU_CLK frequency is 80 MHz
PLL_CLK (480 MHz)	1	1	1	CPU_CLK = PLL_CLK / 3 CPU_CLK frequency is 160 MHz
PLL_CLK (480 MHz)	1	1	2	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 240 MHz
PLL_CLK (320 MHz)	1	0	0	CPU_CLK = PLL_CLK / 4 CPU_CLK frequency is 80 MHz
PLL_CLK (320 MHz)	1	0	1	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 160 MHz
RTC8M_CLK	2	-	-	CPU_CLK = RTC8M_CLK / (SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
APLL_CLK	3	0	0	CPU_CLK = APLL_CLK / 4
APLL_CLK	3	0	1	CPU_CLK = APLL_CLK / 2

*SEL_0: The value of register SYSTEM_SOC_CLK_SEL.

*SEL_1: The value of register SYSTEM_PLL_FREQ_SEL.

*SEL_2: The value of register SYSTEM_CPUPERIOD_SEL.

Note:

- When users select XTAL_CLK as CPU clock source and adjust the divider value by configuring register SYSTEM_PRE_DIV_CNT, the rules below should be followed.
  - If current divider value is 2 (SYSTEM_PRE_DIV_CNT = 1) and the target is x (x ≠ 1), users should set the divider first to 1 (SYSTEM_PRE_DIV_CNT = 0) and then to x (SYSTEM_PRE_DIV_CNT = x - 1).
  - If current divider value is x (SYSTEM_PRE_DIV_CNT = x - 1) and the target is 2, users should set the divider first to 1 (SYSTEM_PRE_DIV_CNT = 0) and then to 2 (SYSTEM_PRE_DIV_CNT = 1).
  - For other target divider value x, users can adjust the register directly (SYSTEM_PRE_DIV_CNT = x - 1).

2.2.4 Peripheral Clock

Peripheral clocks include APB_CLK, REF_TICK, LEDC_PWM_CLK, APLL_CLK and PLL_160M_CLK. Table 10 shows which clock can be used by which peripheral.

Table 10: Peripheral Clock Usage

Peripheral	APB_CLK	REF_TICK	LEDC_PWM_CLK	APLL_CLK	PLL_160M_CLK
TIMG	Y	Y			
I²S	Y			Y	Y
UHCI	Y				
UART	Y	Y			
RMT	Y	Y			
LED_PWM	Y	Y	Y		
I²C	Y	Y			
SPI	Y				
PCNT	Y				
eFuse Controller	Y				
SARADC/DAC	Y			Y	
USB	Y				
CRYPTO					Y
TWAI Controller	Y				
System Timer	Y				

2.2.4.1 APB_CLK Source

APB_CLK is determined by the clock source of CPU_CLK as shown in Table 11.

Table 11: APB_CLK Source

CPU_CLK Source	APB_CLK
PLL_CLK	80 MHz
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

2.2.4.2 REF_TICK Source

REF_TICK is derived from XTAL_CLK or RTC8M_CLK via a divider. When PLL_CLK, APLL_CLK or XTAL_CLK is set as CPU clock source, REF_TICK will be divided from XTAL_CLK. When RTC8M_CLK is set as CPU clock source, REF_TICK will be divided from RTC8M_CLK. In such way, REF_TICK frequency remains unchanged when APB_CLK changes its clock source. Table 12 shows the configuration of these clock divider registers.

Table 12: REF_TICK Source

CPU_CLK Source	Clock Divider Register
PLL_CLK | XTAL_CLK | APLL_CLK	APB_CTRL_XTAL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

Normally, one REF_TICK cycle lasts for 1 μs, so APB_CTRL_XTAL_TICK_NUM should be configured to 39 (default), and APB_CTRL_CK8M_TICK_NUM to 7 (default).

2.2.4.3 LEDC_PWM_CLK Source

LEDC_PWM_CLK clock source is selected by configuring register LEDC_APB_CLK_SEL, as shown in Table 13.

Table 13: LEDC_PWM_CLK Source

LEDC_APB_CLK_SEL Value	LEDC_PWM_CLK Source
0 (Default)	-
1	APB_CLK
2	RTC8M_CLK
3	XTAL_CLK

2.2.4.4 APLL_SCLK Source

APLL_CLK is sourced from PLL_CLK, and its output frequency is configured using APLL configuration registers. See Section 2.2.7 for more information.

2.2.4.5 PLL_160M_CLK Source

PLL_160M_CLK is divided from PLL_CLK according to current PLL frequency.

2.2.4.6 Clock Source Considerations

Peripherals that need to work with other clocks, such as RMT and I²C, generally operate using PLL_CLK frequency as a reference. When this frequency changes, peripherals should update their clock configuration to operate at the same frequency after the change. Peripherals accessing REF_TICK can continue operating normally without changing their clock configuration when switching clock sources. Please see Table 10.

LED module uses RTC8M_CLK as clock source when APB_CLK is disabled. In other words, when the system is in low-power mode, most peripherals will be halted (APB_CLK is turned off), but LED can work normally via RTC8M_CLK.

2.2.5 Wi-Fi Clock

Wi-Fi can work only when APB_CLK uses PLL_CLK as its clock source. Suspending PLL_CLK requires that Wi-Fi has entered low-power mode first.

LOW_POWER_CLK uses XTAL32K_CLK, XTAL_CLK, RTC8M_CLK or SLOW_CLK (the low clock selected by RTC) as its clock source for Wi-Fi in low-power mode.

2.2.6 RTC Clock

The clock sources for SLOW_CLK and FAST_CLK are low-frequency clocks. RTC module can operate when most other clocks are stopped.

SLOW_CLK derived from RTC_CLK, XTAL32K_CLK or RTC8M_D256_CLK is used to clock Power Management module.

FAST_CLK is used to clock On-chip Sensor module. It can be sourced from a divided XTAL_CLK or from RTC8M_CLK.

2.2.7 Audio PLL Clock

The operation of audio and other time-critical data-transfer applications requires highly-configurable, low-jitter and accurate clock sources. The clock sources derived from system clocks that serve digital peripherals may carry jitter and, therefore, are not suitable for a high-precision clock frequency setting.

Providing an integrated precision clock source can minimize system cost. To this end, ESP32-S2 integrates an audio PLL to clock I²S module.

Audio PLL formula is as follows:

$$f_{\text{out}} = \frac{f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4)}{2(odiv + 2)}$$

Parameters are defined below:

- f_{xtal}: the frequency of crystal oscillator, usually 40 MHz;
- sdm0: the value is 0 ~ 255;
- sdm1: the value is 0 ~ 255;
- sdm2: the value is 0 ~ 63;
- odiv: the value is 0 ~ 31;
- The operating frequency range of the numerator is 350 MHz ~ 500 MHz.

$$350MHz < f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4) < 500MHz$$

Audio PLL can be manually enabled or disabled via registers RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD, respectively. Disabling it takes priority over enabling it. When RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD are 0, PLL follows the state of the system. When the system enters sleep mode, PLL will be disabled automatically; when the system wakes up, PLL will be enabled automatically.

3. Chip Boot Control

3.1 Overview

ESP32-S2 has three strapping pins:

- GPIO0
- GPIO45
- GPIO46

Software can read the values of the three strapping pins from register GPIO_STRAPPING. During chip reset triggered by power-on, brown-out or by analog super watchdog (see Chapter 2 *Reset and Clock*), hardware samples and stores the voltage level of strapping pins as strapping bit of "0" or "1" in latches, and hold these bits until the chip is powered down or shut down.

By default, GPIO0, GPIO45 and GPIO46 are connected to internal pull-up/pull-down during chip reset. Consequently, if the three GPIOs are unconnected or their connected external circuits are high-impedance, the internal weak pull-up/pull-down will determine the default input level of the strapping pins (see Table 14).

Table 14: Default Configuration of Strapping Pins

Pin	GPIO0	GPIO45	GPIO46
Default	Pull-up	Pull-down	Pull-down

To change the default configuration of strapping pins, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-S2. After the reset is released, the strapping pins work as normal-function pins.

3.2 Boot Mode

GPIO0 and GPIO46 control the boot mode after reset.

Table 15: Boot Mode

Pin	SPI Boot	Download Boot
GPIO0	1	0
GPIO46	x	0

Table 15 shows the strapping pin values and the associated boot modes. "x" means that this value is ignored. Currently only the two boot modes shown are supported. The strapping combination of GPIO0 = 0 and GPIO46 = 1 is not supported and will trigger unexpected behavior.

In SPI boot mode, the CPU boots the system by reading the program stored in SPI flash.

In download boot mode, users can download code to SRAM or Flash using UART0, UART1, QPI or USB interface. It is also possible to load a program into SRAM and execute it in this mode.

The following eFuses control boot mode behavior:

- EFUSE_DIS_FORCE_DOWNLOAD. If this eFuse is 0 (default), software can switch the chip from SPI boot mode to download boot mode by setting register RTC_CNTL_FORCE_DOWNLOAD_BOOT and triggering

a CPU reset. If this eFuse is 1, this register is disabled.

- EFUSE_DIS_DOWNLOAD_MODE. If this eFuse is 1, download boot mode is disabled.
- EFUSE_ENABLE_SECURITY_DOWNLOAD. If this eFuse is 1, download boot mode only allows reading, writing and erasing plaintext flash and does not support any SRAM or register operations. This eFuse is ignored if download boot mode is disabled.

3.3 ROM Code Printing to UART

GPIO46 controls ROM code printing of information during the early boot process. This GPIO is used together with the UART_PRINT_CONTROL eFuse.

Table 16: ROM Code Printing Control

UART_PRINT_CONTROL	GPIO46	ROM Code Printing
0	-	ROM code will always print information to UART during boot. GPIO46 is not used.
1	0	Print is enabled during boot
	1	Print is disabled
2	0	Print is disabled
	1	Print is enabled during boot
3	-	Print is always disabled during boot. GPIO46 is not used.

ROM code will print to pin U0TXD (default) or to pin DAC_1, depending on the eFuse bit UART_PRINT_CHANNEL (0: UART0; 1: DAC_1).

3.4 VDD_SPI Voltage

GPIO45 is used to select the VDD_SPI power supply voltage at reset:

- GPIO45 = 0, VDD_SPI pin is powered directly from VDD3P3_RTC_IO via resistor R_{SPI}. Typically this voltage is 3.3 V.
- GPIO45 = 1, VDD_SPI pin is powered from internal 1.8 V LDO.

This functionality can be overridden by setting eFuse bit VDD_SPI_FORCE to 1, in which case the VDD_SPI_TIEH eFuse value determines the VDD_SPI voltage:

- VDD_SPI_FORCE = 1 and VDD_SPI_TIEH = 0, VDD_SPI connects to 1.8 V LDO.
- VDD_SPI_FORCE = 1 and VDD_SPI_TIEH = 1, VDD_SPI connects to VDD3P3_RTC_IO.

4. Interrupt Matrix

4.1 Overview

The interrupt matrix embedded in ESP32-S2 independently allocates peripheral interrupt sources to the CPU peripheral interrupts, so as to timely inform the CPU to process the interrupts once the interrupt signals are generated. This flexible function is applicable to a variety of application scenarios.

4.2 Features

- Accept 95 peripheral interrupt sources as input
- Generate 26 peripheral interrupts to the CPU as output
- Disable CPU non-maskable interrupt (NMI) sources
- Query current interrupt status of peripheral interrupt sources

The structure of the interrupt matrix is shown in Figure 4-1.

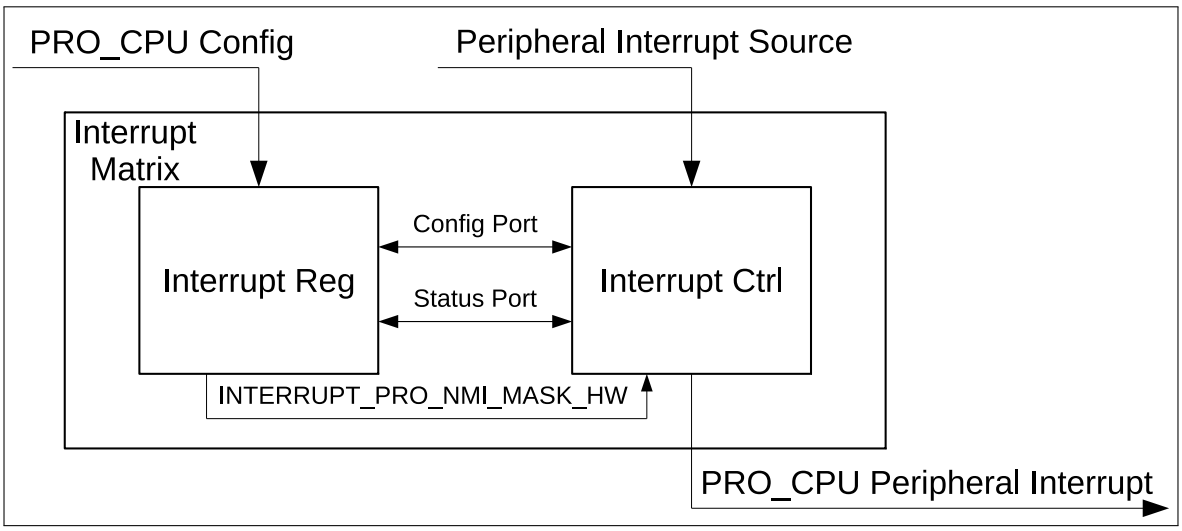


Figure 4-1. Interrupt Matrix Structure

4.3 Functional Description

4.3.1 Peripheral Interrupt Sources

ESP32-S2 has 95 peripheral interrupt sources in total, all of which can be allocated to the CPU. For the peripheral interrupt sources and their configuration/status registers, please refer to Table 17.

Table 17: CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources

No.	Source	Configuration Register	Status Register	
			Bit	Name
0	MAC_INTR	INTERRUPT_PRO_MAC_INTR_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_0_REG
1	MAC_NMI	INTERRUPT_PRO_MAC_NMI_MAP_REG	1	
2	PWR_INTR	INTERRUPT_PRO_PWR_INTR_MAP_REG	2	
3	BB_INT	INTERRUPT_PRO_BB_INT_MAP_REG	3	
4	BT_MAC_INT	INTERRUPT_PRO_BT_MAC_INT_MAP_REG	4	
5	BT_BB_INT	INTERRUPT_PRO_BT_BB_INT_MAP_REG	5	
6	BT_BB_NMI	INTERRUPT_PRO_BT_BB_NMI_MAP_REG	6	
7	RWBT_IRQ	INTERRUPT_PRO_RWBT_IRQ_MAP_REG	7	
8	RWBLE_IRQ	INTERRUPT_PRO_RWBLE_IRQ_MAP_REG	8	
9	RWBT_NMI	INTERRUPT_PRO_RWBT_NMI_MAP_REG	9	
10	RWBLE_NMI	INTERRUPT_PRO_RWBLE_NMI_MAP_REG	10	
11	SLC0_INTR	INTERRUPT_PRO_SLC0_INTR_MAP_REG	11	
12	SLC1_INTR	INTERRUPT_PRO_SLC1_INTR_MAP_REG	12	
13	UHCI0_INTR	INTERRUPT_PRO_UHCI0_INTR_MAP_REG	13	
14	UHCI1_INTR	INTERRUPT_PRO_UHCI1_INTR_MAP_REG	14	
15	TG_T0_LEVEL_INT	INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG	15	
16	TG_T1_LEVEL_INT	INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG	16	
17	TG_WDT_LEVEL_INT	INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG	17	
18	TG_LACT_LEVEL_INT	INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG	18	
19	TG1_T0_LEVEL_INT	INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG	19	
20	TG1_T1_LEVEL_INT	INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG	20	
21	TG1_WDT_LEVEL_INT	INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	21	
22	TG1_LACT_LEVEL_INT	INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	22	
23	GPIO_INTERRUPT_PRO	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG	23	
24	GPIO_INTERRUPT_PRO_NMI	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	24	
25	GPIO_INTERRUPT_APP	INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP_REG	25	
26	GPIO_INTERRUPT_APP_NMI	INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP_REG	26	
27	DEDICATED_GPIO_IN_INTR	INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG	27	
28	CPU_INTR_FROM_CPU_0	INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	28	
29	CPU_INTR_FROM_CPU_1	INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	29	
30	CPU_INTR_FROM_CPU_2	INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	30	
31	CPU_INTR_FROM_CPU_3	INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	31	
32	SPI_INTR_1	INTERRUPT_PRO_SPI_INTR_1_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
33	SPI_INTR_2	INTERRUPT_PRO_SPI_INTR_2_MAP_REG	1	
34	SPI_INTR_3	INTERRUPT_PRO_SPI_INTR_3_MAP_REG	2	

No.	Source	Configuration Register	Status Register	
			Bit	Name
35	I2S0_INT	INTERRUPT_PRO_I2S0_INT_MAP_REG	3	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
36	I2S1_INT	INTERRUPT_PRO_I2S1_INT_MAP_REG	4	
37	UART_INTR	INTERRUPT_PRO_UART_INTR_MAP_REG	5	
38	UART1_INTR	INTERRUPT_PRO_UART1_INTR_MAP_REG	6	
39	UART2_INTR	INTERRUPT_PRO_UART2_INTR_MAP_REG	7	
40	SDIO_HOST_INTERRUPT	INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP_REG	8	
41	PWM0_INTR	INTERRUPT_PRO_PWM0_INTR_MAP_REG	9	
42	PWM1_INTR	INTERRUPT_PRO_PWM1_INTR_MAP_REG	10	
43	PWM2_INTR	INTERRUPT_PRO_PWM2_INTR_MAP_REG	11	
44	PWM3_INTR	INTERRUPT_PRO_PWM3_INTR_MAP_REG	12	
45	LEDC_INT	INTERRUPT_PRO_LEDC_INT_MAP_REG	13	
46	EFUSE_INT	INTERRUPT_PRO_EFUSE_INT_MAP_REG	14	
47	CAN_INT	INTERRUPT_PRO_CAN_INT_MAP_REG	15	
48	USB_INTR	INTERRUPT_PRO_USB_INTR_MAP_REG	16	
49	RTC_CORE_INTR	INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG	17	
50	RMT_INTR	INTERRUPT_PRO_RMT_INTR_MAP_REG	18	
51	PCNT_INTR	INTERRUPT_PRO_PCNT_INTR_MAP_REG	19	
52	I2C_EXT0_INTR	INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG	20	
53	I2C_EXT1_INTR	INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG	21	
54	RSA_INTR	INTERRUPT_PRO_RSA_INTR_MAP_REG	22	
55	SHA_INTR	INTERRUPT_PRO_SHA_INTR_MAP_REG	23	
56	AES_INTR	INTERRUPT_PRO_AES_INTR_MAP_REG	24	
57	SPI2_DMA_INT	INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG	25	
58	SPI3_DMA_INT	INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG	26	
59	WDG_INT	INTERRUPT_PRO_WDG_INT_MAP_REG	27	
60	TIMER_INT	INTERRUPT_PRO_TIMER_INT1_MAP_REG	28	
61	TIMER_INT2	INTERRUPT_PRO_TIMER_INT2_MAP_REG	29	
62	TG_T0_EDGE_INT	INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	30	
63	TG_T1_EDGE_INT	INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	31	
64	TG_WDT_EDGE_INT	INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_2_REG
65	TG_LACT_EDGE_INT	INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	1	
66	TG1_T0_EDGE_INT	INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	2	
67	TG1_T1_EDGE_INT	INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	3	
68	TG1_WDT_EDGE_INT	INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	4	
69	TG1_LACT_EDGE_INT	INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	5	
70	CACHE_IA_INT	INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	6	
71	SYSTIMER_TARGET0_INT	INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	7	

No.	Source	Configuration Register	Status Register	
			Bit	Name
72	SYSTIMER_TARGET1_INT	INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	8	
73	SYSTIMER_TARGET2_INT	INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	9	
74	ASSIST_DEBUG_INTR	INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	10	
75	PMS_PRO_IRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	11	
76	PMS_PRO_DRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	12	
77	PMS_PRO_DPORT_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	13	
78	PMS_PRO_AHB_ILG_INTR	INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	14	
79	PMS_PRO_CACHE_ILG_INTR	INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	15	
80	PMS_DMA_APB_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG	16	
81	PMS_DMA_RX_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG	17	
82	PMS_DMA_TX_I_ILG_INTR	INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG	18	
83	SPI_MEM_REJECT_INTR	INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG	19	
84	DMA_COPY_INTR	INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG	20	
85	SPI4_DMA_INT	INTERRUPT_PRO_SPI4_DMA_INT_MAP_REG	21	
86	DCACHE_PRELOAD_INT	INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	22	
87	DCACHE_PRELOAD_INT	INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	23	INTERRUPT_PRO_INTR_STATUS_REG_2_REG
88	ICACHE_PRELOAD_INT	INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG	24	
89	APB_ADC_INT	INTERRUPT_PRO_APB_ADC_INT_MAP_REG	25	
90	CRYPTO_DMA_INT	INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG	26	
91	CPU_PERI_ERROR_INT	INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG	27	
92	APB_PERI_ERROR_INT	INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG	28	
93	DCACHE_SYNC_INT	INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG	29	
94	ICACHE_SYNC_INT	INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG	30	

4.3.2 CPU Interrupts

The CPU has 32 interrupts, including 26 peripheral interrupts and six internal interrupts. Table 18 lists all the interrupts.

- Peripheral Interrupts:
  - Level-triggered interrupts: triggered by high level signal. The interrupt sources should hold the level till the CPU handles the interrupts.
  - Edge-triggered interrupts: triggered on rising edge. The CPU responds to this kind of interrupts immediately.
  - NMI interrupt: once triggered, the NMI interrupt can not be masked by software using the CPU registers.
- Internal Interrupts:
  - Timer interrupts: triggered by internal timers and are used to generate periodic interrupts.
  - Software interrupts: triggered when software writes to special registers.
  - Profiling interrupt: triggered for performance monitoring and analysis.

ESP32-S2 supports the above-mentioned 32 interrupts at six levels as shown in the table below. A higher level corresponds to a higher priority. NMI has the highest interrupt priority and once triggered, the CPU must handle such interrupt.

Table 18: CPU Interrupts

No.	Category	Type	Priority Level
0	Peripheral	Level-triggered	1
1	Peripheral	Level-triggered	1
2	Peripheral	Level-triggered	1
3	Peripheral	Level-triggered	1
4	Peripheral	Level-triggered	1
5	Peripheral	Level-triggered	1
6	Internal	Timer.0	1
7	Internal	Software	1
8	Peripheral	Level-triggered	1
9	Peripheral	Level-triggered	1
10	Peripheral	Edge-triggered	1
11	Internal	Profiling	3
12	Peripheral	Level-triggered	1
13	Peripheral	Level-triggered	1
14	Peripheral	NMI	NMI
15	Internal	Timer.1	3
16	Internal	Timer.2	5
17	Peripheral	Level-triggered	1
18	Peripheral	Level-triggered	1
19	Peripheral	Level-triggered	2
20	Peripheral	Level-triggered	2

No.	Category	Type	Priority Level
21	Peripheral	Level-triggered	2
22	Peripheral	Edge-triggered	3
23	Peripheral	Level-triggered	3
24	Peripheral	Level-triggered	4
25	Peripheral	Level-triggered	4
26	Peripheral	Level-triggered	5
27	Peripheral	Level-triggered	3
28	Peripheral	Edge-triggered	4
29	Internal	Software	3
30	Peripheral	Edge-triggered	4
31	Peripheral	Level-triggered	5

4.3.3 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 particular peripheral interrupt source, wherein, X means the number of this interrupt source in Table 17.
- INTERRUPT_PRO_X_MAP_REG: stands for a peripheral interrupt configuration register, corresponding to the peripheral interrupt source Source_X. In Table 17, the registers listed in column "Configuration Register" correspond to the peripheral interrupt sources listed in column "Source". For example, the configuration register for source MAC_INTR is INTERRUPT_PRO_MAC_INTR_MAP_REG.
- Interrupt_P: stands for the CPU peripheral interrupt numbered as Num_P. The value of Num_P can be 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28 and 30 ~ 31 (see Table 18).
- Interrupt_I: stands for the CPU internal interrupt numbered as Num_I. The value of Num_I can be 6, 7, 11, 15, 16 and 29 (see Table 18).

4.3.3.1 Allocate one peripheral interrupt source Source_X to CPU

Setting the corresponding configuration register INTERRUPT_PRO_X_MAP_REG of Source_X to Num_P will allocate this interrupt source to Interrupt_P. Num_P here can be any value from 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28 and 30 ~ 31. Note that one CPU interrupt can be shared by multiple peripherals.

4.3.3.2 Allocate multiple peripheral interrupt sources Source_X*n* to CPU

Setting the corresponding configuration register INTERRUPT_PRO_X*n*_MAP_REG of each interrupt source to the same Num_P will allocate all the sources to the same Interrupt_P. Any of these sources will trigger CPU Interrupt_P. When an interrupt signal is generated, software should check the interrupt status registers to figure out which peripheral the signal comes from.

4.3.3.3 Disable CPU peripheral interrupt source Source_X

Setting the corresponding configuration register INTERRUPT_PRO_X_MAP_REG of the source to any Num_I will disable this interrupt Source_X. The choice of Num_I does not matter, as none of Num_I is connected to the CPU. Therefore this functionality can be used to disable peripheral interrupt sources.

4.3.4 Disable CPU NMI Interrupt Sources

The interrupt matrix is able to mask all peripheral interrupt sources allocated to CPU No.14 NMI interrupt using hardware, depending on the internal signal INTERRUPT_PRO_NMI_MASK_HW. The signal comes from "Interrupt Reg" register configuration submodule inside interrupt matrix, see Figure 4-1. If the signal is set to high level, CPU will not respond to NMI interrupt.

4.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

Current interrupt status of a peripheral interrupt source can be read via the bit value in INTERRUPT_PRO_INTR_STATUS_REG_*n*. For the mapping between INTERRUPT_PRO_INTR_STATUS_REG_*n* and peripheral interrupt sources, please refer to Table 17.

4.4 Base Address

Users can access interrupt matrix with the base address as shown in Table 19. For more information, see Chapter 1 *System and Memory*.

Table 19: Interrupt Matrix Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C2000

4.5 Register Summary

The address in the following table represents the address offset (relative address) with the respect to the peripheral base address, not the absolute address. For detailed information about the interrupt matrix base address, please refer to Section 4.4.

Name	Description	Address	Access
Configuration registers			
INTERRUPT_PRO_MAC_INTR_MAP_REG	MAC_INTR interrupt configuration register	0x0000	R/W
INTERRUPT_PRO_MAC_NMI_MAP_REG	MAC_NMI interrupt configuration register	0x0004	R/W
INTERRUPT_PRO_PWR_INTR_MAP_REG	PWR_INTR interrupt configuration register	0x0008	R/W
INTERRUPT_PRO_BB_INT_MAP_REG	BB_INT interrupt configuration register	0x000C	R/W
INTERRUPT_PRO_BT_MAC_INT_MAP_REG	BT_MAC_INT interrupt configuration register	0x0010	R/W
INTERRUPT_PRO_BT_BB_INT_MAP_REG	BT_BB_INT interrupt configuration register	0x0014	R/W
INTERRUPT_PRO_BT_BB_NMI_MAP_REG	BT_BB_NMI interrupt configuration register	0x0018	R/W
INTERRUPT_PRO_RWBT_IRQ_MAP_REG	RWBT_IRQ interrupt configuration register	0x001C	R/W
INTERRUPT_PRO_RWBLE_IRQ_MAP_REG	RWBLE_IRQ interrupt configuration register	0x0020	R/W
INTERRUPT_PRO_RWBT_NMI_MAP_REG	RWBT_NMI interrupt configuration register	0x0024	R/W
INTERRUPT_PRO_RWBLE_NMI_MAP_REG	RWBLE_NMI interrupt configuration register	0x0028	R/W
INTERRUPT_PRO_SLC0_INTR_MAP_REG	SLC0_INTR interrupt configuration register	0x002C	R/W
INTERRUPT_PRO_SLC1_INTR_MAP_REG	SLC1_INTR interrupt configuration register	0x0030	R/W
INTERRUPT_PRO_UHCI0_INTR_MAP_REG	UHCI0_INTR interrupt configuration register	0x0034	R/W
INTERRUPT_PRO_UHCI1_INTR_MAP_REG	UHCI1_INTR interrupt configuration register	0x0038	R/W
INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG	TG_T0_LEVEL_INT interrupt configuration register	0x003C	R/W
INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG	TG_T1_LEVEL_INT interrupt configuration register	0x0040	R/W
INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG	TG_WDT_LEVEL_INT interrupt configuration register	0x0044	R/W
INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG	TG_LACT_LEVEL_INT interrupt configuration register	0x0048	R/W
INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG	TG1_T0_LEVEL_INT interrupt configuration register	0x004C	R/W
INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG	TG1_T1_LEVEL_INT interrupt configuration register	0x0050	R/W
INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	TG1_WDT_LEVEL_INT interrupt configuration register	0x0054	R/W
INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	TG1_LACT_LEVEL_INT interrupt configuration register	0x0058	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO interrupt configuration register	0x005C	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	GPIO_INTERRUPT_PRO_NMI interrupt configuration register	0x0060	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP_REG	GPIO_INTERRUPT_APP interrupt configuration register	0x0064	R/W
INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP_REG	GPIO_INTERRUPT_APP_NMI interrupt configuration register	0x0068	R/W
INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG	DEDICATED_GPIO_IN_INTR interrupt configuration register	0x006C	R/W

Name	Description	Address	Access
INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 interrupt configuration register	0x0070	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 interrupt configuration register	0x0074	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 interrupt configuration register	0x0078	R/W
INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 interrupt configuration register	0x007C	R/W
INTERRUPT_PRO_SPI_INTR_1_MAP_REG	SPI_INTR_1 interrupt configuration register	0x0080	R/W
INTERRUPT_PRO_SPI_INTR_2_MAP_REG	SPI_INTR_2 interrupt configuration register	0x0084	R/W
INTERRUPT_PRO_SPI_INTR_3_MAP_REG	SPI_INTR_3 interrupt configuration register	0x0088	R/W
INTERRUPT_PRO_I2S0_INT_MAP_REG	I2S0_INT interrupt configuration register	0x008C	R/W
INTERRUPT_PRO_I2S1_INT_MAP_REG	I2S1_INT interrupt configuration register	0x0090	R/W
INTERRUPT_PRO_UART_INTR_MAP_REG	UART_INT interrupt configuration register	0x0094	R/W
INTERRUPT_PRO_UART1_INTR_MAP_REG	UART1_INT interrupt configuration register	0x0098	R/W
INTERRUPT_PRO_UART2_INTR_MAP_REG	UART2_INT interrupt configuration register	0x009C	R/W
INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP_REG	SDIO_HOST_INTERRUPT configuration register	0x00A0	R/W
INTERRUPT_PRO_PWM0_INTR_MAP_REG	PWM0_INTR interrupt configuration register	0x00A4	R/W
INTERRUPT_PRO_PWM1_INTR_MAP_REG	PWM1_INTR interrupt configuration register	0x00A8	R/W
INTERRUPT_PRO_PWM2_INTR_MAP_REG	PWM2_INTR interrupt configuration register	0x00AC	R/W
INTERRUPT_PRO_PWM3_INTR_MAP_REG	PWM3_INTR interrupt configuration register	0x00B0	R/W
INTERRUPT_PRO_LEDC_INT_MAP_REG	LEDC_INTR interrupt configuration register	0x00B4	R/W
INTERRUPT_PRO_EFUSE_INT_MAP_REG	EFUSE_INT interrupt configuration register	0x00B8	R/W
INTERRUPT_PRO_CAN_INT_MAP_REG	CAN_INT interrupt configuration register	0x00BC	R/W
INTERRUPT_PRO_USB_INTR_MAP_REG	USB_INT interrupt configuration register	0x00C0	R/W
INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG	RTC_CORE_INTR interrupt configuration register	0x00C4	R/W
INTERRUPT_PRO_RMT_INTR_MAP_REG	RMT_INTR interrupt configuration register	0x00C8	R/W
INTERRUPT_PRO_PCNT_INTR_MAP_REG	PCNT_INTR interrupt configuration register	0x00CC	R/W
INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG	I2C_EXT0_INTR interrupt configuration register	0x00D0	R/W
INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG	I2C_EXT1_INTR interrupt configuration register	0x00D4	R/W
INTERRUPT_PRO_RSA_INTR_MAP_REG	RSA_INTR interrupt configuration register	0x00D8	R/W
INTERRUPT_PRO_SHA_INTR_MAP_REG	SHA_INTR interrupt configuration register	0x00DC	R/W
INTERRUPT_PRO_AES_INTR_MAP_REG	AES_INTR interrupt configuration register	0x00E0	R/W

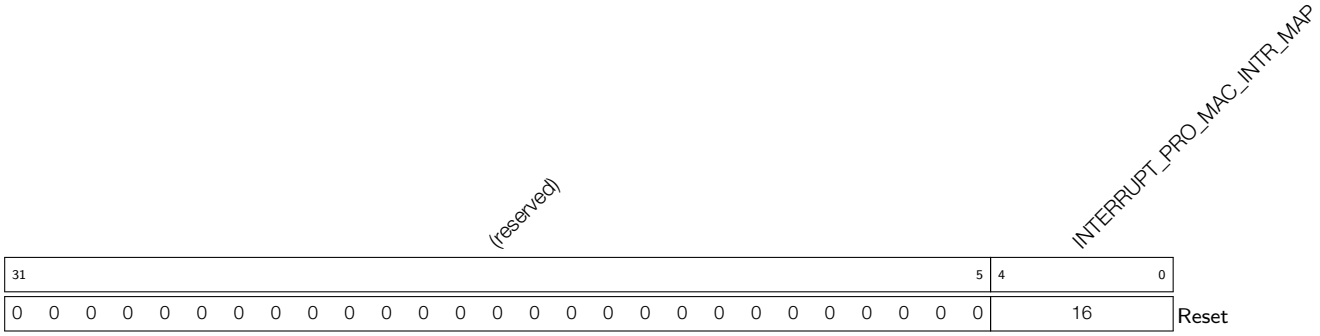
Name	Description	Address	Access
INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG	SPI2_DMA_INT interrupt configuration register	0x00E4	R/W
INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG	SPI3_DMA_INT interrupt configuration register	0x00E8	R/W
INTERRUPT_PRO_WDG_INT_MAP_REG	WDG_INT interrupt configuration register	0x00EC	R/W
INTERRUPT_PRO_TIMER_INT1_MAP_REG	TIMER_INT1 interrupt configuration register	0x00F0	R/W
INTERRUPT_PRO_TIMER_INT2_MAP_REG	TIMER_INT2 interrupt configuration register	0x00F4	R/W
INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	TG_T0_EDGE_INT interrupt configuration register	0x00F8	R/W
INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	TG_T1_EDGE_INT interrupt configuration register	0x00FC	R/W
INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	TG_WDT_EDGE_INT interrupt configuration register	0x0100	R/W
INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	TG_LACT_EDGE_INT interrupt configuration register	0x0104	R/W
INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	TG1_T0_EDGE_INT interrupt configuration register	0x0108	R/W
INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	TG1_T1_EDGE_INT interrupt configuration register	0x010C	R/W
INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	TG1_WDT_EDGE_INT interrupt configuration register	0x0110	R/W
INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	TG1_LACT_EDGE_INT interrupt configuration register	0x0114	R/W
INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	CACHE_IA_INT interrupt configuration register	0x0118	R/W
INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INT interrupt configuration register	0x011C	R/W
INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INT interrupt configuration register	0x0120	R/W
INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2 interrupt configuration register	0x0124	R/W
INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR interrupt configuration register	0x0128	R/W
INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	PMS_PRO_IRAM0_ILG interrupt configuration register	0x012C	R/W
INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	PMS_PRO_DRAM0_ILG interrupt configuration register	0x0130	R/W
INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	PMS_PRO_DPORT_ILG interrupt configuration register	0x0134	R/W
INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	PMS_PRO_AHB_ILG interrupt configuration register	0x0138	R/W
INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	PMS_PRO_CACHE_ILG interrupt configuration register	0x013C	R/W
INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG	PMS_DMA_APB_I_ILG interrupt configuration register	0x0140	R/W
INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG	PMS_DMA_RX_I_ILG interrupt configuration register	0x0144	R/W
INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG	PMS_DMA_TX_I_ILG interrupt configuration register	0x0148	R/W
INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG	SPI_MEM_REJECT_INTR interrupt configuration register	0x014C	R/W
INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG	DMA_COPY_INTR interrupt configuration register	0x0150	R/W
INTERRUPT_PRO_SPI4_DMA_INT_MAP_REG	SPI4_DMA_INT interrupt configuration register	0x0154	R/W

Name	Description	Address	Access
INTERRUPT_PRO_SPI_INTR_4_MAP_REG	SPI_INTR_4 interrupt configuration register	0x0158	R/W
INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	DCACHE_PRELOAD_INT interrupt configuration register	0x015C	R/W
INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG	ICACHE_PRELOAD_INT interrupt configuration register	0x0160	R/W
INTERRUPT_PRO_APB_ADC_INT_MAP_REG	APB_ADC_INT interrupt configuration register	0x0164	R/W
INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG	CRYPTO_DMA_INT interrupt configuration register	0x0168	R/W
INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG	CPU_PERI_ERROR_INT interrupt configuration register	0x016C	R/W
INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG	APB_PERI_ERROR_INT interrupt configuration register	0x0170	R/W
INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG	DCACHE_SYNC_INT interrupt configuration register	0x0174	R/W
INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG	ICACHE_SYNC_INT interrupt configuration register	0x0178	R/W
INTERRUPT_CLOCK_GATE_REG	NMI interrupt signals mask register	0x0188	R/W
Interrupt status registers			
INTERRUPT_PRO_INTR_STATUS_REG_0_REG	Interrupt status register 0	0x017C	RO
INTERRUPT_PRO_INTR_STATUS_REG_1_REG	Interrupt status register 1	0x0180	RO
INTERRUPT_PRO_INTR_STATUS_REG_2_REG	Interrupt status register 2	0x0184	RO
Version register			
INTERRUPT_REG_DATE_REG	Version control register	0x0FFC	R/W

4.6 Registers

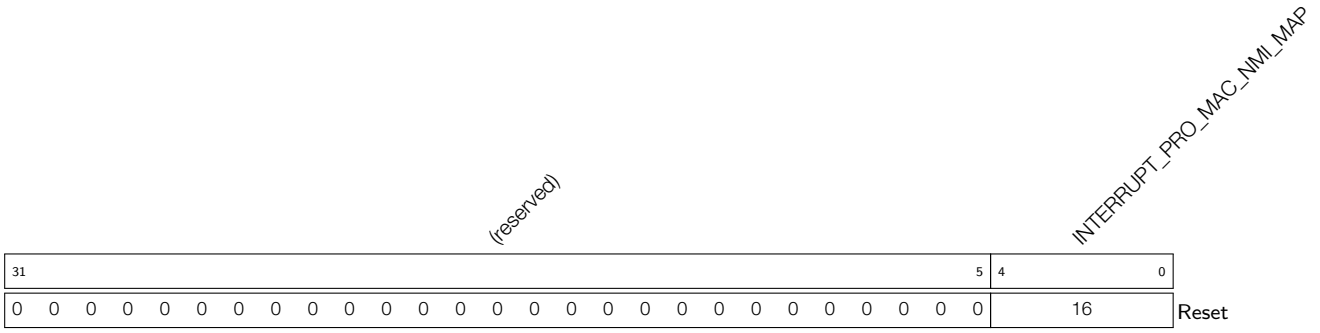
The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the interrupt matrix base address, please refer to Section 4.4.

Register 4.1: INTERRUPT_PRO_MAC_INTR_MAP_REG (0x0000)



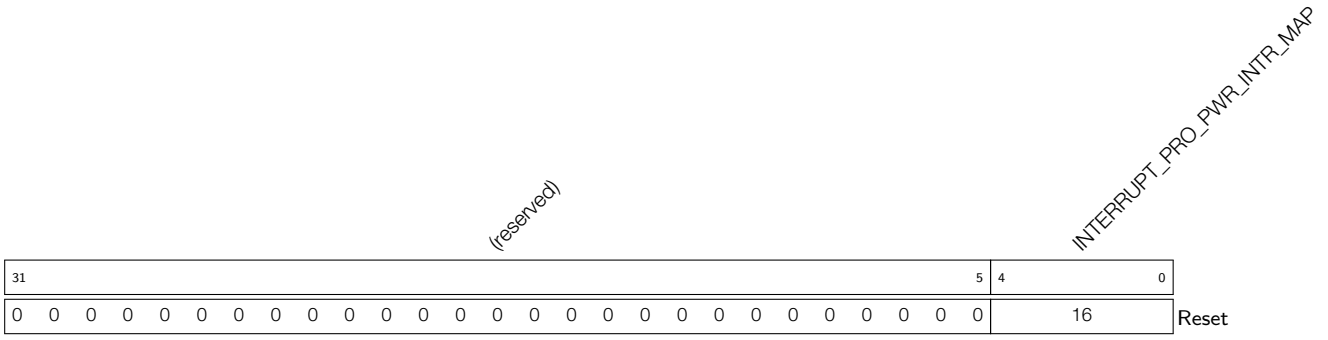
INTERRUPT_PRO_MAC_INTR_MAP This register is used to map MAC_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.2: INTERRUPT_PRO_MAC_NMI_MAP_REG (0x0004)



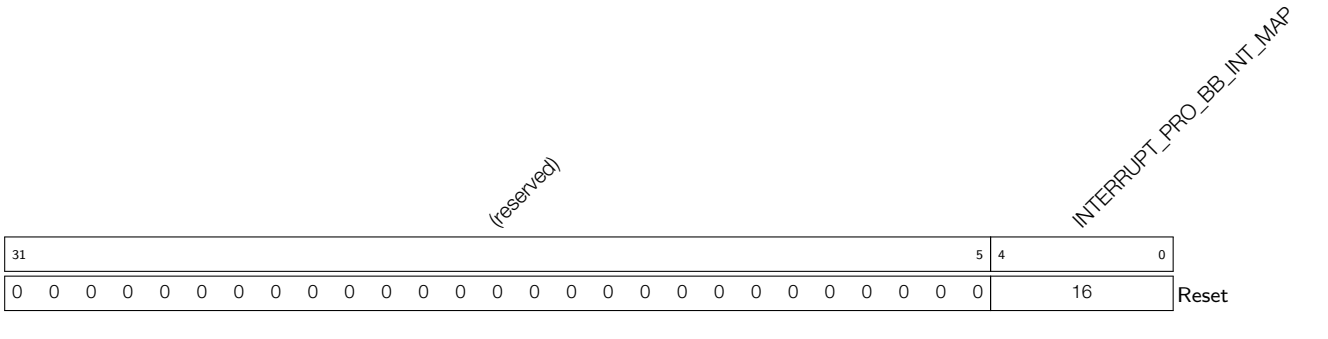
INTERRUPT_PRO_MAC_NMI_MAP This register is used to map MAC_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.3: INTERRUPT_PRO_PWR_INTR_MAP_REG (0x0008)



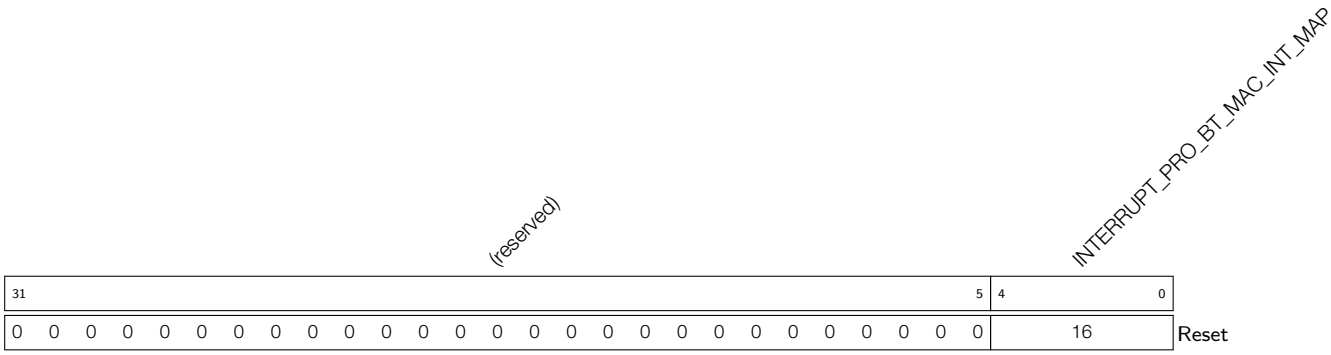
INTERRUPT_PRO_PWR_INTR_MAP This register is used to map PWR_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.4: INTERRUPT_PRO_BB_INT_MAP_REG (0x000C)



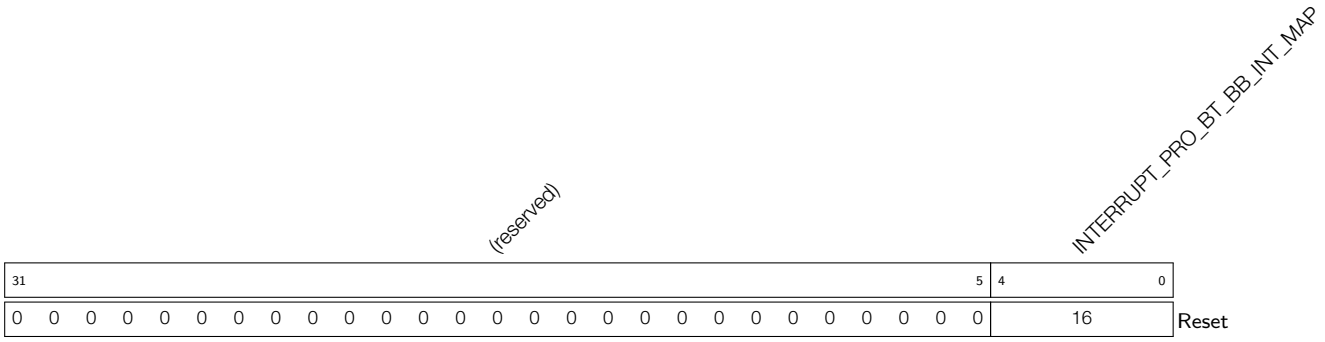
INTERRUPT_PRO_BB_INT_MAP This register is used to map BB_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.5: INTERRUPT_PRO_BT_MAC_INT_MAP_REG (0x0010)



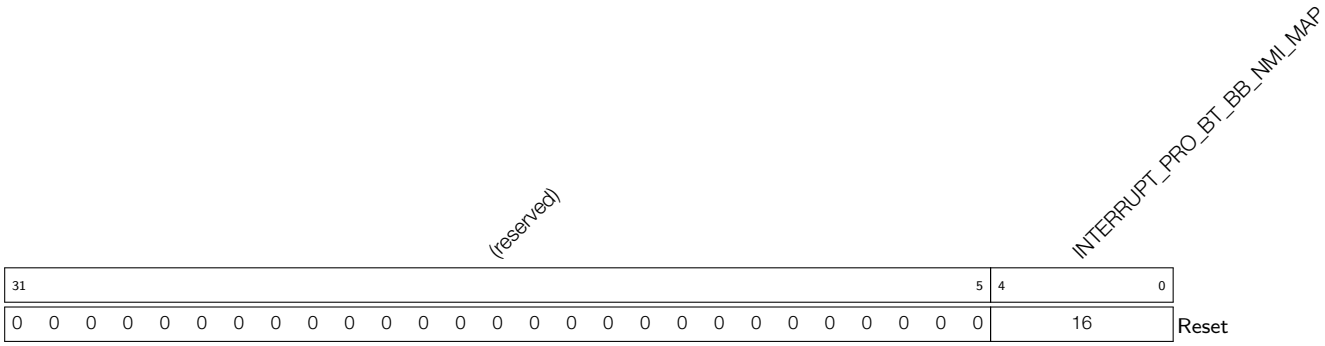
INTERRUPT_PRO_BT_MAC_INT_MAP This register is used to map BT_MAC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.6: INTERRUPT_PRO_BT_BB_INT_MAP_REG (0x0014)



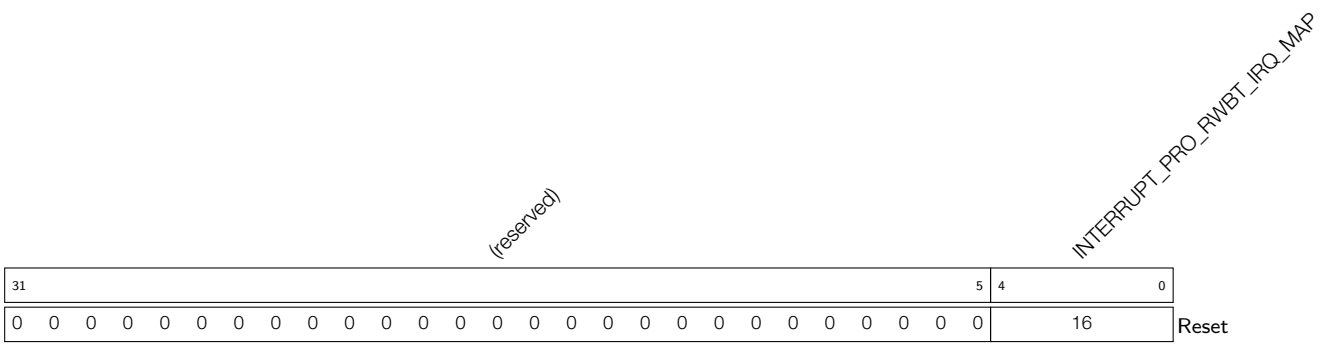
INTERRUPT_PRO_BT_BB_INT_MAP This register is used to map BT_BB_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.7: INTERRUPT_PRO_BT_BB_NMI_MAP_REG (0x0018)



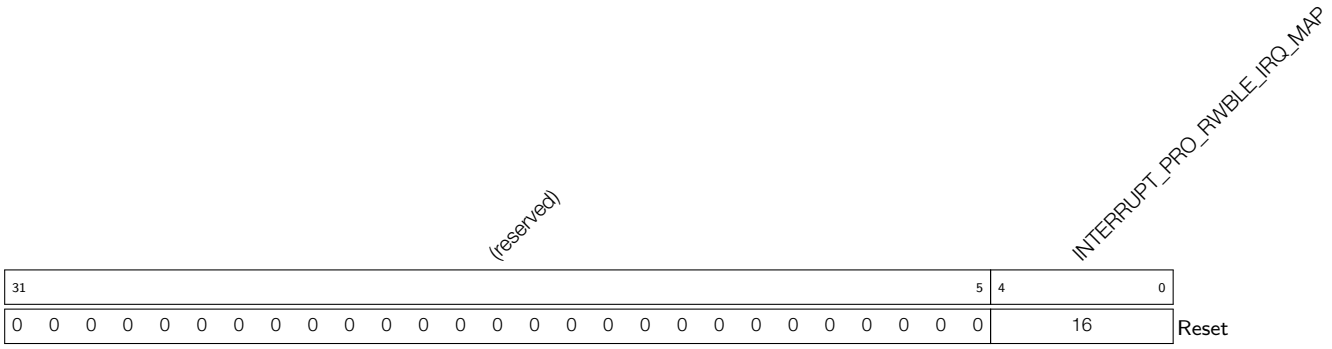
INTERRUPT_PRO_BT_BB_NMI_MAP This register is used to map BT_BB_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.8: INTERRUPT_PRO_RWBT_IRQ_MAP_REG (0x001C)



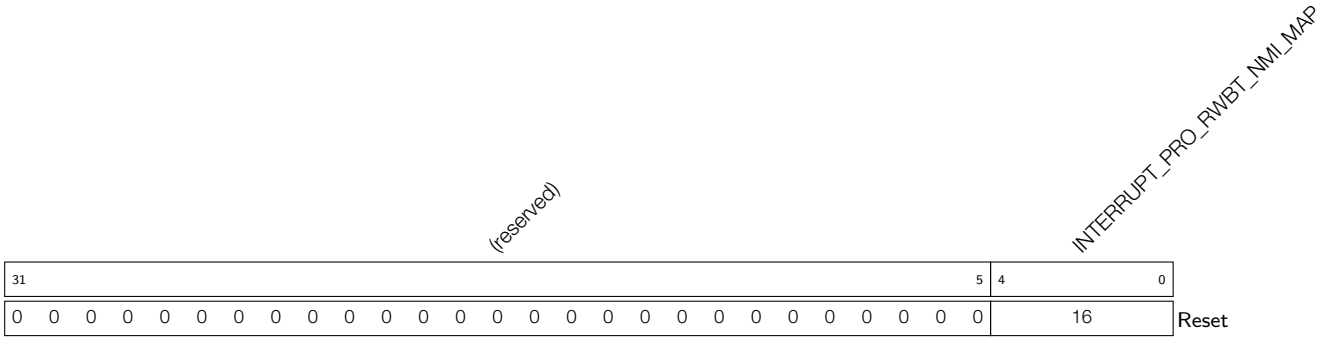
INTERRUPT_PRO_RWBT_IRQ_MAP This register is used to map RWBT_IRQ interrupt signal to one of the CPU interrupts. (R/W)

Register 4.9: INTERRUPT_PRO_RWBLE_IRQ_MAP_REG (0x0020)



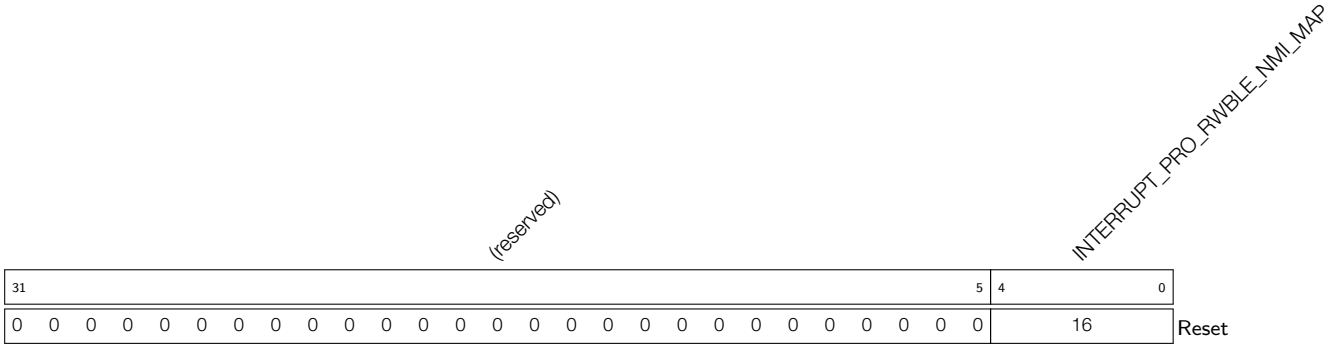
INTERRUPT_PRO_RWBLE_IRQ_MAP This register is used to map RWBLE_IRQ interrupt signal to one of the CPU interrupts. (R/W)

Register 4.10: INTERRUPT_PRO_RWBT_NMI_MAP_REG (0x0024)



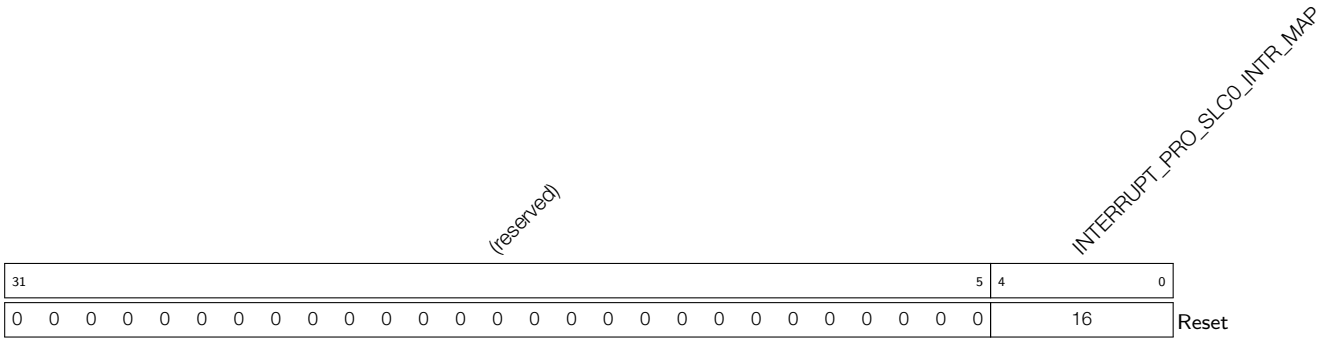
INTERRUPT_PRO_RWBT_NMI_MAP This register is used to map RWBT_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.11: INTERRUPT_PRO_RWBLE_NMI_MAP_REG (0x0028)



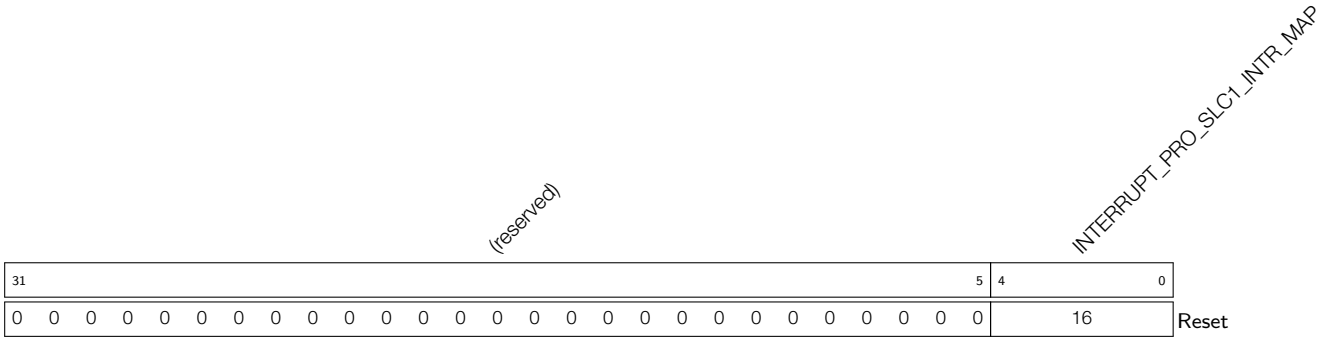
INTERRUPT_PRO_RWBLE_NMI_MAP This register is used to map RWBLE_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.12: INTERRUPT_PRO_SLC0_INTR_MAP_REG (0x002C)



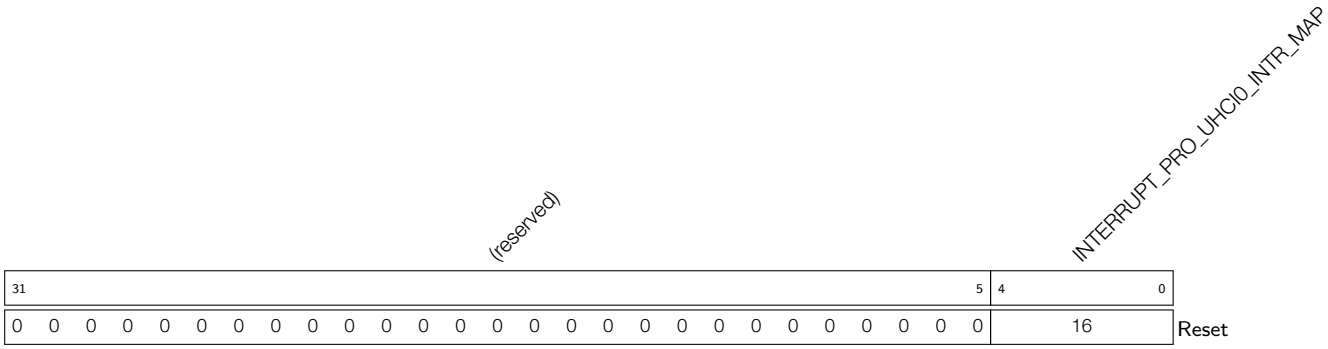
INTERRUPT_PRO_SLC0_INTR_MAP This register is used to map SLC0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.13: INTERRUPT_PRO_SLC1_INTR_MAP_REG (0x0030)



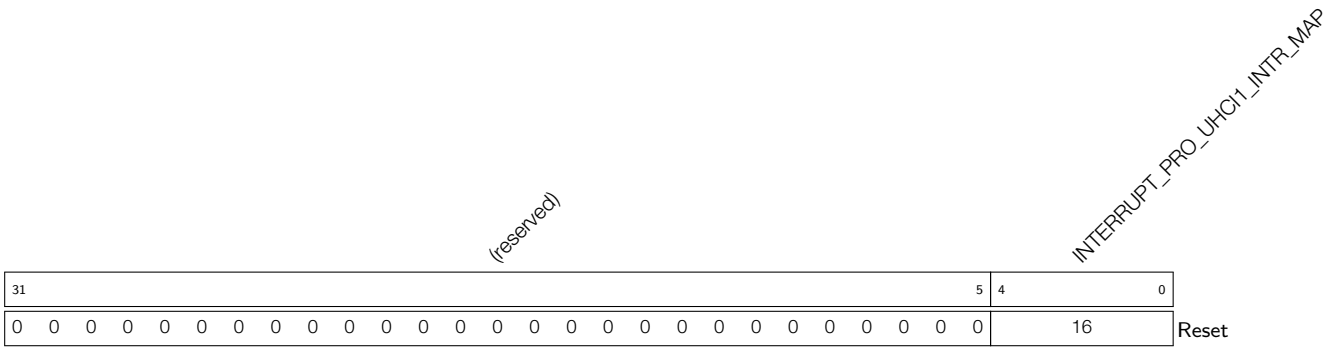
INTERRUPT_PRO_SLC1_INTR_MAP This register is used to map SLC1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.14: INTERRUPT_PRO_UHCI0_INTR_MAP_REG (0x0034)



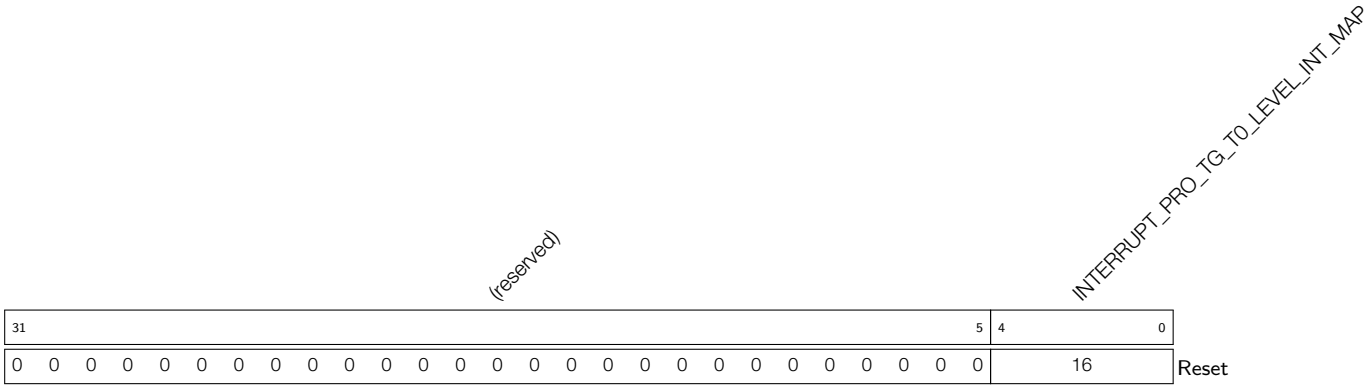
INTERRUPT_PRO_UHCI0_INTR_MAP This register is used to map UHCI0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.15: INTERRUPT_PRO_UHCI1_INTR_MAP_REG (0x0038)



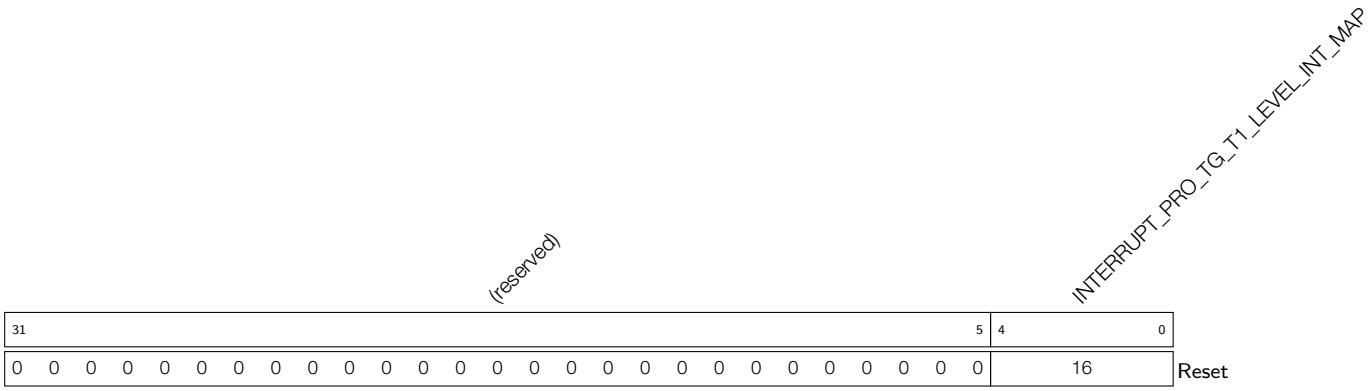
INTERRUPT_PRO_UHCI1_INTR_MAP This register is used to map UHCI1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.16: INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG (0x003C)



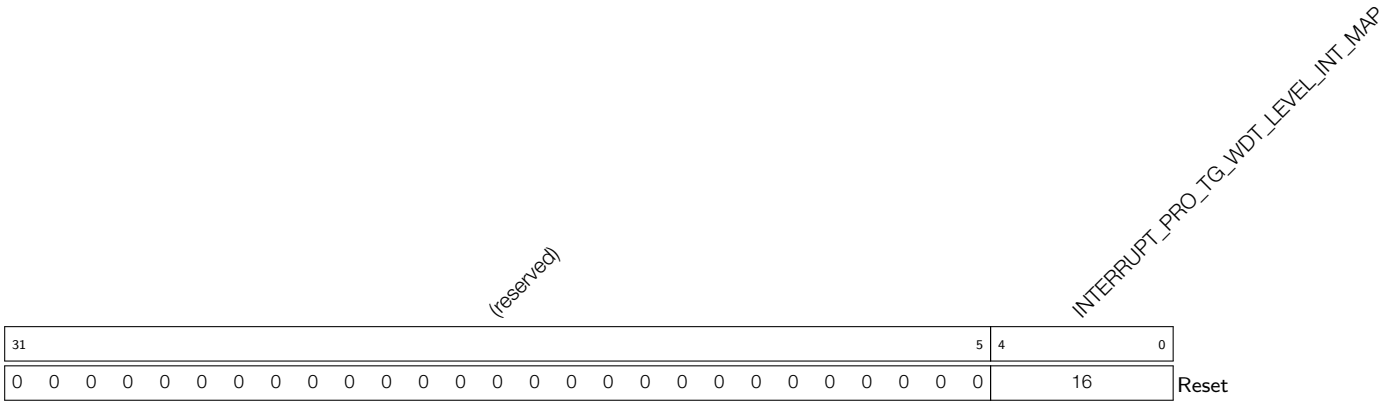
INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP This register is used to map TG_T0_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.17: INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG (0x0040)



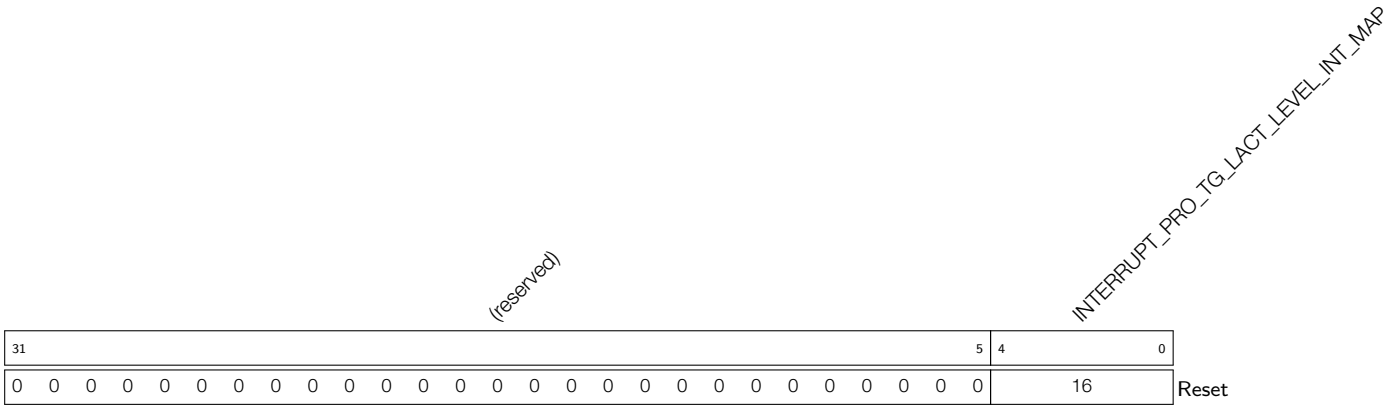
INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP This register is used to map TG_T1_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.18: INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG (0x0044)



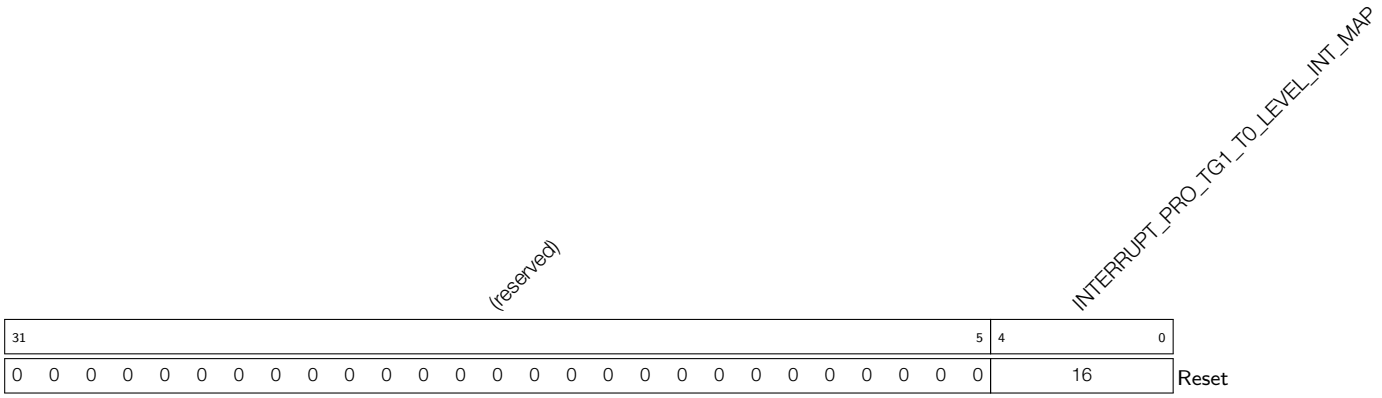
INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP This register is used to map TG_WDT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.19: INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG (0x0048)



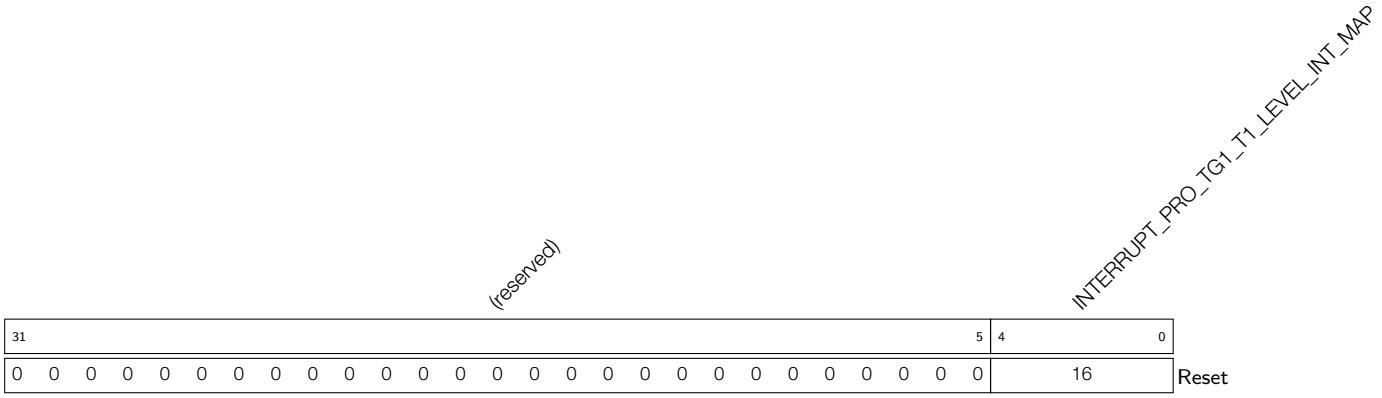
INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP This register is used to map TG_LACT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.20: INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG (0x004C)



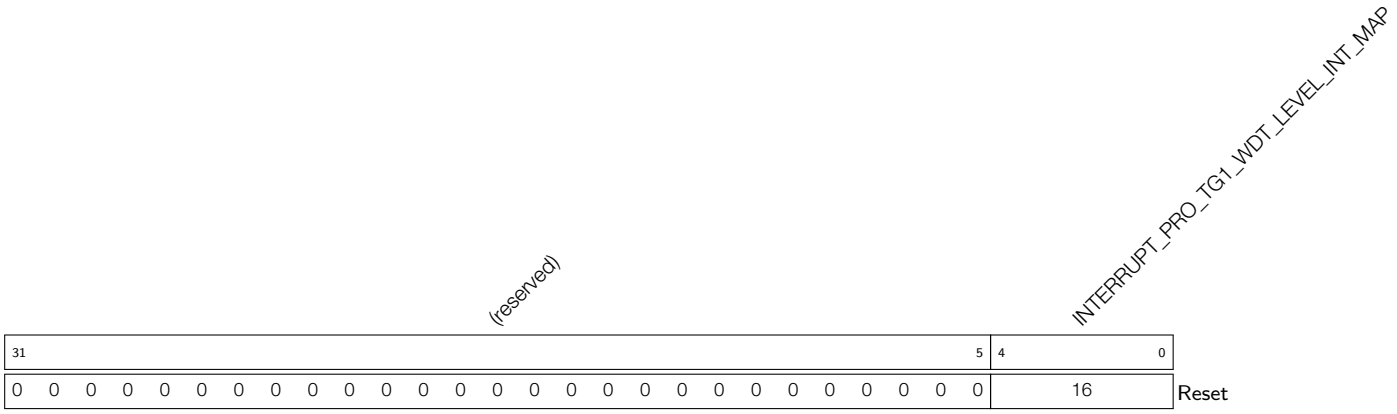
INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP This register is used to map TG1_T0_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.21: INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG (0x0050)



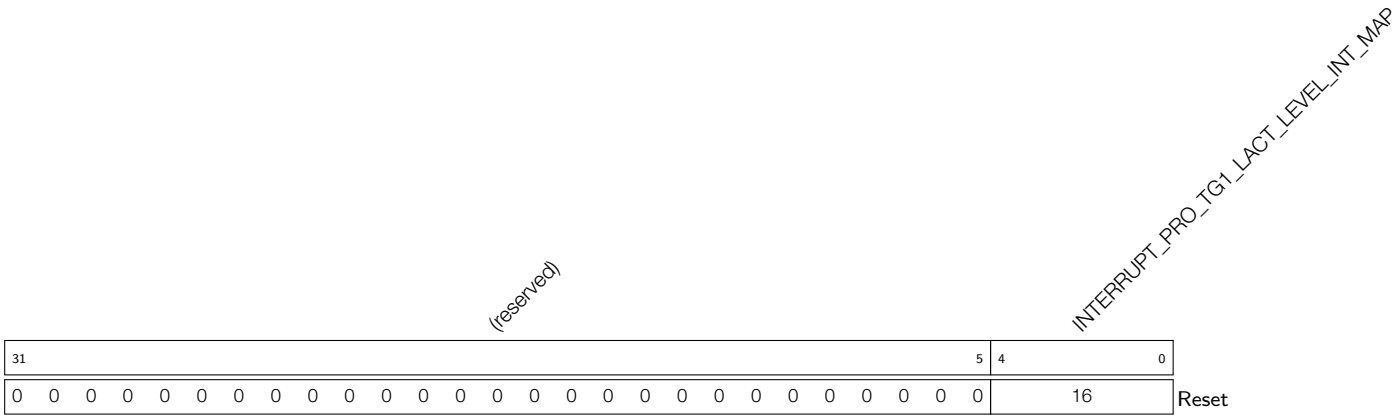
INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP This register is used to map TG1_T1_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.22: INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG (0x0054)



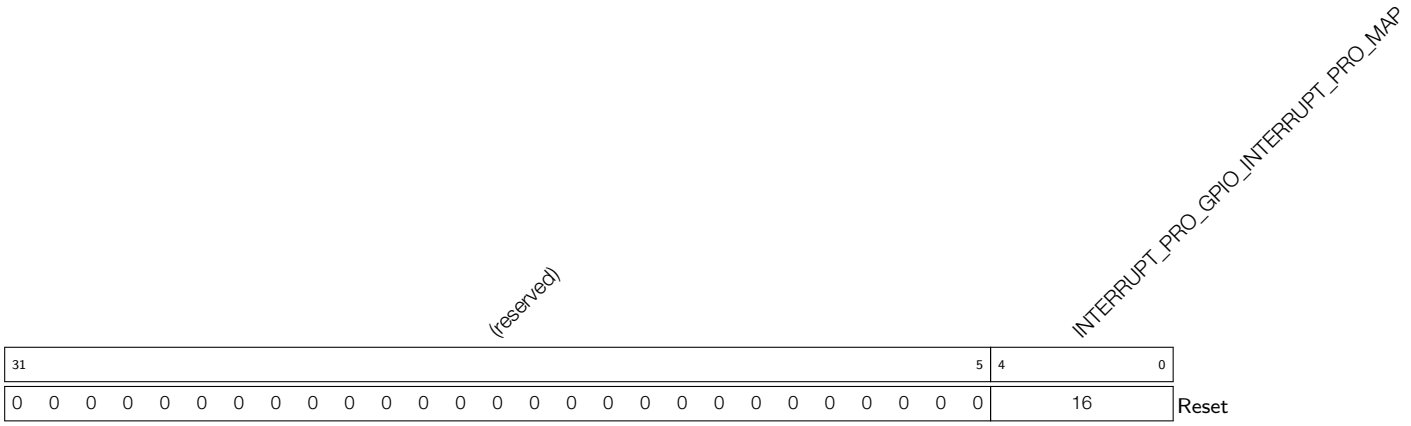
INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP This register is used to map TG1_WDT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.23: INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG (0x0058)



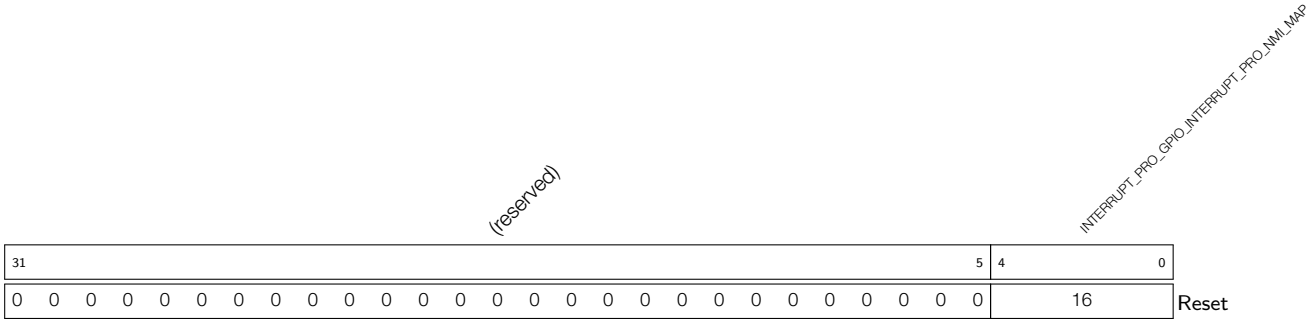
INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP This register is used to map TG1_LACT_LEVEL_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.24: INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG (0x005C)



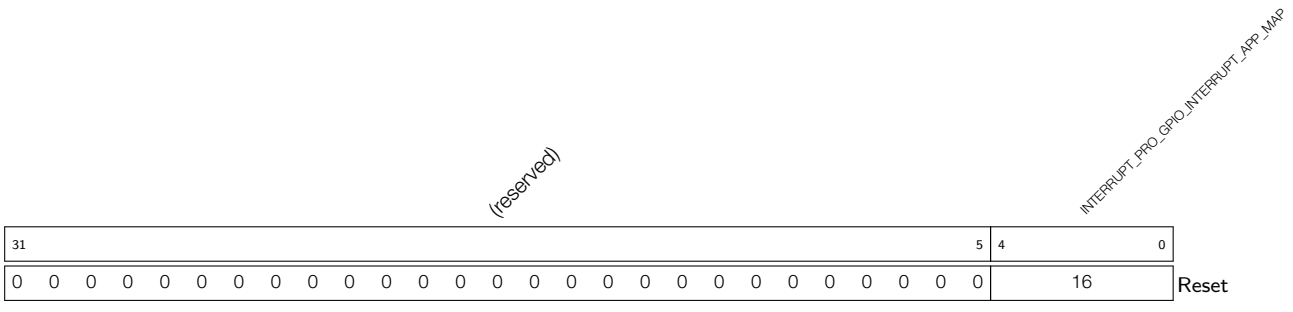
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP This register is used to map GPIO_INTERRUPT_PRO interrupt signal to one of the CPU interrupts. (R/W)

Register 4.25: INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG (0x0060)



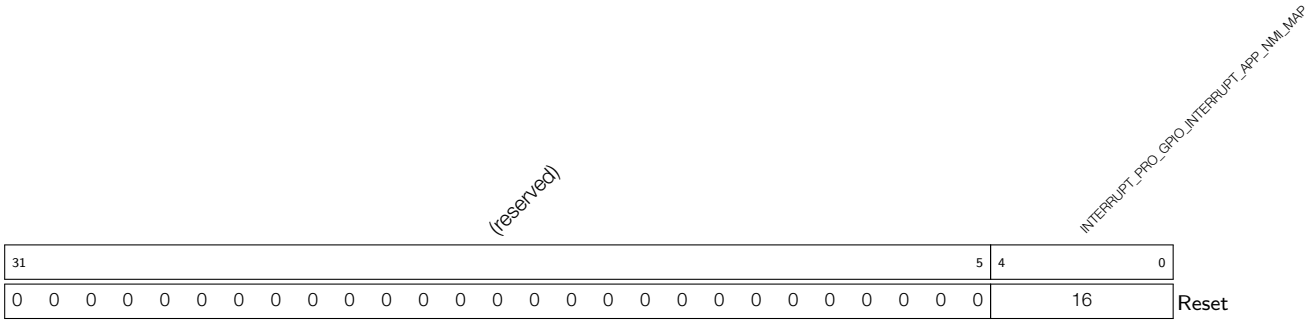
INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP This register is used to map GPIO_INTERRUPT_PRO_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.26: INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP_REG (0x0064)



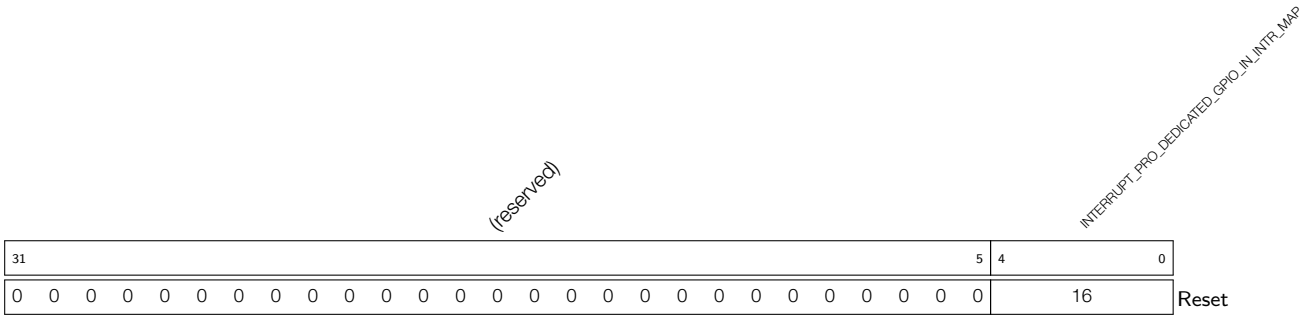
INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP This register is used to map GPIO_INTERRUPT_APP interrupt signal to one of the CPU interrupts. (R/W)

Register 4.27: INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP_REG (0x0068)



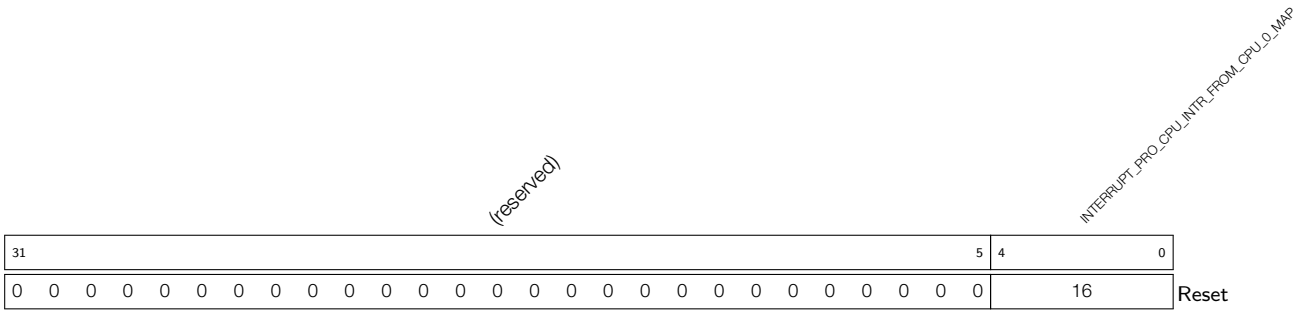
INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP This register is used to map GPIO_INTERRUPT_APP_NMI interrupt signal to one of the CPU interrupts. (R/W)

Register 4.28: INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG (0x006C)



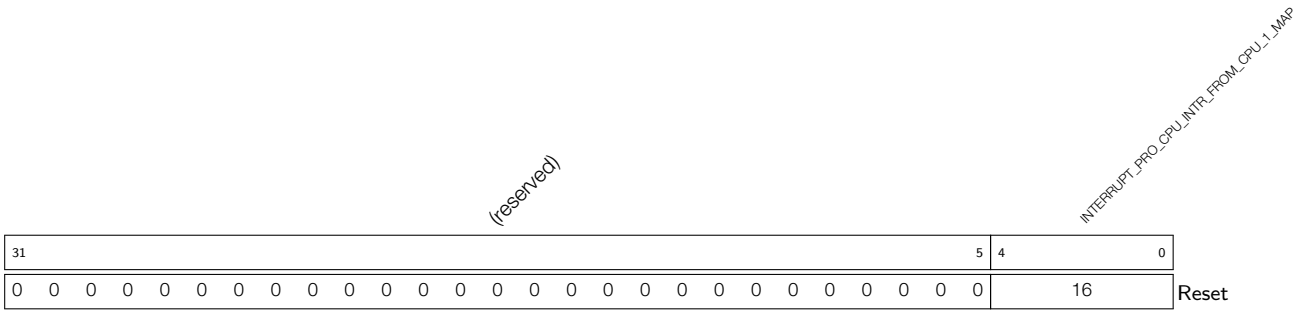
INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP This register is used to map DEDICATED_GPIO_IN_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.29: INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG (0x0070)



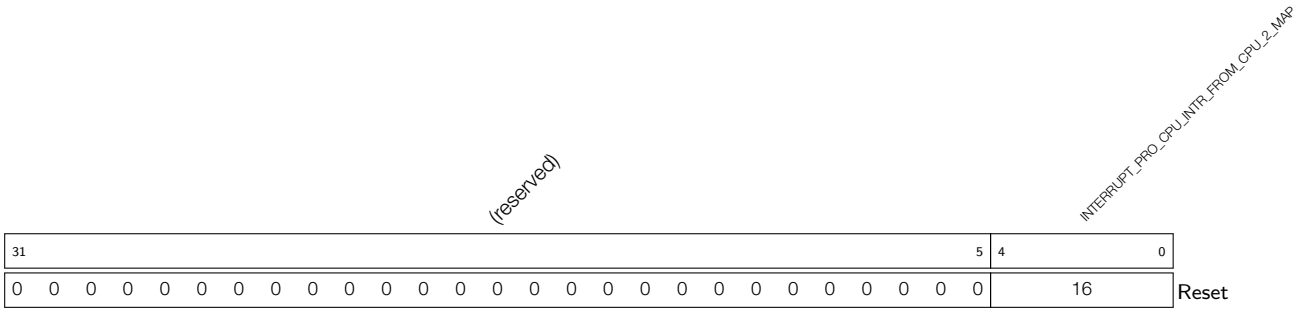
INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP This register is used to map CPU_INTR_FROM_CPU_0 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.30: INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG (0x0074)



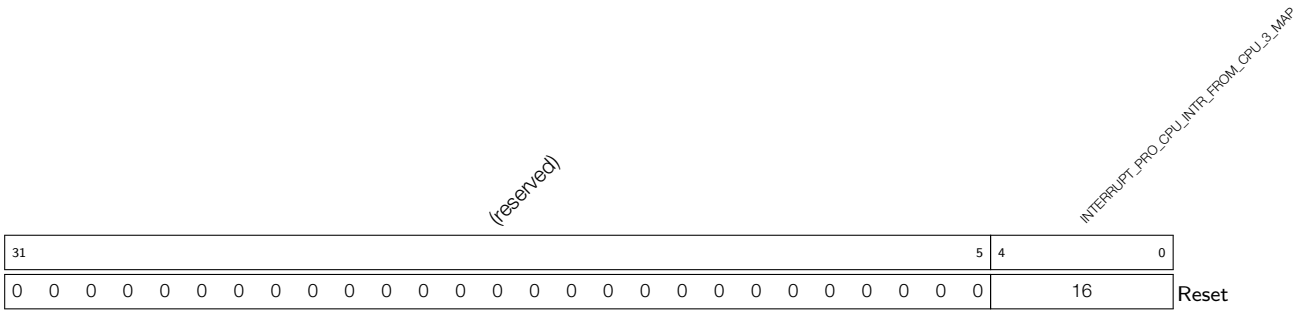
INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP This register is used to map CPU_INTR_FROM_CPU_1 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.31: INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG (0x0078)



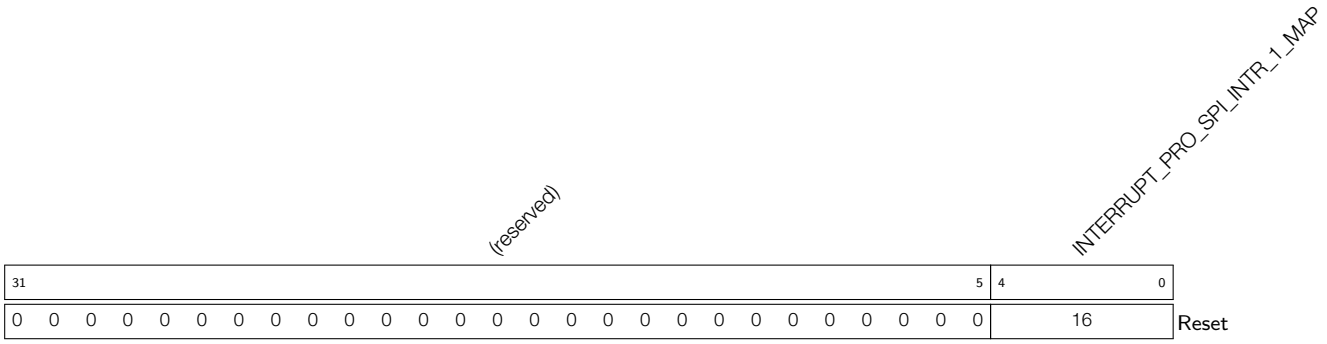
INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP This register is used to map CPU_INTR_FROM_CPU_2 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.32: INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG (0x007C)



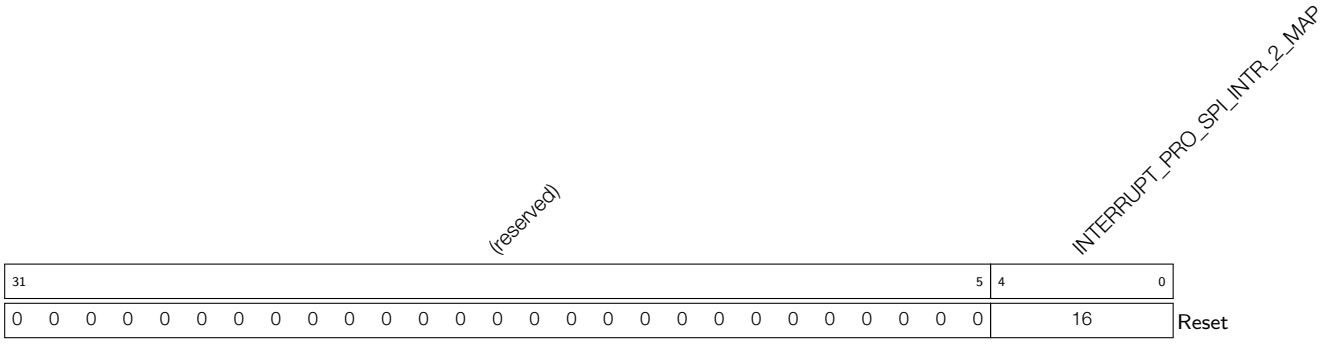
INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP This register is used to map CPU_INTR_FROM_CPU_3 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.33: INTERRUPT_PRO_SPI_INTR_1_MAP_REG (0x0080)



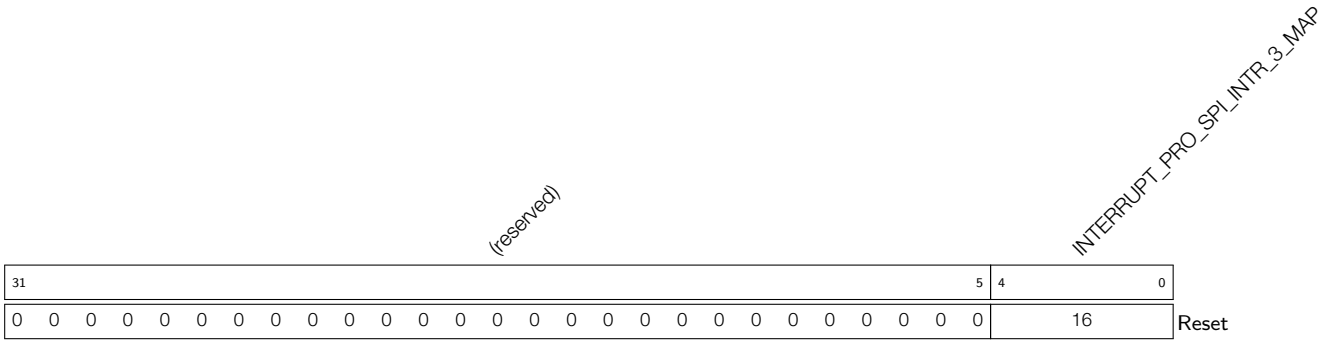
INTERRUPT_PRO_SPI_INTR_1_MAP This register is used to map SPI_INTR_1 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.34: INTERRUPT_PRO_SPI_INTR_2_MAP_REG (0x0084)



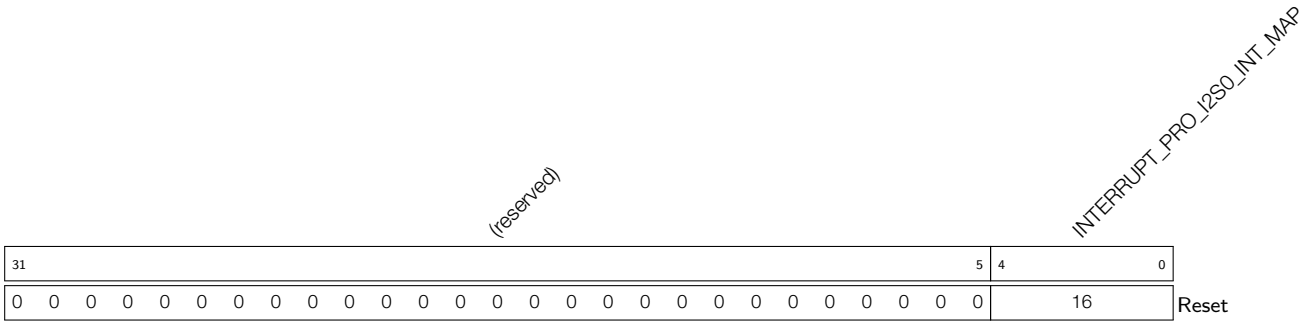
INTERRUPT_PRO_SPI_INTR_2_MAP This register is used to map SPI_INTR_2 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.35: INTERRUPT_PRO_SPI_INTR_3_MAP_REG (0x0088)



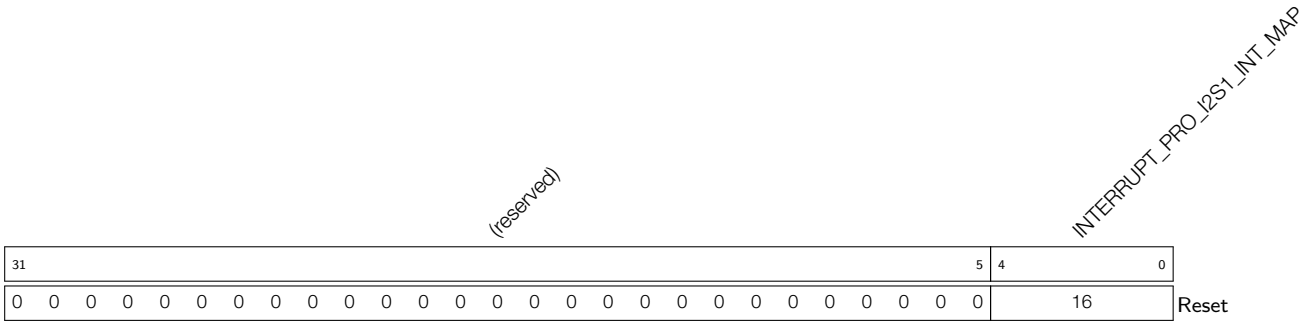
INTERRUPT_PRO_SPI_INTR_3_MAP This register is used to map SPI_INTR_3 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.36: INTERRUPT_PRO_I2S0_INT_MAP_REG (0x008C)



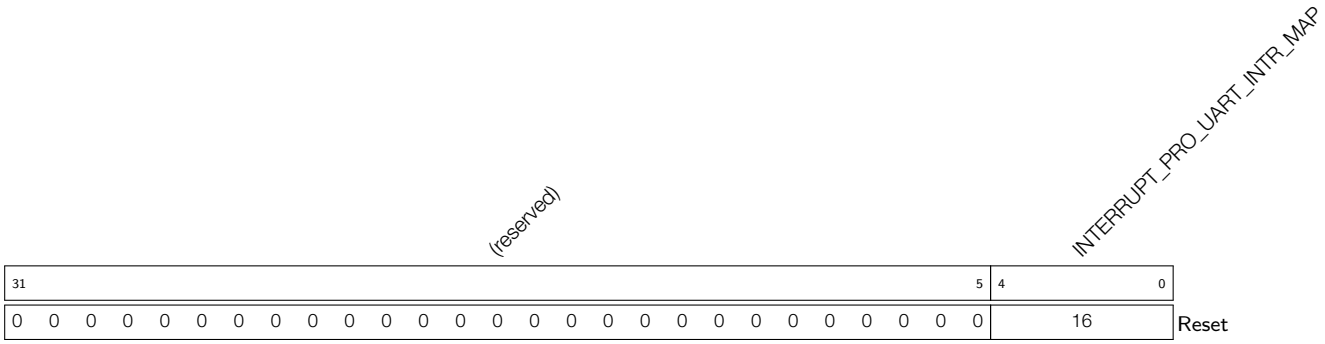
INTERRUPT_PRO_I2S0_INT_MAP This register is used to map I2S0_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.37: INTERRUPT_PRO_I2S1_INT_MAP_REG (0x0090)



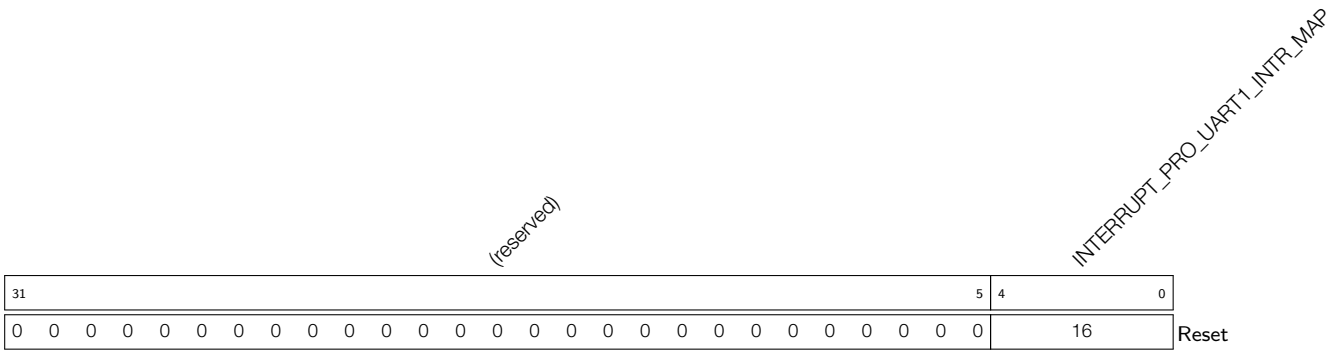
INTERRUPT_PRO_I2S1_INT_MAP This register is used to map I2S1_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.38: INTERRUPT_PRO_UART_INTR_MAP_REG (0x0094)



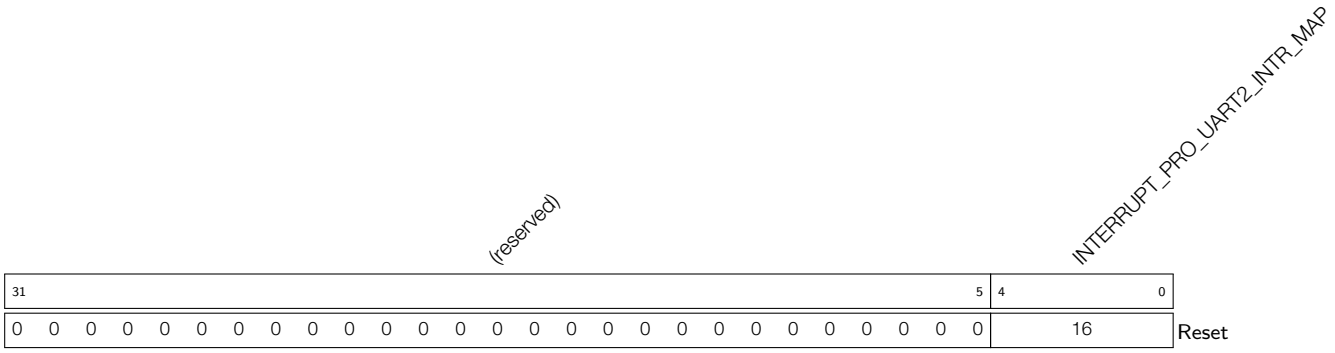
INTERRUPT_PRO_UART_INTR_MAP This register is used to map UART_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.39: INTERRUPT_PRO_UART1_INTR_MAP_REG (0x0098)



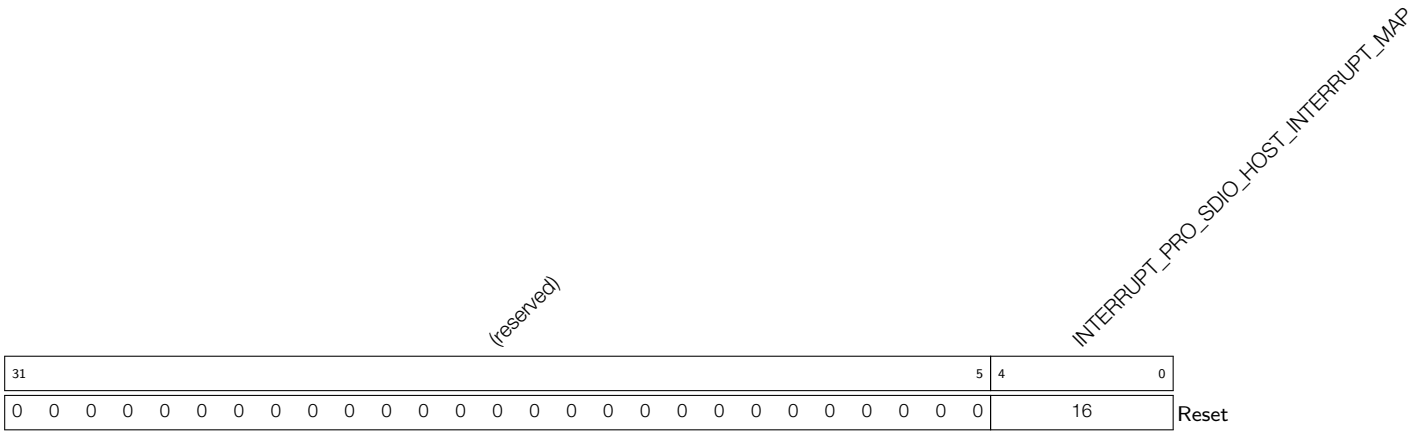
INTERRUPT_PRO_UART1_INTR_MAP This register is used to map UART1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.40: INTERRUPT_PRO_UART2_INTR_MAP_REG (0x009C)



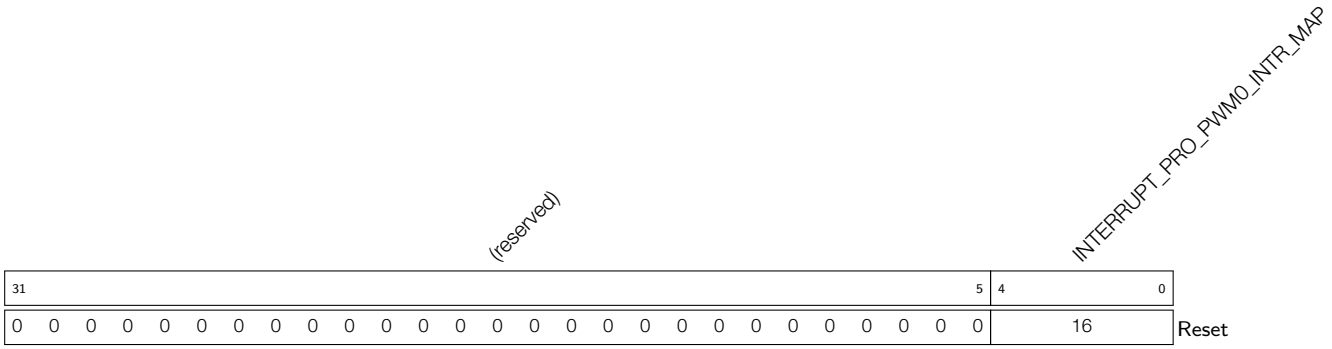
INTERRUPT_PRO_UART2_INTR_MAP This register is used to map UART2_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.41: INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP_REG (0x00A0)



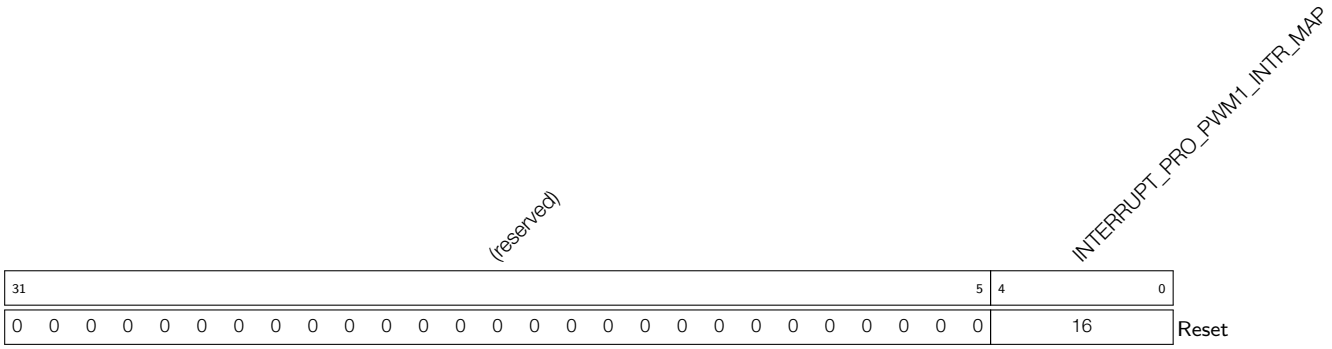
INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP This register is used to map SDIO_HOST_INTERRUPT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.42: INTERRUPT_PRO_PWM0_INTR_MAP_REG (0x00A4)



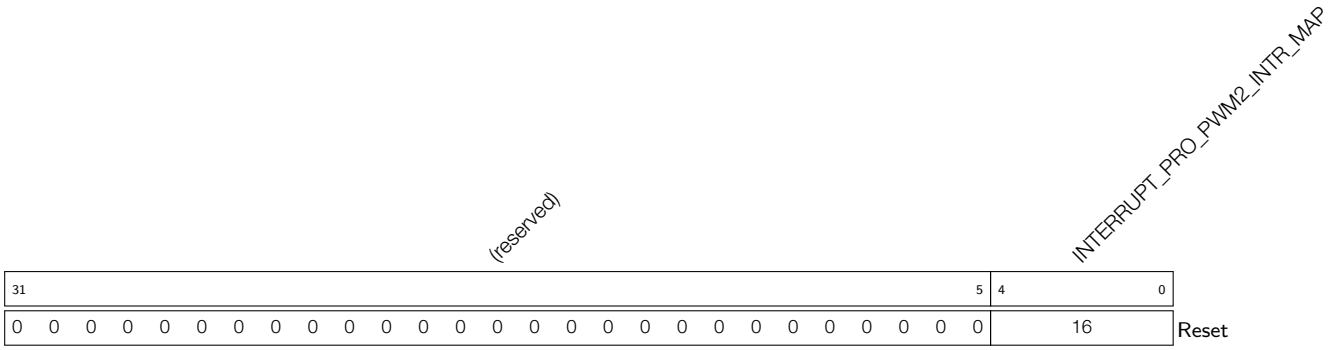
INTERRUPT_PRO_PWM0_INTR_MAP This register is used to map PWM0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.43: INTERRUPT_PRO_PWM1_INTR_MAP_REG (0x00A8)



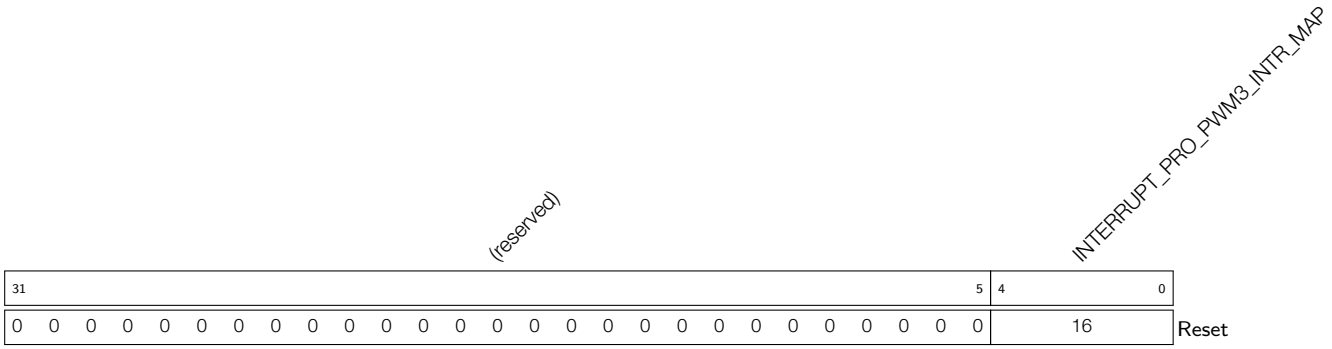
INTERRUPT_PRO_PWM1_INTR_MAP This register is used to map PWM1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.44: INTERRUPT_PRO_PWM2_INTR_MAP_REG (0x00AC)



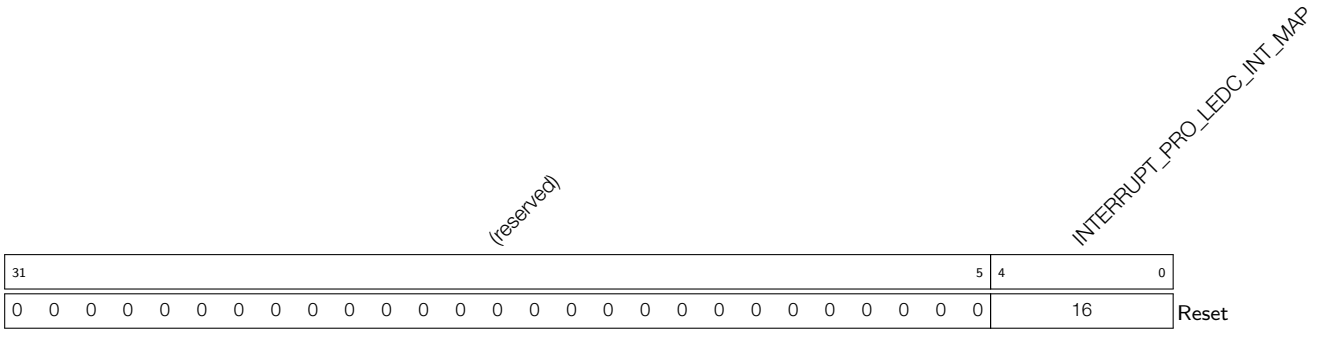
INTERRUPT_PRO_PWM2_INTR_MAP This register is used to map PWM2_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.45: INTERRUPT_PRO_PWM3_INTR_MAP_REG (0x00B0)



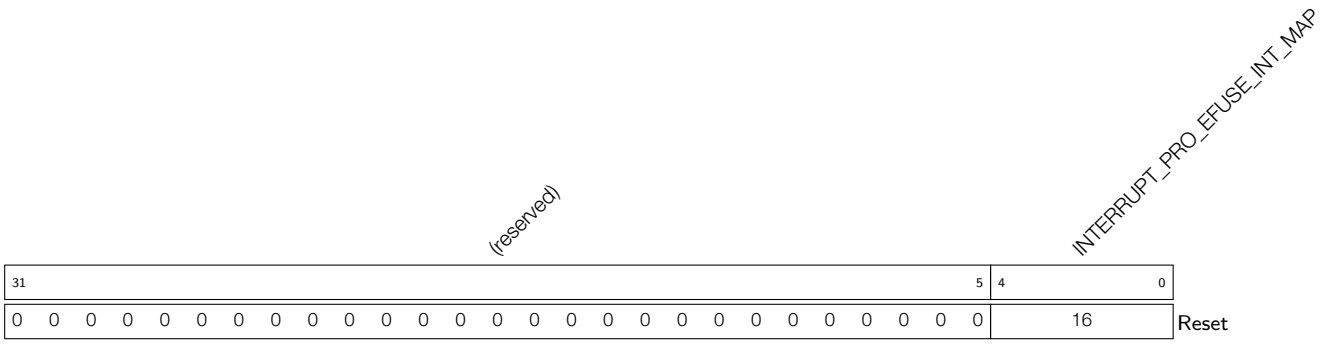
INTERRUPT_PRO_PWM3_INTR_MAP This register is used to map PWM3_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.46: INTERRUPT_PRO_LEDC_INT_MAP_REG (0x00B4)



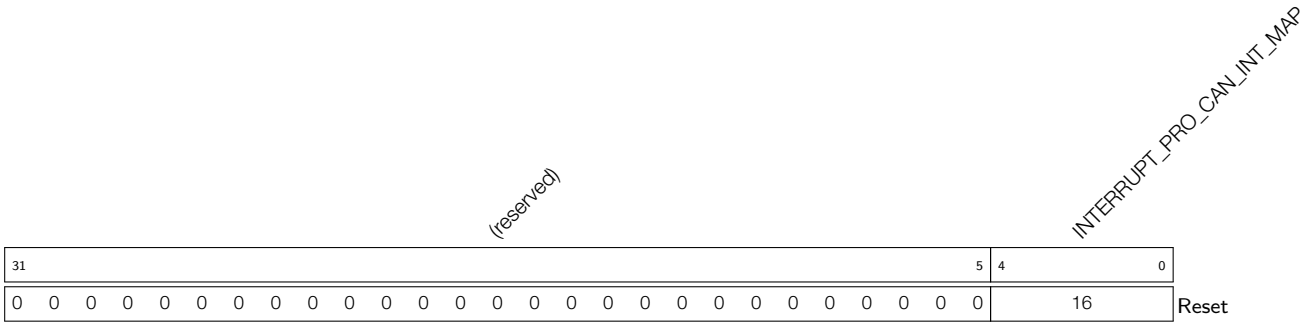
INTERRUPT_PRO_LEDC_INT_MAP This register is used to map LEDC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.47: INTERRUPT_PRO_EFUSE_INT_MAP_REG (0x00B8)



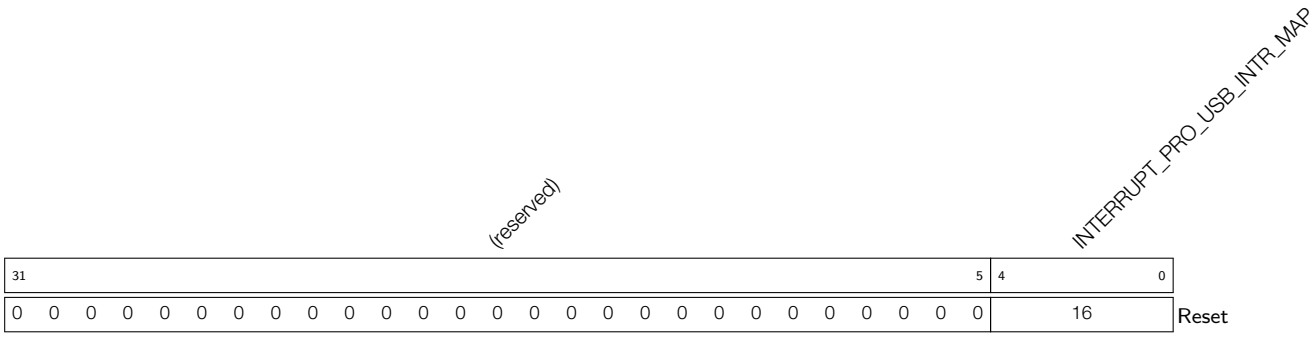
INTERRUPT_PRO_EFUSE_INT_MAP This register is used to map EFUSE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.48: INTERRUPT_PRO_CAN_INT_MAP_REG (0x00BC)



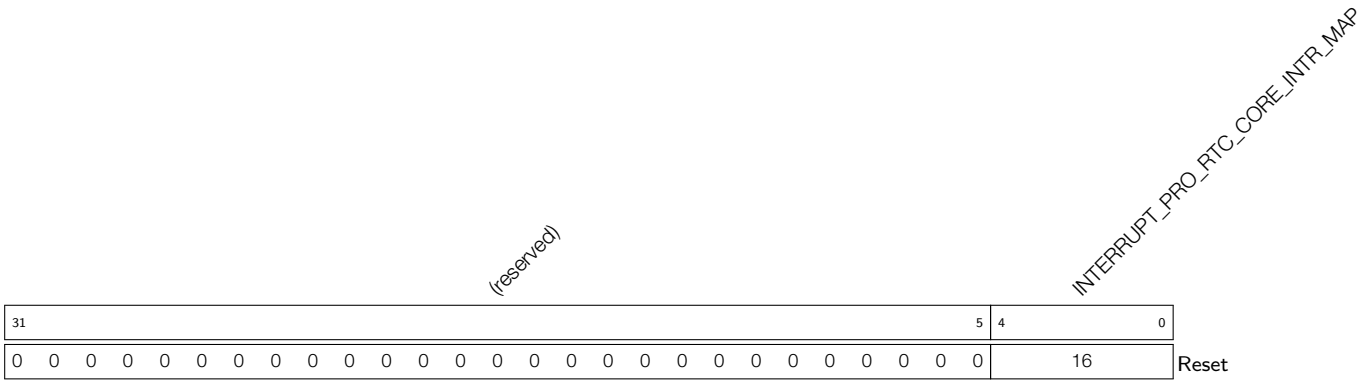
INTERRUPT_PRO_CAN_INT_MAP This register is used to map CAN_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.49: INTERRUPT_PRO_USB_INTR_MAP_REG (0x00C0)



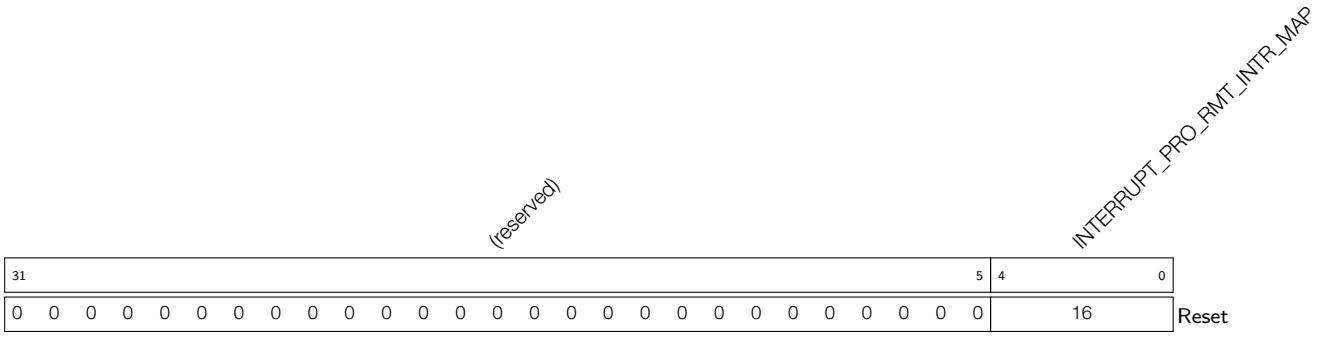
INTERRUPT_PRO_USB_INTR_MAP This register is used to map USB_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.50: INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG (0x00C4)



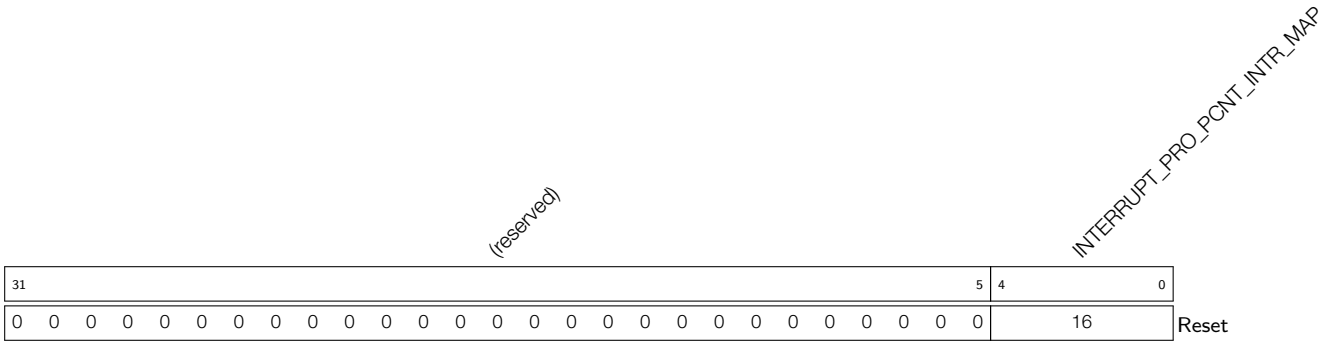
INTERRUPT_PRO_RTC_CORE_INTR_MAP This register is used to map RTC_CORE_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.51: INTERRUPT_PRO_RMT_INTR_MAP_REG (0x00C8)



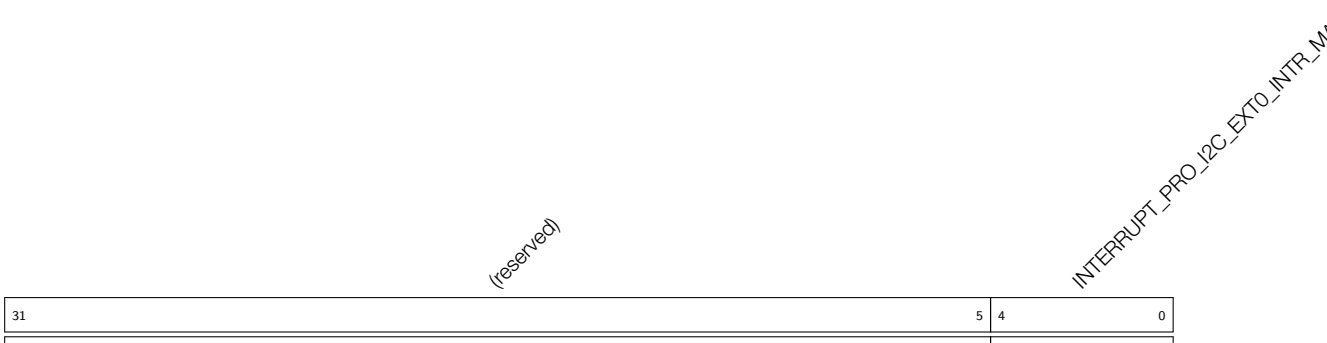
INTERRUPT_PRO_RMT_INTR_MAP This register is used to map RMT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.52: INTERRUPT_PRO_PCNT_INTR_MAP_REG (0x00CC)



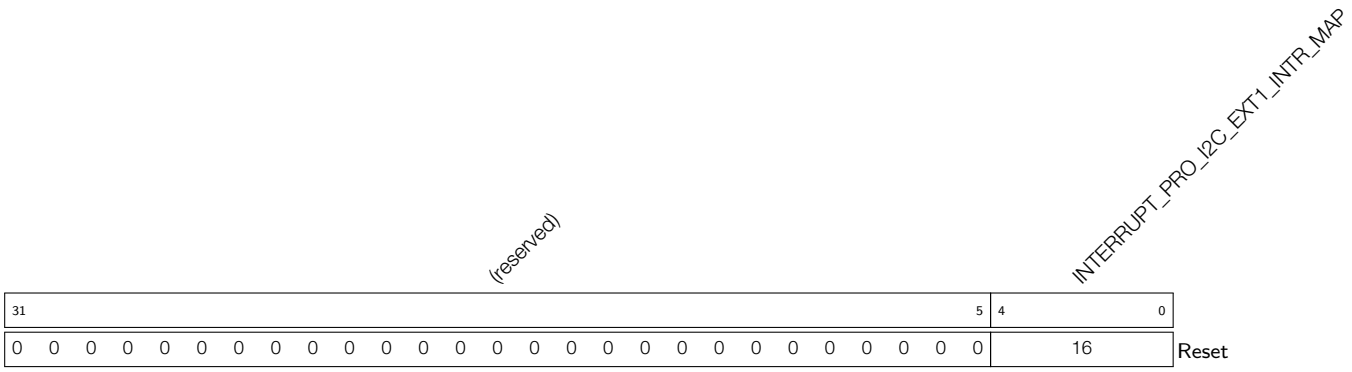
INTERRUPT_PRO_PCNT_INTR_MAP This register is used to map PCNT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.53: INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG (0x00D0)



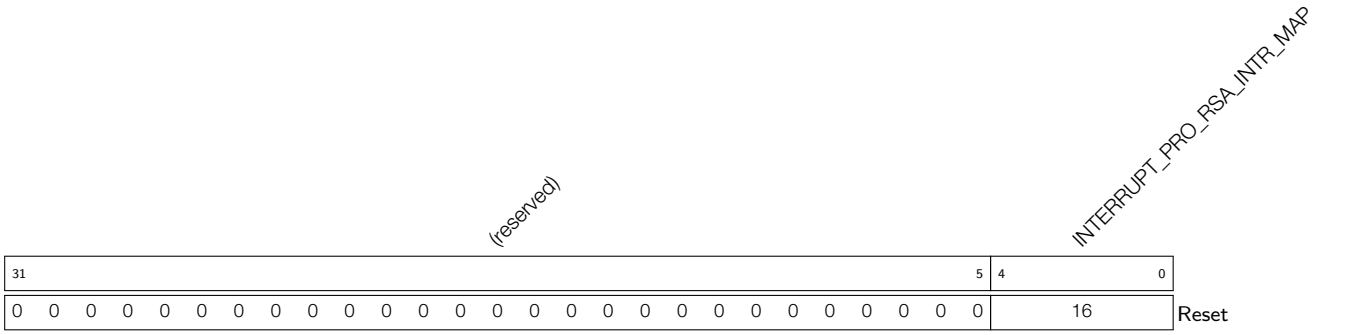
INTERRUPT_PRO_I2C_EXT0_INTR_MAP This register is used to map I2C_EXT0_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.54: INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG (0x00D4)



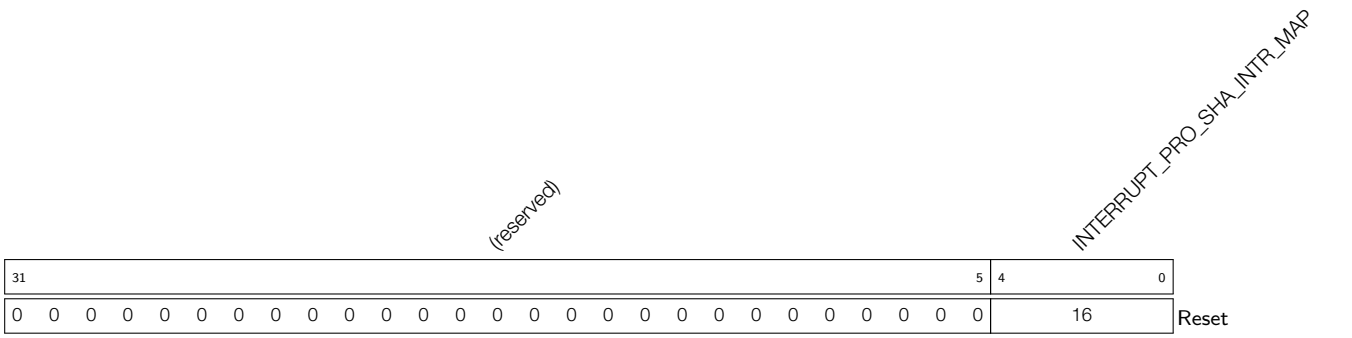
INTERRUPT_PRO_I2C_EXT1_INTR_MAP This register is used to map I2C_EXT1_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.55: INTERRUPT_PRO_RSA_INTR_MAP_REG (0x00D8)



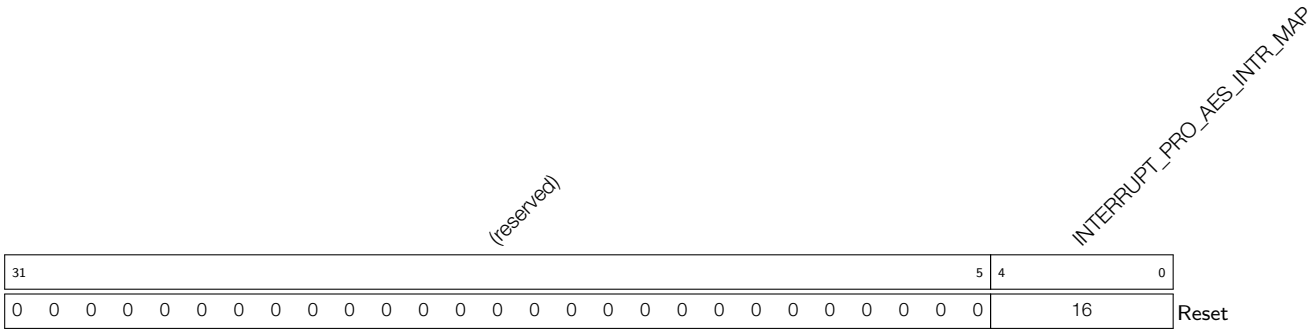
INTERRUPT_PRO_RSA_INTR_MAP This register is used to map RSA_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.56: INTERRUPT_PRO_SHA_INTR_MAP_REG (0x00DC)



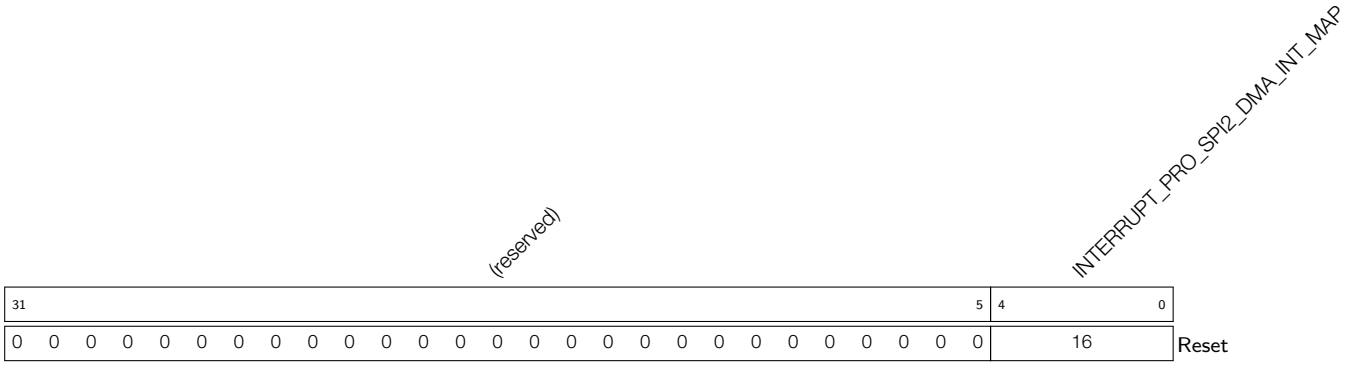
INTERRUPT_PRO_SHA_INTR_MAP This register is used to map SHA_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.57: INTERRUPT_PRO_AES_INTR_MAP_REG (0x00E0)



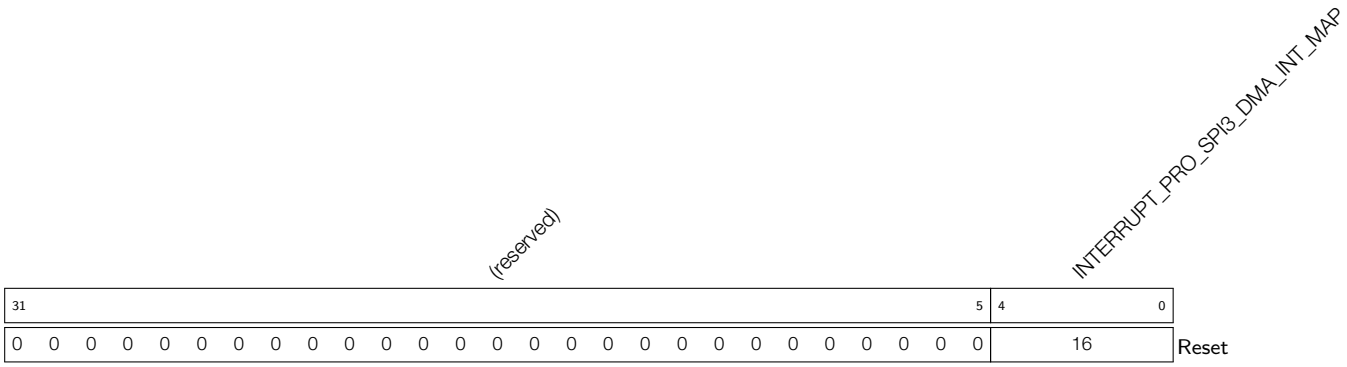
INTERRUPT_PRO_AES_INTR_MAP This register is used to map AES_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.58: INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG (0x00E4)



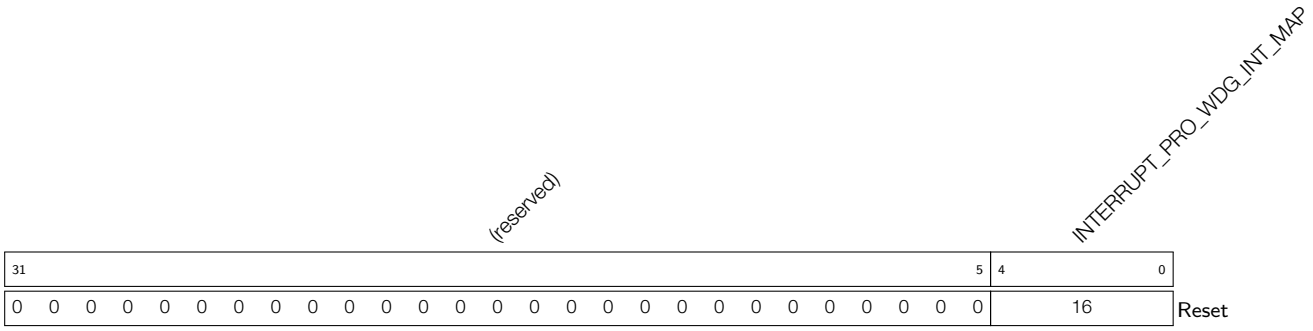
INTERRUPT_PRO_SPI2_DMA_INT_MAP This register is used to map SPI2_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.59: INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG (0x00E8)



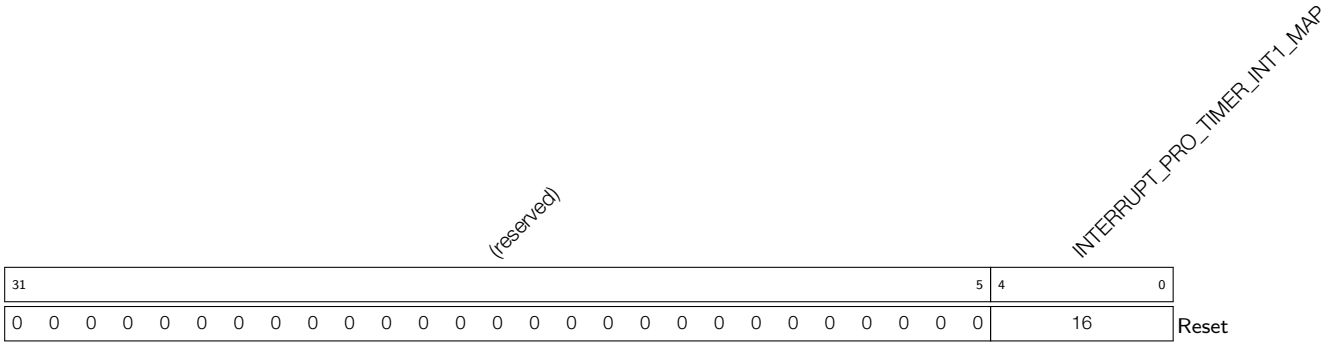
INTERRUPT_PRO_SPI3_DMA_INT_MAP This register is used to map SPI3_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.60: INTERRUPT_PRO_WDG_INT_MAP_REG (0x00EC)



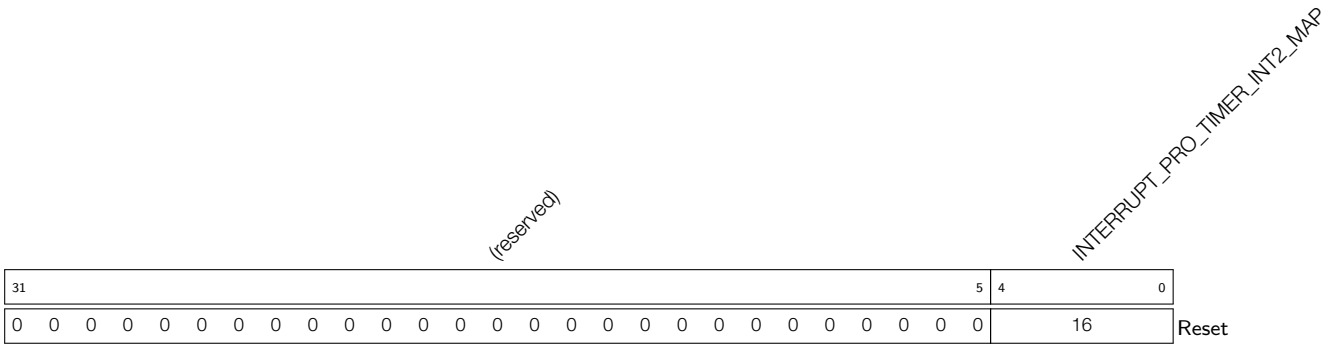
INTERRUPT_PRO_WDG_INT_MAP This register is used to map WDG_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.61: INTERRUPT_PRO_TIMER_INT1_MAP_REG (0x00F0)



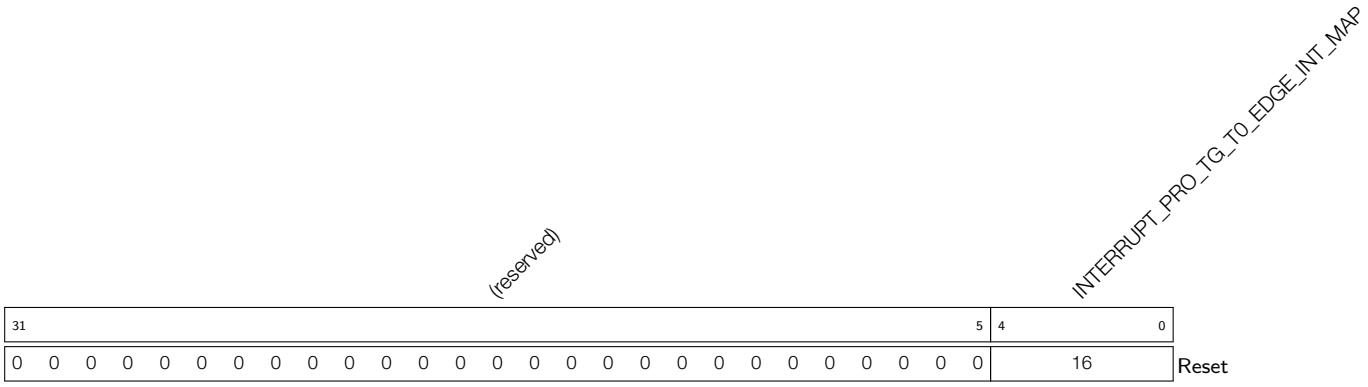
INTERRUPT_PRO_TIMER_INT1_MAP This register is used to map TIMER_INT1 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.62: INTERRUPT_PRO_TIMER_INT2_MAP_REG (0x00F4)



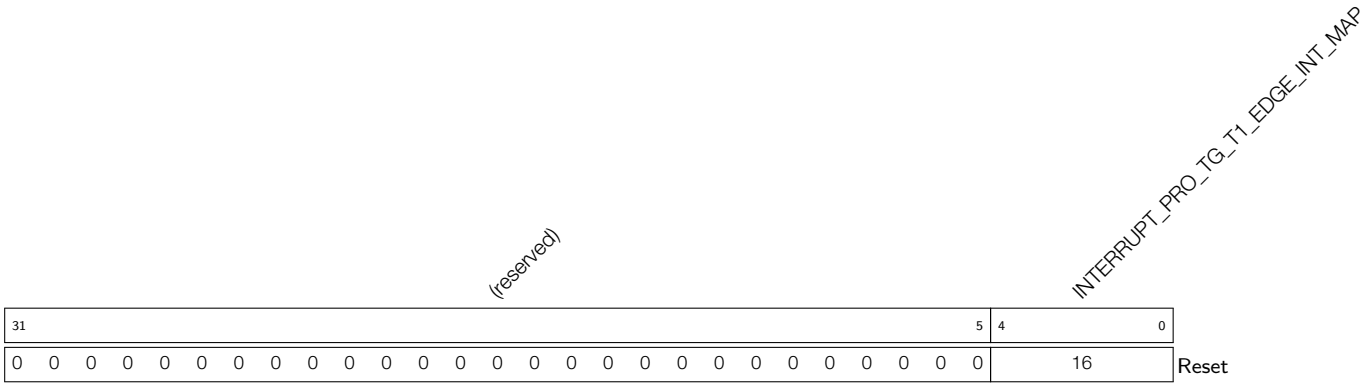
INTERRUPT_PRO_TIMER_INT2_MAP This register is used to map TIMER_INT2 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.63: INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG (0x00F8)



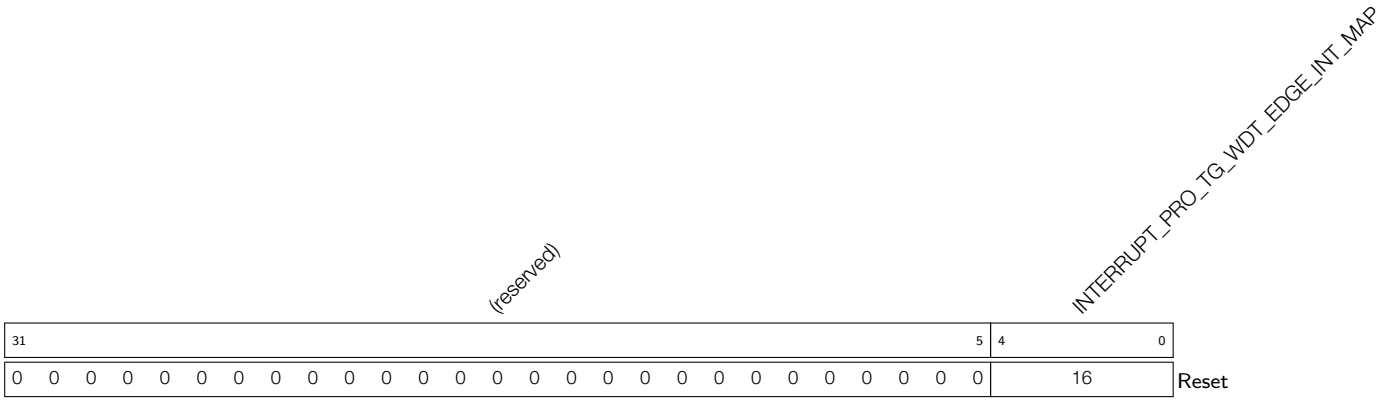
INTERRUPT_PRO_TG_T0_EDGE_INT_MAP This register is used to map TG_T0_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.64: INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG (0x00FC)



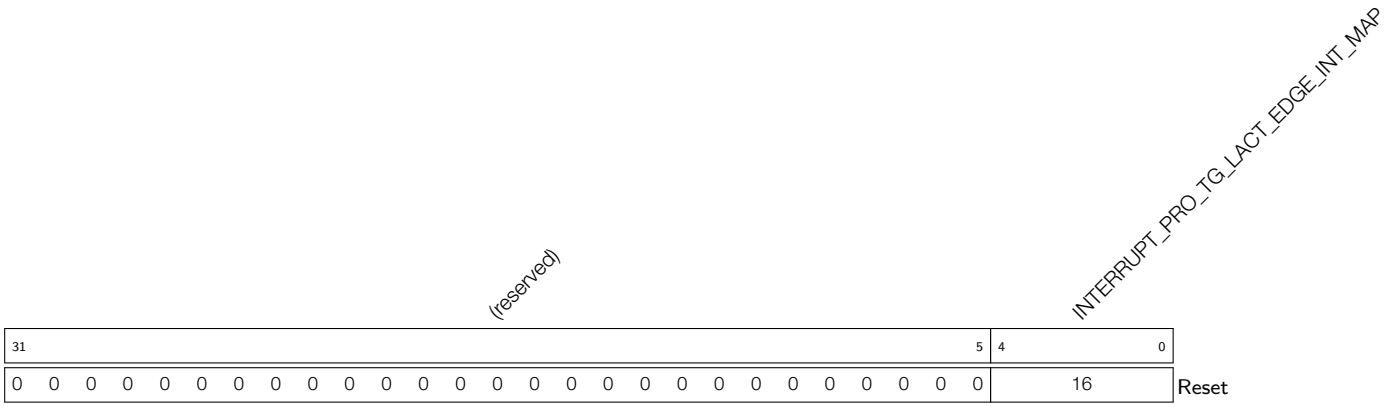
INTERRUPT_PRO_TG_T1_EDGE_INT_MAP This register is used to map TG_T1_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.65: INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG (0x0100)



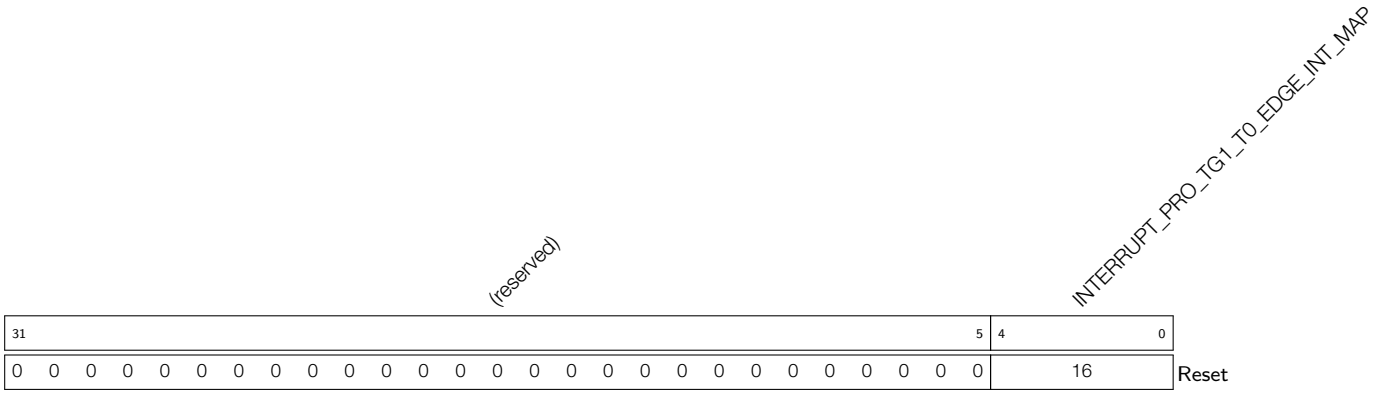
INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP This register is used to map TG_WDT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.66: INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG (0x0104)



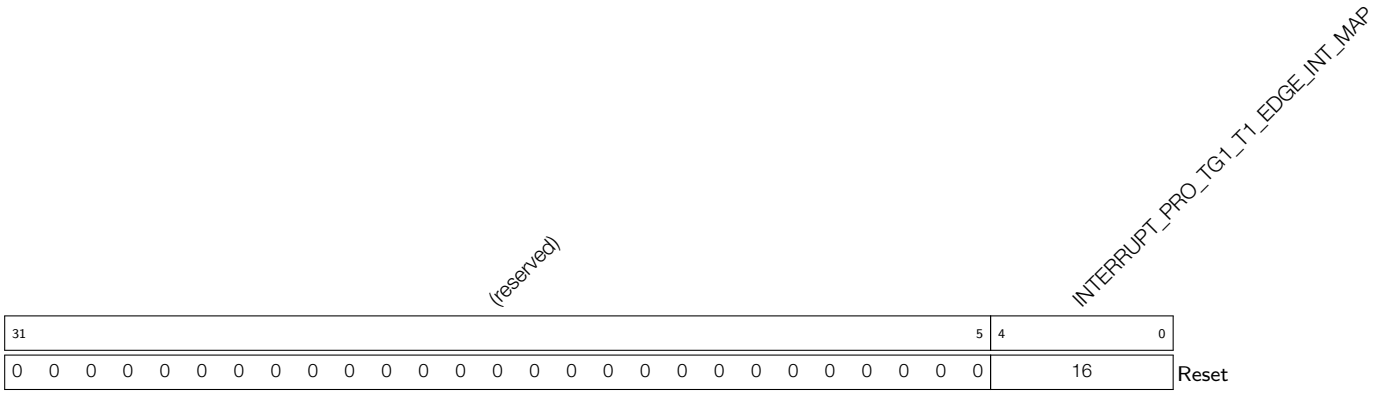
INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP This register is used to map TG_LACT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.67: INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG (0x0108)



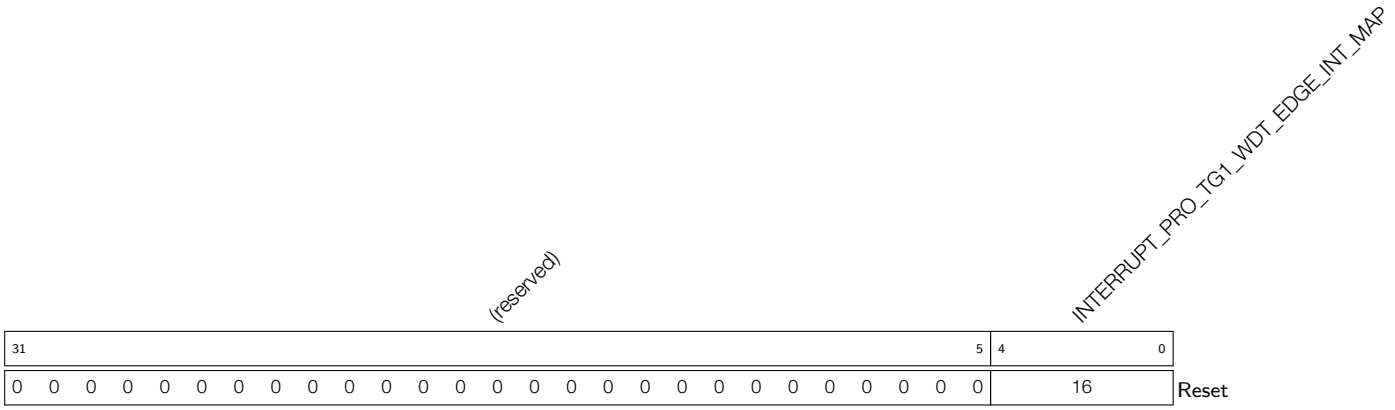
INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP This register is used to map TG1_T0_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.68: INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG (0x010C)



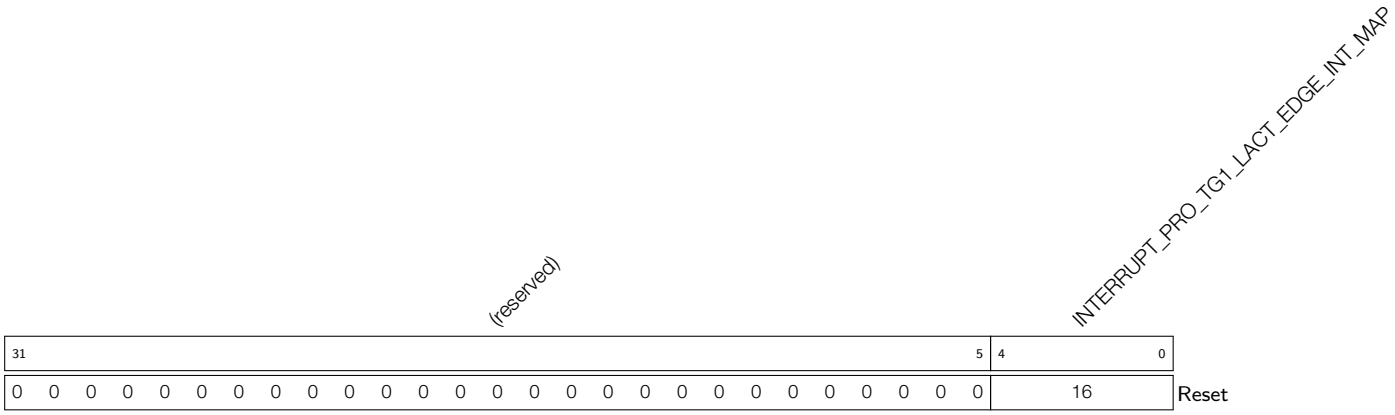
INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP This register is used to map TG1_T1_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.69: INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG (0x0110)



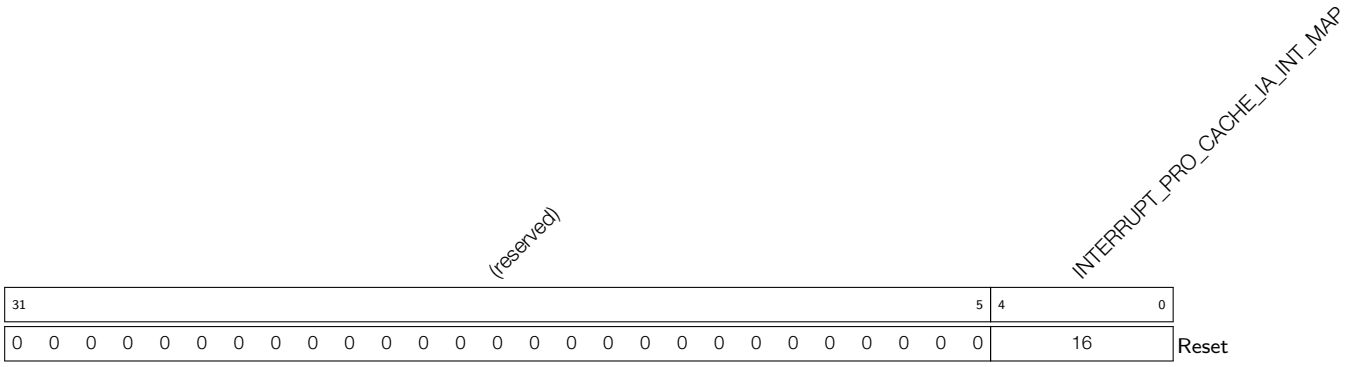
INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP This register is used to map TG1_WDT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.70: INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG (0x0114)



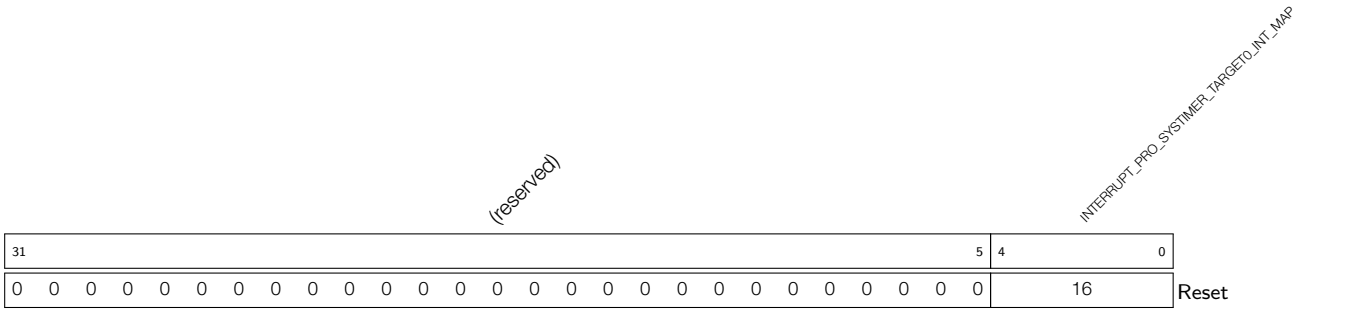
INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP This register is used to map TG1_LACT_EDGE_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.71: INTERRUPT_PRO_CACHE_IA_INT_MAP_REG (0x0118)



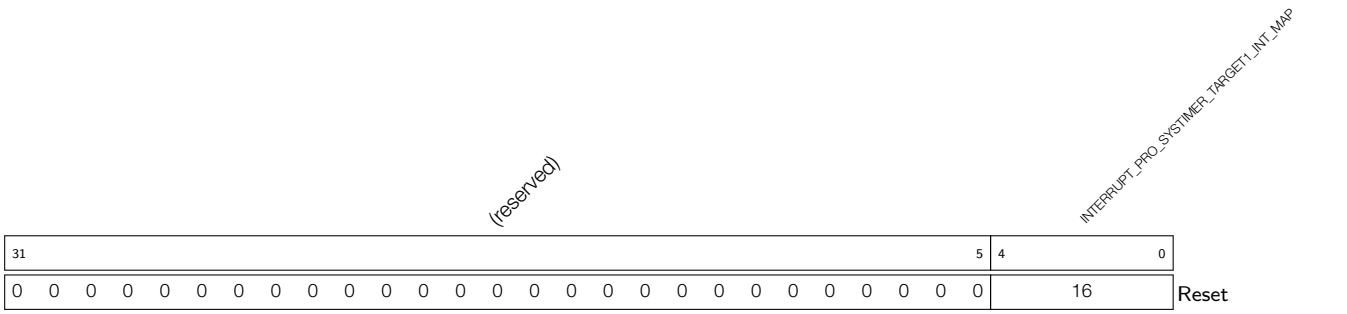
INTERRUPT_PRO_CACHE_IA_INT_MAP This register is used to map CACHE_IA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.72: INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG (0x011C)



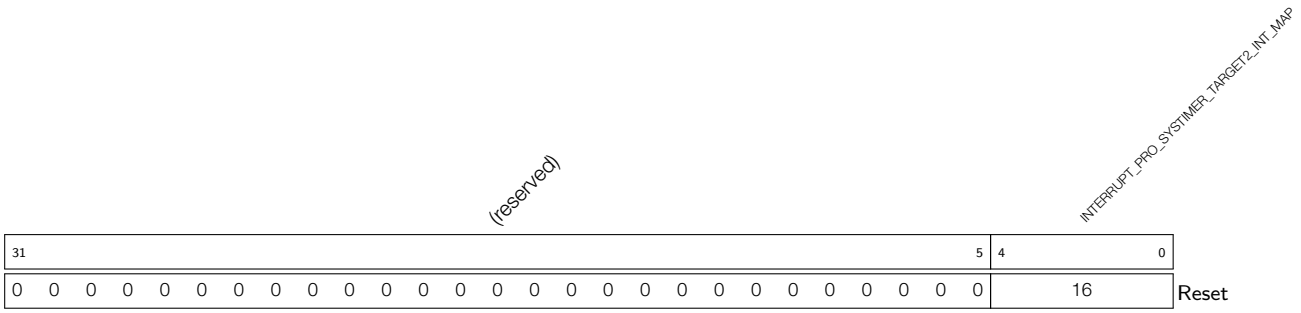
INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP This register is used to map SYSTIMER_TARGET0_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.73: INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG (0x0120)



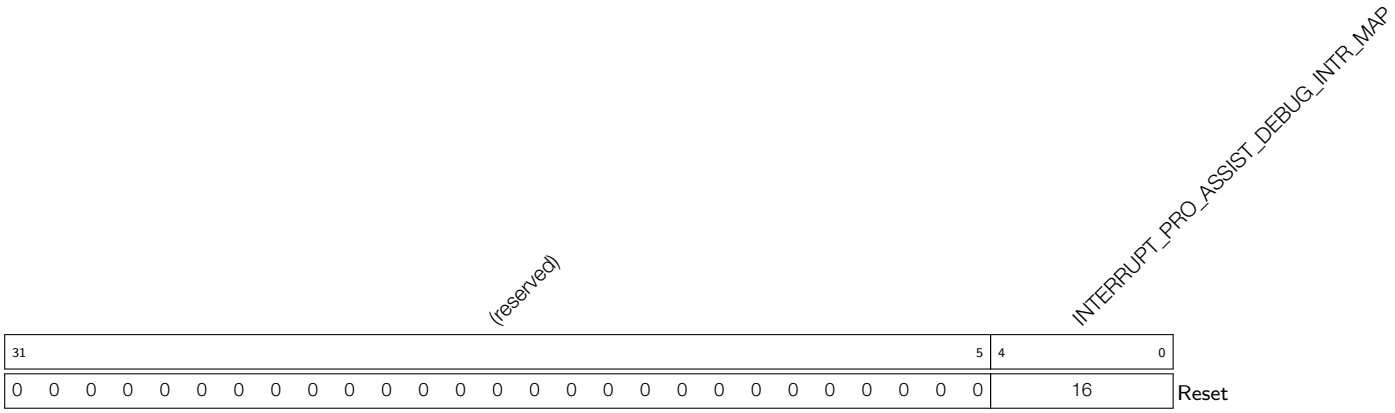
INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP This register is used to map SYSTIMER_TARGET1_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.74: INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG (0x0124)



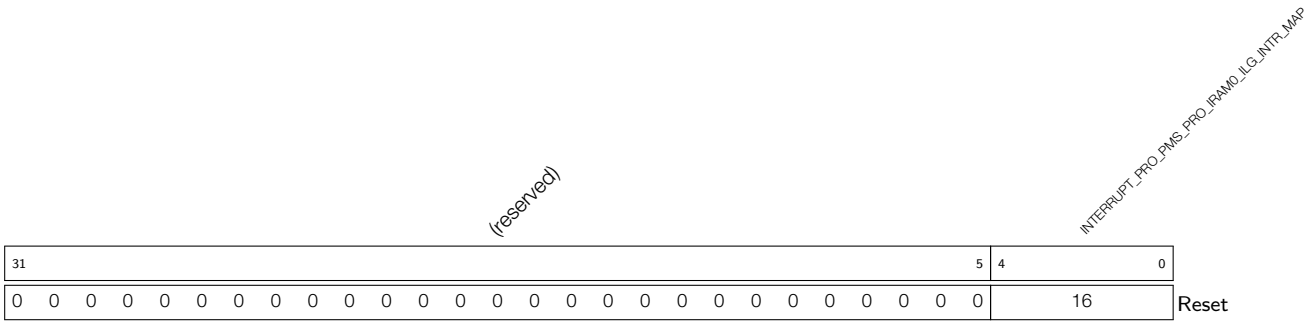
INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP This register is used to map SYSTIMER_TARGET2_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.75: INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG (0x0128)



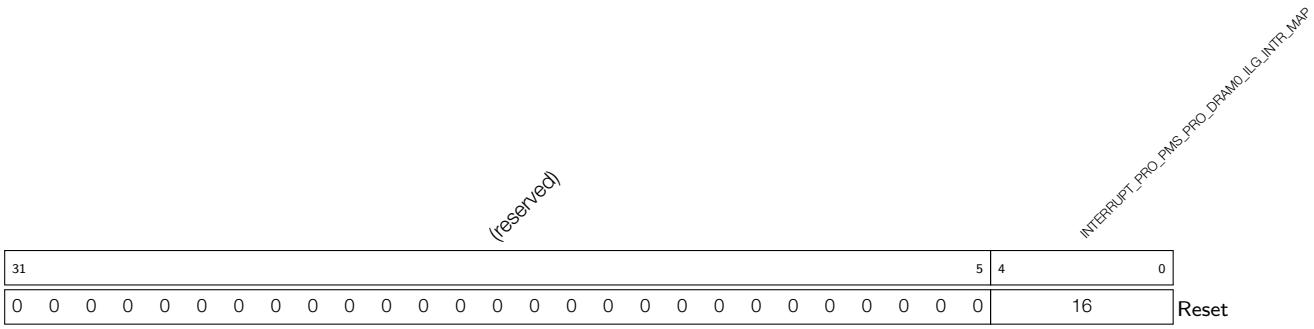
INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP This register is used to map ASSIST_DEBUG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.76: INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG (0x012C)



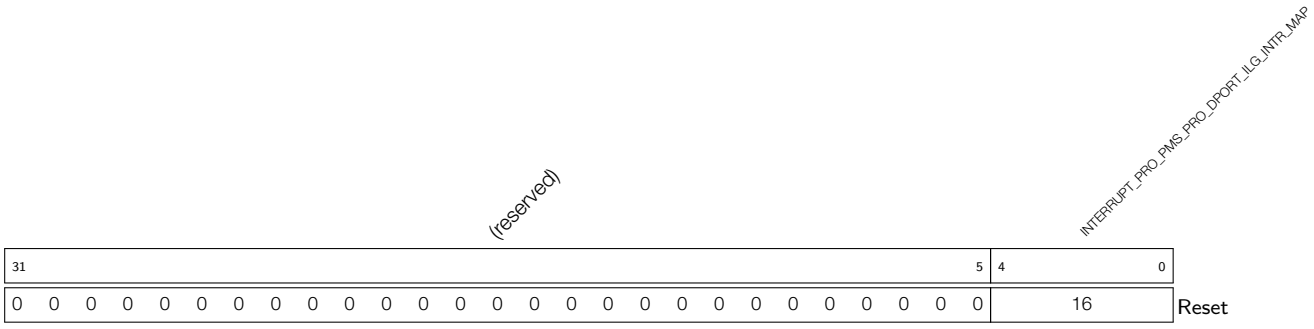
INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP This register is used to map PMS_PRO_IRAM0_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.77: INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG (0x0130)



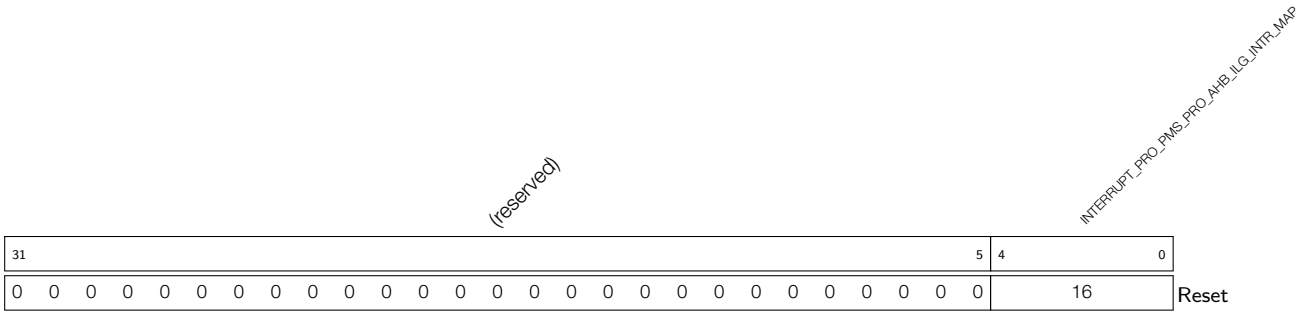
INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP This register is used to map PMS_PRO_DRAM0_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.78: INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG (0x0134)



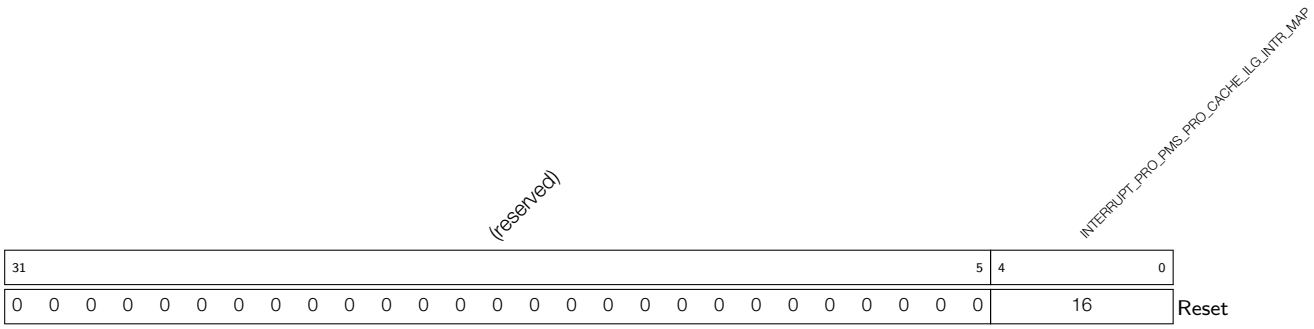
INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP This register is used to map PMS_PRO_DPORT_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.79: INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG (0x0138)



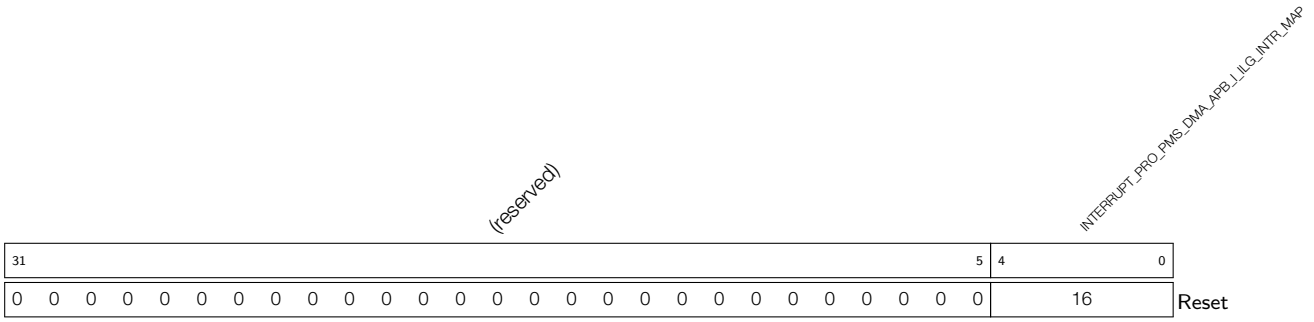
INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP This register is used to map PMS_PRO_AHB_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.80: INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG (0x013C)



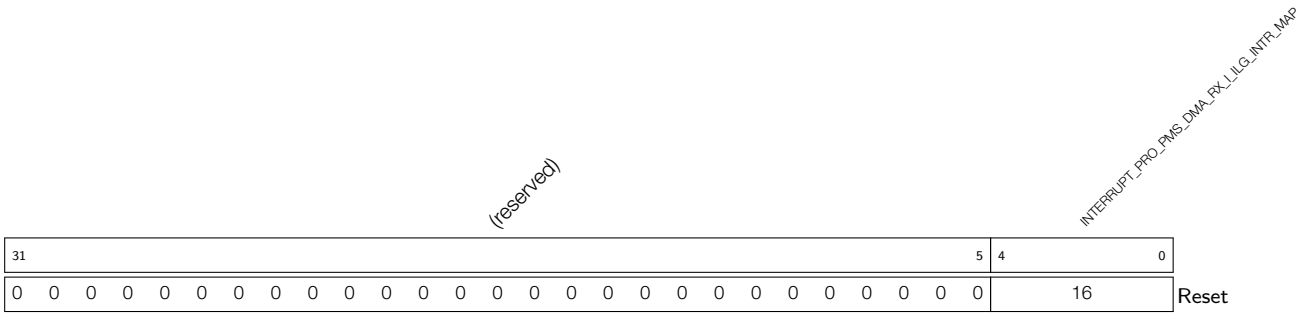
INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP This register is used to map PMS_PRO_CACHE_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.81: INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG (0x0140)



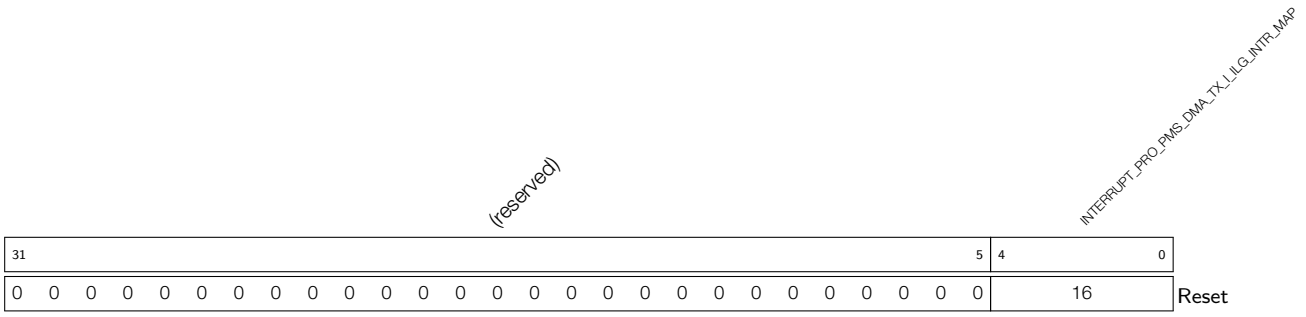
INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP This register is used to map PMS_DMA_APB_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.82: INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG (0x0144)



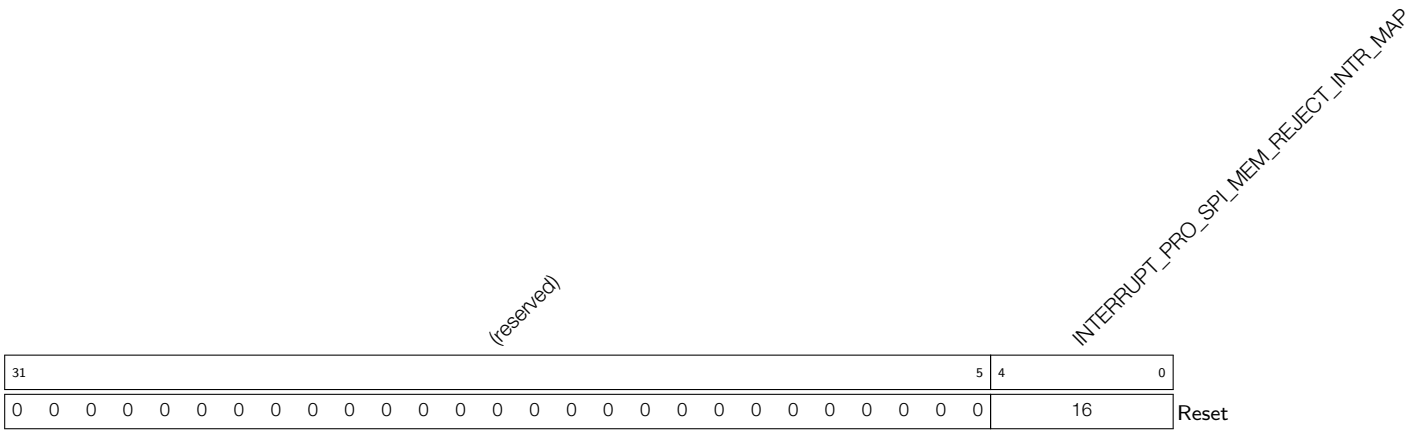
INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP This register is used to map PMS_DMA_RX_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.83: INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG (0x0148)



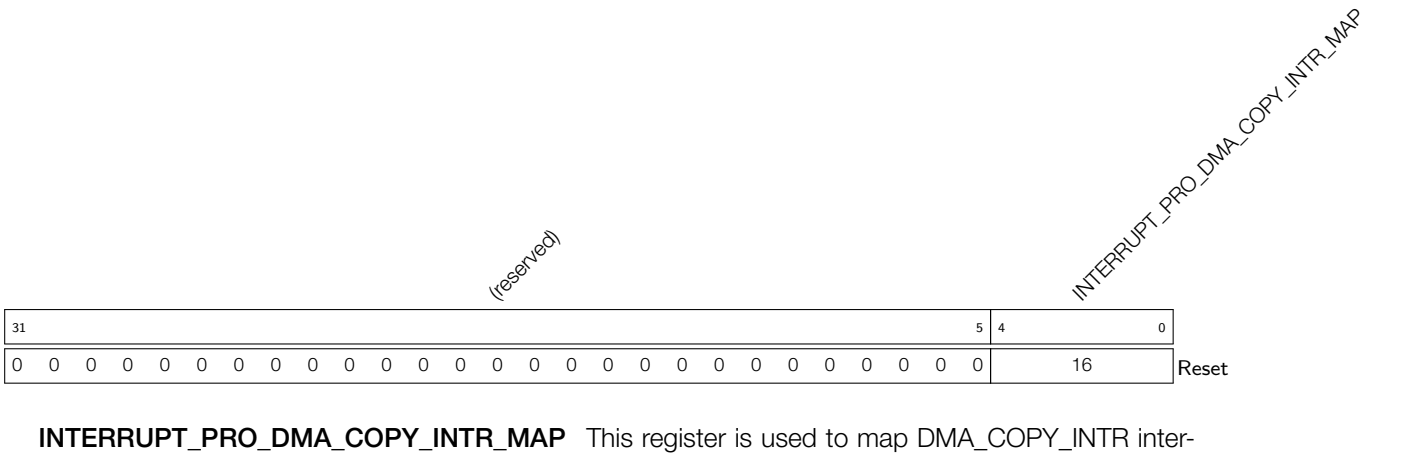
INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP This register is used to map PMS_DMA_TX_I_ILG_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.84: INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG (0x014C)



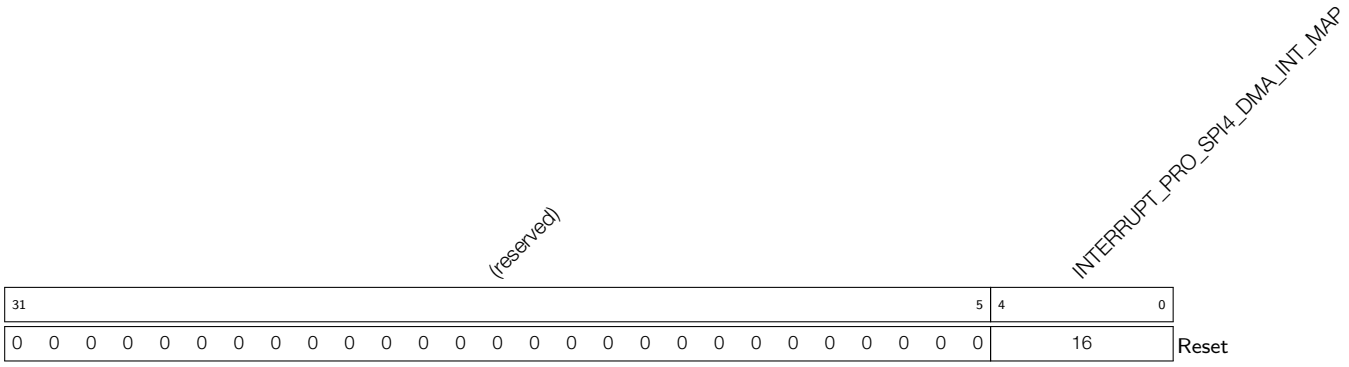
INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP This register is used to map SPI_MEM_REJECT_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.85: INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG (0x0150)



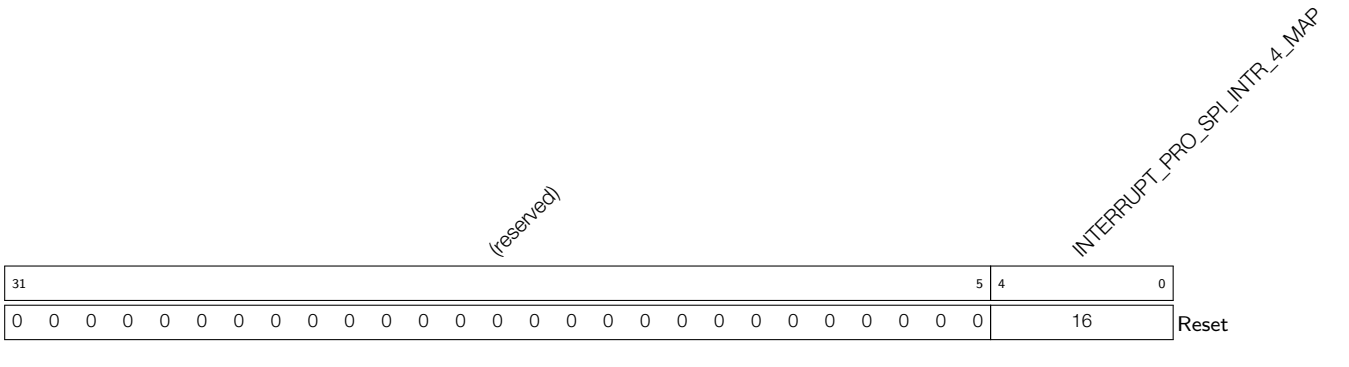
INTERRUPT_PRO_DMA_COPY_INTR_MAP This register is used to map DMA_COPY_INTR interrupt signal to one of the CPU interrupts. (R/W)

Register 4.86: INTERRUPT_PRO_SPI4_DMA_INT_MAP_REG (0x0154)



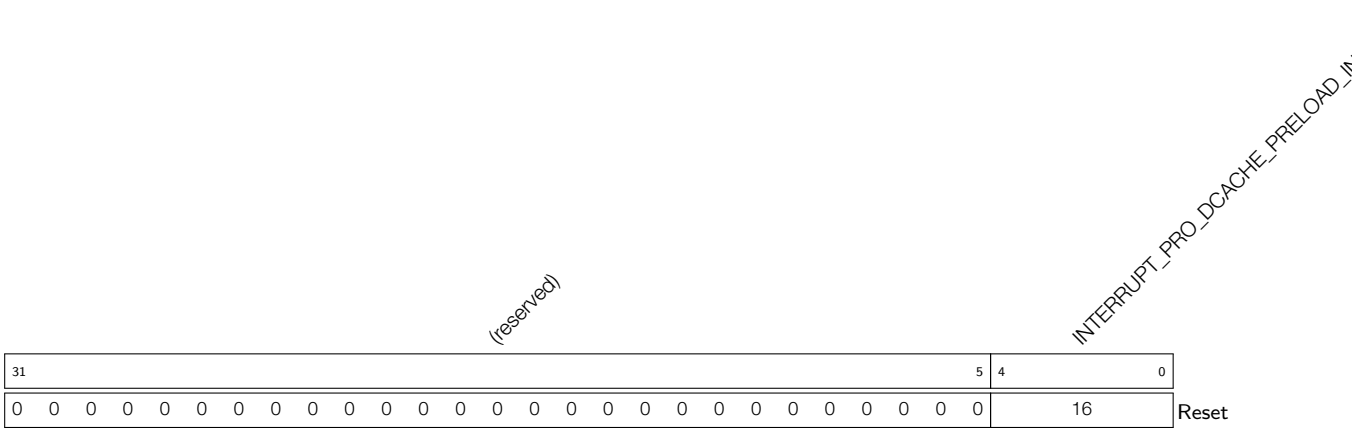
INTERRUPT_PRO_SPI4_DMA_INT_MAP This register is used to map SPI4_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.87: INTERRUPT_PRO_SPI_INTR_4_MAP_REG (0x0158)



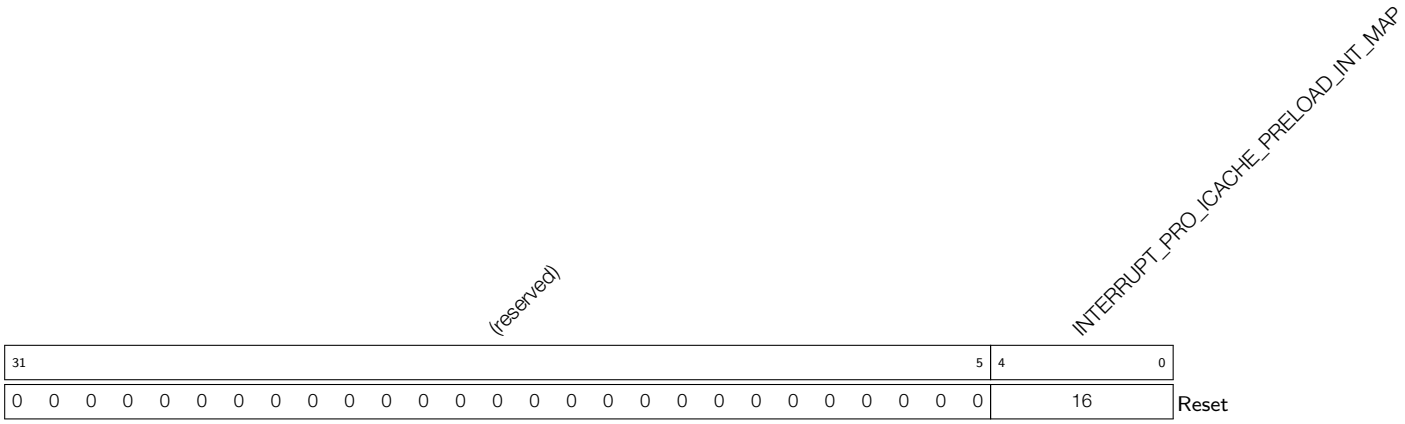
INTERRUPT_PRO_SPI_INTR_4_MAP This register is used to map SPI_INTR_4 interrupt signal to one of the CPU interrupts. (R/W)

Register 4.88: INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG (0x015C)



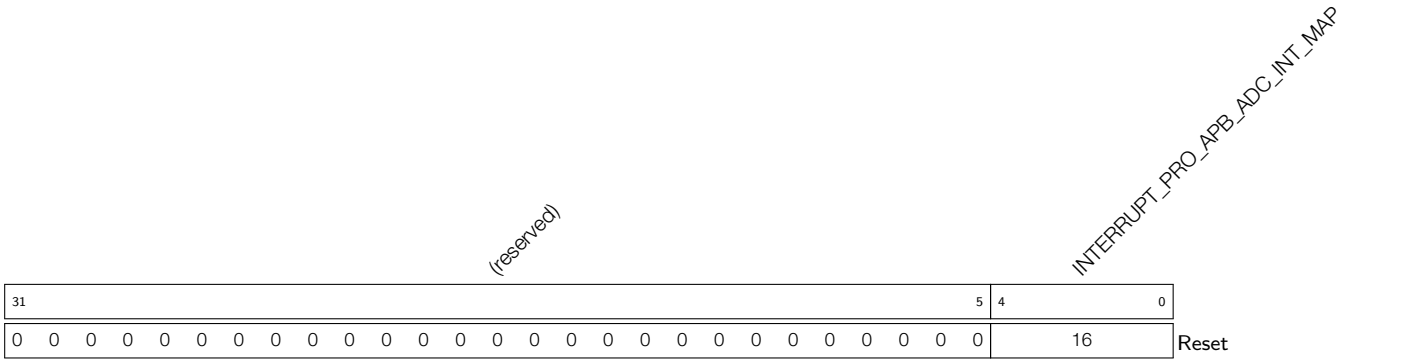
INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP This register is used to map DCACHE_PRELOAD_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.89: INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG (0x0160)



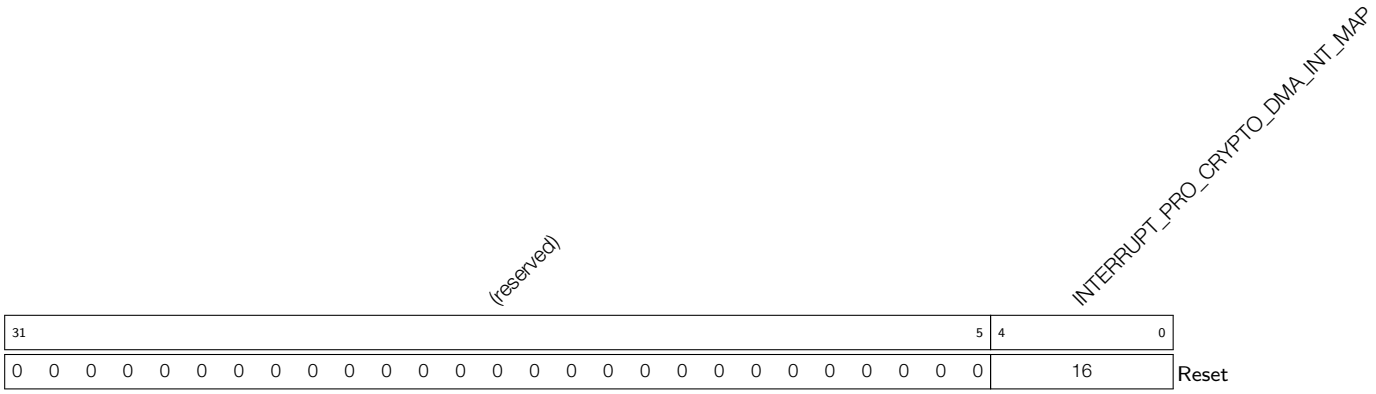
INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP This register is used to map ICACHE_PRELOAD_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.90: INTERRUPT_PRO_APB_ADC_INT_MAP_REG (0x0164)



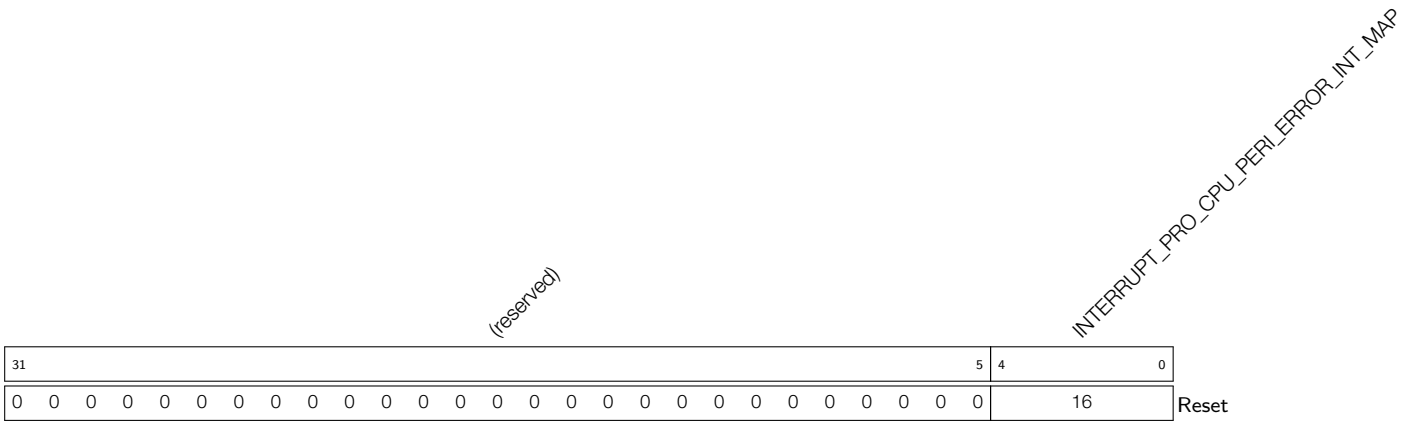
INTERRUPT_PRO_APB_ADC_INT_MAP This register is used to map APB_ADC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.91: INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG (0x0168)



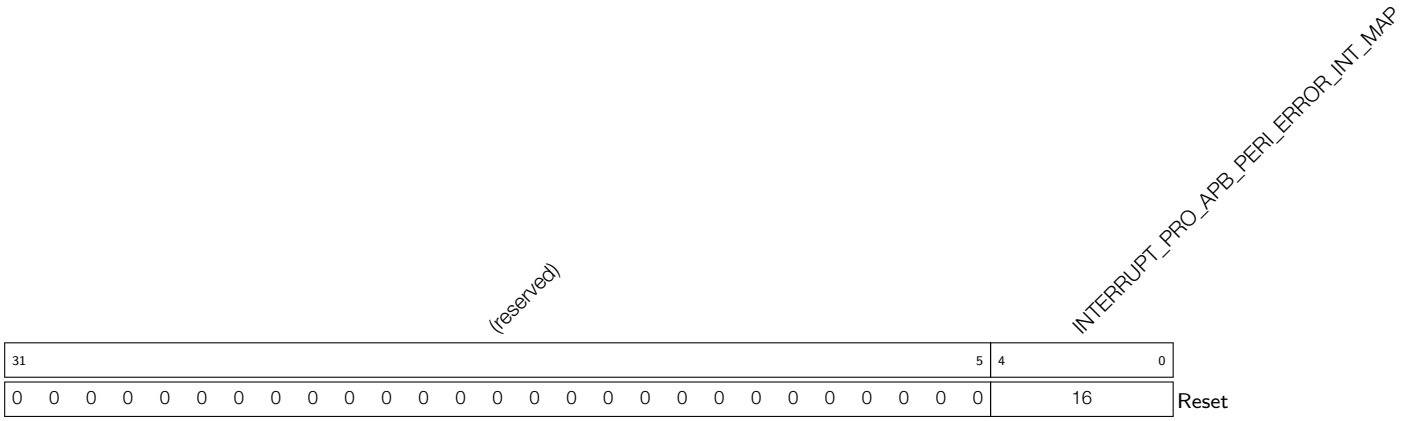
INTERRUPT_PRO_CRYPTO_DMA_INT_MAP This register is used to map CRYPTO_DMA_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.92: INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG (0x016C)



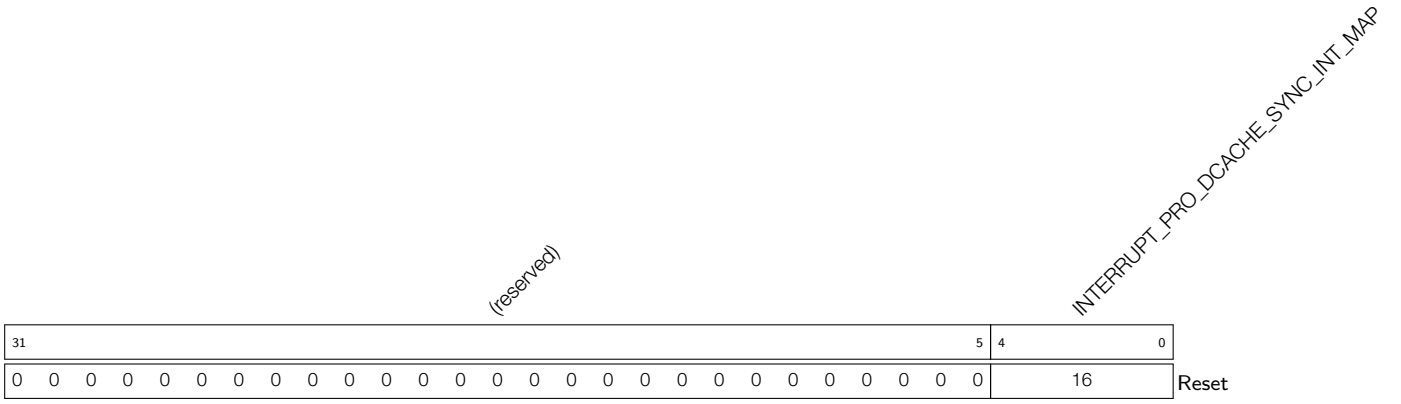
INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP This register is used to map CPU_PERI_ERROR_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.93: INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG (0x0170)



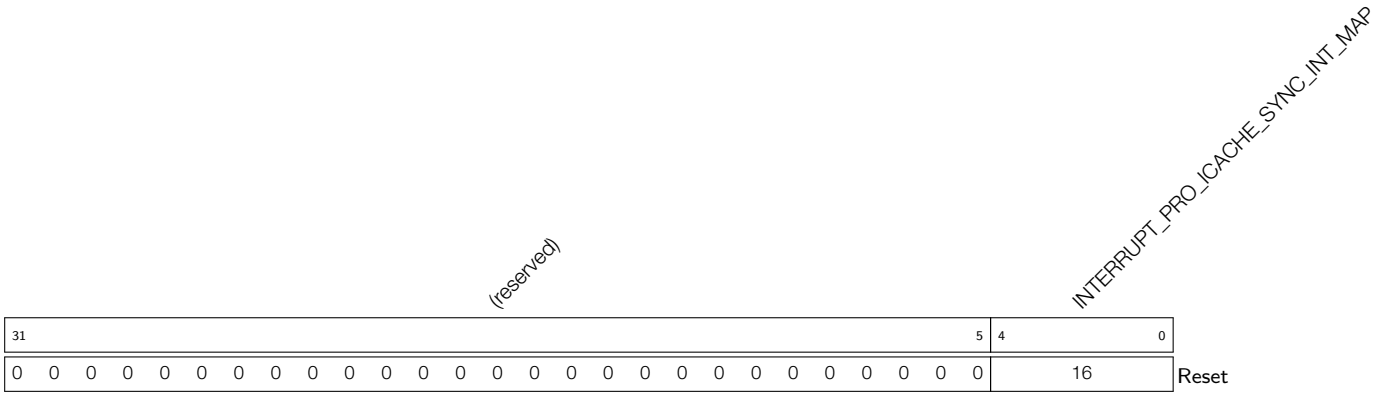
INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP This register is used to map APB_PERI_ERROR_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.94: INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG (0x0174)



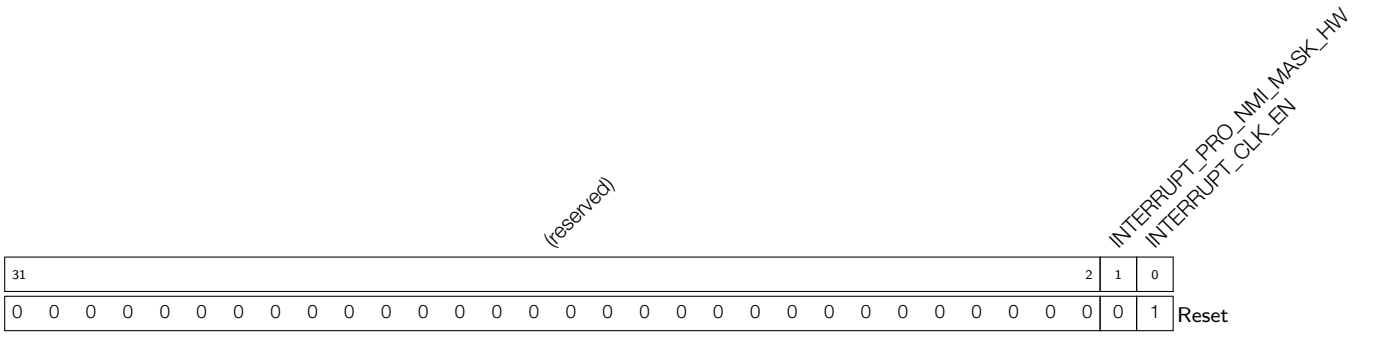
INTERRUPT_PRO_DCACHE_SYNC_INT_MAP This register is used to map DCACHE_SYNC_INT interrupt signal to one of the CPU interrupts. (R/W)

Register 4.95: INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG (0x0178)



INTERRUPT_PRO_ICACHE_SYNC_INT_MAP This register is used to map ICACHE_SYNC_INT interrupt signal to one of the CPU interrupts. (R/W)

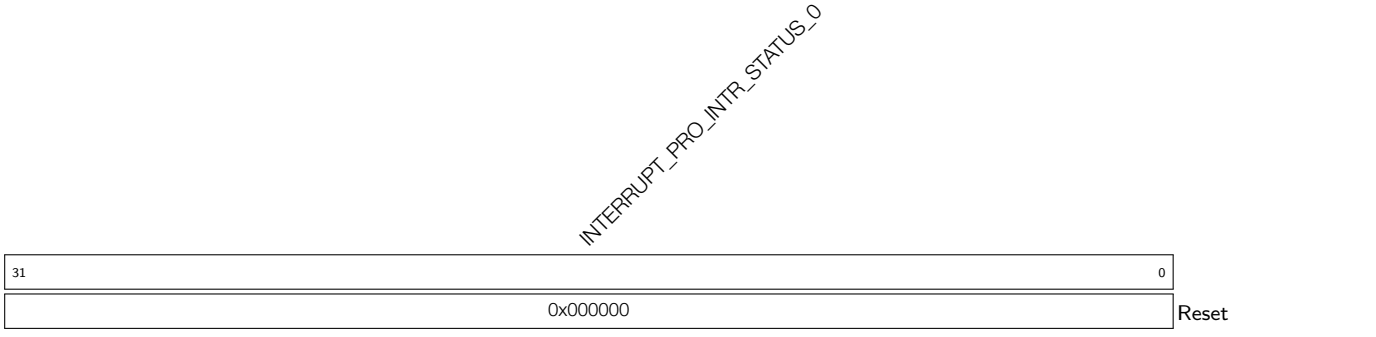
Register 4.96: INTERRUPT_CLOCK_GATE_REG (0x0188)



INTERRUPT_CLK_EN This bit is used to enable or disable the clock of interrupt matrix. 1: enable the clock; 0: disable the clock. (R/W)

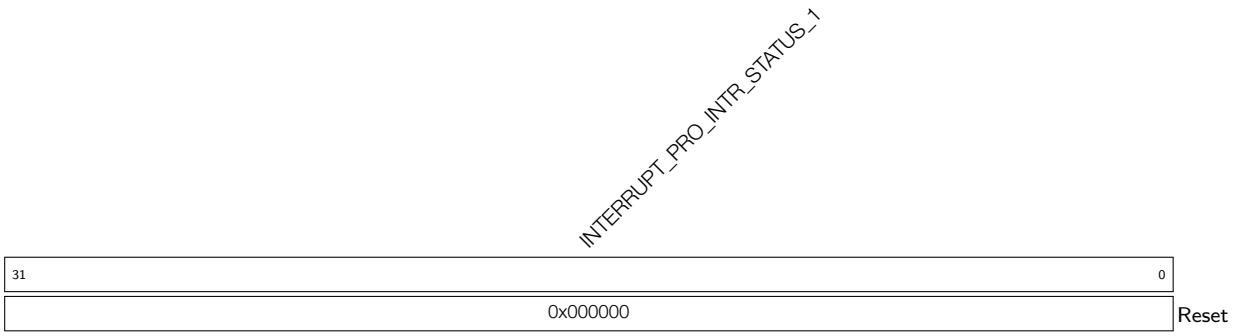
INTERRUPT_PRO_NMI_MASK_HW This bit is used to disable all NMI interrupt signals to CPU. (R/W)

Register 4.97: INTERRUPT_PRO_INTR_STATUS_REG_0_REG (0x017C)



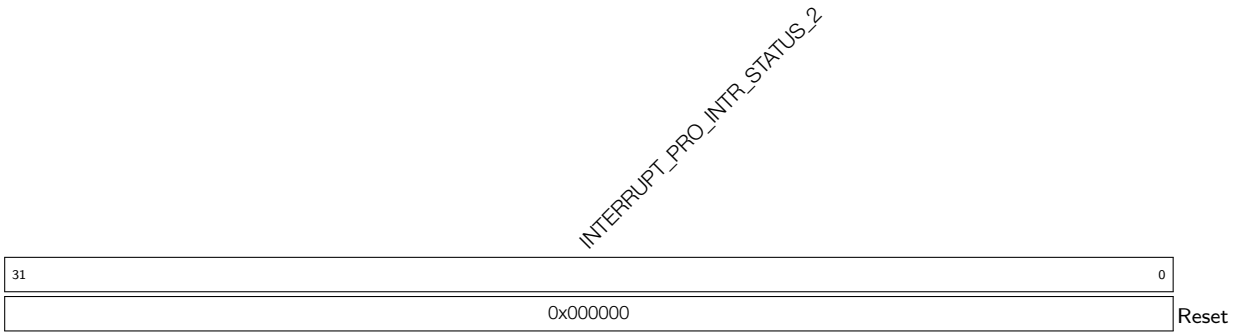
INTERRUPT_PRO_INTR_STATUS_0 This register stores the status of the first 32 input interrupt sources. (RO)

Register 4.98: INTERRUPT_PRO_INTR_STATUS_REG_1_REG (0x0180)



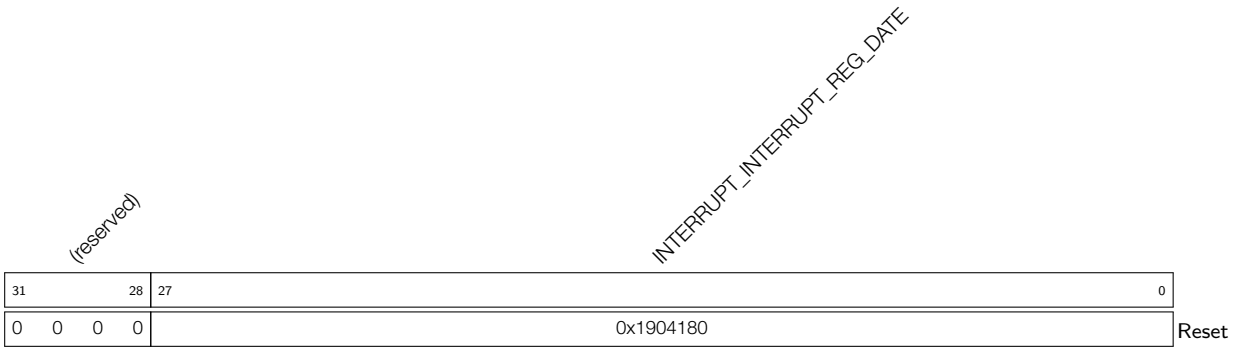
INTERRUPT_PRO_INTR_STATUS_1 This register stores the status of the second 32 input interrupt sources. (RO)

Register 4.99: INTERRUPT_PRO_INTR_STATUS_REG_2_REG (0x0184)



INTERRUPT_PRO_INTR_STATUS_2 This register stores the status of the last 31 input interrupt sources. (RO)

Register 4.100: INTERRUPT_REG_DATE_REG (0x0FFC)



INTERRUPT_DATE Version control register. (R/W)

5. IO MUX and GPIO Matrix

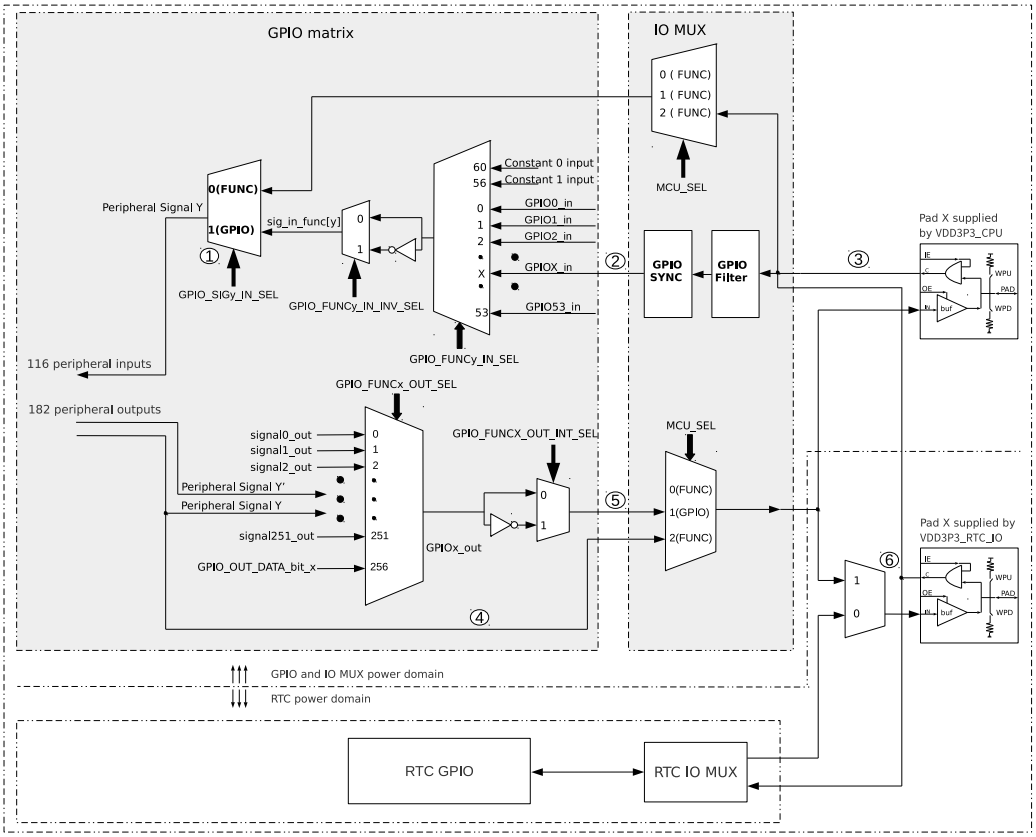
5.1 Overview

The ESP32-S2 chip features 43 physical GPIO pads. Each pad can be used as a general-purpose I/O, or be connected to an internal peripheral signal. The IO MUX, RTC IO MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pads. Together these modules provide highly configurable I/O.

Note that the GPIO pads are numbered from 0 ~ 21 and 26 ~ 46, while GPIO46 is input-only.

This chapter describes the selection and connection of the internal signals for the 43 digital pads and control signals: FUN_SEL, IE, OE, WPU, WPD, etc. These internal signals include:

- 116 digital peripheral input signals, control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.
- 182 digital peripheral output signals, control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.
- fast peripheral input and output signals, control signals: IE, OE, etc.
- 22 RTC GPIO signals



①Not every peripheral input has the MUX, only peripheral input signals (Y: 0~3, 7~11, 14~15, 17~18, 108~117, 127~130, 167~171) have this MUX. The other peripheral inputs only get the input from GPIO Matrix.

②There are 43 GPIO pads(X: 0~21, 26~46), therefore only 43 inputs from GPIO SYNC to GPIO Matrix.

③There are few control signals, IE, OE, WPU, WPD, connected to Pad supplied by VDD_3P3_CPU.

④Only few peripheral outputs have this path, please refer to IO MUX Pad Table in this chapter.

⑤There are only 42 outputs (GPIO pad X: 0~21, 26~45) from GPIO matrix to IO MUX.

⑥There are few control signals, IE, OE, WPU, WPD, connected to Pad supplied by VDD3P3_RTC_IO.

Figure 5-1. IO MUX, RTC IO MUX and GPIO Matrix Overview

Figure 5-1 shows the overview of IO MUX, RTC IO MUX and GPIO matrix.

1. IO MUX provides one configuration register IO_MUX_*n*_REG for each GPIO pad. The pad can be configured to
   - perform GPIO function routed by GPIO matrix;
   - or perform direct connection bypassing GPIO matrix.

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

   See Section 5.11 for the IO MUX functions for each I/O pad.
2. GPIO matrix is a full-switching matrix between the peripheral input/output signals and the pads.
   - For input to the chip: each of the 116 internal peripheral inputs can select any GPIO pad as their input source.
   - For output from the chip: each GPIO pad can select any of the 182 peripheral output signals for its output.

   See Section 5.10 for the list of peripheral signals via GPIO matrix.
3. RTC IO MUX is used to connect GPIO pads to their low-power and analog functions. Only a subset of GPIO pads have these optional RTC functions.

   See Section 5.12 for the list of RTC IO MUX functions.

5.2 Peripheral Input via GPIO Matrix

5.2.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 43 GPIOs (0 ~ 21, 26 ~ 46), see Table 22. Meanwhile, register corresponding to the peripheral should be set to receive input signal via GPIO matrix.

5.2.2 Synchronization

When signals are directed using the GPIO matrix, the signal will be synchronized to the APB bus clock by the GPIO SYNC hardware. This synchronization applies to all GPIO matrix signals but does not apply when using the IO MUX, see Figure 5-1.

Figure 5-2 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.

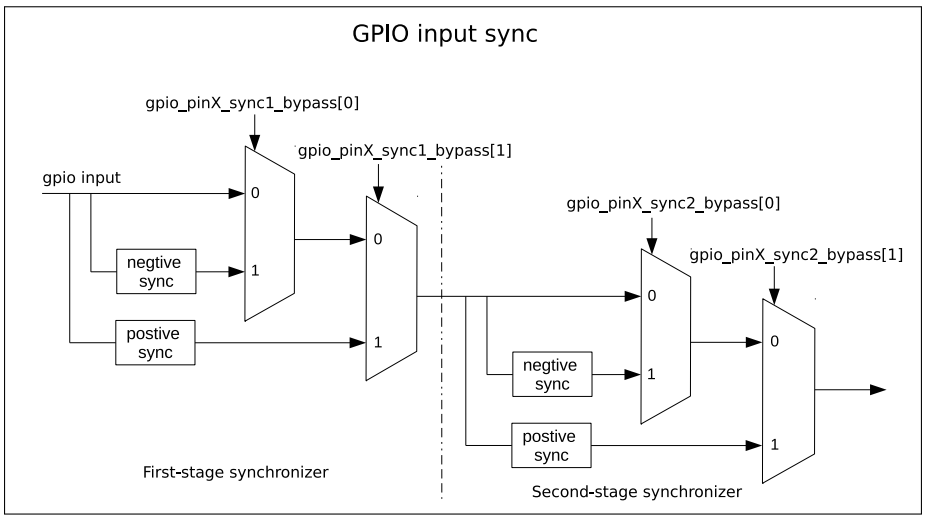


Figure 5-2. GPIO Input Synchronized on Clock Rising Edge or on Falling Edge

5.2.3 Functional Description

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

1. Configure register GPIO_FUNC*y*_IN_SEL_CFG_REG corresponding to peripheral signal *Y* in GPIO matrix:
   - Set GPIO_SIG*y*_IN_SEL to enable peripheral signal input via GPIO matrix.
   - Set GPIO_FUNC*y*_IN_SEL to the value corresponding to GPIO pad *X*.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, there is no MUX module in Figure 5-1 for these signals (see note 1 below Figure 5-1). These peripherals can only receive input signals via GPIO matrix.
2. Enable the filter for pad input signals by setting the register IO_MUX_FILTER_EN. Only the signals with a valid width of more than two clock cycles can be sampled, see Figure 5-3.

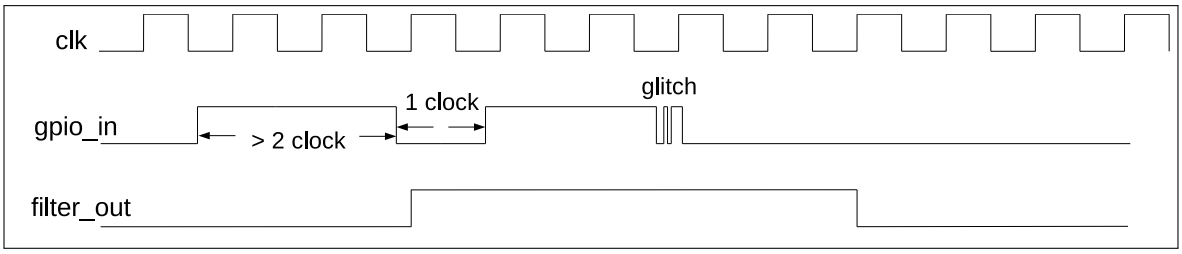


Figure 5-3. Filter Timing Diagram of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pad *X* as follows:
   - Set GPIO_PIN*x*_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first clock, see Figure 5-2.
   - Set GPIO_PIN*x*_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second clock, see Figure 5-2.
4. Configure IO MUX register to enable pad input. For this end, please set IO_MUX_*x*_REG corresponding to GPIO pad *X* as follows:
   - Set IO_MUX_FUN_IE to enable input.

- Set or clear IO_MUX_FUN_WPU and IO_MUX_FUN_WPD, as desired, to enable or disable pull-up and pull-down resistors.

For example, to connect RMT channel 0 input signal (rmt_sig_in0, signal index 83) to GPIO40, please follow the steps below. Note that GPIO40 is also named as MTDO pin.

1. Set GPIO_SIG83_IN_SEL in register GPIO_FUNC83_IN_SEL_CFG_REG to enable peripheral signal input via GPIO matrix.
2. Set GPIO_FUNC83_IN_SEL in register GPIO_FUNC83_IN_SEL_CFG_REG to 40.
3. Set IO_MUX_FUN_IE in register IO_MUX_GPIO40_REG to enable pad input.

Note:

- One input pad can be connected to multiple peripheral input signals.
- The input signal can be inverted by configuring GPIO_FUNC*y*_IN_INV_SEL.
- It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pad. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
  - When GPIO_FUNC*y*_IN_SEL is 0x3C, input signal *X* is always 0.
  - When GPIO_FUNC*y*_IN_SEL is 0x38, input signal *X* is always 1.

5.2.4 Simple GPIO Input

GPIO_IN_REG/GPIO_IN1_REG holds the input values of each GPIO pad. The input value of any GPIO pad can be read at any time without configuring GPIO matrix for a particular peripheral signal. However, it is necessary to enable the input in IO MUX by setting IO_MUX_FUN_IE bit in register IO_MUX_*n*_REG corresponding to pad *X*, as mentioned in Section 5.2.2.

5.3 Peripheral Output via GPIO Matrix

5.3.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (0 ~ 11, 14 ~ 18, and etc.) to one of the 42 GPIOs (0 ~ 21, 26 ~ 45). See Table 22.

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 pad to GPIO function. This causes the output GPIO signal to be connected to the pad.

Note:
There is a range of peripheral output signals (223 ~ 227) which are not connected to any peripheral. These can be used to input a signal from one GPIO pad and output directly to another GPIO pad.

5.3.2 Functional Description

Some of the 182 output signals can be set to go through GPIO matrix into IO MUX and then to a pad. Figure 5-1 illustrates the configuration.

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

1. Configure register GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pad *X* in GPIO matrix. Recommended operation: use corresponding W1TS (write 1 to set) and W1TC (write 1 to clear) registers to set or clear GPIO_ENABLE_REG.
   - Set the GPIO_FUNC*x*_OUT_SEL field in register GPIO_FUNC*x*_OUT_SEL_CFG_REG to the index of the desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OEN_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG or in register GPIO_ENABLE1_W1TS_REG, corresponding to GPIO pad *X*. To have the output enable signal decided by internal logic (see the column "Output enable of output signals" in Table 22), clear GPIO_FUNC*x*_OEN_SEL bit instead.
   - Clear the corresponding bit in register GPIO_ENABLE_W1TC_REG or in register GPIO_ENABLE1_W1TC_REG to disable the output from the GPIO pad.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pad *X*.
3. Configure IO MUX register to enable output via GPIO matrix. Set the IO_MUX_*x*_REG corresponding to GPIO pad *X* as follows:
   - Set the field IO_MUX_MCU_SEL to IO_MUX function corresponding to GPIO pad *X*. This is Function 1, numeric value 1, for all pins.
   - Set the IO_MUX_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 value)
     - 3: ~40 mA
   - If using open drain mode, set/clear the IO_MUX_FUN_WPU and IO_MUX_FUN_WPD bits to enable/disable the internal pull-up/down resistors.

Note:

- The output signal from a single peripheral can be sent to multiple pads simultaneously.
- GPIO46 can not be used as an output.
- The output signal can be inverted by setting GPIO_FUNC*n*_OUT_INV_SEL bit.

5.3.3 Simple GPIO Output

GPIO matrix can also be used for simple GPIO output. This can be done as below:

- Set GPIO matrix GPIO_FUNC*n*_OUT_SEL with a special peripheral index 256 (0x100);
- Set the corresponding bit in GPIO_OUT_REG[31:0] or GPIO_OUT1_REG[21:0] register to the desired GPIO output value.

Note:

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31] correspond to GPIO0 ~ GPIO31, and GPIO_OUT1_REG[25:22] are invalid.
- GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[13] correspond to GPIO32 ~ GPIO45, and GPIO_OUT1_REG[21:14] 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/GPIO_OUT1_REG.

5.3.4 Sigma Delta Modulated Output

5.3.4.1 Functional Description

ESP32-S2 provides a second-order sigma delta modulation module and eight independent modulation channels. The channels are capable to output 1-bit signals (output index: 100 ~ 107) with sigma delta modulation, and by default output is enabled for these channels. This module can also output PDM (pulse density modulation) signal with configurable duty cycle. The transfer function is:

$$H(z) = \text{X(z)z}^{-1} + \text{E(z)(1-z}^{-1})^2$$

E(z) is quantization error and X(z) is the input.
Sigma Delta modulator supports scaling down of APB_CLK by divider 1 ~ 256:

- Set GPIOSD_FUNCTION_CLK_EN to enable the modulator clock.
- Configure register GPIOSD_SD*n*_PRESCALE (*n* is 0 ~ 7 for eight channels).

After scaling, the clock cycle is equal to one pulse output cycle from the modulator.

GPIOSD_SD*n*_IN is a signed number with a range of [-128, 127] and is used to control the duty cycle [1] of PDM output signal.

- GPIOSD_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIOSD_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIOSD_SD*n*_IN = 127, the duty cycle of the output signal is close to 100%.

The formula for calculating PDM signal duty cycle is shown as below:

$$Duty\_Cycle = \frac{GPIOSD\_SDn\_IN + 128}{256}$$

Note:
For PDM signals, duty cycle refers to the percentage of high level cycles to the whole statistical period (several pulse cycles, for example 256 pulse cycles).

5.3.4.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pad via GPIO matrix, see Section 5.3.2.
- Enable the modulator clock by setting the register GPIOSD_FUNCTION_CLK_EN.

- Configure the divider value by setting the register GPIOSD_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting the register GPIOSD_SD*n*_IN.

5.4 Dedicated GPIO

5.4.1 Overview

The dedicated GPIO module, consisting of eight input/output channels, is specially designed for CPU interaction with GPIO matrix and IO MUX. Peripheral input/output signals for input/output channels both are indexed from 235 to 242. By default, the output is enabled for output channels.

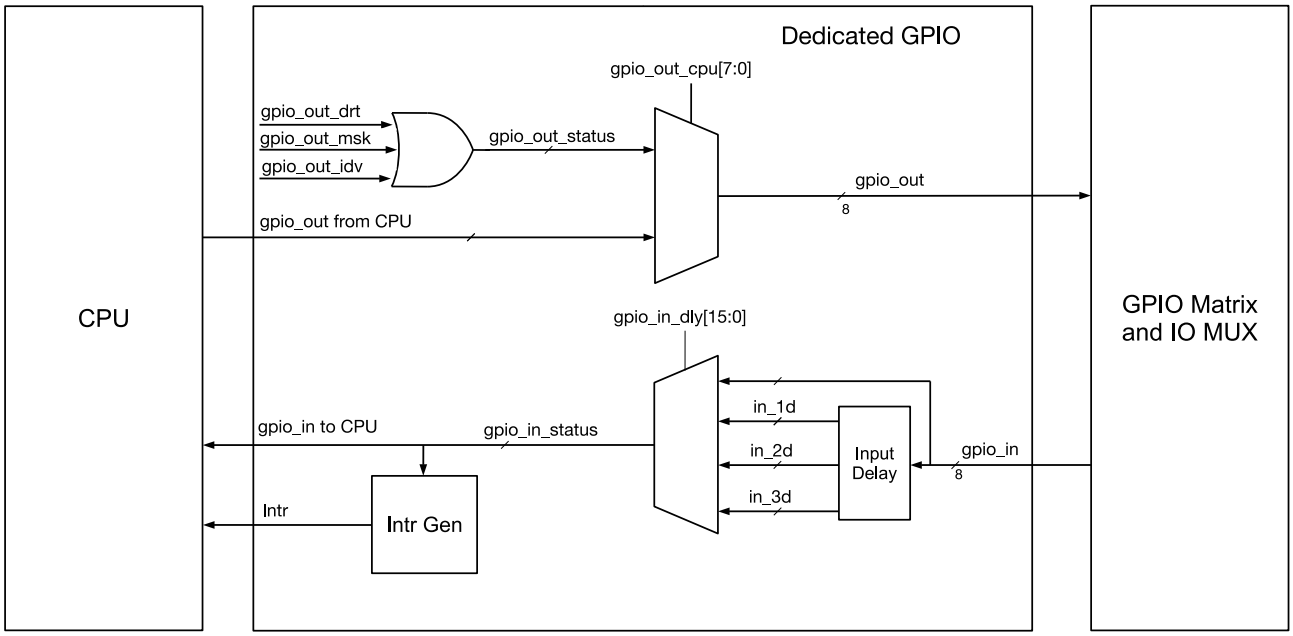


Figure 5-4. Dedicated GPIO Diagram

Figure 5-4 shows the structure of dedicated GPIO module. Users can enable the clock of this module by setting the bit SYSTEM_CLK_EN_DEDICATED_GPIO in register SYSTEM_CPU_PERI_CLK_EN_REG, and reset this module by setting the bit SYSTEM_RST_EN_DEDICATED_GPIO in register SYSTEM_CPU_PERI_RST_EN_REG first, and then clearing this bit. For more information, please refer to Table 33 *Peripheral Clock Gating and Reset Bits* in Chapter 6 *System Registers*.

5.4.2 Features

Dedicated GPIO module has the following features:

- Eight output and eight input channels
- Each channel accessible with registers or directly by CPU instructions
- Configurable delay on input channels
- Interrupts on input channels

5.4.3 Functional Description

Dedicated GPIOs may be accessed using registers or directly by calling specific CPU instructions.

When accessing output channels, select between registers and CPU by configuring DEDIC_GPIO_OUT_CPU_REG:

- DEDIC_GPIO_OUT_CPU_SEL*n* = 0, drive GPIO output via registers.
- DEDIC_GPIO_OUT_CPU_SEL*n* = 1, drive GPIO output via CPU instructions.

The dedicated GPIO module also provides two ways to read input channels:

- Query GPIO input value via registers.
- Read GPIO input value via CPU instructions.

5.4.3.1 Accessing GPIO via Registers

Users can control GPIO output via registers in the following ways:

- Write GPIO output value directly by configuring the register DEDIC_GPIO_OUT_DRT_REG.
- Write GPIO output value via masked access by configuring the register DEDIC_GPIO_OUT_MSK_REG.
- Write GPIO output value via individual bits by configuring the register DEDIC_GPIO_OUT_IDV_REG.

User can read the register DEDIC_GPIO_OUT_SCAN_REG to check GPIO status, i.e. gpio_out_status in Figure 5-4, via software.

Users can get a dedicated GPIO input value by reading the register DEDIC_GPIO_IN_SCAN_REG, i.e. the gpio_in_status in Figure 5-4, via software.

The dedicated GPIO module supports for a delay of 1/2/3 clock cycle(s) for input signals, or with no delay, which can be controlled by configuring the register DEDIC_GPIO_IN_DLY_REG for each individual channel. GPIO input status is indicated by interrupts. Users can configure the register DEDIC_GPIO_INTR_RCGN_REG to set trigger modes:

- 0/1: no interrupt
- 2: low level trigger
- 3: high level trigger
- 4: falling edge trigger
- 5: rising edge trigger
- 6/7: edges trigger

5.4.3.2 Accessing GPIO with CPU

CPU can also read/write a dedicated GPIO via instructions.

- Set bits in output channel.
  Assembly syntax: SET_BIT_GPIO_OUT mask
  Function: write 1 to set the corresponding bits in user register GPIO_OUT, i.e. the gpio_out in Figure 5-4. The “mask” is 8 bits wide. The bits in GPIO_OUT, corresponding to the bits in “mask” with the value of 1, will be set to 1, while the other bits in GPIO_OUT remain unaffected.

- Clear bits in output channel.
  Assembly syntax: CLR_BIT_GPIO_OUT mask
  Function: write 1 to clear the corresponding bits in user register GPIO_OUT. The "mask" is 8 bits wide. The bits in GPIO_OUT, corresponding to the bits in "mask" with the value of 1, will be cleared, while the other bits in GPIO_OUT remain unaffected.
- Set or clear bits in output channel with masked access
  Assembly syntax: WR_MASK_GPIO_OUT value, mask
  Function: write value to user register GPIO_OUT via masked access. The "value" is 8 bits wide, which represents the value to write. The "mask" is 8 bits wide, which represents the bits in GPIO_OUT to be manipulated. For example, mask 0x03 (0000 0011) indicates that the write value is only valid for GPIO_OUT[0] and GPIO_OUT[1]. Only the bits in GPIO_OUT, corresponding to the bits with the value of 1 in the "mask", are updated, that is, updated to the value stored in "value".
- Write "art" register to the output channel.
  Assembly syntax: WUR.GPIO_OUT art
  Function: write the value of the address register "art" to the user register GPIO_OUT. Register "art" is 32 bits wide, and only low 8 bits are valid when using this instruction.
- Read the output channel to "arr" register.
  Assembly syntax: RUR.GPIO_OUT arr
  Function: read the value of the user register GPIO_OUT to the address register "arr". Register "arr" is 32 bits wide, and only low 8 bits are valid when using this instruction.
- Read the input channel to "I" register.
  Assembly syntax: GET_GPIO_IN I
  Function: read the value of the user register GPIO_IN, i.e. the gpio_in in Figure 5-4, to the address register "I". Register "I" is 32 bits wide, the high 24 bits of which are 0, and low 8 bits of which corresponds to the value of the user register GPIO_IN.

5.5 Direct I/O via IO MUX

5.5.1 Overview

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

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

5.5.2 Functional Description

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

1. IO_MUX_MCU_SEL for the GPIO pad must be set to the required pad function. For the list of pad functions, please refer to Section 5.11.
2. Set GPIO_SIG*n*_IN_SEL to low level to route the input directly to the peripheral.

To bypass GPIO matrix for peripheral output signals, IO_MUX_MCU_SEL for the GPIO pad must be set to the required pad function. For the list of pad functions, please refer to Section 5.11.

Note:

For peripheral I/O signals, not all signals can be connected to peripheral via IO MUX. Some specific input signals and some specific output signals can only be connected to peripheral via GPIO matrix.

5.6 RTC IO MUX for Low Power and Analog I/O

5.6.1 Overview

22 GPIO pads have low power capabilities (RTC domain) and analog functions which are handled by the RTC subsystem of ESP32-S2. IO MUX and GPIO matrix are not used for these functions, rather, RTC IO MUX is used to redirect input/output signals to the RTC subsystem.

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

Section 5.12 lists the RTC_MUX pins and their functions.

5.6.2 Functional Description

Each pad with analog and RTC functions is controlled by RTCIO_TOUCH_PAD*n*_MUX_SEL bit in register RTCIO_TOUCH_PAD*n*_REG. By default all bits in these registers are set to 0, routing all input/output signals via IO MUX.

If RTCIO_TOUCH_PAD*n*_MUX_SEL is set to 1, then input/output signals to and from that pad is routed to the RTC subsystem. In this mode, RTCIO_TOUCH_PAD*n*_REG is used for digital input/output and the analog features of the pad are also available.

Please refer to Section 5.12 for the list of RTC pin functions and the mapping table of GPIO pads to their analog functions. Note that RTCIO_TOUCH_PAD*n*_REG applies the RTC GPIO pin numbering, not the GPIO pad numbering.

5.7 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-S2 is in Light-sleep mode. If IO_MUX_SLP_SEL in register IO_MUX_*n*_REG for a GPIO pad is set to 1, a different set of bits will be used to control the pad when the chip is in Light-sleep mode.

Table 21: Pin Function Register for IO MUX Light-sleep Mode

IO MUX Function	Normal Execution OR IO_MUX_SLP_SEL = 0	Light-sleep Mode AND IO_MUX_SLP_SEL = 1
Output Drive Strength	IO_MUX_FUN_DRV	IO_MUX_FUN_DRV
Pullup Resistor	IO_MUX_FUN_WPU	IO_MUX_MCU_WPU
Pulldown Resistor	IO_MUX_FUN_WPD	IO_MUX_MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)[1]	IO_MUX_MCU_OE

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

Note:
Please refer to Section 5.3.2 for how to enable output in normal execution (when IO_MUX_SLP_SEL = 0).

5.8 Pad Hold Feature

Each IO pad (including the RTC pads) has an individual hold function controlled by a RTC register. When the pad 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 pads to retain the pad state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

Note:

- For digital pads, to maintain pad input/output status in Deep-sleep mode, users can set RTC_CNTL_DG_PAD_FORCE_UNHOLD to 0 before powering down. For RTC pads, the input and output values are controlled by the corresponding bits of register RTC_CNTL_PAD_HOLD_REG, and users can set it to 1 to hold the value or set it to 0 to unhold the value.
- For digital pads, to disable the hold function after the chip is woken up, users can set RTC_CNTL_DG_PAD_FORCE_UNHOLD to 1. To maintain the hold function of the pad, users can change the corresponding bit in register RTC_CNTL_PAD_HOLD_REG to 1.

5.9 I/O Pad Power Supplies

For more information on the power supply for IO pads, please refer to Pin Definition in *ESP32-S2 Datasheet*.

5.9.1 Power Supply Management

Each ESP32-S2 digital pin is connected to one of the four different power domains.

- VDD3P3_RTC_IO: the input power supply for both RTC and CPU
- VDD3P3_CPU: the input power supply for CPU
- VDD3P3_RTC: the input power supply for RTC analog part
- VDD_SPI: configurable power supply

VDD_SPI can be configured to use an internal LDO. The LDO input is VDD3P3_RTC_IO and the output is 1.8 V. If the LDO is not enabled, VDD_SPI is connected directly to the same power supply as VDD3P3_RTC_IO.

The VDD_SPI configuration is determined by the value of strapping pin GPIO45, or can be overriden by eFuse and/or register settings. See *ESP32-S2 Datasheet* sections Power Scheme and Strapping Pins for more details.

5.10 Peripheral Signal List

Table 22 shows the peripheral input/output signals via GPIO matrix.

Table 22: GPIO Matrix

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe
1	SPID_in	0	yes	SPID_out	SPID_oe
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
4	-	-	-	SPICLK_out_mux	SPICLK_oe
5	-	-	-	SPICS0_out	SPICS0_oe
6	-	-	-	SPICS1_out	SPICS1_oe
7	SPID4_in	0	yes	SPID4_out	SPID4_oe
8	SPID5_in	0	yes	SPID5_out	SPID5_oe
9	SPID6_in	0	yes	SPID6_out	SPID6_oe
10	SPID7_in	0	yes	SPID7_out	SPID7_oe
11	SPIDQS_in	0	yes	SPIDQS_out	SPIDQS_oe
14	U0RXD_in	0	yes	U0TXD_out	1'd1
15	U0CTS_in	0	yes	U0RTS_out	1'd1
16	U0DSR_in	0	no	U0DTR_out	1'd1
17	U1RXD_in	0	yes	U1TXD_out	1'd1
18	U1CTS_in	0	yes	U1RTS_out	1'd1
21	U1DSR_in	0	no	U1DTR_out	1'd1
23	I2S0O_BCK_in	0	no	I2S0O_BCK_out	1'd1
25	I2S0O_WS_in	0	no	I2S0O_WS_out	1'd1
27	I2S0I_BCK_in	0	no	I2S0I_BCK_out	1'd1
28	I2S0I_WS_in	0	no	I2S0I_WS_out	1'd1
29	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe
30	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe
39	pcnt_sig_ch0_in0	0	no	gpio_wlan_prio	1'd1
40	pcnt_sig_ch1_in0	0	no	gpio_wlan_active	1'd1
41	pcnt_ctrl_ch0_in0	0	no	-	1'd1
42	pcnt_ctrl_ch1_in0	0	no	-	1'd1
43	pcnt_sig_ch0_in1	0	no	-	1'd1
44	pcnt_sig_ch1_in1	0	no	-	1'd1
45	pcnt_ctrl_ch0_in1	0	no	-	1'd1
46	pcnt_ctrl_ch1_in1	0	no	-	1'd1
47	pcnt_sig_ch0_in2	0	no	-	1'd1
48	pcnt_sig_ch1_in2	0	no	-	1'd1
49	pcnt_ctrl_ch0_in2	0	no	-	1'd1
50	pcnt_ctrl_ch1_in2	0	no	-	1'd1
51	pcnt_sig_ch0_in3	0	no	-	1'd1
52	pcnt_sig_ch1_in3	0	no	-	1'd1
53	pcnt_ctrl_ch0_in3	0	no	-	1'd1
54	pcnt_ctrl_ch1_in3	0	no	-	1'd1
64	usb_otg_iddig_in	0	no	-	1'd1
65	usb_otg_avalid_in	0	no	-	1'd1
66	usb_srp_bvalid_in	0	no	usb_otg_idpullup	1'd1
67	usb_otg_vbusvalid_in	0	no	usb_otg_dppulldown	1'd1
68	usb_srp_sessend_in	0	no	usb_otg_dmpulldown	1'd1

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
69	-	-	-	usb_otg_drvvbus	1'd1
70	-	-	-	usb_srp_chrgvbus	1'd1
71	-	-	-	usb_srp_dischrgvbus	1'd1
72	SPI3_CLK_in	0	no	SPI3_CLK_out_mux	SPI3_CLK_oe
73	SPI3_Q_in	0	no	SPI3_Q_out	SPI3_Q_oe
74	SPI3_D_in	0	no	SPI3_D_out	SPI3_D_oe
75	SPI3_HD_in	0	no	SPI3_HD_out	SPI3_HD_oe
76	SPI3_CS0_in	0	no	SPI3_CS0_out	SPI3_CS0_oe
77	-	-	-	SPI3_CS1_out	SPI3_CS1_oe
78	-	-	-	SPI3_CS2_out	SPI3_CS2_oe
79	-	-	-	ledc_ls_sig_out0	1'd1
80	-	-	-	ledc_ls_sig_out1	1'd1
81	-	-	-	ledc_ls_sig_out2	1'd1
82	-	-	-	ledc_ls_sig_out3	1'd1
83	rmt_sig_in0	0	no	ledc_ls_sig_out4	1'd1
84	rmt_sig_in1	0	no	ledc_ls_sig_out5	1'd1
85	rmt_sig_in2	0	no	ledc_ls_sig_out6	1'd1
86	rmt_sig_in3	0	no	ledc_ls_sig_out7	1'd1
87	-	-	-	rmt_sig_out0	1'd1
88	-	-	-	rmt_sig_out1	1'd1
89	-	-	-	rmt_sig_out2	1'd1
90	-	-	-	rmt_sig_out3	1'd1
95	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe
96	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe
100	-	-	-	gpio_sd0_out	1'd1
101	-	-	-	gpio_sd1_out	1'd1
102	-	-	-	gpio_sd2_out	1'd1
103	-	-	-	gpio_sd3_out	1'd1
104	-	-	-	gpio_sd4_out	1'd1
105	-	-	-	gpio_sd5_out	1'd1
106	-	-	-	gpio_sd6_out	1'd1
107	-	-	-	gpio_sd7_out	1'd1
108	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe
109	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe
110	FSPID_in	0	yes	FSPID_out	FSPID_oe
111	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe
112	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe
113	FSPIIO4_in	0	yes	FSPIIO4_out	FSPIIO4_oe
114	FSPIIO5_in	0	yes	FSPIIO5_out	FSPIIO5_oe
115	FSPIIO6_in	0	yes	FSPIIO6_out	FSPIIO6_oe
116	FSPIIO7_in	0	yes	FSPIIO7_out	FSPIIO7_oe

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
117	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe
118	-	-	-	FSPICS1_out	FSPICS1_oe
119	-	-	-	FSPICS2_out	FSPICS2_oe
120	-	-	-	FSPICS3_out	FSPICS3_oe
121	-	-	-	FSPICS4_out	FSPICS4_oe
122	-	-	-	FSPICS5_out	FSPICS5_oe
123	can_rx	1	no	can_tx	1'd1
124	-	-	-	can_bus_off_on	1'd1
125	-	-	-	can_clkout	1'd1
126	-	-	-	SUBSPICLK_out_mux	SUBSPICLK_oe
127	SUBSPIQ_in	0	yes	SUBSPIQ_out	SUBSPIQ_oe
128	SUBSPID_in	0	yes	SUBSPID_out	SUBSPID_oe
129	SUBSPIHD_in	0	yes	SUBSPIHD_out	SUBSPIHD_oe
130	SUBSPIWP_in	0	yes	SUBSPIWP_out	SUBSPIWP_oe
131	-	-	-	SUBSPICS0_out	SUBSPICS0_oe
132	-	-	-	SUBSPICS1_out	SUBSPICS1_oe
133	-	-	-	FSPIDQS_out	FSPIDQS_oe
134	-	-	-	FSPI_HSYNC_out	FSPI_HSYNC_oe
135	-	-	-	FSPI_VSYNC_out	FSPI_VSYNC_oe
136	-	-	-	FSPI_DE_out	FSPI_DE_oe
137	-	-	-	FSPICD_out	FSPICD_oe
139	-	-	-	SPI3_CD_out	SPI3_CD_oe
140	-	-	-	SPI3_DQS_out	SPI3_DQS_oe
143	I2S0I_DATA_in0	0	no	I2S0O_DATA_out0	1'd1
144	I2S0I_DATA_in1	0	no	I2S0O_DATA_out1	1'd1
145	I2S0I_DATA_in2	0	no	I2S0O_DATA_out2	1'd1
146	I2S0I_DATA_in3	0	no	I2S0O_DATA_out3	1'd1
147	I2S0I_DATA_in4	0	no	I2S0O_DATA_out4	1'd1
148	I2S0I_DATA_in5	0	no	I2S0O_DATA_out5	1'd1
149	I2S0I_DATA_in6	0	no	I2S0O_DATA_out6	1'd1
150	I2S0I_DATA_in7	0	no	I2S0O_DATA_out7	1'd1
151	I2S0I_DATA_in8	0	no	I2S0O_DATA_out8	1'd1
152	I2S0I_DATA_in9	0	no	I2S0O_DATA_out9	1'd1
153	I2S0I_DATA_in10	0	no	I2S0O_DATA_out10	1'd1
154	I2S0I_DATA_in11	0	no	I2S0O_DATA_out11	1'd1
155	I2S0I_DATA_in12	0	no	I2S0O_DATA_out12	1'd1
156	I2S0I_DATA_in13	0	no	I2S0O_DATA_out13	1'd1
157	I2S0I_DATA_in14	0	no	I2S0O_DATA_out14	1'd1
158	I2S0I_DATA_in15	0	no	I2S0O_DATA_out15	1'd1
159	-	-	-	I2S0O_DATA_out16	1'd1
160	-	-	-	I2S0O_DATA_out17	1'd1

Signal No.	Input signals	Default value if unassigned *	Same input signal from IO MUX core	Output signals	Output enable of output signals
161	-	-	-	I2S0O_DATA_out18	1'd1
162	-	-	-	I2S0O_DATA_out19	1'd1
163	-	-	-	I2S0O_DATA_out20	1'd1
164	-	-	-	I2S0O_DATA_out21	1'd1
165	-	-	-	I2S0O_DATA_out22	1'd1
166	-	-	-	I2S0O_DATA_out23	1'd1
167	SUBSPID4_in	0	yes	SUBSPID4_out	SUBSPID4_oe
168	SUBSPID5_in	0	yes	SUBSPID5_out	SUBSPID5_oe
169	SUBSPID6_in	0	yes	SUBSPID6_out	SUBSPID6_oe
170	SUBSPID7_in	0	yes	SUBSPID7_out	SUBSPID7_oe
171	SUBSPIDQS_in	0	yes	SUBSPIDQS_out	SUBSPIDQS_oe
193	I2S0I_H_SYNC	0	no	-	1'd1
194	I2S0I_V_SYNC	0	no	-	1'd1
195	I2S0I_H_ENABLE	0	no	-	1'd1
215	-	-	-	ant_sel0	1'd1
216	-	-	-	ant_sel1	1'd1
217	-	-	-	ant_sel2	1'd1
218	-	-	-	ant_sel3	1'd1
219	-	-	-	ant_sel4	1'd1
220	-	-	-	ant_sel5	1'd1
221	-	-	-	ant_sel6	1'd1
222	-	-	-	ant_sel7	1'd1
223	sig_in_func_223	0	no	sig_in_func223	1'd1
224	sig_in_func_224	0	no	sig_in_func224	1'd1
225	sig_in_func_225	0	no	sig_in_func225	1'd1
226	sig_in_func_226	0	no	sig_in_func226	1'd1
227	sig_in_func_227	0	no	sig_in_func227	1'd1
235	pro_alonegpio_in0	0	no	pro_alonegpio_out0	1'd1
236	pro_alonegpio_in1	0	no	pro_alonegpio_out1	1'd1
237	pro_alonegpio_in2	0	no	pro_alonegpio_out2	1'd1
238	pro_alonegpio_in3	0	no	pro_alonegpio_out3	1'd1
239	pro_alonegpio_in4	0	no	pro_alonegpio_out4	1'd1
240	pro_alonegpio_in5	0	no	pro_alonegpio_out5	1'd1
241	pro_alonegpio_in6	0	no	pro_alonegpio_out6	1'd1
242	pro_alonegpio_in7	0	no	pro_alonegpio_out7	1'd1
251	-	-	-	clk_i2s_mux	1'd1

5.11 IO MUX Pad List

Table 23 shows the IO MUX functions of each I/O pad:

Table 23: IO MUX Pad List

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
0	GPIO0	GPIO0	GPIO0	-	-	-	3	R
1	GPIO1	GPIO1	GPIO1	-	-	-	1	R
2	GPIO2	GPIO2	GPIO2	-	-	-	1	R
3	GPIO3	GPIO3	GPIO3	-	-	-	0	R
4	GPIO4	GPIO4	GPIO4	-	-	-	0	R
5	GPIO5	GPIO5	GPIO5	-	-	-	0	R
6	GPIO6	GPIO6	GPIO6	-	-	-	0	R
7	GPIO7	GPIO7	GPIO7	-	-	-	0	R
8	GPIO8	GPIO8	GPIO8	-	SUBSPICS1	-	0	R
9	GPIO9	GPIO9	GPIO9	-	SUBSPIHD	FSPIHD	1	R
10	GPIO10	GPIO10	GPIO10	FSPIIO4	SUBSPICS0	FSPICS0	1	R
11	GPIO11	GPIO11	GPIO11	FSPIIO5	SUBSPID	FSPID	1	R
12	GPIO12	GPIO12	GPIO12	FSPIIO6	SUBSPICLK	FSPICLK	1	R
13	GPIO13	GPIO13	GPIO13	FSPIIO7	SUBSPIQ	FSPIQ	1	R
14	GPIO14	GPIO14	GPIO14	FSPIDQS	SUBSPIWP	FSPIWP	1	R
15	XTAL_32K_P	XTAL_32K_P	GPIO15	U0RTS	-	-	0	R
16	XTAL_32K_N	XTAL_32K_N	GPIO16	U0CTS	-	-	0	R
17	DAC_1	DAC_1	GPIO17	U1TXD	-	-	1	R
18	DAC_2	DAC_2	GPIO18	U1RXD	CLK_OUT3	-	1	R
19	GPIO19	GPIO19	GPIO19	U1RTS	CLK_OUT2	-	0	R
20	GPIO20	GPIO20	GPIO20	U1CTS	CLK_OUT1	-	0	R
21	GPIO21	GPIO21	GPIO21	-	-	-	0	R
26	SPICS1	SPICS1	GPIO26	-	-	-	3	-
27	SPIHD	SPIHD	GPIO27	-	-	-	3	-
28	SPIWP	SPIWP	GPIO28	-	-	-	3	-
29	SPICS0	SPICS0	GPIO29	-	-	-	3	-
30	SPICLK	SPICLK	GPIO30	-	-	-	3	-
31	SPIQ	SPIQ	GPIO31	-	-	-	3	-
32	SPID	SPID	GPIO32	-		-	3	-
33	GPIO33	GPIO33	GPIO33	FSPIHD	SUBSPIHD	SPIIO4	1	-
34	GPIO34	GPIO34	GPIO34	FSPICS0	SUBSPICS0	SPIIO5	1	-
35	GPIO35	GPIO35	GPIO35	FSPID	SUBSPID	SPIIO6	1	-
36	GPIO36	GPIO36	GPIO36	FSPICLK	SUBSPICLK	SPIIO7	1	-
37	GPIO37	GPIO37	GPIO37	FSPIQ	SUBSPIQ	SPIDQS	1	-
38	GPIO38	GPIO38	GPIO38	FSPIWP	SUBSPIWP	-	1	-
39	MTCK	MTCK	GPIO39	CLK_OUT3	SUBSPICS1	-	1	-
40	MTDO	MTDO	GPIO40	CLK_OUT2	-	-	1	-
41	MTDI	MTDI	GPIO41	CLK_OUT1	-	-	1	-
42	MTMS	MTMS	GPIO42	-	-	-	1	-
43	U0TXD	U0TXD	GPIO43	CLK_OUT1	-	-	3	-
44	U0RXD	U0RXD	GPIO44	CLK_OUT2	-	-	3	-
45	GPIO45	GPIO45	GPIO45	-	-	-	2	-

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
46	GPIO46	GPIO46	GPIO46	-	-	-	2	I

Reset Configurations

"Reset" column shows the default configuration of each pad 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)

Note:

- **R** - Pad has RTC/analog functions via RTC IO MUX.
- **I** - Pad can only be configured as input GPIO.

Please refer to Appendix A – ESP32-S2 Pin Lists in *ESP32-S2 Datasheet* for more details.

5.12 RTC IO MUX Pin List

Table 24 shows the RTC pins and how they correspond to GPIO pads.

Table 24: RTC IO MUX Pin Summary

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			
			1	2	3	4
0	0	TOUCH_PAD0*	RTC_GPIO0	-	-	sar_i2c_scl_0
1	1	TOUCH_PAD1	RTC_GPIO1	-	-	sar_i2c_sda_0
2	2	TOUCH_PAD2	RTC_GPIO2	-	-	sar_i2c_scl_1
3	3	TOUCH_PAD3	RTC_GPIO3	-	-	sar_i2c_sda_1
4	4	TOUCH_PAD4	RTC_GPIO4	-	-	-
5	5	TOUCH_PAD5	RTC_GPIO5	-	-	-
6	6	TOUCH_PAD6	RTC_GPIO6	-	-	-
7	7	TOUCH_PAD7	RTC_GPIO7	-	-	-
8	8	TOUCH_PAD8	RTC_GPIO8	-	-	-
9	9	TOUCH_PAD9	RTC_GPIO9	-	-	-
10	10	TOUCH_PAD10	RTC_GPIO10	-	-	-
11	11	TOUCH_PAD11	RTC_GPIO11	-	-	-
12	12	TOUCH_PAD12	RTC_GPIO12	-	-	-
13	13	TOUCH_PAD13	RTC_GPIO13	-	-	-
14	14	TOUCH_PAD14	RTC_GPIO14	-	-	-
15	15	X32P	RTC_GPIO15	-	-	-
16	16	X32N	RTC_GPIO16	-	-	-
17	17	PDAC1	RTC_GPIO17	-	-	-
18	18	PDAC2	RTC_GPIO18	-	-	-
19	19	RTC_PAD19	RTC_GPIO19	-	-	-

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			
			1	2	3	4
20	20	RTC_PAD20	RTC_GPIO20	-	-	-
21	21	RTC_PAD21	RTC_GPIO21	-	-	-

Note: TOUCH_PAD0 is an internal channel and its analog functions are not lead to a corresponding external GPIO.

5.13 Base Address

Users can access the modules described in this chapter via the base addresses shown in the following table. For more information about accessing peripherals from different buses, please see Chapter 1 *System and Memory*.

Table 25: Module Base Addresses

Module Name	Access to Access Peripheral	Base Address
GPIO	PeriBUS1	0x3F404000
	PeriBUS2	0x60004000
IO MUX	PeriBUS1	0x3F409000
	PeriBUS2	0x60009000
GPIOSD	PeriBUS2	0x60004F00
Deicated GPIO	PeriBUS1	0x3F4CF000
RTCIO	PeriBUS1	0x3F408400
	PeriBUS2	0x60008400

5.14 Register Summary

The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the base address, please refer to Section 5.13.

5.14.1 GPIO Matrix Register Summary

Name	Description	Address	Access
GPIO Configuration Registers			
GPIO_BT_SELECT_REG	GPIO bit selection register	0x0000	R/W
GPIO_OUT_REG	GPIO0 ~ 31 output register	0x0004	R/W
GPIO_OUT_W1TS_REG	GPIO0 ~ 31 output bit set register	0x0008	WO
GPIO_OUT_W1TC_REG	GPIO0 ~ 31 output bit clear register	0x000C	WO
GPIO_OUT1_REG	GPIO32 ~ 53 output register	0x0010	R/W
GPIO_OUT1_W1TS_REG	GPIO32 ~ 53 output bit set register	0x0014	WO
GPIO_OUT1_W1TC_REG	GPIO32 ~ 53 output bit clear register	0x0018	WO
GPIO_SDIO_SELECT_REG	GPIO SDIO selection register	0x001C	R/W
GPIO_ENABLE_REG	GPIO0 ~ 31 output enable register	0x0020	R/W
GPIO_ENABLE_W1TS_REG	GPIO0 ~ 31 output enable bit set register	0x0024	WO
GPIO_ENABLE_W1TC_REG	GPIO0 ~ 31 output enable bit clear register	0x0028	WO

Name	Description	Address	Access
GPIO_ENABLE1_REG	GPIO32 ~ 53 output enable register	0x002C	R/W
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ 53 output enable bit set register	0x0030	WO
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ 53 output enable bit clear register	0x0034	WO
GPIO_STRAP_REG	Bootstrap pin value register	0x0038	RO
GPIO_IN_REG	GPIO0 ~ 31 input register	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ 53 input register	0x0040	RO
GPIO_PIN0_REG	Configuration for GPIO pin 0	0x0074	R/W
GPIO_PIN1_REG	Configuration for GPIO pin 1	0x0078	R/W
GPIO_PIN2_REG	Configuration for GPIO pin 2	0x007C	R/W
...	...	...	...
GPIO_PIN51_REG	Configuration for GPIO pin 51	0x0140	R/W
GPIO_PIN52_REG	Configuration for GPIO pin 52	0x0144	R/W
GPIO_PIN53_REG	Configuration for GPIO pin 53	0x0148	R/W
GPIO_FUNC0_IN_SEL_CFG_REG	Peripheral function 0 input selection register	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Peripheral function 1 input selection register	0x0158	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	Peripheral function 2 input selection register	0x015C	R/W
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	Peripheral function 253 input selection register	0x0548	R/W
GPIO_FUNC254_IN_SEL_CFG_REG	Peripheral function 254 input selection register	0x054C	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Peripheral function 255 input selection register	0x0550	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	Peripheral output selection for GPIO0	0x0554	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	Peripheral output selection for GPIO1	0x0558	R/W
GPIO_FUNC2_OUT_SEL_CFG_REG	Peripheral output selection for GPIO2	0x055C	R/W
..	...	...	...
GPIO_FUNC51_OUT_SEL_CFG_REG	Peripheral output selection for GPIO51	0x0620	R/W
GPIO_FUNC52_OUT_SEL_CFG_REG	Peripheral output selection for GPIO52	0x0624	R/W
GPIO_FUNC53_OUT_SEL_CFG_REG	Peripheral output selection for GPIO53	0x0628	R/W
GPIO_CLOCK_GATE_REG	GPIO clock gating register	0x062C	R/W
Interrupt Configuration Registers			
GPIO_STATUS_W1TS_REG	GPIO0 ~ 31 interrupt status bit set register	0x0048	WO
GPIO_STATUS_W1TC_REG	GPIO0 ~ 31 interrupt status bit clear register	0x004C	WO
GPIO_STATUS1_W1TS_REG	GPIO32 ~ 53 interrupt status bit set register	0x0054	WO
GPIO_STATUS1_W1TC_REG	GPIO32 ~ 53 interrupt status bit clear register	0x0058	WO
GPIO Interrupt Source Registers			
GPIO_STATUS_NEXT_REG	GPIO0 ~ 31 interrupt source register	0x014C	RO
GPIO_STATUS_NEXT1_REG	GPIO32 ~ 53 interrupt source register	0x0150	RO
Interrupt Status Registers			
GPIO_STATUS_REG	GPIO0 ~ 31 interrupt status register	0x0044	R/W
GPIO_STATUS1_REG	GPIO32 ~ 53 interrupt status register	0x0050	R/W
GPIO_PCPU_INT_REG	GPIO0 ~ 31 PRO_CPU interrupt status register	0x005C	RO
GPIO_PCPU_NMI_INT_REG	GPIO0 ~ 31 PRO_CPU non-maskable interrupt status register	0x0060	RO
GPIO_PCPU_INT1_REG	GPIO32 ~ 53 PRO_CPU interrupt status register	0x0068	RO

Name	Description	Address	Access
GPIO_PCPU_NMI_INT1_REG	GPIO32 ~ 53 PRO_CPU non-maskable interrupt status register	0x006C	RO

5.14.2 IO MUX Register Summary

Name	Description	Address	Access
IO_MUX_PIN_CTRL	Clock output configuration register	0x0000	R/W
IO_MUX_GPIO0_REG	Configuration register for pad GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	Configuration register for pad GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	Configuration register for pad GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	Configuration register for pad GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	Configuration register for pad GPIO4	0x0014	R/W
IO_MUX_GPIO5_REG	Configuration register for pad GPIO5	0x0018	R/W
IO_MUX_GPIO6_REG	Configuration register for pad GPIO6	0x001C	R/W
IO_MUX_GPIO7_REG	Configuration register for pad GPIO7	0x0020	R/W
IO_MUX_GPIO8_REG	Configuration register for pad GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	Configuration register for pad GPIO9	0x0028	R/W
IO_MUX_GPIO10_REG	Configuration register for pad GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	Configuration register for pad GPIO11	0x0030	R/W
IO_MUX_GPIO12_REG	Configuration register for pad GPIO12	0x0034	R/W
IO_MUX_GPIO13_REG	Configuration register for pad GPIO13	0x0038	R/W
IO_MUX_GPIO14_REG	Configuration register for pad GPIO14	0x003C	R/W
IO_MUX_XTAL_32K_P_REG	Configuration register for pad XTAL_32K_P	0x0040	R/W
IO_MUX_XTAL_32K_N_REG	Configuration register for pad XTAL_32K_N	0x0044	R/W
IO_MUX_DAC_1_REG	Configuration register for pad DAC_1	0x0048	R/W
IO_MUX_DAC_2_REG	Configuration register for pad DAC_2	0x004C	R/W
IO_MUX_GPIO_19_REG	Configuration register for pad GPIO19	0x0050	R/W
IO_MUX_GPIO_20_REG	Configuration register for pad GPIO20	0x0054	R/W
IO_MUX_GPIO_21_REG	Configuration register for pad GPIO21	0x0058	R/W
IO_MUX_SPICS1_REG	Configuration register for pad SPICS1	0x006C	R/W
IO_MUX_SPIHD_REG	Configuration register for pad SPIHD	0x0070	R/W
IO_MUX_SPIWP_REG	Configuration register for pad SPIWP	0x0074	R/W
IO_MUX_SPICS0_REG	Configuration register for pad SPICS0	0x0078	R/W
IO_MUX_SPICLK_REG	Configuration register for pad SPICLK	0x007C	R/W
IO_MUX_SPIQ_REG	Configuration register for pad SPIQ	0x0080	R/W
IO_MUX_SPID_REG	Configuration register for pad SPID	0x0084	R/W
IO_MUX_GPIO_33_REG	Configuration register for pad GPIO33	0x0088	R/W
IO_MUX_GPIO_34_REG	Configuration register for pad GPIO34	0x008C	R/W
IO_MUX_GPIO_35_REG	Configuration register for pad GPIO35	0x0090	R/W
IO_MUX_GPIO_36_REG	Configuration register for pad GPIO36	0x0094	R/W
IO_MUX_GPIO_37_REG	Configuration register for pad GPIO37	0x0098	R/W
IO_MUX_GPIO_38_REG	Configuration register for pad GPIO38	0x009C	R/W
IO_MUX_MTCK_REG	Configuration register for pad MTCK	0x00A0	R/W
IO_MUX_MTDO_REG	Configuration register for pad MTDO	0x00A4	R/W

Name	Description	Address	Access
IO_MUX_MTDI_REG	Configuration register for pad MTDI	0x00A8	R/W
IO_MUX_MTMS_REG	Configuration register for pad MTMS	0x00AC	R/W
IO_MUX_U0TXD_REG	Configuration register for pad U0TXD	0x00B0	R/W
IO_MUX_U0RXD_REG	Configuration register for pad U0RXD	0x00B4	R/W
IO_MUX_GPIO_45_REG	Configuration register for pad GPIO45	0x00B8	R/W
IO_MUX_GPIO_46_REG	Configuration register for pad GPIO46	0x00BC	R/W

5.14.3 Sigma Delta Modulated Output Register Summary

Name	Description	Address	Access
Configuration Registers			
GPIOSD_SIGMADELTA0_REG	Duty Cycle Configure Register of SDM0	0x0000	R/W
GPIOSD_SIGMADELTA1_REG	Duty Cycle Configure Register of SDM1	0x0004	R/W
GPIOSD_SIGMADELTA2_REG	Duty Cycle Configure Register of SDM2	0x0008	R/W
GPIOSD_SIGMADELTA3_REG	Duty Cycle Configure Register of SDM3	0x000C	R/W
GPIOSD_SIGMADELTA4_REG	Duty Cycle Configure Register of SDM4	0x0010	R/W
GPIOSD_SIGMADELTA5_REG	Duty Cycle Configure Register of SDM5	0x0014	R/W
GPIOSD_SIGMADELTA6_REG	Duty Cycle Configure Register of SDM6	0x0018	R/W
GPIOSD_SIGMADELTA7_REG	Duty Cycle Configure Register of SDM7	0x001C	R/W
GPIOSD_SIGMADELTA_CG_REG	Clock Gating Configure Register	0x0020	R/W
GPIOSD_SIGMADELTA_MISC_REG	MISC Register	0x0024	R/W
GPIOSD_SIGMADELTA_VERSION_REG	Version Control Register	0x0028	R/W

5.14.4 Dedicated GPIO Register Summary

Name	Description	Address	Access
Configuration registers			
DEDIC_GPIO_OUT_DRT_REG	Dedicated GPIO direct output register	0x0000	WO
DEDIC_GPIO_OUT_MSK_REG	Dedicated GPIO mask output register	0x0004	WO
DEDIC_GPIO_OUT_IDV_REG	Dedicated GPIO individual output register	0x0008	WO
DEDIC_GPIO_OUT_CPU_REG	Dedicated GPIO output mode selection register	0x0010	R/W
DEDIC_GPIO_IN_DLY_REG	Dedicated GPIO input delay configuration register	0x0014	R/W
DEDIC_GPIO_INTR_RCGN_REG	Dedicated GPIO interrupts generation mode register	0x001C	R/W
Status registers			
DEDIC_GPIO_OUT_SCAN_REG	Dedicated GPIO output status register	0x000C	RO
DEDIC_GPIO_IN_SCAN_REG	Dedicated GPIO input status register	0x0018	RO
Interrupt registers			
DEDIC_GPIO_INTR_RAW_REG	Raw interrupt status	0x0020	RO
DEDIC_GPIO_INTR_RLS_REG	Interrupt enable bits	0x0024	R/W
DEDIC_GPIO_INTR_ST_REG	Masked interrupt status	0x0028	RO
DEDIC_GPIO_INTR_CLR_REG	Interrupt clear bits	0x002C	WO

5.14.5 RTC IO MUX Register Summary

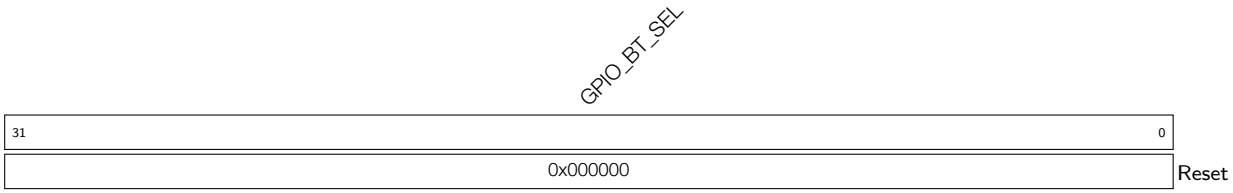
Name	Description	Address	Access
GPIO Configuration and Data Registers			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO output register	0x0000	R/W
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register	0x0004	WO
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register	0x0008	WO
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO output enable register	0x000C	R/W
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit set register	0x0010	WO
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register	0x0014	WO
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register	0x0018	R/W
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register	0x001C	WO
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register	0x0020	WO
RTCIO_RTC_GPIO_IN_REG	RTC GPIO input register	0x0024	RO
RTCIO_RTC_GPIO_PIN0_REG	RTC configuration for pin 0	0x0028	R/W
RTCIO_RTC_GPIO_PIN1_REG	RTC configuration for pin 1	0x002C	R/W
RTCIO_RTC_GPIO_PIN2_REG	RTC configuration for pin 2	0x0030	R/W
RTCIO_RTC_GPIO_PIN3_REG	RTC configuration for pin 3	0x0034	R/W
...	...	...	...
RTCIO_RTC_GPIO_PIN19_REG	RTC configuration for pin 19	0x0074	R/W
RTCIO_RTC_GPIO_PIN20_REG	RTC configuration for pin 20	0x0078	R/W
RTCIO_RTC_GPIO_PIN21_REG	RTC configuration for pin 21	0x007C	R/W
GPIO RTC Function Configuration Registers			
RTCIO_TOUCH_PAD0_REG	Touch pad 0 configuration register	0x0084	R/W
RTCIO_TOUCH_PAD1_REG	Touch pad 1 configuration register	0x0088	R/W
RTCIO_TOUCH_PAD2_REG	Touch pad 2 configuration register	0x008C	R/W
...	...	...	...
RTCIO_TOUCH_PAD13_REG	Touch pad 13 configuration register	0x00B8	R/W
RTCIO_TOUCH_PAD14_REG	Touch pad 14 configuration register	0x00BC	R/W
RTCIO_XTAL_32P_PAD_REG	32KHz crystal P-pad configuration register	0x00C0	R/W
RTCIO_XTAL_32N_PAD_REG	32KHz crystal N-pad configuration register	0x00C4	R/W
RTCIO_PAD_DAC1_REG	DAC1 configuration register	0x00C8	R/W
RTCIO_PAD_DAC2_REG	DAC2 configuration register	0x00CC	R/W
RTCIO_RTC_PAD19_REG	Touch pad 19 configuration register	0x00D0	R/W
RTCIO_RTC_PAD20_REG	Touch pad 20 configuration register	0x00D4	R/W
RTCIO_RTC_PAD21_REG	Touch pad 21 configuration register	0x00D8	R/W
RTCIO_XTL_EXT_CTR_REG	Crystal power down enable GPIO source	0x00E0	R/W
RTCIO_SAR_I2C_IO_REG	RTC I2C pad selection	0x00E4	R/W

5.15 Registers

The address in the following part represents the address offset (relative address) with respect to the peripheral base address, not the absolute address. For detailed information about the base address, please refer to Section 5.13.

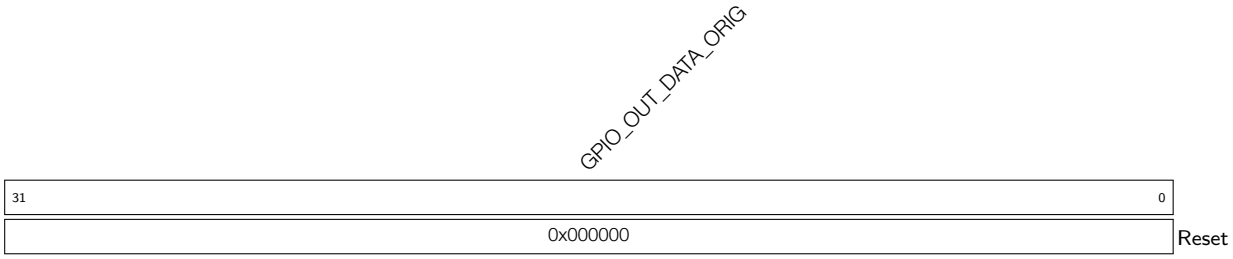
5.15.1 GPIO Matrix Registers

Register 5.1: GPIO_BT_SELECT_REG (0x0000)



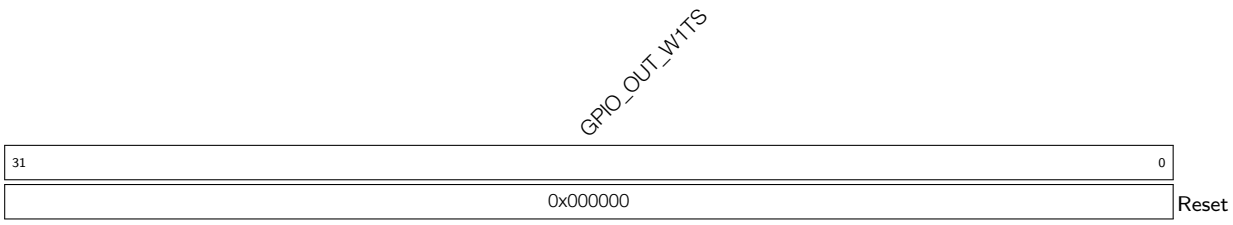
GPIO_BT_SEL Reserved (R/W)

Register 5.2: GPIO_OUT_REG (0x0004)



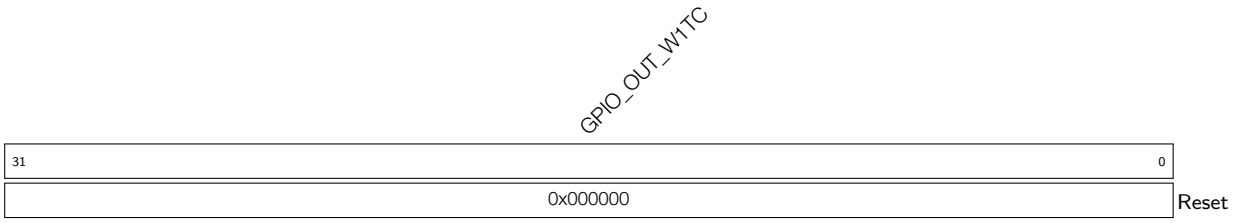
GPIO_OUT_DATA_ORIG GPIO0 ~ 31 output value in simple GPIO output mode. The values of bit0 ~ bit31 correspond to the output value of GPIO0 ~ GPIO31 respectively. Bit22 ~ bit25 are invalid. (R/W)

Register 5.3: GPIO_OUT_W1TS_REG (0x0008)



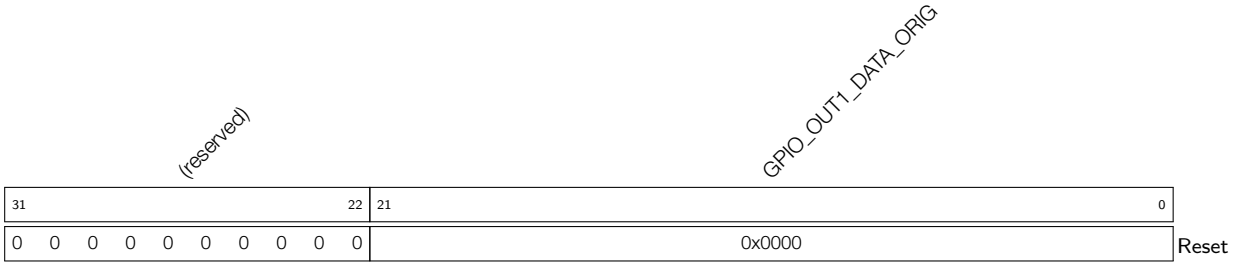
GPIO_OUT_W1TS GPIO0 ~ 31 output set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT_REG. (WO)

Register 5.4: GPIO_OUT_W1TC_REG (0x000C)



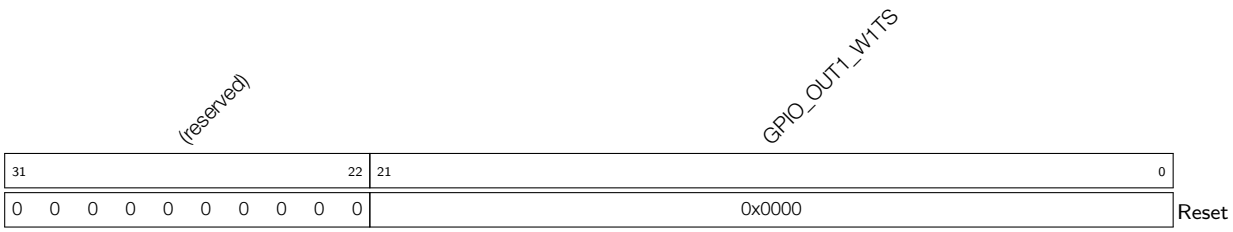
GPIO_OUT_W1TC GPIO0 ~ 31 output clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT_REG. (WO)

Register 5.5: GPIO_OUT1_REG (0x0010)



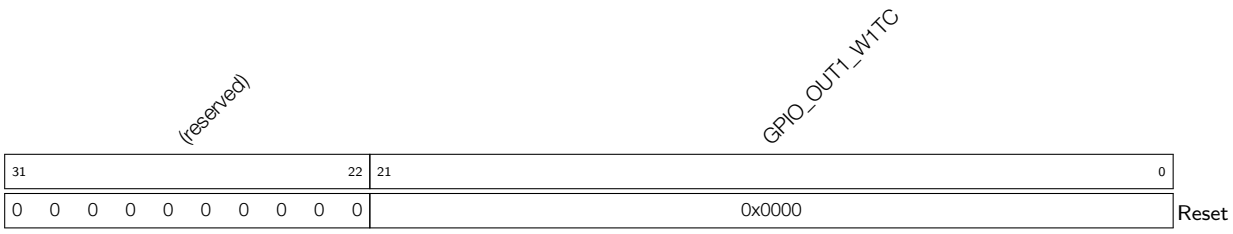
GPIO_OUT1_DATA_ORIG GPIO32 ~ 53 output value in simple GPIO output mode. The values of bit0 ~ bit13 correspond to GPIO32 ~ GPIO45. Bit14 ~ bit21 are invalid. (R/W)

Register 5.6: GPIO_OUT1_W1TS_REG (0x0014)



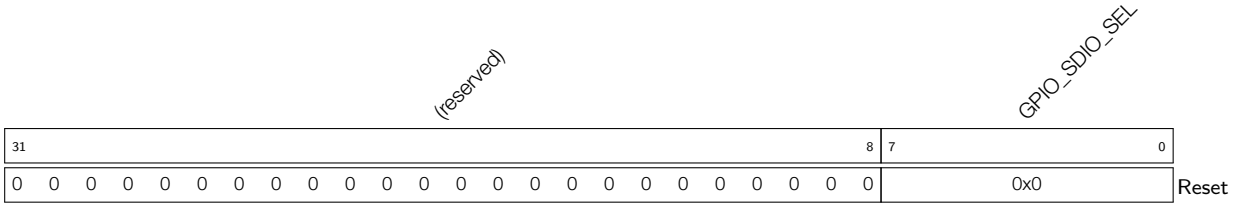
GPIO_OUT1_W1TS GPIO32 ~ 53 output value set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT1_REG. (WO)

Register 5.7: GPIO_OUT1_W1TC_REG (0x0018)



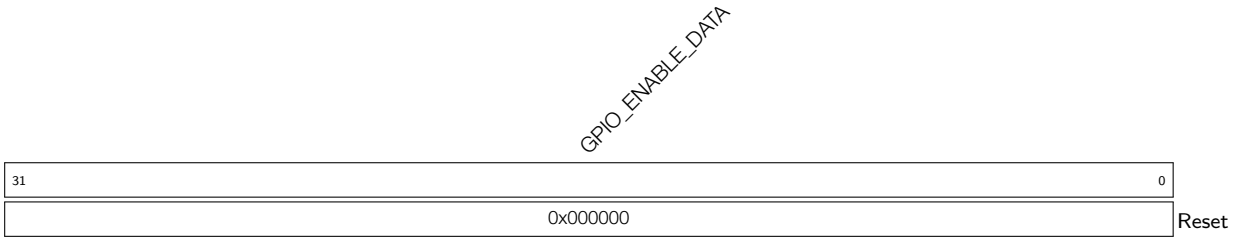
GPIO_OUT1_W1TC GPIO32 ~ 53 output value clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT1_REG. (WO)

Register 5.8: GPIO_SDIO_SELECT_REG (0x001C)



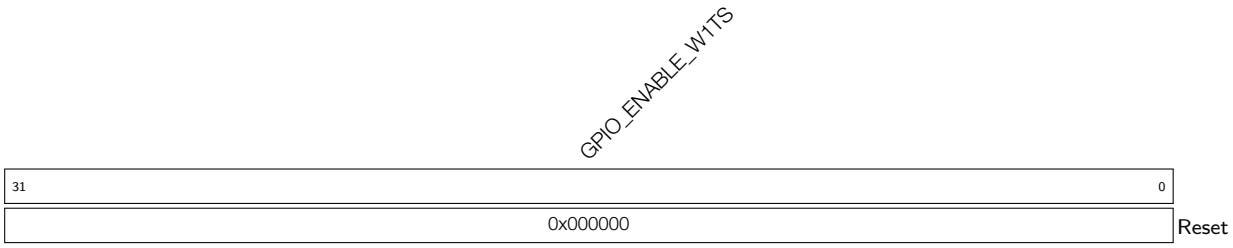
GPIO_SDIO_SEL Reserved (R/W)

Register 5.9: GPIO_ENABLE_REG (0x0020)



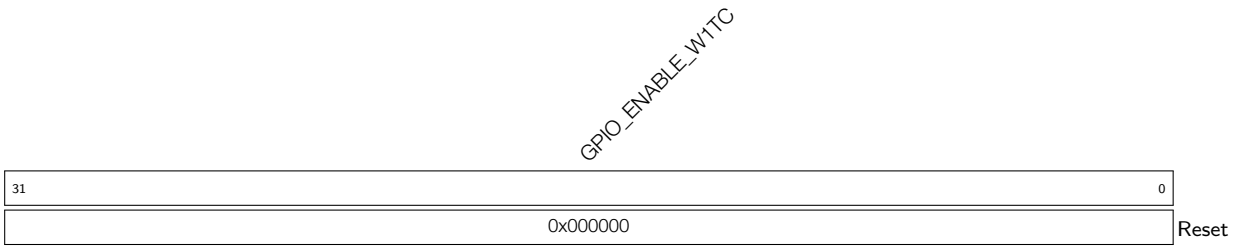
GPIO_ENABLE_DATA GPIO0 ~ 31 output enable register. (R/W)

Register 5.10: GPIO_ENABLE_W1TS_REG (0x0024)



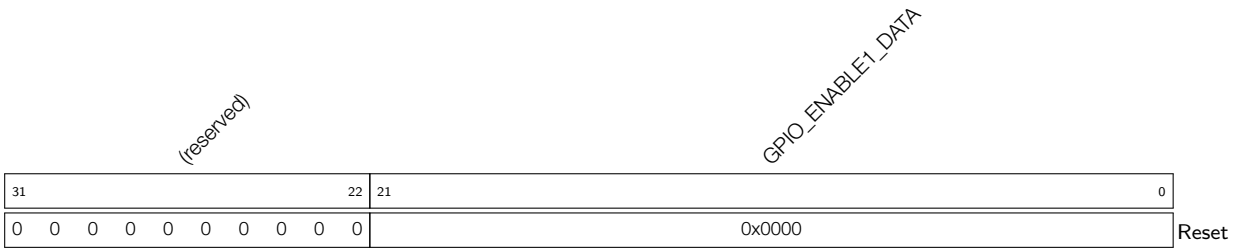
GPIO_ENABLE_W1TS GPIO0 ~ 31 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE_REG. (WO)

Register 5.11: GPIO_ENABLE_W1TC_REG (0x0028)



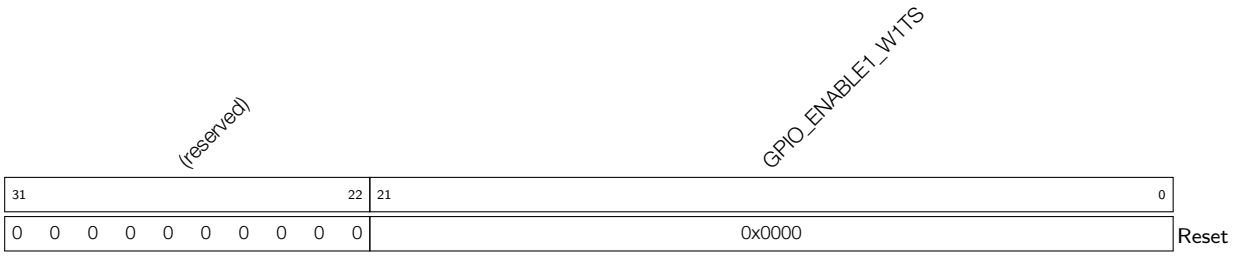
GPIO_ENABLE_W1TC GPIO0 ~ 31 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE_REG. (WO)

Register 5.12: GPIO_ENABLE1_REG (0x002C)



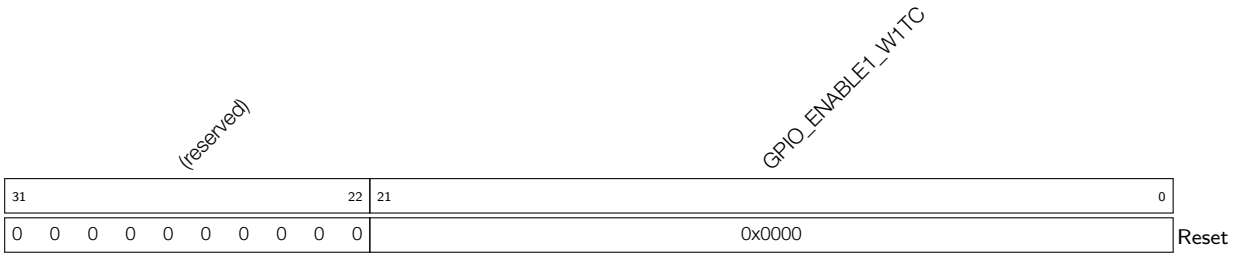
GPIO_ENABLE1_DATA GPIO32 ~ 53 output enable register. (R/W)

Register 5.13: GPIO_ENABLE1_W1TS_REG (0x0030)



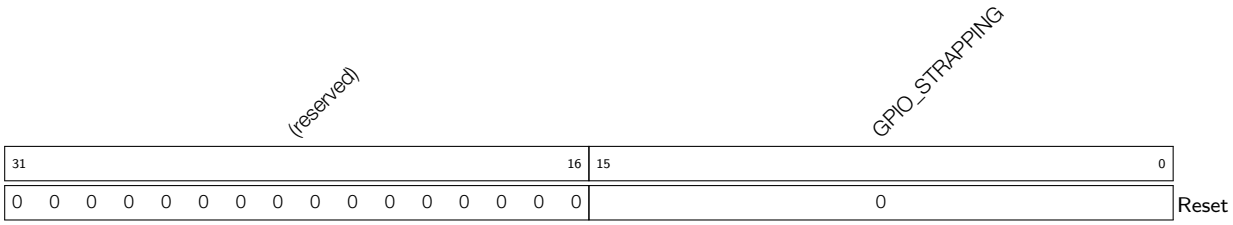
GPIO_ENABLE1_W1TS GPIO32 ~ 53 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE1_REG. (WO)

Register 5.14: GPIO_ENABLE1_W1TC_REG (0x0034)



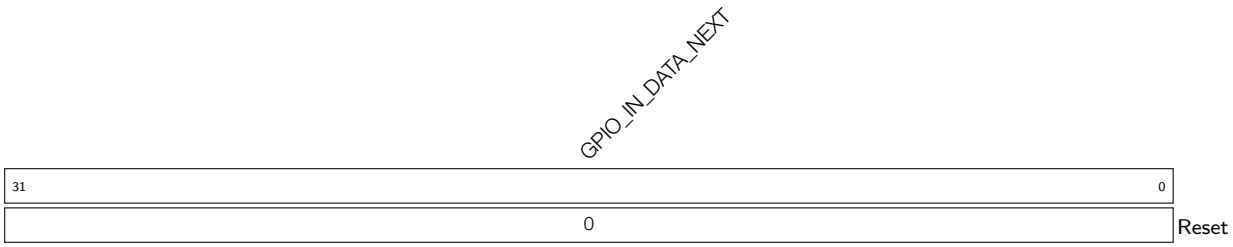
GPIO_ENABLE1_W1TC GPIO32 ~ 53 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE1_REG. (WO)

Register 5.15: GPIO_STRAP_REG (0x0038)



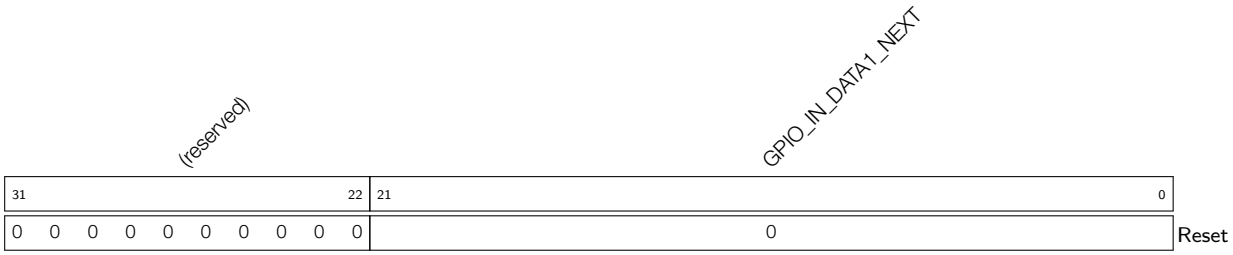
GPIO_STRAPPING GPIO strapping values: bit4 ~ bit2 correspond to stripping pins GPIO45, GPIO0, and GPIO46 respectively. (RO)

Register 5.16: GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT GPIO0 ~ 31 input value. Each bit represents a pad input value, 1 for high level and 0 for low level. (RO)

Register 5.17: GPIO_IN1_REG (0x0040)



GPIO_IN_DATA1_NEXT GPIO32 ~ 53 input value. Each bit represents a pad input value. (RO)

Register 5.18: GPIO_PIN*n*_REG (*n*: 0-53) (0x0074+4n*)**

GPIO_PIN*n*_SYNC2_BYPASS For the second stage 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*_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain output. (R/W)

GPIO_PIN*n*_SYNC1_BYPASS For the first stage 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)

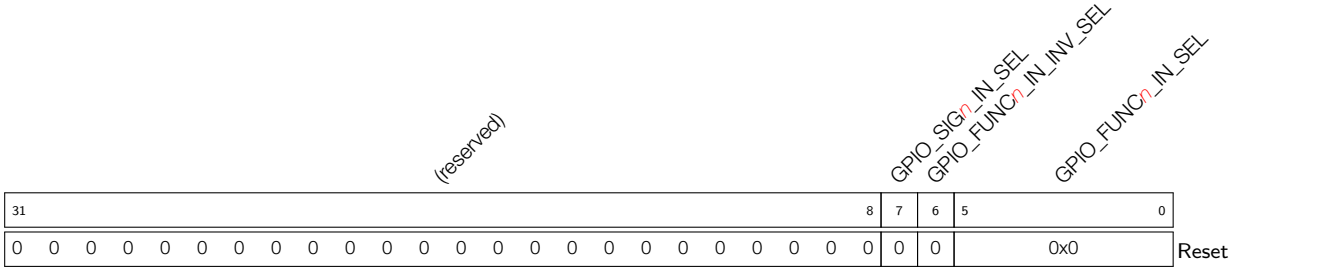
GPIO_PIN*n*_INT_TYPE Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable bit, only wakes up the CPU from Light-sleep (R/W)

GPIO_PIN*n*_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 5.19: GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-255) (0x0154+4n*)**

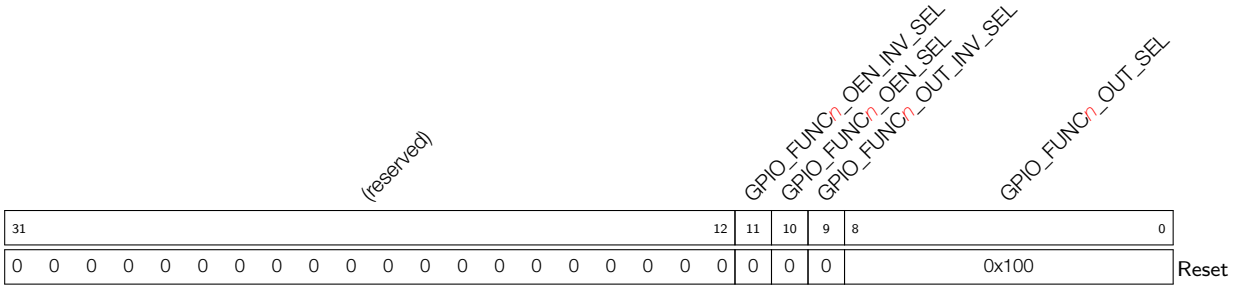


GPIO_FUNC*n*_IN_SEL Selection control for peripheral input signal *m*, selects a pad from the 54 GPIO matrix pads to connect this input signal. Or selects 0x38 for a constantly high input or 0x3C for a constantly low input. (R/W)

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 5.20: GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-53) (0x0554+4n*)**



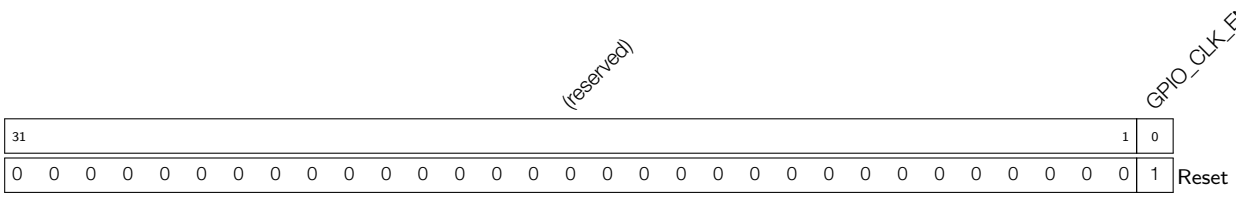
GPIO_FUNC*n*_OUT_SEL Selection control for GPIO output *n*. If a value *s* (0<=*s*<256) is written to this field, the peripheral output signal *s* will be connected to GPIO output *n*. If a value 256 is written to this field, bit *n* of GPIO_OUT_REG/GPIO_OUT1_REG and GPIO_ENABLE_REG/GPIO_ENABLE1_REG 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)

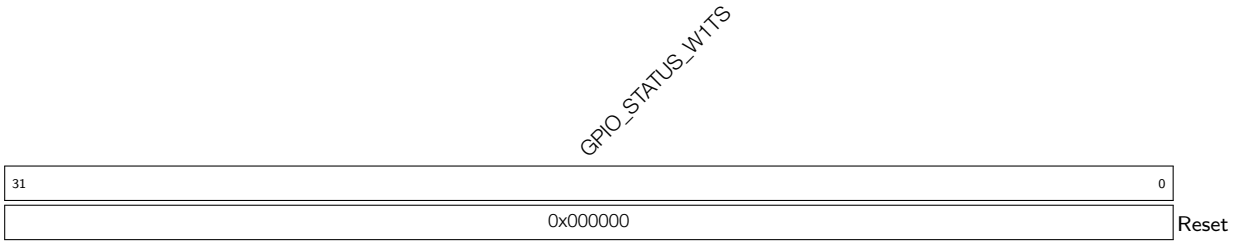
GPIO_FUNC*n*_OEN_INV_SEL 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

Register 5.21: GPIO_CLOCK_GATE_REG (0x062C)



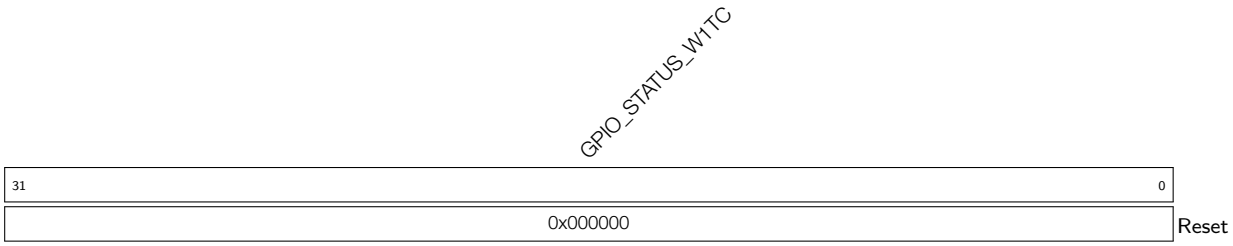
GPIO_CLK_EN Clock gating enable bit. If set to 1, the clock is free running. (R/W)

Register 5.22: GPIO_STATUS_W1TS_REG (0x0048)



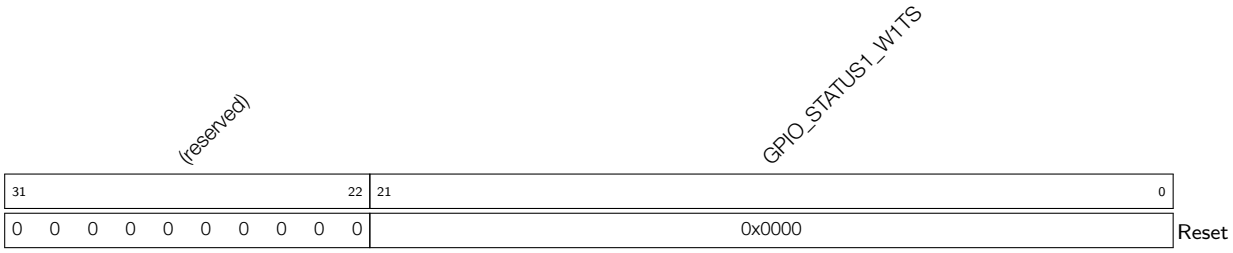
GPIO_STATUS_W1TS GPIO0 ~ 31 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1. Recommended operation: use this register to set GPIO_STATUS_INTERRUPT. (WO)

Register 5.23: GPIO_STATUS_W1TC_REG (0x004C)



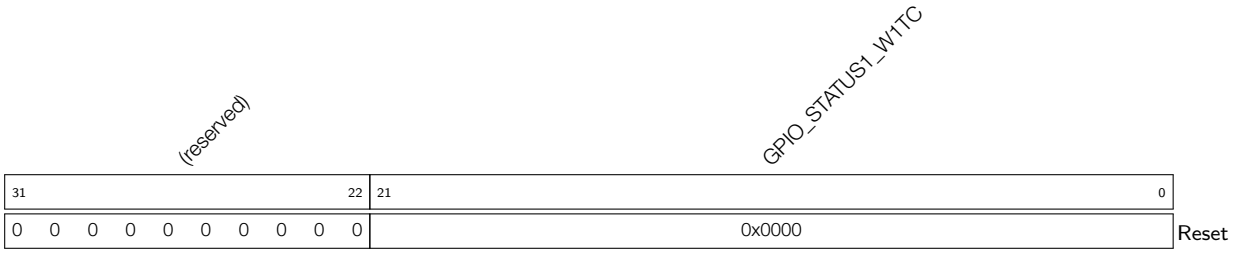
GPIO_STATUS_W1TC GPIO0 ~ 31 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT. (WO)

Register 5.24: GPIO_STATUS1_W1TS_REG (0x0054)



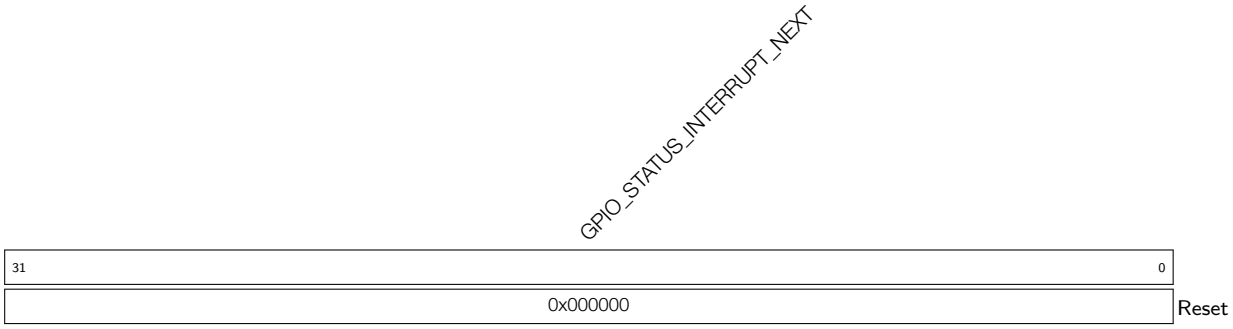
GPIO_STATUS1_W1TS GPIO32 ~ 53 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be set to 1. Recommended operation: use this register to set GPIO_STATUS1_REG. (WO)

Register 5.25: GPIO_STATUS1_W1TC_REG (0x0058)



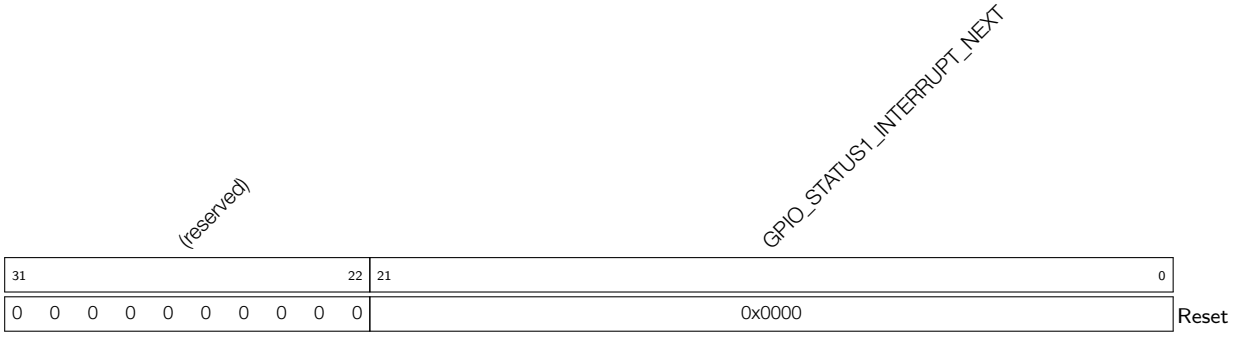
GPIO_STATUS1_W1TC GPIO32 ~ 53 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be cleared. Recommended operation: use this register to clear GPIO_STATUS1_REG. (WO)

Register 5.26: GPIO_STATUS_NEXT_REG (0x014C)



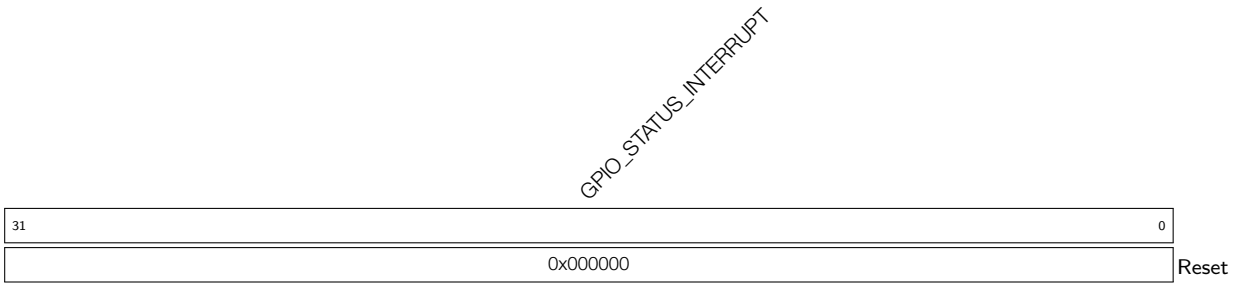
GPIO_STATUS_INTERRUPT_NEXT Interrupt source signal of GPIO0 ~ 31, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. (RO)

Register 5.27: GPIO_STATUS_NEXT1_REG (0x0150)



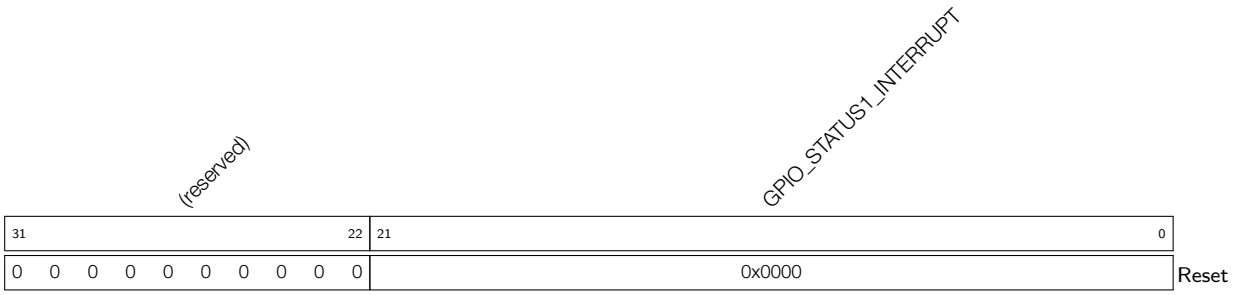
GPIO_STATUS1_INTERRUPT_NEXT Interrupt source signal of GPIO32 ~ 53. (RO)

Register 5.28: GPIO_STATUS_REG (0x0044)



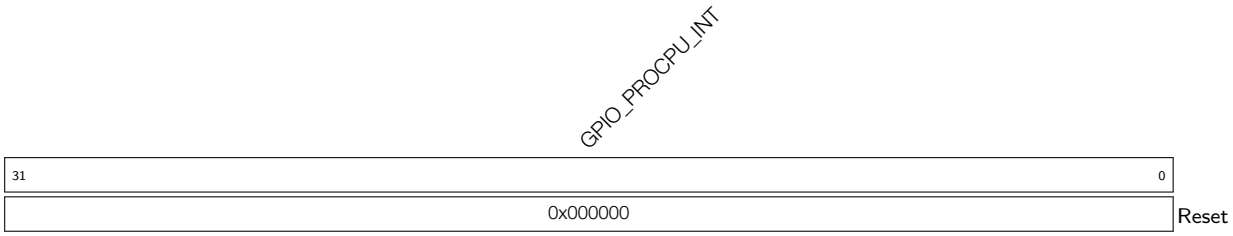
GPIO_STATUS_INTERRUPT GPIO0 ~ 31 interrupt status register. (R/W)

Register 5.29: GPIO_STATUS1_REG (0x0050)



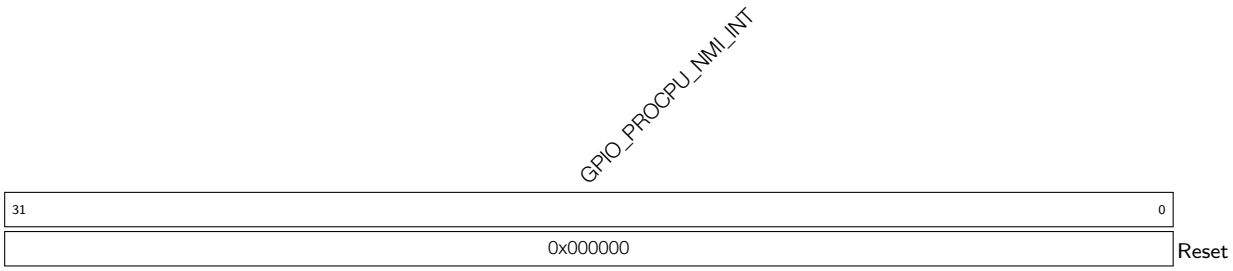
GPIO_STATUS1_INTERRUPT GPIO32 ~ 53 interrupt status register. (R/W)

Register 5.30: GPIO_PCPU_INT_REG (0x005C)



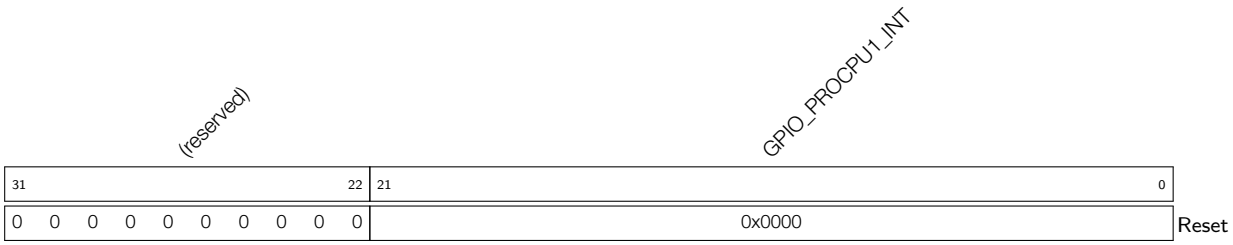
GPIO_PROCPU_INT GPIO0 ~ 31 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 5.31: GPIO_PCPU_NMI_INT_REG (0x0060)



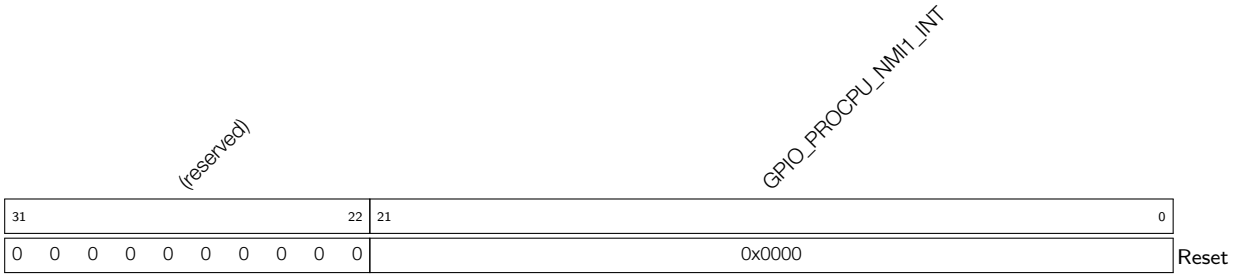
GPIO_PROCPU_NMI_INT GPIO0 ~ 31 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

Register 5.32: GPIO_PCPU_INT1_REG (0x0068)



GPIO_PROCPU1_INT GPIO32 ~ 53 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 13 of GPIO_PIN*n*_REG). (RO)

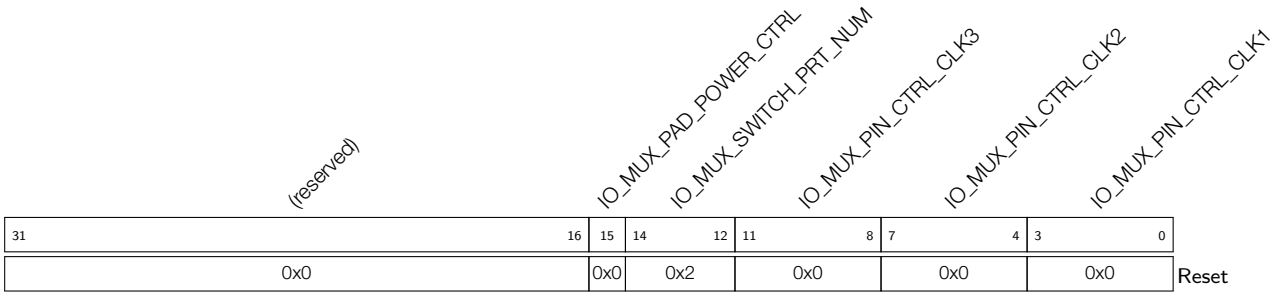
Register 5.33: GPIO_PCPU_NMI_INT1_REG (0x006C)



GPIO_PROCPU_NMI1_INT GPIO32 ~ 53 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

5.15.2 IO MUX Registers

Register 5.34: IO_MUX_PIN_CTRL (0x0000)



IO_MUX_PIN_CTRL_CLK*x* If you want to output clock for I2S0 to:

CLK_OUT1 then set IO_MUX_PIN_CTRL[3:0] = 0x0

CLK_OUT2 then set IO_MUX_PIN_CTRL[3:0] = 0x0 and IO_MUX_PIN_CTRL[7:4] = 0x0;

CLK_OUT3 then set IO_MUX_PIN_CTRL[3:0] = 0x0 and IO_MUX_PIN_CTRL[11:8] = 0x0.

Note:

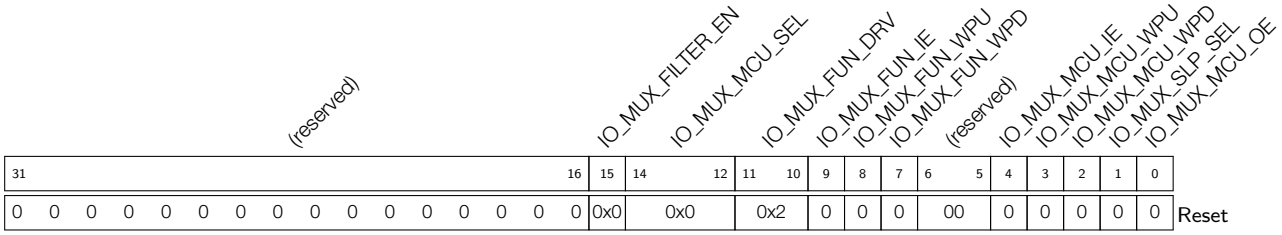
Only the above mentioned combinations of clock source and clock output pins are possible.

The CLK_OUT1 ~ 3 can be found in IO_MUX Pad List.

IO_MUX_SWITCH_PRT_NUM IO pad power switch delay, delay unit is one APB clock.

IO_MUX_PAD_POWER_CTRL Select power voltage for GPIO33 ~ 37. 1: select VDD_SPI 1.8 V; 0: select VDD3P3_CPU 3.3 V.

Register 5.35: IO_MUX_*n*_REG (*n*: GPIO0-GPIO21, GPIO26-GPIO46) (0x0010+4n*)**



IO_MUX_MCU_OE Output enable of the pad in sleep mode. 1: Output enabled; 0: Output disabled. (R/W)

IO_MUX_SLP_SEL Sleep mode selection of this pad. Set to 1 to put the pad in sleep mode. (R/W)

IO_MUX_MCU_WPD Pull-down enable of the pad during sleep mode. 1: Internal pull-down enabled; 0: Internal pull-down disabled. (R/W)

IO_MUX_MCU_WPU Pull-up enable of the pad during sleep mode. 1: Internal pull-up enabled; 0: Internal pull-up disabled.

IO_MUX_MCU_IE Input enable of the pad during sleep mode. 1: Input enabled; 0: Input disabled. (R/W)

IO_MUX_FUN_WPD Pull-down enable of the pad. 1: Pull-down enabled; 0: Pull-down disabled. (R/W)

IO_MUX_FUN_WPU Pull-up enable of the pad. 1: Internal pull-up enabled; 0: Internal pull-up disabled. (R/W)

IO_MUX_FUN_IE Input enable of the pad. 1: Input enabled; 0: Input disabled. (R/W)

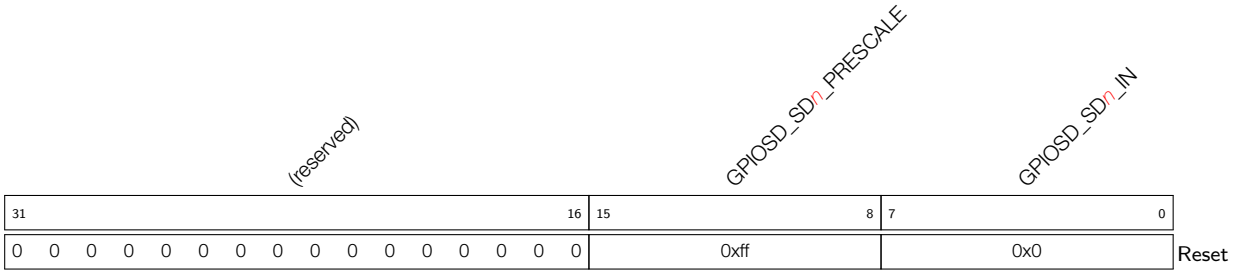
IO_MUX_FUN_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

IO_MUX_MCU_SEL Select IO MUX function for this signal. 0: Select Function 0; 1: Select Function 1, etc. (R/W)

IO_MUX_FILTER_EN Enable filter for pin input signals. 1: Filter enabled; 2: Filter disabled. (R/W)

5.15.3 Sigma Delta Modulated Output Registers

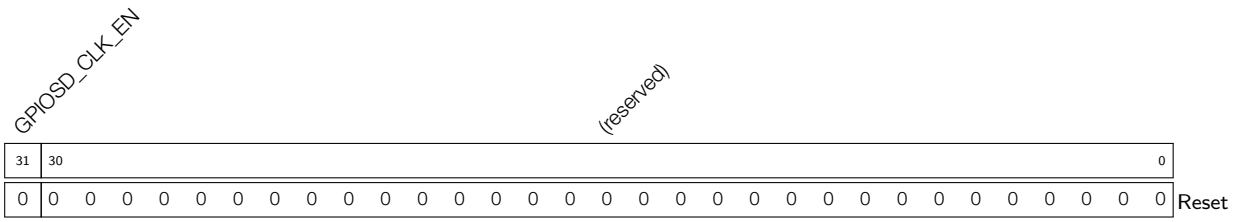
Register 5.36: GPIOSD_SIGMADELTA*n*_REG (*n*: 0-7) (0x0000+4n*)**



GPIOSD_SD*n*_IN This field is used to configure the duty cycle of sigma delta modulation output. (R/W)

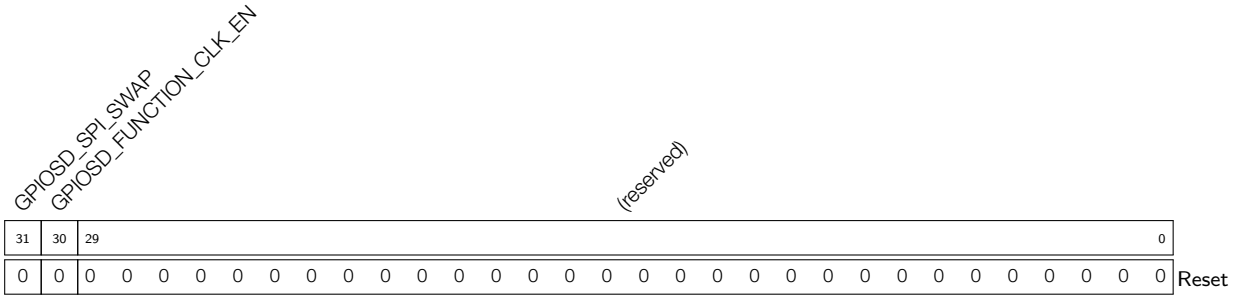
GPIOSD_SD*n*_PRESCALE This field is used to set a divider value to divide APB clock. (R/W)

Register 5.37: GPIOSD_SIGMADELTA_CG_REG (0x0020)



GPIOSD_CLK_EN Clock enable bit of configuration registers for sigma delta modulation. (R/W)

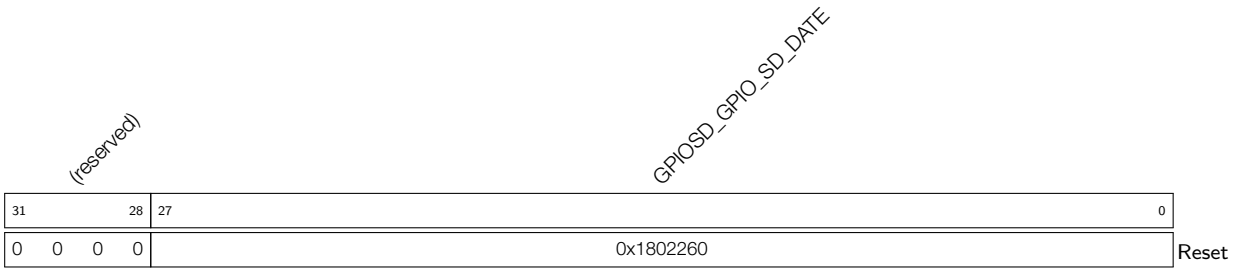
Register 5.38: GPIOSD_SIGMADELTA_MISC_REG (0x0024)



GPIOSD_FUNCTION_CLK_EN Clock enable bit of sigma delta modulation. (R/W)

GPIOSD_SPI_SWAP Reserved. (R/W)

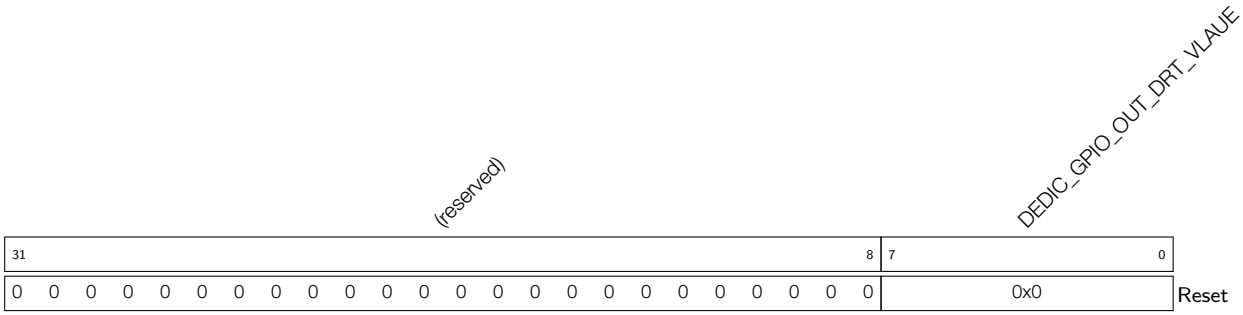
Register 5.39: GPIOSD_SIGMADELTA_VERSION_REG (0x0028)



GPIOSD_GPIO_SD_DATE Version control register. (R/W)

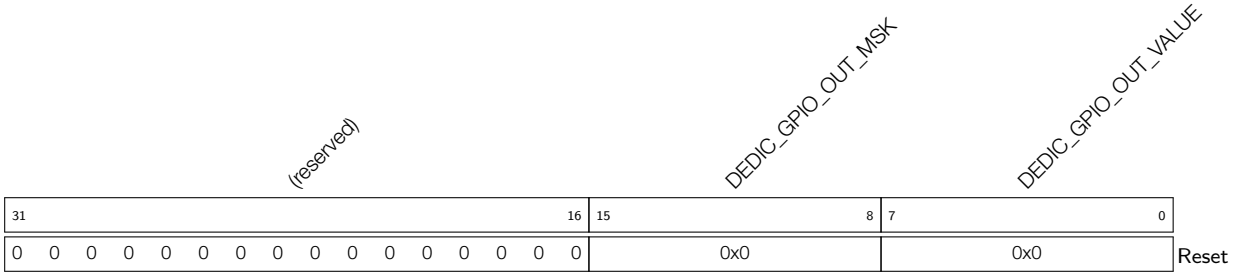
5.15.4 Dedicated GPIO Registers

Register 5.40: DEDIC_GPIO_OUT_DRT_REG (0x0000)



DEDIC_GPIO_OUT_DRT_VLAUE This register is used to configure direct output value of 8-channel dedicated GPIO. (WO)

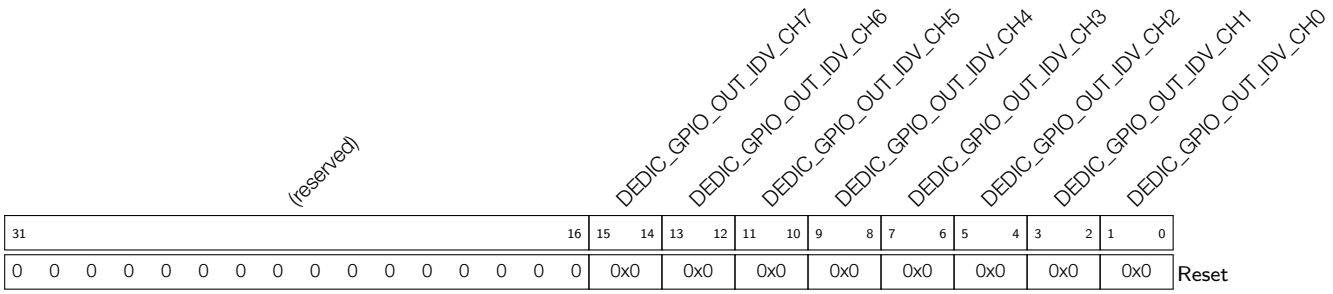
Register 5.41: DEDIC_GPIO_OUT_MSK_REG (0x0004)



DEDIC_GPIO_OUT_VALUE This register is used to configure updated output value of 8-channel dedicated GPIO. (WO)

DEDIC_GPIO_OUT_MSK This register is used to configure channels which would be updated. 1: corresponding channel's output would be updated. (WO)

Register 5.42: DEDIC_GPIO_OUT_IDV_REG (0x0008)



DEDIC_GPIO_OUT_IDV_CH0 Configure channel 0 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH1 Configure channel 1 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH2 Configure channel 2 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH3 Configure channel 3 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

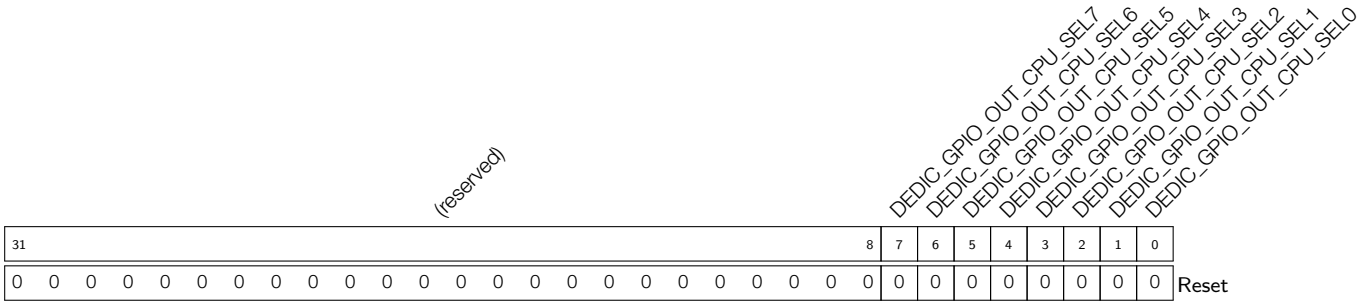
DEDIC_GPIO_OUT_IDV_CH4 Configure channel 4 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH5 Configure channel 5 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH6 Configure channel 6 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

DEDIC_GPIO_OUT_IDV_CH7 Configure channel 7 output value. 0: hold output value. 1: set output value. 2: clear output value. 3: inverse output value. (WO)

Register 5.43: DEDIC_GPIO_OUT_CPU_REG (0x0010)



DEDIC_GPIO_OUT_CPU_SEL0 Select GPIO out value configured by registers or CPU instructions for channel 0. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL1 Select GPIO out value configured by registers or CPU instructions for channel 1. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL2 Select GPIO out value configured by registers or CPU instructions for channel 2. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL3 Select GPIO out value configured by registers or CPU instructions for channel 3. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL4 Select GPIO out value configured by registers or CPU instructions for channel 4. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL5 Select GPIO out value configured by registers or CPU instructions for channel 5. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL6 Select GPIO out value configured by registers or CPU instructions for channel 6. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

DEDIC_GPIO_OUT_CPU_SEL7 Select GPIO out value configured by registers or CPU instructions for channel 7. 0: Configured by registers. 1: Configured by CPU instructions. (R/W)

Register 5.44: DEDIC_GPIO_IN_DLY_REG (0x0014)

DEDIC_GPIO_IN_DLY_CH0 Configure GPIO0 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH1 Configure GPIO1 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH2 Configure GPIO2 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH3 Configure GPIO3 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

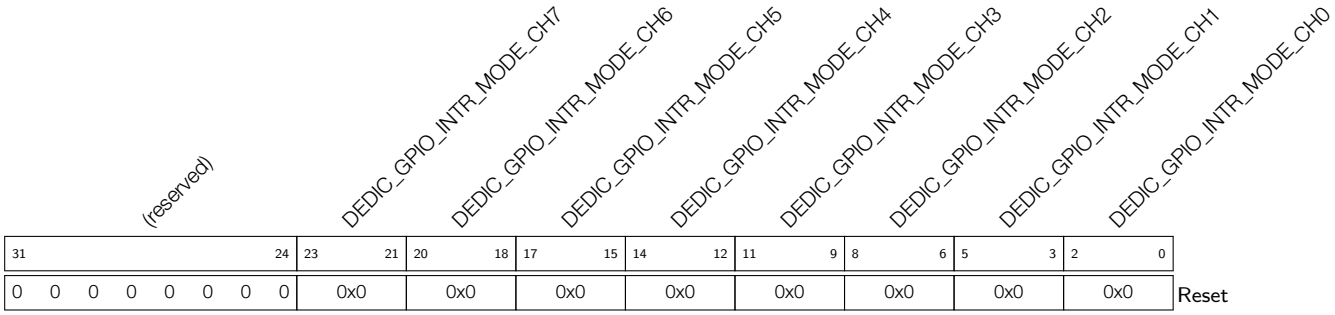
DEDIC_GPIO_IN_DLY_CH4 Configure GPIO4 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH5 Configure GPIO5 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH6 Configure GPIO6 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

DEDIC_GPIO_IN_DLY_CH7 Configure GPIO7 input delay. 0: no delay. 1: one clock delay. 2: two clock delay. 3: three clock delay. (R/W)

Register 5.45: DEDIC_GPIO_INTR_RCGN_REG (0x001C)



DEDIC_GPIO_INTR_MODE_CH0 Configure channel 0 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH1 Configure channel 1 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH2 Configure channel 2 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH3 Configure channel 3 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

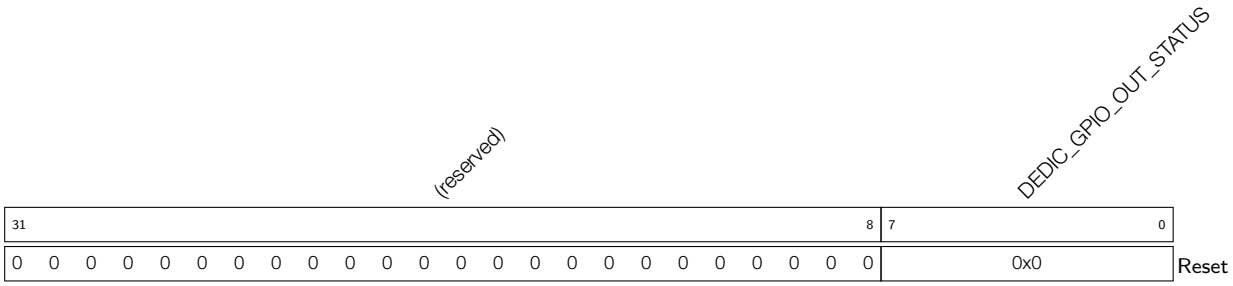
DEDIC_GPIO_INTR_MODE_CH4 Configure channel 4 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH5 Configure channel 5 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

DEDIC_GPIO_INTR_MODE_CH6 Configure channel 6 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

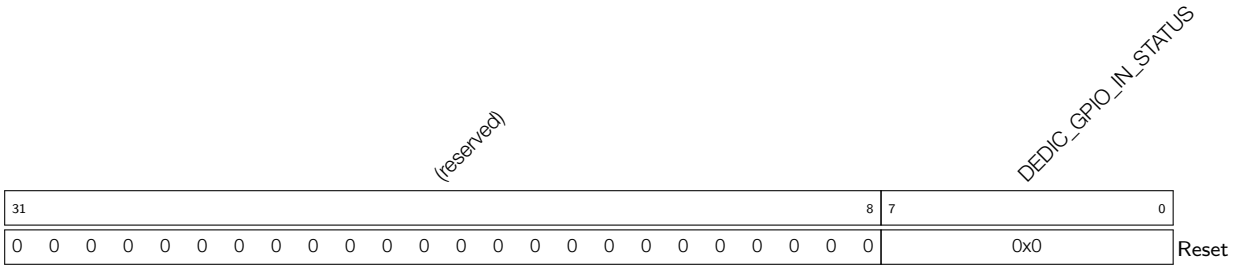
DEDIC_GPIO_INTR_MODE_CH7 Configure channel 7 interrupt generate mode. 0/1: do not generate interrupt. 2: low level trigger. 3: high level trigger. 4: falling edge trigger. 5: raising edge trigger. 6/7: falling and raising edge trigger. (R/W)

Register 5.46: DEDIC_GPIO_OUT_SCAN_REG (0x000C)



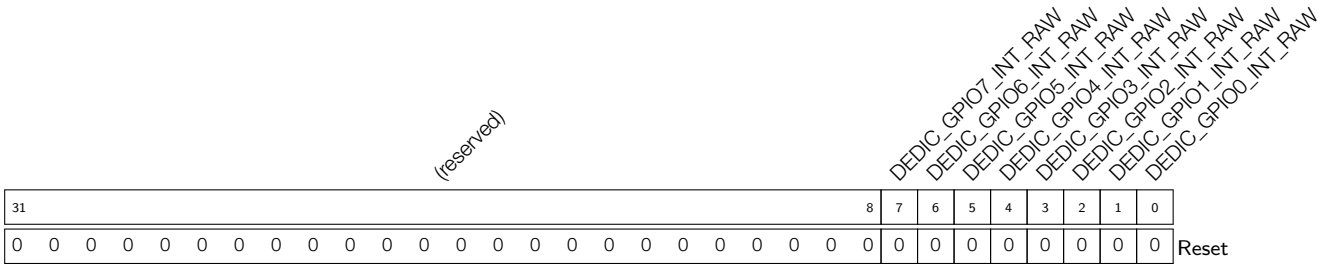
DEDIC_GPIO_OUT_STATUS GPIO output value configured by DEDIC_GPIO_OUT_DRT_REG, DEDIC_GPIO_OUT_MSK_REG, and DEDIC_GPIO_OUT_IDV_REG. (RO)

Register 5.47: DEDIC_GPIO_IN_SCAN_REG (0x0018)



DEDIC_GPIO_IN_STATUS GPIO input value after configured by DEDIC_GPIO_IN_DLY_REG. (RO)

Register 5.48: DEDIC_GPIO_INTR_RAW_REG (0x0020)



DEDIC_GPIO0_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO0 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO1_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO1 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO2_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO2 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO3_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO3 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

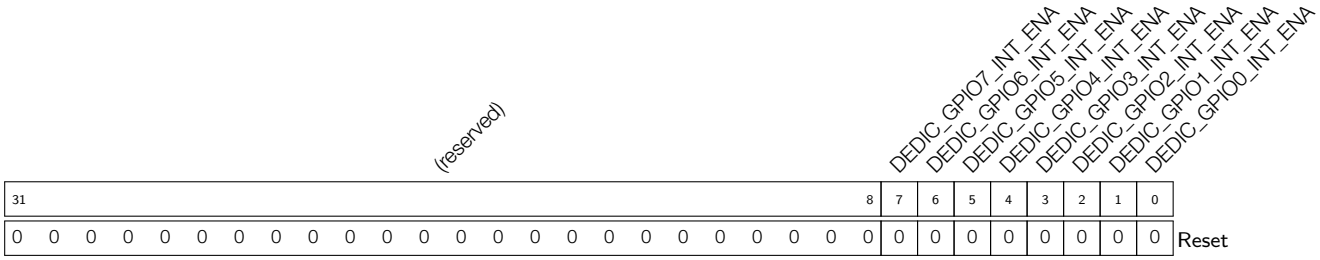
DEDIC_GPIO4_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO4 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO5_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO5 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO6_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO6 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

DEDIC_GPIO7_INT_RAW This interrupt raw bit turns to high level when dedicated GPIO7 has level/edge change configured by DEDIC_GPIO_INTR_RCGN_REG. (RO)

Register 5.49: DEDIC_GPIO_INTR_RLS_REG (0x0024)



DEDIC_GPIO0_INT_ENA The enable bit for DEDIC_GPIO0_INT_ST register. (R/W)

DEDIC_GPIO1_INT_ENA The enable bit for DEDIC_GPIO1_INT_ST register. (R/W)

DEDIC_GPIO2_INT_ENA The enable bit for DEDIC_GPIO2_INT_ST register. (R/W)

DEDIC_GPIO3_INT_ENA The enable bit for DEDIC_GPIO3_INT_ST register. (R/W)

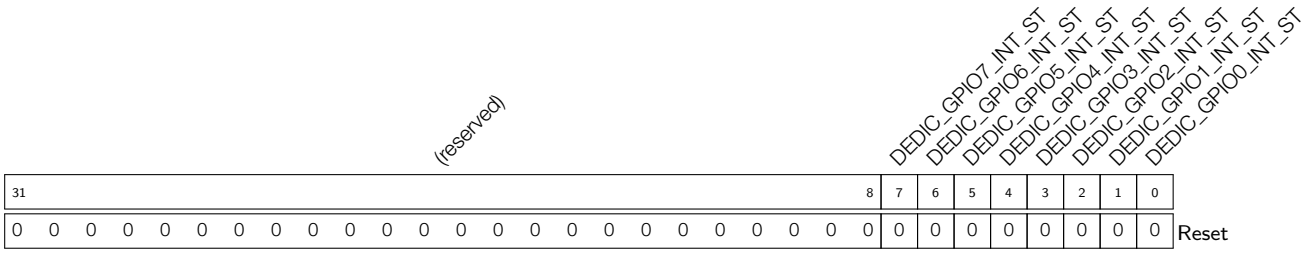
DEDIC_GPIO4_INT_ENA The enable bit for DEDIC_GPIO4_INT_ST register. (R/W)

DEDIC_GPIO5_INT_ENA The enable bit for DEDIC_GPIO5_INT_ST register. (R/W)

DEDIC_GPIO6_INT_ENA The enable bit for DEDIC_GPIO6_INT_ST register. (R/W)

DEDIC_GPIO7_INT_ENA The enable bit for DEDIC_GPIO7_INT_ST register. (R/W)

Register 5.50: DEDIC_GPIO_INTR_ST_REG (0x0028)



DEDIC_GPIO0_INT_ST This is the status bit for DEDIC_GPIO0_INT_RAW when DEDIC_GPIO0_INT_ENA is set to 1. (RO)

DEDIC_GPIO1_INT_ST This is the status bit for DEDIC_GPIO1_INT_RAW when DEDIC_GPIO1_INT_ENA is set to 1. (RO)

DEDIC_GPIO2_INT_ST This is the status bit for DEDIC_GPIO2_INT_RAW when DEDIC_GPIO2_INT_ENA is set to 1. (RO)

DEDIC_GPIO3_INT_ST This is the status bit for DEDIC_GPIO3_INT_RAW when DEDIC_GPIO3_INT_ENA is set to 1. (RO)

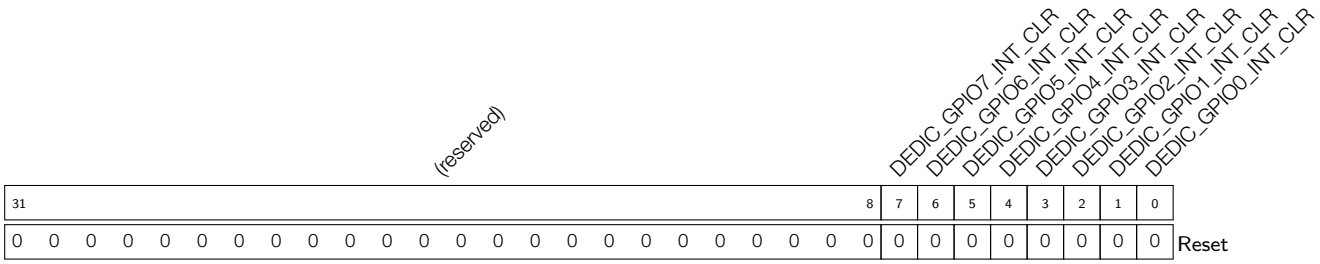
DEDIC_GPIO4_INT_ST This is the status bit for DEDIC_GPIO4_INT_RAW when DEDIC_GPIO4_INT_ENA is set to 1. (RO)

DEDIC_GPIO5_INT_ST This is the status bit for DEDIC_GPIO5_INT_RAW when DEDIC_GPIO5_INT_ENA is set to 1. (RO)

DEDIC_GPIO6_INT_ST This is the status bit for DEDIC_GPIO6_INT_RAW when DEDIC_GPIO6_INT_ENA is set to 1. (RO)

DEDIC_GPIO7_INT_ST This is the status bit for DEDIC_GPIO7_INT_RAW when DEDIC_GPIO7_INT_ENA is set to 1. (RO)

Register 5.51: DEDIC_GPIO_INTR_CLR_REG (0x002C)



DEDIC_GPIO0_INT_CLR Set this bit to clear the DEDIC_GPIO0_INT_RAW interrupt. (WO)

DEDIC_GPIO1_INT_CLR Set this bit to clear the DEDIC_GPIO1_INT_RAW interrupt. (WO)

DEDIC_GPIO2_INT_CLR Set this bit to clear the DEDIC_GPIO2_INT_RAW interrupt. (WO)

DEDIC_GPIO3_INT_CLR Set this bit to clear the DEDIC_GPIO3_INT_RAW interrupt. (WO)

DEDIC_GPIO4_INT_CLR Set this bit to clear the DEDIC_GPIO4_INT_RAW interrupt. (WO)

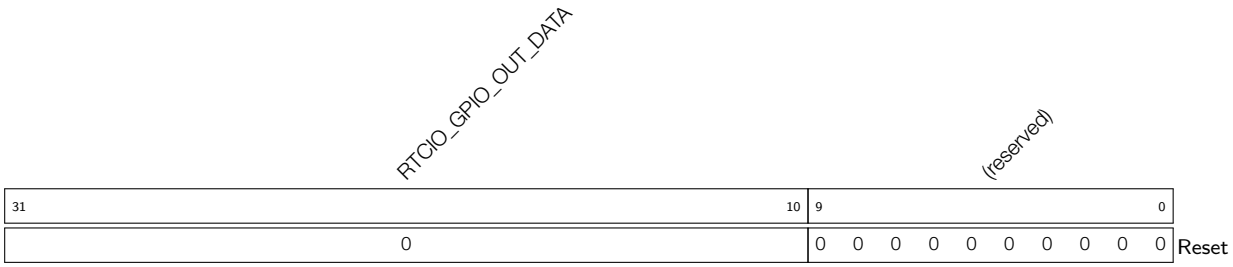
DEDIC_GPIO5_INT_CLR Set this bit to clear the DEDIC_GPIO5_INT_RAW interrupt. (WO)

DEDIC_GPIO6_INT_CLR Set this bit to clear the DEDIC_GPIO6_INT_RAW interrupt. (WO)

DEDIC_GPIO7_INT_CLR Set this bit to clear the DEDIC_GPIO7_INT_RAW interrupt. (WO)

5.15.5 RTC IO MUX Registers

Register 5.52: RTCIO_RTC_GPIO_OUT_REG (0x0000)

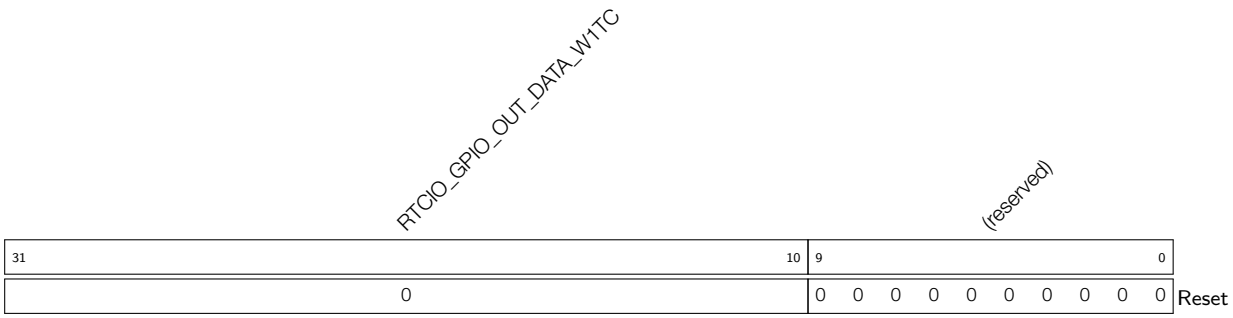


RTCIO_GPIO_OUT_DATA GPIO0 ~ 21 output register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. (R/W)

Register 5.53: RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0004)

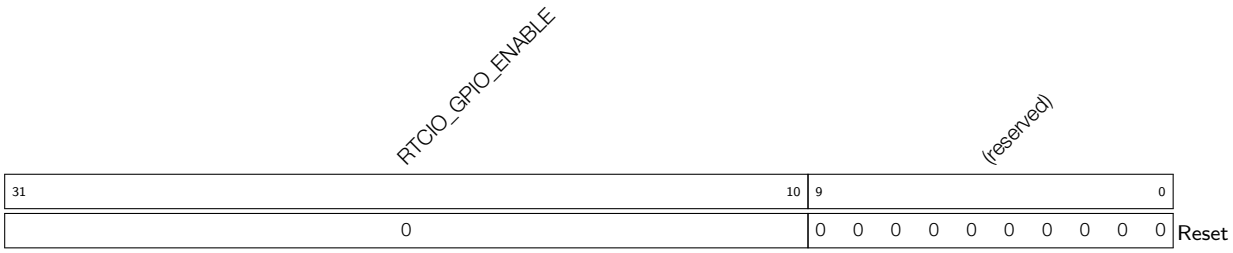
RTCIO_GPIO_OUT_DATA_W1TS GPIO0 ~ 21 output set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set RTCIO_RTC_GPIO_OUT_REG. (WO)

Register 5.54: RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0008)



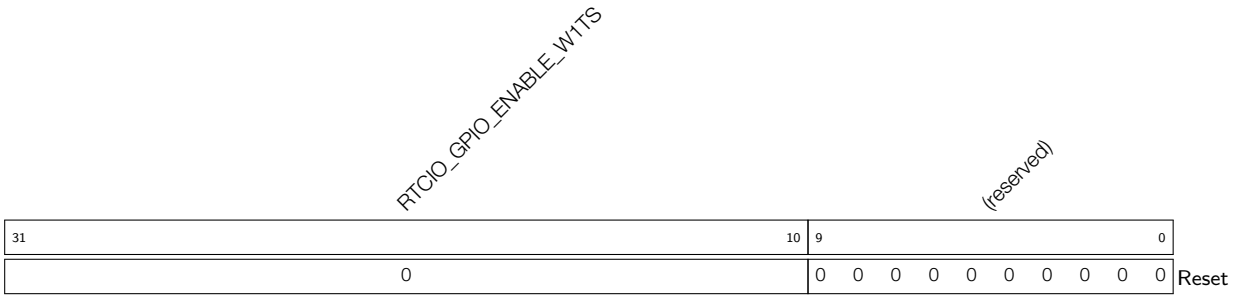
RTCIO_GPIO_OUT_DATA_W1TC GPIO0 ~ 21 output clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear RTCIO_RTC_GPIO_OUT_REG. (WO)

Register 5.55: RTCIO_RTC_GPIO_ENABLE_REG (0x000C)



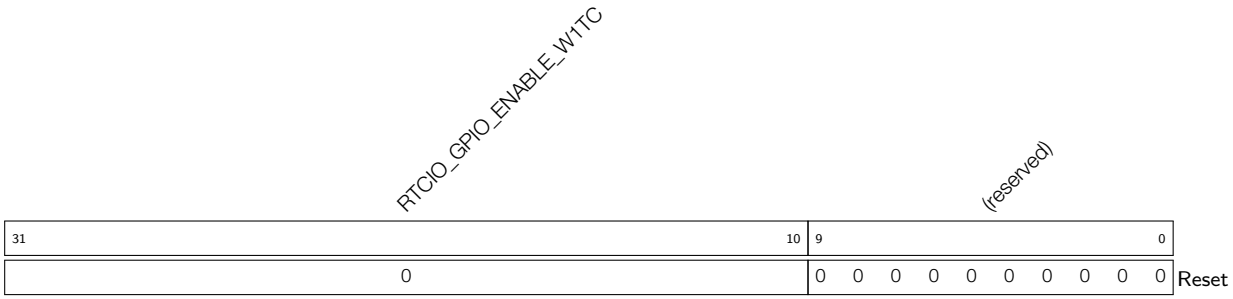
RTCIO_GPIO_ENABLE GPIO0 ~ 21 output enable. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. If the bit is set to 1, it means this GPIO pad is output. (R/W)

Register 5.56: RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0010)



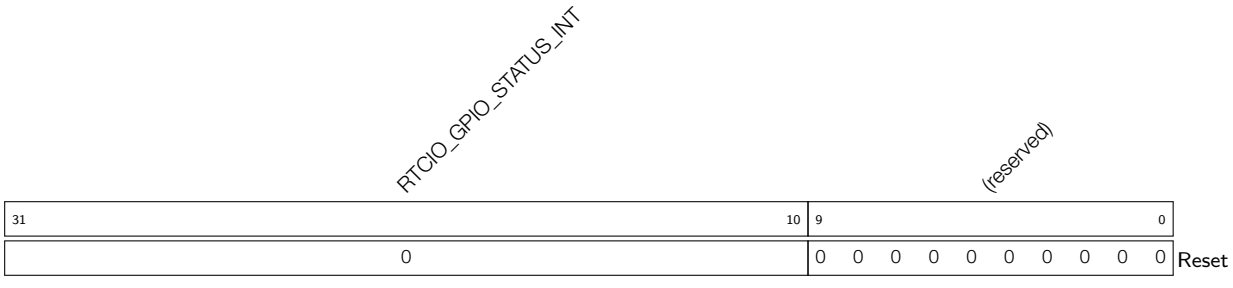
RTCIO_GPIO_ENABLE_W1TS GPIO0 ~ 21 output enable set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set RTCIO_RTC_GPIO_ENABLE_REG. (WO)

Register 5.57: RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0014)



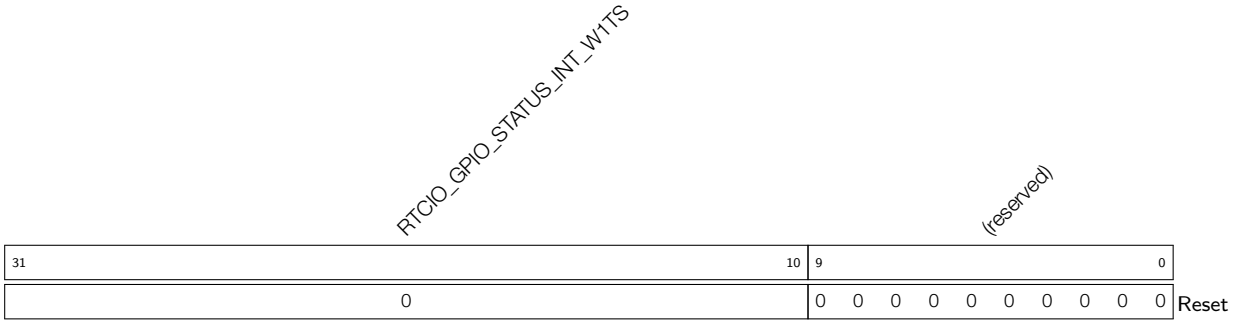
RTCIO_GPIO_ENABLE_W1TC GPIO0 ~ 21 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear RTCIO_RTC_GPIO_ENABLE_REG. (WO)

Register 5.58: RTCIO_RTC_GPIO_STATUS_REG (0x0018)



RTCIO_GPIO_STATUS_INT GPIO0 ~ 21 interrupt status register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. This register should be used together with RTCIO_RTC_GPIO_PIN*n*_INT_TYPE in RTCIO_RTC_GPIO_PIN*n*_REG. 0: no interrupt; 1: corresponding interrupt. (R/W)

Register 5.59: RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x001C)

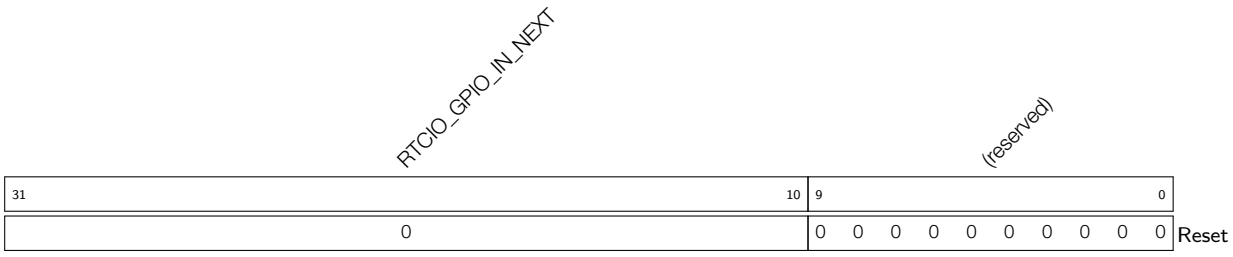


RTCIO_GPIO_STATUS_INT_W1TS GPIO0 ~ 21 interrupt set register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_GPIO_STATUS_INT will be set to 1. Recommended operation: use this register to set RTCIO_GPIO_STATUS_INT. (WO)

Register 5.60: RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0020)

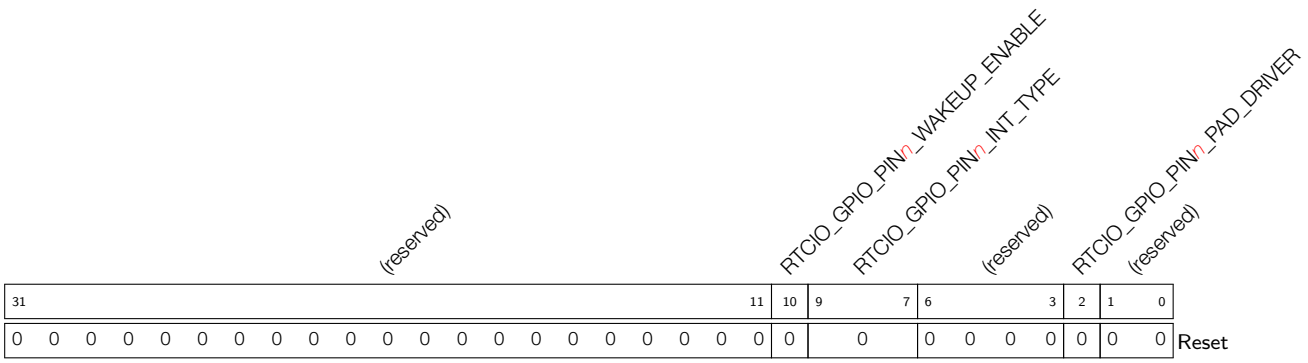
RTCIO_GPIO_STATUS_INT_W1TC GPIO0 ~ 21 interrupt clear register. If the value 1 is written to a bit here, the corresponding bit in RTCIO_GPIO_STATUS_INT will be cleared. Recommended operation: use this register to clear RTCIO_GPIO_STATUS_INT. (WO)

Register 5.61: RTCIO_RTC_GPIO_IN_REG (0x0024)



RTCIO_GPIO_IN_NEXT GPIO0 ~ 21 input value. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. Each bit represents a pad input value, 1 for high level, and 0 for low level. (RO)

Register 5.62: RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-21) (0x0028+4n*)**



RTCIO_GPIO_PIN*n*_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain. (R/W)

RTCIO_GPIO_PIN*n*_INT_TYPE GPIO 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)

RTCIO_GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable. This will only wake up ESP32-S2 from Light-sleep. (R/W)

Register 5.63: RTCIO_TOUCH_PADn_REG (n: 0-14) (0x0084+4*n)

Bits	Field	Reset
31	(reserved)	0
30:29	RTCIO_TOUCH_PADn_DRV	2
28	RTCIO_TOUCH_PADn_RDE	1
27	RTCIO_TOUCH_PADn_RUE	0
26	(reserved)	0
25:23	RTCIO_TOUCH_PADn_DAC	0x4
22	RTCIO_TOUCH_PADn_START	0
21	RTCIO_TOUCH_PADn_TIE_OPT	0
20	RTCIO_TOUCH_PADn_XPD	0
19	RTCIO_TOUCH_PADn_MUX_SEL	0
18:17	RTCIO_TOUCH_PADn_FUN_SEL	0
16	RTCIO_TOUCH_PADn_SLP_SEL	0
15	RTCIO_TOUCH_PADn_SLP_IE	0
14	RTCIO_TOUCH_PADn_SLP_OE	0
13	RTCIO_TOUCH_PADn_FUN_IE	0
12:0	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0

RTCIO_TOUCH_PADn_FUN_IE Input enable in normal execution. (R/W)

RTCIO_TOUCH_PADn_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_TOUCH_PADn_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_TOUCH_PADn_SLP_SEL 0: no sleep mode; 1: enable sleep mode. (R/W)

RTCIO_TOUCH_PADn_FUN_SEL Function selection. (R/W)

RTCIO_TOUCH_PADn_MUX_SEL Connect the RTC pad input to digital pad input. 0 is available. (R/W)

RTCIO_TOUCH_PADn_XPD Touch sensor power on. (R/W)

RTCIO_TOUCH_PADn_TIE_OPT The tie option of touch sensor. 0: tie low; 1: tie high. (R/W)

RTCIO_TOUCH_PADn_START Start touch sensor. (R/W)

RTCIO_TOUCH_PADn_DAC Touch sensor slope control. 3-bit for each touch pad, defaults to 0x4. (R/W)

RTCIO_TOUCH_PADn_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_TOUCH_PADn_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_TOUCH_PADn_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.64: RTCIO_XTAL_32P_PAD_REG (0x00C0)

RTCIO_X32P_FUN_IE Input enable in normal execution. (R/W)

RTCIO_X32P_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_X32P_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_X32P_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_X32P_FUN_SEL Function selection (R/W)

RTCIO_X32P_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_X32P_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_X32P_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_X32P_DRV Select the drive strength of the pad. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.65: RTCIO_XTAL_32N_PAD_REG (0x00C4)

RTCIO_X32N_FUN_IE Input enable in normal execution. (R/W)

RTCIO_X32N_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_X32N_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_X32N_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_X32N_FUN_SEL Function selection (R/W)

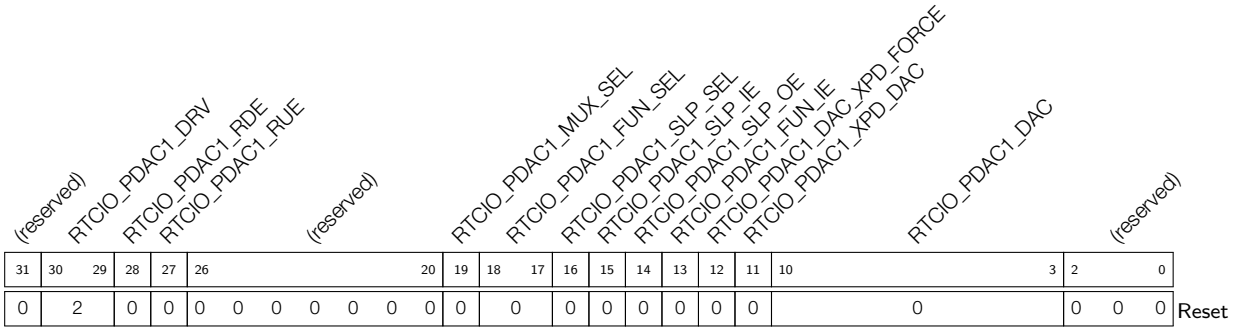
RTCIO_X32N_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_X32N_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_X32N_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_X32N_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.66: RTCIO_PAD_DAC1_REG (0x00C8)



RTCIO_PDAC1_DAC Configure DAC_1 output when RTCIO_PDAC1_DAC_XPD_FORCE is set to 1. (R/W)

RTCIO_PDAC1_XPD_DAC When RTCIO_PDAC1_DAC_XPD_FORCE is set to 1, 1: enable DAC_1 output; 0: disable DAC_1 output. (R/W)

RTCIO_PDAC1_DAC_XPD_FORCE 1: use RTCIO_PDAC1_XPD_DAC to control DAC_1 output; 0: use SAR ADC FSM to control DAC_1 output. (R/W)

RTCIO_PDAC1_FUN_IE Input enable in normal execution. (R/W)

RTCIO_PDAC1_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_PDAC1_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_PDAC1_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_PDAC1_FUN_SEL DAC_1 function selection. (R/W)

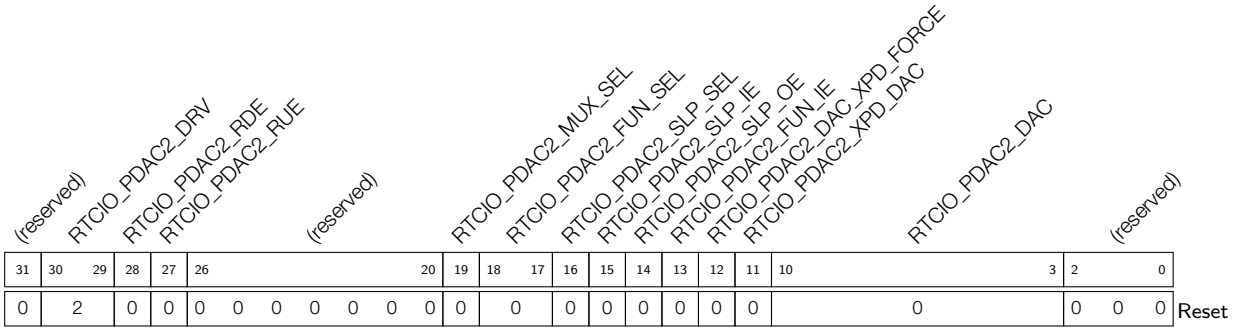
RTCIO_PDAC1_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_PDAC1_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_PDAC1_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_PDAC1_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.67: RTCIO_PAD_DAC2_REG (0x00CC)



RTCIO_PDAC2_DAC Configure DAC_2 output when RTCIO_PDAC2_DAC_XPD_FORCE is set to 1. (R/W)

RTCIO_PDAC2_XPD_DAC When RTCIO_PDAC2_DAC_XPD_FORCE is set to 1, 1: enable DAC_2 output; 0: disable DAC_2 output. (R/W)

RTCIO_PDAC2_DAC_XPD_FORCE 1: use RTCIO_PDAC2_XPD_DAC to control DAC_2 output; 0: use SAR ADC FSM to control DAC_2 output. (R/W)

RTCIO_PDAC2_FUN_IE Input enable in normal execution. (R/W)

RTCIO_PDAC2_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_PDAC2_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_PDAC2_SLP_SEL 1: enable sleep mode; 0: no sleep mode (R/W)

RTCIO_PDAC2_FUN_SEL DAC_2 function selection. (R/W)

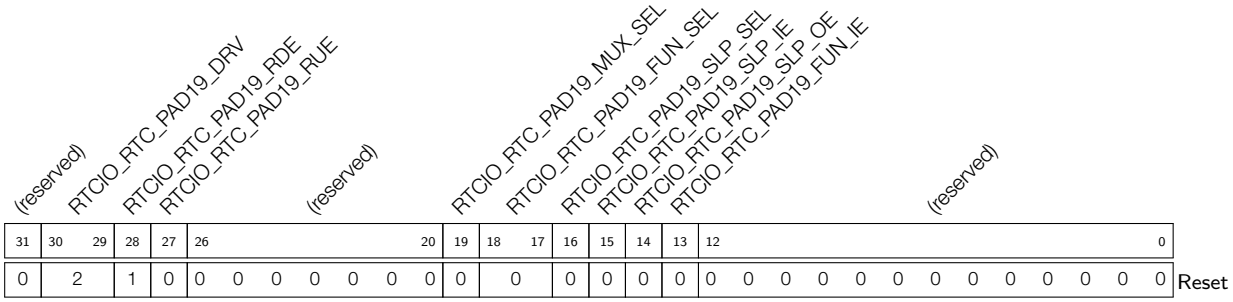
RTCIO_PDAC2_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO. (R/W)

RTCIO_PDAC2_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_PDAC2_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_PDAC2_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.68: RTCIO_RTC_PAD19_REG (0x00D0)



RTCIO_RTC_PAD19_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD19_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD19_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD19_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD19_FUN_SEL Function selection (R/W)

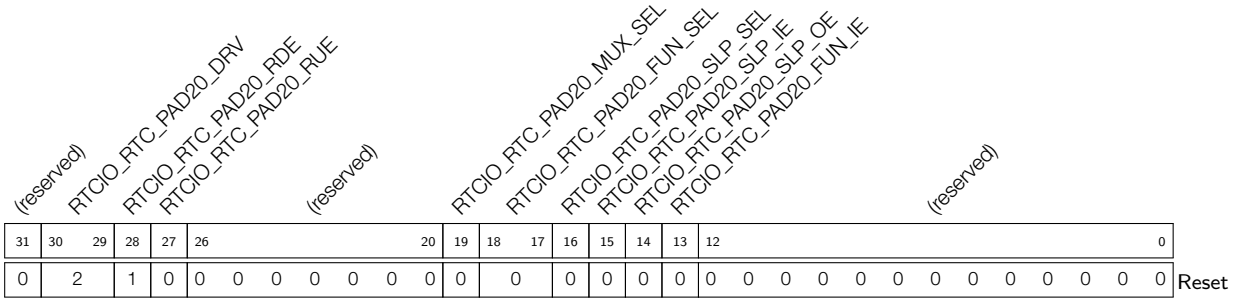
RTCIO_RTC_PAD19_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO (R/W)

RTCIO_RTC_PAD19_RUE Pull-up enable of the pad. 1: internal pull-up enabled, 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD19_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_RTC_PAD19_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.69: RTCIO_RTC_PAD20_REG (0x00D4)



RTCIO_RTC_PAD20_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD20_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD20_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD20_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD20_FUN_SEL Function selection. (R/W)

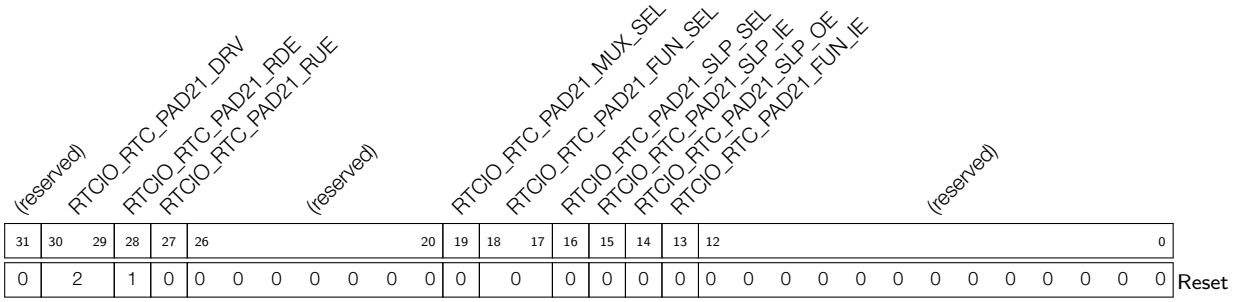
RTCIO_RTC_PAD20_MUX_SEL 1: use RTC GPIO, 0: use digital GPIO. (R/W)

RTCIO_RTC_PAD20_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD20_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

RTCIO_RTC_PAD20_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.70: RTCIO_RTC_PAD21_REG (0x00D8)



RTCIO_RTC_PAD21_FUN_IE Input enable in normal execution. (R/W)

RTCIO_RTC_PAD21_SLP_OE Output enable in sleep mode. (R/W)

RTCIO_RTC_PAD21_SLP_IE Input enable in sleep mode. (R/W)

RTCIO_RTC_PAD21_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTCIO_RTC_PAD21_FUN_SEL Function selection. (R/W)

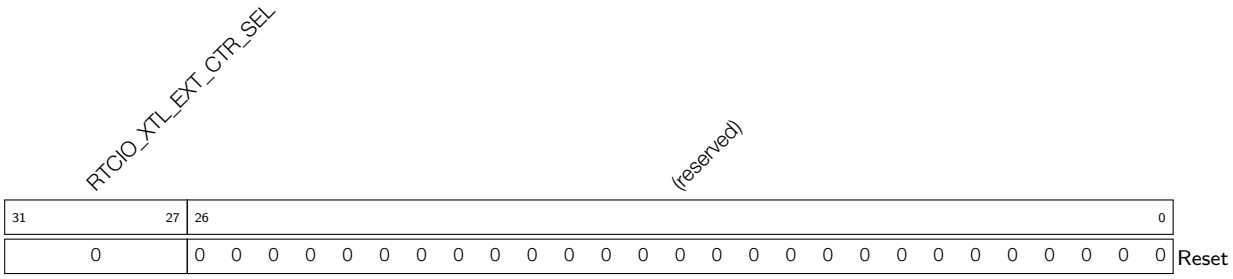
RTCIO_RTC_PAD21_MUX_SEL 1: use RTC GPIO,0: use digital GPIO. (R/W)

RTCIO_RTC_PAD21_RUE Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTCIO_RTC_PAD21_RDE Pull-down enable of the pad. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

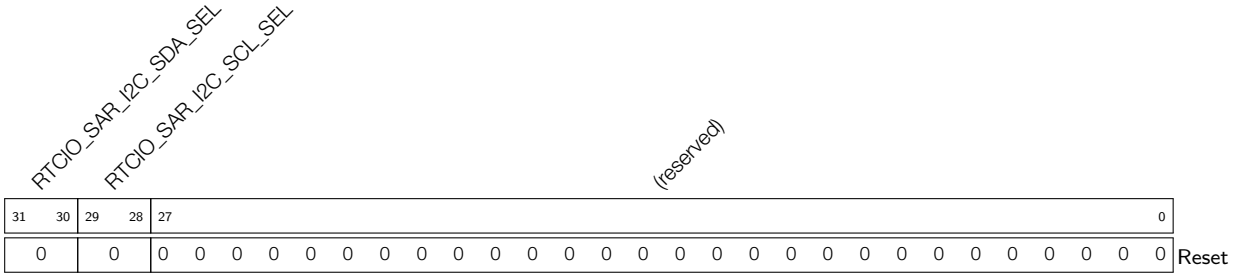
RTCIO_RTC_PAD21_DRV Select drive strength of the pad. 0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA. (R/W)

Register 5.71: RTCIO_XTL_EXT_CTR_REG (0x00E0)



RTCIO_XTL_EXT_CTR_SEL Select the external crystal power down enable source to get into sleep mode. 0: select GPIO0; 1: select GPIO1, etc. The input value on this pin XOR RTC_CNTL_EXT_XTL_CONF_REG[30] is the crystal power down enable signal. (R/W)

Register 5.72: RTCIO_SAR_I2C_IO_REG (0x00E4)



RTCIO_SAR_I2C_SCL_SEL Selects a pad the RTC I2C SCL signal connects to. 0: use TOUCH PAD0; 1: use TOUCH PAD2. (R/W)

RTCIO_SAR_I2C_SDA_SEL Selects a pad the RTC I2C SDA signal connects to. 0: use TOUCH PAD1; 1: use TOUCH PAD3. (R/W)

6. System Registers

6.1 Overview

The ESP32-S2 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. Specifically, ESP32-S2 has a various of 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.

6.2 Features

ESP32-S2 system registers can be used to control the following peripheral blocks and core modules:

- System and memory
- Reset and clock
- Interrupt matrix
- eFuse controller
- Low-power management
- Peripheral clock gating and reset

6.3 Function Description

6.3.1 System and Memory Registers

The following registers are used for system and memory configuration, such as cache configuration and memory remapping. For additional information, please refer to Chapter 1 *System and Memory*.

- SYSTEM_ROM_CTRL_0_REG
- SYSTEM_ROM_CTRL_1_REG
- SYSTEM_SRAM_CTRL_0_REG
- SYSTEM_SRAM_CTRL_1_REG
- SYSTEM_SRAM_CTRL_2_REG
- SYSTEM_RSA_PD_CTRL_REG
- SYSTEM_MEM_PD_MASK_REG
- SYSTEM_CACHE_CONTROL_REG
- SYSTEM_BUSTOEXTMEM_ENA_REG
- SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG

ROM Power Consumption Control

Registers SYSTEM_ROM_CTRL_0_REG and SYSTEM_ROM_CTRL_1_REG can be used to control the power consumption of ESP32-S2's ROM. Specifically:

- Setting different bits of the SYSTEM_ROM_FO field in register SYSTEM_ROM_CTRL_0_REG forces on the clock gates of different blocks of ROM.
- Setting different bits of the SYSTEM_ROM_FORCE_PD field in register SYSTEM_ROM_CTRL_1_REG powers down different blocks of internal ROM.
- Setting different bits of the SYSTEM_ROM_FORCE_PU field in register SYSTEM_ROM_CTRL_1_REG powers up different blocks of internal ROM.

For detailed information about the controlling bits of different blocks, please see Table 31 below.

Table 31: ROM Controlling Bit

ROM	Lowest Address1	Highest Address1	Lowest Address2	Highest Address2	Controlling Bit
Block0	0x4000_0000	0x4000_FFFF	-	-	Bit0
Block1	0x4001_2000	0x4001_FFFF	0x3FFA_0000	0x3FFA_FFFF	Bit1

SRAM Power Consumption Control

Registers SYSTEM_SRAM_CTRL_0_REG, SYSTEM_SRAM_CTRL_1_REG, and SYSTEM_SRAM_CTRL_2_REG can be used to control the power consumption of ESP32-S2's internal SRAM. Specifically,

- Setting different bits of the SYSTEM_SRAM_FO field in register SYSTEM_SRAM_CTRL_0_REG forces on the clock gates of different blocks of internal SRAM.
- Setting different bits of the SYSTEM_SRAM_FORCE_PD field in register SYSTEM_SRAM_CTRL_1_REG powers down different blocks of internal SRAM.
- Setting different bits of the SYSTEM_SRAM_FORCE_PU field in register SYSTEM_SRAM_CTRL_2_REG powers up different blocks of internal SRAM.

For detailed information about the controlling bits of different blocks, please see Table 32 below.

Table 32: SRAM Controlling Bit

SRAM	Lowest Address1	Highest Address1	Lowest Address2	Highest Address2	Controlling Bit
Block0	0x4002_0000	0x4002_1FFF	0x3FFB_0000	0x3FFB_1FFF	Bit0
Block1	0x4002_2000	0x4002_3FFF	0x3FFB_2000	0x3FFB_3FFF	Bit1
Block2	0x4002_4000	0x4002_5FFF	0x3FFB_4000	0x3FFB_5FFF	Bit2
Block3	0x4002_6000	0x4002_7FFF	0x3FFB_6000	0x3FFB_7FFF	Bit3
Block4	0x4002_8000	0x4002_BFFF	0x3FFB_8000	0x3FFB_BFFF	Bit4
Block5	0x4002_C000	0x4002_FFFF	0x3FFB_C000	0x3FFB_FFFF	Bit5
Block6	0x4003_0000	0x4003_3FFF	0x3FFC_0000	0x3FFC_3FFF	Bit6
Block7	0x4003_4000	0x4003_7FFF	0x3FFC_4000	0x3FFC_7FFF	Bit7
Block8	0x4003_8000	0x4003_BFFF	0x3FFC_8000	0x3FFC_BFFF	Bit8
Block9	0x4003_C000	0x4003_FFFF	0x3FFC_C000	0x3FFC_FFFF	Bit9
Block10	0x4004_0000	0x4004_3FFF	0x3FFD_0000	0x3FFD_3FFF	Bit10

Block11	0x4004_4000	0x4004_7FFF	0x3FFD_4000	0x3FFD_7FFF	Bit11
Block12	0x4004_8000	0x4004_BFFF	0x3FFD_8000	0x3FFD_BFFF	Bit12
Block13	0x4004_C000	0x4004_FFFF	0x3FFD_C000	0x3FFD_FFFF	Bit13
Block14	0x4005_0000	0x4005_3FFF	0x3FFE_0000	0x3FFE_3FFF	Bit14
Block15	0x4005_4000	0x4005_7FFF	0x3FFE_4000	0x3FFE_7FFF	Bit15
Block16	0x4005_8000	0x4005_BFFF	0x3FFE_8000	0x3FFE_BFFF	Bit16
Block17	0x4005_C000	0x4005_FFFF	0x3FFE_C000	0x3FFE_FFFF	Bit17
Block18	0x4006_0000	0x4006_3FFF	0x3FFF_0000	0x3FFF_3FFF	Bit18
Block19	0x4006_4000	0x4006_7FFF	0x3FFF_4000	0x3FFF_7FFF	Bit19
Block20	0x4006_8000	0x4006_BFFF	0x3FFF_8000	0x3FFF_BFFF	Bit20
Block21	0x4006_C000	0x4006_FFFF	0x3FFF_C000	0x3FFF_FFFF	Bit21

6.3.2 Reset and Clock Registers

The following registers are used for reset and clock. For additional information, please refer to Chapter 2 *Reset and Clock*.

- SYSTEM_CPU_PER_CONF_REG
- SYSTEM_SYSCLK_CONF_REG
- SYSTEM_BT_LPCK_DIV_FRAC_REG

6.3.3 Interrupt Matrix Registers

The following registers are used for generating the CPU interrupt signals for the interrupt matrix. For additional information, please refer to Chapter 4 *Interrupt Matrix*

- 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

6.3.4 JTAG Software Enable Registers

The following registers are used for revoking the temporary disable of eFuse to JTAG.

- SYSTEM_JTAG_CTRL_0_REG
- SYSTEM_JTAG_CTRL_1_REG
- SYSTEM_JTAG_CTRL_2_REG
- SYSTEM_JTAG_CTRL_3_REG
- SYSTEM_JTAG_CTRL_4_REG
- SYSTEM_JTAG_CTRL_5_REG
- SYSTEM_JTAG_CTRL_6_REG
- SYSTEM_JTAG_CTRL_7_REG

6.3.5 Low-power Management Registers

The following registers are used for low-power management.

- SYSTEM_RTC_FASTMEM_CONFIG_REG
- SYSTEM_RTC_FASTMEM_CRC_REG

6.3.6 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 33.

- SYSTEM_CPU_PERI_CLK_EN_REG
- SYSTEM_CPU_PERI_RST_EN_REG
- SYSTEM_PERIP_CLK_EN0_REG
- SYSTEM_PERIP_RST_EN0_REG
- SYSTEM_PERIP_CLK_EN1_REG
- SYSTEM_PERIP_RST_EN1_REG

Table 33: Peripheral Clock Gating and Reset Bits

Peripheral	Clock Enabling Bit[1]	Reset Controlling Bit[2][3]
CPU Peripherals	**SYSTEM_CPU_PERI_CLK_EN_REG**	**SYSTEM_CPU_PERI_RST_EN_REG**
DEDICATED GPIO	SYSTEM_CLK_EN_DEDICATED_GPIO	SYSTEM_RST_EN_DEDICATED_GPIO
Peripherals	**SYSTEM_PERIP_CLK_EN0_REG**	**SYSTEM_PERIP_RST_EN0_REG**
Timers	SYSTEM_TIMERS_CLK_EN	SYSTEM_TIMERS_RST
Timer Group0	SYSTEM_TIMERGROUP_CLK_EN	SYSTEM_TIMERGROUP_RST
Timer Group1	SYSTEM_TIMERGROUP1_CLK_EN	SYSTEM_TIMERGROUP1_RST
System Timer	SYSTEM_SYSTIMER_CLK_EN	SYSTEM_SYSTIMER_RST
UART0	SYSTEM_UART_CLK_EN	SYSTEM_UART_RST
UART1	SYSTEM_UART1_CLK_EN	SYSTEM_UART1_RST
UART MEM	SYSTEM_UART_MEM_CLK_EN [4]	SYSTEM_UART_MEM_RST
SPI0, SPI1	SYSTEM_SPI01_CLK_EN	SYSTEM_SPI01_RST
SPI2	SYSTEM_SPI2_CLK_EN	SYSTEM_SPI2_RST
SPI3	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_RST
SPI4	SYSTEM_SPI4_CLK_EN	SYSTEM_SPI4_RST
SPI2 DMA	SYSTEM_SPI2_DMA_CLK_EN	SYSTEM_SPI2_DMA_RST
SPI3 DMA	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_DMA_RST
I2C0	SYSTEM_I2C_EXT0_CLK_EN	SYSTEM_I2C_EXT0_RST
I2C1	SYSTEM_I2C_EXT1_CLK_EN	SYSTEM_I2C_EXT1_RST
I2S0	SYSTEM_I2S0_CLK_EN	SYSTEM_I2S0_RST
I2S1	SYSTEM_I2S1_CLK_EN	SYSTEM_I2S1_RST
TWAI Controller	SYSTEM_CAN_CLK_EN	SYSTEM_CAN_RST
UHCI0	SYSTEM_UHCI0_CLK_EN	SYSTEM_UHCI0_RST
UHCI1	SYSTEM_UHCI1_CLK_EN	SYSTEM_UHCI1_RST
USB	SYSTEM_USB_CLK_EN	SYSTEM_USB_RST
RMT	SYSTEM_RMT_CLK_EN	SYSTEM_RMT_RST

PCNT	SYSTEM_PCNT_CLK_EN	SYSTEM_PCNT_RST
PWM0	SYSTEM_PWM0_CLK_EN	SYSTEM_PWM0_RST
PWM1	SYSTEM_PWM1_CLK_EN	SYSTEM_PWM1_RST
PWM2	SYSTEM_PWM2_CLK_EN	SYSTEM_PWM2_RST
PWM3	SYSTEM_PWM3_CLK_EN	SYSTEM_PWM3_RST
LED_PWM Controller	SYSTEM_LEDC_CLK_EN	SYSTEM_LEDC_RST
eFuse	SYSTEM_EFUSE_CLK_EN	SYSTEM_EFUSE_RST
APB SARADC	SYSTEM_APB_SARADC_CLK_EN	SYSTEM_APB_SARADC_RST
ADC2 ARB	SYSTEM_ADC2_ARB_CLK_EN	SYSTEM_ADC2_ARB_RST
WDG	SYSTEM_WDG_CLK_EN	SYSTEM_WDG_RST
Accelerators	**SYSTEM_PERIP_CLK_EN1_REG**	**SYSTEM_PERIP_RST_EN1_REG**
DMA	SYSTEM_CRYPTO_DMA_CLK_EN	SYSTEM_CRYPTO_DMA_RST [5]
HMAC	SYSTEM_CRYPTO_HMAC_CLK_EN	SYSTEM_CRYPTO_HMAC_RST [6]
Digital Signature	SYSTEM_CRYPTO_DS_CLK_EN	SYSTEM_CRYPTO_DS_RST [7]
RSA Accelerator	SYSTEM_CRYPTO_RSA_CLK_EN	SYSTEM_CRYPTO_RSA_RST
SHA Accelerator	SYSTEM_CRYPTO_SHA_CLK_EN	SYSTEM_CRYPTO_SHA_RST
AES Accelerator	SYSTEM_CRYPTO_AES_CLK_EN	SYSTEM_CRYPTO_AES_RST

Note:

1. Set the clock enable register to 1 to enable the clock, and to 0 to disable the clock;
2. Set the reset enabling register to 1 to reset a peripheral, and to 0 to disable the reset.
3. Reset registers are not cleared by hardware.
4. 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.
5. Crypto DMA is shared by AES and SHA accelerators.
6. Resetting this bit also resets the SHA accelerator.
7. Resetting this bit also resets the AES, SHA, and RSA accelerators.

6.4 Base Address

Users can access the system registers with base address, which can be seen in the following table. For more information about accessing system registers, please see Chapter 1 *System and Memory*.

Table 34: System Register Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C0000

6.5 Register Summary

The addresses in the following table are relative to the system registers base addresses provided in Section 6.4.

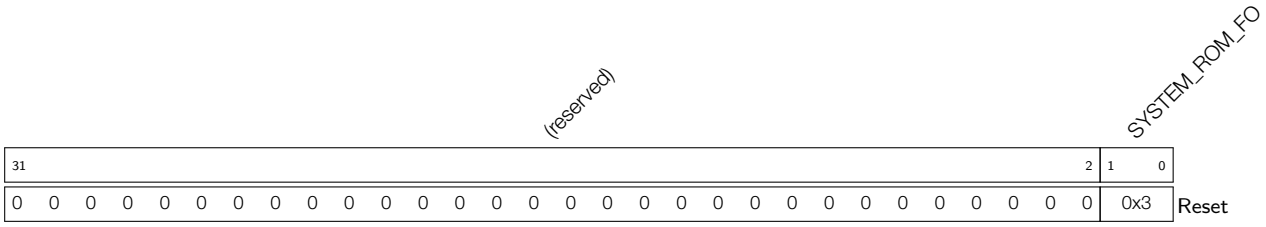
Name	Description	Address	Access
System and Memory Registers			
SYSTEM_ROM_CTRL_0_REG	System ROM configuration register 0	0x0000	R/W
SYSTEM_ROM_CTRL_1_REG	System ROM configuration register 1	0x0004	R/W
SYSTEM_SRAM_CTRL_0_REG	System SRAM configuration register 0	0x0008	R/W
SYSTEM_SRAM_CTRL_1_REG	System SRAM configuration register 1	0x000C	R/W
SYSTEM_SRAM_CTRL_2_REG	System SRAM configuration register 2	0x0088	R/W
SYSTEM_MEM_PD_MASK_REG	Memory power-related controlling register (under low-sleep)	0x003C	R/W
SYSTEM_RSA_PD_CTRL_REG	RSA memory remapping register	0x0068	R/W
SYSTEM_BUSTOEXTMEM_ENA_REG	EDMA enable register	0x006C	R/W
SYSTEM_CACHE_CONTROL_REG	Cache control register	0x0070	R/W
SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	External memory encrypt and decrypt controlling register	0x0074	R/W
Reset and Clock Registers			
SYSTEM_CPU_PER_CONF_REG	CPU peripheral clock configuration register	0x0018	R/W
SYSTEM_BT_LPCK_DIV_FRAC_REG	Divider fraction configuration register for low-power clock	0x0054	R/W
SYSTEM_SYSCLK_CONF_REG	SoC clock configuration register	0x008C	Varies
Interrupt Matrix Registers			
SYSTEM_CPU_INTR_FROM_CPU_0_REG	CPU interrupt controlling register 0	0x0058	R/W
SYSTEM_CPU_INTR_FROM_CPU_1_REG	CPU interrupt controlling register 1	0x005C	R/W
SYSTEM_CPU_INTR_FROM_CPU_2_REG	CPU interrupt controlling register 2	0x0060	R/W
SYSTEM_CPU_INTR_FROM_CPU_3_REG	CPU interrupt controlling register 3	0x0064	R/W
JTAG Software Enable Registers			
SYSTEM_JTAG_CTRL_0_REG	JTAG configuration register 0	0x001C	WO
SYSTEM_JTAG_CTRL_1_REG	JTAG configuration register 1	0x0020	WO
SYSTEM_JTAG_CTRL_2_REG	JTAG configuration register 2	0x0024	WO
SYSTEM_JTAG_CTRL_3_REG	JTAG configuration register 3	0x0028	WO
SYSTEM_JTAG_CTRL_4_REG	JTAG configuration register 4	0x002C	WO
SYSTEM_JTAG_CTRL_5_REG	JTAG configuration register 5	0x0030	WO
SYSTEM_JTAG_CTRL_6_REG	JTAG configuration register 6	0x0034	WO

Name	Description	Address	Access
SYSTEM_JTAG_CTRL_7_REG	JTAG configuration register 7	0x0038	WO
Low-Power Management Registers			
SYSTEM_RTC_FASTMEM_CONFIG_REG	RTC fast memory configuration register	0x0078	Varies
SYSTEM_RTC_FASTMEM_CRC_REG	RTC fast memory CRC controlling register	0x007C	RO
Peripheral Clock Gating and Reset Registers			
SYSTEM_CPU_PERI_CLK_EN_REG	CPU peripheral clock enable register	0x0010	R/W
SYSTEM_CPU_PERI_RST_EN_REG	CPU peripheral reset register	0x0014	R/W
SYSTEM_PERIP_CLK_EN0_REG	System peripheral clock (for hardware accelerators) enable register 0	0x0040	R/W
SYSTEM_PERIP_CLK_EN1_REG	System peripheral clock (for hardware accelerators) enable register 1	0x0044	R/W
SYSTEM_PERIP_RST_EN0_REG	System peripheral (hardware accelerators) reset register 0	0x0048	R/W
SYSTEM_PERIP_RST_EN1_REG	System peripheral (hardware accelerators) reset register 1	0x004C	R/W
Version Register			
SYSTEM_REG_DATE_REG	Version control register	0x0FFC	R/W

6.6 Registers

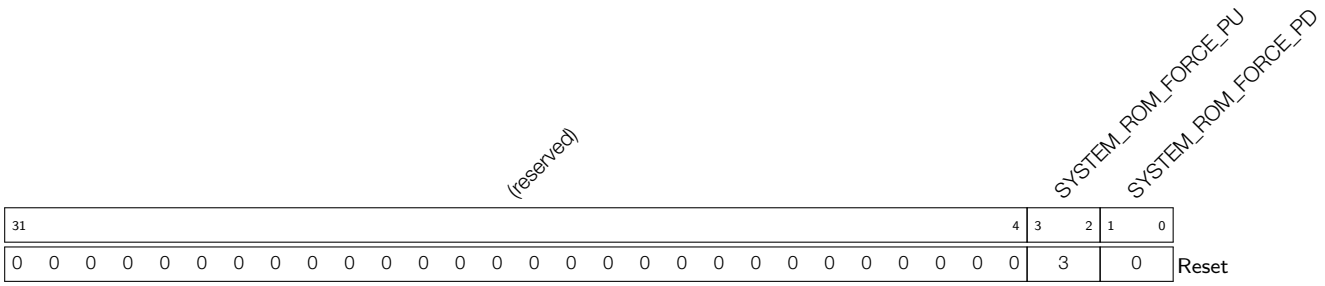
The addresses below are relative to the system registers base addresses provided in Section 6.4.

Register 6.1: SYSTEM_ROM_CTRL_0_REG (0x0000)



SYSTEM_ROM_FO This field is used to force on clock gate of internal ROM. For details, please refer to Table 31. (R/W)

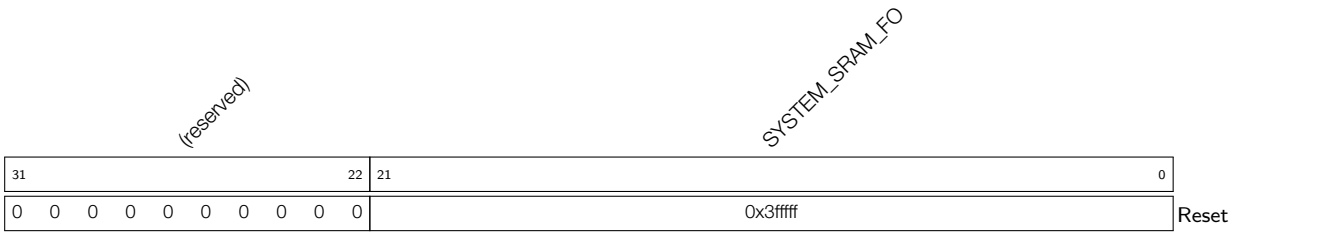
Register 6.2: SYSTEM_ROM_CTRL_1_REG (0x0004)



SYSTEM_ROM_FORCE_PD This field is used to power down internal ROM. For details, please refer to Table 31. (R/W)

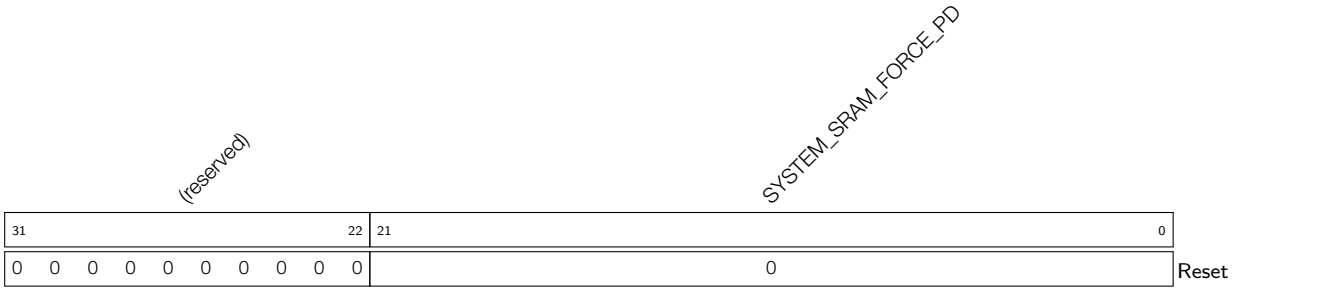
SYSTEM_ROM_FORCE_PU This field is used to power up internal ROM. For details, please refer to Table 31. (R/W)

Register 6.3: SYSTEM_SRAM_CTRL_0_REG (0x0008)



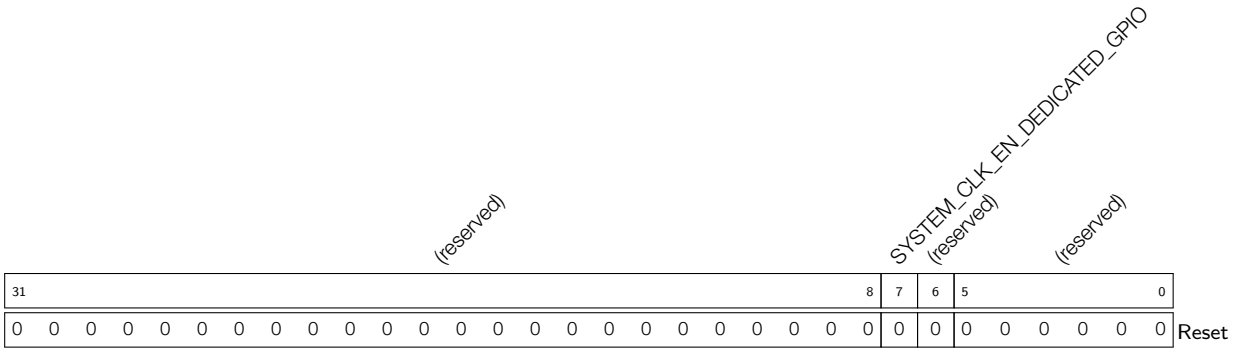
SYSTEM_SRAM_FO This field is used to force on clock gate of internal SRAM. For details, please refer to Table 32. (R/W)

Register 6.4: SYSTEM_SRAM_CTRL_1_REG (0x000C)



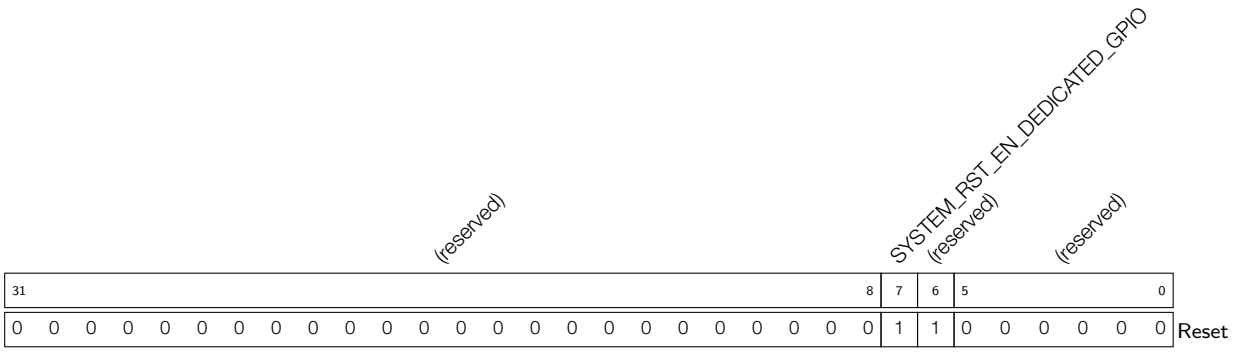
SYSTEM_SRAM_FORCE_PD This field is used to power down internal SRAM. For details, please refer to Table 32. (R/W)

Register 6.5: SYSTEM_CPU_PERI_CLK_EN_REG (0x0010)



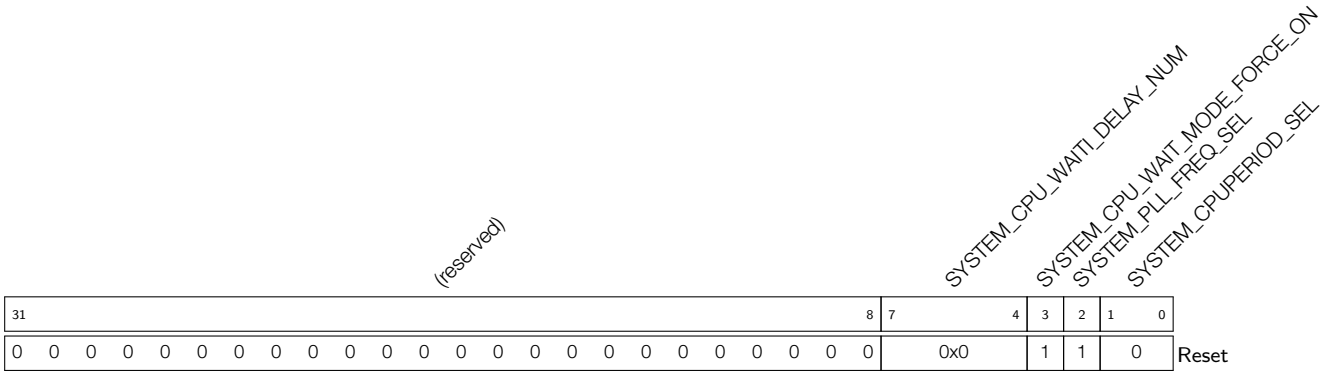
SYSTEM_CLK_EN_DEDICATED_GPIO Set this bit to enable clock of DEDICATED GPIO module.

Register 6.6: SYSTEM_CPU_PERI_RST_EN_REG (0x0014)



SYSTEM_RST_EN_DEDICATED_GPIO Set this bit to reset DEDICATED GPIO module. (R/W)

Register 6.7: SYSTEM_CPU_PER_CONF_REG (0x0018)



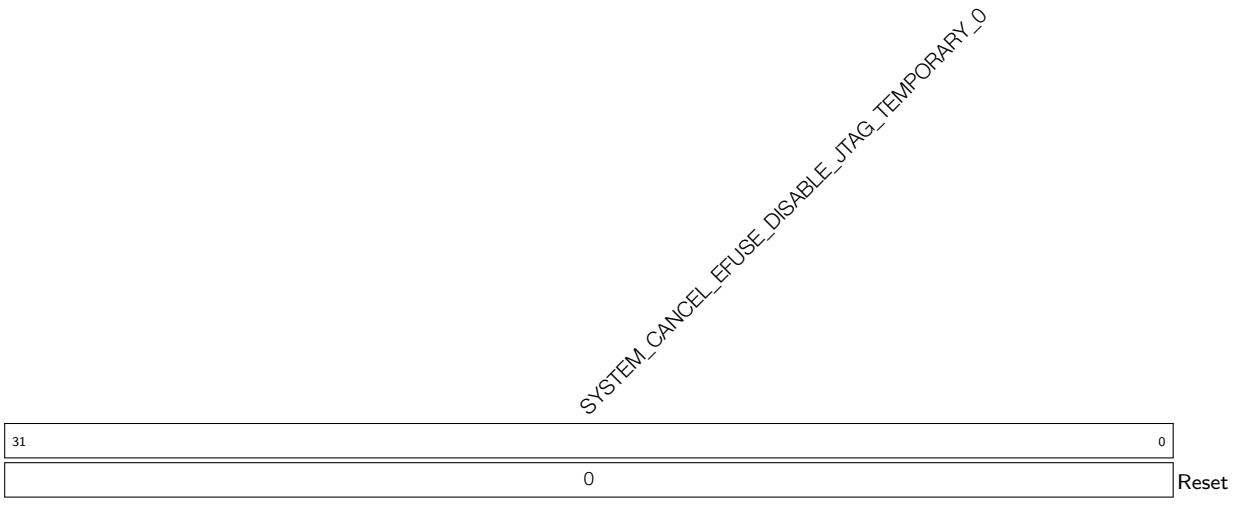
SYSTEM_CPUPERIOD_SEL This field is used to select the clock frequency of CPU or CPU period. For details, please refer to Table 8 in Chapter 2 *Reset and Clock*. (R/W)

SYSTEM_PLL_FREQ_SEL This field is used to select the PLL clock frequency based on CPU period. For details, please refer to Table 8 in Chapter 2 *Reset and Clock*. (R/W)

SYSTEM_CPU_WAIT_MODE_FORCE_ON Set this bit to force on CPU wait mode. In this mode, the clock gate of CPU is turned off until any interrupts happen. This mode could also be force on via WAITI instruction. (R/W)

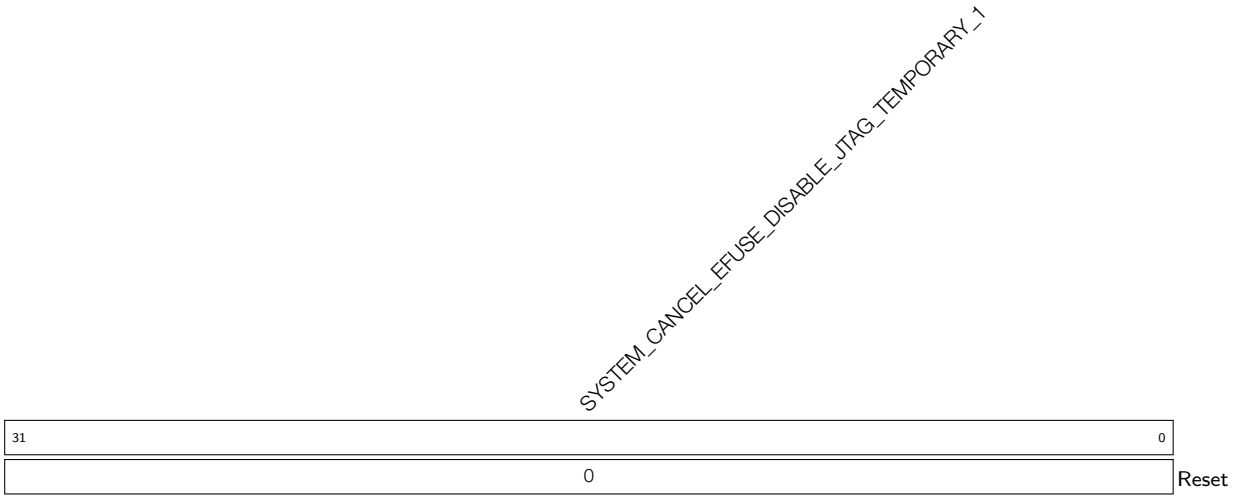
SYSTEM_CPU_WAITI_DELAY_NUM Sets the number of delay cycles to enter CPU wait mode after a WAITI instruction. (R/W)

Register 6.8: SYSTEM_JTAG_CTRL_0_REG (0x001C)



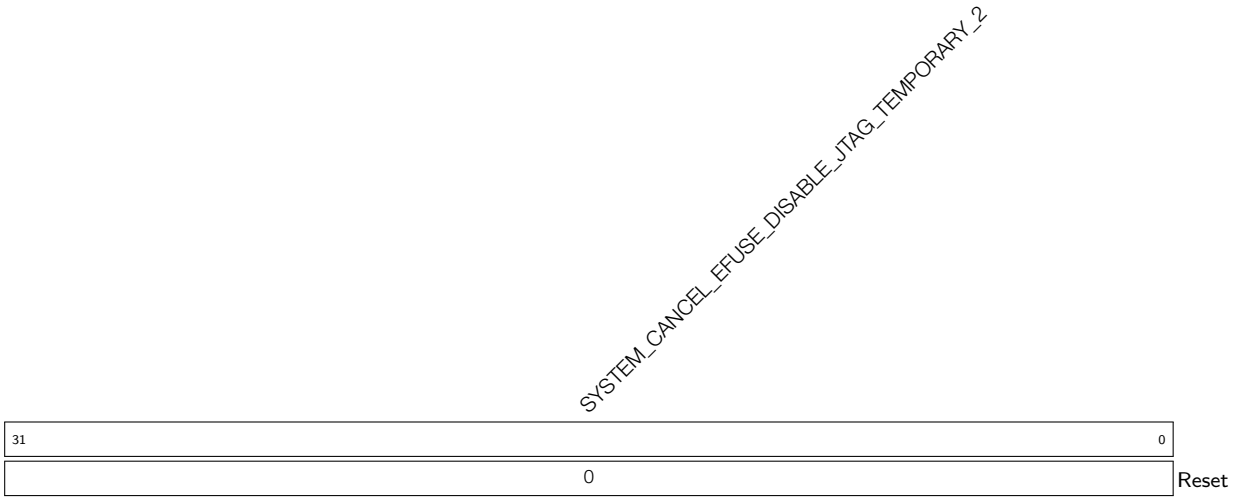
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 Stores the 0 to 31 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.9: SYSTEM_JTAG_CTRL_1_REG (0x0020)



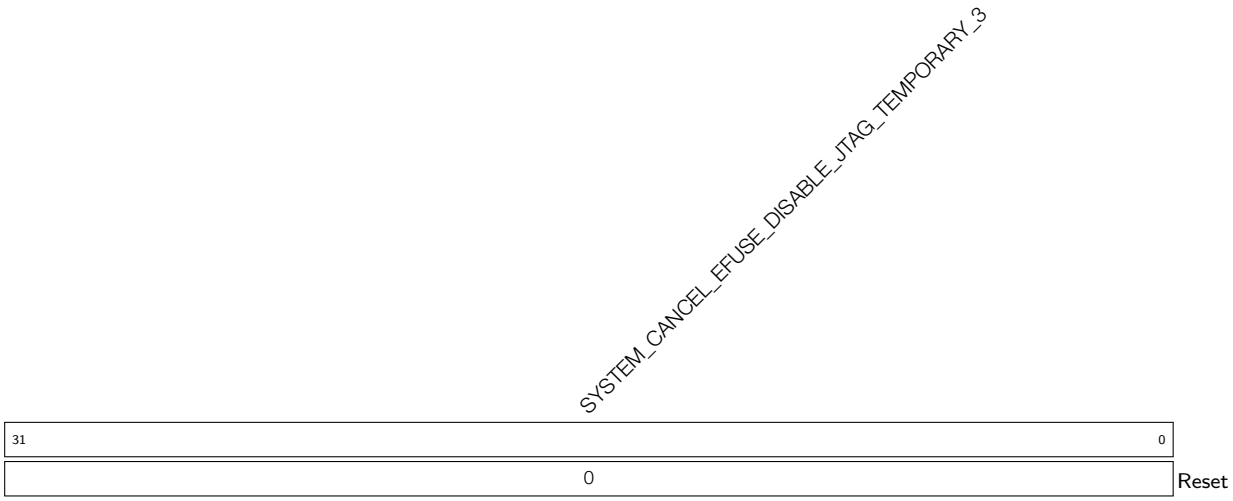
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_1 Stores the 32 to 63 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.10: SYSTEM_JTAG_CTRL_2_REG (0x0024)



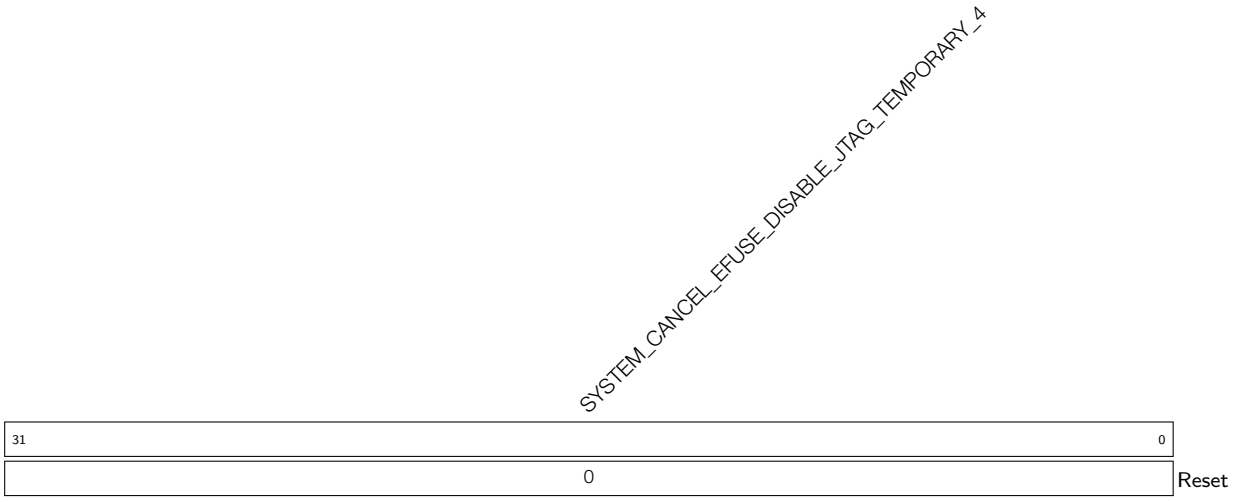
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_2 Stores the 64 to 95 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.11: SYSTEM_JTAG_CTRL_3_REG (0x0028)



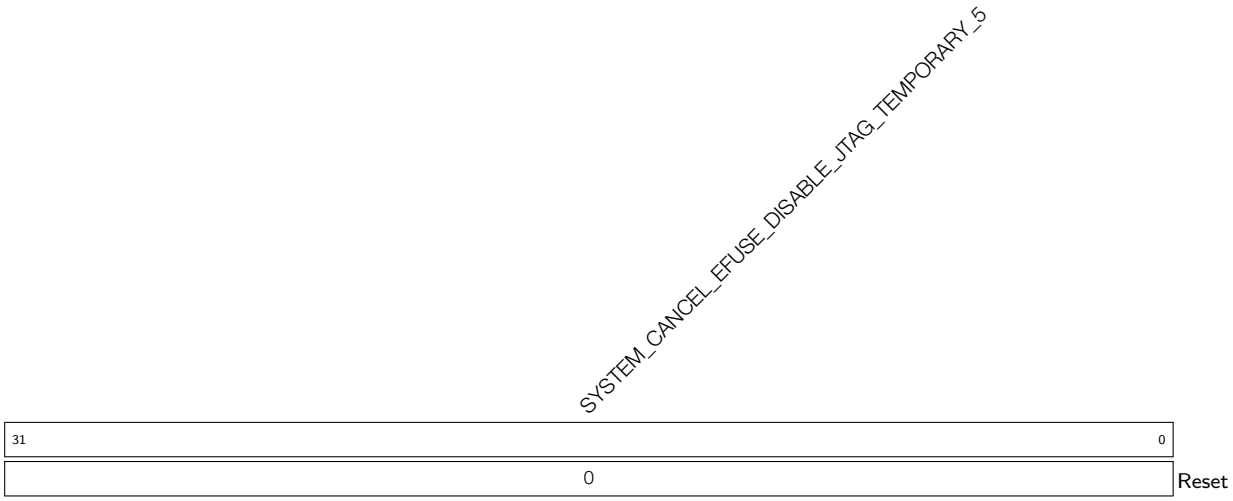
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_3 Stores the 96 to 127 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.12: SYSTEM_JTAG_CTRL_4_REG (0x002C)



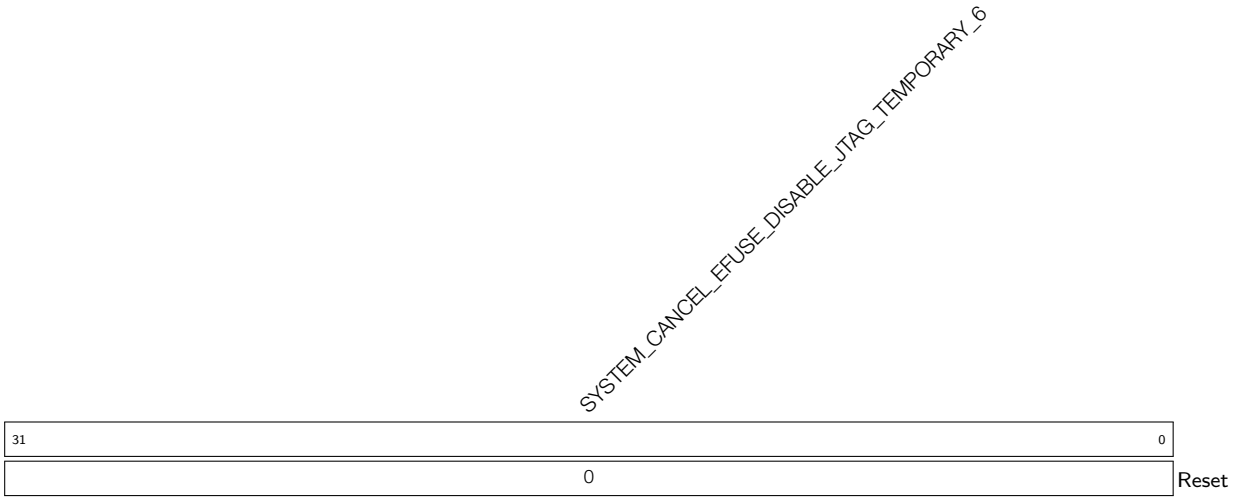
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_4 Stores the 128 to 159 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.13: SYSTEM_JTAG_CTRL_5_REG (0x0030)



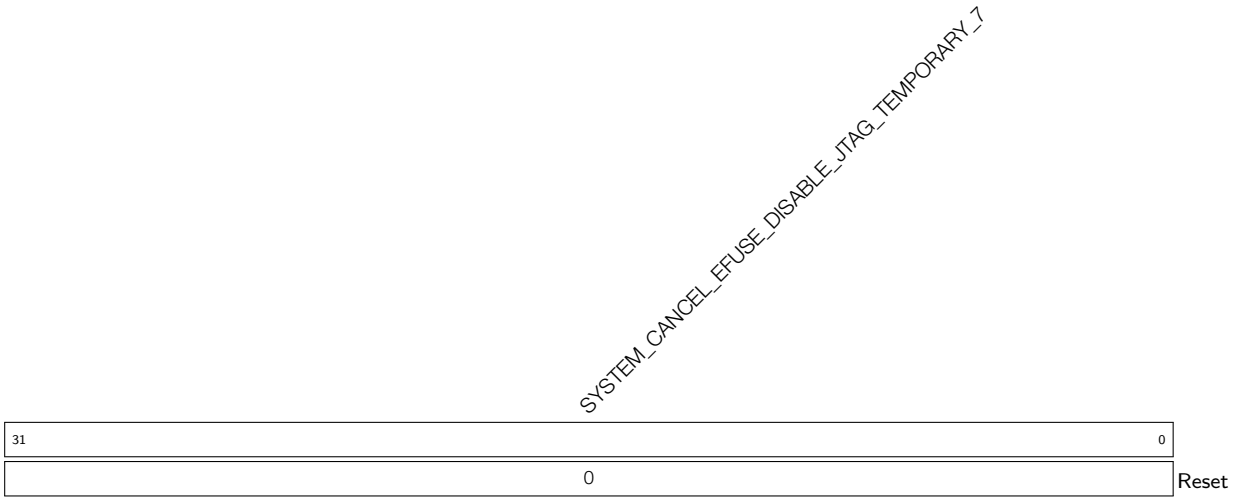
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_5 Stores the 160 to 191 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.14: SYSTEM_JTAG_CTRL_6_REG (0x0034)



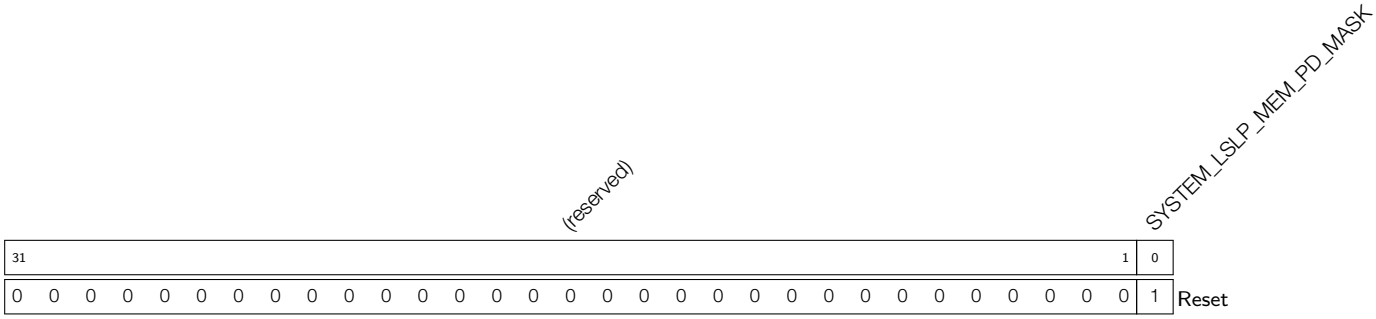
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_6 Stores the 192 to 223 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.15: SYSTEM_JTAG_CTRL_7_REG (0x0038)



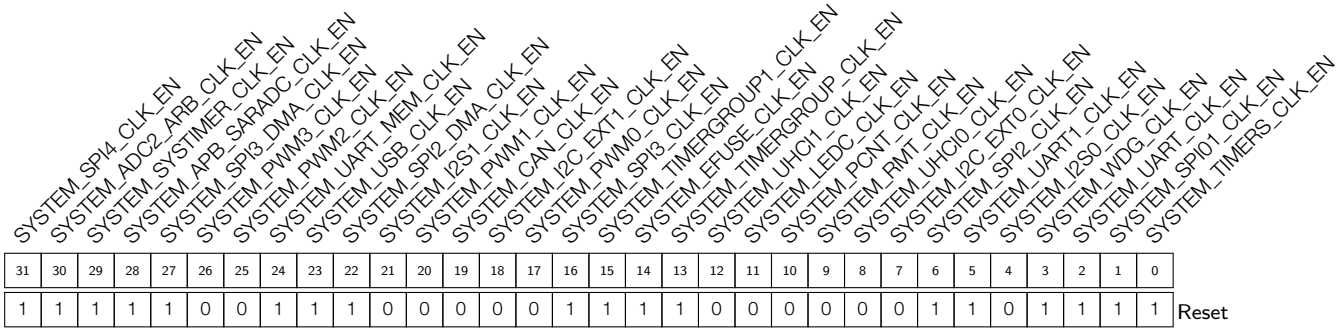
SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7 Stores the 224 to 255 bits of the 256 bits register used to cancel the temporary disable of eFuse to JTAG.

Register 6.16: SYSTEM_MEM_PD_MASK_REG (0x003C)



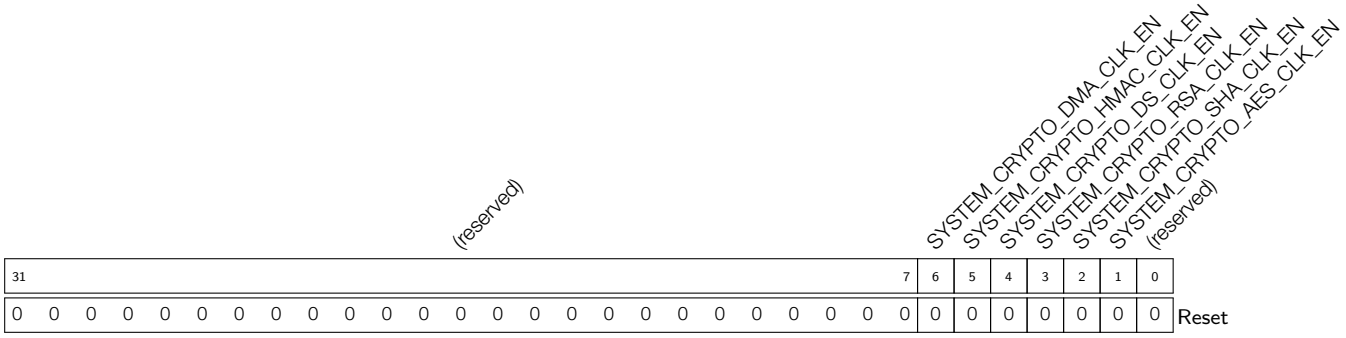
SYSTEM_LSLP_MEM_PD_MASK Set this bit to allow the memory to work as usual when the chip enters the light-sleep state. (R/W)

Register 6.17: SYSTEM_PERIP_CLK_EN0_REG (0x0040)



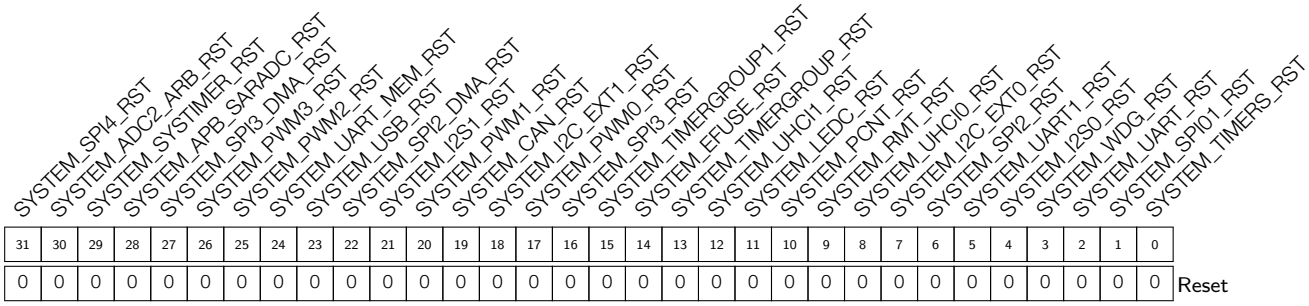
SYSTEM_CPU_PERI_CLK_EN0_REG Configures this register to enable different peripheral clocks. For details, please refer to Table 33.

Register 6.18: SYSTEM_PERIP_CLK_EN1_REG (0x0044)



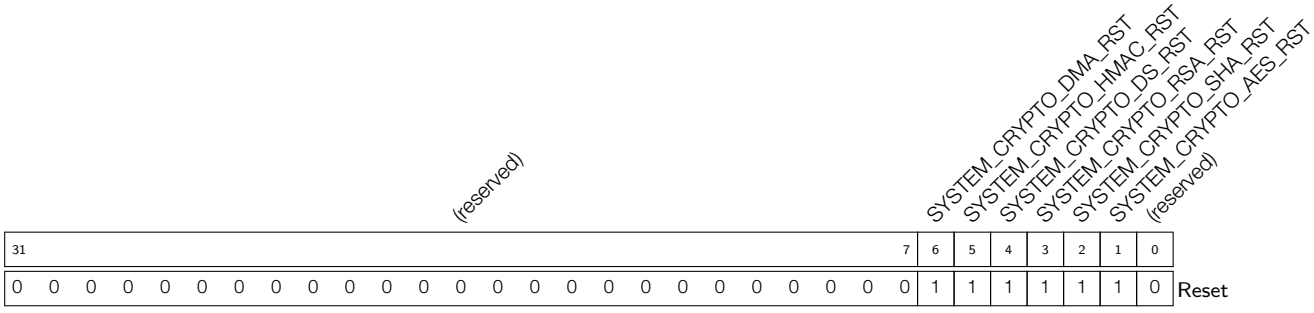
SYSTEM_PERIP_CLK_EN1_REG Configures this register to enable different accelerator clocks. For details, please refer to Table 33.

Register 6.19: SYSTEM_PERIP_RST_EN0_REG (0x0048)



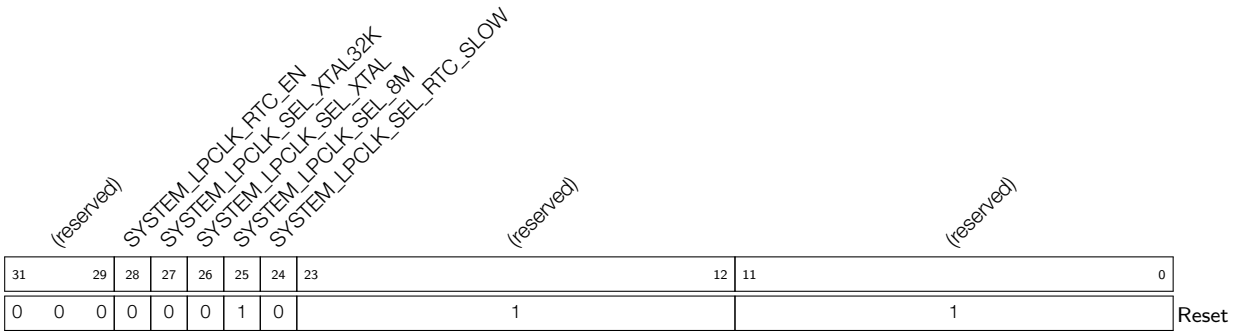
SYSTEM_PERIP_RST_EN0_REG Configures this register to reset different peripherals. For details, please refer to Table 33.

Register 6.20: SYSTEM_PERIP_RST_EN1_REG (0x004C)



SYSTEM_PERIP_RST_EN1_REG Configures this register to reset different accelerators. For details, please refer to Table 33.

Register 6.21: SYSTEM_BT_LPCK_DIV_FRAC_REG (0x0054)



SYSTEM_LPCLK_SEL_RTC_SLOW Set this bit to select RTC slow clock as the low power clock. (R/W)

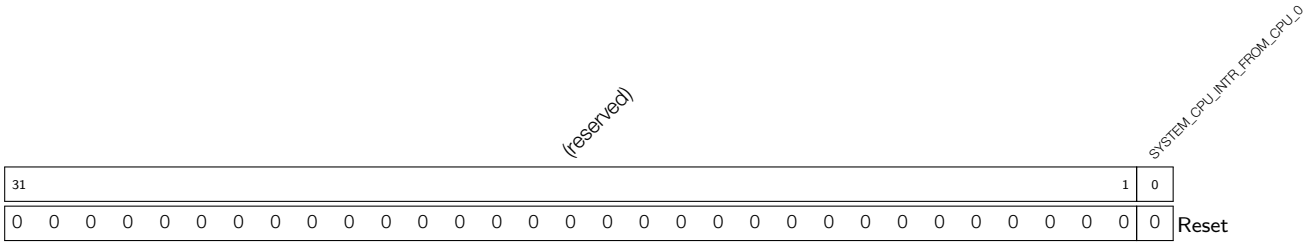
SYSTEM_LPCLK_SEL_8M Set this bit to select 8m clock as the low power clock. (R/W)

SYSTEM_LPCLK_SEL_XTAL Set this bit to select xtal clock as the low power clock. (R/W)

SYSTEM_LPCLK_SEL_XTAL32K Set this bit to select xtal32k clock as the low power clock. (R/W)

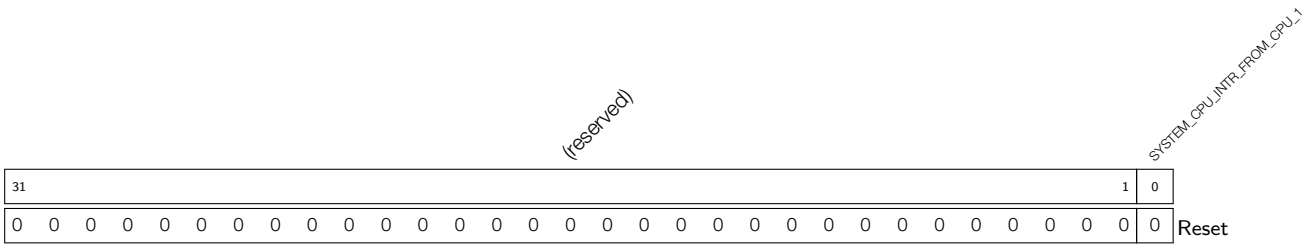
SYSTEM_LPCLK_RTC_EN Set this bit to enable the RTC low power clock. (R/W)

Register 6.22: SYSTEM_CPU_INTR_FROM_CPU_0_REG (0x0058)



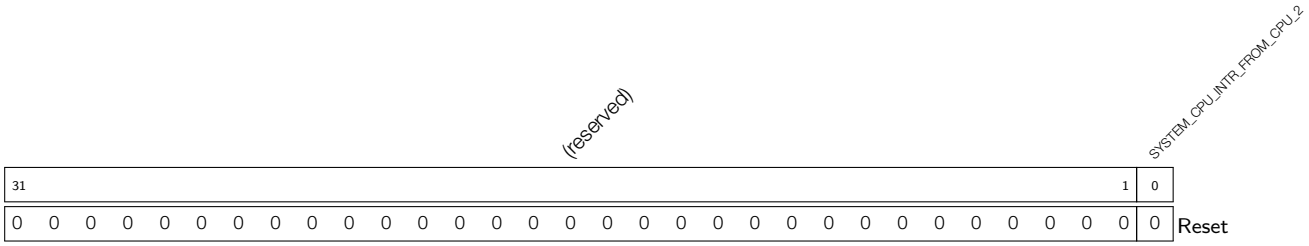
SYSTEM_CPU_INTR_FROM_CPU_0 Set this bit to generate CPU interrupt 0. This bit needs to be reset by software in the ISR process. (R/W)

Register 6.23: SYSTEM_CPU_INTR_FROM_CPU_1_REG (0x005C)



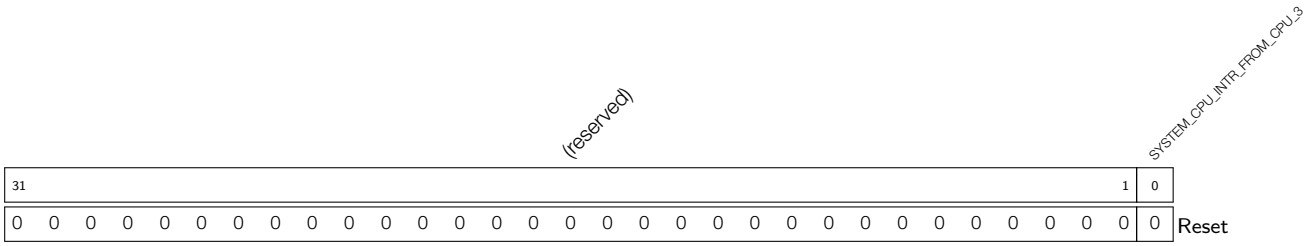
SYSTEM_CPU_INTR_FROM_CPU_1 Set this bit to generate CPU interrupt 1. This bit needs to be reset by software in the ISR process. (R/W)

Register 6.24: SYSTEM_CPU_INTR_FROM_CPU_2_REG (0x0060)



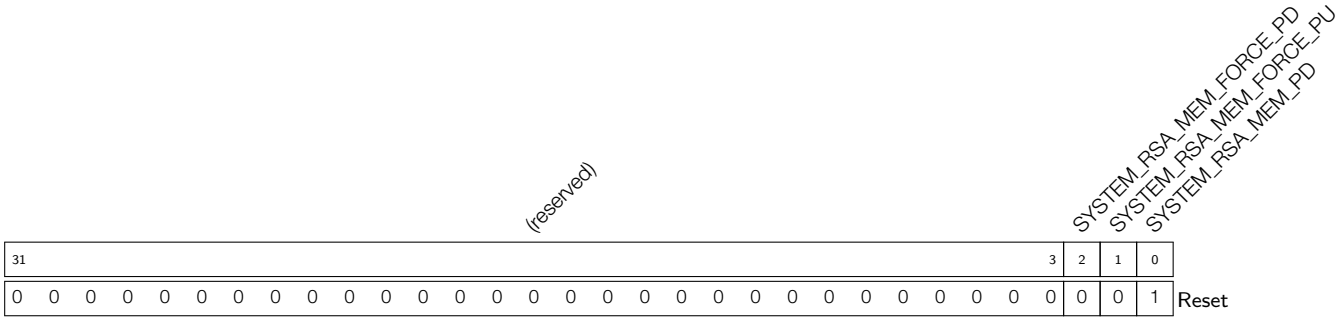
SYSTEM_CPU_INTR_FROM_CPU_2 Set this bit to generate CPU interrupt 2. This bit needs to be reset by software in the ISR process. (R/W)

Register 6.25: SYSTEM_CPU_INTR_FROM_CPU_3_REG (0x0064)



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 6.26: SYSTEM_RSA_PD_CTRL_REG (0x0068)

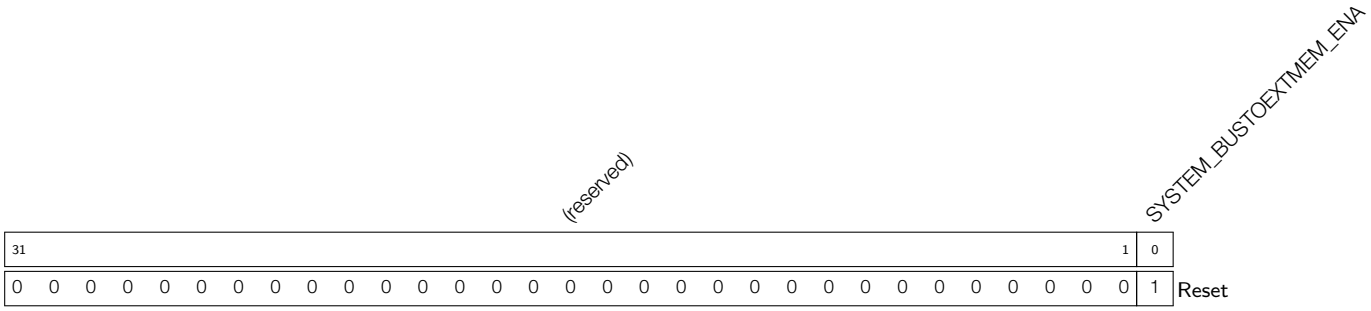


SYSTEM_RSA_MEM_PD Set this bit to power down RSA memory. This bit has the lowest priority. When Digital Signature occupies the RSA, this bit is invalid. (R/W)

SYSTEM_RSA_MEM_FORCE_PU Set this bit to force power up RSA memory. This bit has the second highest priority. (R/W)

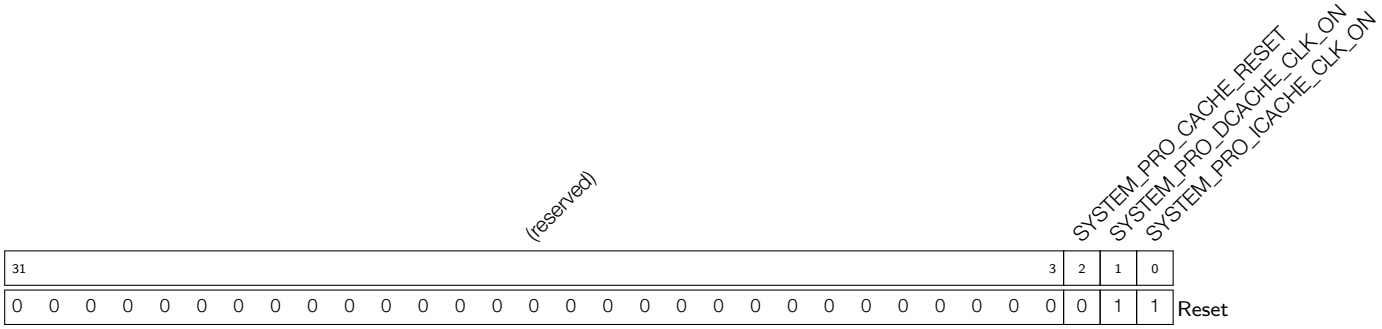
SYSTEM_RSA_MEM_FORCE_PD Set this bit to force power down RSA memory. This bit has the highest priority. (R/W)

Register 6.27: SYSTEM_BUSTOEXTMEM_ENA_REG (0x006C)



SYSTEM_BUSTOEXTMEM_ENA Set this bit to enable bus to EDMA. (R/W)

Register 6.28: SYSTEM_CACHE_CONTROL_REG (0x0070)

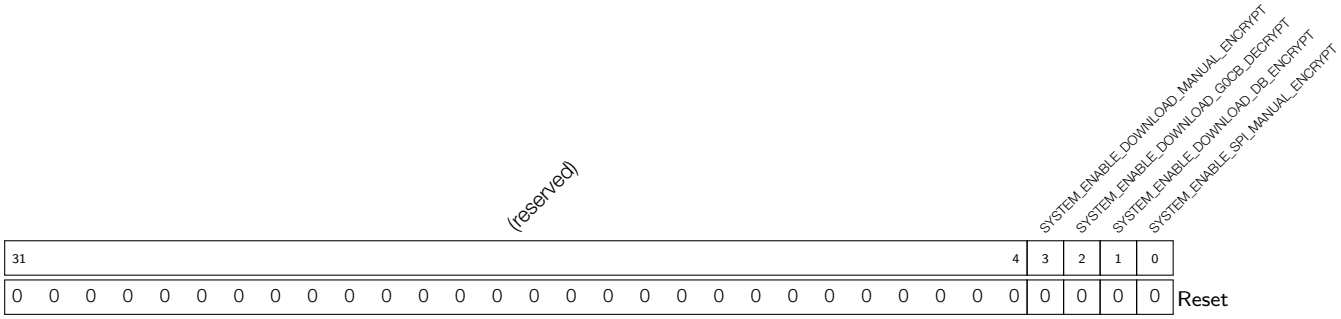


SYSTEM_PRO_ICACHE_CLK_ON Set this bit to enable clock of i-cache. (R/W)

SYSTEM_PRO_DCACHE_CLK_ON Set this bit to enable clock of d-cache. (R/W)

SYSTEM_PRO_CACHE_RESET Set this bit to reset cache. (R/W)

Register 6.29: SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0074)



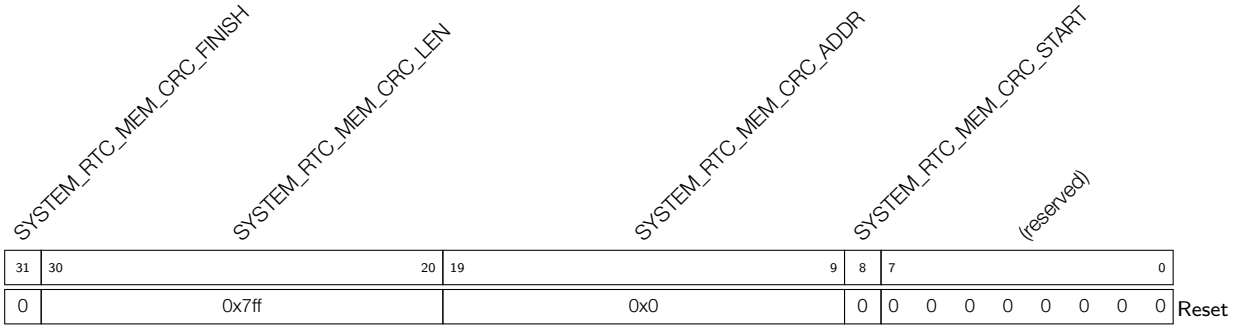
SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT Set this bit to enable Manual Encryption under SPI Boot mode. (R/W)

SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT Set this bit to enable Auto Encryption under Download Boot mode. (R/W)

SYSTEM_ENABLE_DOWNLOAD_G0CB_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 6.30: SYSTEM_RTC_FASTMEM_CONFIG_REG (0x0078)



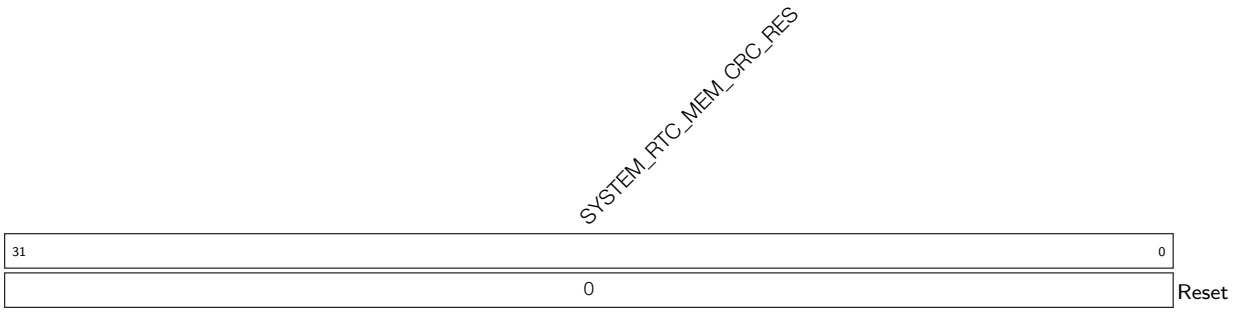
SYSTEM_RTC_MEM_CRC_START Set this bit to start the CRC of RTC memory. (R/W)

SYSTEM_RTC_MEM_CRC_ADDR This field is used to set address of RTC memory for CRC. (R/W)

SYSTEM_RTC_MEM_CRC_LEN This field is used to set length of RTC memory for CRC based on start address. (R/W)

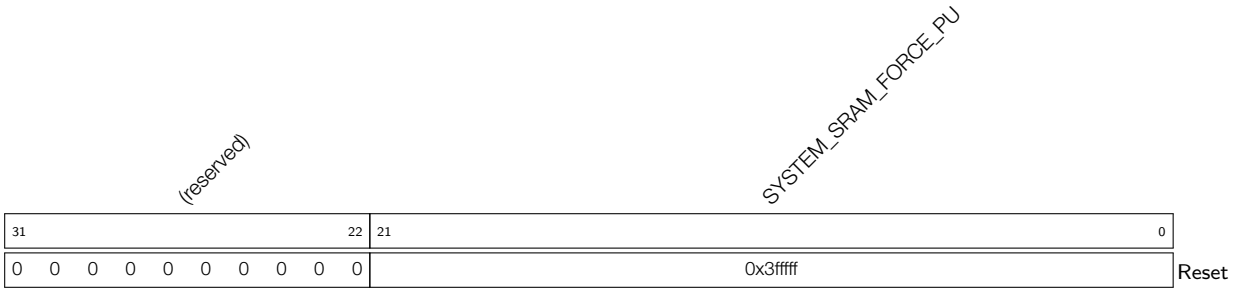
SYSTEM_RTC_MEM_CRC_FINISH This bit stores the status of RTC memory CRC. High level means finished while low level means not finished. (RO)

Register 6.31: SYSTEM_RTC_FASTMEM_CRC_REG (0x007C)



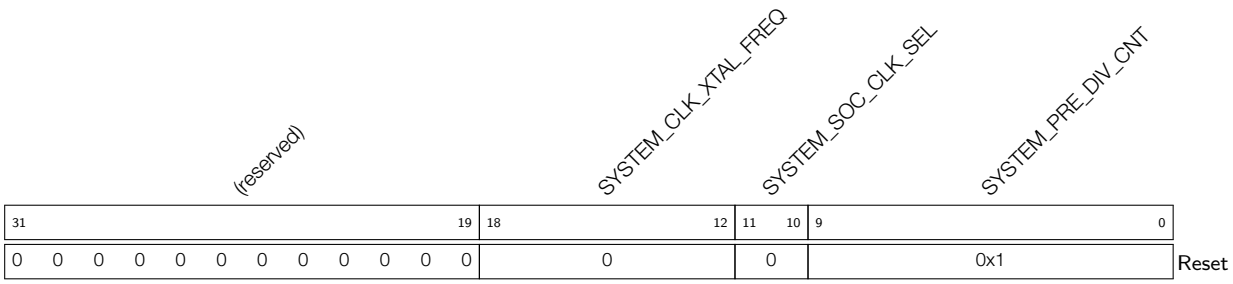
SYSTEM_RTC_MEM_CRC_RES This field stores the CRC result of RTC memory. (RO)

Register 6.32: SYSTEM_SRAM_CTRL_2_REG (0x0088)



SYSTEM_SRAM_FORCE_PU This field is used to power up internal SRAM. For details, please refer to Table 32. (R/W)

Register 6.33: SYSTEM_SYSCLK_CONF_REG (0x008C)

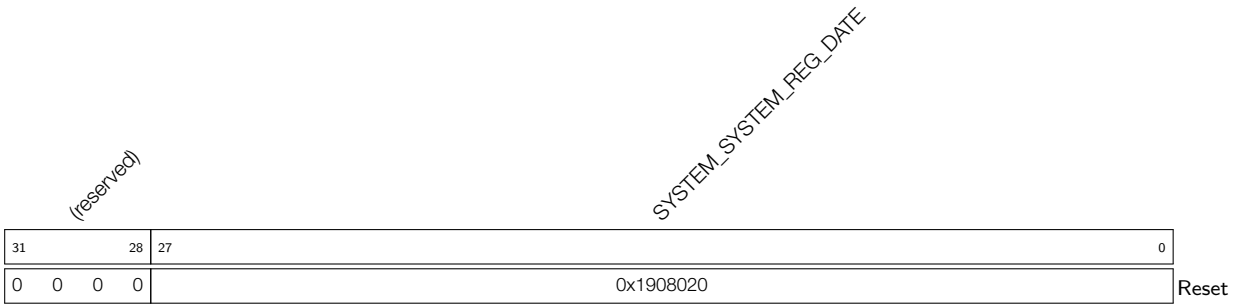


SYSTEM_PRE_DIV_CNT This field is used to set the count of prescaler of XTAL_CLK. For details, please refer to Table 10 in Chapter 2 *Reset and Clock*. (R/W)

SYSTEM_SOC_CLK_SEL This field is used to select SOC clock. For details, please refer to Table 8 in Chapter 2 *Reset and Clock*. (R/W)

SYSTEM_CLK_XTAL_FREQ This field is used to read XTAL frequency in MHz. (RO)

Register 6.34: SYSTEM_REG_DATE_REG (0x0FFC)



SYSTEM_DATE Version control register. (R/W)

7. DMA Controller

7.1 Overview

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

ESP32-S2 has three types of DMA, namely Internal DMA, EDMA and Copy DMA. Internal DMA can only access internal RAM and is used for data transfer between internal RAM and peripherals. EDMA can access both internal RAM and external RAM and is used for data transfer between internal RAM, external RAM and peripherals. Copy DMA can only access internal RAM and is used for data transfer from one location in internal RAM to another.

Eight peripherals on ESP32-S2 have DMA features. As shown in figure 7-1, UART0 and UART1 share one Internal DMA; SPI3 and ADC Controller share one Internal DMA; AES Accelerator and SHA Accelerator share one EDMA; SPI2 and I²S0 have their individual EDMA. Besides, the CPU Peripheral module on ESP32-S2 also has one Copy DMA.

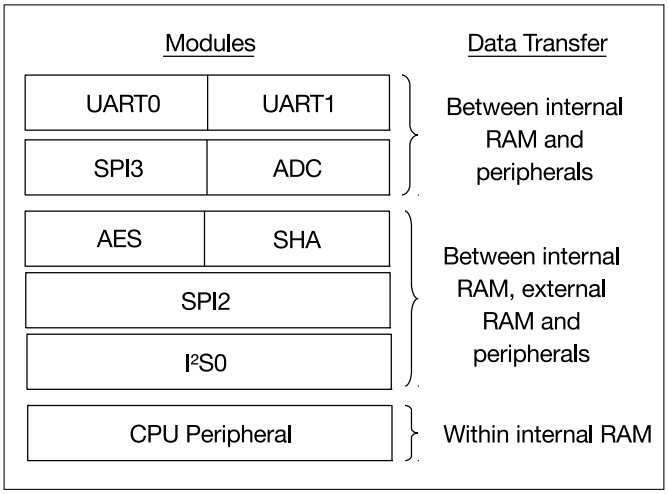


Figure 7-1. Modules with DMA and Supported Data Transfers

7.2 Features

The DMA controller has the following features:

- AHB bus architecture
- Half-duplex and full-duplex mode
- Programmable length of data to be transferred in bytes
- INCR burst transfer when accessing internal RAM
- Access to an address space of 320 KB at most in internal RAM
- Access to an address space of 10.5 MB at most in external RAM
- High-speed data transfer using DMA

7.3 Functional Description

In ESP32-S2, all modules that need high-speed data transfer support DMA. The DMA controller and CPU data bus have access to the same address space in internal RAM and external RAM. DMA controllers for different modules vary in functions according to needs, but their architecture is identical.

7.3.1 DMA Engine Architecture

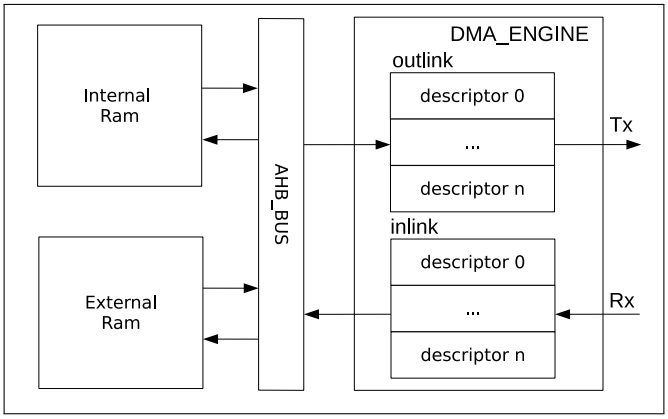


Figure 7-2. DMA Engine Architecture

A DMA engine reads/writes data to/from external RAM or internal RAM via the AHB_BUS. Figure 7-2 shows the basic architecture of a DMA engine. For how to access RAM, please see Chapter 1 *System and Memory*. Software can use the DMA engine through linked lists. The DMA_ENGINE transmits data in corresponding RAM according to the outlink (i.e. a linked list of transmit descriptors), and stores received data into specific address space in RAM according to the inlink (i.e. a linked list of receive descriptors).

7.3.2 Linked List

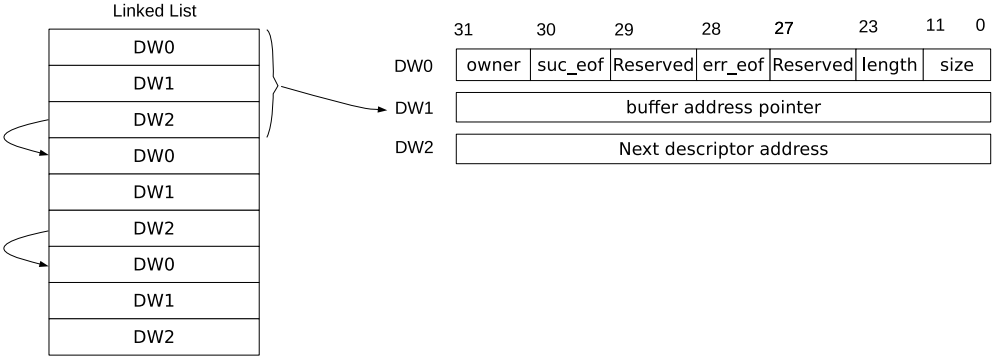


Figure 7-3. Structure of a Linked List

Figure 7-3 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 DMA 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 DMA controller can access the buffer.
  When the DMA controller stops using the buffer, this bit is cleared by hardware. You can set *PERI*_IN_LOOP_TEST bit to disable automatic clearing by hardware. When software loads a linked list, this

bit should be set to 1.
Note: *PERI* refers to modules that support DMA transfers, e.g. I²S, SPI, UHCI, etc.

- suc_eof (DW0) [30]: Specifies whether this descriptor is the last descriptor in the list.
  1'b0: This descriptor is not the last one;
  1'b1: This descriptor is the last one.
  When a packet has been received, this bit in the last receive descriptor is set to 1 by hardware, and this bit in the last transmit descriptor is set to 1 by software.
- Reserved (DW0) [29]: Reserved.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  This bit is used only when UART DMA receives data. When an error is detected in the received packet, 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 bytes have been stored into the buffer.
  When the DMA controller accesses external RAM, this filed must be a multiple of 16/32/64 bytes. Please see more details in Section 7.3.6 *Accessing External RAM*.
- Size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
  When the DMA controller accesses external RAM, this field must be a multiple of 16/32/64 bytes. Please see more details in Section 7.3.6 *Accessing External RAM*.
- Buffer address pointer (DW1): Pointer to the buffer.
  When the DMA controller accesses external RAM, the destination address must be aligned with *PERI*_EXT_MEM_BK_SIZE field. Please see more details in Section 7.3.6 *Accessing External RAM*.
- Next descriptor address (DW2): Pointer to the next descriptor. If the current descriptor is the last one (suc_eof = 1), this value is 0. This field can only point to internal RAM.

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

7.3.3 Enabling DMA

Software uses the DMA controller through linked lists. When the DMA controller receives data, software loads an inlink, configures *PERI*_INLINK_ADDR field with address of the first receive descriptor, and sets *PERI*_INLINK_START bit to enable DMA. When the DMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures *PERI*_OUTLINK_ADDR field with address of the first transmit descriptor, and sets *PERI*_OUTLINK_START bit to enable DMA. *PERI*_INLINK_START bit and *PERI*_OUTLINK_START bit are cleared automatically by hardware.

The DMA controller can be restarted. If you are not sure whether the loaded linked list has been used up or not and want to load a new linked list, you can use this Restart function without affecting the loaded linked list. 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, loads the new list as shown in Figure 7-4, and set *PERI*_INLINK_RESTART bit or *PERI*_OUTLINK_RESTART bit (these two bits are cleared automatically by hardware). By doing so, hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and

then read the new list.

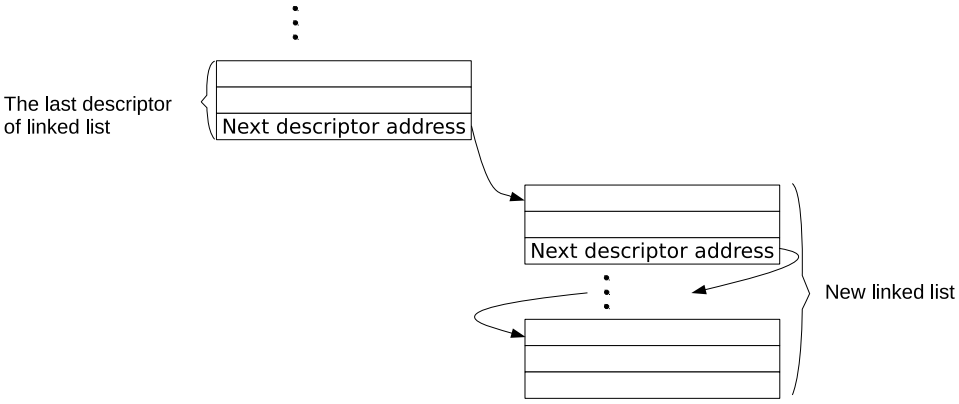


Figure 7-4. Relationship among Linked Lists

7.3.4 Linked List reading process

Once configured and enabled by software, the DMA controller starts to read the linked list from internal RAM. *PERI*_IN_DSCR_ERR_INT_ENA bit or *PERI*_OUT_DSCR_ERR_INT_ENA bit can be set to enable descriptor error interrupt. If the buffer address pointer (DW1) does not point to 0x3FFB0000 ~ 0x3FFFFFFF when the DMA controller accesses internal RAM, or does not point to 0x3F500000 ~ 0x3FF7FFFF when the DMA controller accesses external RAM, a descriptor error interrupt is generated.

Note: The third word (DW2) in a descriptor can only point to internal RAM; it points to the next descriptor to use and descriptors must be in internal memory.

7.3.5 EOF

The DMA controller uses EOF (end of file) flags to indicate the completion of data transfer.

Before the DMA controller transmits data, *PERI*_OUT_TOTAL_EOF_INT_ENA bit should be set. If data in the buffer pointed by the last descriptor has been transmitted, a *PERI*_OUT_TOTAL_EOF_INT interrupt is generated.

Before the DMA controller receives data, *PERI*_IN_SUC_EOF_INT_ENA bit should be set. If data has been received successfully, a *PERI*_IN_SUC_EOF_INT interrupt is generated. In addition to *PERI*_IN_SUC_EOF_INT interrupt, UART DMA also supports UHCI_IN_ERR_EOF_INT. This interrupt is enabled by setting UHCI_IN_ERR_EOF_INT_ENA bit, and it indicates that a data packet has been received with errors.

When a *PERI*_OUT_TOTAL_EOF_INT or a *PERI*_IN_SUC_EOF_INT interrupt is detected, software can record the value of field *PERI*_OUTLINK_DSCR_ADDR or *PERI*_INLINK_DSCR_ADDR, i.e. address of the last descriptor (right shifted 2 bits). Therefore, software can tell which descriptors have been used and reclaim them.

7.3.6 Accessing External RAM

Of all DMA controllers for ESP32-S2 peripherals, only I^2S0, SPI2, AES and SHA DMA have access to external RAM. The address space of external RAM that the DMA controller can access is 0x3F500000 ~ 0x3FF7FFFF. Please note that destination addresses (in this case addresses for writing data to external RAM) must be 16-byte, 32-byte or 64-byte aligned. Table 36 illustrates the value of *PERI*_EXT_MEM_BK_SIZE bit when destination address is 16-byte, 32-byte and 64-byte aligned respectively.

Note: Source addresses (in this case, namely addresses for reading data from external RAM) do not need to be aligned.

Table 36: Relationship Between Configuration Register and Destination Address

*PERI*_EXT_MEM_BK_SIZE	Destination address alignment
0	16-bit aligned
1	32-bit aligned
2	64-bit aligned

7.4 Copy DMA Controller

Copy DMA is used for data transfer from one location in internal RAM to another. Figure 7-5 shows the architecture of a Copy DMA engine. Unlike Internal DMA and EDMA, Copy DMA first reads data to be transferred from internal RAM, stores the data into the DMA FIFO via an outlink, and then writes the data to the target internal RAM via an inlink.

Copy DMA should be configured by software as follows:

1. Set CP_DMA_IN_RST, CP_DMA_OUT_RST, CP_DMA_FIFO_RST and CP_DMA_CMDFIFO_RST bit first to 1 and then to 0, to reset Copy DMA state machine and FIFO pointer;
2. Set CP_DMA_FIFO_RST bit first to 1 and then to 0, to reset FIFO pointer;
3. Load an outlink, and configure CP_DMA_OUTLINK_ADDR with address of the first transmit descriptor;
4. Load an inlink, and configure CP_DMA_INLINK_ADDR with address of the first receive descriptor;
5. Set CP_DMA_OUTLINK_START to enable DMA transmission;
6. Set CP_DMA_INLINK_START to enable DMA reception.

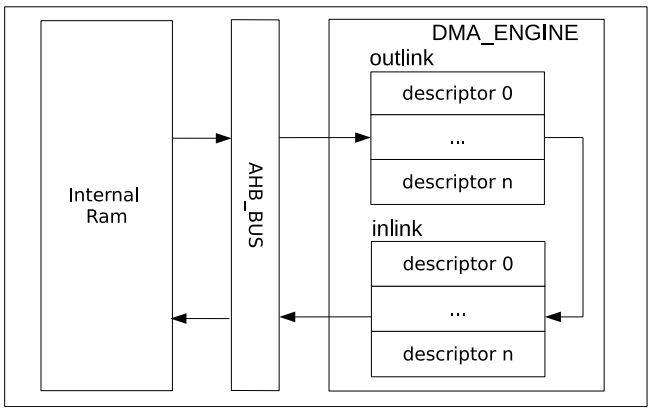


Figure 7-5. Copy DMA Engine Architecture

7.5 UART DMA (UDMA) Controller

ESP32-S2 has two UART controllers. They share one UDMA controller. UHCI_UART_CE specifies which UART controller gets access to UDMA.

Figure 7-6 shows how data is transferred using UDMA. Before UDMA receives data, software prepares an inlink. UHCI_INLINK_ADDR points to the first receive descriptor in the inlink. After UHCI_INLINK_START is set, UHCI sends data that UART has received to the Decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of UDMA.

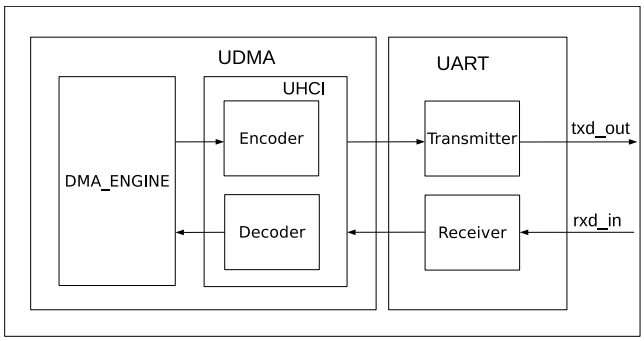


Figure 7-6. Data Transfer in UDMA Mode

Before UDMA sends data, software prepares an outlink and data to be sent. UHCI_OUTLINK_ADDR points to the first transmit descriptor in the outlink. After UHCI_OUTLINK_START is set, UDMA reads data from the RAM pointed by outlink. The data is then encoded by the Encoder, and sent sequentially by the UART transmitter.

Data packets of UDMA have separators at the beginning and the end, with data bits in the middle. The encoder inserts separators in front of and after data bits, and replaces data bits identical to separators with special characters. The decoder removes separators in front of and after data bits, and replaces special characters with separators. There can be more than one continuous separator at the beginning and the end of a data packet. The separator is configured by UHCI_SEPER_CHAR, 0xC0 by default. The special character is configured by UHCI_ESC_SEQ0_CHAR0 (0xDB by default) and UHCI_ESC_SEQ0_CHAR1 (0xDD by default). When all data has been sent, a UHCI_OUT_TOTAL_EOF_INT interrupt is generated. When all data has been received, a UHCI_IN_SUC_EOF_INT is generated.

7.6 SPI DMA Controller

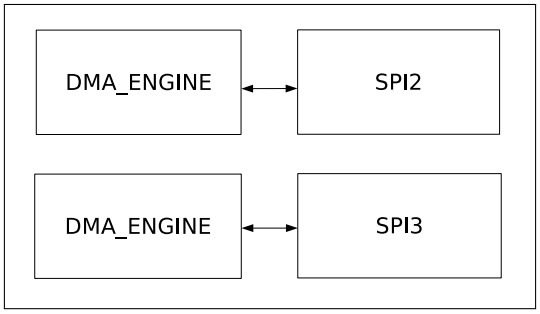


Figure 7-7. SPI DMA

As shown in Figure 7-7, SPI2 and SPI3 have separate DMA controllers.

SPI DMA receives and transmits data through descriptors at least one byte at a time. The transmission of data can be done in bursts.

SPI_OUTLINK_START bit of SPI_DMA_OUT_LINK_REG register and SPI_INLINK_START bit of SPI_DMA_IN_LINK_REG register are used to enable the DMA engine and are cleared by hardware. When SPI_OUTLINK_START is set to 1, the DMA engine loads an outlink and prepares data to be transferred; when SPI_INLINK_START is set to 1, the DMA engine loads an inlink and prepares to receive data.

When receiving data, SPI DMA should be configured by software as follows:

1. Set SPI_DMA_IN_RST, SPI_AHBM_FIFO_RST and SPI_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an inlink, and configure SPI_DMA_INLINK_ADDR with address of the first receive descriptor;
3. Set SPI_DMA_INLINK_START to enable DMA reception.

When transmitting data, SPI DMA should be configured by software as follows:

1. Set SPI_DMA_OUT_RST, SPI_AHBM_FIFO_RST and SPI_AHBM_RST bit first to 1 and then to 0, to reset RMA state machine and FIFO pointer;
2. Load an outlink, and configure SPI_DMA_OUTLINK_ADDR with address of the first transmit descriptor;
3. Set SPI_DMA_OUTLINK_START to enable DMA transmission.

Note: When SPI DMA transfers data between internal RAM and external RAM, SPI_MEM_TRANS_EN should be set.

SPI DMA also supports data transfer in segments.

7.7 I²S DMA Controller

ESP32-S2 I²S has an individual DMA. I2S_DSCR_EN bit of I2S_FIFO_CONF_REG register is used to enable DMA transfer of I²S. I²S DMA receives and transmits data through linked lists. The transmission of data can be done in bursts. I2S_RX_EOF_NUM[31:0] bit of I2S_RXEOF_NUM_REG register is used to configure how many words of data to be received at a time.

I2S_OUTLINK_START bit of I2S_OUT_LINK_REG and I2S_INLINK_START bit of I2S_IN_LINK_REG register are used to enable the DMA engine and are cleared by hardware. When I2S_OUTLINK_START bit is set to 1,the DMA engine loads an outlink and prepares data to be transferred; when I2S_INLINK_START is set to 1,the DMA engine loads an inlink and prepares to receive data.

When receiving data, I²S DMA should be configured by software as follows:

1. Set I2S_IN_RST, I2S_AHBM_FIFO_RST and I2S_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an inlink, and configure I2S_INLINK_ADDR with address of the first receive descriptor;
3. Set I2S_INLINK_START to enable DMA reception.

When transmitting data, I²S DMA should be configured by software as follows:

1. Set I2S_OUT_RST, I2S_AHBM_FIFO_RST and I2S_AHBM_RST bit first to 1 and then to 0, to reset DMA state machine and FIFO pointer;
2. Load an outlink, and configure I2S_OUTLINK_ADDR with address of the first transmit descriptor;
3. Set I2S_OUTLINK_START to enable DMA transmission.

Note: When I²S DMA transfers data between internal RAM and external RAM using I²S DMA, I2S_MEM_TRANS_EN should be set.

8. UART Controller

8.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full duplex mode. ESP32-S2 has two UART controllers compatible with various UART devices. They support Infrared Data Association (IrDA) and RS-485 transmission.

ESP32-S2 has two UART controllers. Each 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.

8.2 Features

Each UART controller has the following features:

- Programmable baud rate
- 512 x 8-bit RAM shared by TX FIFOs and RX FIFOs of two UART controllers
- Full duplex asynchronous communication
- Automatic baud rate detection
- Data bits ranging from 5 to 8
- Stop bits whose length can be 1, 1.5, 2 or 3 bits
- Parity bits
- Special character AT_CMD detection
- RS-485 protocol
- IrDA protocol
- High-speed data communication using DMA
- UART as wake-up source
- Software and hardware flow control

8.3 Functional Description

8.3.1 UART Introduction

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to 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 and parity bit.

A UART data packet usually begins with one start bit, followed by data bits, one parity bit (optional) and one or more stop bits. UART controllers on ESP32-S2 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as DMA for seamless high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

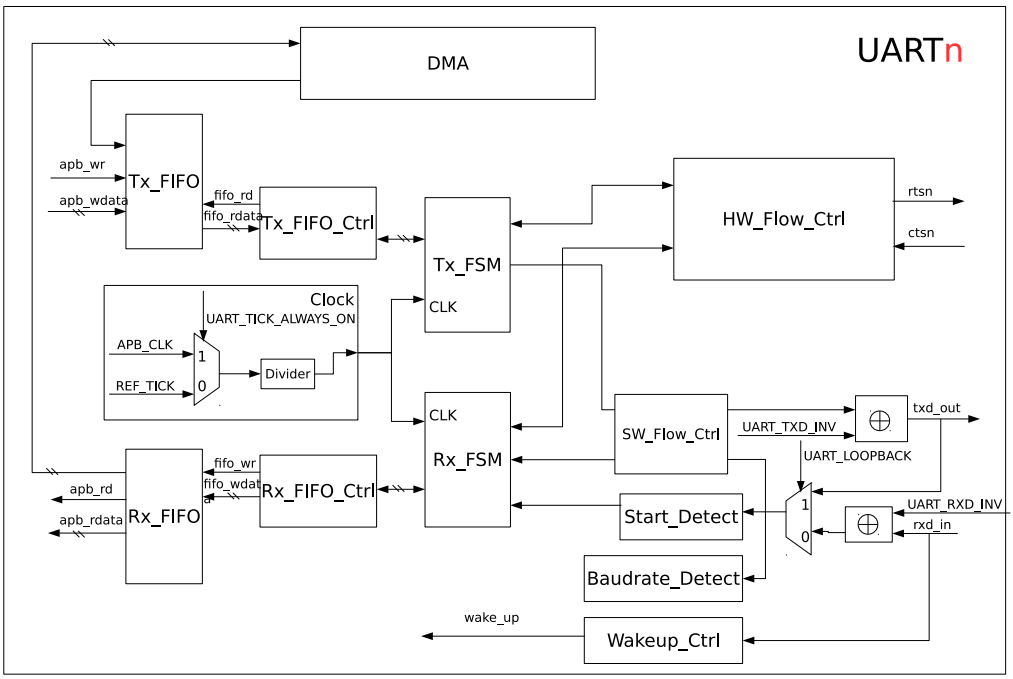


Figure 8-1. UART Structure

8.3.2 UART Structure

Figure 8-1 shows the basic structure of a UART controller. It has two possible clock sources: a 80 MHz APB_CLK and a reference clock REF_TICK (for details, please refer to Chapter 2 *Reset and Clock*), which are selected by configuring UART_TICK_REF_ALWAYS_ON. The selected clock source is divided by a divider to generate clock signals that drive the UART controller. 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, or move data to Tx_FIFO using DMA. Tx_FIFO_Ctrl controls writing and reading Tx_FIFO. When Tx_FIFO is not empty, Tx_FSM reads bytes via Tx_FIFO_Ctrl, and converts them into a bitstream. The levels of output signal txd_out can be inverted by configuring UART_TXD_INV register.

The receiver contains a RX FIFO, which buffers data to be processed. Software can read data from Rx_FIFO via the APB bus, or receive data using DMA. The levels of input signal rxd_in can be inverted by configuring UART_RXD_INV register, and the signal is then input to the Rx components of the UART Controller: 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.

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 the Light-sleep mode, Wakeup_Ctrl counts up rising edges of rxd_in. When the number reaches (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC, which then wakes up the ESP32-S2 chip.

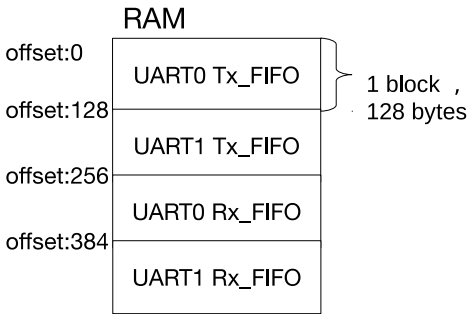


Figure 8-2. UART Controllers Sharing RAM

8.3.3 UART RAM

The two UART controllers on ESP32-S2 share 512 × 8 bits of FIFO RAM. As figure 8-2 illustrates, RAM is divided into 4 blocks, each has 128 × 8 bits. Figure 8-2 shows by default how many RAM blocks are allocated to TX FIFOs and RX FIFOs of the two UART controllers. 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. The size of UART0 Tx_FIFO can be increased to 4 blocks (the whole RAM), the size of UART1 Tx_FIFO can be increased to 3 blocks (from offset 128 to the end address), the size of UART0 Rx_FIFO can be increased to 2 blocks (from offset 256 to the end address), but the size of UART1 Rx_FIFO cannot be increased. Please note that 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 255). 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 or using DMA, read automatically and converted from a frame into a bitstream by hardware Tx_FSM; data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using DMA. The two UART controllers share one DMA controller.

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.

8.3.4 Baud Rate Generation and Detection

8.3.4.1 Baud Rate Generation

Before a UART controller sends or receives data, the baud rate should be configured by setting corresponding registers. A UART Controller baud rate generator 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 an 80 MHz input clock, the UART controller supports a maximum baud rate of 5 Mbaud.

The divisor of the baud rate divider is equal to UART_CLKDIV + (UART_CLKDIV_FRAG / 16), meaning that the final baud rate is equal to INPUT_FREQ / (UART_CLKDIV + (UART_CLKDIV_FRAG / 16)). For example, if UART_CLKDIV = 694 and UART_CLKDIV_FRAG = 7 then the divisor value is (694 + 7/16) = 694.4375. If the input clock frequency is 80MHz APB_CLK, the baud rate will be (80MHz / 69.4375) = 115201.

When UART_CLKDIV_FRAG is zero, 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 zero, the divider is fractional and the output baud rate clock pulses are not strictly uniform. As shown in figure 8-3, 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 8-3 below, to make the output timing more uniform:

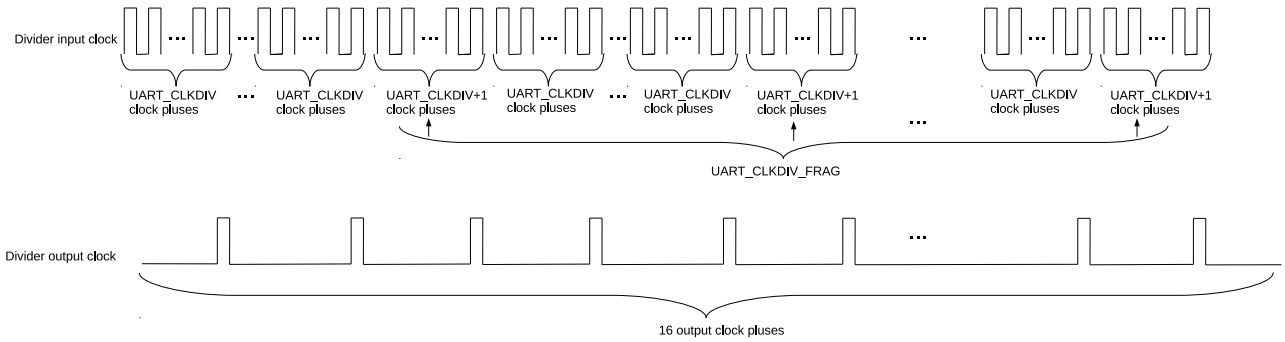


Figure 8-3. UART Controllers Division

To support IrDA (see Section 8.3.7 *IrDA* for details), the fractional clock divider for IrDA data transmission generates clock signals divided by 16× 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.

8.3.4.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in figure 8-1 will measure pulse widths while filtering any noise whose pulse width is shorter than UART_GLITCH_FILT.

Before communication starts, the transmitter could send random data to the receiver for baud rate detection. UART_LOWPULSE_MIN_CNT stores the minimum low pulse width, UART_HIGHPULSE_MIN_CNT stores the minimum high pulse width, UART_POSEDGE_MIN_CNT stores the minimum pulse width between two rising edges, and UART_NEGEDGE_MIN_CNT stores the minimum pulse width between two falling edges. These four registers are read by software to determine the transmitter's baud rate.

Baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or falling edges in semi-stable state, which results in inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a weighted average of these

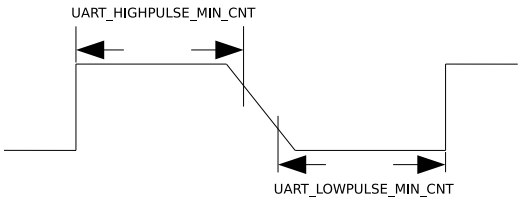


Figure 8-4. The Timing Diagram of Weak UART Signals Along Falling Edges

two values to eliminate errors. In this case, 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. If UART signals are weak along falling edges as shown in figure 8-4, which leads to 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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{\text{UART\_POSEDGE\_MIN\_CNT}/2}$$

3. If UART signals are weak along rising edges, use UART_NEGEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{\text{UART\_NEGEDGE\_MIN\_CNT}/2}$$

8.3.5 UART Data Frame

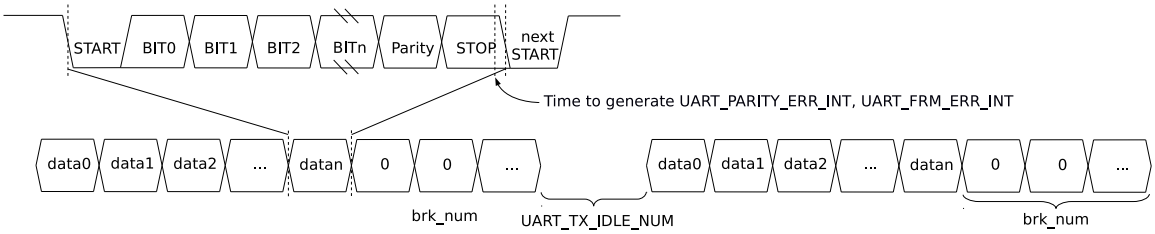


Figure 8-5. Structure of UART Data Frame

Figure 8-5 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, 2 or 3 bits long, configured by UART_STOP_BIT_NUM, UART_DL1_EN and UART_DL0_EN. 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 data received, a UART_PARITY_ERR_INT interrupt is generated and data received is still stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and 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 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.

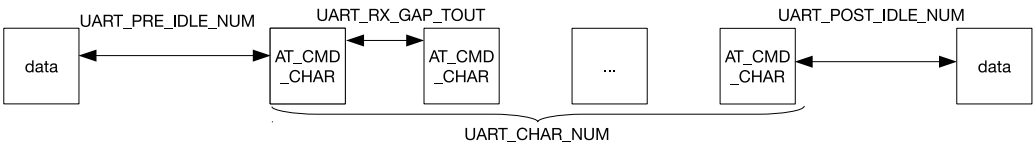


Figure 8-6. AT_CMD Character Structure

Figure 8-6 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 single-bit cycles.
- The interval between two AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT single-bit 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 single-bit cycles.

8.3.6 RS485

The two UART controllers support RS485 standard. This standard 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.

8.3.6.1 Driver Control

As shown in figure 8-7, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion. A RS485 transceiver contains a driver and a receiver. When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver. When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

The receiving UART converts differential signals to single-ended signals via an external receiver. RE is the enable control signal for the receiver. When RE is 0, the receiver is enabled; when RE is 1, the receiver is disabled. If RE is configured as 0, the UART controller is allowed to snoop data on the bus, including data sent by itself.

DE can be controlled by either software or hardware. To reduce cost of software, in our design DE is controlled by hardware. As shown in figure 8-7, DE is connected to dtrn_out of UART (please refer to Section8.3.9.1 *Hardware Flow Control* for more details).

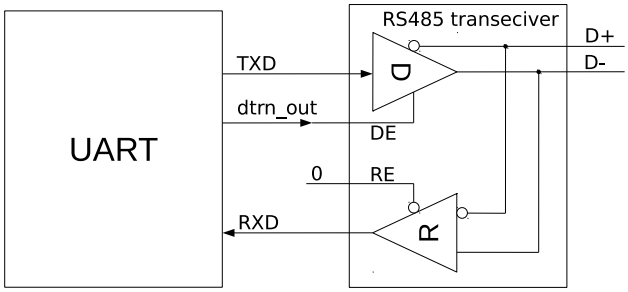


Figure 8-7. Driver Control Diagram in RS485 Mode

8.3.6.2 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 at least one cycle after the stop bit. The transmitter supports turnaround delay of two cycles added after the stop bit. When UART_DL1_EN is set, turnaround delay of one single-bit cycle is added; when UART_DL0_EN is set, turnaround delay of a second cycle is added.

8.3.6.3 Bus Snooping

By default, an RS485 device is not allowed to transmit and receive data simultaneously. However, the UART controller peripheral supports snooping this bus by receiving while transmitting. If UART_RS485TX_RX_EN is set and the external RS485 transceiver is configured as in figure 8-7, 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 data sent by themselves. In transmitter mode, when a UART controller monitors a collision between data sent and data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitor a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a polarity error, a UART_RS485_PARITY_ERR_INT is generated.

8.3.7 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 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 8-8, the IrDA encoder converts a NRZ (non-return to zero code) signal to a RZI (return to zero code) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits to indicate logic “0”, and low levels to indicate logic “1”. The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the encoder output polarity is the opposite of the decoder input polarity. If a low pulse is detected, it indicates that a start bit has been received.

When IrDA function is enabled, one bit is divided into 16 clock cycles. If the bit to be sent is zero, then the 9th, 10th and 11th clock cycle is high.

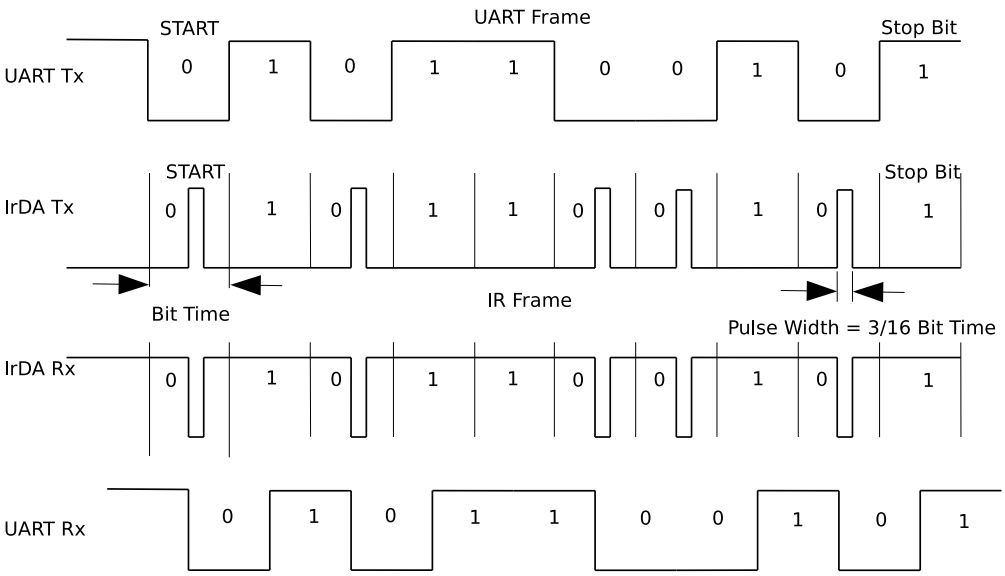


Figure 8-8. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in figure 8-9, 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.

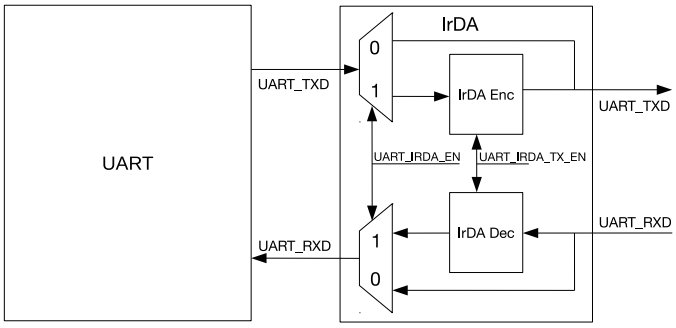


Figure 8-9. IrDA Encoding and Decoding Diagram

8.3.8 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 greater than (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC, which then wakes up ESP32-S2.

8.3.9 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 data flow sent and detecting special characters in data flow received.

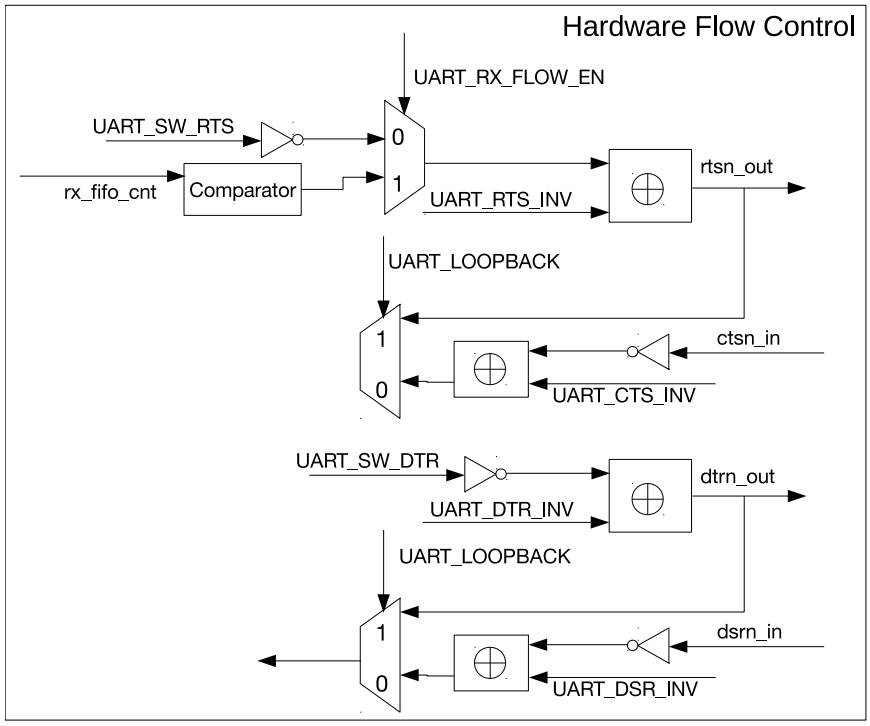


Figure 8-10. Hardware Flow Control Diagram

8.3.9.1 Hardware Flow Control

Figure 8-10 shows hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 8-11 illustrates how these signals are connected between ESP32-S2 UART (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

When rtsn_out of IU0 is low, EU0 is allowed to send data; when rtsn_out of IU0 is high, EU0 is notified to stop sending data until rtsn_out of IU0 returns to low. Output signal rtsn_out can be controlled in two ways.

- Software control: Enter this mode by setting 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.

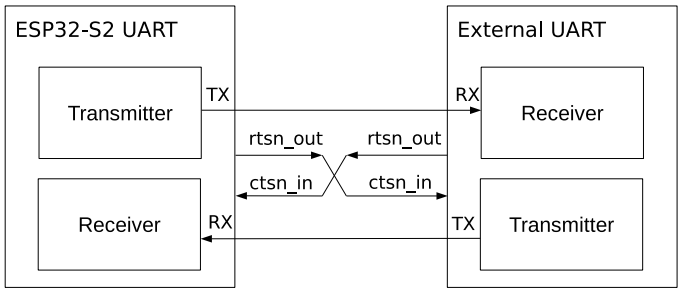


Figure 8-11. Connection between Hardware Flow Control Signals

When ctsn_in of IU0 is low, IU0 is allowed to send data; when ctsn_in is high, IU0 is not allowed to send data. When IU0 detects an edge change of ctsn_in, a UART_CTS_CHG_INT interrupt is generated.

If dtrn_out of IU0 is high, it indicates that IU0 is ready to transmit data. dtrn_out is generated by configuring register UART_SW_DTR. When the IU0 transmitter detects a edge change of dsrn_in, a UART_DSR_CHG_INT

interrupt is generated. After this interrupt is detected, software can obtain the level of input signal dsrn_in by reading UART_DSRN. If dsrn_in is high, it indicates that EU0 is ready to transmit data.

In a two-wire RS485 multidrop network enabled by setting UART_RS485_EN, dtrn_out is used for transmit/receive turnaround. In this case, dtrn_out is generated by hardware. 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 turnaround delay of one cycle added after the stop bit, dtrn_out is pulled low after the delay.

UART loopback test is enabled by setting UART_LOOPBACK. In the test, UART output signal txd_out is connected to its input signal rxd_in, rtsn_out is connected to ctsn_in, and dtrn_out is connected to dsrn_out. If data sent matches data received, it indicates that UART controllers are working properly.

8.3.9.2 Software Flow Control

Instead of CTS/RTS lines, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting UART_SW_FLOW_CON_EN to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in 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 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 after the current byte in transmission.

8.3.10 UDMA

The two UART controllers on ESP32-S2 share one UDMA (UART DMA), which supports the decoding and encoding of HCI data packets. For more information, please refer to Chapter 7: *DMA Controller*.

8.3.11 UART Interrupts

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.

- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter sends a NULL character after all data in TX FIFO has 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 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 data frame error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

8.3.12 UHCI Interrupts

- UHCI_DMA_INFIFO_FULL_WM_INT: Triggered when the counter value of DMA RX FIFO exceeds UHCI_DMA_INFIFO_FULL_THRS.
- UHCI_SEND_A_REG_Q_INT: Triggered when DMA has sent a series of short packets using always_send.
- UHCI_SEND_S_REG_Q_INT: Triggered when DMA has sent a series of short packets using single_send.
- UHCI_OUT_TOTAL_EOF_INT: Triggered when all data has been sent.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_IN_DSCR_EMPTY_INT: Triggered when there are not enough receive descriptors for DMA.
- UHCI_OUT_DSCR_ERR_INT: Triggered when an error is detected in a transmit descriptor.
- UHCI_IN_DSCR_ERR_INT: Triggered when an error is detected in an receive descriptor.
- UHCI_OUT_EOF_INT: Triggered when the EOF bit in a descriptor is 1.
- UHCI_OUT_DONE_INT: Triggered when a transmit descriptor is completed.
- UHCI_IN_ERR_EOF_INT: Triggered when an EOF error is detected in an receive descriptor.

- UHCI_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- UHCI_IN_DONE_INT: Triggered when an receive descriptor is completed.
- UHCI_TX_HUNG_INT: Triggered when DMA spends too much time on reading RAM.
- UHCI_RX_HUNG_INT: Triggered when DMA spends too much time on receiving data.
- UHCI_TX_START_INT: Triggered when DMA detects a separator character.
- UHCI_RX_START_INT: Triggered when a separator character has been sent.

8.4 Base Address

Users can access UART0, UART1 and UHCI0 respectively with two register base addresses shown in the following table. For more information about accessing peripherals from different buses please see Chapter 1: *System and Memory*.

Table 37: Base addresses of UART0, UART1 and UHCI0

Name	Bus to Access Peripheral	Base Address
UART0	PeriBUS1	0x3F400000
	PeriBUS2	0x60000000
UART1	PeriBUS1	0x3F410000
	PeriBUS2	0x60010000
UHCI0	PeriBUS1	0x3F414000
	PeriBUS2	0x60014000

8.5 Register Summary

The addresses in the following table are relative to the UART base addresses provided in Section 8.4.

Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_MEM_CONF_REG	UART threshold and allocation configuration	0x005C	R/W
Interrupt registers			
UART_INT_RAW_REG	Raw interrupt status	0x0004	RO
Interrupt Register			
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WO
Configuration Register			
UART_CLKDIV_REG	Clock divider configuration	0x0014	R/W
UART_CONF0_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_FLOW_CONF_REG	Software flow control configuration	0x0034	R/W
UART_SLEEP_CONF_REG	Sleeping mode configuration	0x0038	R/W
UART_SWFC_CONF0_REG	Software flow control character configuration	0x003C	R/W
UART_SWFC_CONF1_REG	Software flow-control character configuration	0x0040	R/W

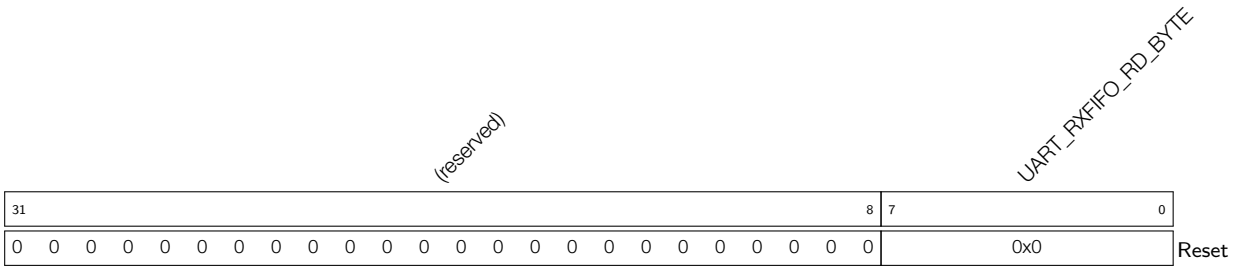
Name	Description	Address	Access
UART_IDLE_CONF_REG	Frame-end idle configuration	0x0044	R/W
UART_RS485_CONF_REG	RS485 mode configuration	0x0048	R/W
Autobaud Register			
UART_AUTOBAUD_REG	Autobaud configuration register	0x0018	R/W
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x0028	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x002C	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0030	RO
UART_POSPULSE_REG	Autobaud high pulse register	0x006C	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0070	RO
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0060	RO
UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x0064	RO
UART_FSM_STATUS_REG	UART transmit and receive status	0x0068	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_REG	Pre-sequence timing configuration	0x004C	R/W
UART_AT_CMD_POSTCNT_REG	Post-sequence timing configuration	0x0050	R/W
UART_AT_CMD_GAPTOUT_REG	Timeout configuration	0x0054	R/W
UART_AT_CMD_CHAR_REG	AT Escape Sequence Selection Configuration	0x0058	R/W
Version Register			
UART_DATE_REG	UART version control register	0x0074	R/W

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x002C	R/W
UHCI_AHB_TEST_REG	AHB test register	0x0048	R/W
UHCI_ESCAPE_CONF_REG	Escape characters configuration	0x0064	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0068	R/W
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0074	R/W
UHCI_REG_Q0_WORD0_REG	Q0_WORD0 quick_sent register	0x0078	R/W
UHCI_REG_Q0_WORD1_REG	Q0_WORD1 quick_sent register	0x007C	R/W
UHCI_REG_Q1_WORD0_REG	Q1_WORD0 quick_sent register	0x0080	R/W
UHCI_REG_Q1_WORD1_REG	Q1_WORD1 quick_sent register	0x0084	R/W
UHCI_REG_Q2_WORD0_REG	Q2_WORD0 quick_sent register	0x0088	R/W
UHCI_REG_Q2_WORD1_REG	Q2_WORD1 quick_sent register	0x008C	R/W
UHCI_REG_Q3_WORD0_REG	Q3_WORD0 quick_sent register	0x0090	R/W
UHCI_REG_Q3_WORD1_REG	Q3_WORD1 quick_sent register	0x0094	R/W
UHCI_REG_Q4_WORD0_REG	Q4_WORD0 quick_sent register	0x0098	R/W
UHCI_REG_Q4_WORD1_REG	Q4_WORD1 quick_sent register	0x009C	R/W
UHCI_REG_Q5_WORD0_REG	Q5_WORD0 quick_sent register	0x00A0	R/W
UHCI_REG_Q5_WORD1_REG	Q5_WORD1 quick_sent register	0x00A4	R/W
UHCI_REG_Q6_WORD0_REG	Q6_WORD0 quick_sent register	0x00A8	R/W

Name	Description	Address	Access
UHCI_REG_Q6_WORD1_REG	Q6_WORD1 quick_sent register	0x00AC	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x00B0	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x00B4	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x00B8	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x00BC	R/W
UHCI_PKT_THRES_REG	Configure register for packet length	0x00C0	R/W
Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	RO
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WO
DMA Status			
UHCI_DMA_OUT_STATUS_REG	DMA data-output status register	0x0014	RO
UHCI_DMA_IN_STATUS_REG	UHCI data-input status register	0x001C	RO
UHCI_STATE0_REG	UHCI decoder status register	0x0030	RO
UHCI_DMA_OUT_EOF_DES_ADDR _REG	Outlink descriptor address when EOF occurs	0x0038	RO
UHCI_DMA_IN_SUC_EOF_DES_ADDR _REG	Inlink descriptor address when EOF occurs	0x003C	RO
UHCI_DMA_IN_ERR_EOF_DES_ADDR _REG	Inlink descriptor address when errors occur	0x0040	RO
UHCI_DMA_OUT_EOF_BFR_DES_ ADDR_REG	Outlink descriptor address before the last transmit descriptor	0x0044	RO
UHCI_DMA_IN_DSCR_REG	The third word of the next receive descriptor	0x004C	RO
UHCI_DMA_IN_DSCR_BF0_REG	The third word of current receive descriptor	0x0050	RO
UHCI_DMA_IN_DSCR_BF1_REG	The second word of current receive descriptor	0x0054	RO
UHCI_DMA_OUT_DSCR_REG	The third word of the next transmit descriptor	0x0058	RO
UHCI_DMA_OUT_DSCR_BF0_REG	The third word of current transmit descriptor	0x005C	RO
UHCI_DMA_OUT_DSCR_BF1_REG	The second word of current transmit descriptor	0x0060	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x0070	RO
UHCI_STATE1_REG	UHCI encoder status register	0x0034	RO
DMA Configuration			
UHCI_DMA_OUT_PUSH_REG	Push control register of data-output FIFO	0x0018	R/W
UHCI_DMA_IN_POP_REG	Pop control register of data-input FIFO	0x0020	varies
UHCI_DMA_OUT_LINK_REG	Link descriptor address and control	0x0024	varies
UHCI_DMA_IN_LINK_REG	Link descriptor address and control	0x0028	varies
Version Register			
UHCI_DATE_REG	UHCI version control register	0x00FC	R/W

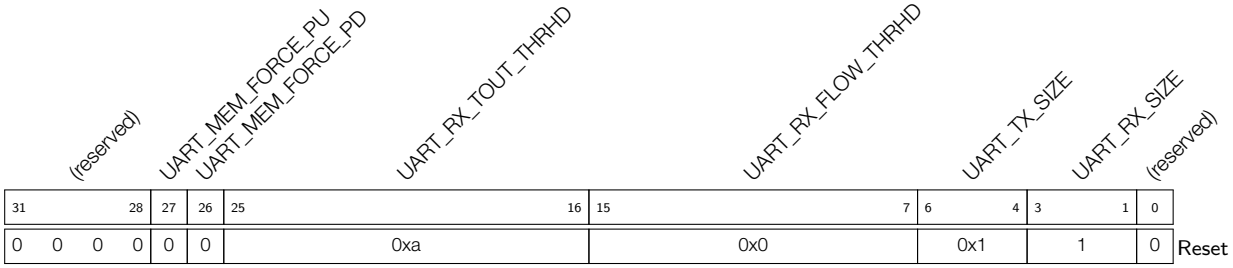
8.6 Registers

Register 8.1: UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE This register stores one byte data read from RX FIFO. (RO)

Register 8.2: UART_MEM_CONF_REG (0x005C)



UART_RX_SIZE This register is used to configure the amount of mem allocated for RX FIFO. The default number is 128 bytes. (R/W)

UART_TX_SIZE This register is used to configure the amount of mem allocated for TX FIFO. The default number is 128 bytes. (R/W)

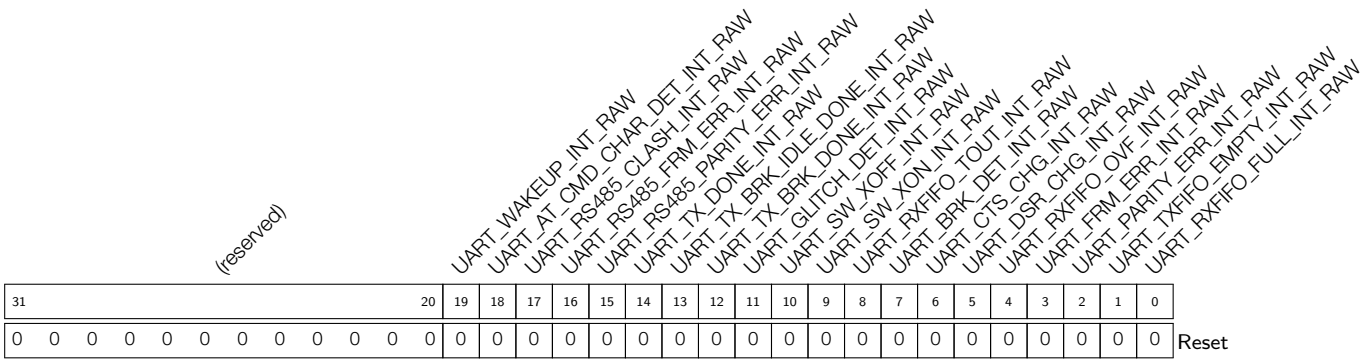
UART_RX_FLOW_THRHD This register is used to configure the maximum amount of data that can be received when hardware flow control works. (R/W)

UART_RX_TOUT_THRHD This register is used to configure the threshold time that receiver takes to receive one byte. 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 memory. (R/W)

UART_MEM_FORCE_PU Set this bit to force power up UART memory. (R/W)

Register 8.3: UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW This interrupt raw bit turns to high level when receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies. (RO)

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. (RO)

UART_PARITY_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a parity error in the data. (RO)

UART_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a data frame error. (RO)

UART_RXFIFO_OVF_INT_RAW This interrupt raw bit turns to high level when receiver receives more data than the FIFO can store. (RO)

UART_DSR_CHG_INT_RAW This interrupt raw bit turns to high level when receiver detects the edge change of DSRn signal. (RO)

UART_CTS_CHG_INT_RAW This interrupt raw bit turns to high level when receiver detects the edge change of CTSn signal. (RO)

UART_BRK_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects a 0 after the stop bit. (RO)

UART_RXFIFO_TOUT_INT_RAW This interrupt raw bit turns to high level when receiver takes more time than UART_RX_TOUT_THRHD to receive a byte. (RO)

UART_SW_XON_INT_RAW This interrupt raw bit turns to high level when receiver receives XON character when UART_SW_FLOW_CON_EN is set to 1. (RO)

UART_SW_XOFF_INT_RAW This interrupt raw bit turns to high level when receiver receives XOFF character when UART_SW_FLOW_CON_EN is set to 1. (RO)

UART_GLITCH_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects a glitch in the middle of a start bit. (RO)

UART_TX_BRK_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter completes sending NULL characters, after all data in TX FIFO are sent. (RO)

UART_TX_BRK_IDLE_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter has kept the shortest duration after sending the last data. (RO)

Continued on the next page...

Register 8.3: UART_INT_RAW_REG (0x0004)

Continued from the previous page...

UART_TX_DONE_INT_RAW This interrupt raw bit turns to high level when transmitter has sent out all data in FIFO. (RO)

UART_RS485_PARITY_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a parity error from the echo of transmitter in RS485 mode. (RO)

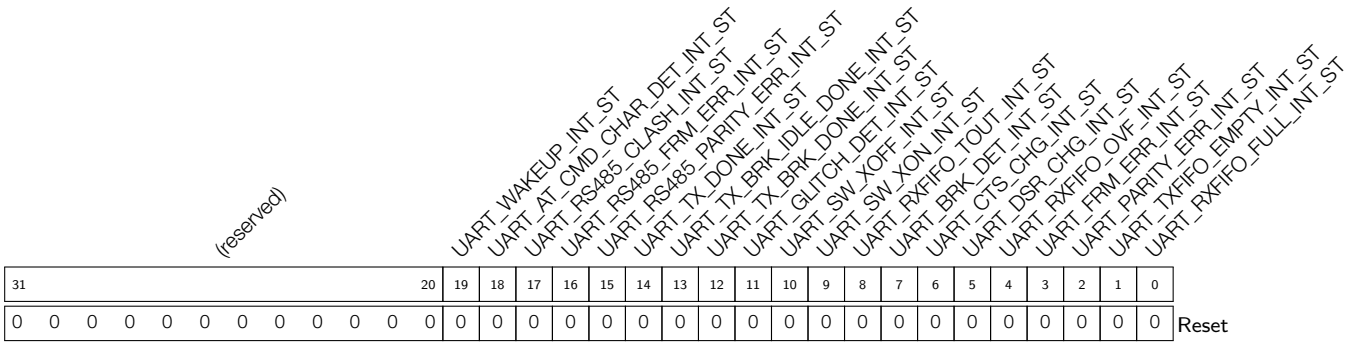
UART_RS485_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a data frame error from the echo of transmitter in RS485 mode. (RO)

UART_RS485_CLASH_INT_RAW This interrupt raw bit turns to high level when detects a clash between transmitter and receiver in RS485 mode. (RO)

UART_AT_CMD_CHAR_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects the configured UART_AT_CMD CHAR. (RO)

UART_WAKEUP_INT_RAW This interrupt raw bit turns to high level when input rxd edge changes more times than what UART_ACTIVE_THRESHOLD specifies in Light-sleep mode. (RO)

Register 8.4: UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST This is the status bit for UART_RXFIFO_FULL_INT_RAW when UART_RXFIFO_FULL_INT_ENA is set to 1. (RO)

UART_TXFIFO_EMPTY_INT_ST This is the status bit for UART_TXFIFO_EMPTY_INT_RAW when UART_TXFIFO_EMPTY_INT_ENA is set to 1. (RO)

UART_PARITY_ERR_INT_ST This is the status bit for UART_PARITY_ERR_INT_RAW when UART_PARITY_ERR_INT_ENA is set to 1. (RO)

UART_FRM_ERR_INT_ST This is the status bit for UART_FRM_ERR_INT_RAW when UART_FRM_ERR_INT_ENA is set to 1. (RO)

UART_RXFIFO_OVF_INT_ST This is the status bit for UART_RXFIFO_OVF_INT_RAW when UART_RXFIFO_OVF_INT_ENA is set to 1. (RO)

UART_DSR_CHG_INT_ST This is the status bit for UART_DSR_CHG_INT_RAW when UART_DSR_CHG_INT_ENA is set to 1. (RO)

UART_CTS_CHG_INT_ST This is the status bit for UART_CTS_CHG_INT_RAW when UART_CTS_CHG_INT_ENA is set to 1. (RO)

UART_BRK_DET_INT_ST This is the status bit for UART_BRK_DET_INT_RAW when UART_BRK_DET_INT_ENA is set to 1. (RO)

UART_RXFIFO_TOUT_INT_ST This is the status bit for UART_RXFIFO_TOUT_INT_RAW when UART_RXFIFO_TOUT_INT_ENA is set to 1. (RO)

UART_SW_XON_INT_ST This is the status bit for UART_SW_XON_INT_RAW when UART_SW_XON_INT_ENA is set to 1. (RO)

UART_SW_XOFF_INT_ST This is the status bit for UART_SW_XOFF_INT_RAW when UART_SW_XOFF_INT_ENA is set to 1. (RO)

UART_GLITCH_DET_INT_ST This is the status bit for UART_GLITCH_DET_INT_RAW when UART_GLITCH_DET_INT_ENA is set to 1. (RO)

UART_TX_BRK_DONE_INT_ST This is the status bit for UART_TX_BRK_DONE_INT_RAW when UART_TX_BRK_DONE_INT_ENA is set to 1. (RO)

UART_TX_BRK_IDLE_DONE_INT_ST This is the status bit for UART_TX_BRK_IDLE_DONE_INT_RAW when UART_TX_BRK_IDLE_DONE_INT_ENA is set to 1. (RO)

Continued on the next page...

Register 8.4: UART_INT_ST_REG (0x0008)

Continued from the previous page...

UART_TX_DONE_INT_ST This is the status bit for UART_TX_DONE_INT_RAW when UART_TX_DONE_INT_ENA is set to 1. (RO)

UART_RS485_PARITY_ERR_INT_ST This is the status bit for UART_RS485_PARITY_ERR_INT_RAW when UART_RS485_PARITY_INT_ENA is set to 1. (RO)

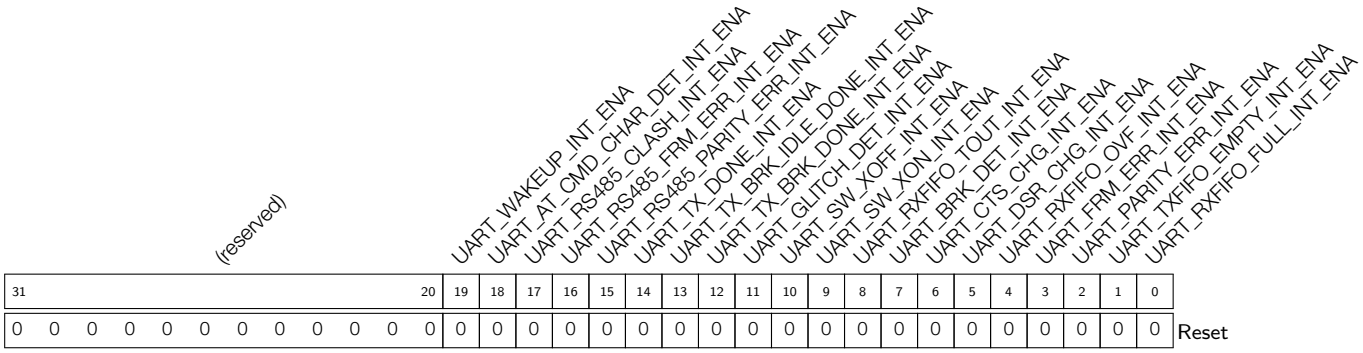
UART_RS485_FRM_ERR_INT_ST This is the status bit for UART_RS485_FRM_ERR_INT_RAW when UART_RS485_FM_ERR_INT_ENA is set to 1. (RO)

UART_RS485_CLASH_INT_ST This is the status bit for UART_RS485_CLASH_INT_RAW when UART_RS485_CLASH_INT_ENA is set to 1. (RO)

UART_AT_CMD_CHAR_DET_INT_ST This is the status bit for UART_AT_CMD_DET_INT_RAW when UART_AT_CMD_CHAR_DET_INT_ENA is set to 1. (RO)

UART_WAKEUP_INT_ST This is the status bit for UART_WAKEUP_INT_RAW when UART_WAKEUP_INT_ENA is set to 1. (RO)

Register 8.5: UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA This is the enable bit for UART_RXFIFO_FULL_INT_ST register. (R/W)

UART_TXFIFO_EMPTY_INT_ENA This is the enable bit for UART_TXFIFO_EMPTY_INT_ST register. (R/W)

UART_PARITY_ERR_INT_ENA This is the enable bit for UART_PARITY_ERR_INT_ST register. (R/W)

UART_FRM_ERR_INT_ENA This is the enable bit for UART_FRM_ERR_INT_ST register. (R/W)

UART_RXFIFO_OVF_INT_ENA This is the enable bit for UART_RXFIFO_OVF_INT_ST register. (R/W)

UART_DSR_CHG_INT_ENA This is the enable bit for UART_DSR_CHG_INT_ST register. (R/W)

UART_CTS_CHG_INT_ENA This is the enable bit for UART_CTS_CHG_INT_ST register. (R/W)

UART_BRK_DET_INT_ENA This is the enable bit for UART_BRK_DET_INT_ST register. (R/W)

UART_RXFIFO_TOUT_INT_ENA This is the enable bit for UART_RXFIFO_TOUT_INT_ST register. (R/W)

UART_SW_XON_INT_ENA This is the enable bit for UART_SW_XON_INT_ST register. (R/W)

UART_SW_XOFF_INT_ENA This is the enable bit for UART_SW_XOFF_INT_ST register. (R/W)

UART_GLITCH_DET_INT_ENA This is the enable bit for UART_GLITCH_DET_INT_ST register. (R/W)

UART_TX_BRK_DONE_INT_ENA This is the enable bit for UART_TX_BRK_DONE_INT_ST register. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA This is the enable bit for UART_TX_BRK_IDLE_DONE_INT_ST register. (R/W)

UART_TX_DONE_INT_ENA This is the enable bit for UART_TX_DONE_INT_ST register. (R/W)

UART_RS485_PARITY_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

UART_RS485_FRM_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

Continued on the next page...

Register 8.5: UART_INT_ENA_REG (0x000C)

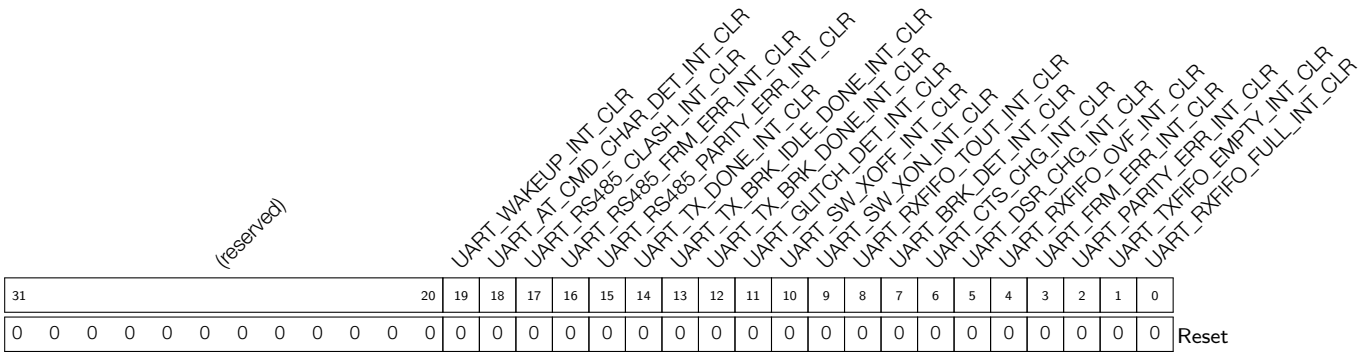
Continued from the previous page...

UART_RS485_CLASH_INT_ENA This is the enable bit for UART_RS485_CLASH_INT_ST register. (R/W)

UART_AT_CMD_CHAR_DET_INT_ENA This is the enable bit for UART_AT_CMD_CHAR_DET_INT_ST register. (R/W)

UART_WAKEUP_INT_ENA This is the enable bit for UART_WAKEUP_INT_ST register. (R/W)

Register 8.6: UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR Set this bit to clear UART_THE RXFIFO_FULL_INT_RAW interrupt. (WO)

UART_TXFIFO_EMPTY_INT_CLR Set this bit to clear UART_TXFIFO_EMPTY_INT_RAW interrupt. (WO)

UART_PARITY_ERR_INT_CLR Set this bit to clear UART_PARITY_ERR_INT_RAW interrupt. (WO)

UART_FRM_ERR_INT_CLR Set this bit to clear UART_FRM_ERR_INT_RAW interrupt. (WO)

UART_RXFIFO_OVF_INT_CLR Set this bit to clear UART_UART_RXFIFO_OVF_INT_RAW interrupt. (WO)

UART_DSR_CHG_INT_CLR Set this bit to clear UART_DSR_CHG_INT_RAW interrupt. (WO)

UART_CTS_CHG_INT_CLR Set this bit to clear UART_CTS_CHG_INT_RAW interrupt. (WO)

UART_BRK_DET_INT_CLR Set this bit to clear UART_BRK_DET_INT_RAW interrupt. (WO)

UART_RXFIFO_TOUT_INT_CLR Set this bit to clear UART_RXFIFO_TOUT_INT_RAW interrupt. (WO)

UART_SW_XON_INT_CLR Set this bit to clear UART_SW_XON_INT_RAW interrupt. (WO)

UART_SW_XOFF_INT_CLR Set this bit to clear UART_SW_XOFF_INT_RAW interrupt. (WO)

UART_GLITCH_DET_INT_CLR Set this bit to clear UART_GLITCH_DET_INT_RAW interrupt. (WO)

UART_TX_BRK_DONE_INT_CLR Set this bit to clear UART_TX_BRK_DONE_INT_RAW interrupt. (WO)

UART_TX_BRK_IDLE_DONE_INT_CLR Set this bit to clear UART_TX_BRK_IDLE_DONE_INT_RAW interrupt. (WO)

UART_TX_DONE_INT_CLR Set this bit to clear UART_TX_DONE_INT_RAW interrupt. (WO)

UART_RS485_PARITY_ERR_INT_CLR Set this bit to clear UART_RS485_PARITY_ERR_INT_RAW interrupt. (WO)

UART_RS485_FRM_ERR_INT_CLR Set this bit to clear UART_RS485_FRM_ERR_INT_RAW interrupt. (WO)

UART_RS485_CLASH_INT_CLR Set this bit to clear UART_RS485_CLASH_INT_RAW interrupt. (WO)

Continued on the next page...

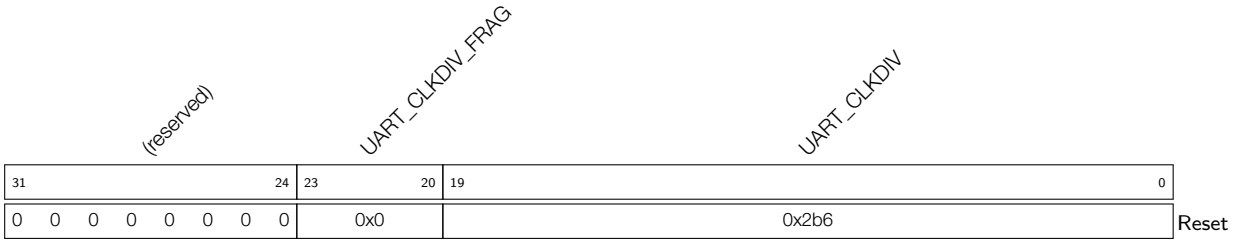
Register 8.6: UART_INT_CLR_REG (0x0010)

Continued from the previous page...

UART_AT_CMD_CHAR_DET_INT_CLR Set this bit to clear UART_AT_CMD_CHAR_DET_INT_RAW interrupt. (WO)

UART_WAKEUP_INT_CLR Set this bit to clear UART_WAKEUP_INT_RAW interrupt. (WO)

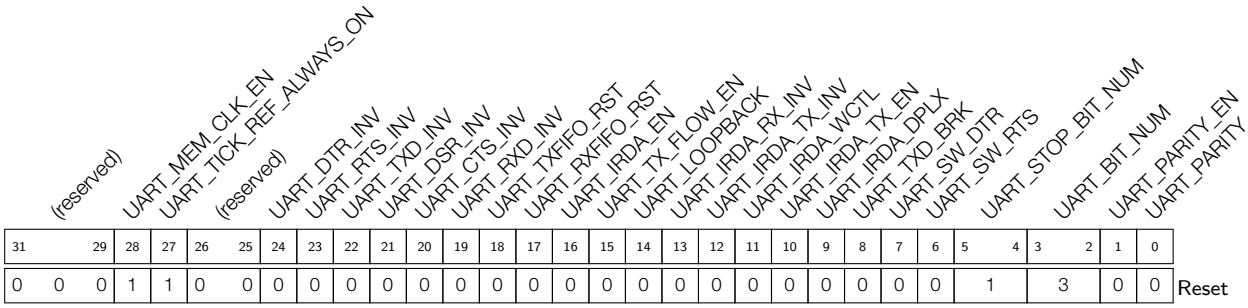
Register 8.7: UART_CLKDIV_REG (0x0014)



UART_CLKDIV The integral part of the frequency divisor. (R/W)

UART_CLKDIV_FRAG The decimal part of the frequency divisor. (R/W)

Register 8.8: UART_CONF0_REG (0x0020)



UART_PARITY This register is used to configure the parity check mode. 1'h0: even. 1'h1: odd. (R/W)

UART_PARITY_EN Set this bit to enable UART parity check. (R/W)

UART_BIT_NUM This register is used to set the length of data. 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits. (R/W)

UART_STOP_BIT_NUM This register is used to set the length of stop bit. 1: 1 bit 2: 1.5 bits 3: 2 bits. (R/W)

UART_SW_RTS This register is used to configure the software RTS signal which is used in software flow control. (R/W)

UART_SW_DTR This register 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 transmitter to send NULL 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'h1: The IrDA transmitter's 11th bit is the same as 10th bit. 1'h0: 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 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 inverse the level value of UART RXD signal. (R/W)

UART_CTS_INV Set this bit to inverse the level value of UART CTS signal. (R/W)

UART_DSR_INV Set this bit to inverse the level value of UART DSR signal. (R/W)

Continued on the next page...

Register 8.8: UART_CONF0_REG (0x0020)

Continued from the previous page...

UART_TXD_INV Set this bit to inverse the level value of UART TXD signal. (R/W)

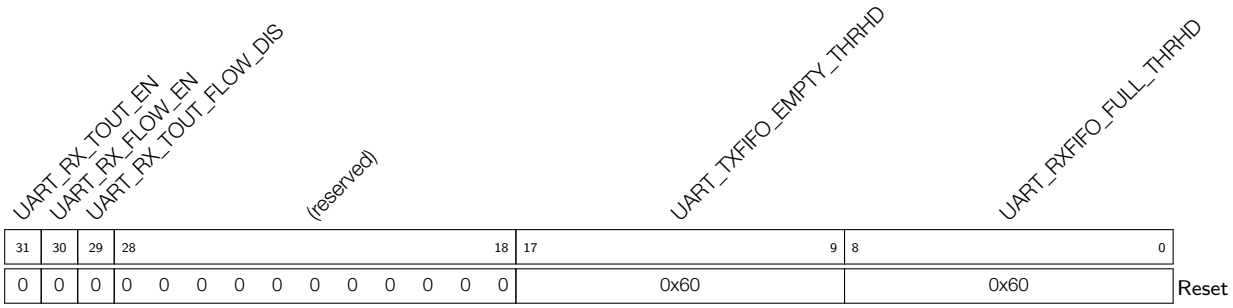
UART_RTS_INV Set this bit to inverse the level value of UART RTS signal. (R/W)

UART_DTR_INV Set this bit to inverse the level value of UART DTR signal. (R/W)

UART_TICK_REF_ALWAYS_ON This register is used to select the clock. 1'h1: APB_CLK. 1'h0: REF_TICK. (R/W)

UART_MEM_CLK_EN UART memory clock gate enable signal. 1: UART memory powers on, the data of which can be read and written. 0: UART memory powers down. (R/W)

Register 8.9: UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD It will produce UART_RXFIFO_FULL_INT interrupt when receiver receives more data than this register value. (R/W)

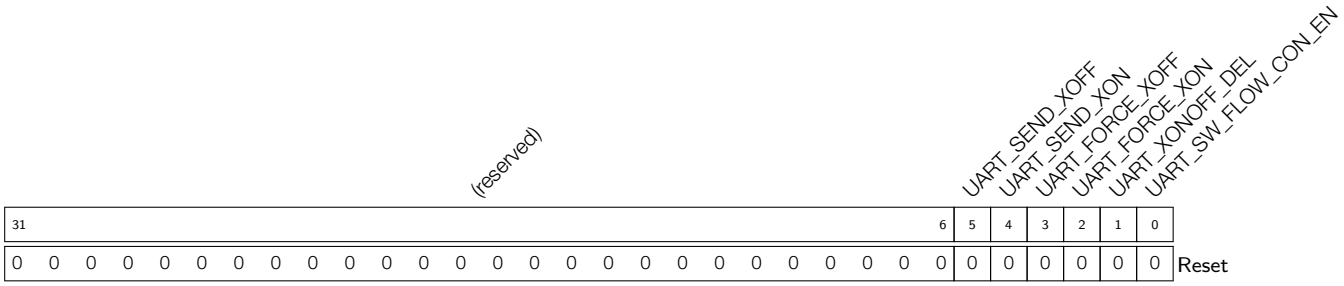
UART_TXFIFO_EMPTY_THRHD It will produce UART_TXFIFO_EMPTY_INT interrupt when the data amount in TX FIFO is less than this register value. (R/W)

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. 1'h1: Choose software flow control with configuring sw_rts signal. 1'h0: Disable software flow control. (R/W)

UART_RX_TOUT_EN This is the enable bit for UART receiver's timeout function. (R/W)

Register 8.10: UART_FLOW_CONF_REG (0x0034)



UART_SW_FLOW_CON_EN Set this bit to enable software flow control. It is used with register SW_XON or SW_XOFF. (R/W)

UART_XONOFF_DEL Set this bit to remove flow control character from the received data. (R/W)

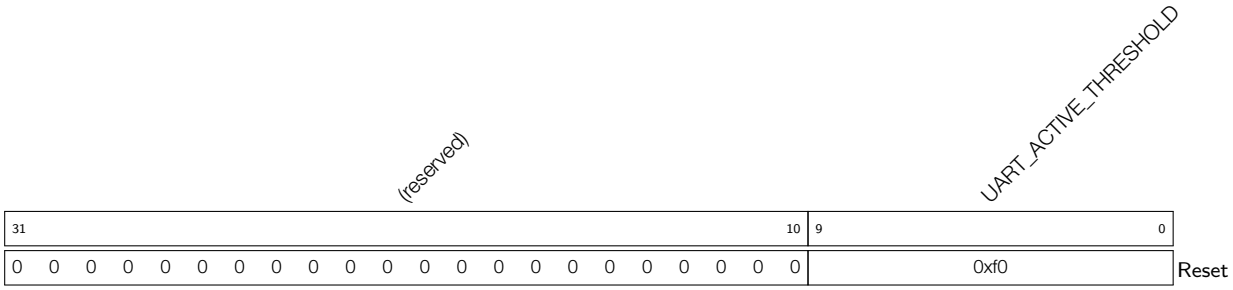
UART_FORCE_XON Set this bit to enable the transmitter to go on sending data. (R/W)

UART_FORCE_XOFF Set this bit to stop the transmitter from sending data. (R/W)

UART_SEND_XON Set this bit to send XON character. It is cleared by hardware automatically. (R/W)

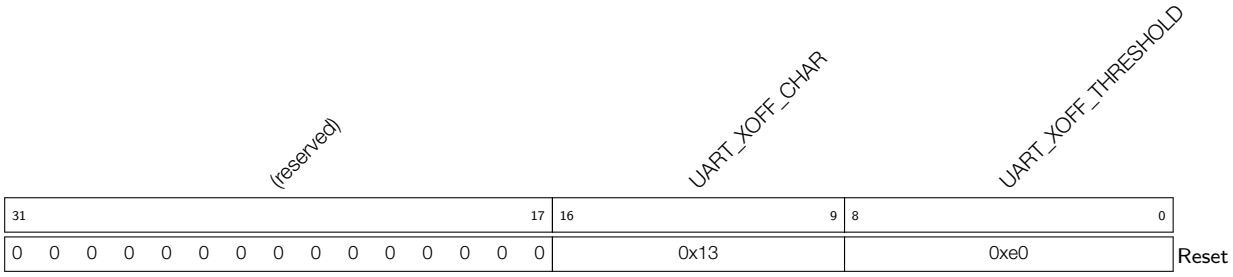
UART_SEND_XOFF Set this bit to send XOFF character. It is cleared by hardware automatically. (R/W)

Register 8.11: UART_SLEEP_CONF_REG (0x0038)



UART_ACTIVE_THRESHOLD The UART is activated from light sleeping mode when the input rxd edge changes more times than this register value. (R/W)

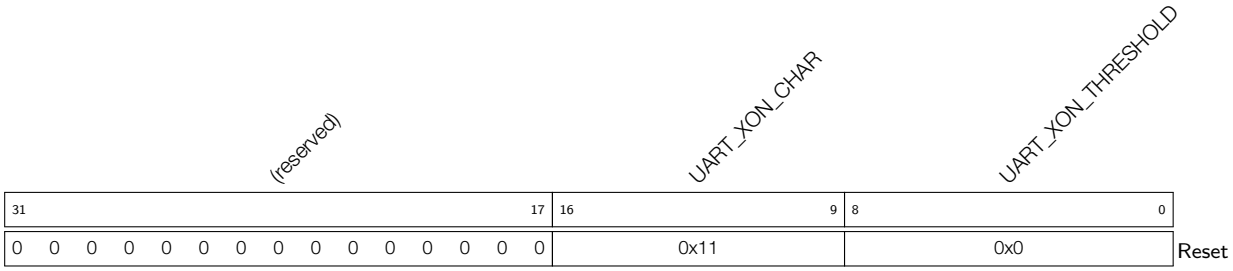
Register 8.12: UART_SWFC_CONF0_REG (0x003C)



UART_XOFF_THRESHOLD When the data amount in RX FIFO is more than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XOFF character. (R/W)

UART_XOFF_CHAR This register stores the XOFF flow control character. (R/W)

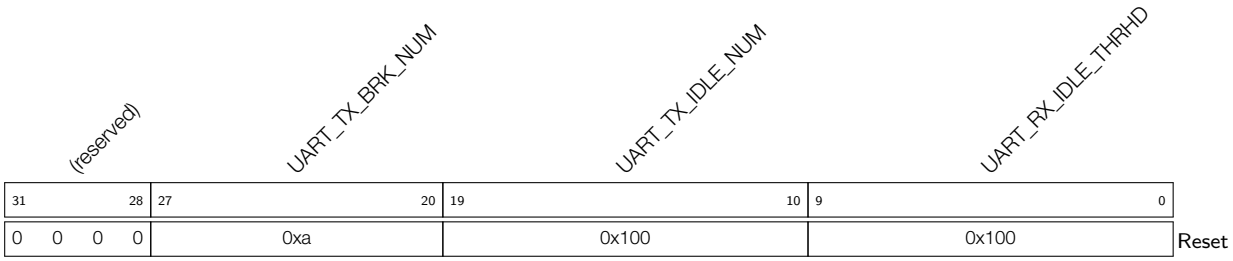
Register 8.13: UART_SWFC_CONF1_REG (0x0040)



UART_XON_THRESHOLD When the data amount in RX FIFO is less than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XON character. (R/W)

UART_XON_CHAR This register stores the XON flow control character. (R/W)

Register 8.14: UART_IDLE_CONF_REG (0x0044)

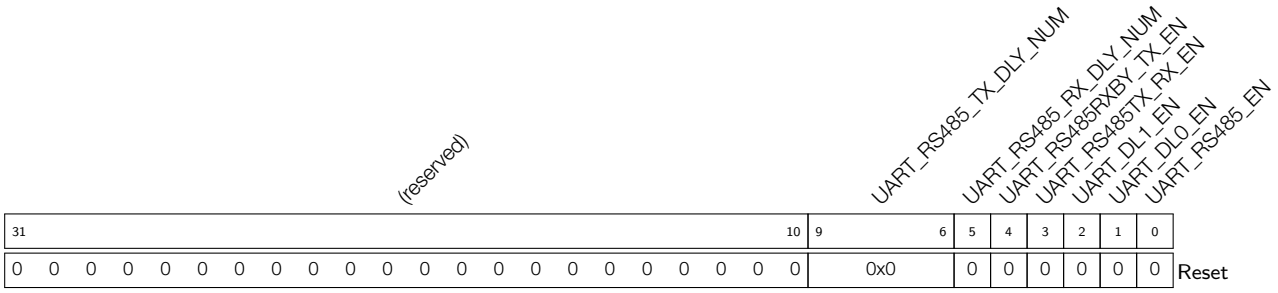


UART_RX_IDLE_THRHD It will produce frame end signal when receiver takes more time to receive one byte data than this register value. (R/W)

UART_TX_IDLE_NUM This register is used to configure the duration time between transfers. (R/W)

UART_TX_BRK_NUM This register 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 8.15: UART_RS485_CONF_REG (0x0048)



UART_RS485_EN Set this bit to choose the RS485 mode. (R/W)

UART_DL0_EN Set this bit to delay the stop bit by 1 bit. (R/W)

UART_DL1_EN Set this bit to delay the stop bit by 1 bit. (R/W)

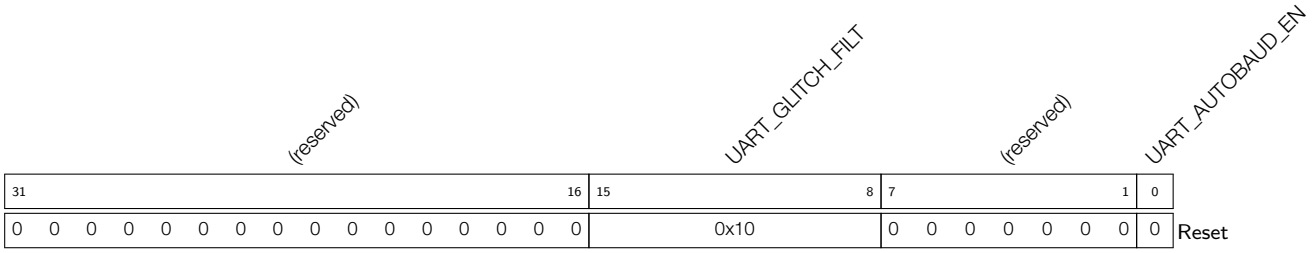
UART_RS485TX_RX_EN Set this bit to enable receiver could receive data when the transmitter is transmitting data in RS485 mode. (R/W)

UART_RS485RXBY_TX_EN 1'h1: enable RS485 transmitter to send data when RS485 receiver line is busy. 1'h0: RS485 transmitter should not send data when its receiver is busy. (R/W)

UART_RS485_RX_DLY_NUM This register is used to delay the receiver's internal data signal. (R/W)

UART_RS485_TX_DLY_NUM This register is used to delay the transmitter's internal data signal. (R/W)

Register 8.16: UART_AUTOBAUD_REG (0x0018)



UART_AUTOBAUD_EN This is the enable bit for detecting baudrate. (R/W)

UART_GLITCH_FILT when input pulse width is lower than this value, the pulse is ignored. This register is used in autobaud detect process. (R/W)

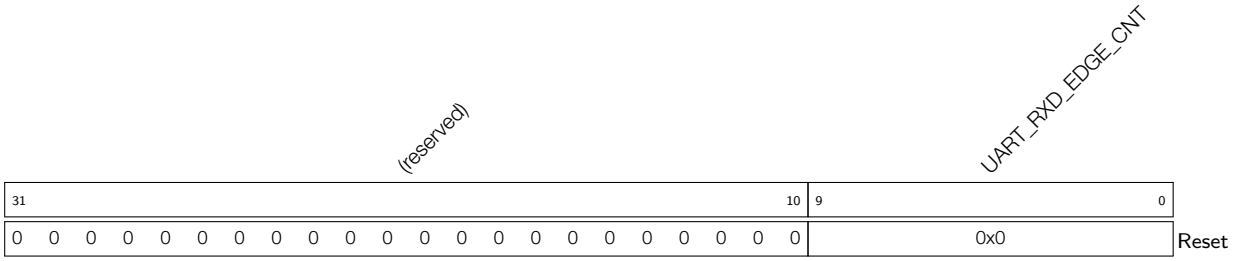
Register 8.17: UART_LOWPULSE_REG (0x0028)

UART_LOWPULSE_MIN_CNT This register stores the value of the minimum duration time of the low level pulse. It is used in baud rate detection. (RO)

Register 8.18: UART_HIGHPULSE_REG (0x002C)

UART_HIGHPULSE_MIN_CNT This register stores the value of the maximum duration time for the high level pulse. It is used in baud rate detection. (RO)

Register 8.19: UART_RXD_CNT_REG (0x0030)



UART_RXD_EDGE_CNT This register stores the count of rxd edge change. It is used in baud rate detection. (RO)

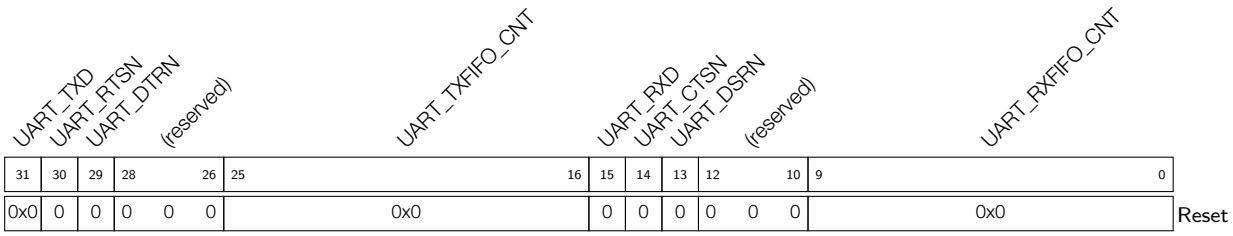
Register 8.20: UART_POSPULSE_REG (0x006C)

UART_POSEDGE_MIN_CNT This register stores the minimal input clock count between two positive edges. It is used in baud rate detection. (RO)

Register 8.21: UART_NEGPULSE_REG (0x0070)

UART_NEGEDGE_MIN_CNT This register stores the minimal input clock count between two negative edges. It is used in baud rate detection. (RO)

Register 8.22: UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Stores the byte number of valid data in RX FIFO. (RO)

UART_DSRN The register represent the level value of the internal UART DSR signal. (RO)

UART_CTSN This register represent the level value of the internal UART CTS signal. (RO)

UART_RXD This register represent the level value of the internal UART RXD signal. (RO)

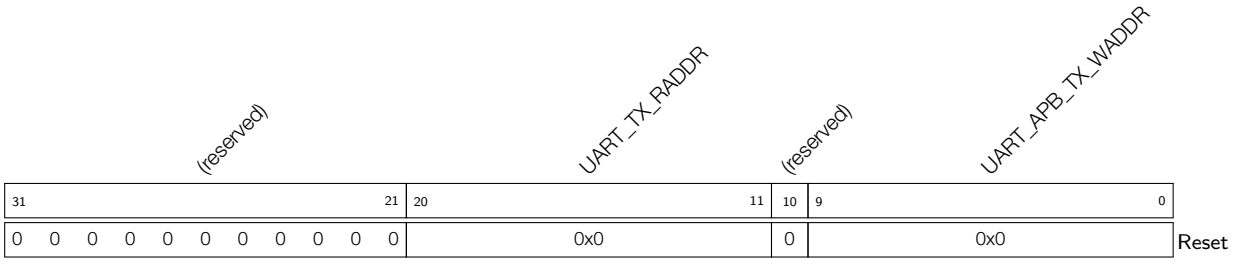
UART_TXFIFO_CNT Stores the byte number of data in TX FIFO. (RO)

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)

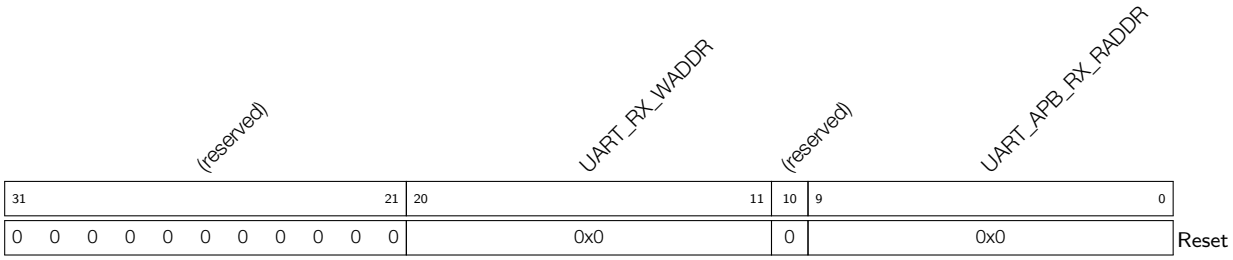
Register 8.23: UART_MEM_TX_STATUS_REG (0x0060)



UART_APB_TX_WADDR This register stores the offset address in TX FIFO when software writes TX FIFO via APB. (RO)

UART_TX_RADDR This register stores the offset address in TX FIFO when TX FSM reads data via Tx_FIFO_Ctrl. (RO)

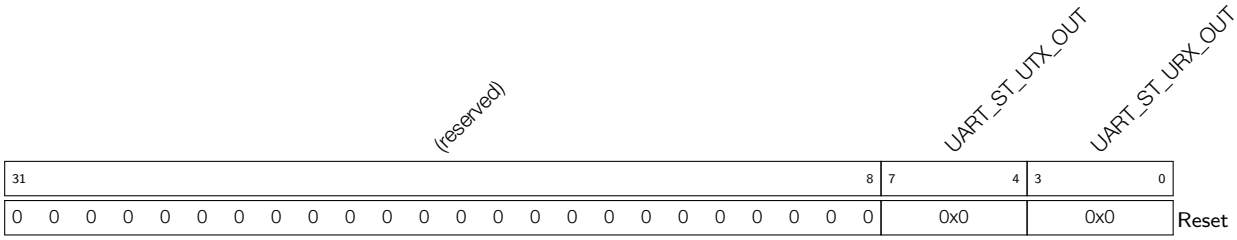
Register 8.24: UART_MEM_RX_STATUS_REG (0x0064)



UART_APB_RX_RADDR This register stores the offset address in RX_FIFO when software reads data from RX FIFO via APB. (RO)

UART_RX_WADDR This register stores the offset address in RX FIFO when Rx_FIFO_Ctrl writes RX FIFO. (RO)

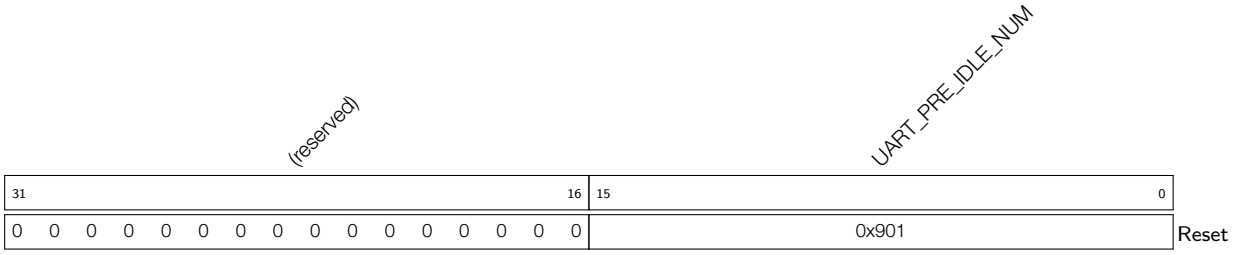
Register 8.25: UART_FSM_STATUS_REG (0x0068)



UART_ST_URX_OUT This is the status register of receiver. (RO)

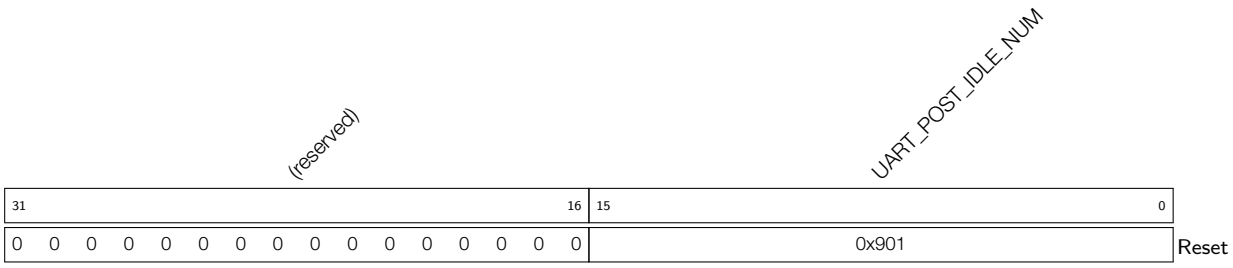
UART_ST_UTX_OUT This is the status register of transmitter. (RO)

Register 8.26: UART_AT_CMD_PRECNT_REG (0x004C)



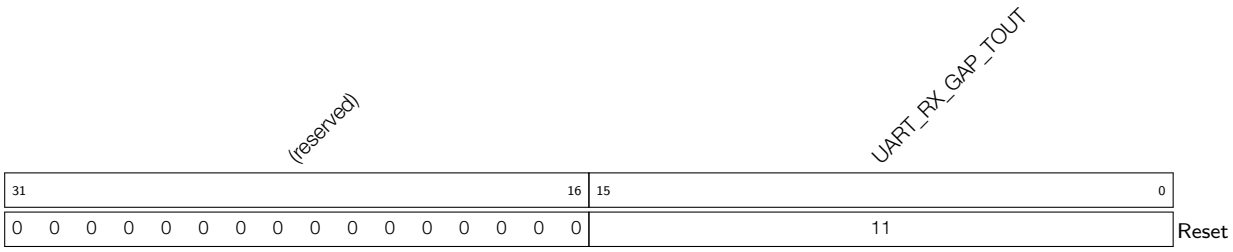
UART_PRE_IDLE_NUM This register is used to configure the idle duration time before the first AT_CMD is received by receiver. It will not take the next data received as AT_CMD character when the duration is less than this register value. (R/W)

Register 8.27: UART_AT_CMD_POSTCNT_REG (0x0050)



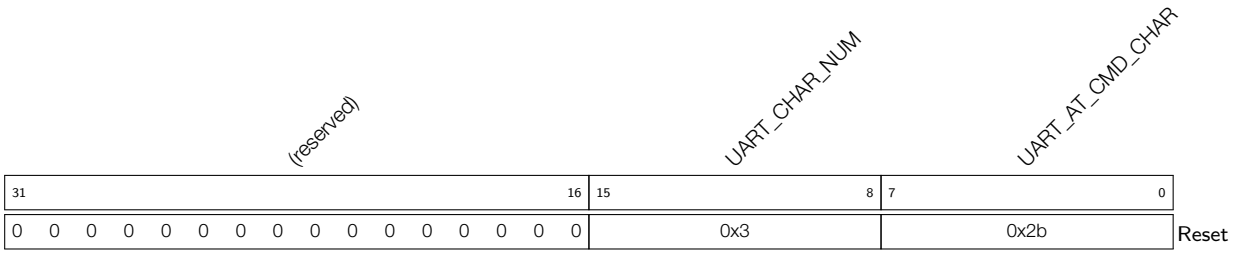
UART_POST_IDLE_NUM This register is used to configure the duration time between the last AT_CMD and the next data. It will not take the previous data as AT_CMD character when the duration is less than this register value. (R/W)

Register 8.28: UART_AT_CMD_GAPTOUT_REG (0x0054)



UART_RX_GAP_TOUT This register is used to configure the duration time between the AT_CMD chars. It will not take the data as continuous AT_CMD chars when the duration time is less than this register value. (R/W)

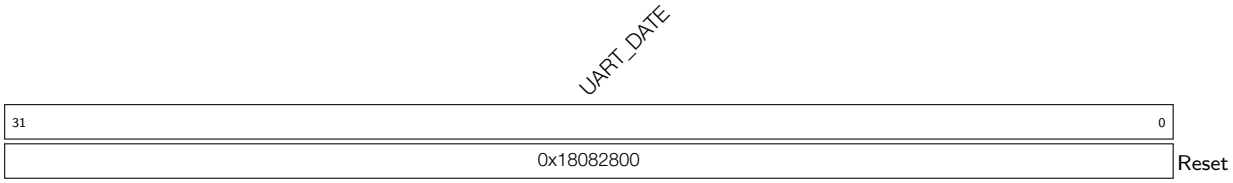
Register 8.29: UART_AT_CMD_CHAR_REG (0x0058)



UART_AT_CMD_CHAR This register is used to configure the content of AT_CMD character. (R/W)

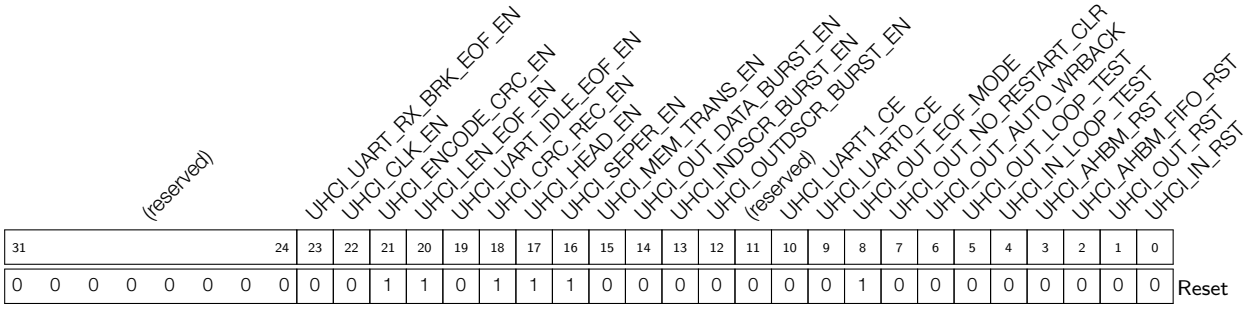
UART_CHAR_NUM This register is used to configure the number of continuous AT_CMD chars received by receiver. (R/W)

Register 8.30: UART_DATE_REG (0x0074)



UART_DATE This is the version control register. (R/W)

Register 8.31: UHCI_CONF0_REG (0x0000)



UHCI_IN_RST Set this bit to reset in DMA FSM. (R/W)

UHCI_OUT_RST Set this bit to reset out DMA FSM. (R/W)

UHCI_AHBM_FIFO_RST Set this bit to reset AHB interface cmdFIFO of DMA. (R/W)

UHCI_AHBM_RST Set this bit to reset AHB interface of DMA. (R/W)

UHCI_IN_LOOP_TEST Reserved. (R/W)

UHCI_OUT_LOOP_TEST Reserved. (R/W)

UHCI_OUT_AUTO_WRBACK Set this bit to enable automatic outlink-writeback when all the data in TX Buffer has been transmitted. (R/W)

UHCI_OUT_NO_RESTART_CLR Reserved. (R/W)

UHCI_OUT_EOF_MODE This register is used to specify the generation mode of UHCI_OUT_EOF_INT interrupt. 1: When DMA has popped all data from FIFO. 0: When AHB has pushed all data to FIFO. (R/W)

UHCI_UART0_CE Set this bit to link up HCI and UART0. (R/W)

UHCI_UART1_CE Set this bit to link up HCI and UART1. (R/W)

UHCI_OUTDSCR_BURST_EN This register is used to specify DMA transmit descriptor transfer mode. 1: burst mode. 0: byte mode. (R/W)

UHCI_INDSCR_BURST_EN This register is used to specify DMA receive descriptor transfer mode. 1: burst mode. 0: byte mode. (R/W)

UHCI_OUT_DATA_BURST_EN This register is used to specify data transfer mode. 1: Transmit data in burst mode. 0: Transmit data in byte mode. (R/W)

UHCI_MEM_TRANS_EN 1: UHCI transmitted data would be write back into DMA INFIFO. (R/W)

UHCI_SEPER_EN Set this bit to separate the data frame using a special char. (R/W)

UHCI_HEAD_EN Set this bit to encode the data packet with a formatting header. (R/W)

UHCI_CRC_REC_EN Set this bit to enable UHCI to receive the 16 bit CRC. (R/W)

UHCI_UART_IDLE_EOF_EN If this bit is set to 1, UHCI will end the payload receiving process when UART has been in idle state. (R/W)

Continued on the next page...

Register 8.31: UHCI_CONF0_REG (0x0000)

Continued from the previous page...

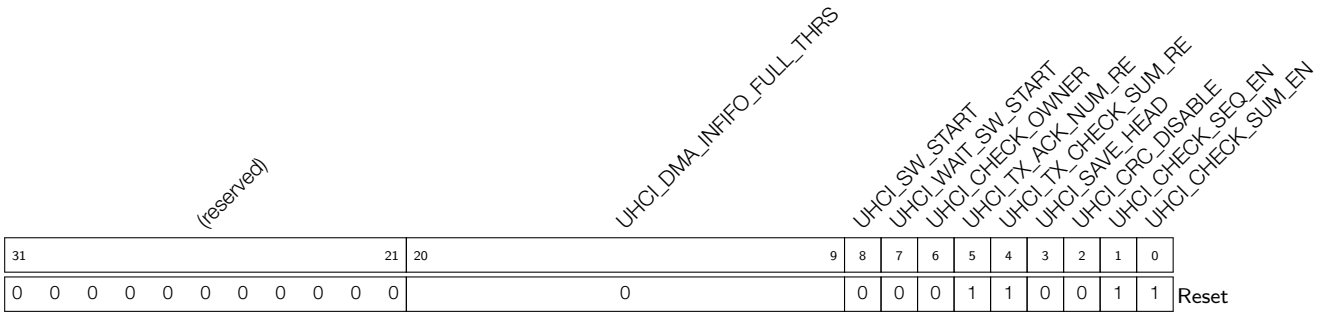
UHCI_LEN_EOF_EN If this bit is set to 1, UHCI decoder receiving payload data is end when the receiving byte count has reached the specified value. The value is payload length indicated by UCHI packet header when UHCI_HEAD_EN is 1 or the value is a configuration value when UHCI_HEAD_EN is 0. If this bit is set to 0, UHCI decoder receiving payload data is end when 0xc0 is received. (R/W)

UHCI_ENCODE_CRC_EN Set this bit to enable data integrity checking by appending a 16 bit CCITT-CRC to the end of the payload. (R/W)

UHCI_CLK_EN 1'b1: Force clock on for register. 1'b0: Support clock only when application writes registers. (R/W)

UHCI_UART_RX_BRK_EOF_EN if this bit is set to 1, UHCI will end payload_rec process when NULL frame is received by UART. (R/W)

Register 8.32: UHCI_CONF1_REG (0x002C)



UHCI_CHECK_SUM_EN This is the enable bit to check header checksum when UHCI receives a data packet. (R/W)

UHCI_CHECK_SEQ_EN This is the enable bit to check sequence number when UHCI receives a data packet. (R/W)

UHCI_CRC_DISABLE Set this bit to support CRC calculation. Data Integrity Check Present bit in UHCI packet frame should be 1. (R/W)

UHCI_SAVE_HEAD Set this bit to save the packet header when HCI receives a data packet. (R/W)

UHCI_TX_CHECK_SUM_RE Set this bit to encode the data packet with a checksum. (R/W)

UHCI_TX_ACK_NUM_RE Set this bit to encode the data packet with an acknowledgement when a reliable packet is to be transmit. (R/W)

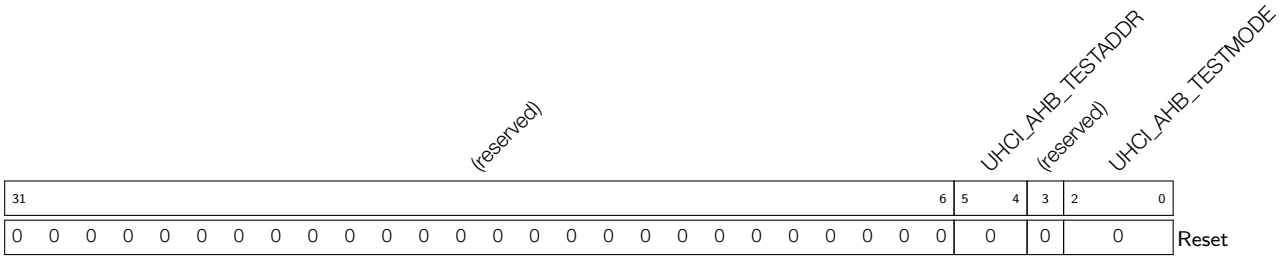
UHCI_CHECK_OWNER 1: Check the link list descriptor when link list owner is DMA controller; 0: Always check link list descriptor. (R/W)

UHCI_WAIT_SW_START The uhci-encoder will jump to ST_SW_WAIT status if this register is set to 1. (R/W)

UHCI_SW_START If current UHCI_ENCODE_STATE is ST_SW_WAIT, the UHCI will start to send data packet out when this bit is set to 1. (R/W)

UHCI_DMA_INFIFO_FULL_THRS This field is used to generate the UHCI_DMA_INFIFO_FULL_WM_INT interrupt when the counter value of DMA RX FIFO exceeds the value of the register. (R/W)

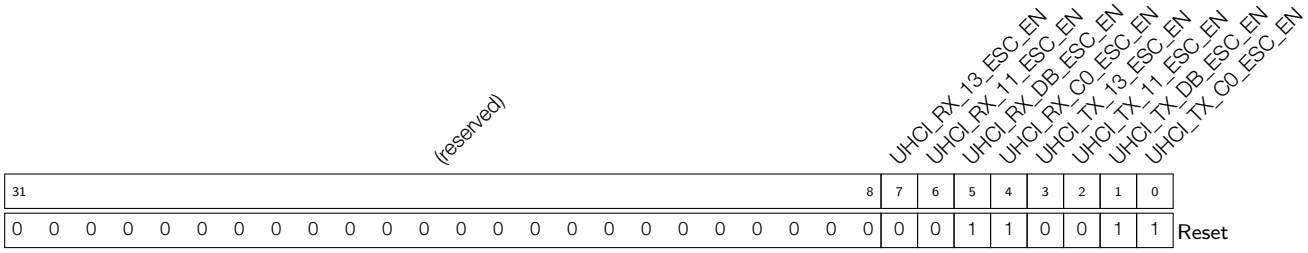
Register 8.33: UHCI_AHB_TEST_REG (0x0048)



UHCI_AHB_TESTMODE Reserved. (R/W)

UHCI_AHB_TESTADDR Reserved. (R/W)

Register 8.34: UHCI_ESCAPE_CONF_REG (0x0064)



UHCI_TX_C0_ESC_EN Set this bit to enable decoding char 0xc0 when DMA receives data. (R/W)

UHCI_TX_DB_ESC_EN Set this bit to enable decoding char 0xdb when DMA receives data. (R/W)

UHCI_TX_11_ESC_EN Set this bit to enable decoding flow control char 0x11 when DMA receives data. (R/W)

UHCI_TX_13_ESC_EN Set this bit to enable decoding flow control char 0x13 when DMA receives data. (R/W)

UHCI_RX_C0_ESC_EN Set this bit to enable replacing 0xc0 by special char when DMA sends data. (R/W)

UHCI_RX_DB_ESC_EN Set this bit to enable replacing 0xdb by special char when DMA sends data. (R/W)

UHCI_RX_11_ESC_EN Set this bit to enable replacing flow control char 0x11 by special char when DMA sends data. (R/W)

UHCI_RX_13_ESC_EN Set this bit to enable replacing flow control char 0x13 by special char when DMA sends data. (R/W)

Register 8.35: UHCI_HUNG_CONF_REG (0x0068)

(reserved)	UHCI_RXFIFO_TIMEOUT_ENA	UHCI_RXFIFO_TIMEOUT_SHIFT	UHCI_RXFIFO_TIMEOUT	UHCI_TXFIFO_TIMEOUT_ENA	UHCI_TXFIFO_TIMEOUT_SHIFT	UHCI_TXFIFO_TIMEOUT	
31 24	23	22 20	19 12	11	10 8	7 0	
0 0 0 0 0 0 0 0	1	0	0x10	1	0	0x10	Reset

UHCI_TXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_TX_HUNG_INT interrupt when DMA takes more time to receive data. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

UHCI_TXFIFO_TIMEOUT_ENA This is the enable bit for Tx-FIFO receive-data timeout. (R/W)

UHCI_RXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_RX_HUNG_INT interrupt when DMA takes more time to read data from RAM. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA This is the enable bit for DMA send-data timeout. (R/W)

Register 8.36: UHCI_QUICK_SENT_REG (0x0074)

(reserved)	UHCI_ALWAYS_SEND_EN	UHCI_ALWAYS_SEND_NUM	UHCI_SINGLE_SEND_EN	UHCI_SINGLE_SEND_NUM	
31 8	7	6 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0x0	0	0x0	Reset

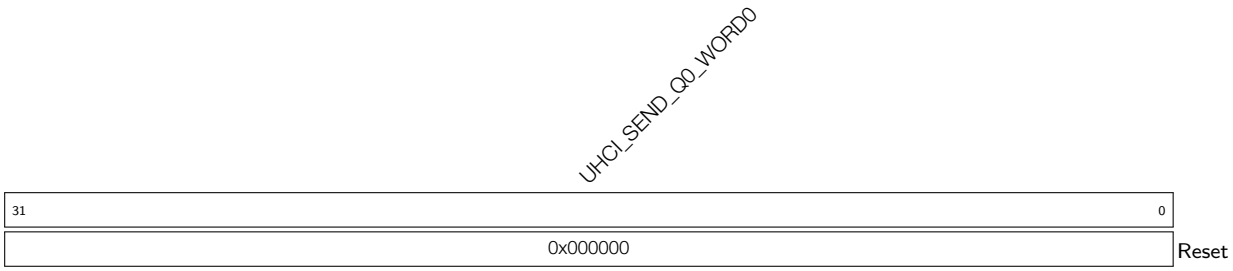
UHCI_SINGLE_SEND_NUM This register is used to specify the single_send register. (R/W)

UHCI_SINGLE_SEND_EN Set this bit to enable single_send mode to send short packet. (R/W)

UHCI_ALWAYS_SEND_NUM This register is used to specify the always_send register. (R/W)

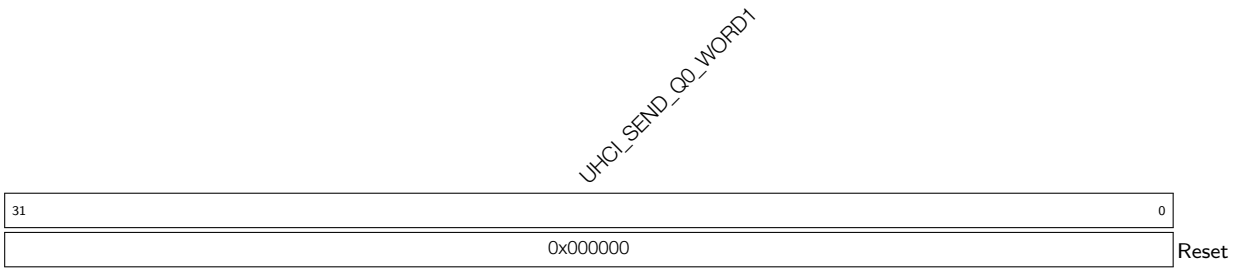
UHCI_ALWAYS_SEND_EN Set this bit to enable always_send mode to send short packet. (R/W)

Register 8.37: UHCI_REG_Q0_WORD0_REG (0x0078)



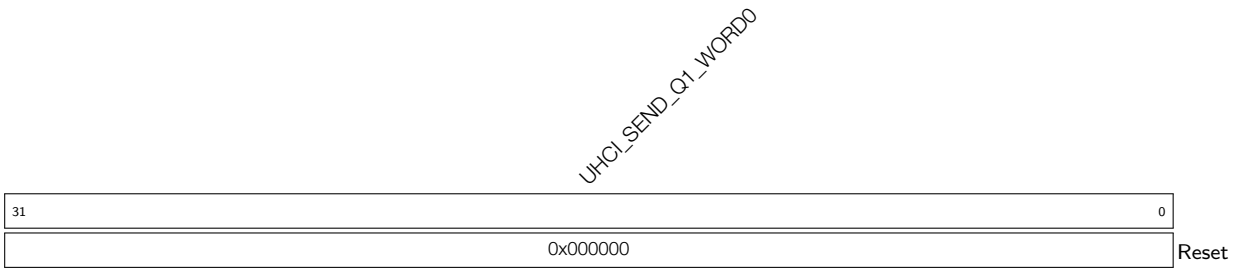
UHCI_SEND_Q0_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.38: UHCI_REG_Q0_WORD1_REG (0x007C)



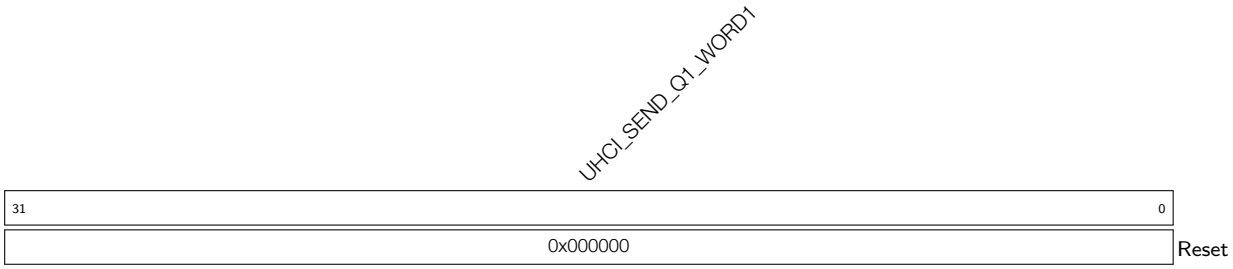
UHCI_SEND_Q0_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.39: UHCI_REG_Q1_WORD0_REG (0x0080)



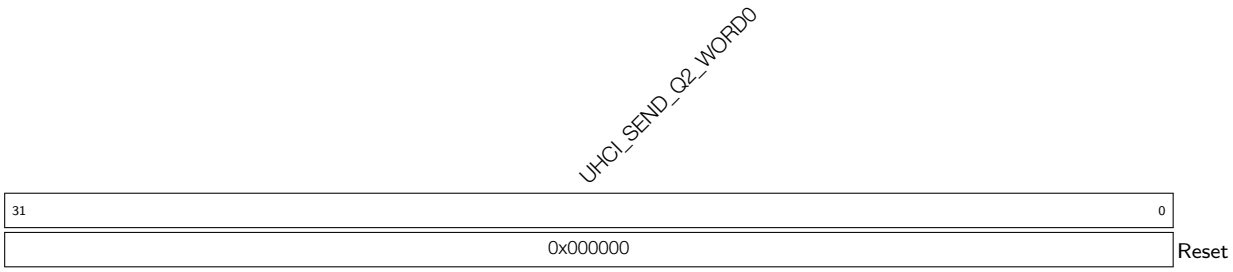
UHCI_SEND_Q1_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.40: UHCI_REG_Q1_WORD1_REG (0x0084)



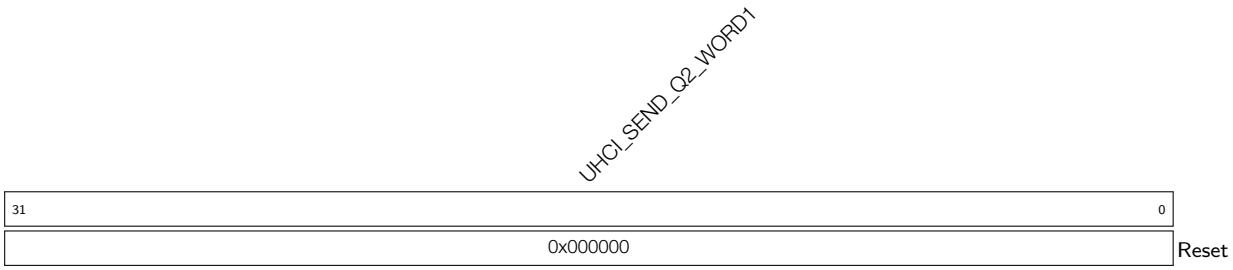
UHCI_SEND_Q1_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.41: UHCI_REG_Q2_WORD0_REG (0x0088)



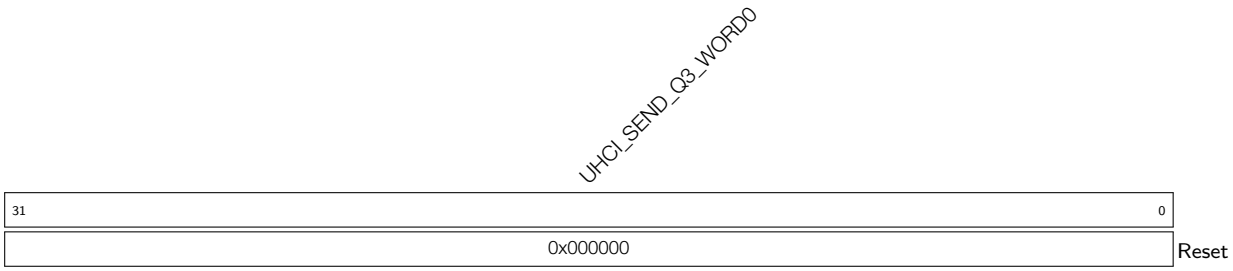
UHCI_SEND_Q2_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.42: UHCI_REG_Q2_WORD1_REG (0x008C)



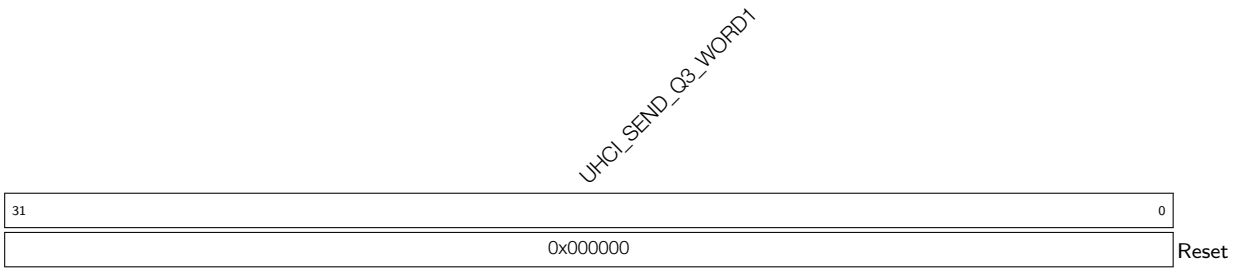
UHCI_SEND_Q2_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.43: UHCI_REG_Q3_WORD0_REG (0x0090)



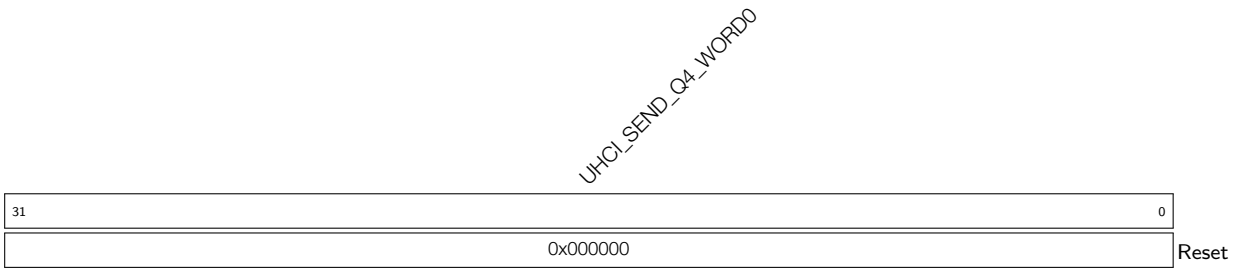
UHCI_SEND_Q3_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.44: UHCI_REG_Q3_WORD1_REG (0x0094)



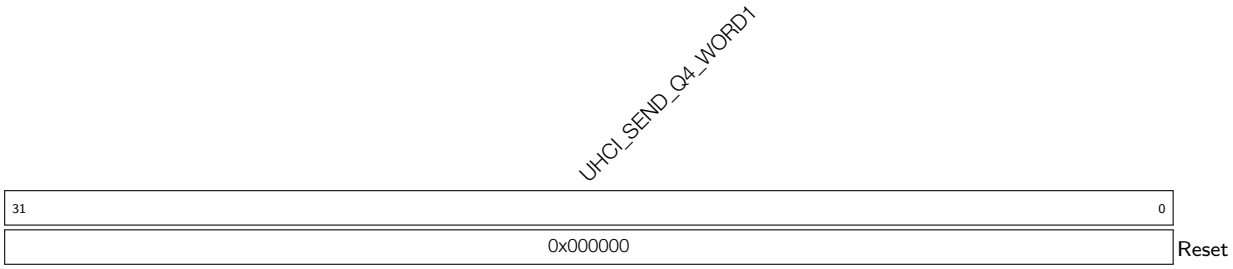
UHCI_SEND_Q3_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.45: UHCI_REG_Q4_WORD0_REG (0x0098)



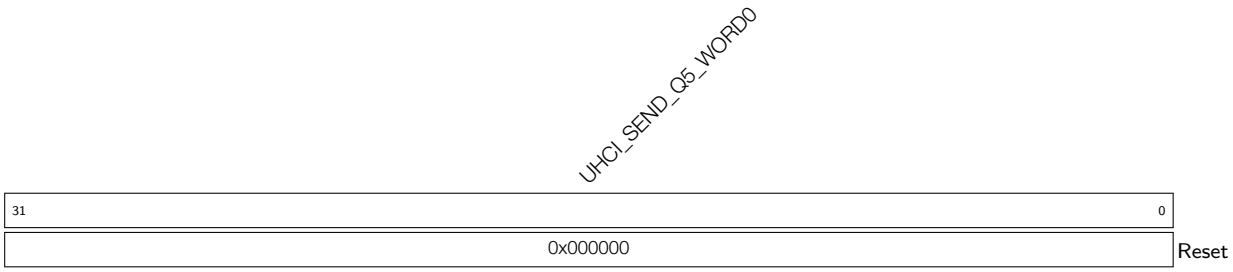
UHCI_SEND_Q4_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.46: UHCI_REG_Q4_WORD1_REG (0x009C)



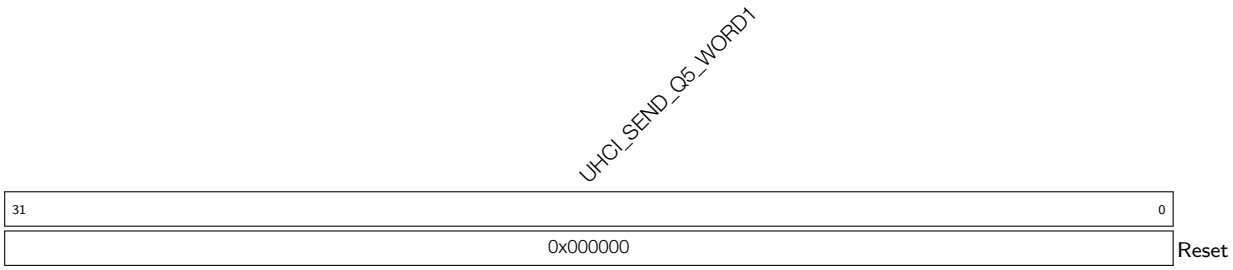
UHCI_SEND_Q4_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.47: UHCI_REG_Q5_WORD0_REG (0x00A0)



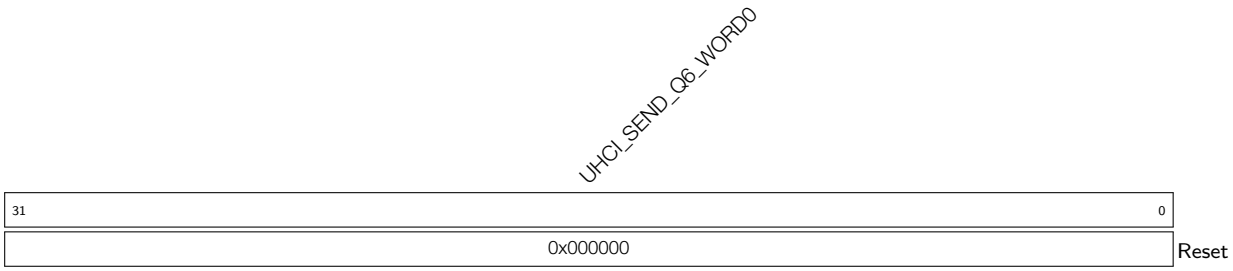
UHCI_SEND_Q5_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.48: UHCI_REG_Q5_WORD1_REG (0x00A4)



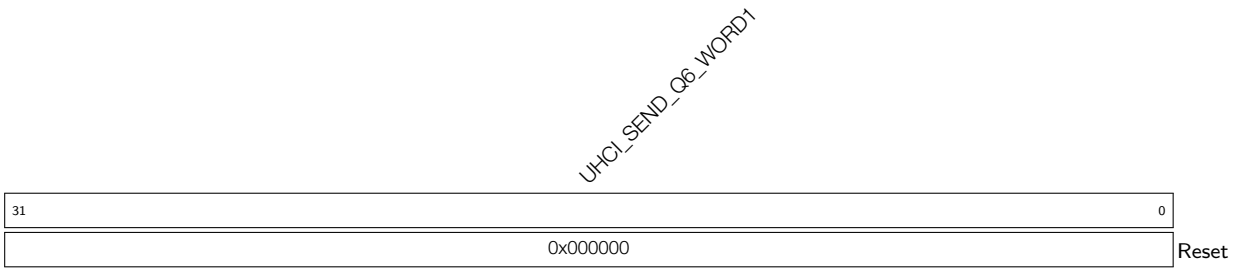
UHCI_SEND_Q5_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.49: UHCI_REG_Q6_WORD0_REG (0x00A8)



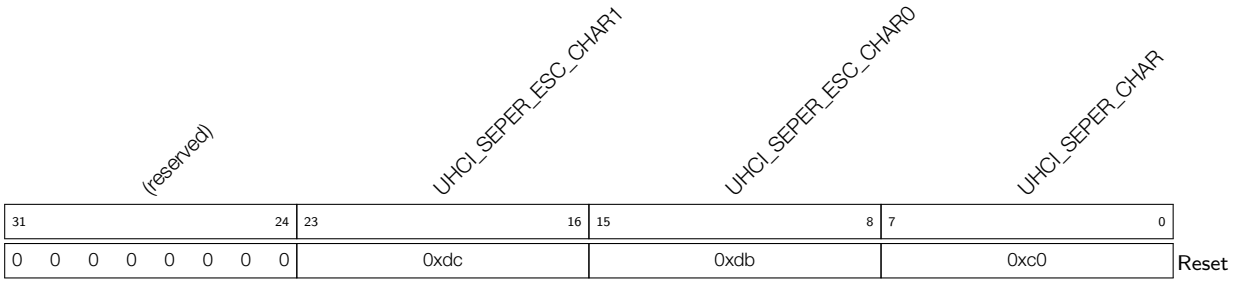
UHCI_SEND_Q6_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.50: UHCI_REG_Q6_WORD1_REG (0x00AC)



UHCI_SEND_Q6_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 8.51: UHCI_ESC_CONF0_REG (0x00B0)

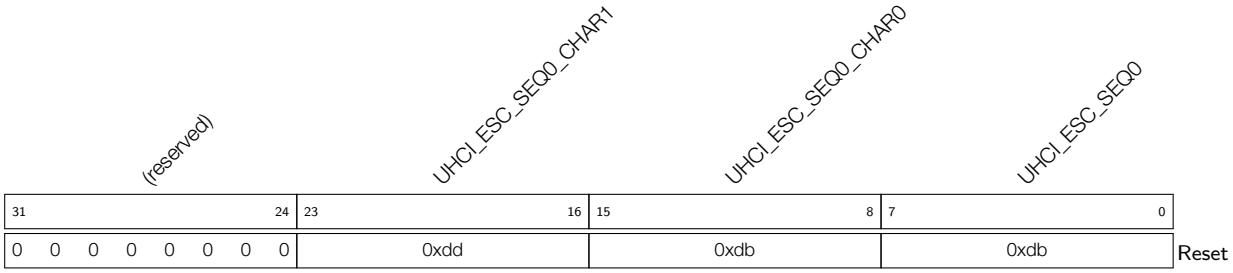


UHCI_SEPER_CHAR This register is used to define the separate char that need to be encoded, default is 0xc0. (R/W)

UHCI_SEPER_ESC_CHAR0 This register is used to define the first char of slip escape sequence when encoding the separate char, default is 0xdb. (R/W)

UHCI_SEPER_ESC_CHAR1 This register is used to define the second char of slip escape sequence when encoding the separate char, default is 0xdc. (R/W)

Register 8.52: UHCI_ESC_CONF1_REG (0x00B4)

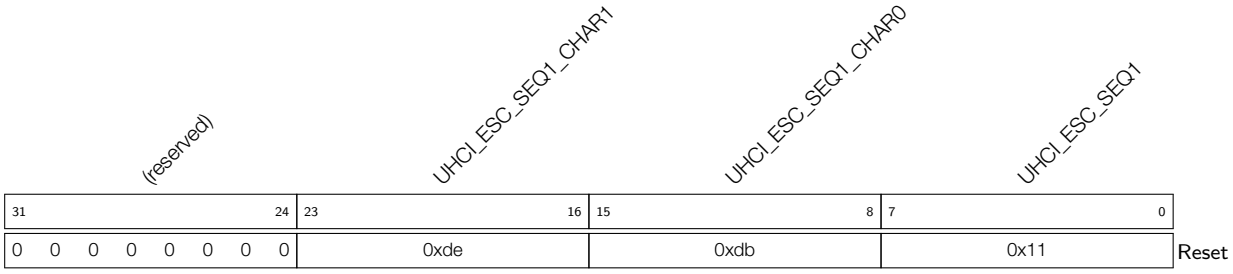


UHCI_ESC_SEQ0 This register is used to define a char that need to be encoded, default is 0xdb that used as the first char of slip escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdb. (R/W)

UHCI_ESC_SEQ0_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdd. (R/W)

Register 8.53: UHCI_ESC_CONF2_REG (0x00B8)

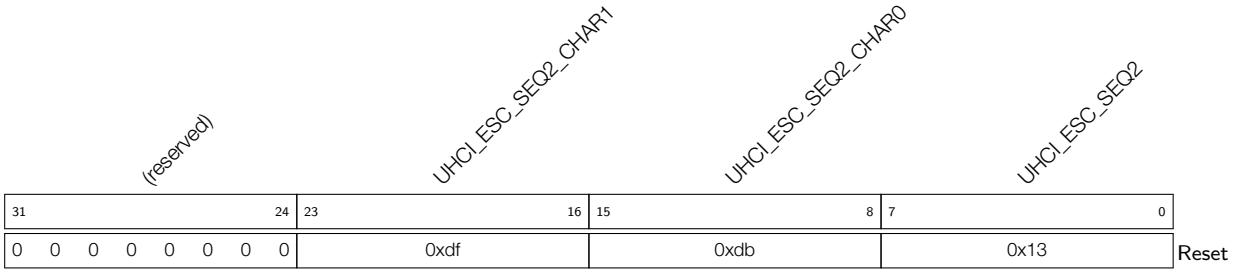


UHCI_ESC_SEQ1 This register is used to define a char that need to be encoded, default is 0x11 that used as flow control char. (R/W)

UHCI_ESC_SEQ1_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xdb. (R/W)

UHCI_ESC_SEQ1_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xde. (R/W)

Register 8.54: UHCI_ESC_CONF3_REG (0x00BC)

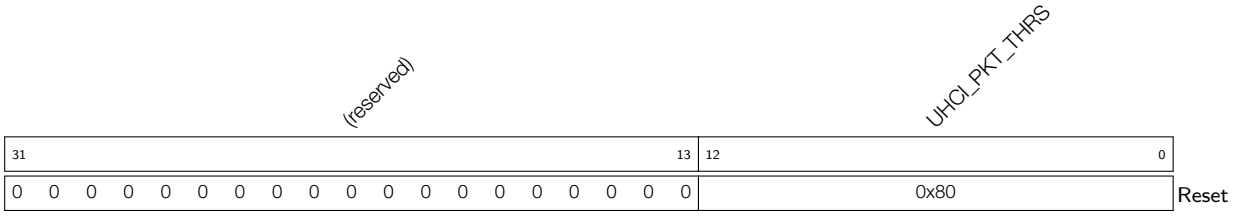


UHCI_ESC_SEQ2 This register is used to define a char that need to be decoded, default is 0x13 that used as flow control char. (R/W)

UHCI_ESC_SEQ2_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdb. (R/W)

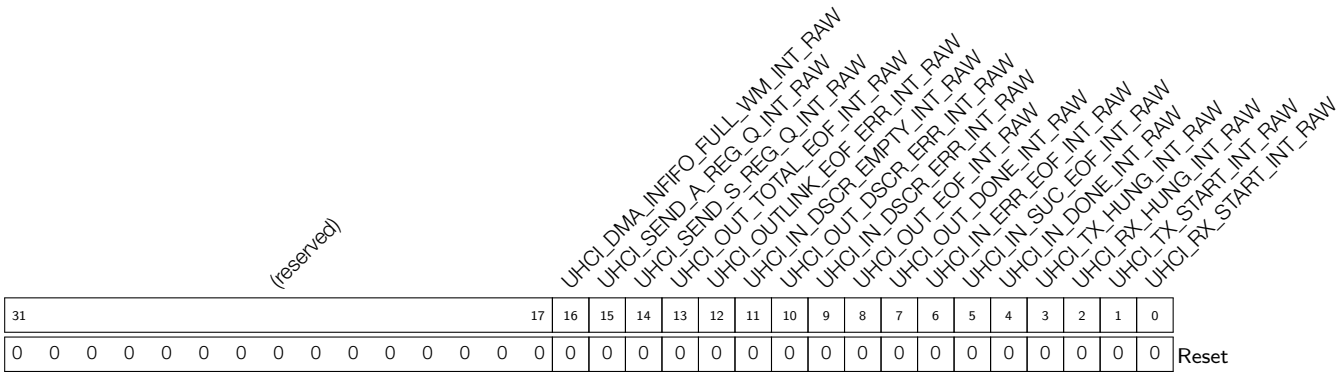
UHCI_ESC_SEQ2_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdf. (R/W)

Register 8.55: UHCI_PKT_THRES_REG (0x00C0)



UHCI_PKT_THRS This register is used to configure the maximum value of the packet length when UHCI_HEAD_EN is 0. (R/W)

Register 8.56: UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW This is the interrupt raw bit. Triggered when a separator char has been sent. (RO)

UHCI_TX_START_INT_RAW This is the interrupt raw bit. Triggered when DMA detects a separator char. (RO)

UHCI_RX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when DMA takes more time to receive data than the configure value. (RO)

UHCI_TX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when DMA takes more time to read data from RAM than the configured value. (RO)

UHCI_IN_DONE_INT_RAW This is the interrupt raw bit. Triggered when an receive descriptor is completed. (RO)

UHCI_IN_SUC_EOF_INT_RAW This is the interrupt raw bit. Triggered when a data packet has been received successfully. (RO)

UHCI_IN_ERR_EOF_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in EOF in the receive descriptor. (RO)

UHCI_OUT_DONE_INT_RAW This is the interrupt raw bit. Triggered when an transmit descriptor is completed. (RO)

UHCI_OUT_EOF_INT_RAW This is the interrupt raw bit. Triggered when the current descriptor's EOF bit is 1. (RO)

UHCI_IN_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in the receive descriptor. (RO)

UHCI_OUT_DSCR_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in the transmit descriptor. (RO)

UHCI_IN_DSCR_EMPTY_INT_RAW This is the interrupt raw bit. Triggered when there are not enough inlinks for DMA. (RO)

UHCI_OUTLINK_EOF_ERR_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in EOF in the transmit descriptor. (RO)

UHCI_OUT_TOTAL_EOF_INT_RAW This is the interrupt raw bit. Triggered when all data in the last buffer address has been sent out. (RO)

Continued on the next page...

Register 8.56: UHCI_INT_RAW_REG (0x0004)

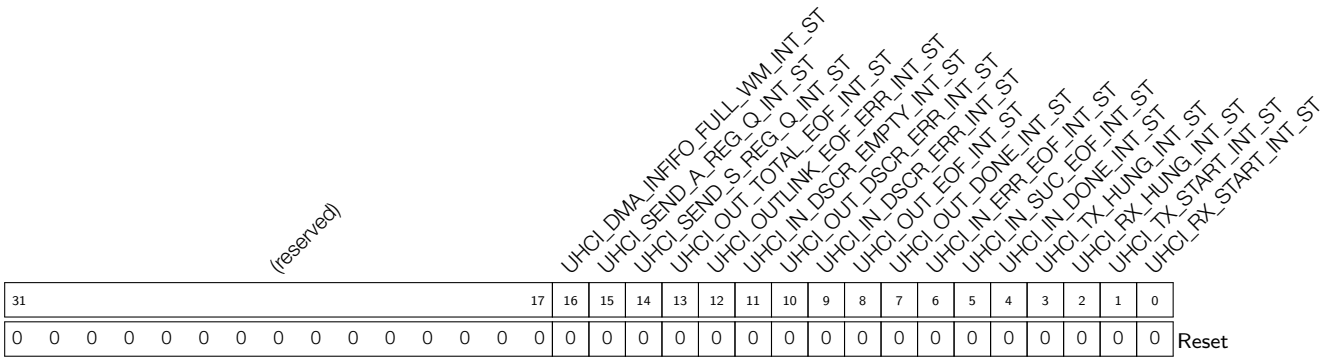
Continued from the previous page...

UHCI_SEND_S_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when DMA has sent out a short packet using single_send registers. (RO)

UHCI_SEND_A_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when DMA has sent out a short packet using always_send registers. (RO)

UHCI_DMA_INFIFO_FULL_WM_INT_RAW This is the interrupt raw bit. Triggered when the DMA INFIFO count has reached the configured threshold value. (RO)

Register 8.57: UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST This is the masked interrupt bit for UHCI_RX_START_INT interrupt when UHCI_RX_START_INT_ENA is set to 1. (RO)

UHCI_TX_START_INT_ST This is the masked interrupt bit for UHCI_TX_START_INT interrupt when UHCI_TX_START_INT_ENA is set to 1. (RO)

UHCI_RX_HUNG_INT_ST This is the masked interrupt bit for UHCI_RX_HUNG_INT interrupt when UHCI_RX_HUNG_INT_ENA is set to 1. (RO)

UHCI_TX_HUNG_INT_ST This is the masked interrupt bit for UHCI_TX_HUNG_INT interrupt when UHCI_TX_HUNG_INT_ENA is set to 1. (RO)

UHCI_IN_DONE_INT_ST This is the masked interrupt bit for UHCI_IN_DONE_INT interrupt when UHCI_IN_DONE_INT_ENA is set to 1. (RO)

UHCI_IN_SUC_EOF_INT_ST This is the masked interrupt bit for UHCI_IN_SUC_EOF_INT interrupt when UHCI_IN_SUC_EOF_INT_ENA is set to 1. (RO)

UHCI_IN_ERR_EOF_INT_ST This is the masked interrupt bit for UHCI_IN_ERR_EOF_INT interrupt when UHCI_IN_ERR_EOF_INT_ENA is set to 1. (RO)

UHCI_OUT_DONE_INT_ST This is the masked interrupt bit for UHCI_OUT_DONE_INT interrupt when UHCI_OUT_DONE_INT_ENA is set to 1. (RO)

UHCI_OUT_EOF_INT_ST This is the masked interrupt bit for UHCI_OUT_EOF_INT interrupt when UHCI_OUT_EOF_INT_ENA is set to 1. (RO)

UHCI_IN_DSCR_ERR_INT_ST This is the masked interrupt bit for UHCI_IN_DSCR_ERR_INT interrupt when UHCI_IN_DSCR_ERR_INT is set to 1. (RO)

UHCI_OUT_DSCR_ERR_INT_ST This is the masked interrupt bit for UHCI_OUT_DSCR_ERR_INT interrupt when UHCI_OUT_DSCR_ERR_INT_ENA is set to 1. (RO)

UHCI_IN_DSCR_EMPTY_INT_ST This is the masked interrupt bit for UHCI_IN_DSCR_EMPTY_INT interrupt when UHCI_IN_DSCR_EMPTY_INT_ENA is set to 1. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST This is the masked interrupt bit for UHCI_OUTLINK_EOF_ERR_INT interrupt when UHCI_OUTLINK_EOF_ERR_INT_ENA is set to 1. (RO)

UHCI_OUT_TOTAL_EOF_INT_ST This is the masked interrupt bit for UHCI_OUT_TOTAL_EOF_INT interrupt when UHCI_OUT_TOTAL_EOF_INT_ENA is set to 1. (RO)

Continued on the next page...

Register 8.57: UHCI_INT_ST_REG (0x0008)

Continued from the previous page...

UHCI_SEND_S_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_S_REG_Q_INT interrupt when UHCI_SEND_S_REG_Q_INT_ENA is set to 1. (RO)

UHCI_SEND_A_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_A_REG_Q_INT interrupt when UHCI_SEND_A_REG_Q_INT_ENA is set to 1. (RO)

UHCI_DMA_INFIFO_FULL_WM_INT_ST This is the masked interrupt bit for UHCI_DMA_INFIFO_FULL_WM_INT INTERRUPT when UHCI_DMA_INFIFO_FULL_WM_INT_ENA is set to 1. (RO)

Register 8.58: UHCI_INT_ENA_REG (0x000C)

UHCI_RX_START_INT_ENA This is the interrupt enable bit for UHCI_RX_START_INT interrupt. (R/W)

UHCI_TX_START_INT_ENA This is the interrupt enable bit for UHCI_TX_START_INT interrupt. (R/W)

UHCI_RX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_RX_HUNG_INT interrupt. (R/W)

UHCI_TX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_TX_HUNG_INT interrupt. (R/W)

UHCI_IN_DONE_INT_ENA This is the interrupt enable bit for UHCI_IN_DONE_INT interrupt. (R/W)

UHCI_IN_SUC_EOF_INT_ENA This is the interrupt enable bit for UHCI_IN_SUC_EOF_INT interrupt. (R/W)

UHCI_IN_ERR_EOF_INT_ENA This is the interrupt enable bit for UHCI_IN_ERR_EOF_INT interrupt. (R/W)

UHCI_OUT_DONE_INT_ENA This is the interrupt enable bit for UHCI_OUT_DONE_INT interrupt. (R/W)

UHCI_OUT_EOF_INT_ENA This is the interrupt enable bit for UHCI_OUT_EOF_INT interrupt. (R/W)

UHCI_IN_DSCR_ERR_INT_ENA This is the interrupt enable bit for UHCI_IN_DSCR_ERR_INT interrupt. (R/W)

UHCI_OUT_DSCR_ERR_INT_ENA This is the interrupt enable bit for UHCI_OUT_DSCR_ERR_INT interrupt. (R/W)

UHCI_IN_DSCR_EMPTY_INT_ENA This is the interrupt enable bit for UHCI_IN_DSCR_EMPTY_INT interrupt. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA This is the interrupt enable bit for UHCI_OUTLINK_EOF_ERR_INT interrupt. (R/W)

UHCI_OUT_TOTAL_EOF_INT_ENA This is the interrupt enable bit for UHCI_OUT_TOTAL_EOF_INT interrupt. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_S_REG_Q_INT interrupt. (R/W)

Continued on the next page...

Register 8.58: UHCI_INT_ENA_REG (0x000C)

Continued from the previous page...

UHCI_SEND_A_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_A_REG_Q_INT interrupt. (R/W)

UHCI_DMA_INFIFO_FULL_WM_INT_ENA This is the interrupt enable bit for UHCI_DMA_INFIFO_FULL_WM_INT interrupt. (R/W)

Register 8.59: UHCI_INT_CLR_REG (0x0010)

Bits	Field	Reset
31–17	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
16	UHCI_DMA_INFIFO_FULL_WM_INT_CLR	0
15	UHCI_SEND_A_REG_Q_INT_CLR	0
14	UHCI_SEND_S_REG_Q_INT_CLR	0
13	UHCI_OUT_TOTAL_EOF_INT_CLR	0
12	UHCI_OUTLINK_EOF_ERR_INT_CLR	0
11	UHCI_IN_DSCR_EMPTY_INT_CLR	0
10	UHCI_OUT_DSCR_ERR_INT_CLR	0
9	UHCI_IN_DSCR_ERR_INT_CLR	0
8	UHCI_OUT_EOF_INT_CLR	0
7	UHCI_OUT_DONE_INT_CLR	0
6	UHCI_IN_ERR_EOF_INT_CLR	0
5	UHCI_IN_SUC_EOF_INT_CLR	0
4	UHCI_IN_DONE_INT_CLR	0
3	UHCI_TX_HUNG_INT_CLR	0
2	UHCI_RX_HUNG_INT_CLR	0
1	UHCI_TX_START_INT_CLR	0
0	UHCI_RX_START_INT_CLR	0

UHCI_RX_START_INT_CLR Set this bit to clear UHCI_RX_START_INT interrupt. (WO)

UHCI_TX_START_INT_CLR Set this bit to clear UHCI_TX_START_INT interrupt. (WO)

UHCI_RX_HUNG_INT_CLR Set this bit to clear UHCI_RX_HUNG_INT interrupt. (WO)

UHCI_TX_HUNG_INT_CLR Set this bit to clear UHCI_TX_HUNG_INT interrupt. (WO)

UHCI_IN_DONE_INT_CLR Set this bit to clear UHCI_IN_DONE_INT interrupt. (WO)

UHCI_IN_SUC_EOF_INT_CLR Set this bit to clear UHCI_IN_SUC_EOF_INT interrupt. (WO)

UHCI_IN_ERR_EOF_INT_CLR Set this bit to clear UHCI_IN_ERR_EOF_INT interrupt. (WO)

UHCI_OUT_DONE_INT_CLR Set this bit to clear UHCI_OUT_DONE_INT interrupt. (WO)

UHCI_OUT_EOF_INT_CLR Set this bit to clear UHCI_OUT_EOF_INT interrupt. (WO)

UHCI_IN_DSCR_ERR_INT_CLR Set this bit to clear UHCI_IN_DSCR_ERR_INT interrupt. (WO)

UHCI_OUT_DSCR_ERR_INT_CLR Set this bit to clear UHCI_OUT_DSCR_ERR_INT interrupt. (WO)

UHCI_IN_DSCR_EMPTY_INT_CLR Set this bit to clear UHCI_IN_DSCR_EMPTY_INT interrupt. (WO)

UHCI_OUTLINK_EOF_ERR_INT_CLR Set this bit to clear UHCI_OUTLINK_EOF_ERR_INT interrupt. (WO)

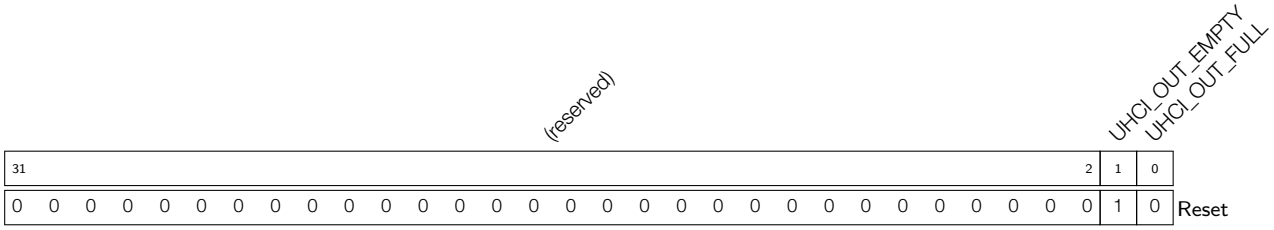
UHCI_OUT_TOTAL_EOF_INT_CLR Set this bit to clear UHCI_OUT_TOTAL_EOF_INT interrupt. (WO)

UHCI_SEND_S_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_S_UHCI_Q_INT interrupt. (WO)

UHCI_SEND_A_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_A_UHCI_Q_INT interrupt. (WO)

UHCI_DMA_INFIFO_FULL_WM_INT_CLR Set this bit to clear UHCI_DMA_INFIFO_FULL_WM_INT interrupt. (WO)

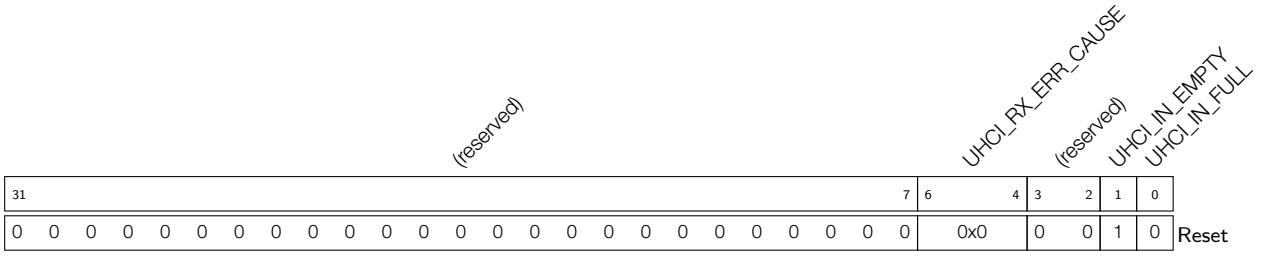
Register 8.60: UHCI_DMA_OUT_STATUS_REG (0x0014)



UHCI_OUT_FULL 1: DMA data-output FIFO is full. (RO)

UHCI_OUT_EMPTY 1: DMA data-output FIFO is empty. (RO)

Register 8.61: UHCI_DMA_IN_STATUS_REG (0x001C)

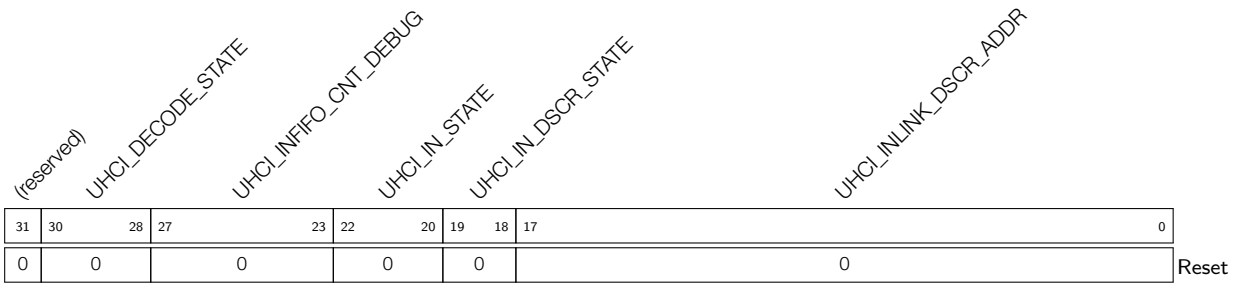


UHCI_IN_FULL Data-input FIFO full signal. (RO)

UHCI_IN_EMPTY Data-input FIFO empty signal. (RO)

UHCI_RX_ERR_CAUSE This register indicates the error type when DMA has received a packet with error. 3'b001: Checksum error in HCI packet; 3'b010: Sequence number error in HCI packet; 3'b011: CRC bit error in HCI packet; 3'b100: 0xc0 is found but received HCI packet is not end; 3'b101: 0xc0 is not found when receiving HCI packet is end; 3'b110: CRC check error. (RO)

Register 8.62: UHCI_STATE0_REG (0x0030)



UHCI_INLINK_DSCR_ADDR This register stores the current receive descriptor's address. (RO)

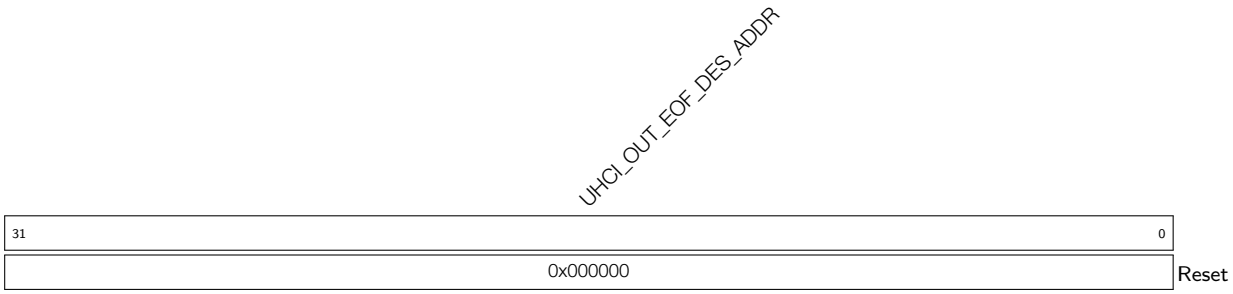
UHCI_IN_DSCR_STATE reserved (RO)

UHCI_IN_STATE reserved (RO)

UHCI_INFIFO_CNT_DEBUG This register stores the byte number of the data in the receive descriptor's FIFO. (RO)

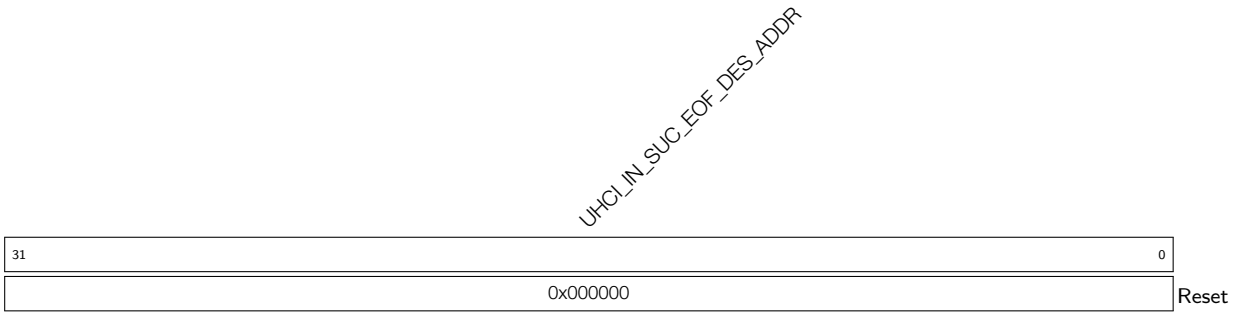
UHCI_DECODE_STATE UHCI decoder status. (RO)

Register 8.63: UHCI_DMA_OUT_EOF_DES_ADDR_REG (0x0038)



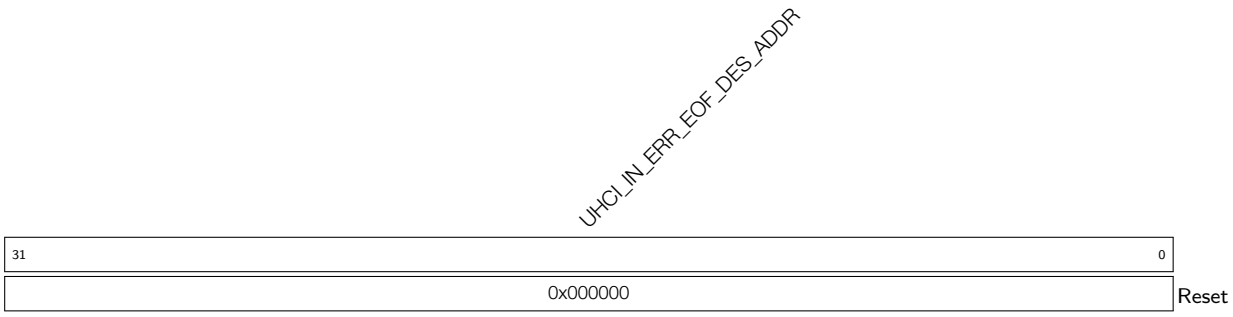
UHCI_OUT_EOF_DES_ADDR This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 8.64: UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG (0x003C)



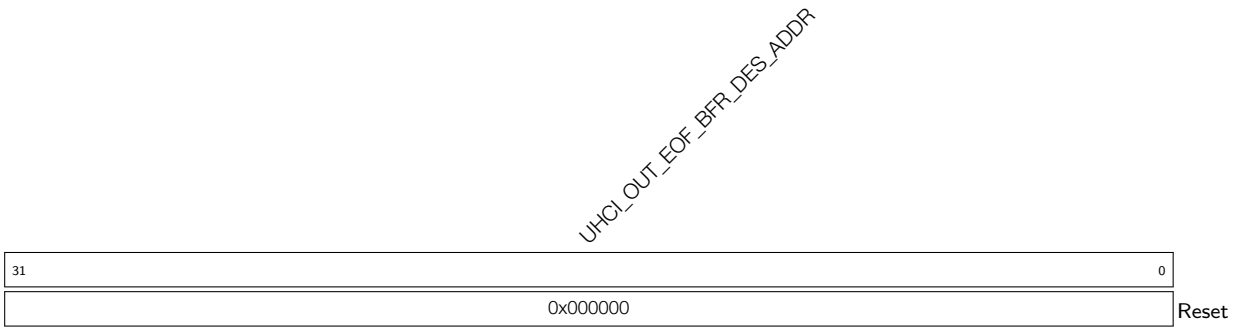
UHCI_IN_SUC_EOF_DES_ADDR This register stores the address of the receive descriptor when received successful EOF. (RO)

Register 8.65: UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG (0x0040)



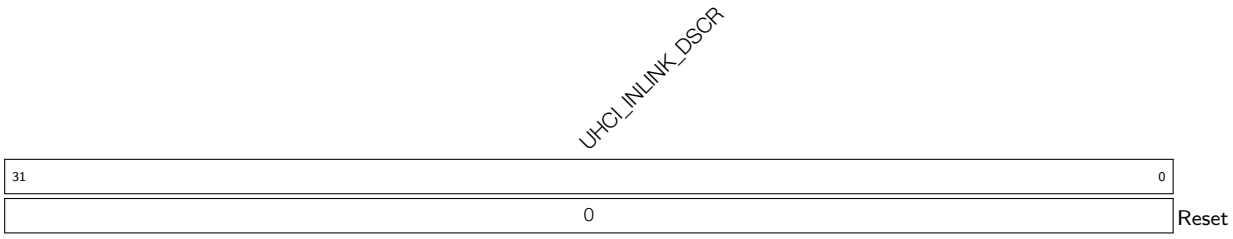
UHCI_IN_ERR_EOF_DES_ADDR This register stores the address of the receive descriptor when there are some errors in this descriptor. (RO)

Register 8.66: UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x0044)



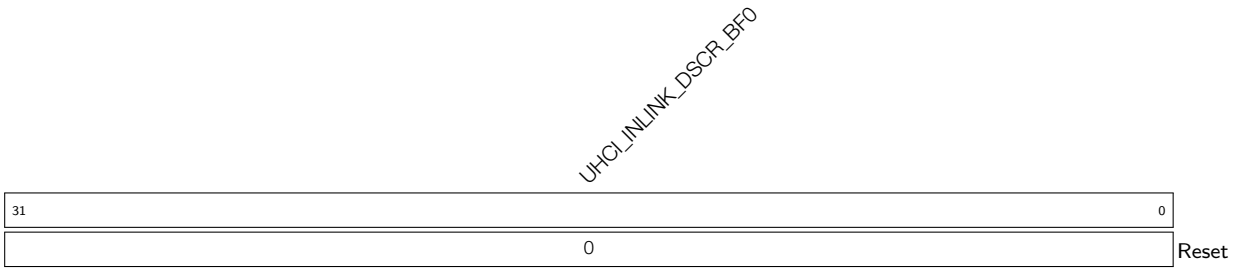
UHCI_OUT_EOF_BFR_DES_ADDR This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 8.67: UHCI_DMA_IN_DSCR_REG (0x004C)



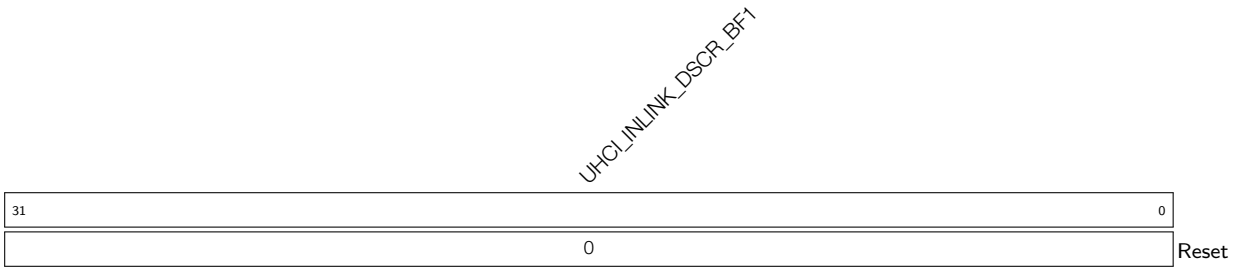
UHCI_INLINK_DSCR This register stores the third word of the next receive descriptor. (RO)

Register 8.68: UHCI_DMA_IN_DSCR_BF0_REG (0x0050)



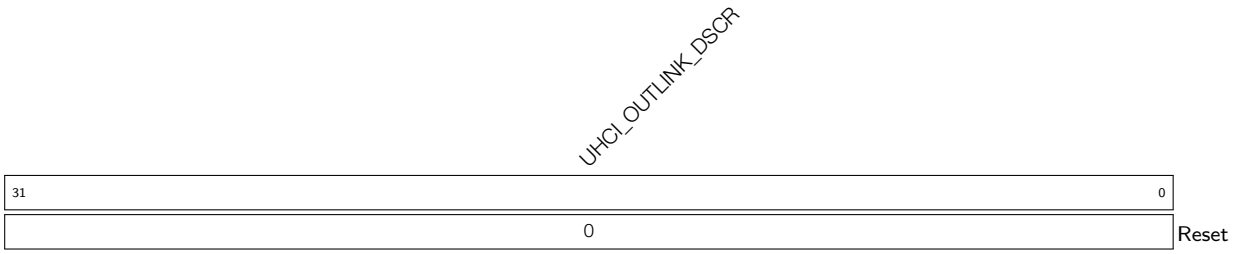
UHCI_INLINK_DSCR_BF0 This register stores the third word of the current receive descriptor. (RO)

Register 8.69: UHCI_DMA_IN_DSCR_BF1_REG (0x0054)



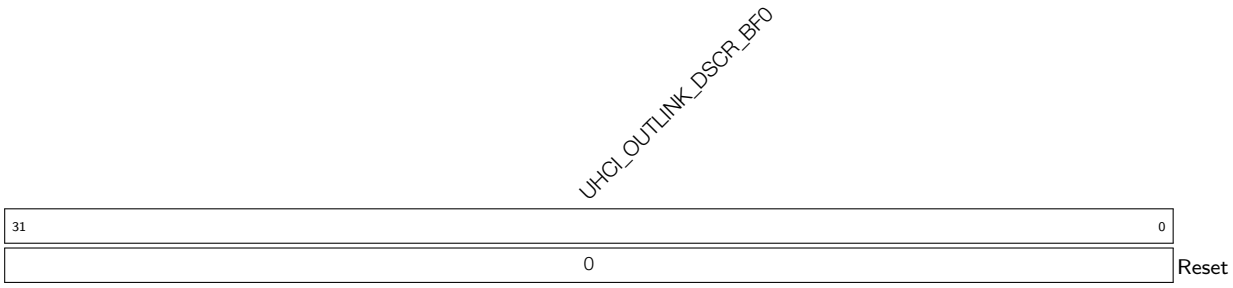
UHCI_INLINK_DSCR_BF1 This register stores the second word of the current receive descriptor. (RO)

Register 8.70: UHCI_DMA_OUT_DSCR_REG (0x0058)



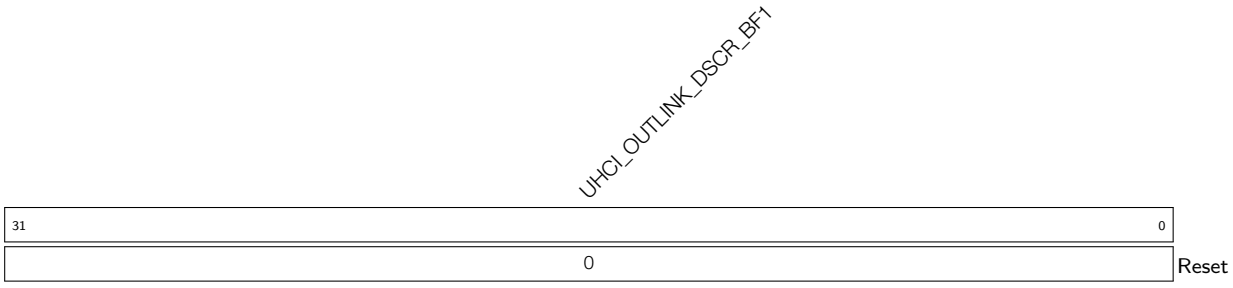
UHCI_OUTLINK_DSCR This register stores the third word of the next transmit descriptor. (RO)

Register 8.71: UHCI_DMA_OUT_DSCR_BF0_REG (0x005C)



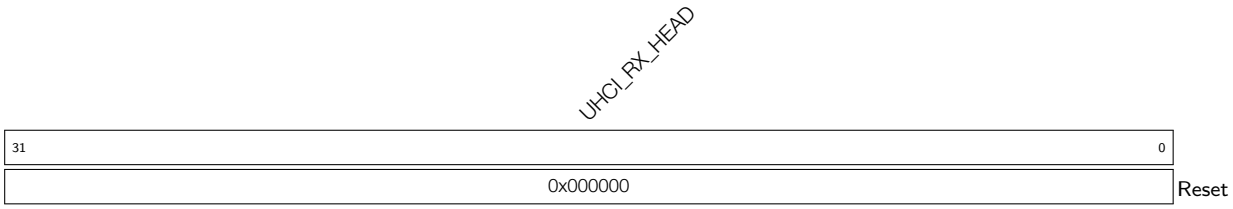
UHCI_OUTLINK_DSCR_BF0 This register stores the third word of the current transmit descriptor. (RO)

Register 8.72: UHCI_DMA_OUT_DSCR_BF1_REG (0x0060)



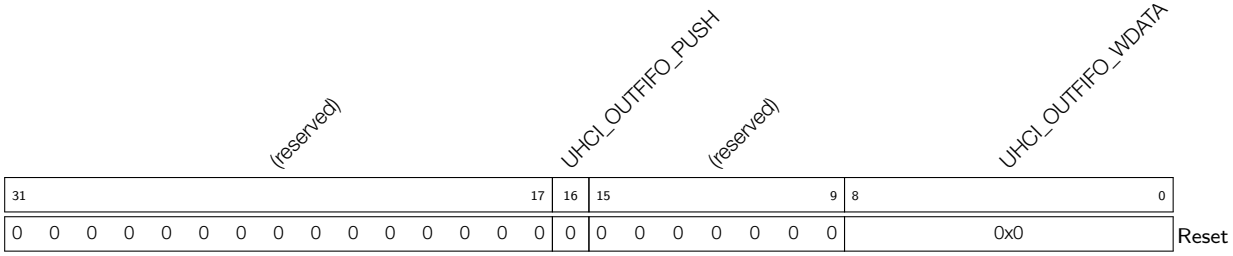
UHCI_OUTLINK_DSCR_BF1 This register stores the second word of the current transmit descriptor. (RO)

Register 8.73: UHCI_RX_HEAD_REG (0x0070)



UHCI_RX_HEAD This register stores the header of the current received packet. (RO)

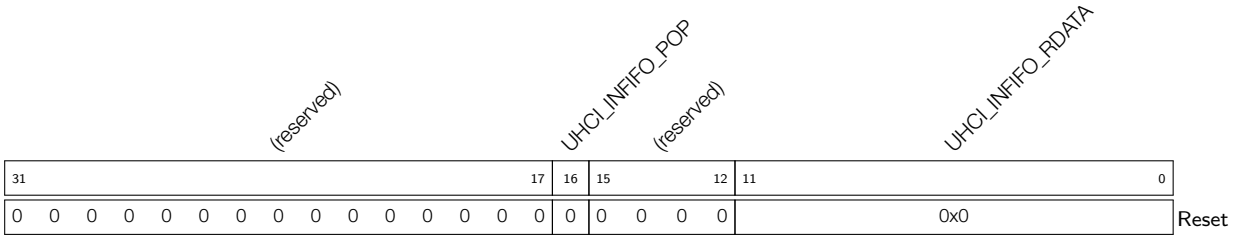
Register 8.74: UHCI_DMA_OUT_PUSH_REG (0x0018)



UHCI_OUTFIFO_WDATA This is the data that need to be pushed into data-output FIFO. (R/W)

UHCI_OUTFIFO_PUSH Set this bit to push data into the data-output FIFO. (R/W)

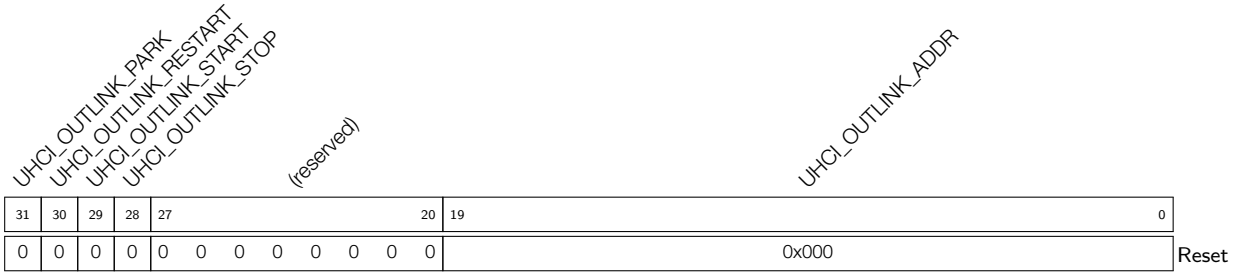
Register 8.75: UHCI_DMA_IN_POP_REG (0x0020)



UHCI_INFIFO_RDATA This register stores the data popping from data-input FIFO. (RO)

UHCI_INFIFO_POP Set this bit to pop data from data-input FIFO. (R/W)

Register 8.76: UHCI_DMA_OUT_LINK_REG (0x0024)



UHCI_OUTLINK_ADDR This register is used to specify the least significant 20 bits of the first transmit descriptor's address. (R/W)

UHCI_OUTLINK_STOP Set this bit to stop dealing with the transmit descriptor. (R/W)

UHCI_OUTLINK_START Set this bit to start a new transmit descriptor. (R/W)

UHCI_OUTLINK_RESTART Set this bit to restart the transmit descriptor from the last address. (R/W)

UHCI_OUTLINK_PARK 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

Register 8.77: UHCI_DMA_IN_LINK_REG (0x0028)

UHCI_INLINK_ADDR This register is used to specify the least significant 20 bits of the first receive descriptor's address. (R/W)

UHCI_INLINK_AUTO_RET This is the enable bit to return to current receive descriptor's address, when there are some errors in current packet. (R/W)

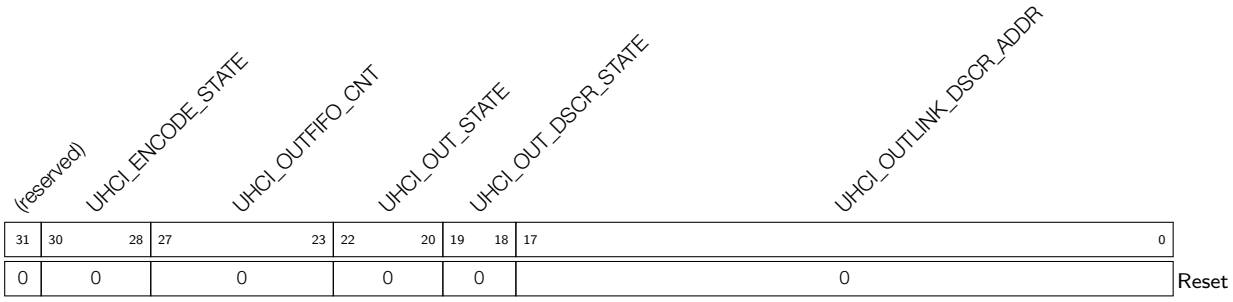
UHCI_INLINK_STOP Set this bit to stop dealing with the receive descriptors. (R/W)

UHCI_INLINK_START Set this bit to start dealing with the receive descriptors. (R/W)

UHCI_INLINK_RESTART Set this bit to restart new receive descriptors. (R/W)

UHCI_INLINK_PARK 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

Register 8.78: UHCI_STATE1_REG (0x0034)



UHCI_OUTLINK_DSCR_ADDR This register stores the current transmit descriptor's address. (RO)

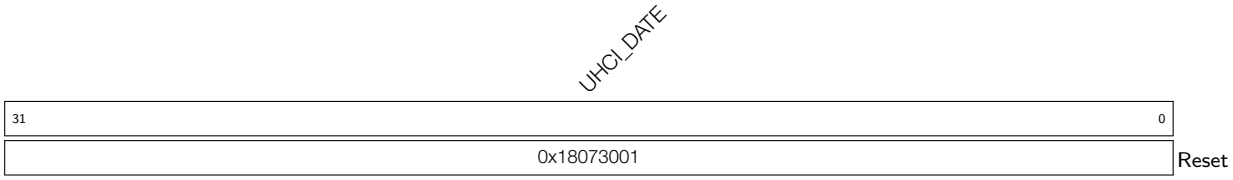
UHCI_OUT_DSCR_STATE reserved (RO)

UHCI_OUT_STATE reserved (RO)

UHCI_OUTFIFO_CNT This register stores the byte number of the data in the transmit descriptor's FIFO. (RO)

UHCI_ENCODE_STATE UHCI encoder status. (RO)

Register 8.79: UHCI_DATE_REG (0x00FC)



UHCI_DATE This is the version control register. (R/W)

9. LED PWM Controller

9.1 Overview

The LED PWM controller is primarily designed to control LED devices, as well as generate PWM signals. It has 14-bit timers and waveform generators.

9.2 Features

The LED PWM controller has the following features:

- Four independent timers that support division by fractions
- Eight independent waveform generators able to produce eight PWM signals
- Fading duty cycle of PWM signals without interference from any processors. An interrupt can be generated after the fade has completed
- Adjustable phase of PWM signal output
- PWM signal output in low-power mode

For the convenience of description, in the following sections the eight waveform generators are collectively referred to as PWM*n*, and the four timers are collectively referred to as Timer*x*.

9.3 Functional Description

9.3.1 Architecture

Figure 9-1 shows the architecture of the LED PWM controller. Figure 9-2 illustrates a PWM generator with its selected timer and a counter.

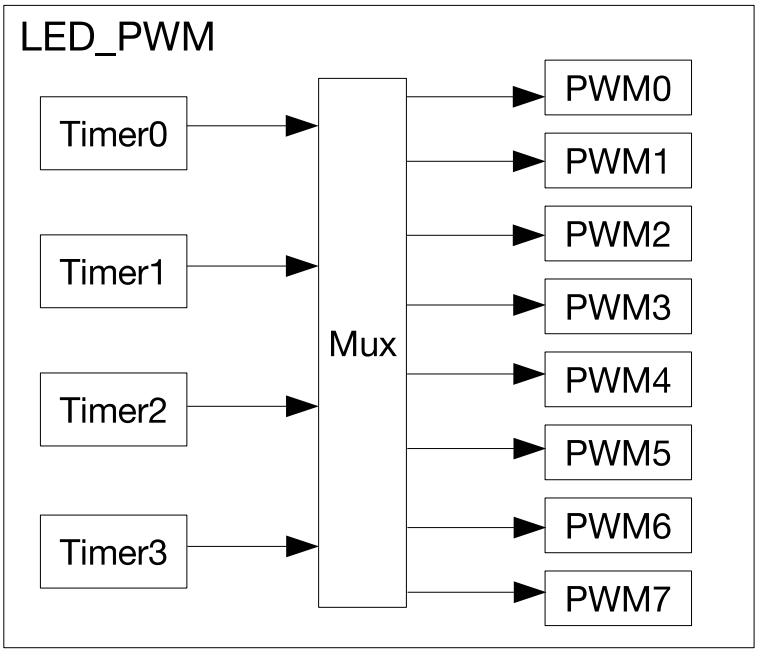


Figure 9-1. LED_PWM Architecture

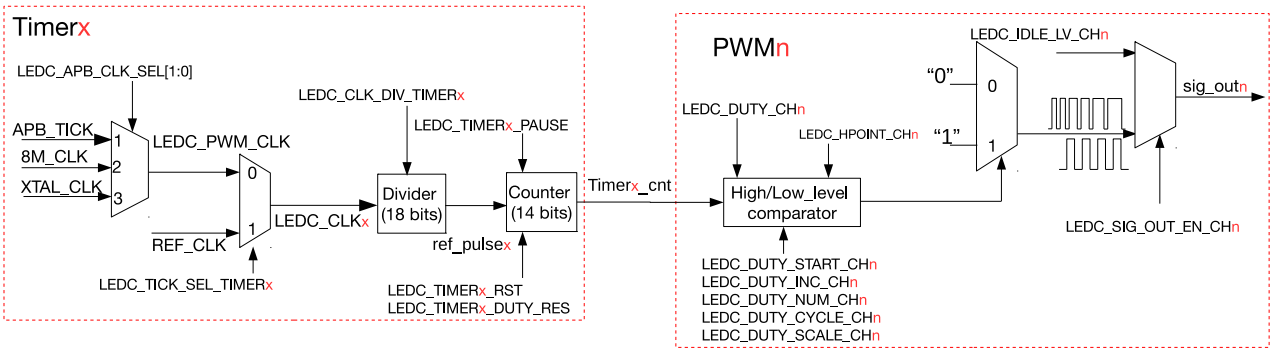


Figure 9-2. LED_PWM generator Diagram

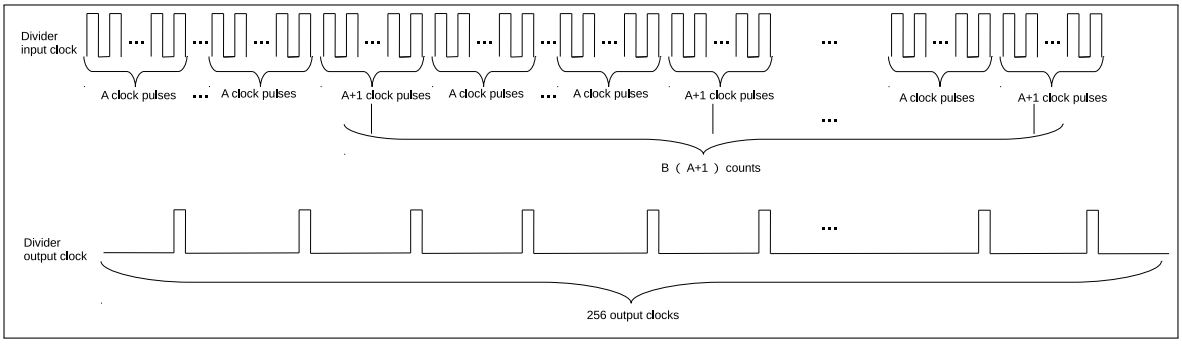


Figure 9-3. LED_PWM Divider

9.3.2 Timers

The clock of the LED PWM controller, LEDC_PWM_CLK, has three clock sources: APB_CLK, RTC8M_CLK and XTAL_CLK, selected by configuring LEDC_APB_CLK_SEL[1:0]. The clock of each LED PWM timer, LEDC_CLK*x*, has two clock sources: LEDC_PWM_CLK or REF_TICK. When REF_TICK is used as the clock source of a timer, LEDC_APB_CLK_SEL[1:0] should be set 1 and the cycle of REF_TICK should be an integral multiple of APB_CLK cycles. Otherwise this clock will be not accurate. For more information on the clock sources, please see Chapter Reset and Clock.

The output clock derived from LEDC_CLK*x* is used as the base clock for the counter. The divider's divisor is configured by LEDC_CLK_DIV_TIMER*x*. It is a fixed-point number: the highest 10 bits is the integer part represented as A, while the lowest eight bits is the fractional part represented as B. This divisor LEDC_CLK_DIV*x* is calculated as:

$$LEDC\_CLK\_DIVx = A + \frac{B}{256}$$

When the fractional part B is not 0, the input and output clock of the divider is shown as in figure 9-3. Among the 256 output clocks, B of them are divided by (A+1), whereas the remaining (256-B) are divided by A. Output clocks divided by (A+1) are evenly distributed in the total 256 output clocks.

The LED PWM controller has a 14-bit counter that counts up to $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$. If the counting value reaches $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$, the counter will overflow and restart counting from 0. The counting value can be reset, suspended or read by software. A LEDC_TIMER*x*_OVF_INT interrupt can be generated every time when the counter overflows or when it overflows for (LEDC_OVF_NUM_CH*n* + 1) times. The interrupt configuration is as follows:

1. Set LEDC_OVF_CNT_EN_CH*n*
2. Configure LEDC_OVF_NUM_CH*n* with the times of overflow minus 1

3. Set LEDC_OVF_CNT_CH*n*_INT_ENA
4. Set LEDC_TIMER*x*_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

The frequency of a PWM generator output signal, sig_out*n*, depends on both the divisor of the divider, as well as the range of the counter:

$$f_{\text{sig\_out}n} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV}x \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

To change the divisor and times of overflow, you should configure LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES respectively, and then set LEDC_TIMER*x*_PARA_UP; otherwise this change is not valid. The newly configured value is updated upon next overflow of the counter.

9.3.3 PWM Generators

As shown in figure 9-2, each PWM generator has a high/low level comparator and two selectors. A PWM generator takes the 14-bit counting value of the selected timer, compares it to values of the comparator Hpoint*n* and Lpoint*n*, and therefore control the level of PWM signals.

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Hpoint*n* is updated by LEDC_HPOINT_CH*n* when the counter overflows. The initial value of Lpoint*n* is the sum of LEDC_DUTY_CH*n*[18..4] and LEDC_HPOINT_CH*n* when the counter overflows. By configuring these two fields, the relative phase and the duty cycle of the PWM output can be set.

Figure 9-4 illustrates PWM's waveform when the duty cycle is fixed.

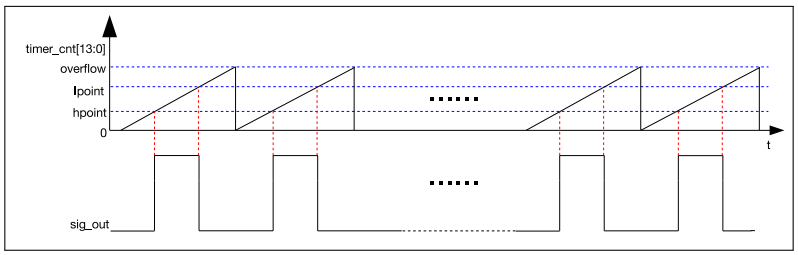


Figure 9-4. LED_PWM Output Signal Diagram

LEDC_DUTY_CH*n* is a fixed-point register with four fractional bits. LEDC_DUTY_CH*n*[18..4] is the integral part used directly for PWM calculation. LEDC_DUTY_CH*n*[3..0] is the fractional part used to dither the output. If LEDC_DUTY_CH*n*[3..0] is non-zero, then among every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3..0] have PWM pulses with width one timer cycle longer than that of (16-LEDC_DUTY_CH*n*[3..0]). This feature effectively increases the resolution of the PWM generator to 18 bits.

9.3.4 Duty Cycle Fading

The PWM generators is able to fade the duty cycle, that is to gradually change the duty cycle from one value to another. This is achieved by configuring LEDC_DUTY_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_DUTY_INC_CH*n*, LEDC_DUTY_NUM_CH*n* and LEDC_DUTY_SCALE_CH*n*.

LEDC_DUTY_START_CH*n* is used to update the value of Lpoint*n*. When this bit is set and the counter overflows, Lpoint*n* increments or decrements automatically, depending on whether the bit LEDC_DUTY_INC_CH*n* is set or cleared.

The duty cycle of sig_out*n* changes every LEDC_DUTY_CYCLE_CH*n* PWM pulse cycles by adding or subtracting the value of LEDC_DUTY_SCALE_CH*n*.

Figure 9-5 is a diagram of fading duty cycle. Upon reaching LEDC_DUTY_NUM_CH*n*, the fade stops and a LEDC_DUTY_CHNG_END_CH*n*_INT interrupt is generated. When configured like this, the duty cycle of sig_out*n* increases by LEDC_DUTY_SCALE_CH*n* every LEDC_DUTY_CYCLE_CH*n* PWM pulse cycles.

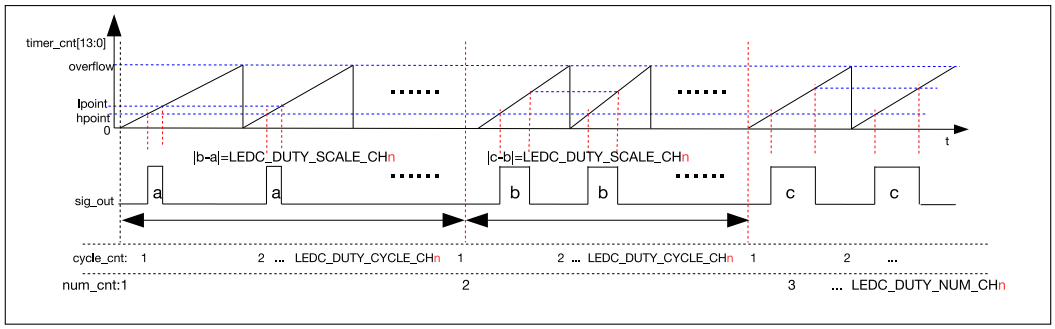


Figure 9-5. Output Signal Diagram of Fading Duty Cycle

LEDC_SIG_OUT_EN_CH*n* used to enable PWM waveform output. When LEDC_SIG_OUT_EN_CH*n* is 0, the level of sig_out*n* is constant as specified in LEDC_IDLE_LV_CH*n*.

If 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* and LEDC_OVF_CNT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* should be set to apply the new configuration.

9.3.5 Interrupts

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for (LEDC_OVF_NUM_CH*n* + 1) times and register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on a LED PWM generator has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when a LED PWM timer has reached its maximum counter value.

9.4 Base Address

Users can access the LED PWM controller with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 40: LED_PWM Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F419000
PeriBUS2	0x60019000

9.5 Register Summary

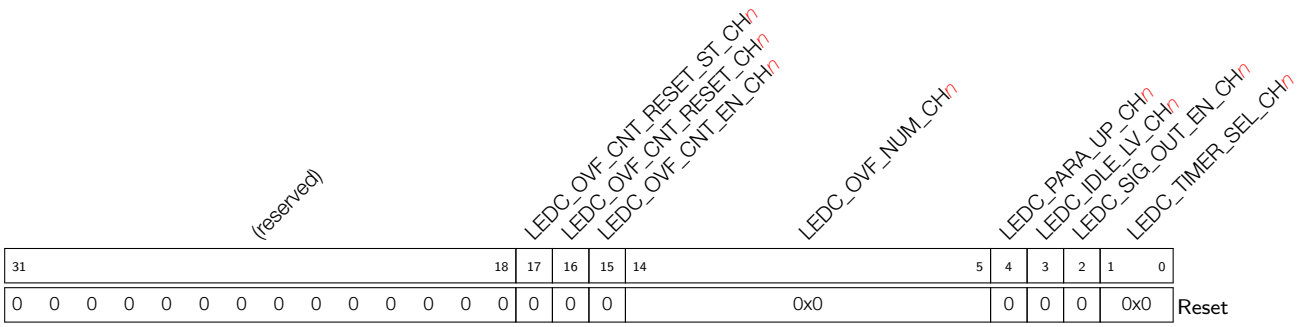
The addresses in the following table are relative to the LED PWM controller base addresses provided in Section 9.4.

Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	R/W
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	R/W
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	R/W
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	R/W
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	R/W
LEDC_CH6_CONF0_REG	Configuration register 0 for channel 6	0x0078	varies
LEDC_CH6_CONF1_REG	Configuration register 1 for channel 6	0x0084	R/W
LEDC_CH7_CONF0_REG	Configuration register 0 for channel 7	0x008C	varies
LEDC_CH7_CONF1_REG	Configuration register 1 for channel 7	0x0098	R/W
LEDC_CONF_REG	Global ledc configuration register	0x00D0	R/W
Hpoint Register			
LEDC_CH0_HPOINT_REG	High point register for channel 0	0x0004	R/W
LEDC_CH1_HPOINT_REG	High point register for channel 1	0x0018	R/W
LEDC_CH2_HPOINT_REG	High point register for channel 2	0x002C	R/W
LEDC_CH3_HPOINT_REG	High point register for channel 3	0x0040	R/W
LEDC_CH4_HPOINT_REG	High point register for channel 4	0x0054	R/W
LEDC_CH5_HPOINT_REG	High point register for channel 5	0x0068	R/W
LEDC_CH6_HPOINT_REG	High point register for channel 6	0x007C	R/W
LEDC_CH7_HPOINT_REG	High point register for channel 7	0x0090	R/W
Duty Cycle Register			
LEDC_CH0_DUTY_REG	Initial duty cycle for channel 0	0x0008	R/W
LEDC_CH0_DUTY_R_REG	Current duty cycle for channel 0	0x0010	RO
LEDC_CH1_DUTY_REG	Initial duty cycle for channel 1	0x001C	R/W
LEDC_CH1_DUTY_R_REG	Current duty cycle for channel 1	0x0024	RO
LEDC_CH2_DUTY_REG	Initial duty cycle for channel 2	0x0030	R/W
LEDC_CH2_DUTY_R_REG	Current duty cycle for channel 2	0x0038	RO
LEDC_CH3_DUTY_REG	Initial duty cycle for channel 3	0x0044	R/W
LEDC_CH3_DUTY_R_REG	Current duty cycle for channel 3	0x004C	RO
LEDC_CH4_DUTY_REG	Initial duty cycle for channel 4	0x0058	R/W
LEDC_CH4_DUTY_R_REG	Current duty cycle for channel 4	0x0060	RO
LEDC_CH5_DUTY_REG	Initial duty cycle for channel 5	0x006C	R/W
LEDC_CH5_DUTY_R_REG	Current duty cycle for channel 5	0x0074	RO

Name	Description	Address	Access
LEDC_CH6_DUTY_REG	Initial duty cycle for channel 6	0x0080	R/W
LEDC_CH6_DUTY_R_REG	Current duty cycle for channel 6	0x0088	RO
LEDC_CH7_DUTY_REG	Initial duty cycle for channel 7	0x0094	R/W
LEDC_CH7_DUTY_R_REG	Current duty cycle for channel 7	0x009C	RO
Timer Register			
LEDC_TIMER0_CONF_REG	Timer 0 configuration	0x00A0	varies
LEDC_TIMER0_VALUE_REG	Timer 0 current counter value	0x00A4	RO
LEDC_TIMER1_CONF_REG	Timer 1 configuration	0x00A8	varies
LEDC_TIMER1_VALUE_REG	Timer 1 current counter value	0x00AC	RO
LEDC_TIMER2_CONF_REG	Timer 2 configuration	0x00B0	varies
LEDC_TIMER2_VALUE_REG	Timer 2 current counter value	0x00B4	RO
LEDC_TIMER3_CONF_REG	Timer 3 configuration	0x00B8	varies
LEDC_TIMER3_VALUE_REG	Timer 3 current counter value	0x00BC	RO
Interrupt Register			
LEDC_INT_RAW_REG	Raw interrupt status	0x00C0	RO
LEDC_INT_ST_REG	Masked interrupt status	0x00C4	RO
LEDC_INT_ENA_REG	Interrupt enable bits	0x00C8	R/W
LEDC_INT_CLR_REG	Interrupt clear bits	0x00CC	WO
Version Register			
LEDC_DATE_REG	Version control register	0x00FC	R/W

9.6 Registers

Register 9.1: LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+20n*)**



LEDC_TIMER_SEL_CH*n* This field is used to select one of timers for channel *n*. 0: select timer0 1: select timer1 2: select timer2 3: select timer3 (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. (R/W)

LEDC_PARA_UP_CH*n* This bit is used to update register LEDC_CH*n*_HPOINT and LEDC_CH*n*_DUTY for channel *n*. (WO)

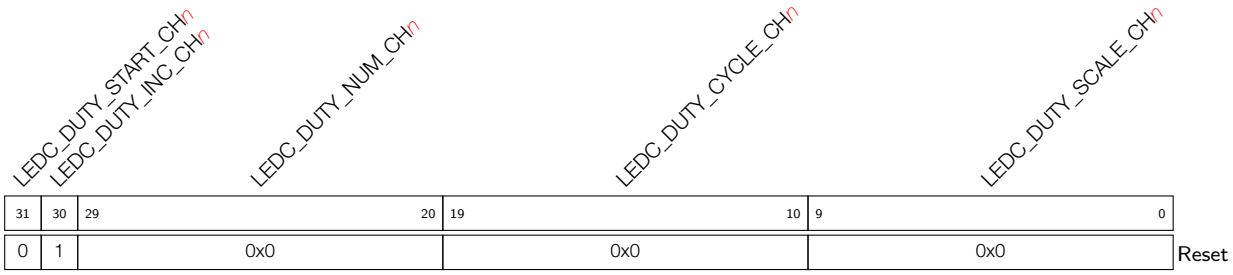
LEDC_OVF_NUM_CH*n* This register is used to configure the maximum times of overflow minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* This bit is used to enable the ovf_cnt of channel *n*. (R/W)

LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the ovf_cnt of channel *n*. (WO)

LEDC_OVF_CNT_RESET_ST_CH*n* This is the status bit of LEDC_OVF_CNT_RESET_CH*n*. (RO)

Register 9.2: LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+20n*)**



LEDC_DUTY_SCALE_CH*n* This register is used to configure the changing step scale of duty on channel *n*. (R/W)

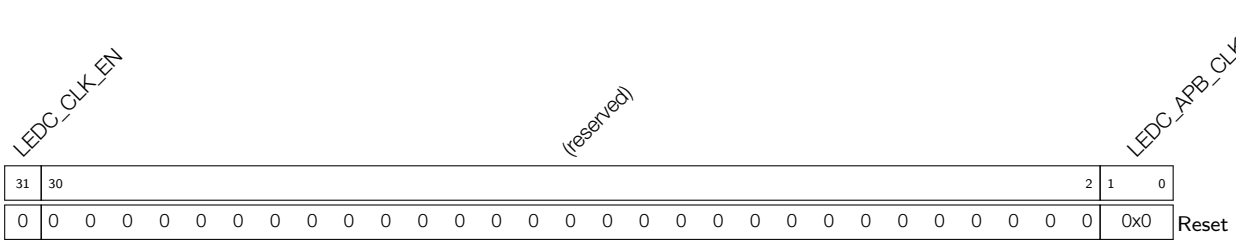
LEDC_DUTY_CYCLE_CH*n* The duty will change every LEDC_DUTY_CYCLE_CH*n* on channel *n*. (R/W)

LEDC_DUTY_NUM_CH*n* This register is used to control the number of times the duty cycle will be changed. (R/W)

LEDC_DUTY_INC_CH*n* This register is used to increase or decrease the duty of output signal on channel *n*. 1: Increase; 0: Decrease. (R/W)

LEDC_DUTY_START_CH*n* Other configured fields in LEDC_CH*n*_CONF1_REG will start to take effect when this bit is set to 1. (R/W)

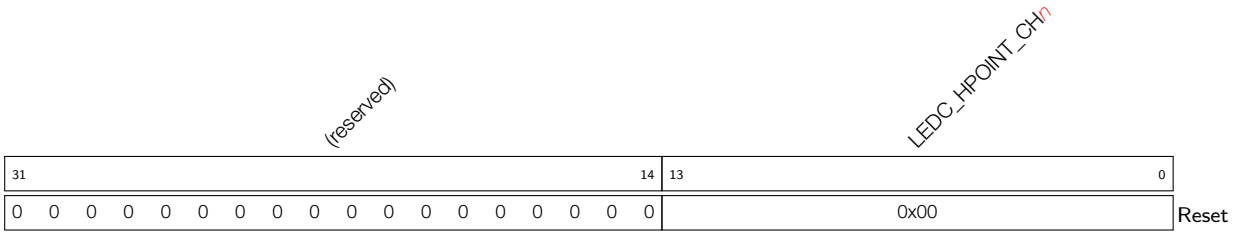
Register 9.3: LEDC_CONF_REG (0x00D0)



LEDC_APB_CLK_SEL This bit is used to select clock source for the 4 timers. 2'd1: APB_CLK 2'd2: RTC8M_CLK 2'd3: XTAL_CLK (R/W)

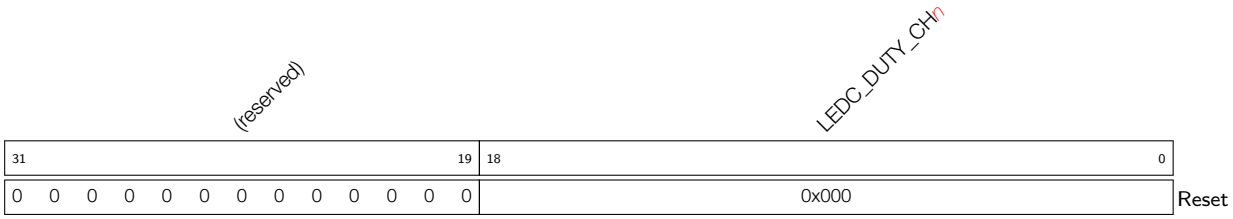
LEDC_CLK_EN This bit is used to control clock. 1'b1: Force clock on for register. 1'h0: Support clock only when application writes registers. (R/W)

Register 9.4: LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+20n*)**



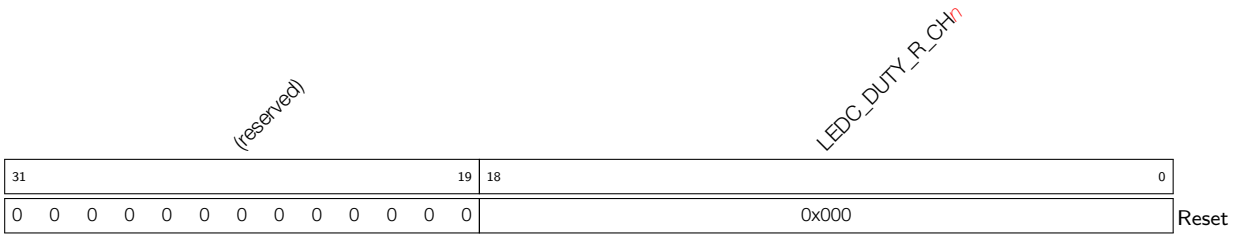
LEDC_HPOINT_CH*n* The output value changes to high when the selected timers has reached the value specified by this register. (R/W)

Register 9.5: LEDC_CH*n*_DUTY_REG (*n*: 0-7) (0x0008+20n*)**



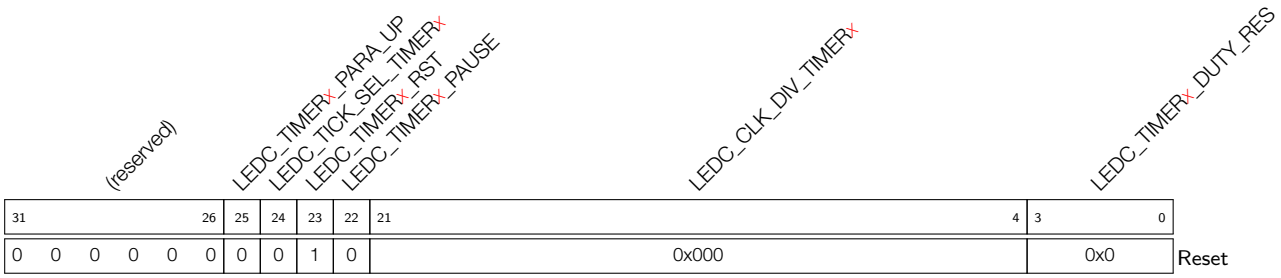
LEDC_DUTY_CH*n* This register is used to change the output duty by controlling the Lpoint. The output value turns to low when the selected timers has reached the Lpoint. (R/W)

Register 9.6: LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+20n*)**



LEDC_DUTY_R_CH*n* This register stores the current duty of output signal on channel *n*. (RO)

Register 9.7: LEDC_TIMERx_CONF_REG (x: 0-3) (0x00A0+8*x)



LEDC_TIMERx_DUTY_RES This register is used to control the range of the counter in timer *x*. (R/W)

LEDC_CLK_DIV_TIMERx This register is used to configure the divisor for the divider in timer *x*. The least significant eight bits represent the fractional part. (R/W)

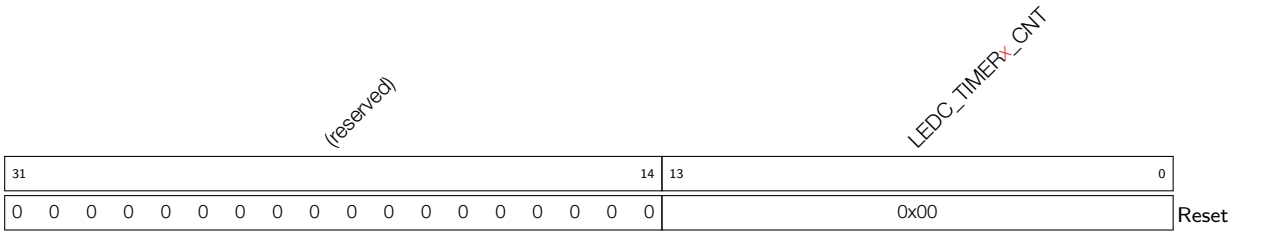
LEDC_TIMERx_PAUSE This bit is used to suspend the counter in timer *x*. (R/W)

LEDC_TIMERx_RST This bit is used to reset timer *x*. The counter will show 0 after reset. (R/W)

LEDC_TICK_SEL_TIMERx This bit is used to select clock for timer *x*. When this bit is set to 1 LEDC_APB_CLK_SEL[1:0] should be 1, otherwise the timer clock may be not accurate. 1'h0: SLOW_CLK 1'h1: REF_TICK (R/W)

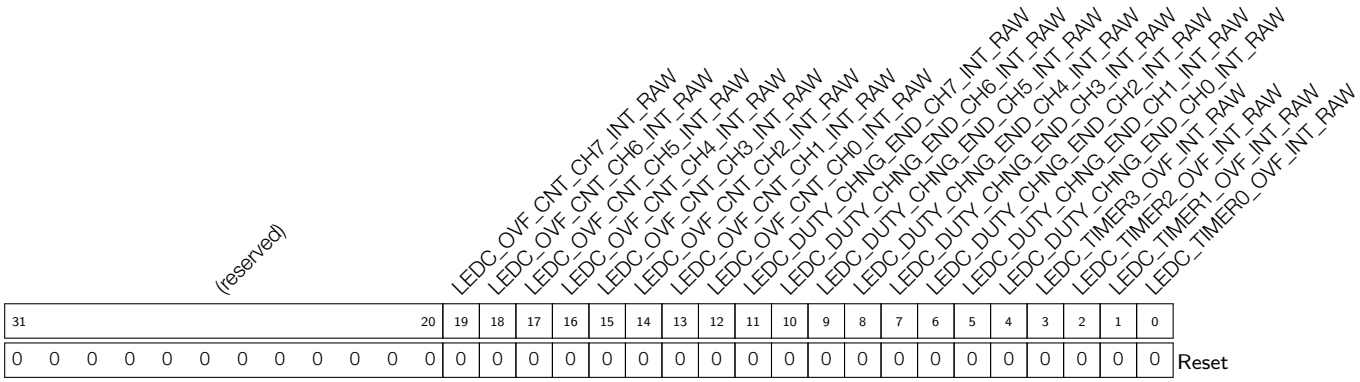
LEDC_TIMERx_PARA_UP Set this bit to update LEDC_CLK_DIV_TIMERx and LEDC_TIMERx_DUTY_RES. (WO)

Register 9.8: LEDC_TIMERx_VALUE_REG (x: 0-3) (0x00A4+8*x)



LEDC_TIMERx_CNT This register stores the current counter value of timer *x*. (RO)

Register 9.9: LEDC_INT_RAW_REG (0x00C0)

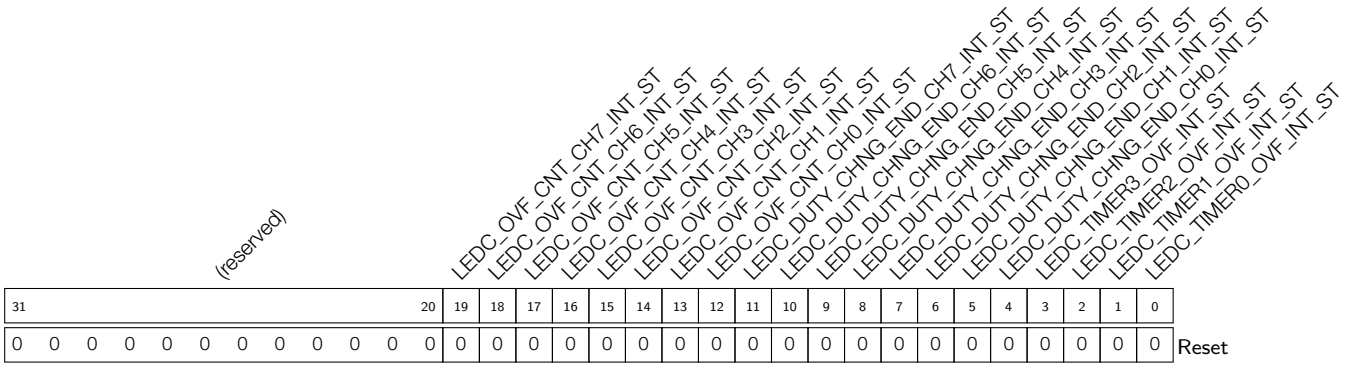


LEDC_TIMER*x*_OVF_INT_RAW Triggered when the timer*x* has reached its maximum counter value. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_RAW Interrupt raw bit for channel *n*. Triggered when the gradual change of duty has finished. (RO)

LEDC_OVF_CNT_CH*n*_INT_RAW Interrupt raw bit for channel *n*. Triggered when the ovf_cnt has reached the value specified by LEDC_OVF_NUM_CH*n*. (RO)

Register 9.10: LEDC_INT_ST_REG (0x00C4)

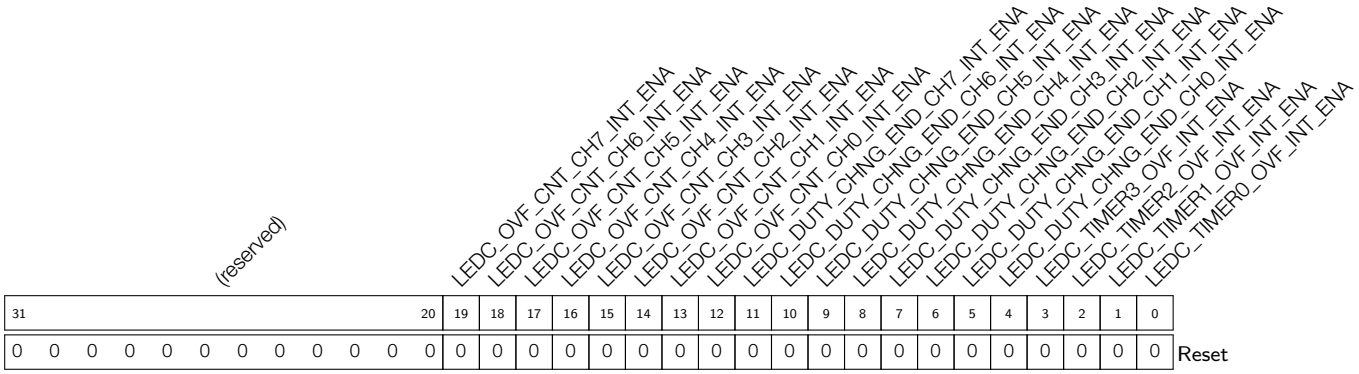


LEDC_TIMER*x*_OVF_INT_ST This is the masked interrupt status bit for the LEDC_TIMER*x*_OVF_INT interrupt when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST This is the masked interrupt status bit for the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt when LEDC_DUTY_CHNG_END_CH*n*_INT_ENAis set to 1. (RO)

LEDC_OVF_CNT_CH*n*_INT_ST This is the masked interrupt status bit for the LEDC_OVF_CNT_CH*n*_INT interrupt when LEDC_OVF_CNT_CH*n*_INT_ENA is set to 1. (RO)

Register 9.11: LEDC_INT_ENA_REG (0x00C8)

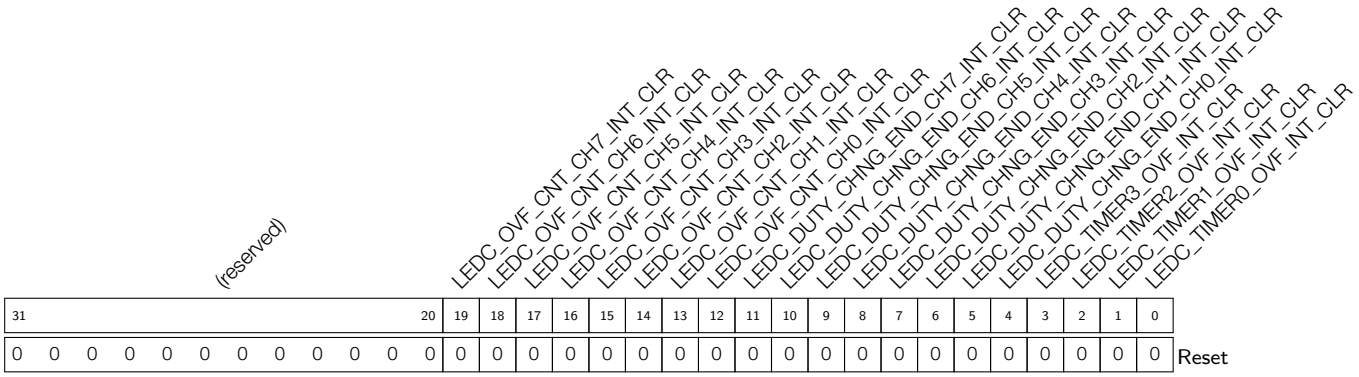


LEDC_TIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_TIMER*x*_OVF_INT interrupt. (R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. (R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA The interrupt enable bit for the LEDC_OVF_CNT_CH*n*_INT interrupt. (R/W)

Register 9.12: LEDC_INT_CLR_REG (0x00CC)

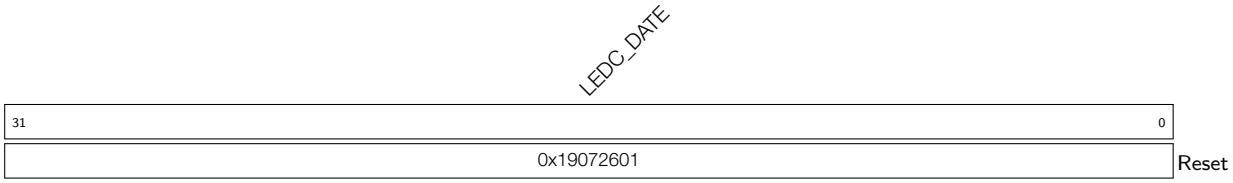


LEDC_TIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_TIMER*x*_OVF_INT interrupt. (WO)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. (WO)

LEDC_OVF_CNT_CH*n*_INT_CLR Set this bit to clear the LEDC_OVF_CNT_CH*n*_INT interrupt. (WO)

Register 9.13: LEDC_DATE_REG (0x00FC)



LEDC_DATE This is the version control register. (R/W)

10. Remote Control Peripheral

10.1 Introduction

The RMT (Remote Control) module is designed to send/receive infrared remote control signals that support for a variety of remote control protocols. The RMT module converts pulse codes stored in the module's built-in RAM into output signals, or converts input signals into pulse codes and stores them back in RAM. In addition, the RMT module optionally modulates its output signals with a carrier wave, or optionally filters its input signals.

The RMT module has four channels, numbered from zero to three. Each channel has the same functionality controlled by dedicated set of registers and is able to independently transmit or receive data. Registers in each channel are indicated by *n* which is used as a placeholder for the channel number.

10.2 Functional Description

10.2.1 RMT Architecture

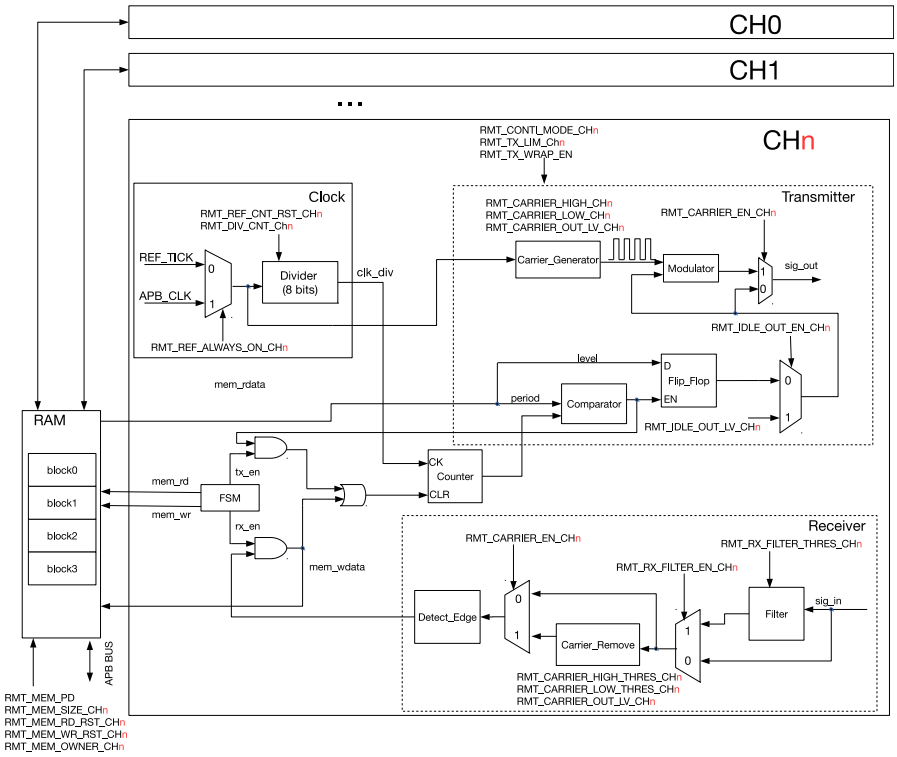


Figure 10-1. RMT Architecture

The RMT module contains four independent channels. Each channel has a clock divider, a counter, a transmitter and a receiver. As for the transmitter and the receiver of a single channel, only one of them can be active. The four channels share a 256 x 32-bit RAM.

10.2.2 RMT RAM

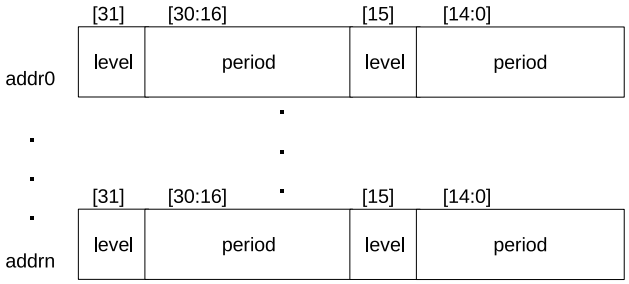


Figure 10-2. Format of Pulse Code in RAM

The format of pulse code in RAM is shown in Figure 10-2. Each pulse code contains a 16-bit entry with two fields, level and period. Level (0 or 1) indicates a high-/low-level value was received or is going to be sent, while "period" points out the clock cycles (see Figure 10-1 clk_div) for which the level lasts. A zero period is interpreted as a transmission end-marker.

The RAM is divided into four 64 x 32-bit blocks. By default, each channel uses one block (block zero for channel zero, block one for channel one, and so on). Usually, only one block of 64 x 32-bit worth of data can be sent or received in channel *n*. If the data size is larger than this block size, users can configure the channel to enable wrap mode or to use more blocks by setting RMT_MEM_SIZE_CH*n*. Setting RMT_MEM_SIZE_CH*n* > 1 will prompt channel *n* to use the memory of subsequent channels, block (*n*) ~ block (*n* + RMT_MEM_SIZE_CH*n* -1). If so, the subsequent channels *n* +1 ~ *n* + RMT_MEM_SIZE_CH*n* - 1 cannot be used once their RAM blocks are occupied. Note that the RAM used by each channel is mapped from low address to high address. In such mode, channel 0 is able to use the RAM blocks for channels 1, 2 and 3 by setting RMT_MEM_SIZE_CH*n*, but channel 3 cannot use the blocks for channels 0, 1, or 2.

The RMT RAM can be accessed via APB bus, or read by the transmitter and written by the receiver. To protect a receiver from overwriting the blocks a transmitter is about to transmit, RMT_MEM_OWNER_CH*n* can be configured to designate the block's owner, be it a transmitter or receiver. This way, if this ownership is violated, an RMT_MEM_OWNER_ERR_CH*n* flag will be generated.

When the RMT module is inactive, the RAM can be put into low-power mode by setting RMT_MEM_FORCE_PD.

10.2.3 Clock

The drive clock of a divider is generated by taking either the APB_CLK or REF_TICK according to the state of RMT_REF_ALWAYS_ON_CH*n*. (For more information on clock sources, please see Chapter Reset and Clock). Divider value is normally equal to the value of RMT_DIV_CNT_CH*n*, except value 0 that represents divider 256. The clock divider can be reset to zero by clearing RMT_REF_CNT_RST_CH*n*. The clock generated from the divider can be used by the clock counter (see Figure 10-2).

10.2.4 Transmitter

When RMT_TX_START_CH*n* is set to 1, the transmitter of channel *n* will start reading and sending pulse codes from the starting address of its RAM block. The codes are sent starting from low-address entry. The transmitter will stop the transmission, return to idle state and generate an RMT_CH*n*_TX_END_INT interrupt, when an end-marker (a zero period) is encountered. Also, setting RMT_TX_STOP_CH*n* to 1 stops the transmission and immediately sets the transmitter back to idle. The output level of a transmitter in idle state is determined by the "level" field of the end-marker or by the content of RMT_IDLE_OUT_LV_CH*n*, depending on the configuration of

RMT_IDLE_OUT_EN_CH*n*.

To transmit more pulse codes than can be fitted in the channel's RAM, users can enable wrap mode by configuring RMT_MEM_TX_WRAP_EN. In this mode, when the transmitter has reached the end-marker in the channel's memory, it will loop back to the first byte. For example, if RMT_MEM_SIZE_CH*n* is set to 1, the transmitter will start sending data from the address 64 * *n*, and then the data from higher RAM address. Once the transmitter finishes sending the data from (64 * (*n* +1) - 1), it will continue sending data from 64 * *n* till encounters an end-marker. Wrap mode is also applicable for RMT_MEM_SIZE_CH*n* > 1.

An RMT_CH*n*_TX_THR_EVENT_INT interrupt will be generated whenever the size of transmitted pulse codes is larger than or equal to the value set by RMT_TX_LIM_CH*n*. In wrap mode, RMT_TX_LIM_CH*n* can be set to a half or a fraction of the size of the channel's RAM block. When an RMT_CH*n*_TX_THR_EVENT_INT interrupt is detected, the already used RAM region should be updated by subsequent user defined events. Therefore, when the wrap mode happens the transmitter will seamlessly continue sending the new events.

The output of the transmitter can be modulated using a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier waveform is configurable. In a carrier cycle, the high level lasts for (RMT_CARRIER_HIGH_CH*n* +1) clock cycles of APB_CLK or REF_TICK, while the low level lasts for (RMT_CARRIER_LOW_CH*n* +1) clock cycles of APB_CLK or REF_TICK. When RMT_CARRIER_OUT_LV_CH*n* is set to 1, carrier wave will be added on high-level output signals; while RMT_CARRIER_OUT_LV_CH*n* is set to 0, carrier wave will be added on low-level output signals. Carrier wave can be added on output signals during modulation, or just added on valid pulse codes (the data stored in RAM), which can be set by configuring RMT_CARRIER_EFF_EN_CH*n*.

The continuous transmission of the transmitter can be enabled by setting RMT_TX_CONTI_MODE_CH*n*. When this register is set, the transmitter will send the pulse codes from RAM in loops. If RMT_TX_LOOP_CNT_EN_CH*n* is set to 1, the transmitter will start counting loop times. Once the counting reaches the value of register RMT_TX_LOOP_NUM_CH*n*, an RMT_CH*n*_TX_LOOP_INT interrupt will be generated.

Setting RMT_TX_SIM_EN to 1 will enable multiple channels to start sending data simultaneously. RMT_TX_SIM_CH*n* will choose which multiple channels are enabled to send data simultaneously.

10.2.5 Receiver

When RMT_RX_EN_CH*n* is set to 1, the receiver in channel *n* becomes active, detecting signal levels and measuring clock cycles the signals lasts for. These data will be written in RAM in the form of pulse codes. Receiving ends, when the receiver detects no change in a signal level for a number of clock cycles more than the value set by RMT_IDLE_THRES_CH*n*. The receiver will return to idle state and generate an RMT_CH*n*_RX_END_INT interrupt.

The receiver has an input signal filter which can be enabled by configuring RMT_RX_FILTER_EN_CH*n*. The filter samples input signals continuously, and will detect the signals which remain unchanged for a continuous RMT_RX_FILTER_THRES_CH*n* APB clock cycles as valid, otherwise, the signals will be detected as invalid. Only the valid signals can pass through this filter. The filter will remove pulses with a length of less than RMT_RX_FILTER_THRES_CH*n* APB clock cycles.

10.2.6 Interrupts

- RMT_CH*n*_ERR_INT: Triggered when channel *n* does not read or write data correctly. For example, if the transmitter still tries to read data from RAM when the RAM is empty, or the receiver still tries to write data into RAM when the RAM is full, this interrupt will be triggered.
- RMT_CH*n*_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the

value of RMT_TX_LIM_CH*n*.

- RMT_CH*n*_TX_END_INT: Triggered when the transmitter has finished transmitting signals.
- RMT_CH*n*_RX_END_INT: Triggered when the receiver has finished receiving signals.
- RMT_CH*n*_TX_LOOP_INT: Triggered when the loop counting reaches the value set by RMT_TX_LOOP_NUM_CH*n*.

10.3 Base Address

Users can access RMT with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 42: RMT Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F416000
PeriBUS2	0x60016000

10.4 Register Summary

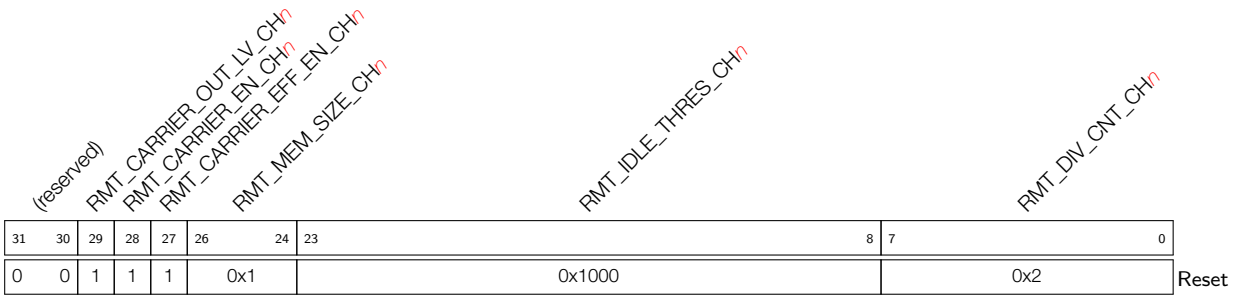
The addresses in the following table are relative to RMT base addresses provided in Section 10.3.

Name	Description	Address	Access
Configuration registers			
RMT_CH0CONF0_REG	Channel 0 configuration register 0	0x0010	R/W
RMT_CH0CONF1_REG	Channel 0 configuration register 1	0x0014	varies
RMT_CH1CONF0_REG	Channel 1 configuration register 0	0x0018	R/W
RMT_CH1CONF1_REG	Channel 1 configuration register 1	0x001C	varies
RMT_CH2CONF0_REG	Channel 2 configuration register 0	0x0020	R/W
RMT_CH2CONF1_REG	Channel 2 configuration register 1	0x0024	varies
RMT_CH3CONF0_REG	Channel 3 configuration register 0	0x0028	R/W
RMT_CH3CONF1_REG	Channel 3 configuration register 1	0x002C	varies
RMT_APB_CONF_REG	RMT APB configuration register	0x0080	R/W
RMT_REF_CNT_RST_REG	RMT clock divider reset register	0x0088	R/W
RMT_CH0_RX_CARRIER_RM_REG	Channel 0 carrier remove register	0x008C	R/W
RMT_CH1_RX_CARRIER_RM_REG	Channel 1 carrier remove register	0x0090	R/W
RMT_CH2_RX_CARRIER_RM_REG	Channel 2 carrier remove register	0x0094	R/W
RMT_CH3_RX_CARRIER_RM_REG	Channel 3 carrier remove register	0x0098	R/W
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Channel 0 duty cycle configuration register	0x0060	R/W
RMT_CH1CARRIER_DUTY_REG	Channel 1 duty cycle configuration register	0x0064	R/W
RMT_CH2CARRIER_DUTY_REG	Channel 2 duty cycle configuration register	0x0068	R/W
RMT_CH3CARRIER_DUTY_REG	Channel 3 duty cycle configuration register	0x006C	R/W
Tx event configuration registers			
RMT_CH0_TX_LIM_REG	Channel 0 Tx event configuration register	0x0070	varies
RMT_CH1_TX_LIM_REG	Channel 1 Tx event configuration register	0x0074	varies
RMT_CH2_TX_LIM_REG	Channel 2 Tx event configuration register	0x0078	varies

Name	Description	Address	Access
RMT_CH3_TX_LIM_REG	Channel 3 Tx event configuration register	0x007C	varies
RMT_TX_SIM_REG	Enable RMT simultaneous transmission	0x0084	R/W
Status registers			
RMT_CH0STATUS_REG	Channel 0 status register	0x0030	RO
RMT_CH1STATUS_REG	Channel 1 status register	0x0034	RO
RMT_CH2STATUS_REG	Channel 2 status register	0x0038	RO
RMT_CH3STATUS_REG	Channel 3 status register	0x003C	RO
RMT_CH0ADDR_REG	Channel 0 address register	0x0040	RO
RMT_CH1ADDR_REG	Channel 1 address register	0x0044	RO
RMT_CH2ADDR_REG	Channel 2 address register	0x0048	RO
RMT_CH3ADDR_REG	Channel 3 address register	0x004C	RO
Version register			
RMT_DATE_REG	Version control register	0x00FC	R/W
FIFO R/W registers			
RMT_CH0DATA_REG	Read and write data for channel 0 via APB FIFO	0x0000	RO
RMT_CH1DATA_REG	Read and write data for channel 1 via APB FIFO	0x0004	RO
RMT_CH2DATA_REG	Read and write data for channel 2 via APB FIFO	0x0008	RO
RMT_CH3DATA_REG	Read and write data for channel 3 via APB FIFO	0x000C	RO
Interrupt registers			
RMT_INT_RAW_REG	Raw interrupt status register	0x0050	RO
RMT_INT_ST_REG	Masked interrupt status register	0x0054	RO
RMT_INT_ENA_REG	Interrupt enable register	0x0058	R/W
RMT_INT_CLR_REG	Interrupt clear register	0x005C	WO

10.5 Registers

Register 10.1: RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0010+8n*)**



RMT_DIV_CNT_CH*n* This field is used to configure clock divider for channel *n*. (R/W)

RMT_IDLE_THRES_CH*n* Receiving ends when no edge is detected on input signals for continuous clock cycles larger than this register value. (R/W)

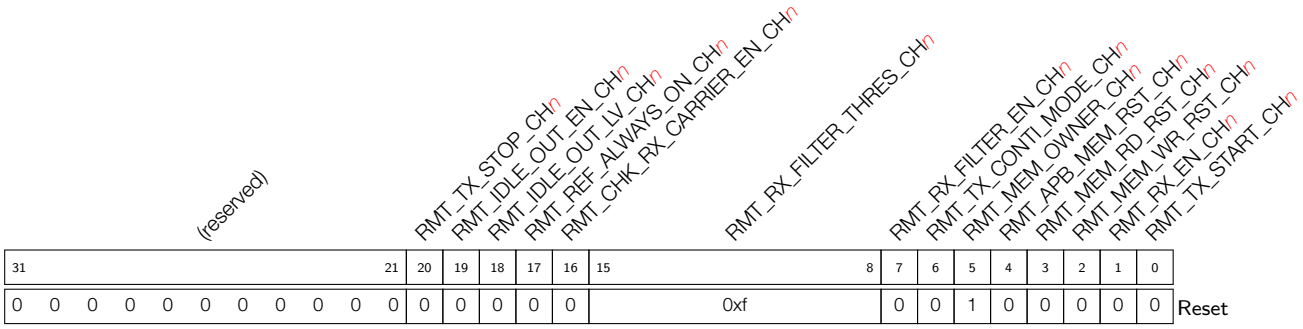
RMT_MEM_SIZE_CH*n* This field is used to configure the maximum blocks allocated to channel *n*. The valid range is from 1 ~ 4-*n*. (R/W)

RMT_CARRIER_EFF_EN_CH*n* 1: Add carrier modulation on output signals only at data sending state for channel *n*. 0: Add carrier modulation on signals at all states for channel *n*. States here include idle state(ST_IDLE), reading data from RAM (ST_RD_MEM), and sending data stored in RAM (ST_SEND). Only valid when RMT_CARRIER_EN_CH*n* is set to 1. (R/W)

RMT_CARRIER_EN_CH*n* This bit is used to enable carrier modulation for channel *n*. 1: Add carrier modulation on output signals. 0: No carrier modulation is added on output signals. (R/W)

RMT_CARRIER_OUT_LV_CH*n* This bit is used to configure the position of carrier wave for channel *n*. 1'h0: Add carrier wave on low-level output signals. 1'h1: Add carrier wave on high-level output signals. (R/W)

Register 10.2: RMT_CH*n*CONF1_REG (*n*: 0-3) (0x0014+8n*)**



RMT_TX_START_CH*n* Set this bit to start sending data on channel *n*. (R/W)

RMT_RX_EN_CH*n* Set this bit to enable receiver to receive data on channel *n*. (R/W)

RMT_MEM_WR_RST_CH*n* Set this bit to reset RAM write address accessed by the receiver for channel *n*. (WO)

RMT_MEM_RD_RST_CH*n* Set this bit to reset RAM read address accessed by the transmitter for channel *n*. (WO)

RMT_MEM_OWNER_CH*n* This bit marks the ownership of channel *n*'s RAM block. 1'h1: Receiver is using the RAM. 1'h0: Transmitter is using the RAM. (R/W)

RMT_TX_CONTI_MODE_CH*n* Set this bit to restart transmission in continuous mode from the first data in channel *n*. (R/W)

RMT_RX_FILTER_EN_CH*n* Set this bit to enable the receiver's filter for channel *n*. (R/W)

RMT_RX_FILTER_THRES_CH*n* Set this field to ignore the input pulse when its width is less than RMT_RX_FILTER_THRES_CH*n* APB clock cycles in receive mode. (R/W)

RMT_CHK_RX_CARRIER_EN_CH*n* Set this bit to enable memory loop read mode when carrier modulation is enabled for channel *n*. (R/W)

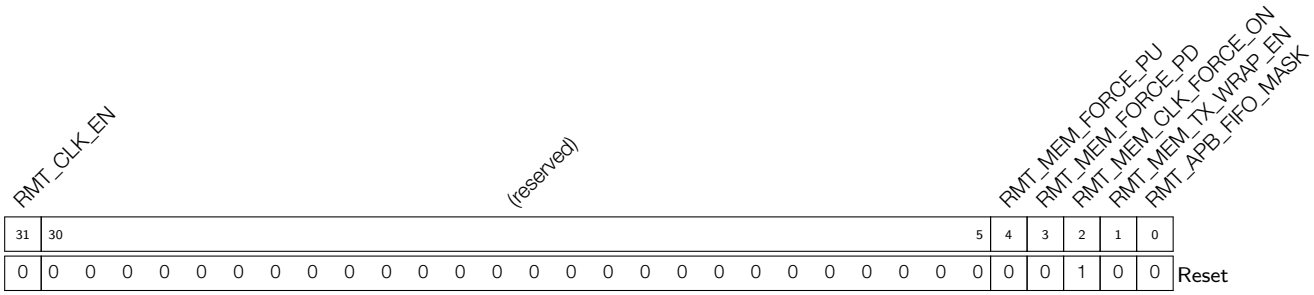
RMT_REF_ALWAYS_ON_CH*n* Set this bit to select a base clock for channel *n*. 1'h1: APB_CLK; 1'h0: REF_TICK (R/W)

RMT_IDLE_OUT_LV_CH*n* This bit configures the level of output signals in channel *n* when the transmitter is in idle state. (R/W)

RMT_IDLE_OUT_EN_CH*n* This is the output enable bit for channel *n* in idle state. (R/W)

RMT_TX_STOP_CH*n* Set this bit to stop the transmitter of channel *n* sending data out. (R/W)

Register 10.3: RMT_APB_CONF_REG (0x0080)



RMT_APB_FIFO_MASK 1'h1: Access memory directly; 1'h0: Access memory via APB FIFO. (R/W)

RMT_MEM_TX_WRAP_EN Set this bit to enable wrap mode. (R/W)

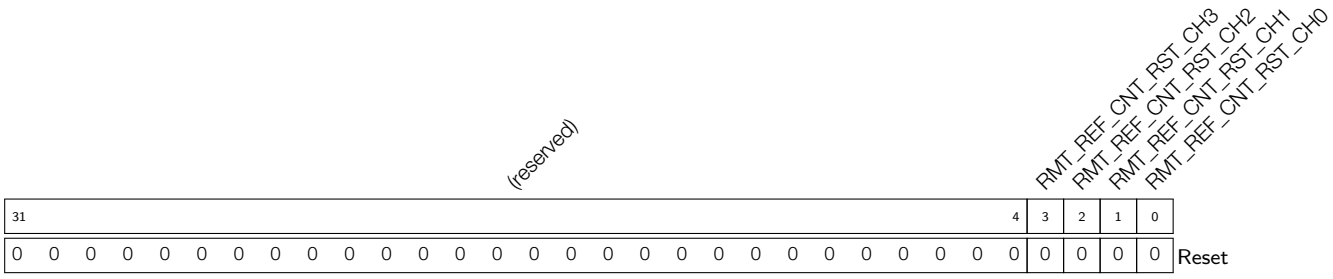
RMT_MEM_CLK_FORCE_ON Set this bit to enable the clock for RAM when RMT module starts working; disable this clock when RMT stops working, to achieve low-power scheme. (R/W)

RMT_MEM_FORCE_PD Set this bit to power down RMT memory. (R/W)

RMT_MEM_FORCE_PU 1: Disable RAM's Light-sleep power down function. 0: power down RMT RAM when RMT is in Light-sleep mode. (R/W)

RMT_CLK_EN Clock gating enable bit for RMT registers to achieve low-power scheme. 1: Power up drive clock for RMT registers. 0: Power down drive clock for RMT registers. (R/W)

Register 10.4: RMT_REF_CNT_RST_REG (0x0088)



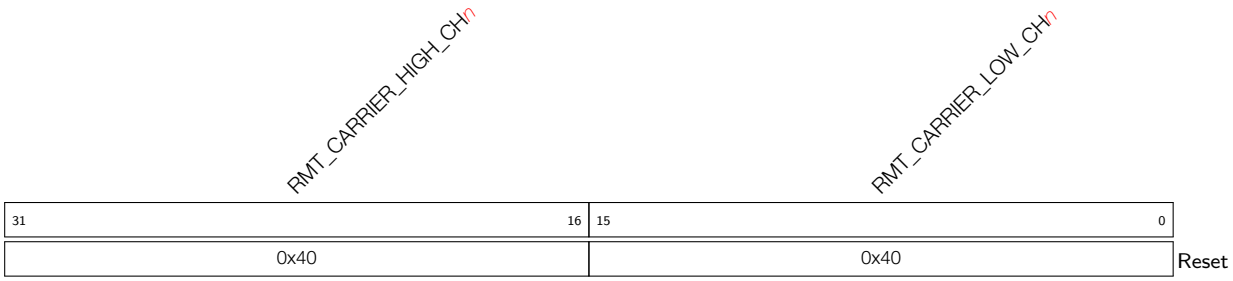
RMT_REF_CNT_RST_CH*n* This bit is used to reset the clock divider of channel *n*. (R/W)

Register 10.5: RMT_CH*n*_RX_CARRIER_RM_REG (*n*: 0-3) (0x008C+4n*)**

RMT_CARRIER_LOW_THRES_CH*n* The low level period in carrier modulation mode is (RMT_CARRIER_LOW_THRES_CH*n* + 1) clock cycles for channel *n*. (R/W)

RMT_CARRIER_HIGH_THRES_CH*n* The high level period in carrier modulation mode is (RMT_CARRIER_HIGH_THRES_CH*n* + 1) clock cycles for channel *n*. (R/W)

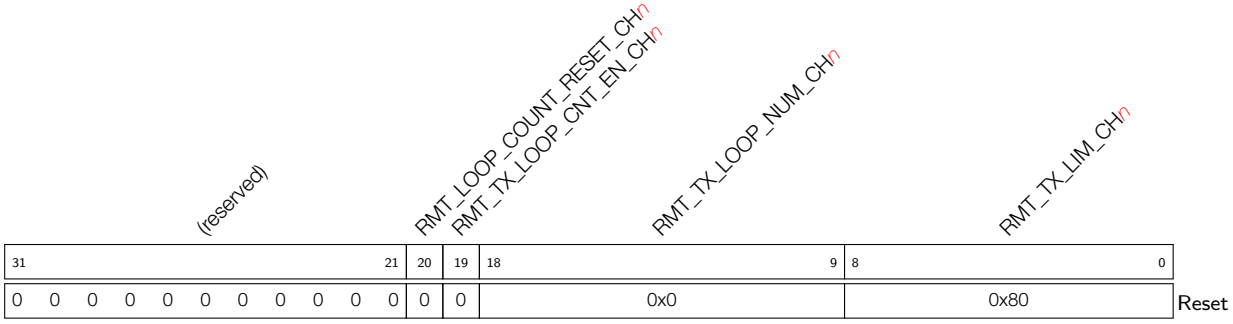
Register 10.6: RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-3) (0x0060+4n*)**



RMT_CARRIER_LOW_CH*n* This field is used to configure the clock cycles of carrier wave at low level for channel *n*. (R/W)

RMT_CARRIER_HIGH_CH*n* This field is used to configure the clock cycles of carrier wave at high level for channel *n*. (R/W)

Register 10.7: RMT_CH*n*_TX_LIM_REG (*n*: 0-3) (0x0070+4*n)



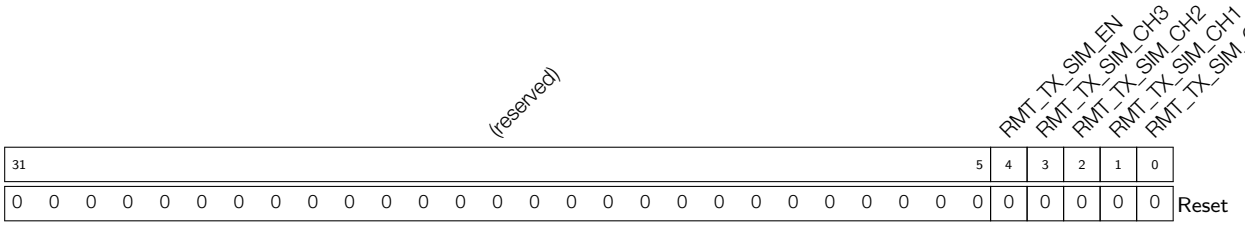
RMT_TX_LIM_CH*n* This field is used to configure the maximum entries channel *n* can send out. When RMT_MEM_SIZE_CH*n* = 1, this field can be set to any value among 0 ~ 128 (64 * 32/16 = 128); when RMT_MEM_SIZE_CH*n* > 1, this field can be set to any value among (0 ~ 128) * RMT_MEM_SIZE_CH*n*. (R/W)

RMT_TX_LOOP_NUM_CH*n* This field is used to configure the maximum loop times when continuous transmission mode is enabled. (R/W)

RMT_TX_LOOP_CNT_EN_CH*n* This bit is used to enable loop counting. (R/W)

RMT_LOOP_COUNT_RESET_CH*n* This bit is used to reset loop counting when continuous transmission mode is valid. (WO)

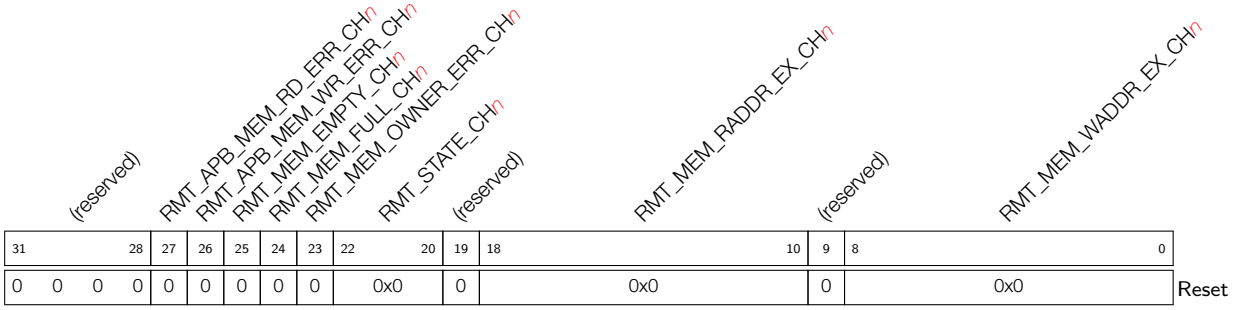
Register 10.8: RMT_TX_SIM_REG (0x0084)



RMT_TX_SIM_CH*n* Set this bit to enable channel *n* to start sending data simultaneously with other enabled channels. (R/W)

RMT_TX_SIM_EN This bit is used to enable multiple channels to start sending data simultaneously. (R/W)

Register 10.9: RMT_CH*n*STATUS_REG (*n*: 0-3) (0x0030+4n*)**



RMT_MEM_WADDR_EX_CH*n* This field records the memory address offset when receiver of channel *n* is using the RAM. (RO)

RMT_MEM_RADDR_EX_CH*n* This field records the memory address offset when transmitter of channel *n* is using the RAM. (RO)

RMT_STATE_CH*n* This field records the FSM status of channel *n*. (RO)

RMT_MEM_OWNER_ERR_CH*n* This status bit will be set when the ownership of memory block is violated. (RO)

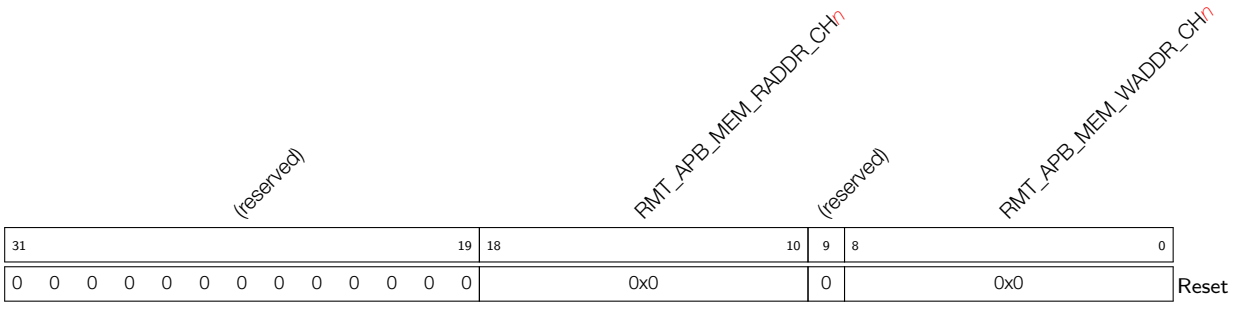
RMT_MEM_FULL_CH*n* This status bit will be set if the receiver receives more data than the memory allows. (RO)

RMT_MEM_EMPTY_CH*n* This status bit will be set when the data to be sent is more than the memory allows and the wrap mode is disabled. (RO)

RMT_APB_MEM_WR_ERR_CH*n* This status bit will be set if the offset address is out of memory size when channel *n* writes RAM via APB bus. (RO)

RMT_APB_MEM_RD_ERR_CH*n* This status bit will be set if the offset address is out of memory size when channel *n* reads RAM via APB bus. (RO)

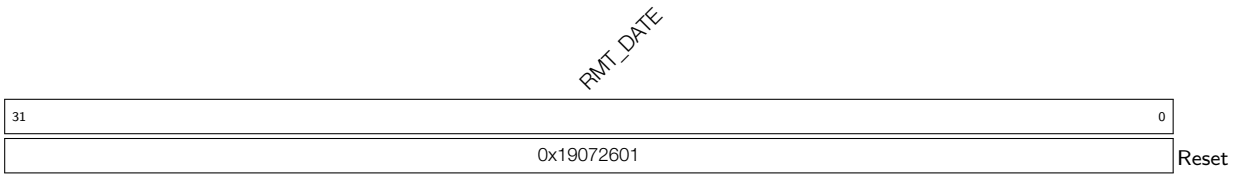
Register 10.10: RMT_CH*n*ADDR_REG (*n*: 0-3) (0x0040+4n*)**



RMT_APB_MEM_WADDR_CH*n* This field records the memory address offset when channel *n* writes RAM via APB bus. (RO)

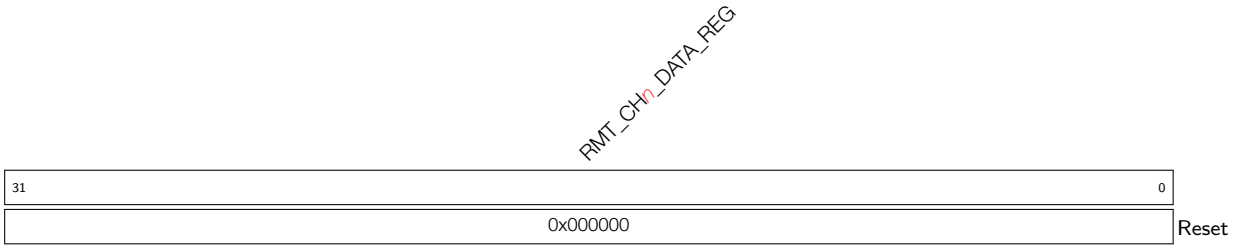
RMT_APB_MEM_RADDR_CH*n* This field records the memory address offset when channel *n* reads RAM via APB bus. (RO)

Register 10.11: RMT_DATE_REG (0x00FC)



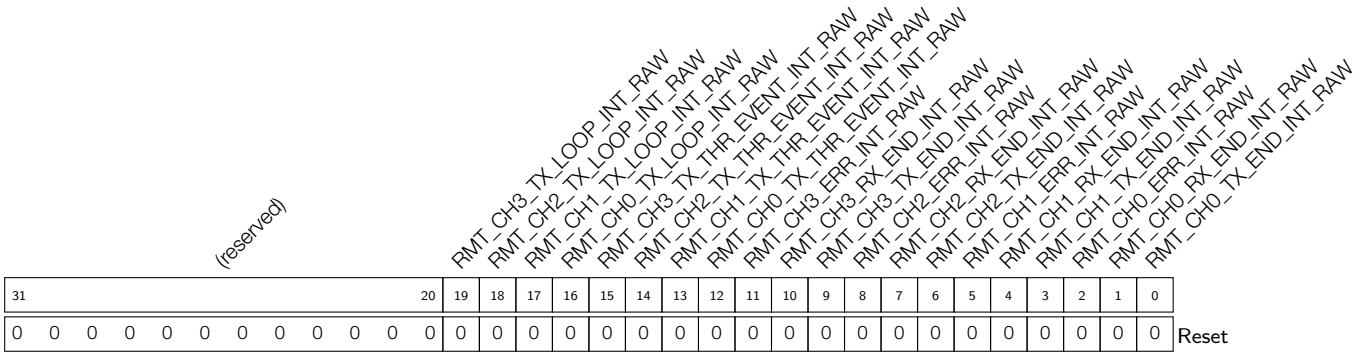
RMT_DATE Version control register. (R/W)

Register 10.12: RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+4n*)**



RMT_CH*n*_DATA_REG This register is used to read and write data for channel *n* via APB FIFO. (RO)

Register 10.13: RMT_INT_RAW_REG (0x0050)



RMT_CH*n*_TX_END_INT_RAW The interrupt raw bit for channel *n*. Triggered when transmitting ends. (RO)

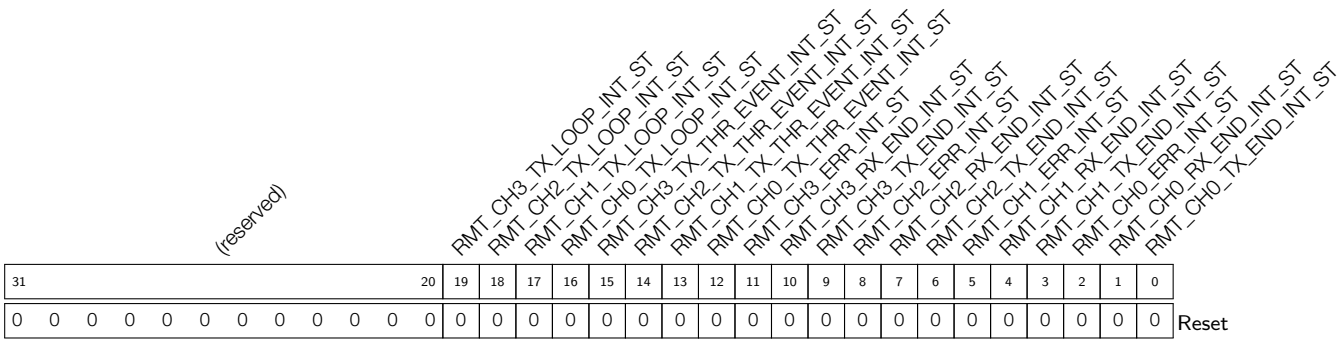
RMT_CH*n*_RX_END_INT_RAW The interrupt raw bit for channel *n*. Triggered when receiving ends. (RO)

RMT_CH*n*_ERR_INT_RAW The interrupt raw bit for channel *n*. Triggered when error occurs. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_RAW The interrupt raw bit for channel *n*. Triggered when transmitter sends more data than configured value. (RO)

RMT_CH*n*_TX_LOOP_INT_RAW The interrupt raw bit for channel *n*. Triggered when loop counting reaches the configured threshold value. (RO)

Register 10.14: RMT_INT_ST_REG (0x0054)



RMT_CH*n*_TX_END_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_END_INT. (RO)

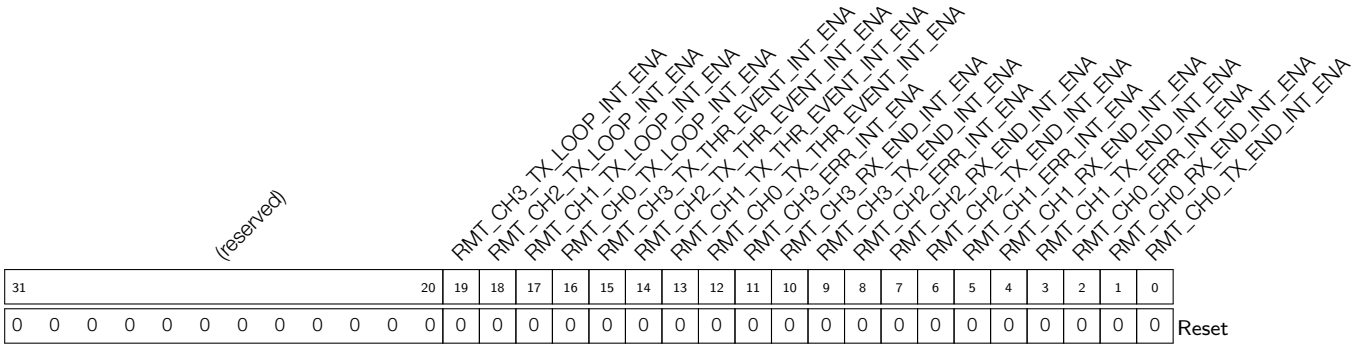
RMT_CH*n*_RX_END_INT_ST The masked interrupt status bit for RMT_CH*n*_RX_END_INT. (RO)

RMT_CH*n*_ERR_INT_ST The masked interrupt status bit for RMT_CH*n*_ERR_INT. (RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_THR_EVENT_INT. (RO)

RMT_CH*n*_TX_LOOP_INT_ST The masked interrupt status bit for RMT_CH*n*_TX_LOOP_INT. (RO)

Register 10.15: RMT_INT_ENA_REG (0x0058)



RMT_CH*n*_TX_END_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_END_INT. (R/W)

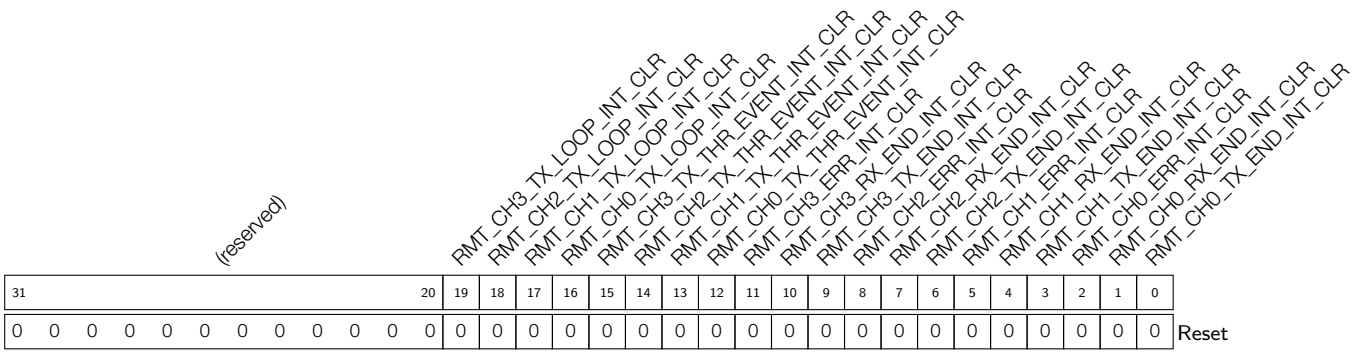
RMT_CH*n*_RX_END_INT_ENA Interrupt enable bit for RMT_CH*n*_RX_END_INT. (R/W)

RMT_CH*n*_ERR_INT_ENA Interrupt enable bit for RMT_CH*n*_ERR_INT. (R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_THR_EVENT_INT. (R/W)

RMT_CH*n*_TX_LOOP_INT_ENA Interrupt enable bit for RMT_CH*n*_TX_LOOP_INT. (R/W)

Register 10.16: RMT_INT_CLR_REG (0x005C)



RMT_CH*n*_TX_END_INT_CLR Set this bit to clear RMT_CH*n*_TX_END_INT interrupt. (WO)

RMT_CH*n*_RX_END_INT_CLR Set this bit to clear RMT_CH*n*_RX_END_INT interrupt. (WO)

RMT_CH*n*_ERR_INT_CLR Set this bit to clear RMT_CH*n*_ERR_INT interrupt. (WO)

RMT_CH*n*_TX_THR_EVENT_INT_CLR Set this bit to clear RMT_CH*n*_TX_THR_EVENT_INT interrupt. (WO)

RMT_CH*n*_TX_LOOP_INT_CLR Set this bit to clear RMT_CH*n*_TX_LOOP_INT interrupt. (WO)

11. Pulse Count Controller

The pulse count controller (PCNT) is designed to count input pulses and generate interrupts. It can increment or decrement a pulse counter value by keeping track of rising (positive) or falling (negative) edges of the input pulse signal. The PCNT has four independent pulse counters, called units which have their groups of registers. In this chapter, *n* denotes the number of a unit from 0 ~ 3.

Each unit includes two channels (ch0 and ch1) which can independently increment or decrement its pulse counter value. The remainder of the chapter will mostly focus on channel 0 (ch0) as the functionality of the two channels is identical.

As shown in Figure 11-1, each channel has two input signals:

1. One control signal (e.g. ctrl_ch0_u*n*, the control signal for ch0 of unit n)
2. One input pulse signal (e.g. sig_ch0_u*n*, the input pulse signal for ch0 unit n)

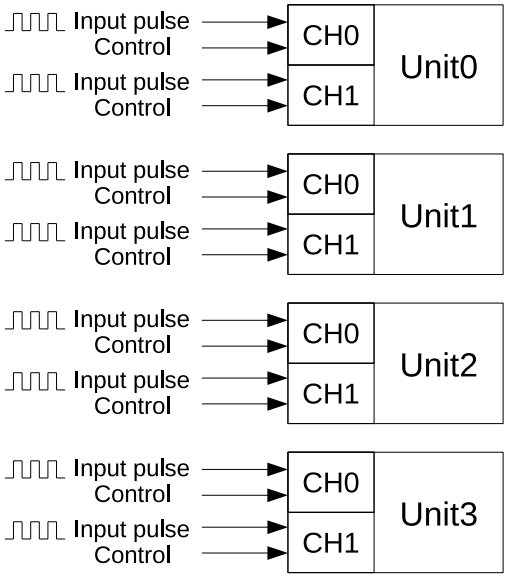


Figure 11-1. PCNT Block Diagram

11.1 Features

A PCNT has the following features:

- Four independent pulse counters (units)
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g. sig_ch0_u*n*) with their corresponding control signals (e.g. ctrl_ch0_u*n*)
- Independent filtering of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit
- Each channel has the following parameters:
    1. Selection between counting on positive or negative edges of the input pulse signal

2. Configuration to Increment, Decrement, or Disable counter mode for control signal's high and low states

11.2 Functional Description

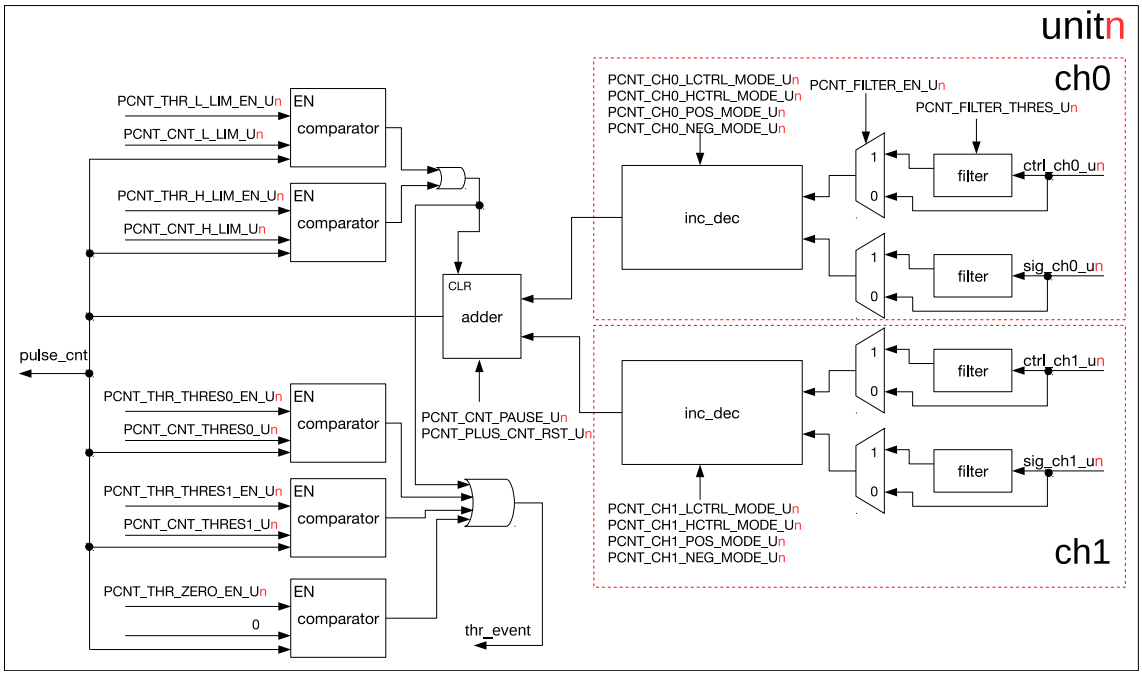


Figure 11-2. PCNT Unit Architecture

Figure 11-2 shows PCNT's architecture. As stated above, ctrl_ch0_u*n* is the control signal for ch0 of unit n. Its high and low states can be assigned different counter modes and used for pulse counting of the channel's input pulse signal sig_ch0_u*n* on negative or positive edges. The available counter modes are as follows:

- Increment mode: When a channel detects an active edge of sig_ch0_u*n* (the one configured for counting), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if PCNT_CNT_PAUSE_U*n* is set before pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.
- Decrement mode: When a channel detects an active edge of sig_ch0_u*n* (the one configured for counting), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If the channel's counter mode is changed or if PCNT_CNT_PAUSE_U*n* is set before pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, then pulse_cnt freezes and its counter mode changes.
- Disable mode: Counting is disabled, and the counter value pulse_cnt freezes.

Table 44 to Table 47 provide information on how to configure the counter mode for channel 0.

Each unit has one filter for all its control and input pulse signals. A filter can be enabled with the bit PCNT_FILTER_EN_U*n*. The filter monitors the signals and ignores all the noise, i.e. the glitches with pulse widths shorter than
PCNT_FILTER_THRES_U*n* APB clock cycles in length.

As previously mentioned, each unit has two channels which process different input pulse signals and increase or decrease values via their respective inc_dec modules, then the two channels send these values to the adder

Table 44: Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 45: Counter Mode. Positive Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 46: Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in Low State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

Table 47: Counter Mode. Negative Edge of Input Pulse Signal. Control Signal in High State

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	Counter Mode
1	0	Increment
	1	Decrement
	Others	Disable
2	0	Decrement
	1	Increment
	Others	Disable
Others	N/A	Disable

module that is 16-bit wide with a sign bit. This adder can be suspended by setting PCNT_CNT_PAUSE_U*n*, and cleared by setting PCNT_PULSE_CNT_RST_U*n*.

The PCNT has five watchpoints that share one interrupt. The interrupt can be enabled or disabled by interrupt enable signals of each individual watchpoint.

- Maximum count value: When pulse_cnt reaches PCNT_CNT_H_LIM_U*n*, an interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_U*n* is high.
- Minimum count value: When pulse_cnt reaches PCNT_CNT_L_LIM_U*n*, an interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_U*n* is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_THRES0_U*n* or PCNT_CNT_THRES1_U*n*, an interrupt is triggered and either PCNT_CNT_THR_THRES0_LAT_U*n* or PCNT_CNT_THR_THRES1_LAT_U*n* is high respectively.
- Zero: When pulse_cnt is 0, an interrupt is triggered and PCNT_CNT_THR_ZERO_LAT_U*n* is valid.

11.3 Applications

In each unit, channel 0 and channel 1 can be configured to work independently or together. The three subsections below provide details of channel 0 incrementing independently, channel 0 decrementing independently, and channel 0 and channel 1 incrementing together. For other working modes not elaborated in this section (e.g. channel 1 incrementing/decremeting independently, or one channel incrementing while the other decrementing), reference can be made to these three subsections.

11.3.1 Channel 0 Incrementing Independently

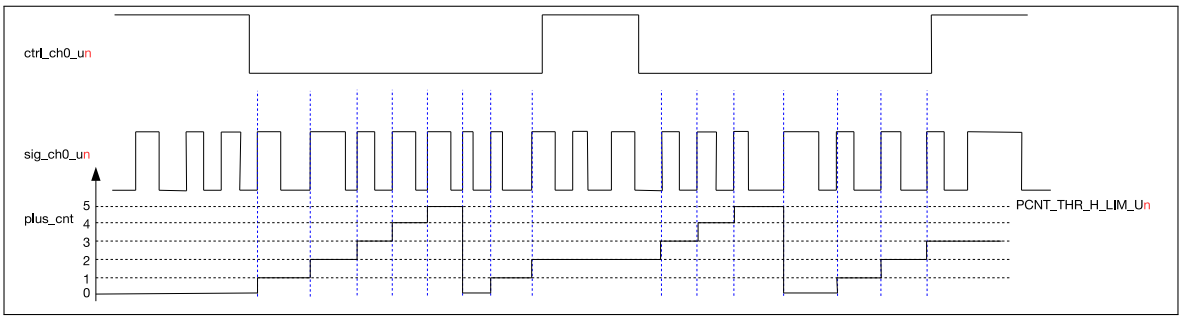


Figure 11-3. Channel 0 Up Counting Diagram

Figure 11-3 illustrates how channel 0 is configured to increment independently on the positive edge of sig_ch0_u*n* while channel 1 is disabled (see subsection 11.2 for how to disable channel 1). The configuration of channel 0 is shown below.

- PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
- PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
- PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch0_u*n*.
- PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch0_u*n*.
- PCNT_CNT_H_LIM_U*n*=5: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

11.3.2 Channel 0 Decrementing Independently

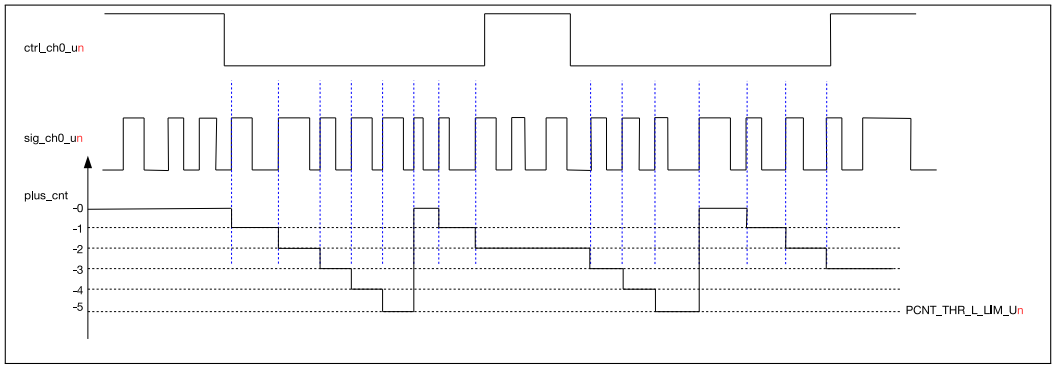


Figure 11-4. Channel 0 Down Counting Diagram

Figure 11-4 illustrates how channel 0 is configured to decrement independently on the positive edge of sig_ch0_u*n* while channel 1 is disabled. The configuration of channel 0 in this case differs from that in Figure 11-3 in the following aspects:

- PCNT_CH0_POS_MODE_U*n*=2: the counter decrements on the positive edge of sig_ch0_u*n*.
- PCNT_CNT_L_LIM_U*n*=-5: when pulse_cnt counts down to PCNT_CNT_L_LIM_U*n*, it is cleared.

11.3.3 Channel 0 and Channel 1 Incrementing Together

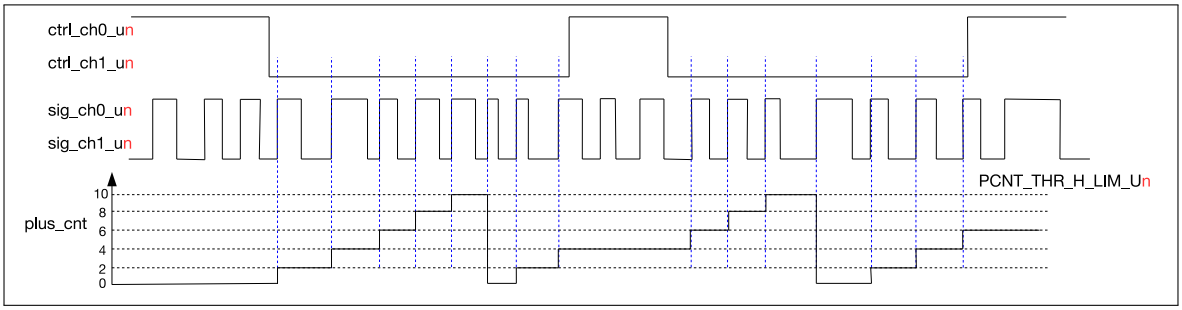


Figure 11-5. Two Channels Up Counting Diagram

Figure 11-5 illustrates how channel 0 and channel 1 are configured to increment on the positive edge of sig_ch0_u*n* and sig_ch1_u*n* respectively at the same time. It can be seen in Figure 11-5 that control signal ctrl_ch0_u*n* and ctrl_ch1_u*n* have the same waveform, so as input pulse signal sig_ch0_u*n* and sig_ch1_u*n*. The configuration procedure is shown below.

- For channel 0:
  - PCNT_CH0_LCTRL_MODE_U*n*=0: When ctrl_ch0_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH0_HCTRL_MODE_U*n*=2: When ctrl_ch0_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH0_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch0_u*n*.
  - PCNT_CH0_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch0_u*n*.
- For channel 1:

  - PCNT_CH1_LCTRL_MODE_U*n*=0: When ctrl_ch1_u*n* is low, the counter mode specified for the low state turns on, in this case it is Increment mode.
  - PCNT_CH1_HCTRL_MODE_U*n*=2: When ctrl_ch1_u*n* is high, the counter mode specified for the low state turns on, in this case it is Disable mode.
  - PCNT_CH1_POS_MODE_U*n*=1: The counter increments on the positive edge of sig_ch1_u*n*.
  - PCNT_CH1_NEG_MODE_U*n*=0: The counter idles on the negative edge of sig_ch1_u*n*.
- PCNT_CNT_H_LIM_U*n*=10: When pulse_cnt counts up to PCNT_CNT_H_LIM_U*n*, it is cleared.

11.4 Base Address

Users can access the PCNT registers with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 48: PCNT Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F417000
PeriBUS2	0x60017000

11.5 Register Summary

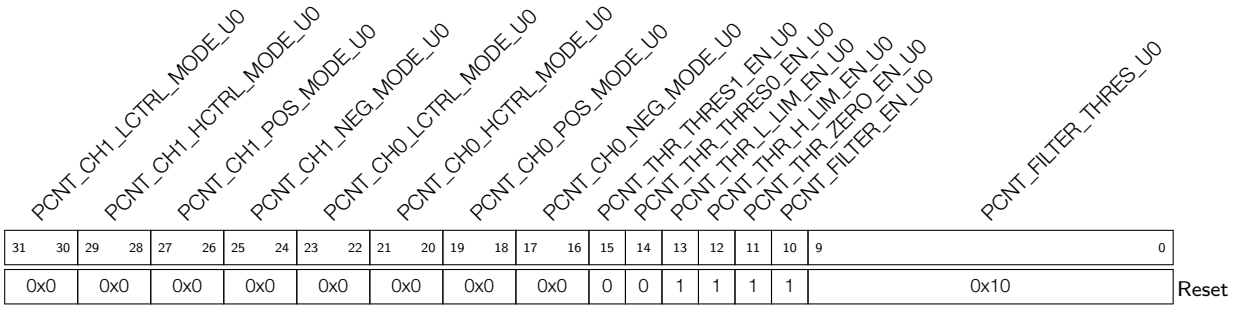
The addresses in the following table are relative to the PCNT base addresses provided in Section 11.4.

Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT unit0 status register	0x0050	RO

Name	Description	Address	Access
PCNT_U1_STATUS_REG	PNCT unit1 status register	0x0054	RO
PCNT_U2_STATUS_REG	PNCT unit2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT unit3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

11.6 Registers

Register 11.1: PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+12n*)**



PCNT_FILTER_THRES_U*n* This sets the maximum threshold, in APB_CLK cycles, for the filter. Any pulses with width less than this will be ignored when the filter is enabled. (R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_CH0_NEG_MODE_U*n* This register sets the behavior when the signal input of channel 0 detects a negative edge. 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter (R/W)

PCNT_CH0_POS_MODE_U*n* This register sets the behavior when the signal input of channel 0 detects a positive edge. 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter (R/W)

PCNT_CH0_HCTRL_MODE_U*n* This register configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high. 0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

PCNT_CH0_LCTRL_MODE_U*n* This register configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low. 0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

PCNT_CH1_NEG_MODE_U*n* This register sets the behavior when the signal input of channel 1 detects a negative edge. 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter (R/W)

PCNT_CH1_POS_MODE_U*n* This register sets the behavior when the signal input of channel 1 detects a positive edge. 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter (R/W)

Continued on the next page...

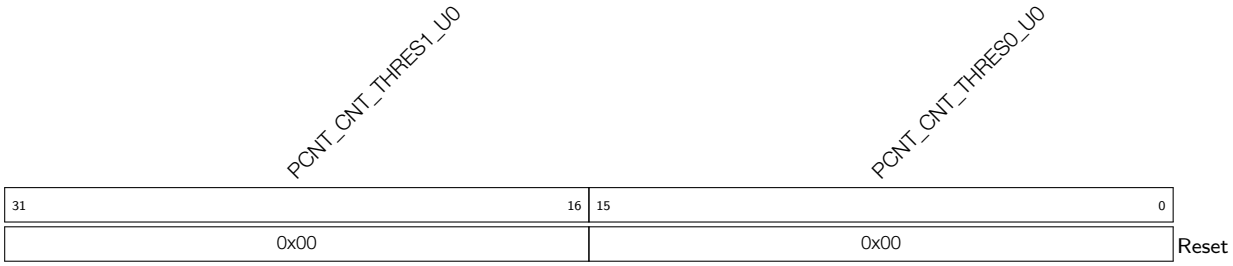
Register 11.1: PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+12n*)**

Continued from the previous page ...

PCNT_CH1_HCTRL_MODE_U*n* This register configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high. 0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

PCNT_CH1_LCTRL_MODE_U*n* This register configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is low. 0: No modification; 1: Invert behavior (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification (R/W)

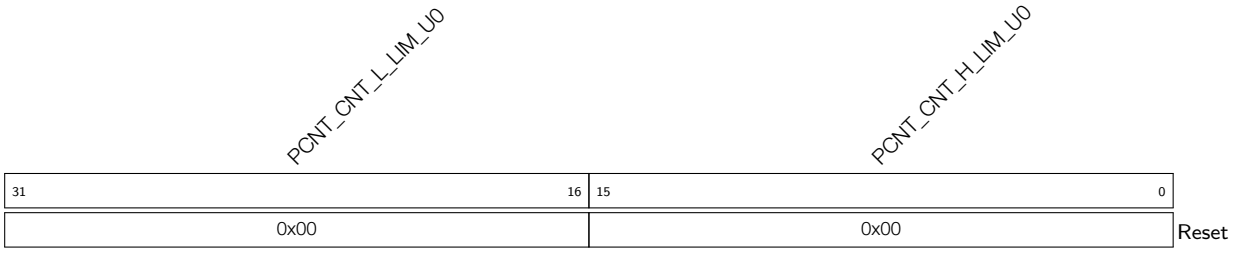
Register 11.2: PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+12n*)**



PCNT_CNT_THRES0_U*n* This register is used to configure the thres0 value for unit *n*. (R/W)

PCNT_CNT_THRES1_U*n* This register is used to configure the thres1 value for unit *n*. (R/W)

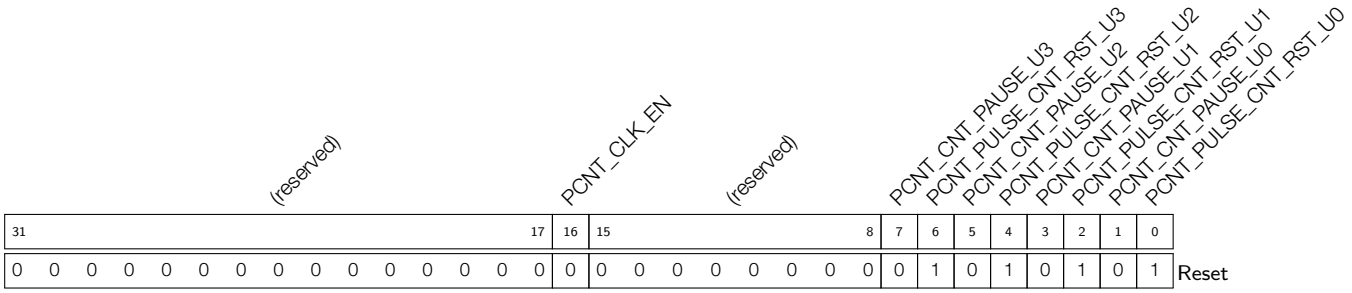
Register 11.3: PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+12n*)**



PCNT_CNT_H_LIM_U*n* This register is used to configure the thr_h_lim value for unit *n*. (R/W)

PCNT_CNT_L_LIM_U*n* This register is used to configure the thr_l_lim value for unit *n*. (R/W)

Register 11.4: PCNT_CTRL_REG (0x0060)

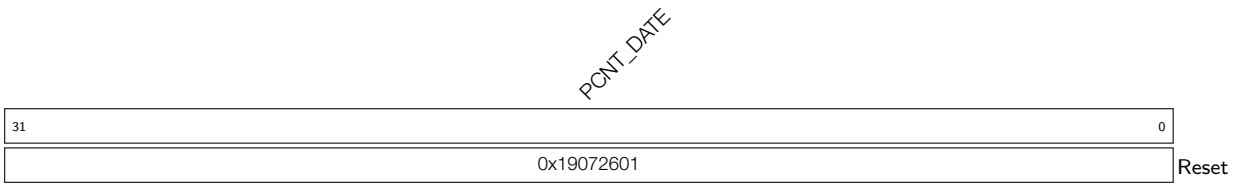


PCNT_PULSE_CNT_RST_U*n* Set this bit to clear unit *n*'s counter. (R/W)

PCNT_CNT_PAUSE_U*n* Set this bit to freeze unit *n*'s counter. (R/W)

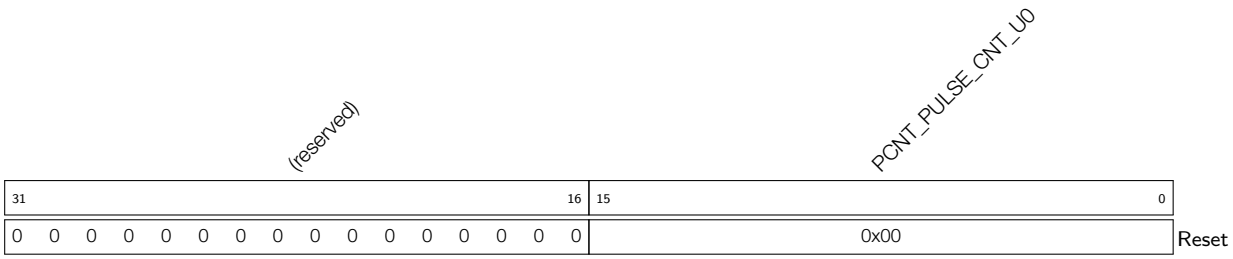
PCNT_CLK_EN The registers clock gate enable signal of PCNT module. 1: the registers can be read and written by application. 0: the registers can not be read or written by application (R/W)

Register 11.5: PCNT_DATE_REG (0x00FC)



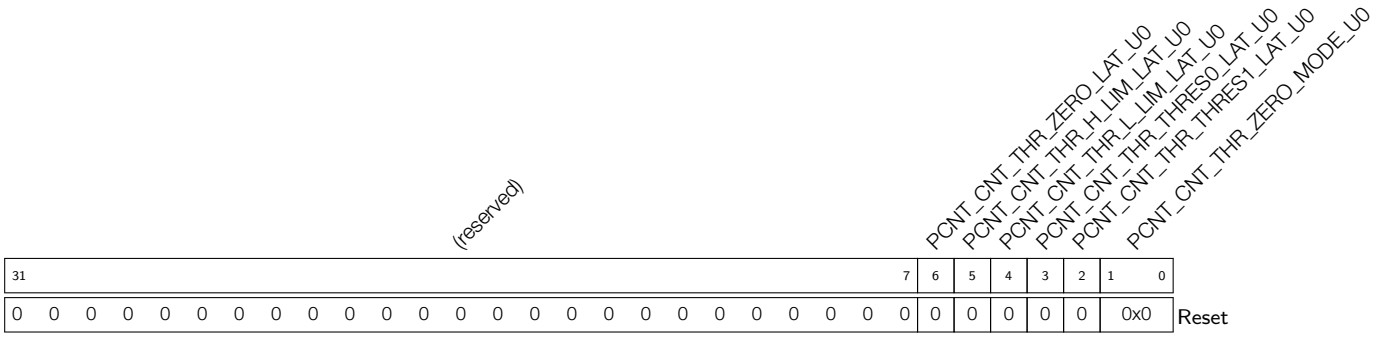
PCNT_DATE This is the PCNT version control register. (R/W)

Register 11.6: PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+4n*)**



PCNT_PULSE_CNT_U*n* This register stores the current pulse count value for unit *n*. (RO)

Register 11.7: PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+4n*)**



PCNT_CNT_THR_ZERO_MODE_U*n* The pulse counter status of PCNT_U*n* corresponding to 0. 0: pulse counter decreases from positive to 0. 1: pulse counter increases from negative to 0. 2: pulse counter is negative. 3: pulse counter is positive. (RO)

PCNT_CNT_THR_THRES1_LAT_U*n* The latched value of thres1 event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thres1 and thres1 event is valid. 0: others (RO)

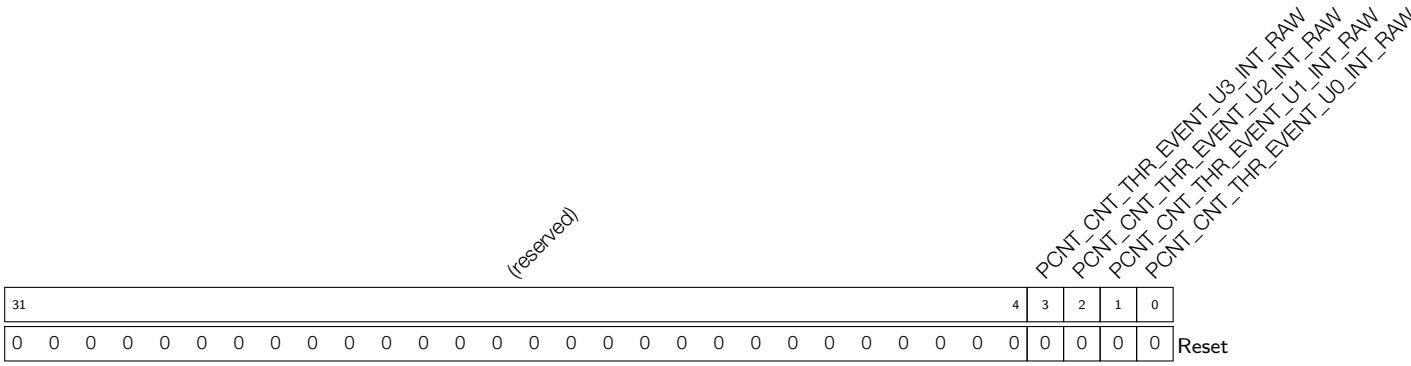
PCNT_CNT_THR_THRES0_LAT_U*n* The latched value of thres0 event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thres0 and thres0 event is valid. 0: others (RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* The latched value of low limit event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thr_l_lim and low limit event is valid. 0: others (RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* The latched value of high limit event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thr_h_lim and high limit event is valid. 0: others (RO)

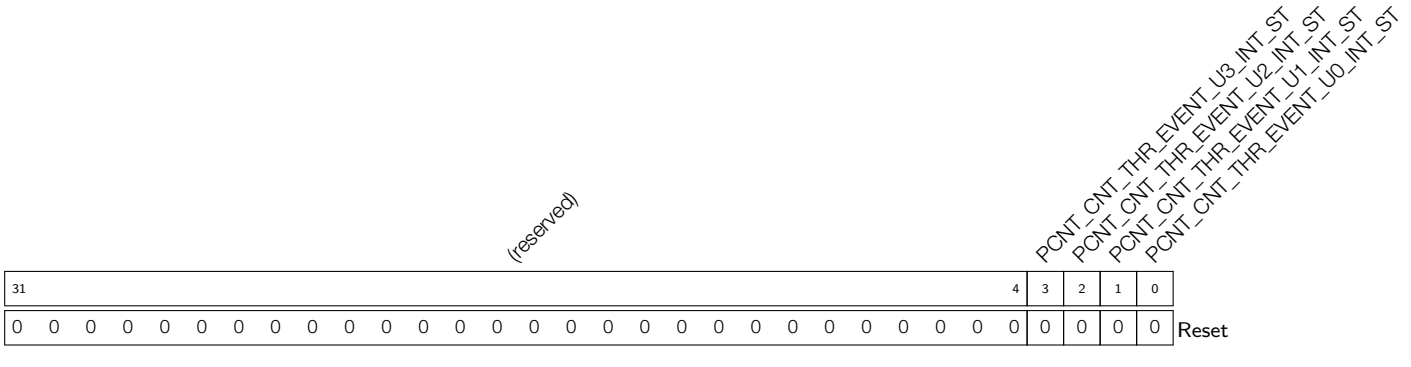
PCNT_CNT_THR_ZERO_LAT_U*n* The latched value of zero threshold event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to 0 and zero threshold event is valid. 0: others (RO)

Register 11.8: PCNT_INT_RAW_REG (0x0040)



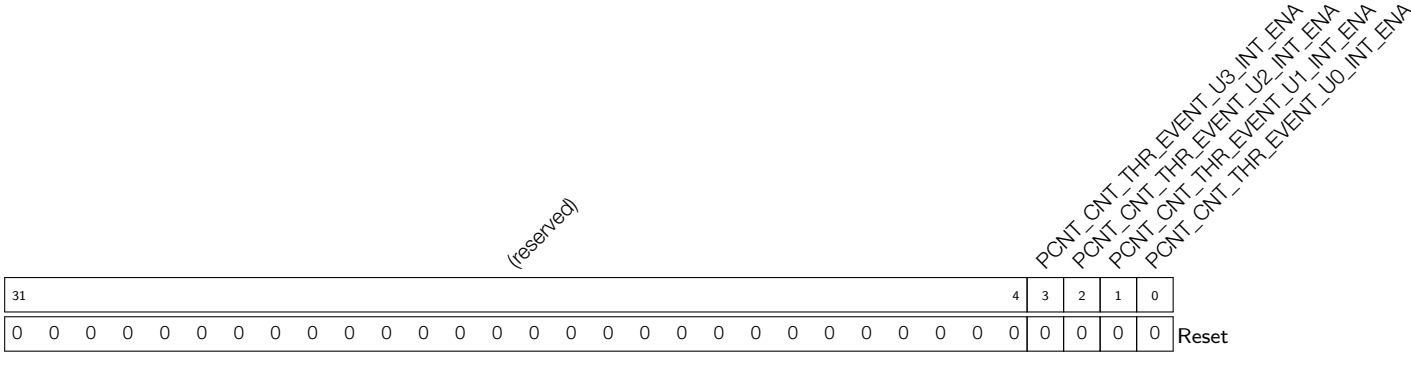
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 11.9: PCNT_INT_ST_REG (0x0044)



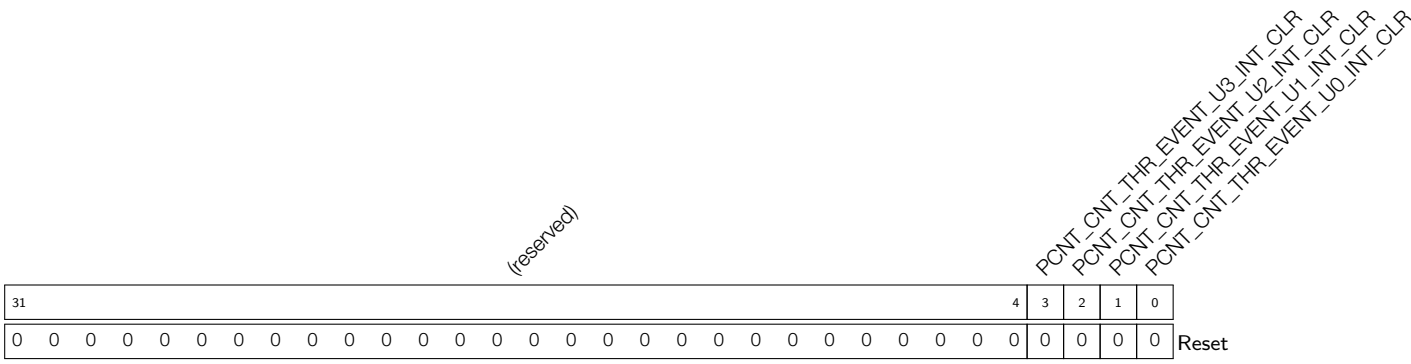
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 11.10: PCNT_INT_ENA_REG (0x0048)



PCNT_CNT_THR_EVENT_U*n*_INT_ENA The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 11.11: PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Set this bit to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

12. 64-bit Timers

12.1 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 12-1, the ESP32-S2 chip contains two timer groups, namely timer group 0 and timer group 1. Each timer group consists of two general purpose timers referred to as T*x* (where *x* is 0 or 1) and one Main System Watchdog Timer. All general purpose timers are based on 16-bit prescalers and 64-bit auto-reload-capable up/down counters.

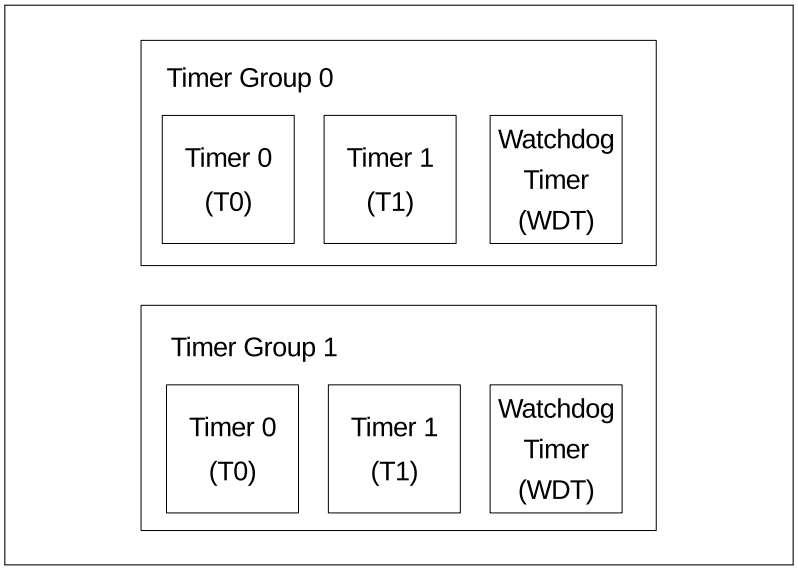


Figure 12-1. Timer Units within Groups

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 13: *Watchdog Timers*. Therefore, the term 'timers' within this chapter refers to the general purpose timers.

The timers' features are summarized as follows:

- A 16-bit clock prescaler, from 1 to 65536
- A 64-bit time-base counter programmable to be incrementing or decrementing
- Able to read real-time value of the time-base counter
- Halting and resuming the time-base counter
- Programmable alarm generation
- Timer value reload (Auto-reload at alarm or software-controlled instant reload)
- Level and edge interrupt generation

12.2 Functional Description

12.2.1 16-bit Prescaler and Clock Selection

Each timer can select between the APB clock (APB_CLK) or external clock (XTAL_CLK) as its clock source by setting the TIMG_T*x*_USE_XTAL field of the TIMG_T*x*CONFIG_REG register. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB_CLK) used by the time-base counter. The 16-bit prescaler is configured by the TIMG_T*x*_DIVIDER field and can take any value from 1 to 65536. Note that programming a value of 0 in TIMG_T*x*_DIVIDER will result in the divisor being 65536.

The timer must be disabled (i.e. TIMG_T*x*_EN should be cleared) before modifying the 16-bit prescaler. Modifying the 16-bit prescaler whilst the timer is enabled can lead to unpredictable results.

12.2.2 64-bit Time-based Counter

The 64-bit time-base counters are based on TB_CLK and can the configured to increment or decrement via the TIMG_T*x*_INCREASE field. The time-base counter can be enabled/disabled by setting/clearing the TIMG_T*x*_EN field. Whilst enabled, the time-base counter will increment/decrement on each cycle of TB_CLK. When disabled, the time-base counter is essentially frozen. Note that the TIMG_T*x*_INCREASE field can be changed whilst TIMG_T*x*_EN is set and will cause the time-base counter to change direction instantly.

To read the 64-bit current timer value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_T*x*UPDATE_REG, the current value of the 64-bit timer is instantly latched into the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers containing the lower and upper 32-bits respectively. TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_T*x*UPDATE_REG is written to again.

12.2.3 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 12.2.4). The 64-bit alarm value is configured in the TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG representing the lower and upper 32-bits of the alarm value respectively. However, the configured alarm value is ineffective until the alarm is enabled by setting the TIMG_T*x*_ALARM_EN field. In order to simply the scenario where the alarm is enabled 'too late' (i.e. the timer value has already passed the alarm value when the alarm is enabled), the alarm value will also trigger immediately if the current timer value is larger/smaller than the alarm value for an up-counting/down-counting timer.

When an alarm occurs, the TIMG_T*x*_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_T*x*_ALARM_EN is set.

12.2.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI registers that correspond to the lower and upper 32-bits of the timer's new value respectively. However, writing a reload value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to TIMG_T*x*LOAD_REG causing the timer's current value to be instantly reloaded. If TIMG_T*x*_EN is set, the timer will continue incrementing/decrementing from the new value. If TIMG_T*x*_EN is cleared, the timer will remain frozen at the new value until counting is

re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs thus allowing the timer to continue incrementing/decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the TIMG_T*x*_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment/decrement past the alarm value after an alarm.

12.2.5 Interrupts

Each timer has its own pair of interrupt lines (for edge and level interrupts) that can be routed to the CPU. Thus, there are a total of six interrupt lines per timer group and they are named as follows:

- TIMG_WDT_LEVEL_INT: Level interrupt line for the watchdog timer in the group, generated when a watchdog timer interrupt stage times out.
- TIMG_WDT_EDGE_INT: Edge interrupt line for the watchdog timer in the group, generated when a watchdog timer interrupt stage times out.
- TIMG_T*x*_LEVEL_INT: Level interrupt for one of the general purpose timers, generated when an alarm event happens.
- TIMG_T*x*_EDGE_INT: Edge interrupt for one of the general purpose timer, generated when an alarm event happens.

Interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupt lines will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. Conversely, edge interrupts will generate a short pulse after an alarm (or stage timeout) occurs. To enable a timer's level or edge interrupt lines, the TIMG_T*x*_LEVEL_INT_EN or TIMG_T*x*_EDGE_INT_EN bits should be set respectively.

The interrupts of each timer group are governed by a set of registers. Each timer within the group will have a corresponding bit in each of these registers:

- TIMG_T*x*_INT_RAW : An alarm event sets it to 1. The bit will remain set until writing to the timer's corresponding bit in TIMG_T*x*_INT_CLR.
- TIMG_WDT_INT_RAW : A stage time out will set the timer's bit to 1. The bit will remain set until writing to the timer's corresponding bit in TIMG_WDT_INT_CLR.
- TIMG_T*x*_INT_ST : Reflects the status of each timer's interrupt and is generated by masking the bits of TIMG_T*x*_INT_RAW with TIMG_T*x*_INT_ENA. For level interrupts, these bits reflect the level on the watchdog timer's level interrupt line.
- 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. For level interrupts, these bits reflect the level on the watchdog timer's level interrupt line.
- TIMG_T*x*_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_T*x*_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The timer's corresponding bit in TIMG_T*x*_INT_RAW and TIMG_T*x*_INT_ST will be cleared as a result. Note that a timer's interrupt must be cleared before the next interrupt occurs when using level interrupts.
- 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 wachdog timer's interrupt must be cleared before the next interrupt occurs when using level interrupts.

12.3 Configuration and Usage

12.3.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting TIMG_T*x*_USE_XTAL field.
   - Configure the 16-bit prescaler by setting TIMG_T*x*_DIVIDER.
   - Configure the timer direction by setting/clearing TIMG_T*x*_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI, then reloading it into the timer by writing any value to TIMG_T*x*LOAD_REG.
2. Start the timer by setting TIMG_T*x*_EN.
3. Getting the timer's current value.
   - Write any value to TIMG_T*x*UPDATE_REG to latch the timer's current value.
   - Read the latched timer value from TIMG_T*x*LO_REG and TIMG_T*x*HI_REG.

12.3.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 12.3.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG.
   - Enable interrupt by setting TIMG_T*x*_LEVEL_INT_EN or TIMG_T*x*_EDGE_INT_EN for level or edge interrupts respectively.
3. Disable auto reload by clearing TIMG_T*x*_AUTORELOAD.
4. Start the timer by setting TIMG_T*x*_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

12.3.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 of Section 12.3.1.
2. Configure the alarm following step 2 of Section 12.3.2.
3. Enable auto reload by setting TIMG_T*x*_AUTORELOAD and setting the reload value in TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.
4. Start the timer by setting TIMG_T*x*_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e. different alarm interval per iteration), then TIMG_T*x*ALARMLO_REG, TIMG_T*x*ALARMHI_REG, TIMG_T*x*_LOAD_LO, and TIMG_T*x*_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_T*x*_ALARM_EN.
6. Stopping the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

12.4 Base Address

Users can access the 64-bit Timer with four base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 50: 64-bit Timers Base Address

Name	Bus to Access Peripheral	Base Address
TIMG0	PeriBUS1	0x3F41F000
	PeriBUS2	0x6001F000
TIMG1	PeriBUS1	0x3F420000
	PeriBUS2	0x60020000

12.5 Register Summary

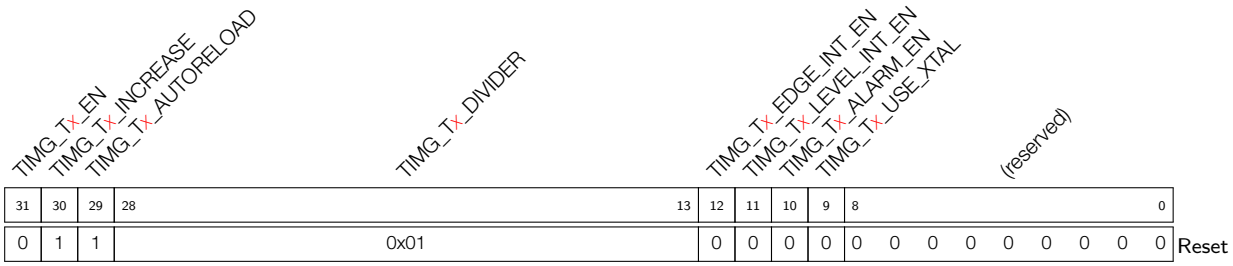
The addresses in the following table are relative to the 64-bit Timer base addresses provided in Section 12.4.

Name	Description	Address	Access
Configuration and Control Register for Timer 0			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	R/W
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 32 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMG_T0LO_REG and TIMG_T0LO_REG	0x000C	R/W
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W

Name	Description	Address	Access
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x0014	R/W
TIMG_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_T0LOADHI_REG	Timer 0 reload value, high 32 bits	0x001C	R/W
TIMG_T0LOAD_REG	Write to reload timer from TIMG_T0LOADLO_REG and TIMG_T0LOADHI_REG	0x0020	WO
Configuration and Control Register for Timer 1			
TIMG_T1CONFIG_REG	Timer 1 configuration register	0x0024	R/W
TIMG_T1LO_REG	Timer 1 current value, low 32 bits	0x0028	RO
TIMG_T1HI_REG	Timer 1 current value, high 32 bits	0x002C	RO
TIMG_T1UPDATE_REG	Write to copy current timer value to TIMG_T1LO_REG and TIMG_T1LO_REG	0x0030	R/W
TIMG_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x0034	R/W
TIMG_T1ALARMHI_REG	Timer 1 alarm value, high bits	0x0038	R/W
TIMG_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x003C	R/W
TIMG_T1LOADHI_REG	Timer 1 reload value, high 32 bits	0x0040	R/W
TIMG_T1LOAD_REG	Write to reload timer from TIMG_T0LOADLO_REG and TIMG_T0LOADHI_REG	0x0044	WO
Configuration and Control Register for WDT			
TIMG_WDTCONFIG0_REG	Watchdog timer configuration register	0x0048	R/W
TIMG_WDTCONFIG1_REG	Watchdog timer prescaler register	0x004C	R/W
TIMG_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x0050	R/W
TIMG_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x0054	R/W
TIMG_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x0058	R/W
TIMG_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x005C	R/W
TIMG_WDTFEED_REG	Write to feed the watchdog timer	0x0060	WO
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
Configuration and Control Register for RTC CALI			
TIMG_RTCCALICFG2_REG	Timer group calibration register	0x00A8	varies
Interrupt Register			
TIMG_INT_ENA_TIMERS_REG	Interrupt enable bits	0x0098	R/W
TIMG_INT_RAW_TIMERS_REG	Raw interrupt status	0x009C	RO
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x00A0	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x00A4	WO
Version Register			
TIMG_TIMERS_DATE_REG	Version control register	0x00F8	R/W
Configuration Register			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

12.6 Registers

Register 12.1: TIMG_TxCONFIG_REG (x: 0-1) (0x0000+36*x)



TIMG_Tx_USE_XTAL 1: Use XTAL_CLK as the source clock of timer group. 0: Use APB_CLK as the source clock of timer group. (R/W)

TIMG_Tx_ALARM_EN When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W)

TIMG_Tx_LEVEL_INT_EN When set, an alarm will generate a level type interrupt. (R/W)

TIMG_Tx_EDGE_INT_EN When set, an alarm will generate an edge type interrupt. (R/W)

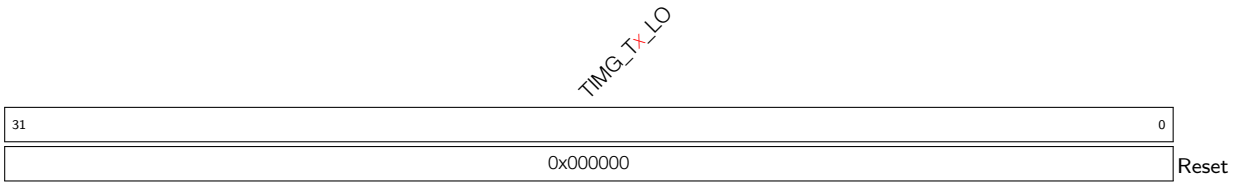
TIMG_Tx_DIVIDER Timer x clock (Tx_clk) prescaler value. (R/W)

TIMG_Tx_AUTORELOAD When set, timer x auto-reload at alarm is enabled. (R/W)

TIMG_Tx_INCREASE When set, the timer x time-base counter will increment every clock tick. When cleared, the timer x time-base counter will decrement. (R/W)

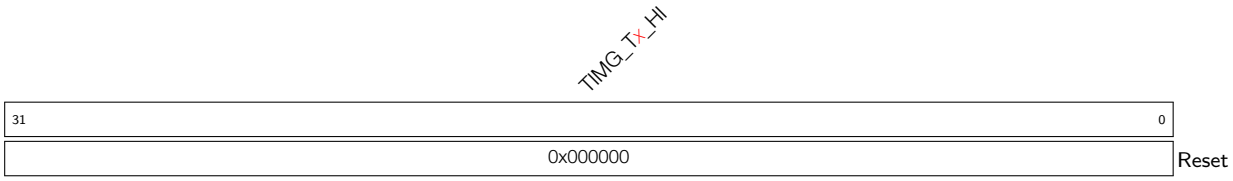
TIMG_Tx_EN When set, the timer x time-base counter is enabled. (R/W)

Register 12.2: TIMG_TxLO_REG (x: 0-1) (0x0004+36*x)



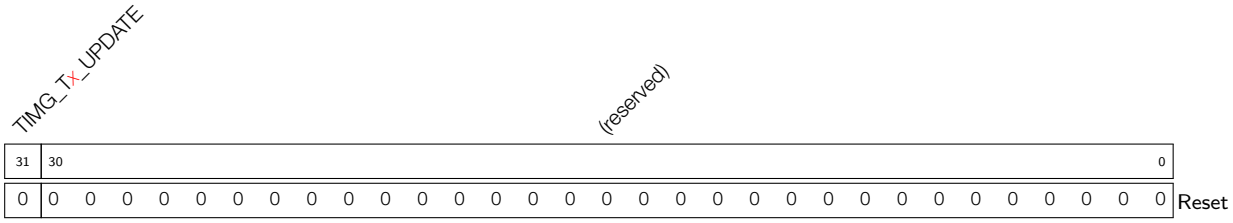
TIMG_Tx_LO After writing to TIMG_TxUPDATE_REG, the low 32 bits of the time-base counter of timer x can be read here. (RO)

Register 12.3: TIMG_TxHI_REG (x: 0-1) (0x0008+36*x)



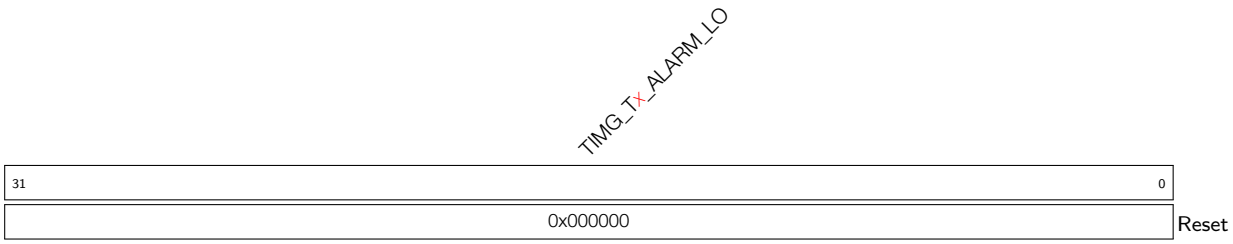
TIMG_Tx_HI After writing to TIMG_TxUPDATE_REG, the high 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 12.4: TIMG_TxUPDATE_REG (x: 0-1) (0x000C+36*x)



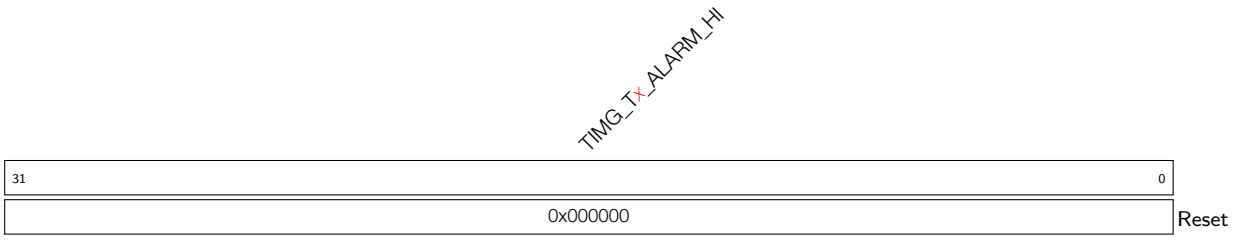
TIMG_Tx_UPDATE After writing 0 or 1 to TIMG_TxUPDATE_REG, the counter value is latched. (R/W)

Register 12.5: TIMG_TxALARMLO_REG (x: 0-1) (0x0010+36*x)



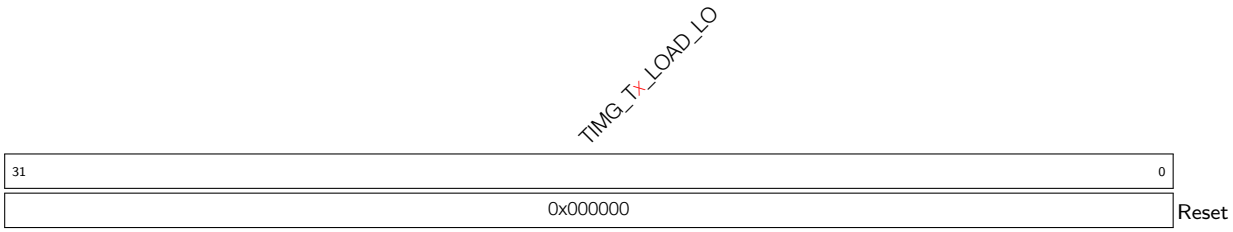
TIMG_Tx_ALARM_LO Timer *x* alarm trigger time-base counter value, low 32 bits. (R/W)

Register 12.6: TIMG_TxALARMHI_REG (x: 0-1) (0x0014+36*x)



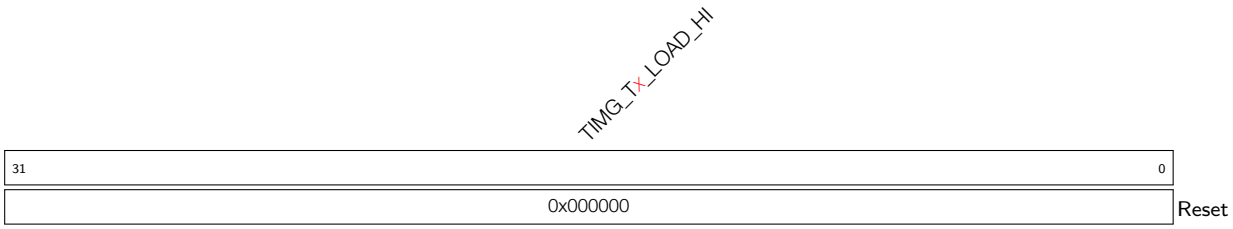
TIMG_Tx_ALARM_HI Timer *x* alarm trigger time-base counter value, high 32 bits. (R/W)

Register 12.7: TIMG_TxLOADLO_REG (x: 0-1) (0x0018+36*x)



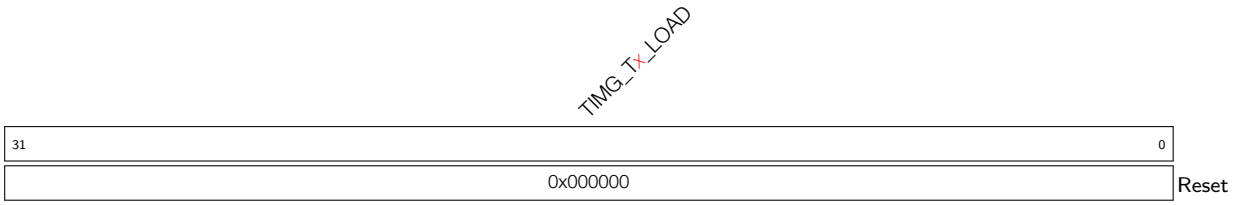
TIMG_Tx_LOAD_LO Low 32 bits of the value that a reload will load onto timer *x* time-base Counter. (R/W)

Register 12.8: TIMG_TxLOADHI_REG (x: 0-1) (0x001C+36*x)



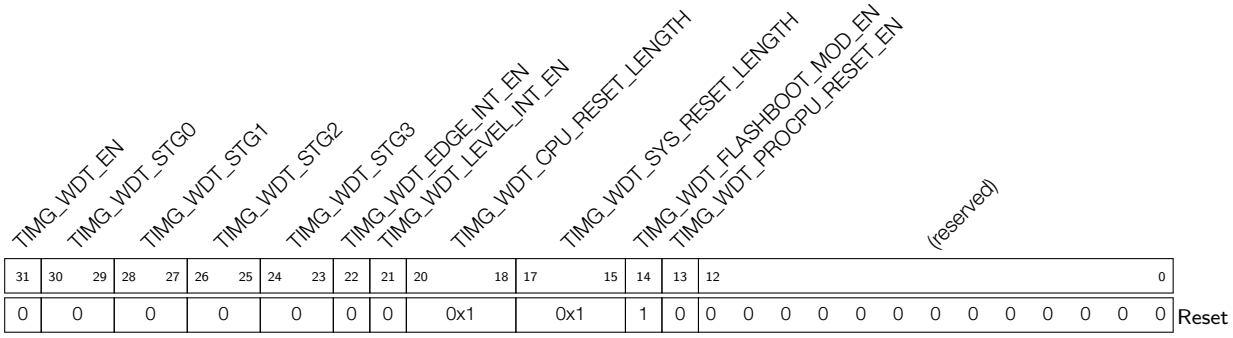
TIMG_Tx_LOAD_HI High 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 12.9: TIMG_TxLOAD_REG (x: 0-1) (0x0020+36*x)



TIMG_Tx_LOAD Write any value to trigger a timer *x* time-base counter reload. (WO)

Register 12.10: TIMG_WDTCONFIG0_REG (0x0048)



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 us, 7: 3.2 us. (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 us, 7: 3.2 us. (R/W)

TIMG_WDT_LEVEL_INT_EN When set, a level type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG_WDT_EDGE_INT_EN When set, an edge type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG_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)

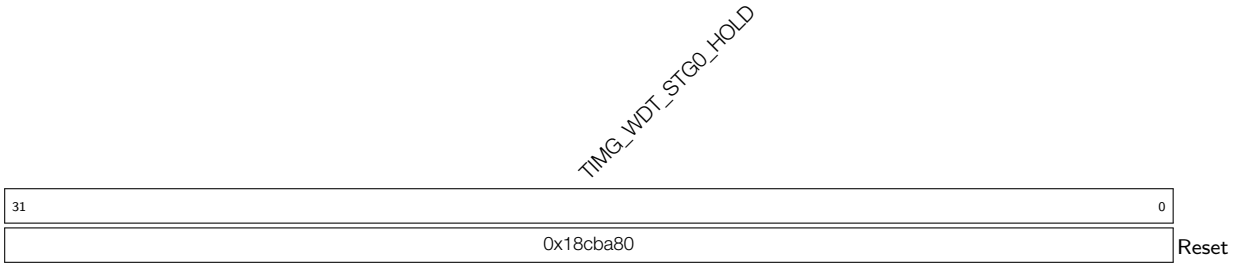
TIMG_WDT_STG0 Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG_WDT_EN When set, MWDT is enabled. (R/W)

Register 12.11: TIMG_WDTCONFIG1_REG (0x004C)

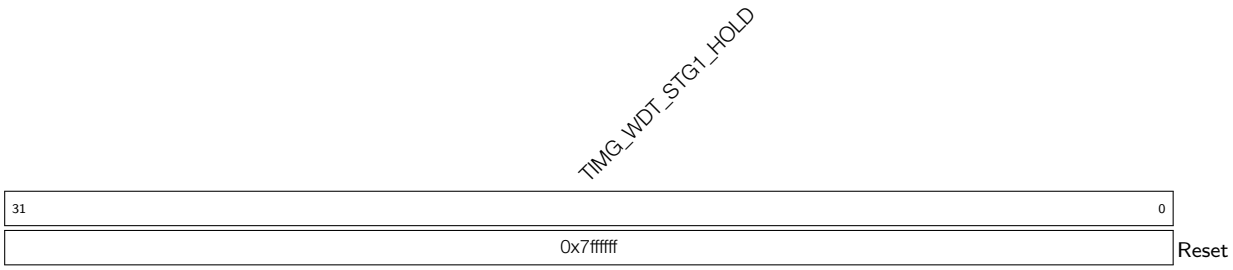
TIMG_WDT_CLK_PRESCALER MWDT clock prescaler value. MWDT clock period = 12.5 ns * TIMG_WDT_CLK_PRESCALE. (R/W)

Register 12.12: TIMG_WDTCONFIG2_REG (0x0050)



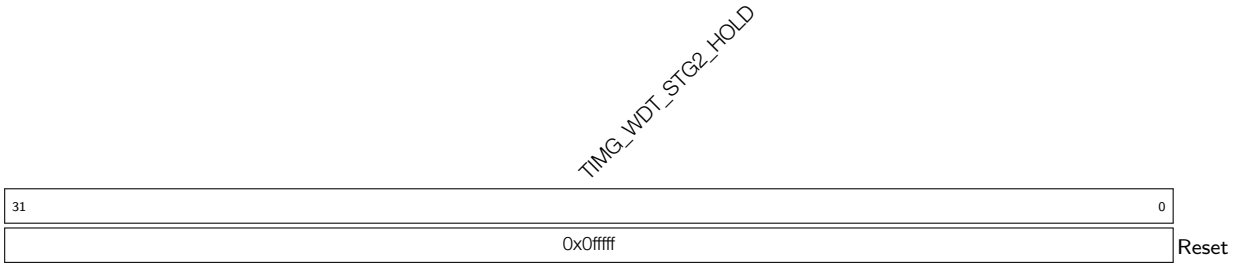
TIMG_WDT_STG0_HOLD Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 12.13: TIMG_WDTCONFIG3_REG (0x0054)



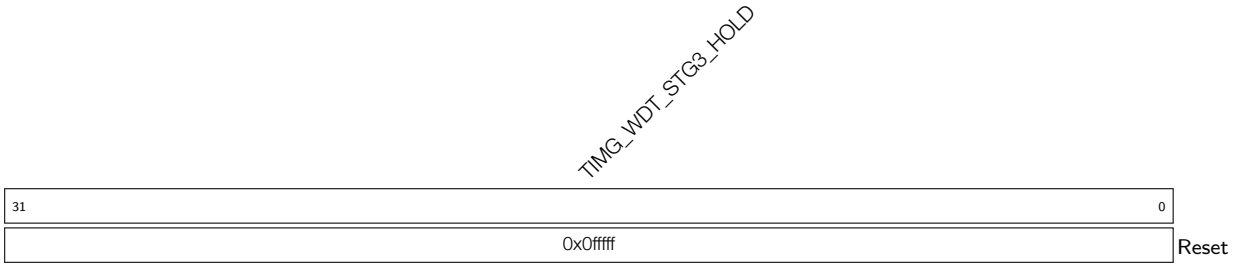
TIMG_WDT_STG1_HOLD Stage 1 timeout value, in MWDT clock cycles. (R/W)

Register 12.14: TIMG_WDTCONFIG4_REG (0x0058)



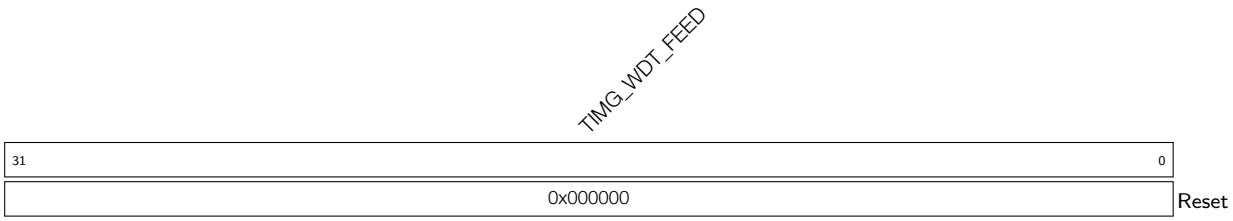
TIMG_WDT_STG2_HOLD Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 12.15: TIMG_WDTCONFIG5_REG (0x005C)



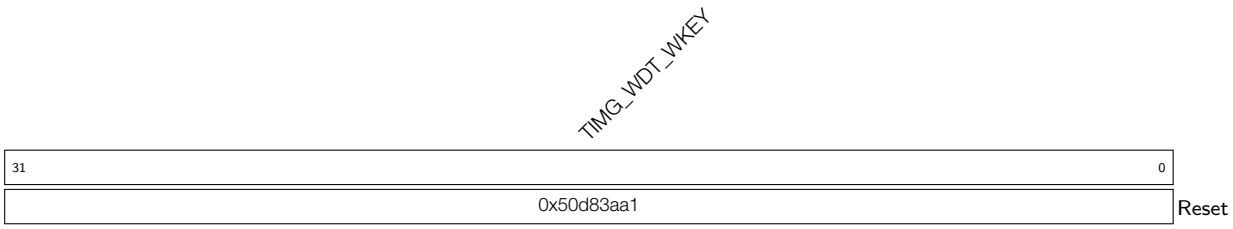
TIMG_WDT_STG3_HOLD Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 12.16: TIMG_WDTFEED_REG (0x0060)



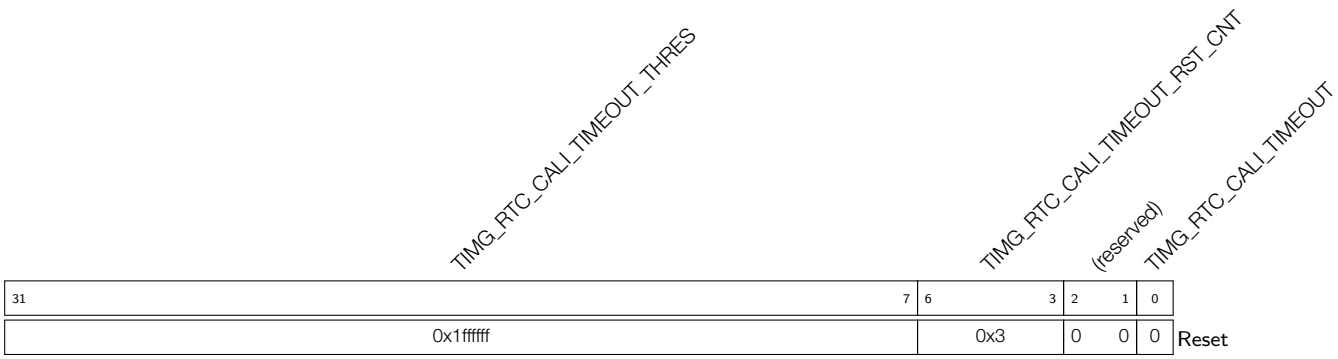
TIMG_WDT_FEED Write any value to feed the MWDT. (WO) (WO)

Register 12.17: 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 12.18: TIMG_RTCCALICFG2_REG (0x00A8)

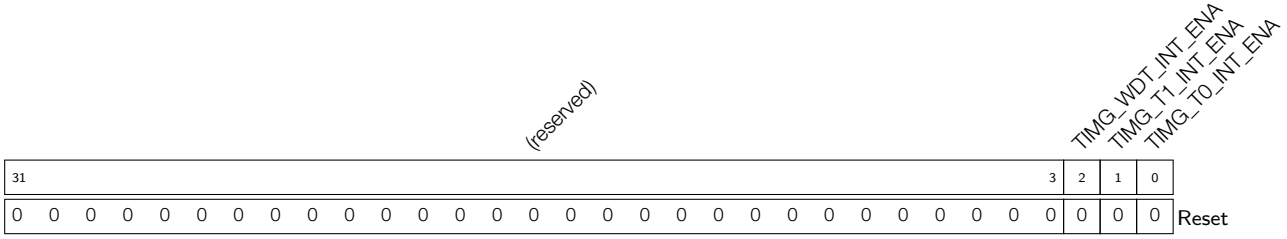


TIMG_RTC_CALI_TIMEOUT RTC calibration timeout indicator (RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT Cycles that release calibration timeout reset (R/W)

TIMG_RTC_CALI_TIMEOUT_THRES Threshold value for the RTC calibration timer. If the calibration timer's value exceeds this threshold, a timeout is triggered. (R/W)

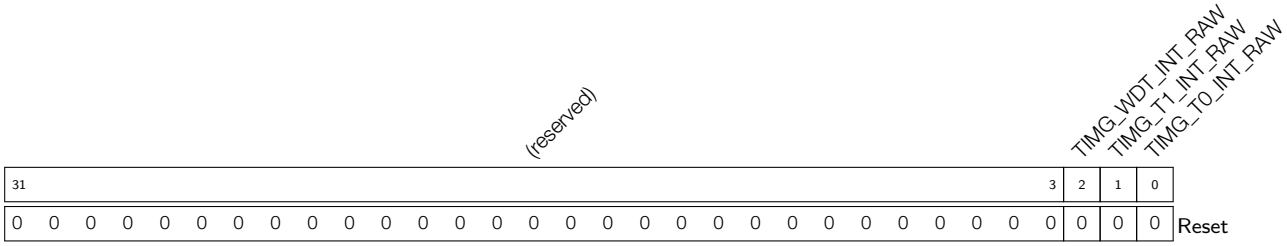
Register 12.19: TIMG_INT_ENA_TIMERS_REG (0x0098)



TIMG_T*x*_INT_ENA The interrupt enable bit for the TIMG_T*x*_INT interrupt. (R/W)

TIMG_WDT_INT_ENA The interrupt enable bit for the TIMG_WDT_INT interrupt. (R/W)

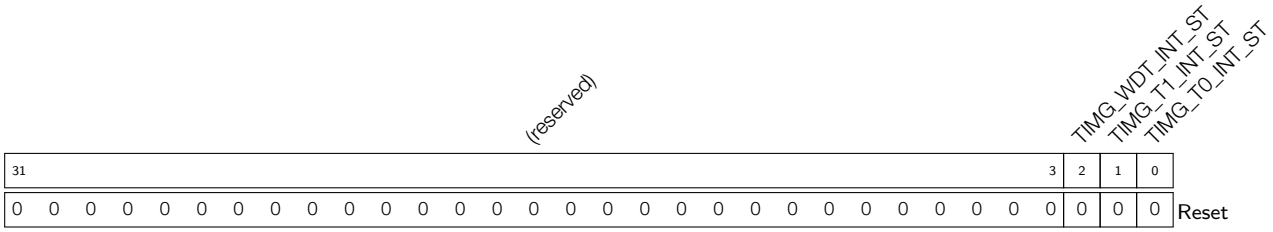
Register 12.20: TIMG_INT_RAW_TIMERS_REG (0x009C)



TIMG_T*x*_INT_RAW The raw interrupt status bit for the TIMG_T*x*_INT interrupt. (RO)

TIMG_WDT_INT_RAW The raw interrupt status bit for the TIMG_WDT_INT interrupt. (RO)

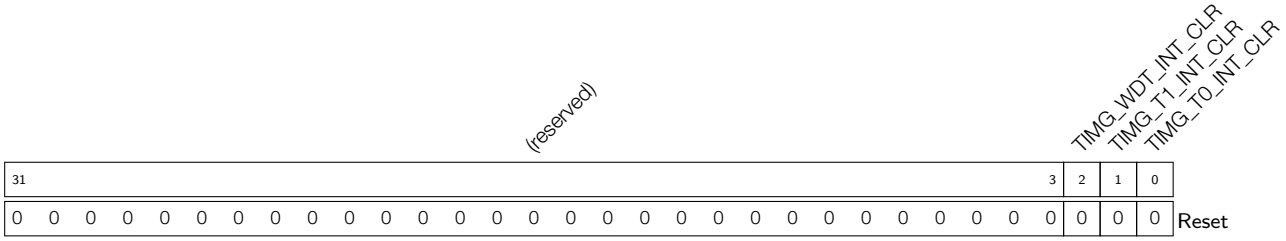
Register 12.21: TIMG_INT_ST_TIMERS_REG (0x00A0)



TIMG_T*x*_INT_ST The masked interrupt status bit for the TIMG_T*x*_INT interrupt. (RO)

TIMG_WDT_INT_ST The masked interrupt status bit for the TIMG_WDT_INT interrupt. (RO)

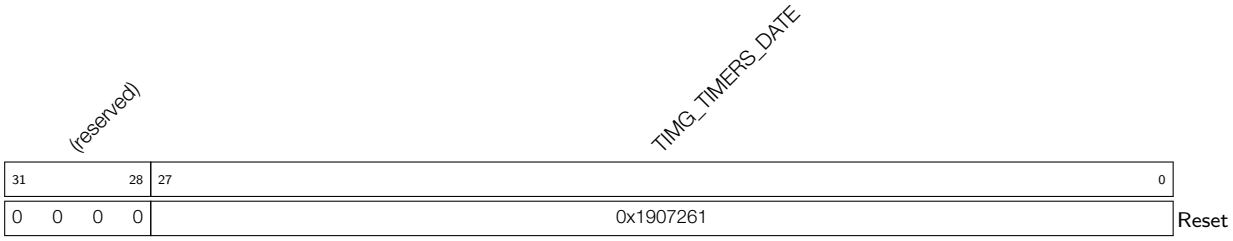
Register 12.22: TIMG_INT_CLR_TIMERS_REG (0x00A4)



TIMG_T*x*_INT_CLR Set this bit to clear the TIMG_T*x*_INT interrupt. (WO)

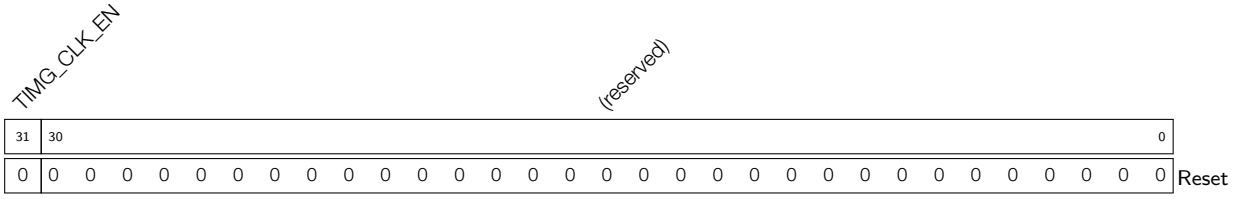
TIMG_WDT_INT_CLR Set this bit to clear the TIMG_WDT_INT interrupt. (WO)

Register 12.23: TIMG_TIMERS_DATE_REG (0x00F8)



TIMG_TIMERS_DATE Version control register. (R/W)

Register 12.24: TIMG_REGCLK_REG (0x00FC)



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)

13. Watchdog Timers

13.1 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 time out. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

The ESP32-S2 contains three watchdog timers: one in each of the two timer groups (called Main System Watchdog Timers, or MWDT) and one in the RTC Module (called the RTC Watchdog Timer, or RWDT). Each watchdog timer allows for four separately configurable stages and each stage can be programmed to take one of three (or four for RWDT) actions upon expiry, unless the watchdog is fed or disabled. The actions upon expiry are: interrupt, CPU reset, core reset and system reset. Only RWDT can trigger a system reset that will reset the entire digital circuits, which is the main system including the RTC itself. A timeout value can be set for each stage individually.

During the flash boot process, RWDT and the first MWDT are enabled automatically in order to detect and recover from booting errors.

Note that while this chapter provides the functional descriptions of the watchdog timer's, their register descriptions are provided in Chapter 12: *64-bit Timers*.

13.2 Features

Watchdog timers have the following features:

- Four stages, each with a programmable timeout value. Each stage can be configured and enabled/disabled separately
- One of three/four (for MWDTs/ RWDT) possible actions (interrupt, CPU reset, core reset and system reset) available upon expiry of each stage
- 32-bit expiry counter
- Write protection, to prevent RWDT and MWDT configuration from being altered inadvertently
- Flash boot protection
  If the boot process from an SPI flash does not complete within a predetermined period of time, the watchdog will reboot the entire main system.

13.3 Functional Description

13.3.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter. The clock source of MWDTs is derived from the APB clock via a pre-MWDT 16-bit configurable prescaler. Conversely, the clock source of RWDT is derived directly from a RTC slow clock (without a prescaler) which is usually running at 32 kHz. The 16-bit prescaler for MWDTs is configured via the TIMG_WDT_CLK_PRESCALER field of TIMG_WDTCONFIG1_REG.

MWDTs and RWDT are enabled by setting the TIMG_WDT_EN and RTC_CNTL_WDT_EN fields respectively. When enabled, the 32-bit counters of each watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e. expiry 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 MDWTs and RTC_CNTL_RTC_WDT_FEED for RWDT.

13.3.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding expiry action. When one stage expires, the expiry action is triggered, the counter value is reset to zero, and the next stage becomes active. MWDTs/ 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 MWDTs are configured in TIMG_WDTCONFIG*i*_REG (where *i* ranges from 2 to 5), whilst timeout values for RWDT are configured in RTC_CNTL_WDT_STG*j*_HOLD_REG registers (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT (T_{hold0}) is determined together by EFUSE_WDT_DELAY_SEL field of an eFuse register and RTC_CTRL_WDT_STG0_HOLD_REG. The relationship is as follows:

T_{hold0} = RTC_CTRL_WDT_STG0_HOLD_REG « EFUSE_WDT_DELAY_SEL + 1

Upon the expiry of each stage, one of the following expiry actions will be executed:

- Trigger an interrupt
  When the stage expires, an interrupt is triggered.
- Reset a CPU core
  When the stage expires, the CPU core will be reset.
- Reset the main system
  When the stage expires, the main system (which includes MWDTs, CPU, and all peripherals) will be reset. The RTC will not be reset.
- Reset the main system and RTC
  When the stage expires the main system and the RTC will both be reset. This action is only available in RWDT.
- Disabled
  This stage will have no effects on the system.

For MWDTs, the expiry action of all stages is configured in TIMG_WDTCONFIG0_REG. Likewise for RWDT, the expiry action is configured in RTC_WDTCONFIG0.

13.3.3 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, MWDTs 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 register 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 register 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 register itself) whilst the write-key register'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 register.
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 register.

13.3.4 Flash Boot Protection

During flash booting process, MWDT in timer group 0 (TIMG0), as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon expiry. Likewise, stage 0 for RWDT is configured to reset the main system and RTC when it expires. 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.

13.4 Registers

MWDT registers are part of the timer submodule and are described in the Chapter 12: *64-bit Timers* Timer Registers section.

14. XTAL32K Watchdog

14.1 Overview

XTAL32K Watchdog on ESP32-S2 is used to monitor the status of external crystal XTAL32K_CLK. This Watchdog can detect the oscillation failure of XTAL32K_CLK, change the clock source of RTC, etc. When XTAL32K_CLK works as the clock source of RTC SLOW_CLK and stops vibrating, XTAL32K Watchdog first switches to BACKUP32K_CLK derived from RTC_CLK and generates an interrupt (If the chip is in deep sleep mode, the CPU will be woken up), and then switches back to XTAL32K_CLK after it is restarted by software.

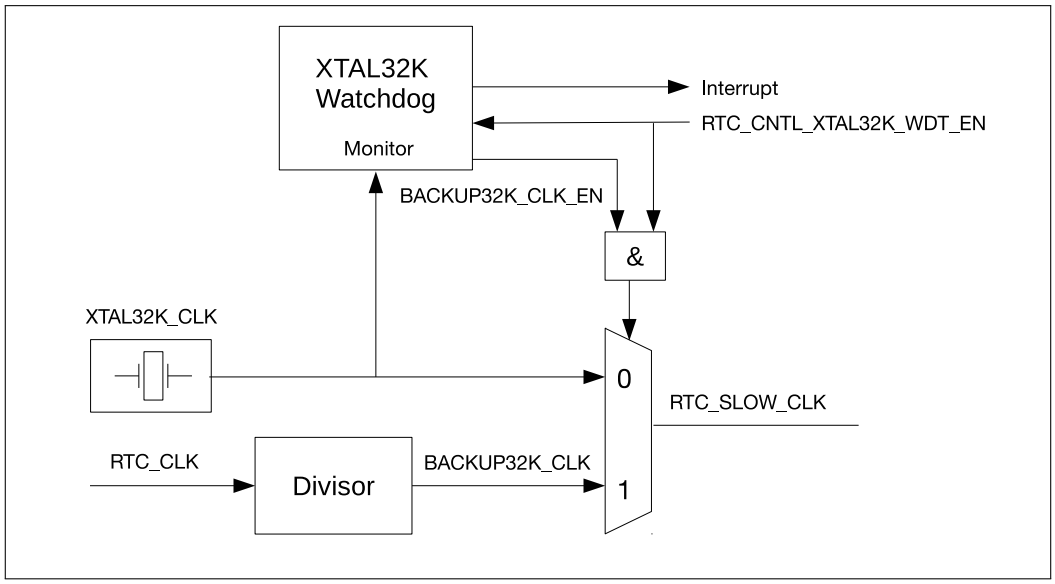


Figure 14-1. XTAL32K Watchdog

14.2 Features

14.2.1 XTAL32K Watchdog Interrupts and Wake-up

When XTAL32K Watchdog detects the oscillation failure of XTAL32K_CLK, an oscillation failure interrupt is generated. At this point the CPU will be woken up if in hibernation.

14.2.2 BACKUP32K_CLK

Once XTAL32K Watchdog detects the oscillation failure of XTAL32K_CLK, it replaces XTAL32K_CLK with BACKUP32K_CLK (with a frequency of 32 kHz) derived from RTC_CLK as RTC's SLOW_CLK, so as to ensure proper functioning of the system.

14.3 Functional Description

14.3.1 Workflow

1. XTAL32K Watchdog starts counting when RTC_CNTL_XTAL_32K_WDT_EN is enabled. The counter keeps counting until it detects the positive edge of XTAL_32K and is then cleared. When the counter reaches RTC_CNTL_XTAL32K_WDT_TIMEOUT, it generates an interrupt or a wake-up signal and is then reset.

2. XTAL32K Watchdog automatically enables BACKUP32K_CLK as the alternative clock source of RTC SLOW_CLK, to ensure the system's proper functioning and the accuracy of timers running on RTC SLOW_CLK (e.g. RTC_TIMER). For information about clock frequency configuration, please refer to Section 14.3.2 *Configuring the Divisor of BACKUP32K_CLK*.
3. To restore XTAL32K Watchdog, software restarts XTAL32K_CLK by turning its XPD signal on and off via RTC_CNTL_XPD_XTAL_32K bit. Then, XTAL32K Watchdog switches back to XTAL32K_CLK by setting RTC_CNTL_XTAL32K_WDT_EN bit to 0. If the chip is in deep sleep mode, XTAL32K Watchdog will wake up the CPU to finish the above steps.

14.3.2 Configuring the Divisor of BACKUP32K_CLK

Chips have different RTC_CLK frequencies due to production process variations. To ensure the accuracy of RTC_TIMER and other timers running on SLOW_CLK when BACKUP32K_CLK is at work, the divisor of BACKUP32K_CLK should be configured according to the actual frequency of RTC_CLK.

Define the frequency of RTC_CLK as f_rtc_clk (unit: kHz), and the eight divisor components as $x_0, x_1, x_2, x_3, x_4, x_5, x_6$, and x_7, respectively. $S = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$.

The following conditions should be fulfilled:

$$S = f\_rtc\_clk \times (4/32)$$
$$M + 1 \geq x_n \geq M (0 \leq n \leq 7)$$
$$M = f\_rtc\_clk/32/2$$

x_n should be an interger. M and S are rounded up or down. Each divisor component (x_0 ~x_7) is 4-bit long, and corresponds to the value of RTC_CNTL_XTAL32K_CLK_FACTOR (32-bit) in order.

For example, if the frequency of RTC_CLK is 163 kHz, then $f\_rtc\_clk = 163, S = 20, M = 2$, and $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} = \{2, 3, 2, 3, 2, 3, 2, 3\}$. As a result, the frequency of BACKUP32K_CLK is 32.6 kHz.

15. System Timer

15.1 Overview

System timer is a 64-bit timer specially for operating system. It can be used to schedule operating system tasks by generating periodical system ticks or certain time delay interrupts. With the help of RTC timer, system timer can keep updated after Light-sleep or Deep-sleep.

15.2 Main Features

- A 64-bit timer
- Clocked with APB_CLK
- Timer value increment step can be configured for each APB_CLK cycle.
- Support automatic time compensation in case of APB_CLK clock source switching between PLL_CLK and XTAL_CLK, to improve timer accuracy.
- Generate three independent interrupts based on different alarm values or periods (targets).
- Support for 64-bit alarm values and 30-bit periods.
- Load back sleep time recorded by RTC timer after Deep-sleep or Light-sleep by software.
- Keep stalled if CPU is stalled or CPU is in on-chip-debugging mode.

15.3 Clock Source Selection

The System Timer is driven using XTAL_CLK or PLL_CLK clock. Selection between the two clock sources is described in Table CPU_CLK Source in Chapter 2 *Reset and Clock*. For the specific clock frequency used for the System Timer, please refer to Table 11 *APB_CLK Source*. On each clock period the timer will be increased by a step value configured in either SYSTIMER_TIMER_XTAL_STEP or SYSTIMER_TIMER_PLL_STEP, depending on which clock source is used.

15.4 Functional Description

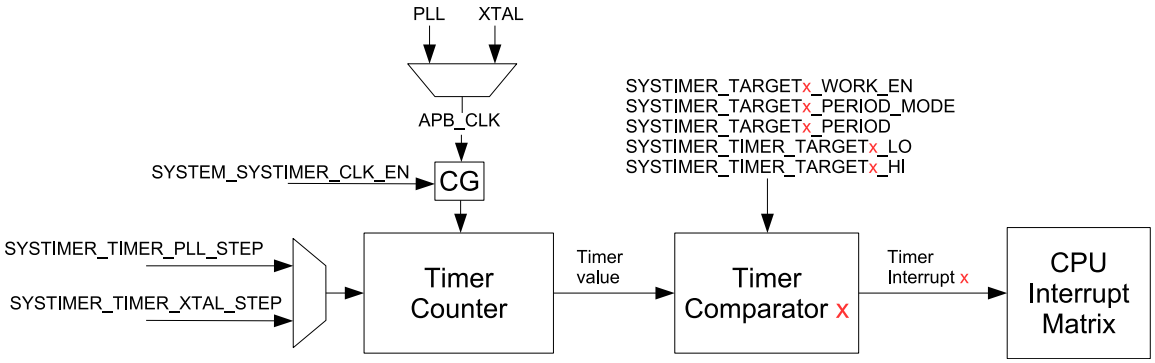


Figure 15-1. System Timer Structure

Figure 15-1 shows the structure of system timer. The system timer can be enabled by setting the bit SYSTEM_SYSTIMER_CLK_EN in register SYSTEM_PERIP_CLK_EN0_REG and be reset via software by setting

the bit SYSTEM_SYSTIMER_RST in register SYSTEM_PERIP_RST_EN0_REG. For more information, please refer to Table 33 *Peripheral Clock Gating and Reset Bits* in Chapter 6 *System Registers*.

15.4.1 Read System Timer Value

1. Set SYSTIMER_TIMER_UPDATE to update the timer value into registers.
2. Wait till SYSTIMER_TIMER_VALUE_VALID is set, which means users now can read the timer values from registers.
3. Read the high 32 bits of the timer value from SYSTIMER_TIMER_VALUE_HI, and the low 32 bits from SYSTIMER_TIMER_VALUE_LO.

15.4.2 Configure a Time-Delay Alarm

1. Read the current value of system timer, see Section 15.4.1. This value will be used to calculate the target in Step 3.
2. Clear SYSTIMER_TARGET*x*_PERIOD_MODE to set the timer into a time-delay alarm mode.
3. Write the high 32 bits of the target (alarm value) to SYSTIMER_TIMER_TARGET*x*_HI, and the low 32 bits to SYSTIMER_TIMER_TARGET*x*_LO.
4. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected work mode.
5. Set SYSTIMER_INT*x*_ENA to enable timer interrupt. When the timer counts to the alarm value, an interrupt will be triggered.

15.4.3 Configure Periodic Alarms

1. Set SYSTIMER_TARGET*x*_PERIOD_MODE to configure the timer into periodic alarms mode.
2. Write the target (alarm period) to SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TARGET*x*_WORK_EN to enable periodical alarms mode.
4. Set SYSTIMER_INT*x*_ENA to enable timer interrupt. An interrupt will be triggered when the timer counts to the target value set in Step 2.

15.4.4 Update after Deep-sleep and Light-sleep

1. Configure RTC timer before the chip goes to Deep-sleep or Light-sleep, to record the exact sleep time.
2. Read the sleep time from RTC timer when the chip is woken up from Deep-sleep or Light-sleep.
3. Read current value of the system timer. See Section 15.4.1 for how to read the timer value.
4. Add current value of the system timer to the time that RTC timer records in Deep-sleep mode or in Light-sleep mode.
5. Write the result into SYSTIMER_TIMER_LOAD_HI (high 32 bits), and into SYSTIMER_TIMER_LOAD_LO (low 32 bits).
6. Set SYSTIMER_TIMER_LOAD to load the new value stored in SYSTIMER_TIMER_LOAD_HI and SYSTIMER_TIMER_LOAD_LO into the system timer. By such way, the system timer is updated.

15.5 Base Address

Users can access system timer registers with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 52: System Timer Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F423000
PeriBUS2	0x60023000

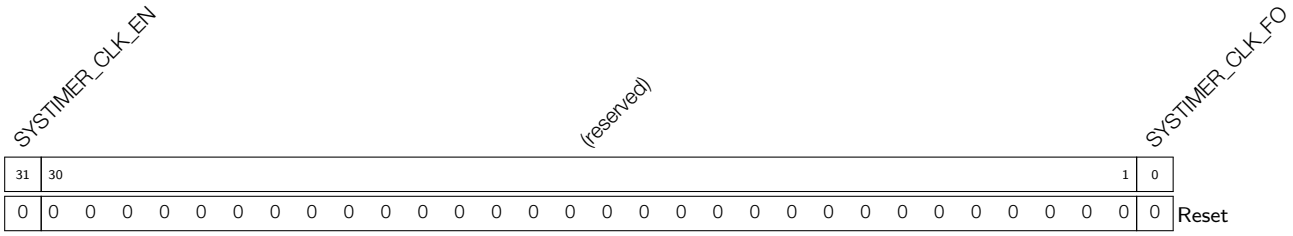
15.6 Register Summary

The addresses in the following table are relative to system timer base addresses provided in Section 15.5.

Name	Description	Address	Access
System Timer Registers			
SYSTIMER_CONF_REG	Configure system timer clock	0x0000	R/W
SYSTIMER_LOAD_REG	Load value to system timer	0x0004	WO
SYSTIMER_LOAD_HI_REG	High 32 bits to be loaded to system timer	0x0008	R/W
SYSTIMER_LOAD_LO_REG	Low 32 bits to be loaded to system timer	0x000C	R/W
SYSTIMER_STEP_REG	System timer accumulation step	0x0010	R/W
SYSTIMER_TARGET0_HI_REG	System timer target 0, high 32 bits	0x0014	R/W
SYSTIMER_TARGET0_LO_REG	System timer target 0, low 32 bits	0x0018	R/W
SYSTIMER_TARGET1_HI_REG	System timer target 1, high 32 bits	0x001C	R/W
SYSTIMER_TARGET1_LO_REG	System timer target 1, low 32 bits	0x0020	R/W
SYSTIMER_TARGET2_HI_REG	System timer target 2, high 32 bits	0x0024	R/W
SYSTIMER_TARGET2_LO_REG	System timer target 2, low 32 bits	0x0028	R/W
SYSTIMER_TARGET0_CONF_REG	Configure work mode for system timer target 0	0x002C	R/W
SYSTIMER_TARGET1_CONF_REG	Configure work mode for system timer target 1	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	Configure work mode for system timer target 2	0x0034	R/W
SYSTIMER_UPDATE_REG	Read out system timer value	0x0038	varies
SYSTIMER_VALUE_HI_REG	System timer value, high 32 bits	0x003C	RO
SYSTIMER_VALUE_LO_REG	System timer value, low 32 bits	0x0040	RO
SYSTIMER_INT_ENA_REG	System timer interrupt enable	0x0044	R/W
SYSTIMER_INT_RAW_REG	System timer interrupt raw	0x0048	RO
SYSTIMER_INT_CLR_REG	System timer interrupt clear	0x004C	WO
Version Register			
SYSTIMER_DATE_REG	Version control register	0x00FC	R/W

15.7 Registers

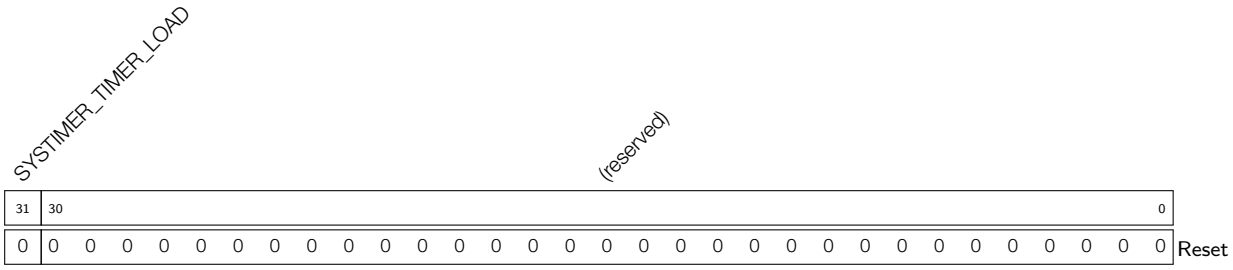
Register 15.1: SYSTIMER_CONF_REG (0x0000)



SYSTIMER_CLK_FO System timer clock force enable. (R/W)

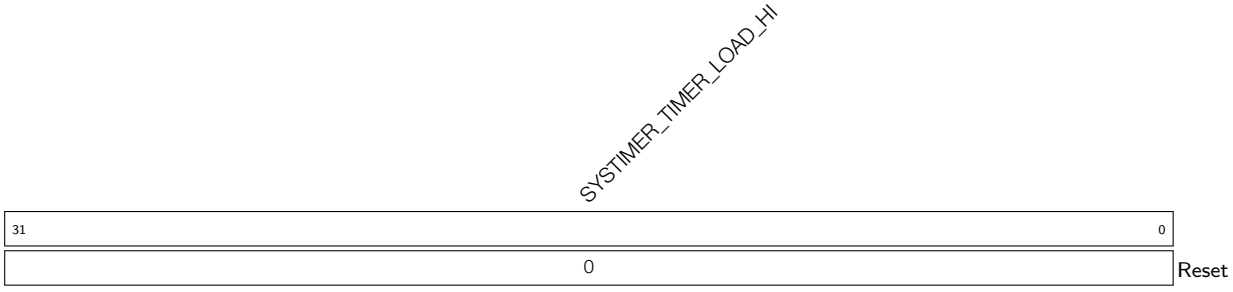
SYSTIMER_CLK_EN Register clock enable. (R/W)

Register 15.2: SYSTIMER_LOAD_REG (0x0004)



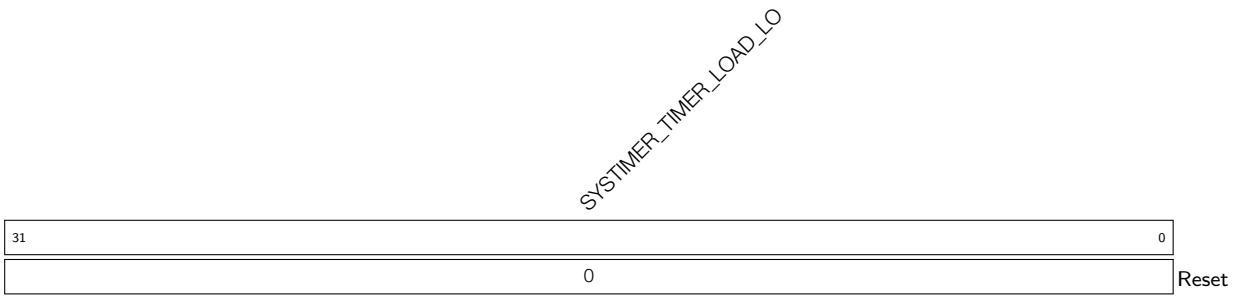
SYSTIMER_TIMER_LOAD Set this bit to 1, the value stored in SYSTIMER_TIMER_LOAD_HI and in SYSTIMER_TIMER_LOAD_LO will be loaded to system timer. (WO)

Register 15.3: SYSTIMER_LOAD_HI_REG (0x0008)



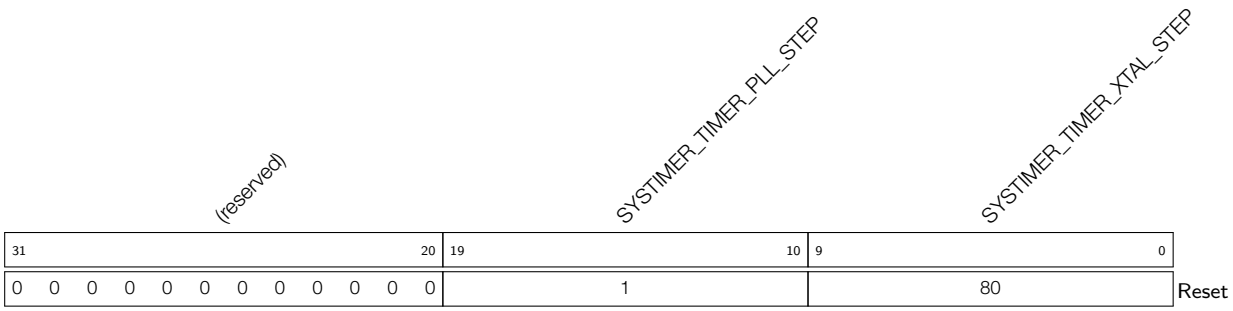
SYSTIMER_TIMER_LOAD_HI The value to be loaded into system timer, high 32 bits. (R/W)

Register 15.4: SYSTIMER_LOAD_LO_REG (0x000C)



SYSTIMER_TIMER_LOAD_LO The value to be loaded into system timer, low 32 bits. (R/W)

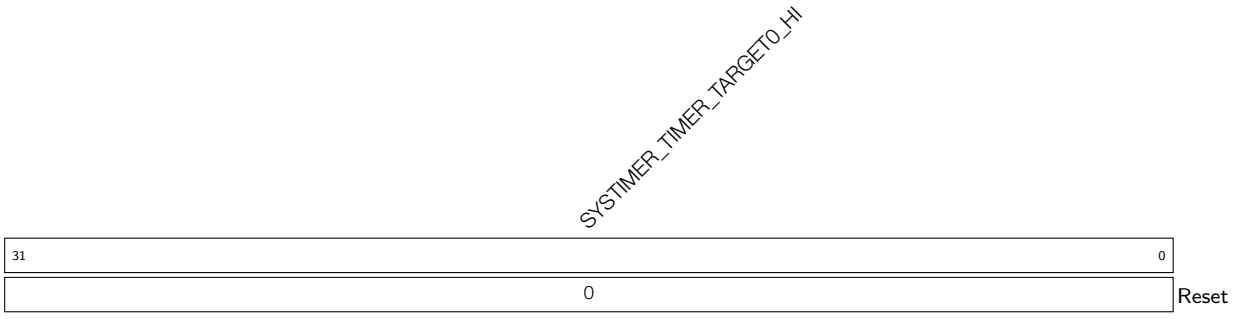
Register 15.5: SYSTIMER_STEP_REG (0x0010)



SYSTIMER_TIMER_XTAL_STEP Set system timer increment step when using XTAL_CLK. (R/W)

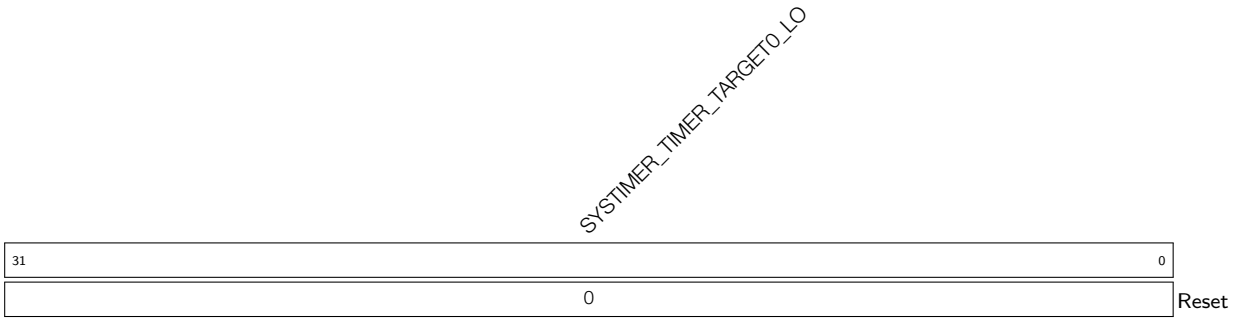
SYSTIMER_TIMER_PLL_STEP Set system timer increment step when using PLL_CLK. (R/W)

Register 15.6: SYSTIMER_TARGET0_HI_REG (0x0014)



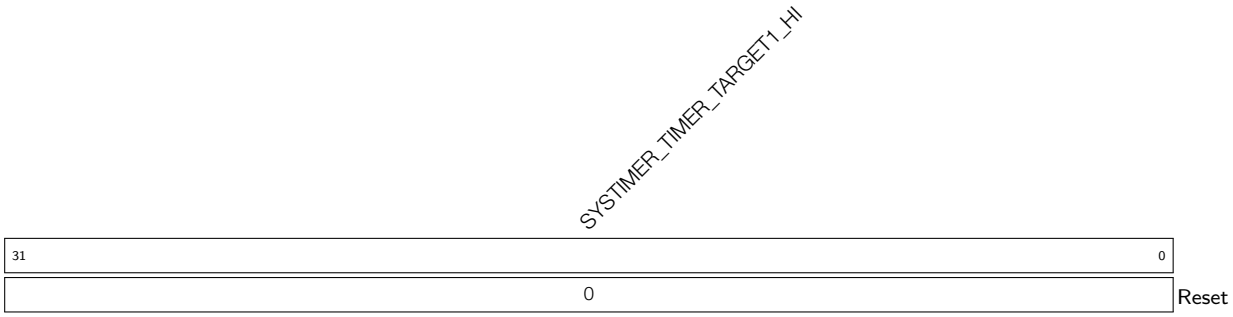
SYSTIMER_TIMER_TARGET0_HI System timer target 0, high 32 bits. (R/W)

Register 15.7: SYSTIMER_TARGET0_LO_REG (0x0018)



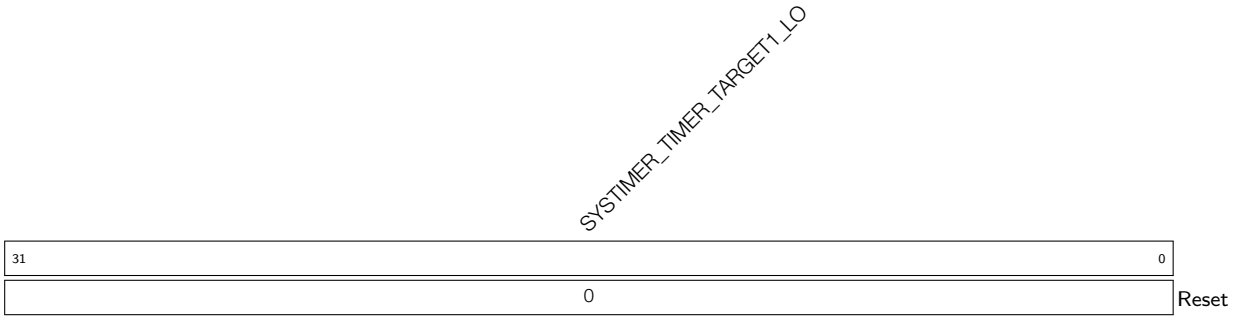
SYSTIMER_TIMER_TARGET0_LO System timer target 0, low 32 bits. (R/W)

Register 15.8: SYSTIMER_TARGET1_HI_REG (0x001C)



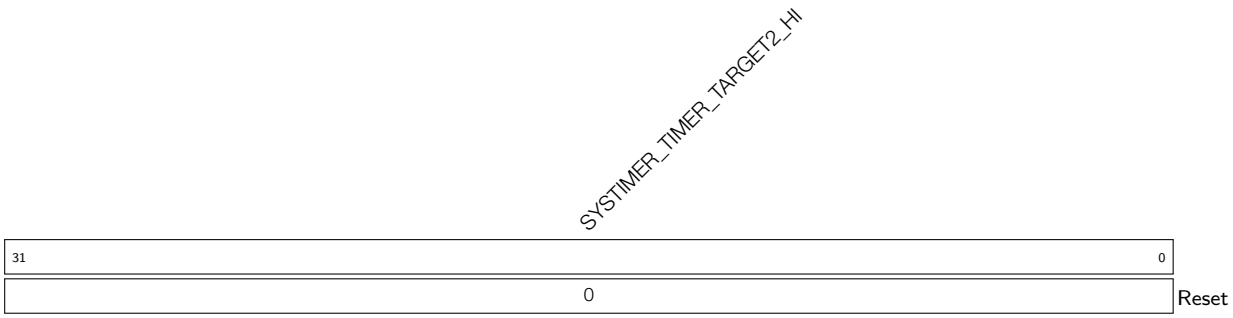
SYSTIMER_TIMER_TARGET1_HI System timer target 1, high 32 bits. (R/W)

Register 15.9: SYSTIMER_TARGET1_LO_REG (0x0020)



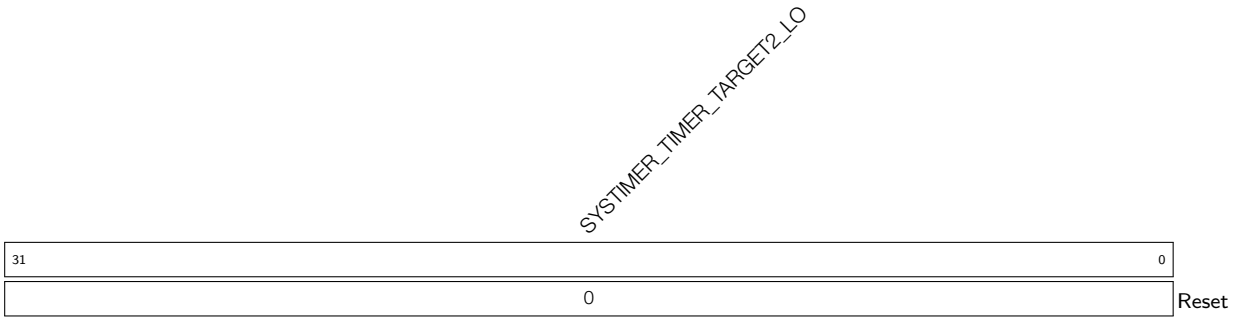
SYSTIMER_TIMER_TARGET1_LO System timer target 1, low 32 bits. (R/W)

Register 15.10: SYSTIMER_TARGET2_HI_REG (0x0024)



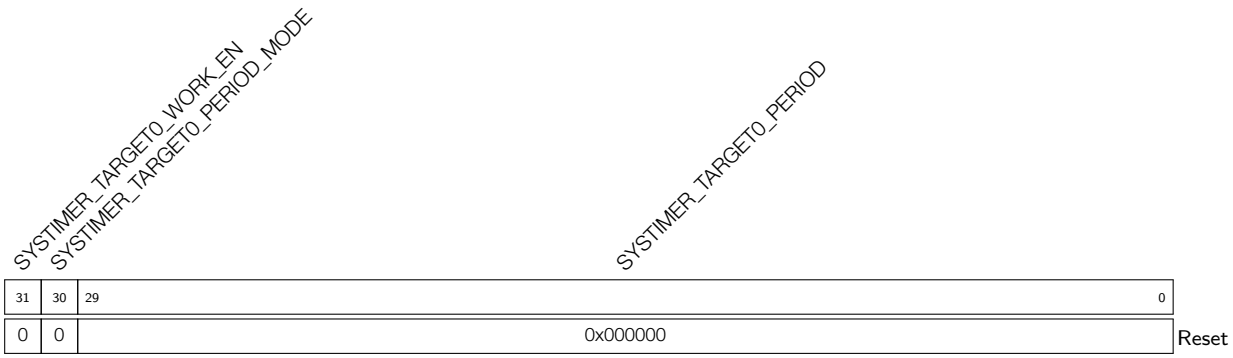
SYSTIMER_TIMER_TARGET2_HI System timer target 2, high 32 bits. (R/W)

Register 15.11: SYSTIMER_TARGET2_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET2_LO System timer target 2, low 32 bits. (R/W)

Register 15.12: SYSTIMER_TARGET0_CONF_REG (0x002C)

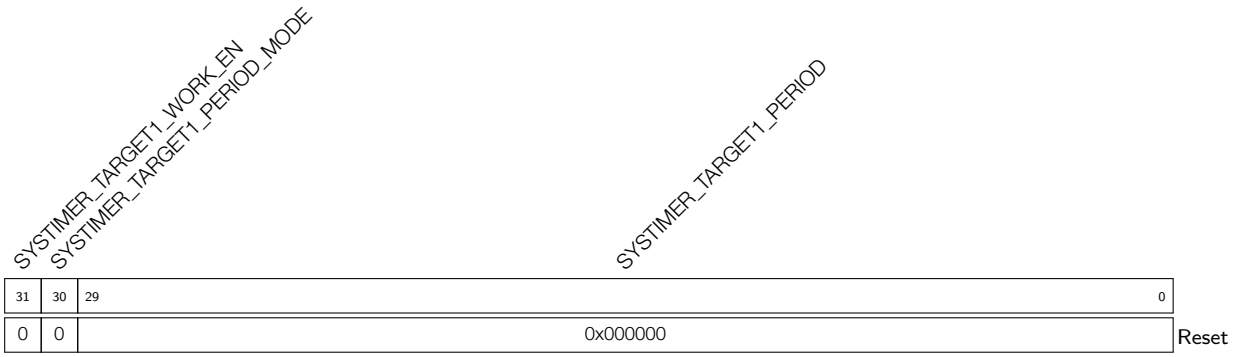


SYSTIMER_TARGET0_PERIOD Set alarm period for system timer target 0, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Set work mode for system timer target 0. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET0_WORK_EN System timer target 0 work enable. (R/W)

Register 15.13: SYSTIMER_TARGET1_CONF_REG (0x0030)

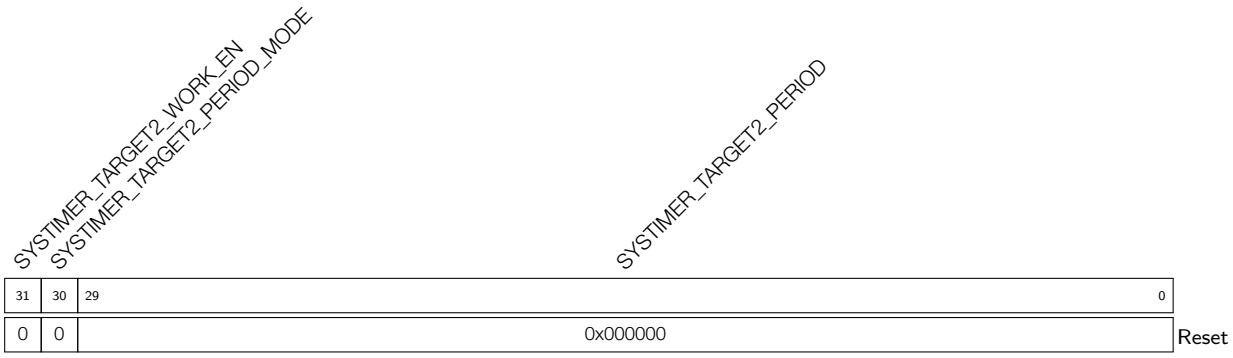


SYSTIMER_TARGET1_PERIOD Set alarm period for system timer target 1, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Set work mode for system timer target 1. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET1_WORK_EN System timer target 1 work enable. (R/W)

Register 15.14: SYSTIMER_TARGET2_CONF_REG (0x0034)

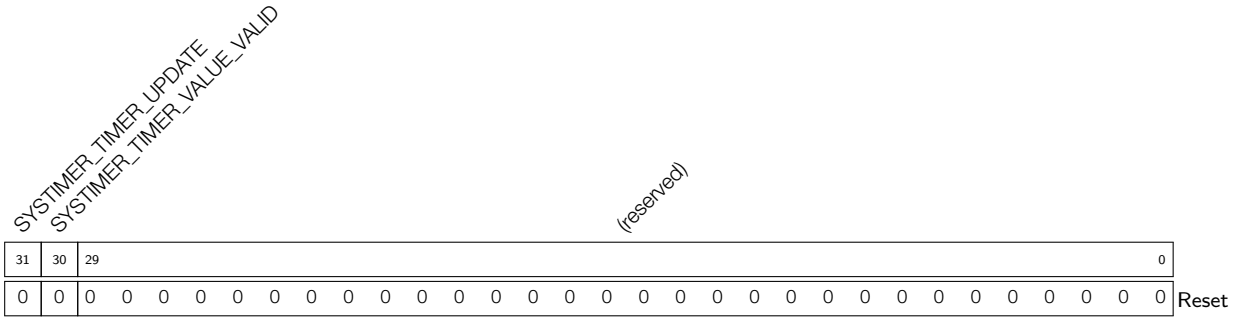


SYSTIMER_TARGET2_PERIOD Set alarm period for sytem timer target 2, only valid in periodic alarms mode. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Set work mode for system timer target 2. 0: work in a time-delay alarm mode; 1: work in periodic alarms mode. (R/W)

SYSTIMER_TARGET2_WORK_EN System timer target 2 work enable. (R/W)

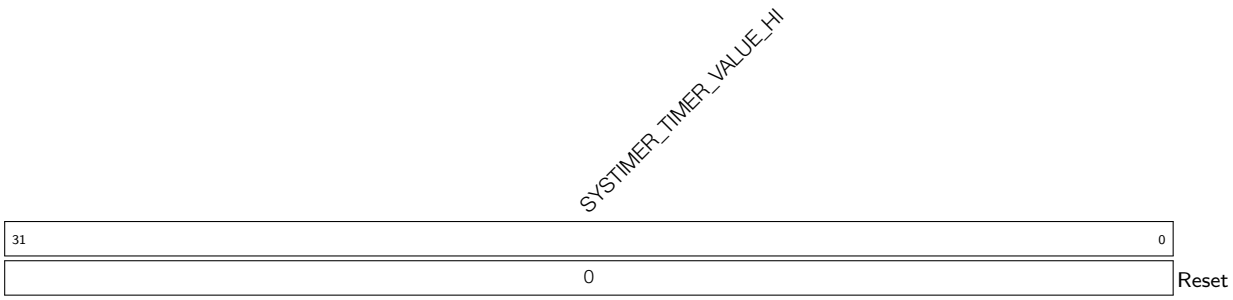
Register 15.15: SYSTIMER_UPDATE_REG (0x0038)



SYSTIMER_TIMER_VALUE_VALID Check if it is valid to read out timer value from registers. 0: Not ready to read timer value from registers; 1: Ready to read timer value from registers. (RO)

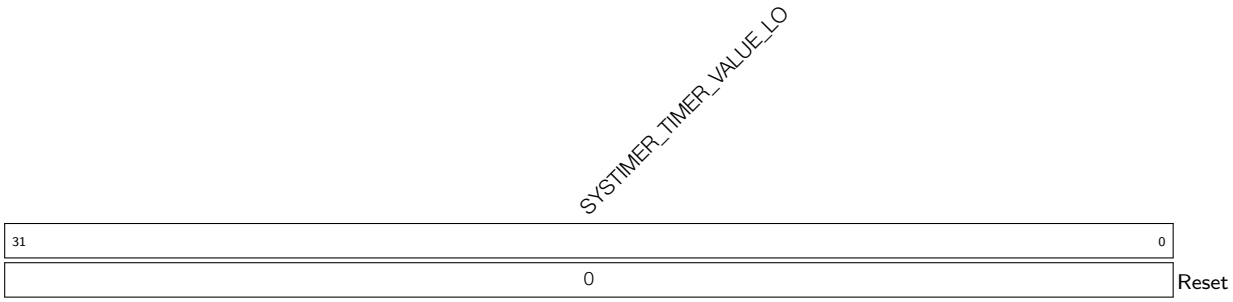
SYSTIMER_TIMER_UPDATE Update system timer value to registers. (WO)

Register 15.16: SYSTIMER_VALUE_HI_REG (0x003C)



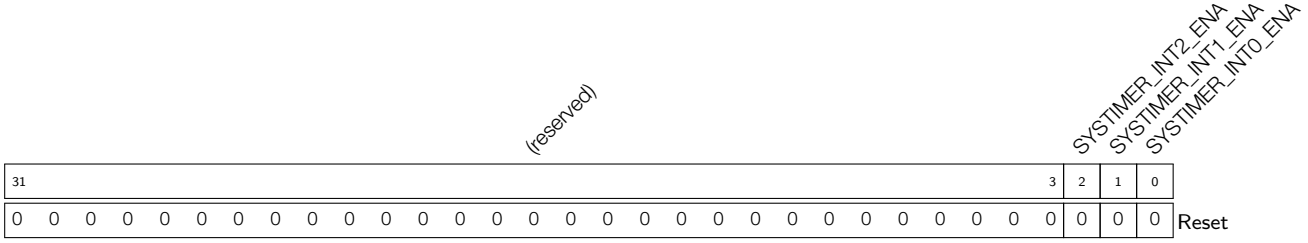
SYSTIMER_TIMER_VALUE_HI System timer value, high 32 bits. (RO)

Register 15.17: SYSTIMER_VALUE_LO_REG (0x0040)



SYSTIMER_TIMER_VALUE_LO System timer value, low 32 bits. (RO)

Register 15.18: SYSTIMER_INT_ENA_REG (0x0044)

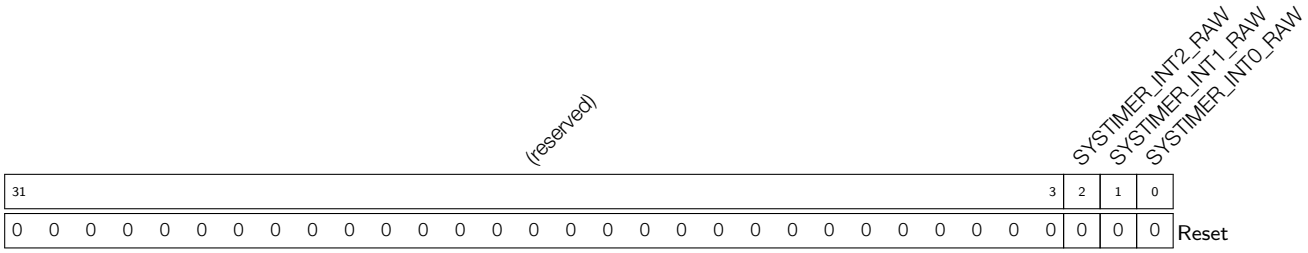


SYSTIMER_INT0_ENA Interrupt enable bit of system timer target 0. (R/W)

SYSTIMER_INT1_ENA Interrupt enable bit of system timer target 1. (R/W)

SYSTIMER_INT2_ENA Interrupt enable bit of system timer target 2. (R/W)

Register 15.19: SYSTIMER_INT_RAW_REG (0x0048)

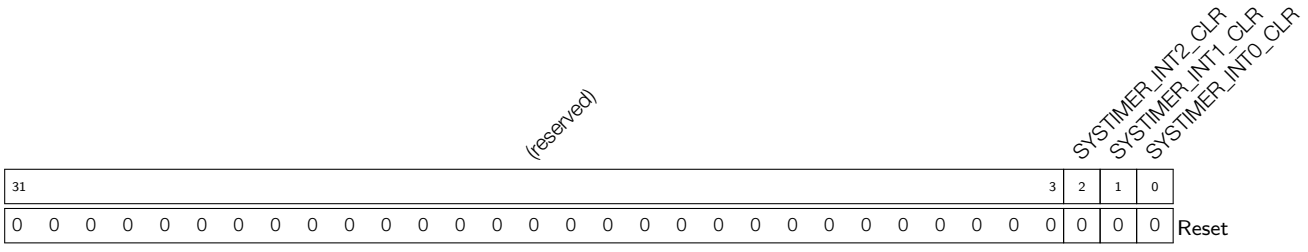


SYSTIMER_INT0_RAW Interrupt raw bit of system timer target 0. (RO)

SYSTIMER_INT1_RAW Interrupt raw bit of system timer target 1. (RO)

SYSTIMER_INT2_RAW Interrupt raw bit of system timer target 2. (RO)

Register 15.20: SYSTIMER_INT_CLR_REG (0x004C)

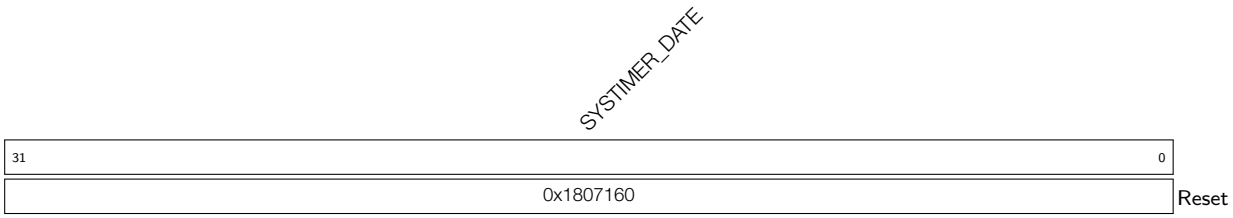


SYSTIMER_INT0_CLR Interrupt clear bit of system timer target 0. (WO)

SYSTIMER_INT1_CLR Interrupt clear bit of system timer target 1. (WO)

SYSTIMER_INT2_CLR Interrupt clear bit of system timer target 2. (WO)

Register 15.21: SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

16. eFuse Controller

16.1 Overview

ESP32-S2 has a 4096-bit eFuse that stores parameters in the SoC. Once an eFuse bit is programmed to 1, it can never be reverted to 0. Software can instruct the eFuse Controller to program individual bits for individual parameters as needed. Some of these parameters can be read by software using the eFuse Controller registers, while some can be directly used by hardware modules.

16.2 Features

- One-time programmable storage
- Configurable write protection
- Configurable software read protection
- Parameters use different hardware encoding schemes to protect against corruption

16.3 Functional Description

16.3.1 Structure

There are 11 eFuse blocks (BLOCK0 ~ BLOCK10).

BLOCK0 holds most core system parameters. Among these parameters, 24 bits are invisible to software and can only be used by hardware and 38 bits are reserved for future use.

Table 54 lists all the parameters in BLOCK0 and their offsets, bit widths, functional description, as well as information on whether they can be used by hardware, and whether they are protected from programming.

The **EFUSE_WR_DIS** parameter is used to restrict the programming of other parameters, while **EFUSE_RD_DIS** is used to restrict software from reading BLOCK4 ~ BLOCK10. More information on these two parameters can be found in sections 16.3.1.1 and 16.3.1.2.

Table 54: Parameters in BLOCK0

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_WR_DIS	0	32	Y	N/A	Disables programming of individual eFuses
EFUSE_RD_DIS	32	7	Y	0	Disables software reading from individual eFuse blocks BLOCK4-10
EFUSE_DIS_RTC_RAM_BOOT	39	1	N	1	Disables boot from RTC RAM
EFUSE_DIS_ICACHE	40	1	Y	2	Disables ICache
EFUSE_DIS_DCACHE	41	1	Y	2	Disables DCache
EFUSE_DIS_DOWNLOAD_ICACHE	42	1	Y	2	Disables Icache when SoC is in Download mode
EFUSE_DIS_DOWNLOAD_DCACHE	43	1	Y	2	Disables Dcache when SoC is in Download mode

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_DIS_FORCE_DOWNLOAD	44	1	Y	2	Disables forcing chip into Download mode
EFUSE_DIS_USB	45	1	Y	2	Disables the USB OTG hardware
EFUSE_DIS_CAN	46	1	Y	2	Disables the TWAI Controller hardware
EFUSE_DIS_BOOT_REMAP	47	1	Y	2	Disables capability to Remap RAM to ROM address space
EFUSE_SOFT_DIS_JTAG	49	1	Y	2	Software disables JTAG. When software disabled, JTAG can be activated temporarily by HMAC peripheral.
EFUSE_HARD_DIS_JTAG	50	1	Y	2	Hardware disables JTAG permanently.
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	51	1	Y	2	Disables flash encryption when in download boot modes
EFUSE_USB_EXCHG_PINS	56	1	Y	30	Exchanges USB D+ and D- pins
EFUSE_EXT_PHY_ENABLE	57	1	N	30	Enables external USB PHY
EFUSE_USB_FORCE_NOPERSIST	58	1	N	30	Forces to set USB BVALID to 1
EFUSE_VDD_SPI_XPD	68	1	Y	3	If VDD_SPI_FORCE is 1, determines if the VDD_SPI regulator is powered on
EFUSE_VDD_SPI_TIEH	69	1	Y	3	If VDD_SPI_FORCE is 1, determines VDD_SPI voltage. 0: VDD_SPI connects to 1.8 V LDO; 1: VDD_SPI connects to VDD_RTC_IO
EFUSE_VDD_SPI_FORCE	70	1	Y	3	When set, XPD_VDD_PSI_REG and VDD_SPI_TIEH will be used to configure VDD_SPI LDO
EFUSE_WDT_DELAY_SEL	80	2	Y	3	Selects RTC WDT timeout threshold at startup
EFUSE_SPI_BOOT_CRYPT_CNT	82	3	Y	4	Enables encryption and decryption, when an SPI boot mode is set. Feature is enabled when 1 or 3 bits are set in the eFuse, disabled otherwise.
EFUSE_SECURE_BOOT_KEY_REVOKE0	85	1	N	5	If set, revokes use of secure boot key digest 0

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_SECURE_BOOT_KEY_REVOKE1	86	1	N	6	If set, revokes use of secure boot key digest 1
EFUSE_SECURE_BOOT_KEY_REVOKE2	87	1	N	7	If set, revokes use of secure boot key digest 2
EFUSE_KEY_PURPOSE_0	88	4	Y	8	KEY0 purpose, see Table 55
EFUSE_KEY_PURPOSE_1	92	4	Y	9	KEY1 purpose, see Table 55
EFUSE_KEY_PURPOSE_2	96	4	Y	10	KEY2 purpose, see Table 55
EFUSE_KEY_PURPOSE_3	100	4	Y	11	KEY3 purpose, see Table 55
EFUSE_KEY_PURPOSE_4	104	4	Y	12	KEY4 purpose, see Table 55
EFUSE_KEY_PURPOSE_5	108	4	Y	13	KEY5 purpose, see Table 55
EFUSE_SECURE_BOOT_EN	116	1	N	15	Enables secure boot
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	117	1	N	16	Enables aggressive secure boot key revocation mode
EFUSE_FLASH_TPUW	124	4	N	18	Configures flash startup delay after SoC power-up, unit is (ms/2). When the value is 15, delay is 7.5 ms.
EFUSE_DIS_DOWNLOAD_MODE	128	1	N	18	Disables all Download boot modes
EFUSE_DIS_LEGACY_SPI_BOOT	129	1	N	18	Disables Legacy SPI boot mode
EFUSE_UART_PRINT_CHANNEL	130	1	N	18	Selects the default UART for printing boot messages, 0: UART0; 1: UART1
EFUSE_DIS_USB_DOWNLOAD_MODE	132	1	N	18	Disables use of USB in UART download boot mode
EFUSE_ENABLE_SECURITY_DOWNLOAD	133	1	N	18	Enables secure UART download mode (read/write flash only)
EFUSE_UART_PRINT_CONTROL	134	2	N	18	Sets the default UART boot message output mode. 2'b00: Enabled; 2'b01: Enable when GPIO 46 is low at reset; 2'b10: Enable when GPIO 46 is high at rest; 2'b11: Disabled.
EFUSE_PIN_POWER_SELECTION	136	1	N	18	Sets default power supply for GPIO33 ~ GPIO37, set when SPI flash is initialized. 0: VDD3P3_CPU; 1: VDD_SPI
EFUSE_FLASH_TYPE	137	1	N	18	Selects SPI flash type, 0: maximum four data lines; 1: eight data lines

Parameters	Offset	Bit Width	Hardware Use	Programming-Protection by EFUSE_WR_DIS Bit Number	Description
EFUSE_FORCE_SEND_RESUME	138	1	N	18	Forces ROM code to send an SPI flash resume command during SPI boot
EFUSE_SECURE_VERSION	139	16	N	18	Secure version (used by ESP-IDF anti-rollback feature)

Table 55 lists all key purpose and their values. Setting the eFuse parameter EFUSE_KEY_PURPOSE_*n* programs the purpose for eFuse block KEY*n* (*n*: 0 ~ 5).

Table 55: Key Purpose Values

Key Purpose Values	Purposes
0	User purposes (software-only use)
1	Reserved
2	XTS_AES_256_KEY_1 (flash/PSRAM encryption)
3	XTS_AES_256_KEY_2 (flash/PSRAM encryption)
4	XTS_AES_128_KEY (flash/PSRAM encryption)
5	HMAC Downstream mode
6	JTAG soft enable key (uses HMAC Downstream mode)
7	Digital Signature peripheral key (uses HMAC Downstream mode)
8	HMAC Upstream mode
9	SECURE_BOOT_DIGEST0 (Secure Boot key digest)
10	SECURE_BOOT_DIGEST1 (Secure Boot key digest)
11	SECURE_BOOT_DIGEST2 (Secure Boot key digest)

Table 56 provides the details on the parameters in BLOCK1 ~ BLOCK10.

Table 56: Parameters in BLOCK1-10

BLOCK	Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Software Read Protection by EFUSE_RD_DIS Bit Number	Description
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC address
	EFUSE_SPI_PAD_CONFIGURE	[0:5]	N	20	N/A	CLK
		[6:11]	N	20	N/A	Q (D1)
		[12:17]	N	20	N/A	D (D0)
		[18:23]	N	20	N/A	CS
		[24:29]	N	20	N/A	HD (D3)
		[30:35]	N	20	N/A	WP (D2)
		[36:41]	N	20	N/A	DQS
		[42:47]	N	20	N/A	D4
		[48:53]	N	20	N/A	D5
		[54:59]	N	20	N/A	D6
		[60:65]	N	20	N/A	D7

BLOCK	Parameters	Bit Width	Hardware Use	Write Protection by EFUSE_WR_DIS Bit Number	Software Read Protection by EFUSE_RD_DIS Bit Number	Description
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	System data
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	System data
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	User data
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	Key0 or user data
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	Key1 or user data
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	Key2 or user data
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	Key3 or user data
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	Key4 or user data
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	Key5 or user data
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	System data

Among these blocks, BLOCK4 ~ 9 stores KEY0 ~ 5, respectively. Up to six 256-bit keys can be programmed into eFuse. Whenever a key is programmed, its purpose value should also be programmed (see table 55). For example, a key for the JTAG function in HMAC Downstream mode is programmed to KEY3 (i.e., BLOCK7), then, the key purpose value 6 should be programmed to EFUSE_KEY_PURPOSE_3.

BLOCK1 ~ BLOCK10 use the RS coding scheme, so there are some restrictions on writing to these parameters (refer to Section 16.3.1.3: *Data Storage* and Section 16.3.2: *Software Programming of Parameters*).

16.3.1.1 EFUSE_WR_DIS

Parameter EFUSE_WR_DIS determines whether individual eFuse parameters are write-protected. After burning EFUSE_WR_DIS, execute an eFuse read operation so the new values will take effect (refer to *Updating eFuse read registers* in Section 16.3.3).

The columns "Programming-Protected by EFUSE_WR_DIS" in Table 54 and Table 56 list the specific bits of EFUSE_WR_DIS that determine the write protected status of each parameter.

When the corresponding bit is 0, the parameter is not write protected and can be programmed if the parameter has not been programmed.

When the corresponding bit is 1, the parameter is write protected and none of its bits can be modified. The non-programmed bits always remain 0, while programmed bits always remain 1.

16.3.1.2 EFUSE_RD_DIS

Only the eFuse blocks BLCOK4 ~ BLOCK10 can be individually software read protected. The corresponding bit in EFUSE_RD_DIS is shown in Table 56. After burning EFUSE_RD_DIS, execute an eFuse read operation so the new values will take effect (refer to *Updating eFuse read registers* in Section 16.3.3).

If the corresponding EFUSE_RD_DIS bit is 0, then the eFuse block can be read by software. If the corresponding EFUSE_RD_DIS bit is 1, then the parameter controlled by this bit is software read protected.

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

Although BLOCK4 ~ BLOCK10 can be set to read-protected, they can still be used by hardware modules, if the EFUSE_KEY_PURPOSE_*n* bit is set accordingly.

16.3.1.3 Data Storage

Internal to the SoC, eFuses use hardware encoding schemes to protect against data corruption.

All BLOCK0 parameters except for EFUSE_WR_DIS are stored with four backups, meaning each bit is stored four times. This backup scheme is automatically applied by the hardware and is not visible to software.

BLOCK1 ~ BLOCK10 use RS (44, 32) coding scheme that supports up to 5 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 that generates the check code is shown in Figure 16-1, where gf_mul_n (n is an integer) is the result of multiplying a byte of data in the $GF(2^8)$ field by the element α^n.

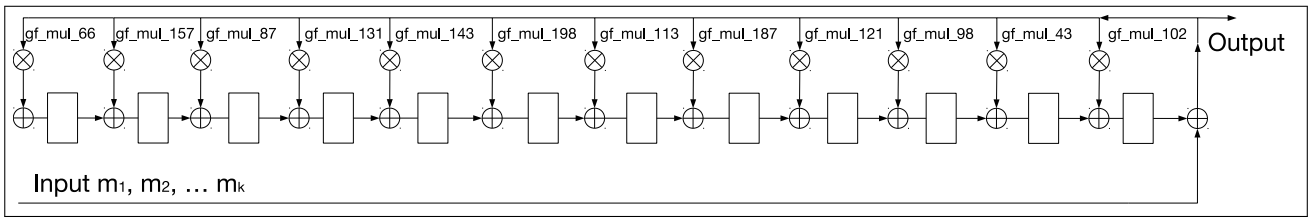


Figure 16-1. Shift Register Circuit

Software must encode the 32-byte parameter using RS (44, 32) to generate a 12-byte check code, and then burn the parameter and the check code into eFuse at the same time. The eFuse Controller automatically decodes the RS encoding and applies error correction when reading back the eFuse block.

Because the RS check codes are generated across the entire 256-bit eFuse block, each block can only be written to one time.

16.3.2 Software Programming of Parameters

The eFuse controller can only write to eFuse parameters in one block at a time. BLOCK0 ~ BLOCK10 share the same registers to store the parameters to be programmed. Configure parameter EFUSE_BLK_NUM to indicate which block is to be programmed.

Programming BLOCK0

When EFUSE_BLK_NUM = 0, BLOCK0 is programmed. Register EFUSE_PGM_DATA0_REG stores EFUSE_WR_DIS. EFUSE_PGM_DATA1_REG ~ EFUSE_PGM_DATA5_REG store the information of new BLOCK0 parameters to be programmed. Note that 24 BLOCK0 bits are reserved and can only be used by hardware. These must always be set to 0 in the programming registers. The specific bits are:

- EFUSE_PGM_DATA1_REG[29:31]
- EFUSE_PGM_DATA1_REG[20:23]
- EFUSE_PGM_DATA2_REG[7:15]
- EFUSE_PGM_DATA2_REG[0:3]

- EFUSE_PGM_DATA3_REG[16:19]

Values written to EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG are ignored when programming BLOCK0.

Programming BLOCK1

When EFUSE_BLK_NUM = 1, EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG store the parameters to be programmed. EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_DATA2_REG store the corresponding RS check codes. Values written to EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG are ignored when programming BLOCK1, the RS check codes should be calculated as if these bits were all 0.

Programming BLOCK2 ~ 10

When EFUSE_BLK_NUM = 2 ~ 10, EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG store the parameters to be programmed to this block. EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG store the corresponding RS check codes.

Programming process

The process of programming parameters is as follows:

1. Set EFUSE_BLK_NUM parameter as described above.
2. Write the parameters to be programmed into registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
3. Ensure the eFuse clock registers are set correctly as described in Section 16.3.4.1: *eFuse-Programming Timing*.
4. Ensure the eFuse programming voltage VDDQ is set correctly as described in Section 16.3.4.2: *eFuse VDDQ Setting*.
5. Set EFUSE_OP_CODE field in register EFUSE_CONF_REG to 0x5A5A.
6. Set EFUSE_PGM_CMD field in register EFUSE_CMD_REG to 1.
7. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a pgm_done interrupt. Information on how to identify a pgm/read_done interrupt is provided at the end of Section 16.3.3.
8. Clear the parameters written into the register.
9. Trigger an eFuse read operation (see Section 16.3.3: *Software Reading of Parameters*) to update eFuse registers with the new values.

Limitations

For BLOCK0, the programming of different parameters and even the programming of different bits of the same parameter does not need to be done at once. It is, however, recommended that users minimize programming cycles and program all the bits of a parameter in one programming action. In addition, after all parameters controlled by a certain bit of EFUSE_WR_DIS are programmed, that bit should be immediately programmed. The programming of parameters controlled by a certain bit of EFUSE_WR_DIS, and the programming of the bit itself can even be completed at the same time. Repeated programming of already programmed bits is strictly forbidden, otherwise, programming errors will occur.

BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.

BLOCK2 ~ 10 can only be programmed once. Repeated programming is not allowed.

16.3.3 Software Reading of Parameters

Software cannot read eFuse bits directly. The eFuse Controller hardware reads all eFuse bits and stores the results to their corresponding registers in the memory space. Then, software can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in the table below.

BLOCK	Read Registers	When Programming This Block
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SPI_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_DATA_PART1_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_DATA_PART2_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating eFuse read registers

The eFuse Controller hardware populates the read registers from the internal eFuse storage. This read operation happens on system reset and can also be triggered manually by software as needed, for example if new eFuse values have been programmed.

The process of triggering an eFuse controller read from software is as follows:

1. Configure the eFuse read timing registers as described in Section 16.3.4.3: *eFuse-Read Timing*.
2. Set the EFUSE_OP_CODE field in register EFUSE_CONF_REG to 0x5AA5.
3. Set the EFUSE_READ_CMD field in register EFUSE_CMD_REG to 1.
4. Poll the EFUSE_CMD_REG register until it is 0x0, or wait for a read_done interrupt. Information on how to identify a pgm/read_done interrupt is provided below.
5. Software reads the values of each parameter from memory.

The eFuse read registers will now hold updated values for all eFuse parameters.

Error detection

Error registers allow software to detect an inconsistency in the stored eFuses.

EFUSE_RD_REPEAT_ERR0 ~ 3_REG indicate inconsistencies in the stored backup copies of the parameters in BLOCK0 (except for EFUSE_WR_DIS) . Value 1 indicates an error was detected, and the bit became invalid. Value 0 indicates no error.

Registers EFUSE_RD_RS_ERR0 ~ 1_REG store the number of corrected bytes as well as the result of RS decoding during eFuse reading BLOCK1 ~ BLOCK10.

Software can read the values of above registers only after the eFuse read registers have been updated.

Identifying program/read operation completion

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

- Method one:

1. Poll bit 1/0 in register EFUSE_INT_RAW_REG until it is 1, which represents the completion of a program/read operation.

- Method two:

1. Set bit 1/0 in register EFUSE_INT_ENA_REG to 1 to enable eFuse Controller to post a pgm/read_done interrupt.
2. Configure Interrupt Matrix to enable the CPU to respond to eFuse interrupt signal.
3. Wait for the pgm/read_done interrupt.
4. Set the bit 1/0 in register EFUSE_INT_CLR_REG to 1 to clear the pgm/read_done interrupt.

16.3.4 Timing

16.3.4.1 eFuse-Programming Timing

Figure 16-2 shows the timing for programming eFuse. Four registers EFUSE_TSUP_A, EFUSE_TPGM, EFUSE_THP_A, and EFUSE_TPGM_INACTIVE are used to configure the timing. Terms used in the timing diagrams in this section are described as follows:

- CSB: Chip select, active low
- VDDQ: eFuse programming voltage
- PGENB: eFuse programming enable signal, active low

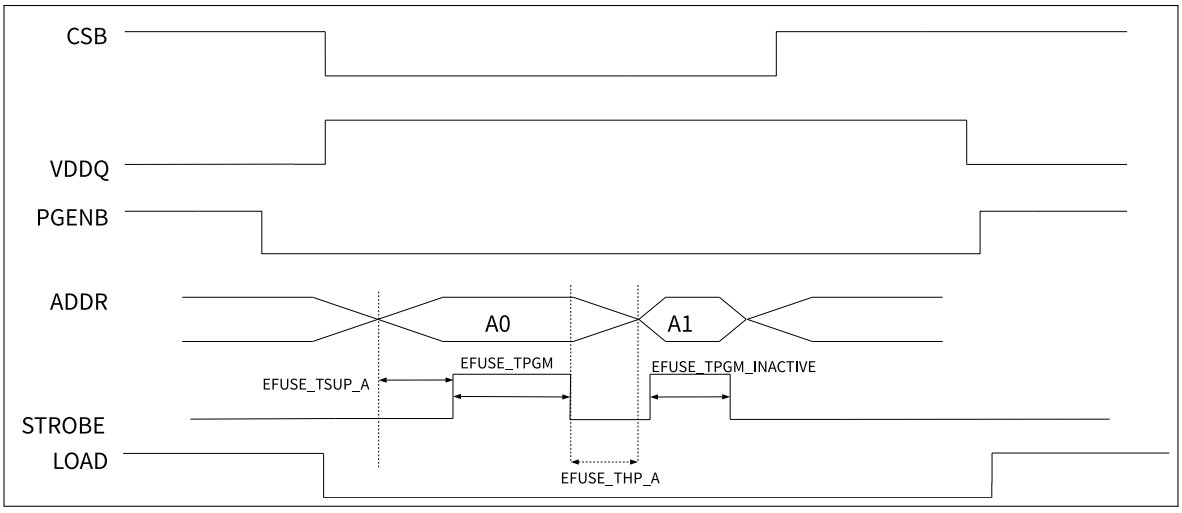


Figure 16-2. eFuse-Programming Timing Diagram

The eFuse block uses the CLK_APB clock, which is configurable. Therefore, the timing parameters should be configured according to the specific clock frequency. After reset, the initial parameters are based on 20 MHz clock frequency.

Table 58: Configuration of eFuse-Programming Timing Parameters

APB Frequency	EFUSE_TSUP_A (> 6.669 ns)	EFUSE_TPGM (9-11 μs, usually 10 μs)	EFUSE_THP_A (> 6.166 ns)	EFUSE_TPGM_INACTIVE (> 35.96 ns)
80 MHz	0x2	0x320	0x2	0x4
40 MHz	0x1	0x190	0x1	0x2
20 MHz	0x1	0xC8	0x1	0x1

In Figure 16-2, Address A0 is programmed, then the corresponding eFuse bit is 1; Address A1 is not programmed, then the corresponding eFuse bit is 0.

16.3.4.2 eFuse VDDQ Timing Setting

VDDQ is the eFuse programming voltage, and its timing parameters should be configured according to the APB clock frequency:

Table 59: Configuration of VDDQ Timing Parameters

APB Frequency	EFUSE_DAC_CLK_DIV (> 1 μs)	EFUSE_PWR_ON_NUM (> EFUSE_DAC_CLK_DIV*255)	EFUSE_PWR_OFF_NUM (> 3 μs)
80 MHz	0xA0	0xA200	0x100
40 MHz	0x50	0x5100	0x80
20 MHz	0x28	0x2880	0x40

16.3.4.3 eFuse-Read Timing

Figure 16-3 shows the timing for reading eFuse. Three registers EFUSE_TSUR_A, EFUSE_TRD, and EFUSE_THR_A are used to configure the timing.

The parameters should be configured according to the specific APB clock frequency. Details can be found in the table below.

Table 60: Configuration of eFuse-Reading Parameters

APB Frequency	EFUSE_TSUR_A (> 6.669 ns)	EFUSE_TRD (> 35.96 ns)	EFUSE_THR_A (> 6.166 ns)
80 MHz	0x2	0x4	0x2
40 MHz	0x1	0x2	0x1
20 MHz	0x1	0x1	0x1

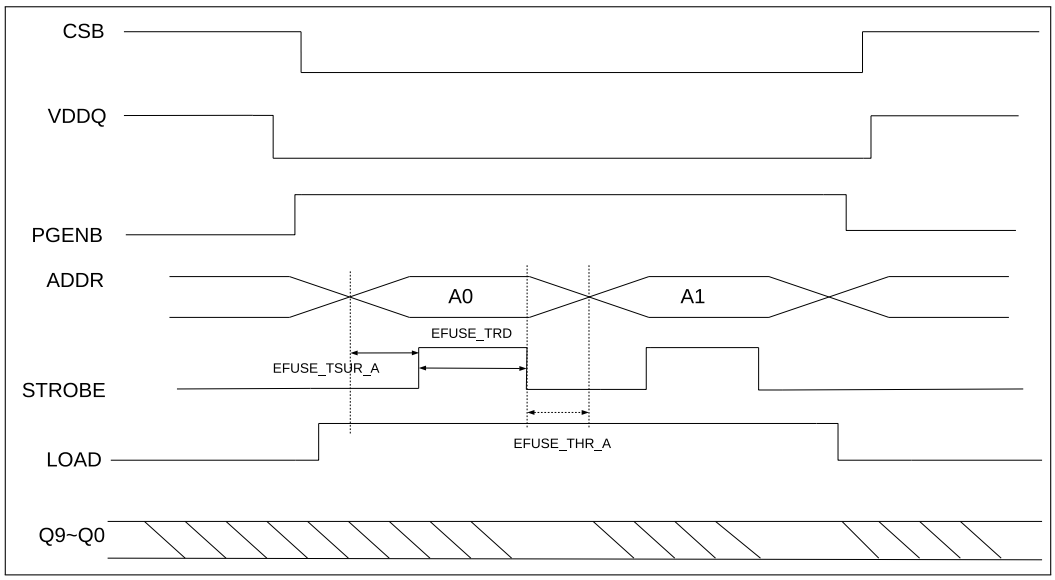


Figure 16-3. Timing Diagram for Reading eFuse

16.3.5 The Use of Parameters by Hardware Modules

Hardware modules are directly hardwired to the ESP32-S2 in order to use the parameters listed in Table 54 and 56, specifically those marked with "Y" in columns "Hardware Use". Software cannot change this behavior.

16.3.6 Interrupts

- pgm_done interrupt: Triggered when eFuse programming has finished. To enable this interrupt, set EFUSE_PGM_DONE_INT_ENA to 1.
- read_done interrupt: Triggered when eFuse reading has finished. To enable this interrupt, set EFUSE_READ_DONE_INT_ENA to 1.

16.4 Base Address

Users can access the eFuse Controller with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 61: eFuse Controller Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x6001A000
PeriBUS2	0x3FC1A000

16.5 Register Summary

The addresses in the following table are relative to the eFuse base addresses provided in Section 16.4.

Name	Description	Address	Access
PGM Data Registers			
EFUSE_PGM_DATA0_REG	Register 0 that stores data to be programmed.	0x0000	R/W
EFUSE_PGM_DATA1_REG	Register 1 that stores data to be programmed.	0x0004	R/W

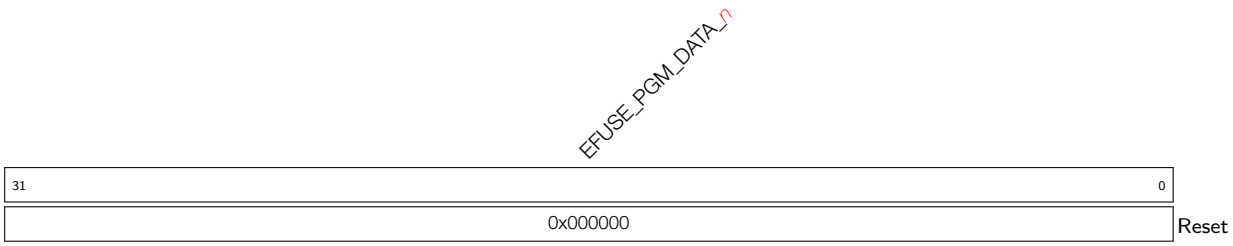
Name	Description	Address	Access
EFUSE_PGM_DATA2_REG	Register 2 that stores data to be programmed.	0x0008	R/W
EFUSE_PGM_DATA3_REG	Register 3 that stores data to be programmed.	0x000C	R/W
EFUSE_PGM_DATA4_REG	Register 4 that stores data to be programmed.	0x0010	R/W
EFUSE_PGM_DATA5_REG	Register 5 that stores data to be programmed.	0x0014	R/W
EFUSE_PGM_DATA6_REG	Register 6 that stores data to be programmed.	0x0018	R/W
EFUSE_PGM_DATA7_REG	Register 7 that stores data to be programmed.	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Register 0 that stores the RS code to be programmed.	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Register 1 that stores the RS code to be programmed.	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Register 2 that stores the RS code to be programmed.	0x0028	R/W
Read Data Registers			
EFUSE_RD_WR_DIS_REG	Register 0 of BLOCK0.	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	Register 1 of BLOCK0.	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	Register 2 of BLOCK0.	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	Register 3 of BLOCK0.	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	Register 4 of BLOCK0.	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	Register 5 of BLOCK0.	0x0040	RO
EFUSE_RD_MAC_SPI_SYS_0_REG	Register 0 of BLOCK1.	0x0044	RO
EFUSE_RD_MAC_SPI_SYS_1_REG	Register 1 of BLOCK1.	0x0048	RO
EFUSE_RD_MAC_SPI_SYS_2_REG	Register 2 of BLOCK1.	0x004C	RO
EFUSE_RD_MAC_SPI_SYS_3_REG	Register 3 of BLOCK1.	0x0050	RO
EFUSE_RD_MAC_SPI_SYS_4_REG	Register 4 of BLOCK1.	0x0054	RO
EFUSE_RD_MAC_SPI_SYS_5_REG	Register 5 of BLOCK1.	0x0058	RO
EFUSE_RD_SYS_DATA_PART1_0_REG	Register 0 of BLOCK2 (system).	0x005C	RO
EFUSE_RD_SYS_DATA_PART1_1_REG	Register 1 of BLOCK2 (system).	0x0060	RO
EFUSE_RD_SYS_DATA_PART1_2_REG	Register 2 of BLOCK2 (system).	0x0064	RO
EFUSE_RD_SYS_DATA_PART1_3_REG	Register 3 of BLOCK2 (system).	0x0068	RO
EFUSE_RD_SYS_DATA_PART1_4_REG	Register 4 of BLOCK2 (system).	0x006C	RO
EFUSE_RD_SYS_DATA_PART1_5_REG	Register 5 of BLOCK2 (system).	0x0070	RO
EFUSE_RD_SYS_DATA_PART1_6_REG	Register 6 of BLOCK2 (system).	0x0074	RO
EFUSE_RD_SYS_DATA_PART1_7_REG	Register 7 of BLOCK2 (system).	0x0078	RO
EFUSE_RD_USR_DATA0_REG	Register 0 of BLOCK3 (user).	0x007C	RO
EFUSE_RD_USR_DATA1_REG	Register 1 of BLOCK3 (user).	0x0080	RO
EFUSE_RD_USR_DATA2_REG	Register 2 of BLOCK3 (user).	0x0084	RO
EFUSE_RD_USR_DATA3_REG	Register 3 of BLOCK3 (user).	0x0088	RO
EFUSE_RD_USR_DATA4_REG	Register 4 of BLOCK3 (user).	0x008C	RO
EFUSE_RD_USR_DATA5_REG	Register 5 of BLOCK3 (user).	0x0090	RO
EFUSE_RD_USR_DATA6_REG	Register 6 of BLOCK3 (user).	0x0094	RO
EFUSE_RD_USR_DATA7_REG	Register 7 of BLOCK3 (user).	0x0098	RO
EFUSE_RD_KEY0_DATA0_REG	Register 0 of BLOCK4 (KEY0).	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	Register 1 of BLOCK4 (KEY0).	0x00A0	RO

Name	Description	Address	Access
EFUSE_RD_KEY0_DATA2_REG	Register 2 of BLOCK4 (KEY0).	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	Register 3 of BLOCK4 (KEY0).	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	Register 4 of BLOCK4 (KEY0).	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	Register 5 of BLOCK4 (KEY0).	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	Register 6 of BLOCK4 (KEY0).	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	Register 7 of BLOCK4 (KEY0).	0x00B8	RO
EFUSE_RD_KEY1_DATA0_REG	Register 0 of BLOCK5 (KEY1).	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	Register 1 of BLOCK5 (KEY1).	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	Register 2 of BLOCK5 (KEY1).	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	Register 3 of BLOCK5 (KEY1).	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	Register 4 of BLOCK5 (KEY1).	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	Register 5 of BLOCK5 (KEY1).	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	Register 6 of BLOCK5 (KEY1).	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	Register 7 of BLOCK5 (KEY1).	0x00D8	RO
EFUSE_RD_KEY2_DATA0_REG	Register 0 of BLOCK6 (KEY2).	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	Register 1 of BLOCK6 (KEY2).	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	Register 2 of BLOCK6 (KEY2).	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	Register 3 of BLOCK6 (KEY2).	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	Register 4 of BLOCK6 (KEY2).	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	Register 5 of BLOCK6 (KEY2).	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	Register 6 of BLOCK6 (KEY2).	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	Register 7 of BLOCK6 (KEY2).	0x00F8	RO
EFUSE_RD_KEY3_DATA0_REG	Register 0 of BLOCK7 (KEY3).	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	Register 1 of BLOCK7 (KEY3).	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	Register 2 of BLOCK7 (KEY3).	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	Register 3 of BLOCK7 (KEY3).	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	Register 4 of BLOCK7 (KEY3).	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	Register 5 of BLOCK7 (KEY3).	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	Register 6 of BLOCK7 (KEY3).	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	Register 7 of BLOCK7 (KEY3).	0x0118	RO
EFUSE_RD_KEY4_DATA0_REG	Register 0 of BLOCK8 (KEY4).	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	Register 1 of BLOCK8 (KEY4).	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	Register 2 of BLOCK8 (KEY4).	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	Register 3 of BLOCK8 (KEY4).	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	Register 4 of BLOCK8 (KEY4).	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	Register 5 of BLOCK8 (KEY4).	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Register 6 of BLOCK8 (KEY4).	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Register 7 of BLOCK8 (KEY4).	0x0138	RO
EFUSE_RD_KEY5_DATA0_REG	Register 0 of BLOCK9 (KEY5).	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	Register 1 of BLOCK9 (KEY5).	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	Register 2 of BLOCK9 (KEY5).	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	Register 3 of BLOCK9 (KEY5).	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	Register 4 of BLOCK9 (KEY5).	0x014C	RO

Name	Description	Address	Access
EFUSE_RD_KEY5_DATA5_REG	Register 5 of BLOCK9 (KEY5).	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	Register 6 of BLOCK9 (KEY5).	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	Register 7 of BLOCK9 (KEY5).	0x0158	RO
EFUSE_RD_SYS_DATA_PART2_0_REG	Register 0 of BLOCK10 (system).	0x015C	RO
EFUSE_RD_SYS_DATA_PART2_1_REG	Register 1 of BLOCK10 (system).	0x0160	RO
EFUSE_RD_SYS_DATA_PART2_2_REG	Register 2 of BLOCK10 (system).	0x0164	RO
EFUSE_RD_SYS_DATA_PART2_3_REG	Register 3 of BLOCK10 (system).	0x0168	RO
EFUSE_RD_SYS_DATA_PART2_4_REG	Register 4 of BLOCK10 (system).	0x016C	RO
EFUSE_RD_SYS_DATA_PART2_5_REG	Register 5 of BLOCK10 (system).	0x0170	RO
EFUSE_RD_SYS_DATA_PART2_6_REG	Register 6 of BLOCK10 (system).	0x0174	RO
EFUSE_RD_SYS_DATA_PART2_7_REG	Register 7 of BLOCK10 (system).	0x0178	RO
Error Status Registers			
EFUSE_RD_REPEAT_ERR0_REG	Programming error record register 0 of BLOCK0.	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	Programming error record register 1 of BLOCK0.	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	Programming error record register 2 of BLOCK0.	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	Programming error record register 3 of BLOCK0.	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	Programming error record register 4 of BLOCK0.	0x0190	RO
EFUSE_RD_RS_ERR0_REG	Programming error record register 0 of BLOCK1-10.	0x01C0	RO
EFUSE_RD_RS_ERR1_REG	Programming error record register 1 of BLOCK1-10.	0x01C4	RO
Control/Status Registers			
EFUSE_CLK_REG	eFuse clock configuration register.	0x01C8	R/W
EFUSE_CONF_REG	eFuse operation mode configuration register.	0x01CC	R/W
EFUSE_CMD_REG	eFuse command register.	0x01D4	R/W
EFUSE_DAC_CONF_REG	Controls the eFuse programming voltage.	0x01E8	R/W
EFUSE_STATUS_REG	eFuse status register.	0x01D0	RO
Interrupt Registers			
EFUSE_INT_RAW_REG	eFuse raw interrupt register.	0x01D8	RO
EFUSE_INT_ST_REG	eFuse interrupt status register.	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse interrupt enable register.	0x01E0	R/W
EFUSE_INT_CLR_REG	eFuse interrupt clear register.	0x01E4	WO
Configuration Registers			
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters.	0x01EC	R/W
EFUSE_WR_TIM_CONF0_REG	Configuration register 0 of eFuse programming timing parameters.	0x01F0	R/W
EFUSE_WR_TIM_CONF1_REG	Configuration register 1 of eFuse programming timing parameters.	0x01F4	R/W
EFUSE_WR_TIM_CONF2_REG	Configuration register 2 of eFuse programming timing parameters.	0x01F8	R/W
Version Register			
EFUSE_DATE_REG	Version control register.	0x01FC	R/W

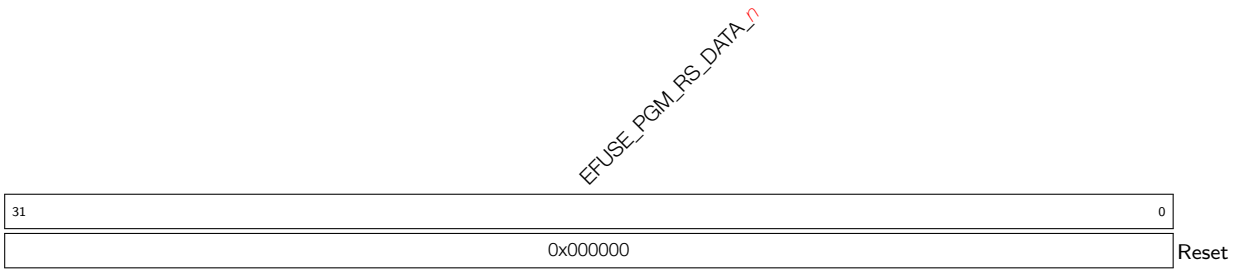
16.6 Registers

Register 16.1: EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+4n*)**



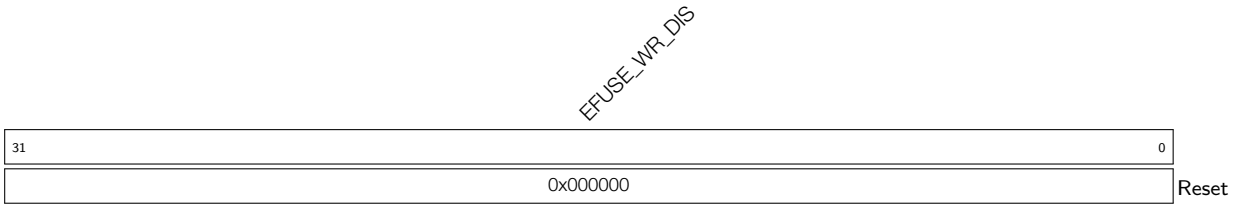
EFUSE_PGM_DATA_*n* The content of the *n*th 32-bit data to be programmed. (R/W)

Register 16.2: EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+4n*)**



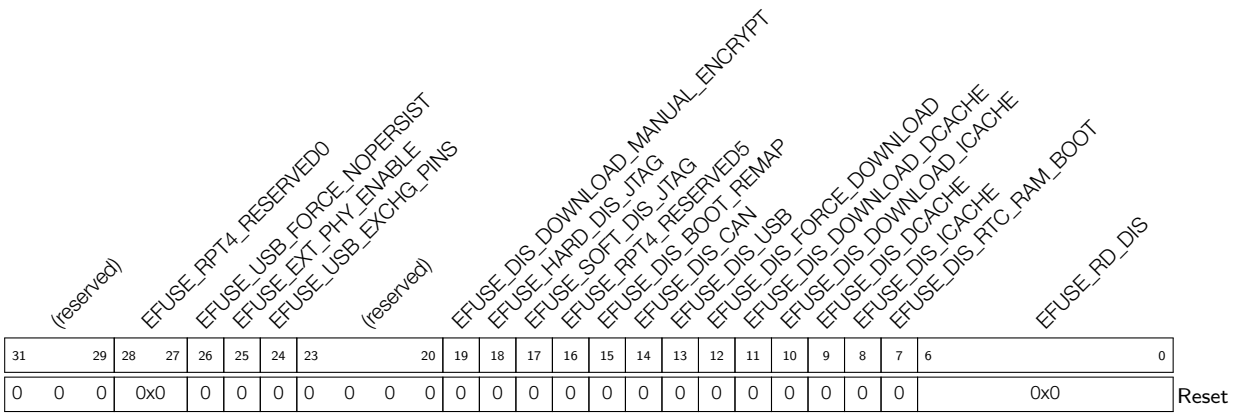
EFUSE_PGM_RS_DATA_*n* The content of the *n*th 32-bit RS code to be programmed. (R/W)

Register 16.3: EFUSE_RD_WR_DIS_REG (0x002C)



EFUSE_WR_DIS Disables programming of individual eFuses. (RO)

Register 16.4: EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS Disables software reading from individual eFuse blocks (BLOCK4-10). (RO)

EFUSE_DIS_RTC_RAM_BOOT Set this bit to disable boot from RTC RAM. (RO)

EFUSE_DIS_ICACHE Set this bit to disable ICache. (RO)

EFUSE_DIS_DCACHE Set this bit to disable DCache. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE Disables Icache when SoC is in Download mode. (RO)

EFUSE_DIS_DOWNLOAD_DCACHE Disables Dcache when SoC is in Download mode. (RO)

EFUSE_DIS_FORCE_DOWNLOAD Set this bit to disable the function that forces chip into download mode. (RO)

EFUSE_DIS_USB Set this bit to disable USB OTG function. (RO)

EFUSE_DIS_CAN Set this bit to disable the TWAI Controller function. (RO)

EFUSE_DIS_BOOT_REMAP Disables capability to Remap RAM to ROM address space. (RO)

EFUSE_RPT4_RESERVED5 Reserved (used for four backups method). (RO)

EFUSE_SOFT_DIS_JTAG Software disables JTAG. When software disabled, JTAG can be activated temporarily by HMAC peripheral. (RO)

EFUSE_HARD_DIS_JTAG Hardware disables JTAG permanently. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Disables flash encryption when in download boot modes. (RO)

EFUSE_USB_EXCHG_PINS Set this bit to exchange USB D+ and D- pins. (RO)

EFUSE_EXT_PHY_ENABLE Set this bit to enable external USB PHY. (RO)

EFUSE_USB_FORCE_NOPERSIST If set, forces USB BVALID to 1. (RO)

EFUSE_RPT4_RESERVED0 Reserved (used for four backups method). (RO)

Register 16.5: EFUSE_RD_REPEAT_DATA1_REG (0x0034)

EFUSE_VDD_SPI_XPD If VDD_SPI_FORCE is 1, this value determines if the VDD_SPI regulator is powered on. (RO)

EFUSE_VDD_SPI_TIEH If VDD_SPI_FORCE is 1, determines VDD_SPI voltage. 0: VDD_SPI connects to 1.8 V LDO; 1: VDD_SPI connects to VDD_RTC_IO. (RO)

EFUSE_VDD_SPI_FORCE Set this bit to use XPD_VDD_PSI_REG and VDD_SPI_TIEH to configure VDD_SPI LDO. (RO)

EFUSE_WDT_DELAY_SEL Selects RTC watchdog timeout threshold at startup. 0: 40,000 slow clock cycles; 1: 80,000 slow clock cycles; 2: 160,000 slow clock cycles; 3: 320,000 slow clock cycles. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT Enables encryption and decryption, when an SPI boot mode is set. Feature is enabled 1 or 3 bits are set in the eFuse, disabled otherwise. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0 If set, revokes use of secure boot key digest 0. (RO)

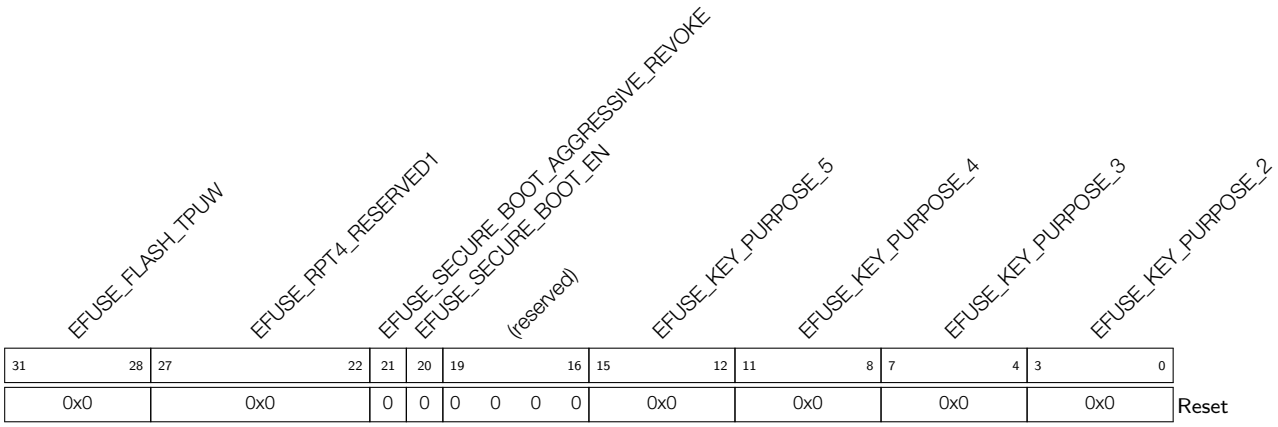
EFUSE_SECURE_BOOT_KEY_REVOKE1 If set, revokes use of secure boot key digest 1. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 If set, revokes use of secure boot key digest 2. (RO)

EFUSE_KEY_PURPOSE_0 Purpose of KEY0. Refer to Table 55 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_1 Purpose of KEY1. Refer to Table 55 *Key Purpose Values*. (RO)

Register 16.6: EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 Purpose of KEY2. Refer to Table 55 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_3 Purpose of KEY3. Refer to Table 55 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_4 Purpose of KEY4. Refer to Table 55 *Key Purpose Values*. (RO)

EFUSE_KEY_PURPOSE_5 Purpose of KEY5. Refer to Table 55 *Key Purpose Values*. (RO)

EFUSE_SECURE_BOOT_EN Set this bit to enable secure boot. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE Set this bit to enable aggressive secure boot key revocation mode. (RO)

EFUSE_RPT4_RESERVED1 Reserved (used for four backups method). (RO)

EFUSE_FLASH_TPUW Configures flash startup delay after SoC power-up, in unit of (ms/2). When the value is 15, delay is 7.5 ms. (RO)

Register 16.7: EFUSE_RD_REPEAT_DATA3_REG (0x003C)

EFUSE_DIS_DOWNLOAD_MODE Set this bit to disable all download boot modes. (RO)

EFUSE_DIS_LEGACY_SPI_BOOT Set this bit to disable Legacy SPI boot mode. (RO)

EFUSE_UART_PRINT_CHANNEL Selects the default UART for printing boot messages. 0: UART0; 1: UART1. (RO)

EFUSE_RPT4_RESERVED3 Reserved (used for four backups method). (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE Set this bit to disable use of USB OTG in UART download boot mode. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Set this bit to enable secure UART download mode (read/write flash only). (RO)

EFUSE_UART_PRINT_CONTROL Set the default UART boot message output mode.

00: Enabled.

01: Enable when GPIO46 is low at reset.

10: Enable when GPIO46 is high at reset.

11: Disabled. (RO)

EFUSE_PIN_POWER_SELECTION Set default power supply for GPIO33-GPIO37, set when SPI flash is initialized. 0: VDD3P3_CPU; 1: VDD_SPI. (RO)

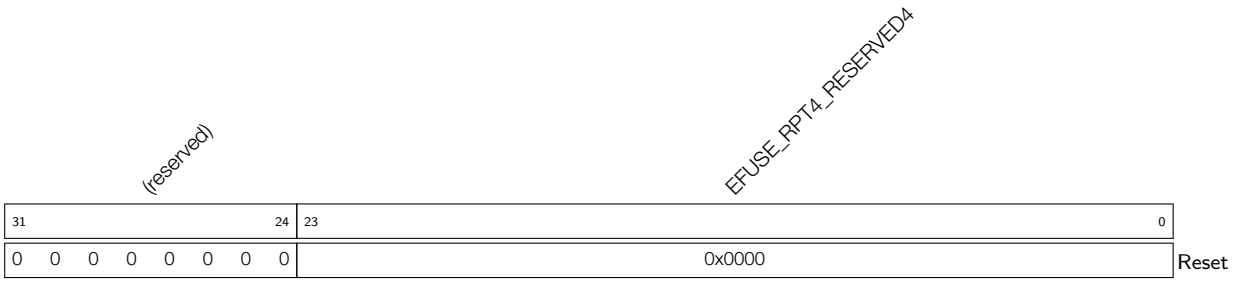
EFUSE_FLASH_TYPE SPI flash type. 0: maximum four data lines, 1: eight data lines. (RO)

EFUSE_FORCE_SEND_RESUME If set, forces ROM code to send an SPI flash resume command during SPI boot. (RO)

EFUSE_SECURE_VERSION Secure version (used by ESP-IDF anti-rollback feature). (RO)

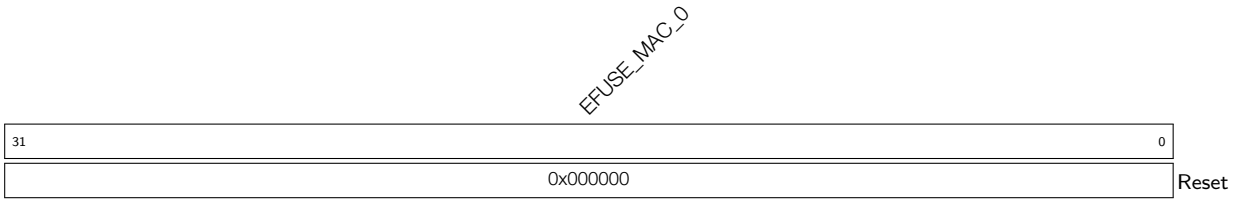
EFUSE_RPT4_RESERVED2 Reserved (used for four backups method). (RO)

Register 16.8: EFUSE_RD_REPEAT_DATA4_REG (0x0040)



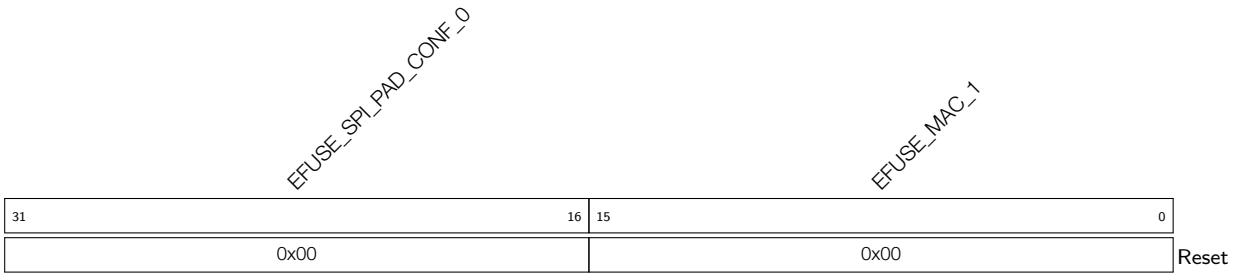
EFUSE_RPT4_RESERVED4 Reserved (used for four backups method). (RO)

Register 16.9: EFUSE_RD_MAC_SPI_SYS_0_REG (0x0044)



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

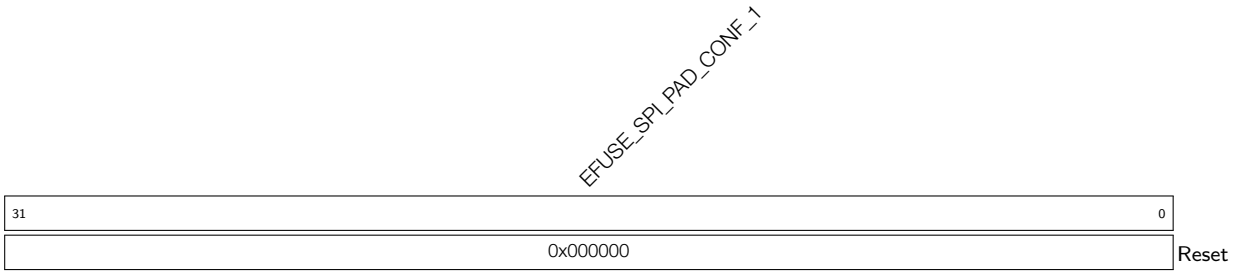
Register 16.10: EFUSE_RD_MAC_SPI_SYS_1_REG (0x0048)



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

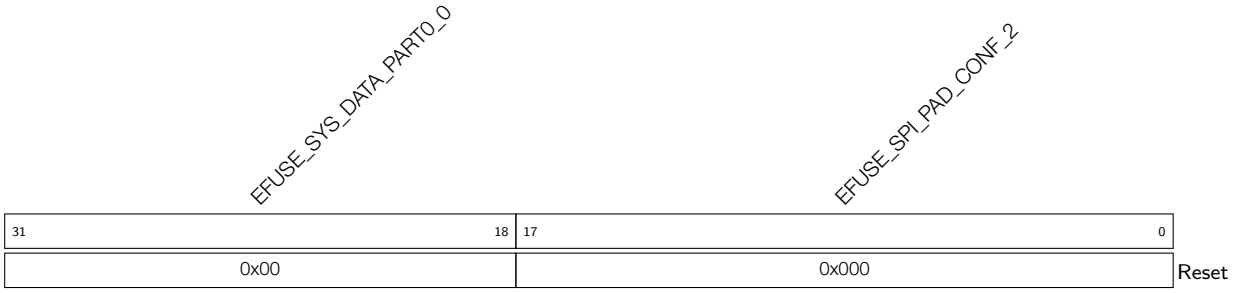
EFUSE_SPI_PAD_CONF_0 Stores the zeroth part of SPI_PAD_CONF. (RO)

Register 16.11: EFUSE_RD_MAC_SPI_SYS_2_REG (0x004C)



EFUSE_SPI_PAD_CONF_1 Stores the first part of SPI_PAD_CONF. (RO)

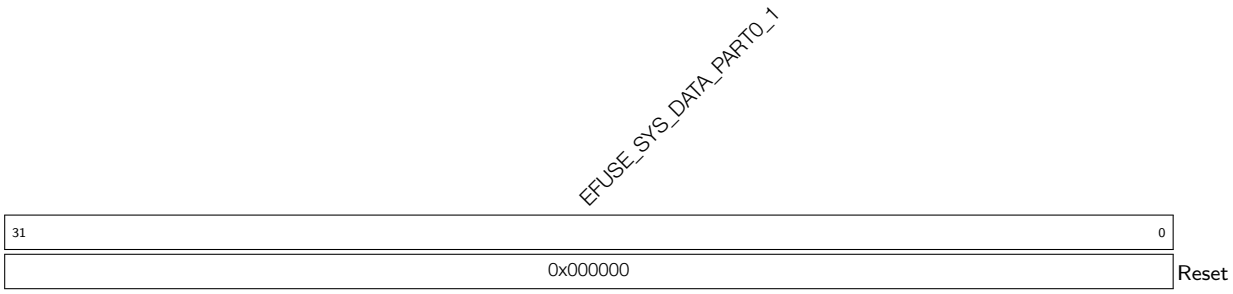
Register 16.12: EFUSE_RD_MAC_SPI_SYS_3_REG (0x0050)



EFUSE_SPI_PAD_CONF_2 Stores the second part of SPI_PAD_CONF. (RO)

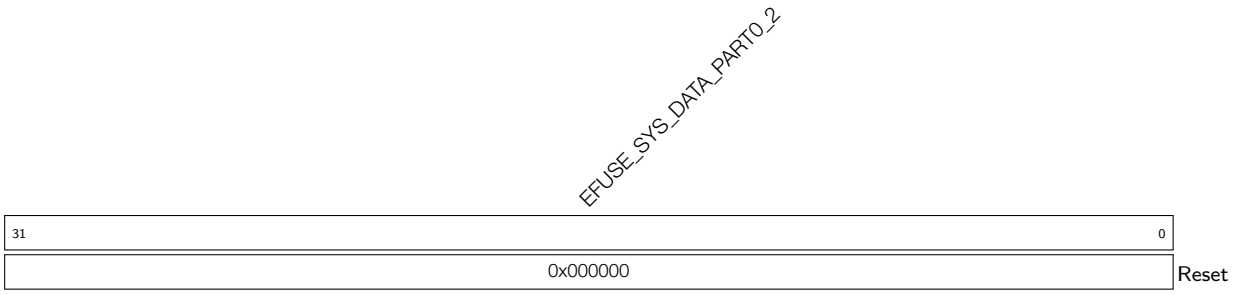
EFUSE_SYS_DATA_PART0_0 Stores the zeroth part of the zeroth part of system data. (RO)

Register 16.13: EFUSE_RD_MAC_SPI_SYS_4_REG (0x0054)



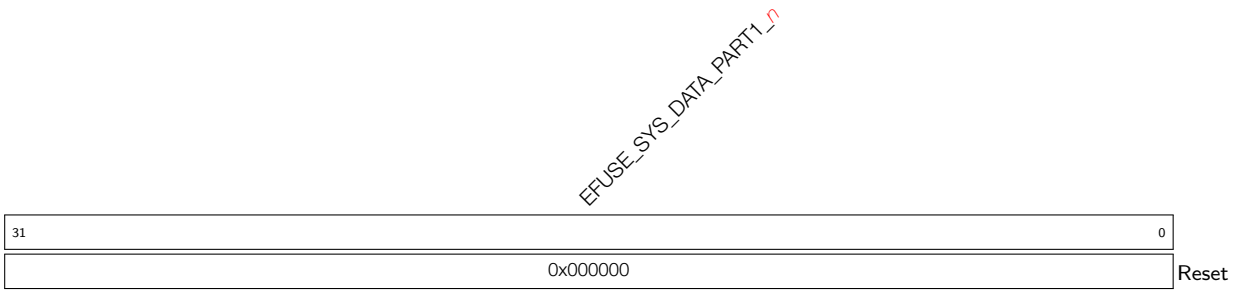
EFUSE_SYS_DATA_PART0_1 Stores the fist part of the zeroth part of system data. (RO)

Register 16.14: EFUSE_RD_MAC_SPI_SYS_5_REG (0x0058)



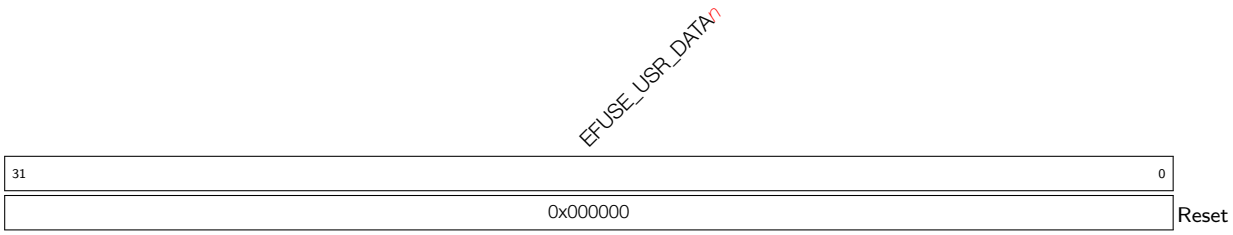
EFUSE_SYS_DATA_PART0_2 Stores the second part of the zeroth part of system data. (RO)

Register 16.15: EFUSE_RD_SYS_DATA_PART1_*n*_REG (*n*: 0-7) (0x005C+4n*)**



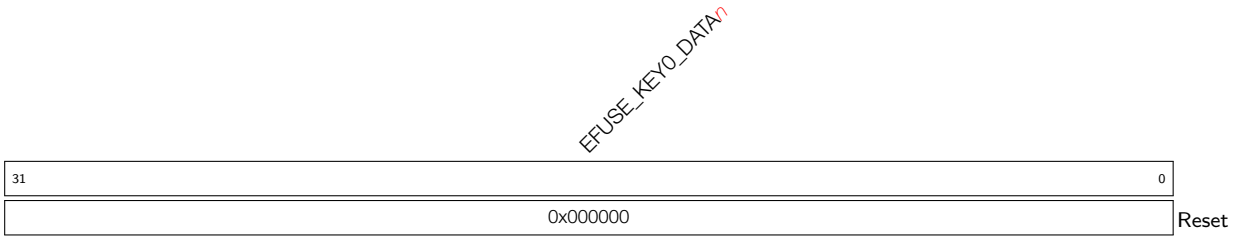
EFUSE_SYS_DATA_PART1_*n* Stores the *n*th 32 bits of the first part of system data. (RO)

Register 16.16: EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+4n*)**



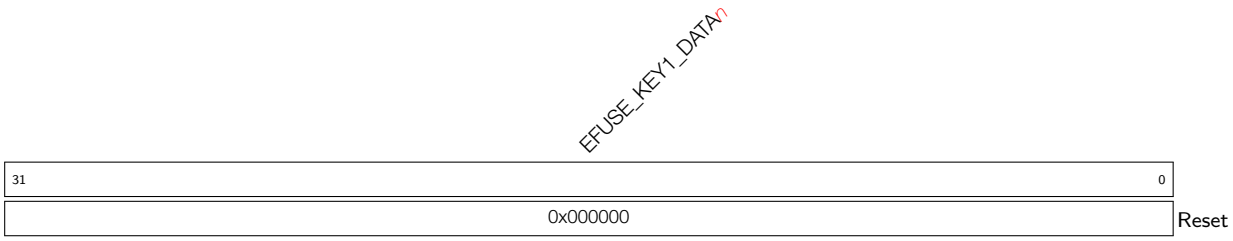
EFUSE_USR_DATA*n* Stores the *n*th 32 bits of BLOCK3 (user). (RO)

Register 16.17: EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+4n*)**



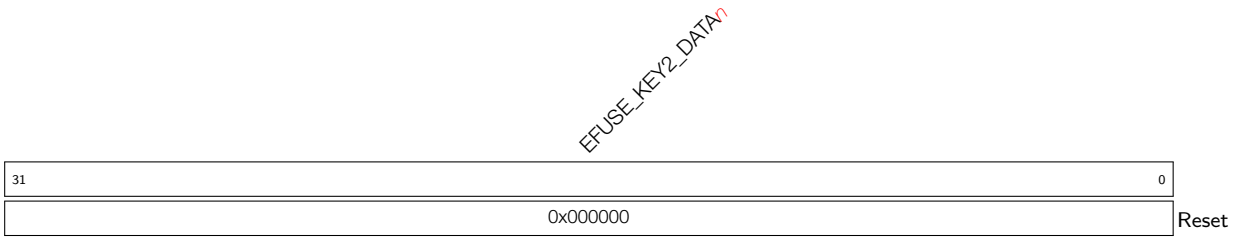
EFUSE_KEY0_DATA*n* Stores the *n*th 32 bits of KEY0. (RO)

Register 16.18: EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+4n*)**



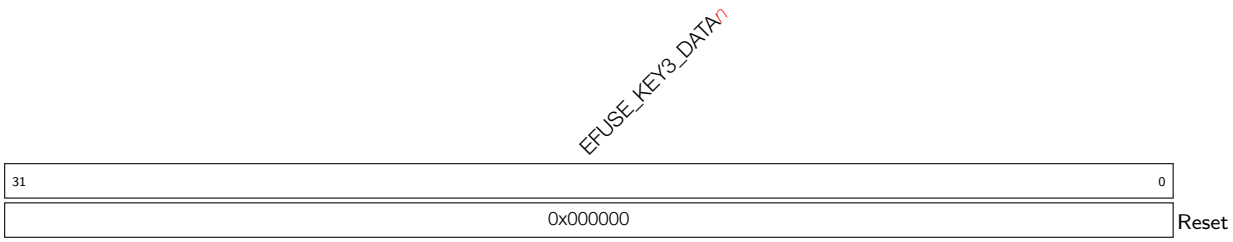
EFUSE_KEY1_DATA*n* Stores the *n*th 32 bits of KEY1. (RO)

Register 16.19: EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+4n*)**



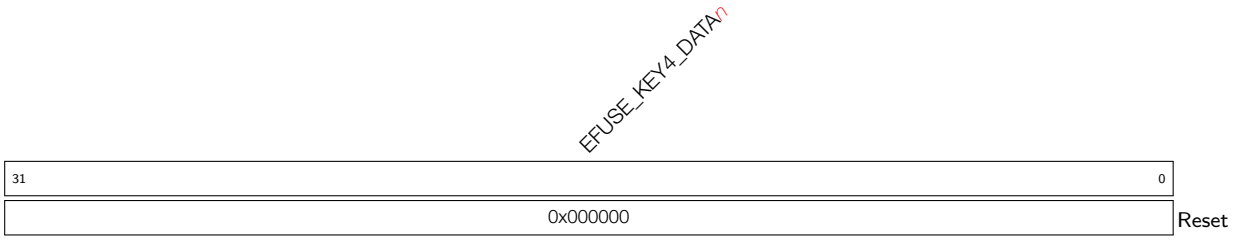
EFUSE_KEY2_DATA*n* Stores the *n*th 32 bits of KEY2. (RO)

Register 16.20: EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+4n*)**



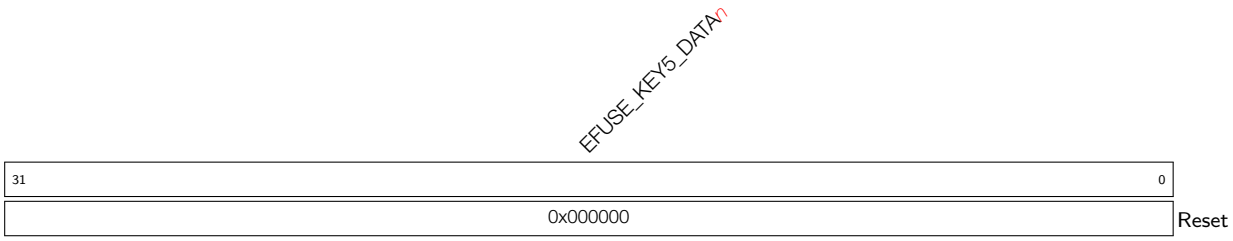
EFUSE_KEY3_DATA*n* Stores the *n*th 32 bits of KEY3. (RO)

Register 16.21: EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+4n*)**



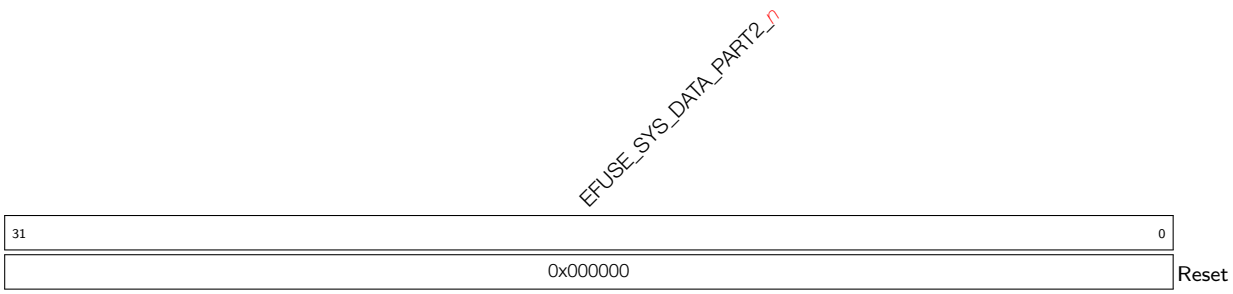
EFUSE_KEY4_DATA*n* Stores the *n*th 32 bits of KEY4. (RO)

Register 16.22: EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+4n*)**



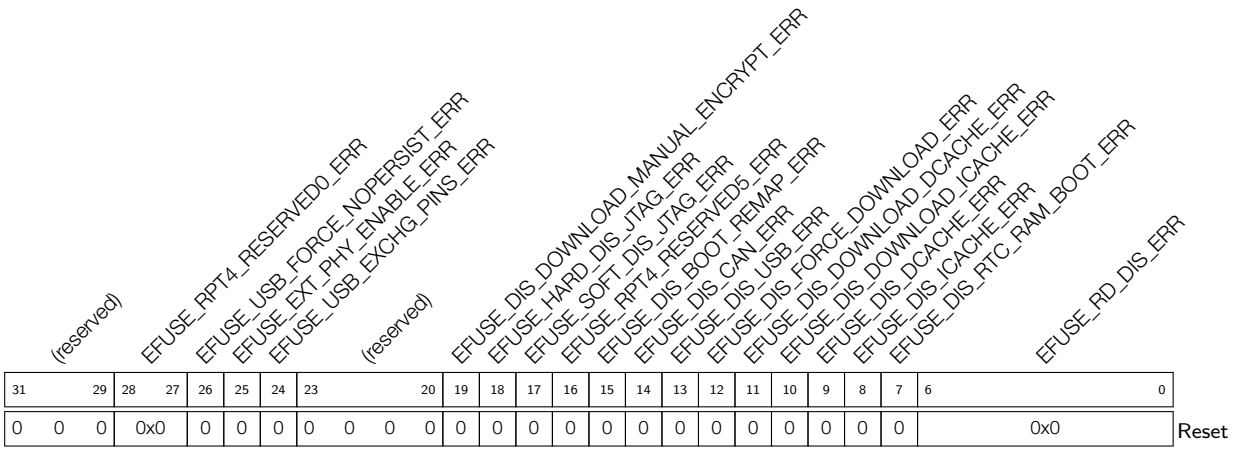
EFUSE_KEY5_DATA*n* Stores the *n*th 32 bits of KEY5. (RO)

Register 16.23: EFUSE_RD_SYS_DATA_PART2_*n*_REG (*n*: 0-7) (0x015C+4n*)**



EFUSE_SYS_DATA_PART2_*n* Stores the *n*th 32 bits of the 2nd part of system data. (RO)

Register 16.24: EFUSE_RD_REPEAT_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR Any bit equal to 1 denotes a programming error in EFUSE_RD_DIS. (RO)

EFUSE_DIS_RTC_RAM_BOOT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_RTC_RAM_BOOT. (RO)

EFUSE_DIS_ICACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_ICACHE. (RO)

EFUSE_DIS_DCACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DCACHE. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_ICACHE. (RO)

EFUSE_DIS_DOWNLOAD_DCACHE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_DCACHE. (RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_FORCE_DOWNLOAD. (RO)

EFUSE_DIS_USB_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_USB. (RO)

EFUSE_DIS_CAN_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_CAN. (RO)

EFUSE_DIS_BOOT_REMAP_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_BOOT_REMAP. (RO)

EFUSE_RPT4_RESERVED5_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED5. (RO)

EFUSE_SOFT_DIS_JTAG_ERR Any bit equal to 1 denotes a programming error in EFUSE_SOFT_DIS_JTAG. (RO)

EFUSE_HARD_DIS_JTAG_ERR Any bit equal to 1 denotes a programming error in EFUSE_HARD_DIS_JTAG. (RO)

Continued on the next page...

Register 16.24: EFUSE_RD_REPEAT_ERR0_REG (0x017C)

Continued from the previous page...

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT. (RO)

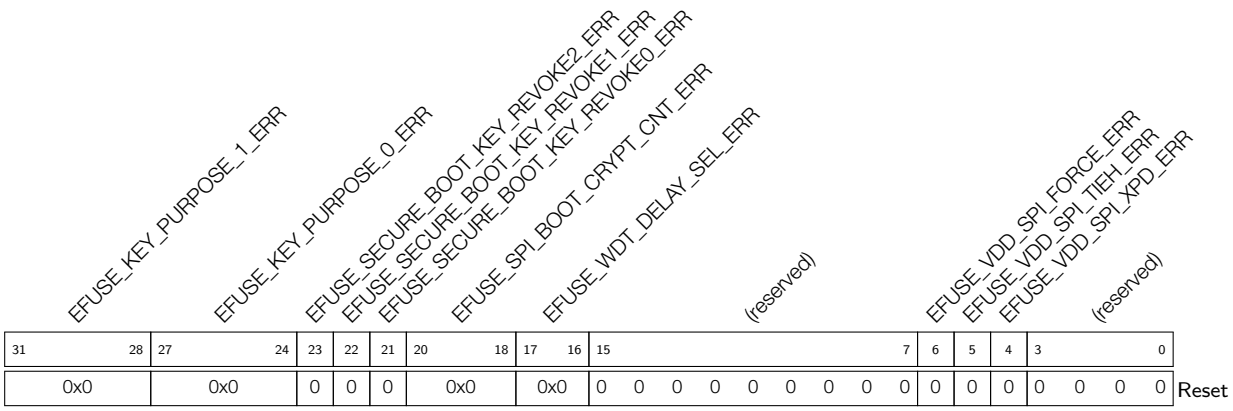
EFUSE_USB_EXCHG_PINS_ERR Any bit equal to 1 denotes a programming error in EFUSE_USB_EXCHG_PINS. (RO)

EFUSE_EXT_PHY_ENABLE_ERR Any bit equal to 1 denotes a programming error in EFUSE_EXT_PHY_ENABLE. (RO)

EFUSE_USB_FORCE_NOPERSIST_ERR Any bit equal to 1 denotes a programming error in EFUSE_USB_FORCE_NOPERSIST. (RO)

EFUSE_RPT4_RESERVED0_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED0. (RO)

Register 16.25: EFUSE_RD_REPEAT_ERR1_REG (0x0180)



EFUSE_VDD_SPI_XPD_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_XPD. (RO)

EFUSE_VDD_SPI_TIEH_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_TIEH. (RO)

EFUSE_VDD_SPI_FORCE_ERR Any bit equal to 1 denotes a programming error in EFUSE_VDD_SPI_FORCE. (RO)

EFUSE_WDT_DELAY_SEL_ERR Any bit equal to 1 denotes a programming error in EFUSE_WDT_DELAY_SEL. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR Any bit equal to 1 denotes a programming error in EFUSE_SPI_BOOT_CRYPT_CNT. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE0. (RO)

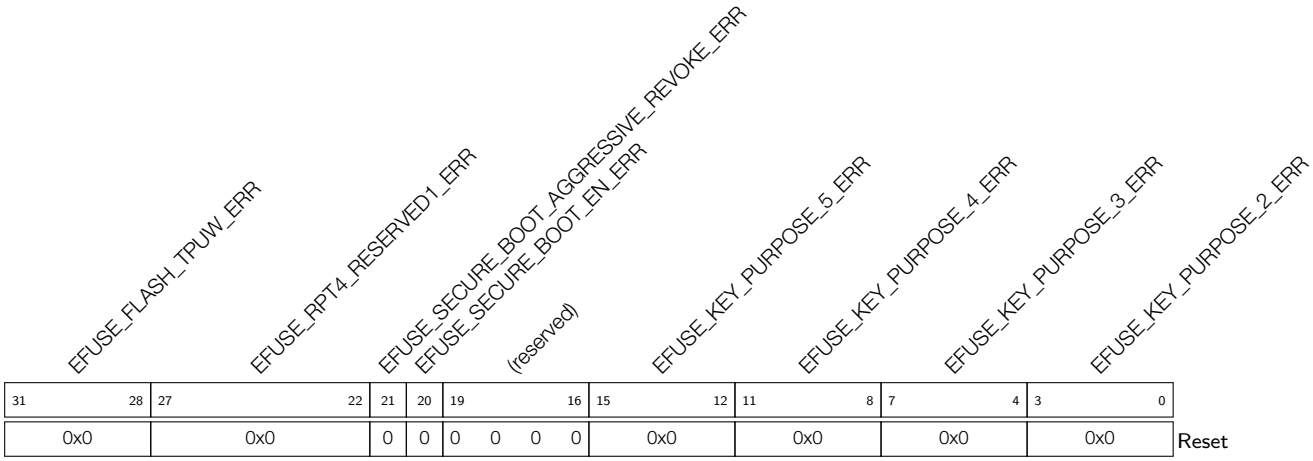
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE1. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_KEY_REVOKE2. (RO)

EFUSE_KEY_PURPOSE_0_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_0. (RO)

EFUSE_KEY_PURPOSE_1_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_1. (RO)

Register 16.26: EFUSE_RD_REPEAT_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_2. (RO)

EFUSE_KEY_PURPOSE_3_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_3. (RO)

EFUSE_KEY_PURPOSE_4_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_4. (RO)

EFUSE_KEY_PURPOSE_5_ERR Any bit equal to 1 denotes a programming error in EFUSE_KEY_PURPOSE_5. (RO)

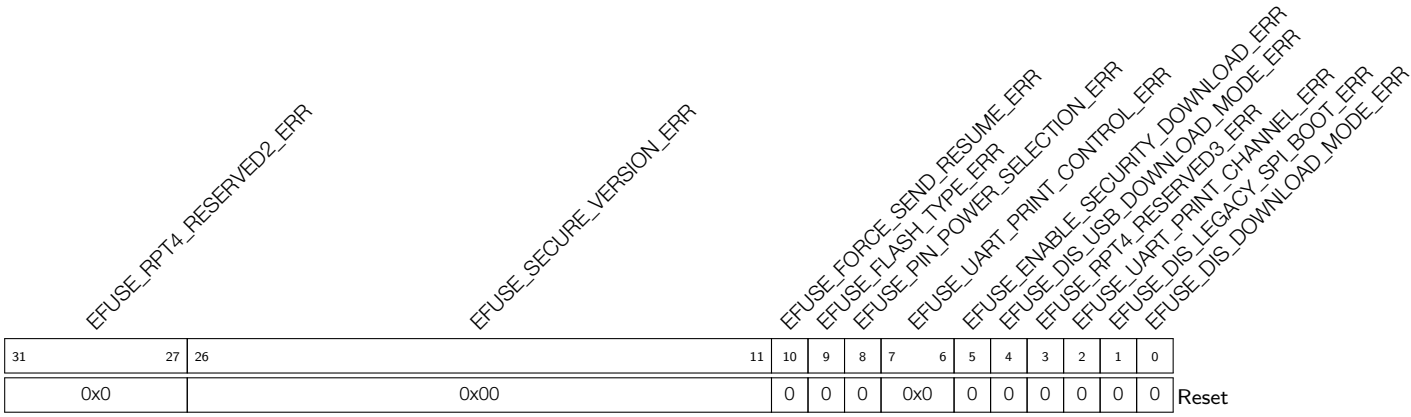
EFUSE_SECURE_BOOT_EN_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_EN. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE. (RO)

EFUSE_RPT4_RESERVED1_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED1. (RO)

EFUSE_FLASH_TPUW_ERR Any bit equal to 1 denotes a programming error in EFUSE_FLASH_TPUW. (RO)

Register 16.27: EFUSE_RD_REPEAT_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_DOWNLOAD_MODE. (RO)

EFUSE_DIS_LEGACY_SPI_BOOT_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_LEGACY_SPI_BOOT. (RO)

EFUSE_UART_PRINT_CHANNEL_ERR Any bit equal to 1 denotes a programming error in EFUSE_UART_PRINT_CHANNEL. (RO)

EFUSE_RPT4_RESERVED3_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED3. (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE_ERR Any bit equal to 1 denotes a programming error in EFUSE_DIS_USB_DOWNLOAD_MODE. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR Any bit equal to 1 denotes a programming error in EFUSE_ENABLE_SECURITY_DOWNLOAD. (RO)

EFUSE_UART_PRINT_CONTROL_ERR Any bit equal to 1 denotes a programming error in EFUSE_UART_PRINT_CONTROL. (RO)

EFUSE_PIN_POWER_SELECTION_ERR Any bit equal to 1 denotes a programming error in EFUSE_PIN_POWER_SELECTION. (RO)

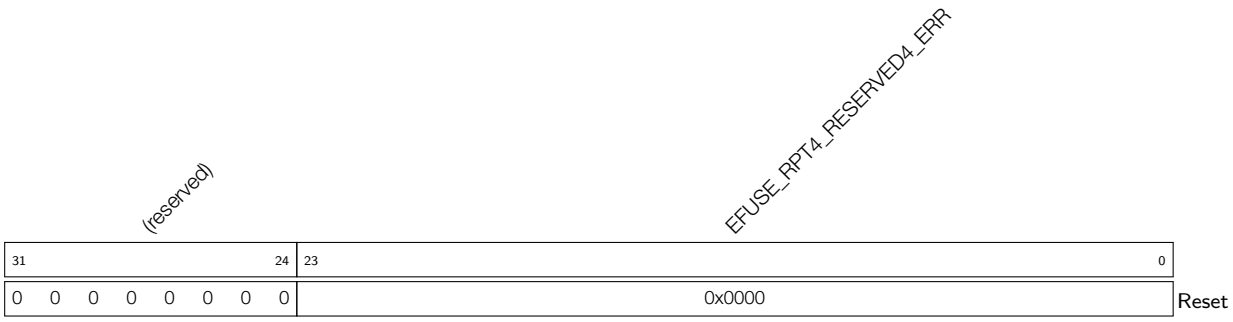
EFUSE_FLASH_TYPE_ERR Any bit equal to 1 denotes a programming error in EFUSE_FLASH_TYPE. (RO)

EFUSE_FORCE_SEND_RESUME_ERR Any bit equal to 1 denotes a programming error in EFUSE_FORCE_SEND_RESUME. (RO)

EFUSE_SECURE_VERSION_ERR Any bit equal to 1 denotes a programming error in EFUSE_SECURE_VERSION. (RO)

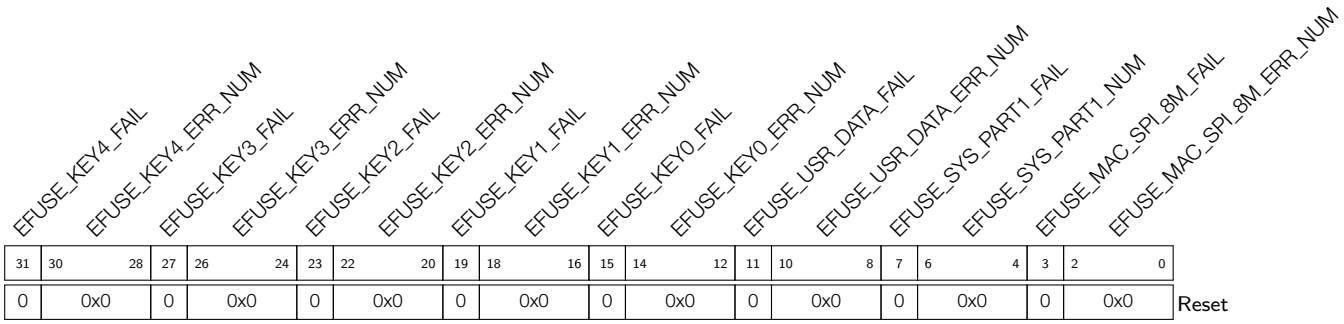
EFUSE_RPT4_RESERVED2_ERR Any bit equal to 1 denotes a programming error in EFUSE_RPT4_RESERVED2. (RO)

Register 16.28: EFUSE_RD_REPEAT_ERR4_REG (0x0190)



EFUSE_RPT4_RESERVED4_ERR If any bit in RPT4_RESERVED4 is 1, there is a programming error in EFUSE_RPT4_RESERVED4. (RO)

Register 16.29: EFUSE_RD_RS_ERR0_REG (0x01C0)



EFUSE_MAC_SPI_8M_ERR_NUM The value of this signal means the number of error bytes in BLOCK1. (RO)

EFUSE_MAC_SPI_8M_FAIL 0: Means no failure and that the data of BLOCK1 is reliable; 1: Means that programming BLOCK1 data failed and the number of error bytes is over 5. (RO)

EFUSE_SYS_PART1_NUM The value of this signal means the number of error bytes in BLOCK2. (RO)

EFUSE_SYS_PART1_FAIL 0: Means no failure and that the data of BLOCK2 is reliable; 1: Means that programming BLOCK2 data failed and the number of error bytes is over 5. (RO)

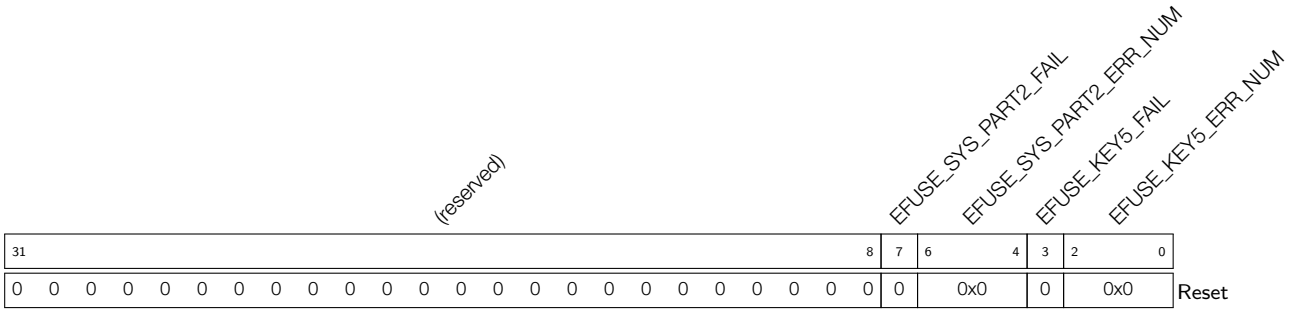
EFUSE_USR_DATA_ERR_NUM The value of this signal means the number of error bytes in BLOCK3. (RO)

EFUSE_USR_DATA_FAIL 0: Means no failure and that the data of BLOCK3 is reliable; 1: Means that programming BLOCK3 data failed and the number of error bytes is over 5. (RO)

EFUSE_KEY*n*_ERR_NUM The value of this signal means the number of error bytes in KEY*n*. (RO)

EFUSE_KEY*n*_FAIL 0: Means no failure and that the data of KEY*n* is reliable; 1: Means that programming KEY*n* failed and the number of error bytes is over 5. (RO)

Register 16.30: EFUSE_RD_RS_ERR1_REG (0x01C4)



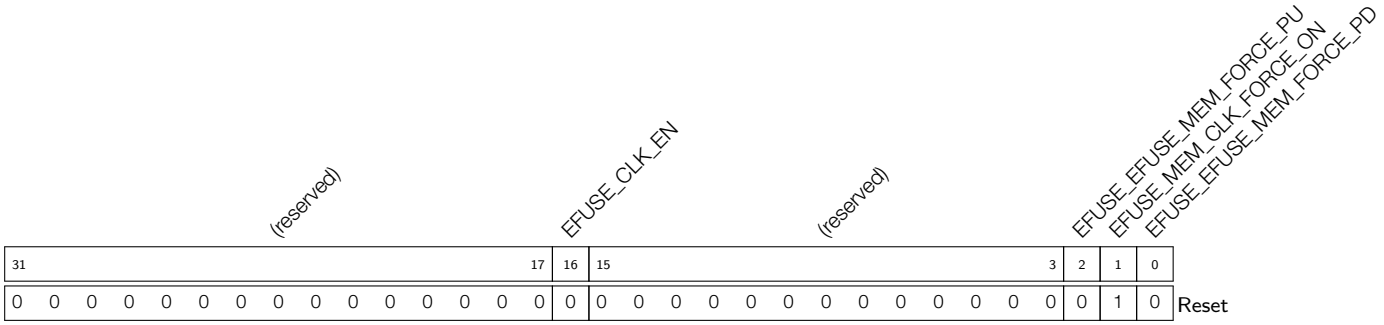
EFUSE_KEY5_ERR_NUM The value of this signal means the number of error bytes in KEY5. (RO)

EFUSE_KEY5_FAIL 0: Means no failure and that the data of KEY5 is reliable; 1: Means that programming user data failed and the number of error bytes is over 5. (RO)

EFUSE_SYS_PART2_ERR_NUM The value of this signal means the number of error bytes in BLOCK10. (RO)

EFUSE_SYS_PART2_FAIL 0: Means no failure and that the data of BLOCK10 is reliable; 1: Means that programming BLOCK10 data failed and the number of error bytes is over 5. (RO)

Register 16.31: EFUSE_CLK_REG (0x01C8)



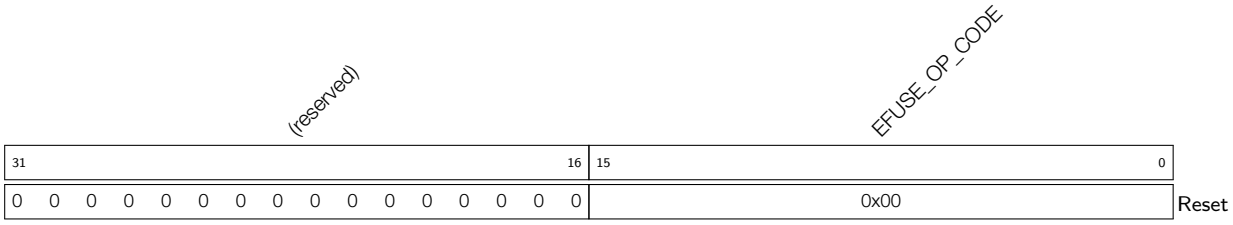
EFUSE_EFUSE_MEM_FORCE_PD If set, forces eFuse SRAM into power-saving mode. (R/W)

EFUSE_MEM_CLK_FORCE_ON If set, forces to activate clock signal of eFuse SRAM. (R/W)

EFUSE_EFUSE_MEM_FORCE_PU If set, forces eFuse SRAM into working mode. (R/W)

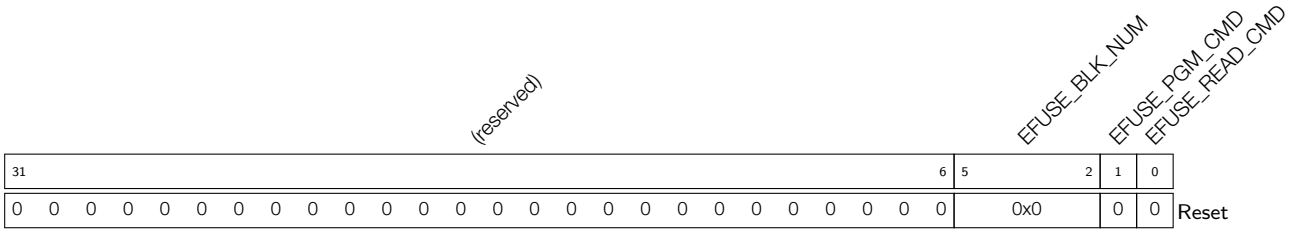
EFUSE_CLK_EN If set, forces to enable clock signal of eFuse memory. (R/W)

Register 16.32: EFUSE_CONF_REG (0x01CC)



EFUSE_OP_CODE 0x5A5A: Operate programming command; 0x5AA5: Operate read command. (R/W)

Register 16.33: EFUSE_CMD_REG (0x01D4)

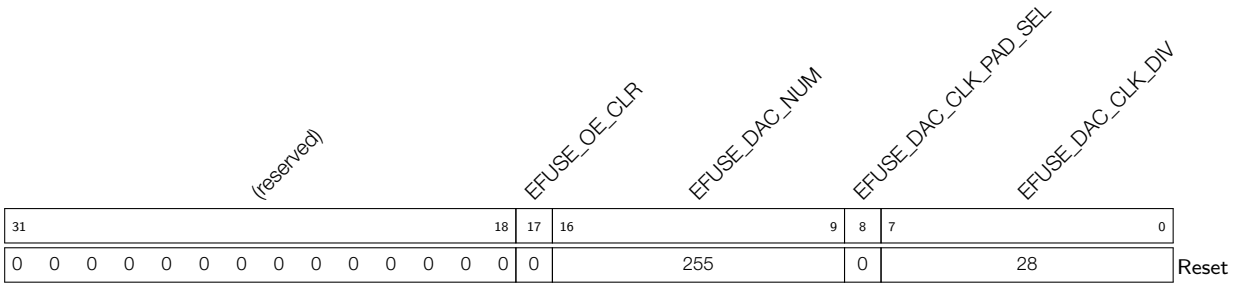


EFUSE_READ_CMD Set this bit to send read command. (R/W)

EFUSE_PGM_CMD Set this bit to send programming command. (R/W)

EFUSE_BLK_NUM The serial number of the block to be programmed. Value 0-10 corresponds to block number 0-10, respectively. (R/W)

Register 16.34: EFUSE_DAC_CONF_REG (0x01E8)



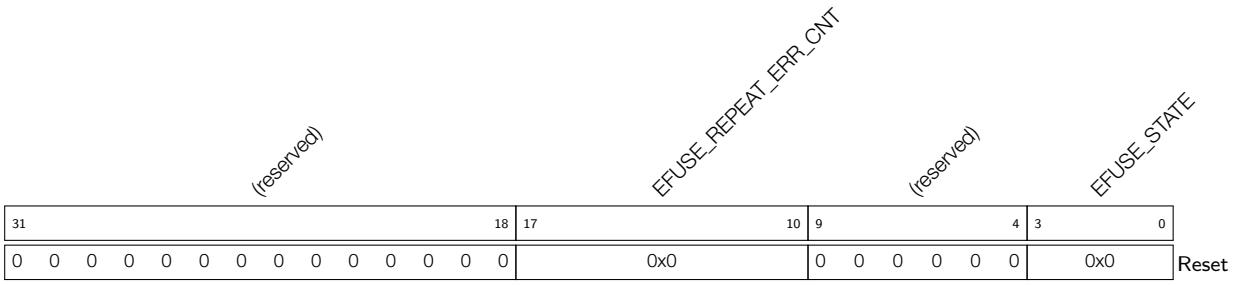
EFUSE_DAC_CLK_DIV Controls the division factor of the rising clock of the programming voltage. (R/W)

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)

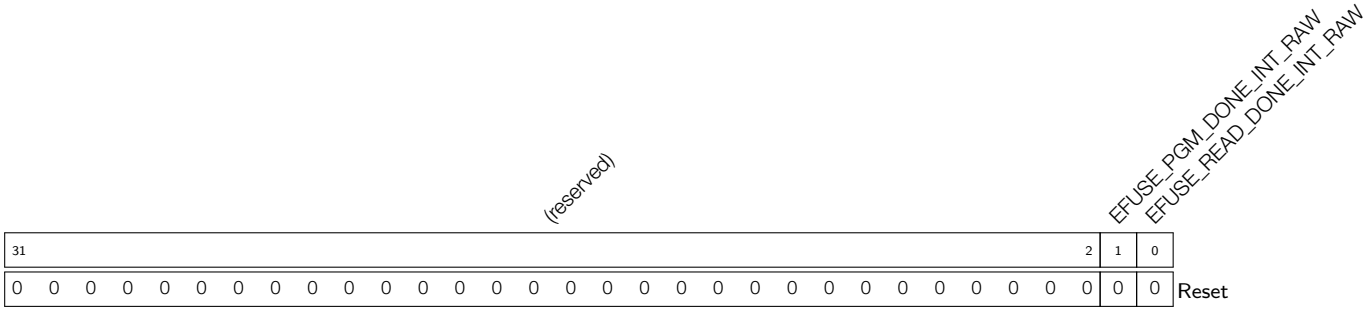
Register 16.35: EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE Indicates the state of the eFuse state machine. (RO)

EFUSE_REPEAT_ERR_CNT Indicates the number of error bits during programming BLOCK0. (RO)

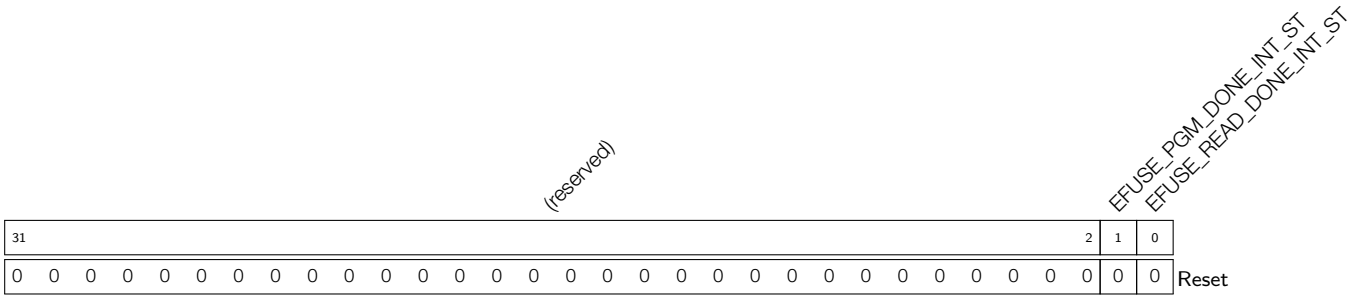
Register 16.36: EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW The raw bit signal for read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_RAW The raw bit signal for pgm_done interrupt. (RO)

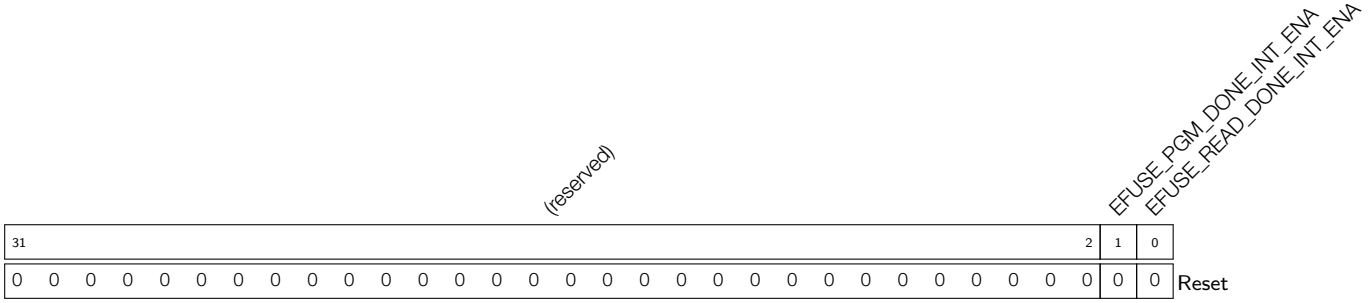
Register 16.37: EFUSE_INT_ST_REG (0x01DC)



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)

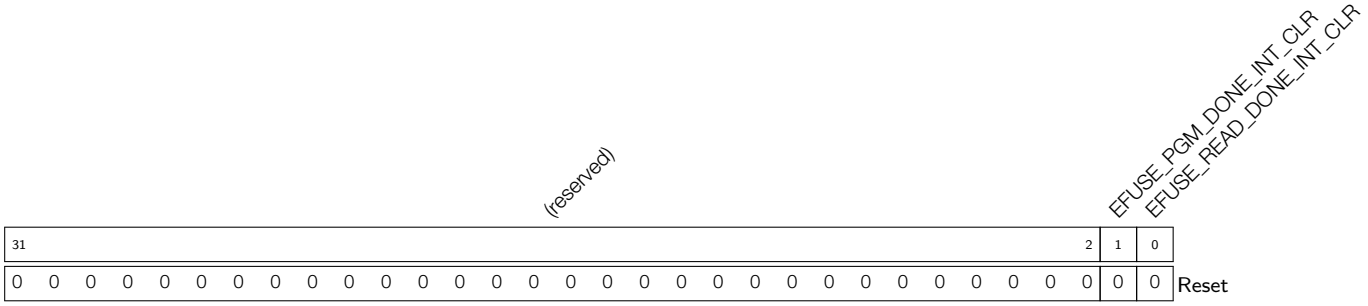
Register 16.38: EFUSE_INT_ENA_REG (0x01E0)



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)

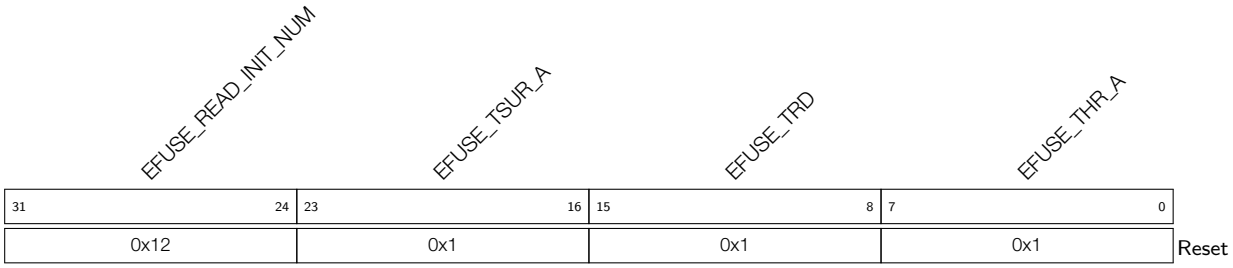
Register 16.39: EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR The clear signal for read_done interrupt. (WO)

EFUSE_PGM_DONE_INT_CLR The clear signal for pgm_done interrupt. (WO)

Register 16.40: EFUSE_RD_TIM_CONF_REG (0x01EC)



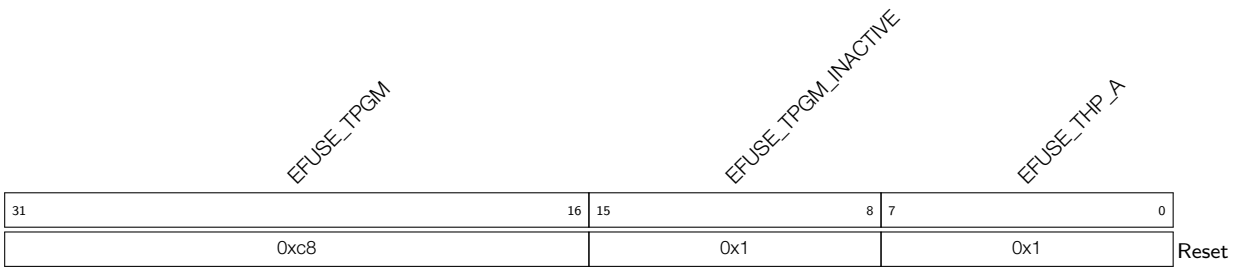
EFUSE_THR_A Configures the hold time of read operation. (R/W)

EFUSE_TRD Configures the length of pulse of read operation. (R/W)

EFUSE_TSUR_A Configures the setup time of read operation. (R/W)

EFUSE_READ_INIT_NUM Configures the initial read time of eFuse. (R/W)

Register 16.41: EFUSE_WR_TIM_CONF0_REG (0x01F0)

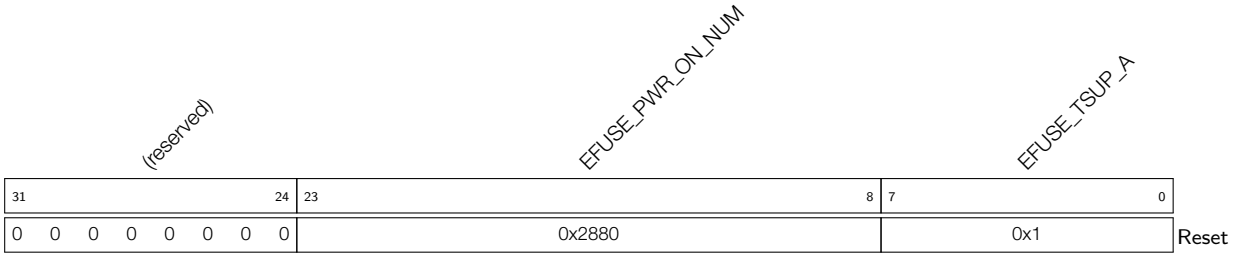


EFUSE_THP_A Configures the hold time of programming operation. (R/W)

EFUSE_TPGM_INACTIVE Configures the length of pulse during programming 0 to eFuse. (R/W)

EFUSE_TPGM Configures the length of pulse during programming 1 to eFuse. (R/W)

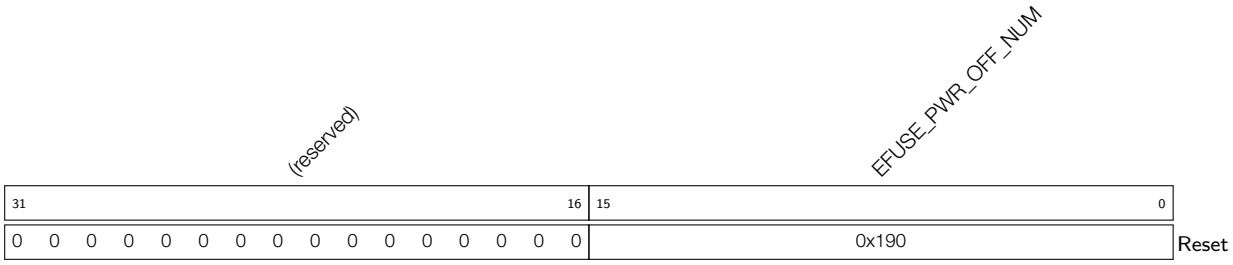
Register 16.42: EFUSE_WR_TIM_CONF1_REG (0x01F4)



EFUSE_TSUP_A Configures the setup time of programming operation. (R/W)

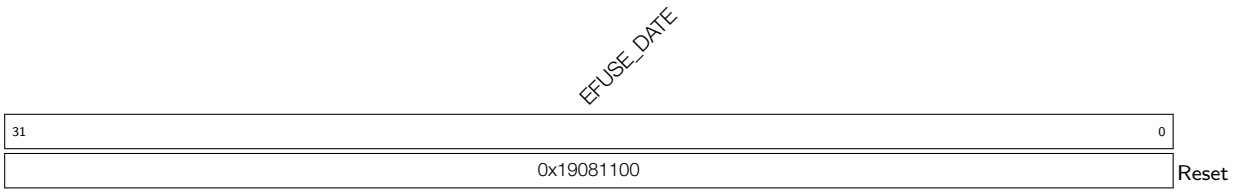
EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. (R/W)

Register 16.43: EFUSE_WR_TIM_CONF2_REG (0x01F8)



EFUSE_PWR_OFF_NUM Configures the power outage time for VDDQ. (R/W)

Register 16.44: EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Version control register. (R/W)

17. I²C Controller

17.1 Overview

The I²C (Inter-Integrated Circuit) controller allows ESP32-S2 to communicate with multiple peripheral devices. These peripheral devices can share one bus.

17.2 Features

The I²C controller has the following features:

- Master mode and slave mode
- Multi-master and multi-slave communication
- Standard mode (100 kbit/s)
- Fast mode (400 kbit/s)
- 7-bit addressing and 10-bit addressing
- Continuous data transfer in master mode achieved by pulling SCL low
- Programmable digital noise filtering
- Double addressing mode

17.3 I²C Functional Description

17.3.1 I²C Introduction

The I²C bus has two lines, namely a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I²C bus is connected to multiple devices, usually a single or multiple masters and a single or multiple slaves. However, only one master device can access a slave at a time.

The master initiates communication by generating a start condition: pulling the SDA line l ow while SCL is high, and sending nine clock pulses via SCL. The first eight pulses are used to transmits a byte, which consists of a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of a slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master and the slave can send or receive data according to the $R/\overline{W}$ bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once 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 RESTART condition, a slave address and a $R/\overline{W}$ bit before changing its operation.

17.3.2 I²C Architecture

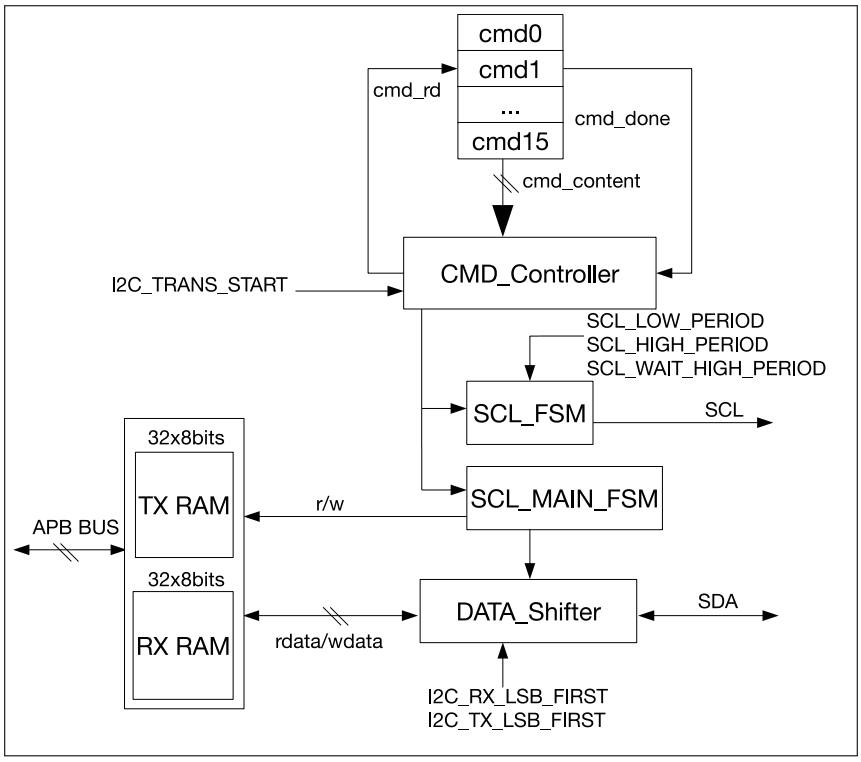


Figure 17-1. I²C Master Architecture

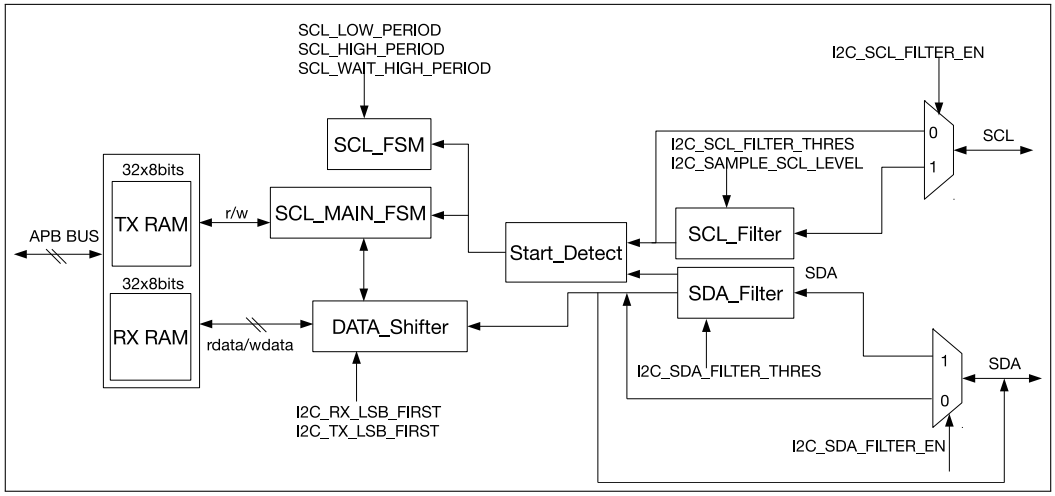


Figure 17-2. I²C Slave Architecture

The I²C controller runs either in master mode or slave mode, which is determined by I2C_MS_MODE bit. Figure 17-1 shows the architecture of a master, while Figure 17-2 shows that of a slave. The I²C 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-to-parallel data converter (DATA_Shifter), filter for SCL (SCL_Filter) and filter for SDA (SDA_Filter).

17.3.2.1 TX/RX RAM

Both TX RAM and RX RAM are 32 × 8 bits. TX RAM stores data that the I²C controller needs to send. During communication, when the I²C controller needs to send data (except acknowledgement bits), it reads data from TX RAM and sends it sequentially via SDA. When the I²C controller works in master mode, 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 dual addressing mode) and data to be sent. When the I²C controller works in slave mode, TX RAM only stores data to be sent.

RX RAM stores data the I²C controller receives during communication. When the I²C controller works in slave mode, neither slave addresses sent by the master nor register addresses (only in double addressing mode) will be stored into RX RAM. Values of RX RAM can be read by software after I²C communication completes.

Both TX RAM and RX RAM can be accessed in FIFO or non-FIFO mode. The I2C_NONFIFO_EN bit sets FIFO or non-FIFO mode.

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 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 I²C Base Address + 0x104, the third byte I²C Base Address + 0x108, and so on. The CPU can only read TX RAM via direct addresses. Unlike addresses for writing, TX RAM must be read back from addresses starting at I²C Base Address + 0x80.

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 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 + 0x100) ~(I²C Base Address + 0x17C). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I²C Base Address + 0x100, the second byte I²C Base Address + 0x104, the third byte I²C Base Address + 0x108 and so on.

Given that addresses for writing to TX RAM have an identical range with those for reading RX RAM, TX RAM and RX RAM can be seen as a 32 x 8 RAM. In following sections TX RAM and RX RAM are referred to as RAM.

17.3.2.2 CMD_Controller

] When the I²C controller works in master mode, the integrated CMD_Controller module reads commands from 16 sequential command registers and controls SCL_FSM and SDA_FSM accordingly.

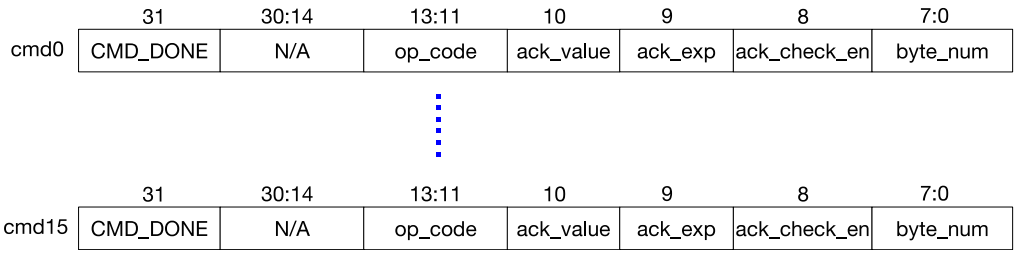


Figure 17-3. Structure of I²C Command Register

Command registers, whose structure is illustrated in Figure 17-3, are active only when the I²C controller works in

master mode. Fields of command registers are:

1. CMD_DONE: Indicates that a command has been executed. After each command has been executed, the corresponding CMD_DONE bit is set to 1 by hardware. By reading this bit, software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. op_code: Indicates the command. The I²C controller supports five commands:
   - RSTART: op_code = 0: The I²C controller sends a START bit and a RESTART bit defined by the I²C protocol.
   - WRITE: op_code = 1: The I²C controller sends a slave address, a register address (only in double addressing mode) and data to the slave.
   - READ: op_code = 2: The I²C controller reads data from the slave.
   - STOP: op_code = 3: The I²C controller sends a STOP bit defined by the I²C protocol. This code also indicates that the command sequence has been executed, and the CMD_Controller stops reading commands. After restarted by software, the CMD_Controller resumes reading commands from command register 0.
   - END: op_code = 4: The I²C controller pulls the SCL pin down 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 controller during a read operation. This bit is ignored during RSTART, STOP, END and WRITE conditions.
4. ack_exp: Used to configure the level of the ACK bit expected by the I²C controller during a write operation. This bit is ignored during RSTART, STOP, END and READ conditions.
5. ack_check_en: Used to enable the I²C controller during a write operation to check whether ACK's 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, all 16 command registers must have a STOP or an END command.

A complete data transfer on the I²C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, separated by END commands. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, data to be transmitted, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I²C communication.

17.3.2.3 SCL_FSM

The integrated SCL_FSM module controls the SCL clock line. The frequency and duty cycle of SCL is configured using I2C_SCL_LOW_PERIOD_REG, I2C_SCL_HIGH_PERIOD_REG and I2C_SCL_WAIT_HIGH_PERIOD. After

being in non-IDLE state for over I2C_SCL_ST_TO clock cycles, SCL_FSM triggers an I2C_SCL_ST_TO_INT interrupt and returns to IDLE state.

17.3.2.4 SCL_MAIN_FSM

The integrated SCL_MAIN_FSM module controls the SDA data line and data storage. After being in non-IDLE state for over I2C_SCL_MAIN_ST_TO clock cycles, SCL_MAIN_FSM triggers an I2C_SCL_MAIN_ST_TO_INT interrupt and returns to IDLE state.

17.3.2.5 DATA_Shifter

The integrated DATA_Shifter module is used for serial/parallel conversion, converting TX RAM byte data to an outgoing serial bitstream or an incoming serial bitstream to RX RAM byte data. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used to select LSB- or MSB-first storage and transmission of data.

17.3.2.6 SCL_Filter and SDA_Filter

The integrated 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.

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 clock cycles. Given that only valid input signals can pass through the filter, SCL_Filter can remove glitches whose pulse width is lower than I2C_SCL_FILTER_THRES APB clock cycles.

SDA_Filter is identical to SCL_Filter, only applied to the SDA line. The threshold pulse width is provided in the I2C_SDA_FILTER_THRES register.

17.3.3 I²C Bus Timing

The I²C controller may use APB_CLK or REF_TICK as its clock source. When I2C_REF_ALWAYS_ON is 1, APB_CLK is used; when I2C_REF_ALWAYS_ON is 0, REF_TICK is used.

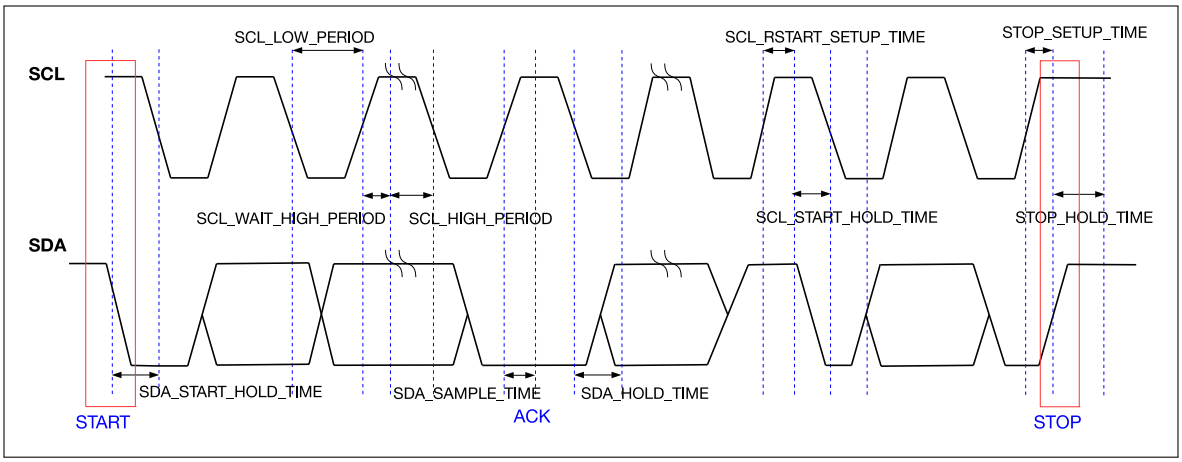


Figure 17-4. I²C Timing Diagram

Figure 17-4 shows the timing diagram of an I²C master. The unit of parameters in this figure is I2C_CLK (T_{I2C_CLK}). Specifically, when I2C_REF_ALWAYS_ON = 1, T_{I2C_CLK} is T_{APB_CLK}; when

I2C_REF_ALWAYS_ON = 0, T_{I2C_CLK} is T_{REF_TICK}. Figure 17-4 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. Parameters in Figure 17-4 are described as the following:

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) × T_{I2C_CLK}. This register is active only when the I²C controller works in master mode.
2. I2C_SCL_LOW_PERIOD: Specifies the low period of SCL. This period lasts (I2C_SCL_START_HOLD_TIME +1) × T_{I2C_CLK}. However, it could be extended when SCL is pulled low by peripheral devices or by an END command executed by the I²C controller, or when the clock is stretched. This register is active only when the I²C controller works in master mode.
3. I2C_SCL_WAIT_HIGH_PERIOD: Specifies time for SCL to go high in T_{I2C_CLK}. Please make sure that SCL will be pulled high within this time period. Otherwise, the high period of SCL may be incorrect. This register is active only when the I²C controller works in master mode.
4. I2C_SCL_HIGH_PERIOD: Specifies the high period of SCL in T_{I2C_CLK}. This register is active only when the I²C controller works in master mode. When SCL goes high within (I2C_SCL_WAIT_HIGH_PERIOD + 1) × T_{I2C_CLK}, its frequency is:

$$f_{scl} = \frac{f_{\text{I2C\_CLK}}}{\text{I2C\_SCL\_LOW\_PERIOD} + 1 + \text{I2C\_SCL\_HIGH\_PERIOD} + \text{I2C\_SCL\_WAIT\_HIGH\_PERIOD}}$$

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. This register is active both in master mode and slave mode.
6. I2C_SDA_HOLD_TIME: Specifies the interval between changing the SDA output level and the falling edge of SCL. This register is active both in master mode and slave mode.

SCL and SDA output drivers must be configured as open drain. There are two ways to achieve this:

1. Set I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT, and configure GPIO_PIN*n*_PAD_DRIVER register 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 determined together by the pull-up resistance and the capacitance on the line. The output frequency 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.

17.4 Typical Applications

For the convenience of description, I²C masters and slaves in all subsequent figures refer to ESP32-S2 I²C peripheral controllers. However, these controllers can communicate with any other I²C devices.

17.4.1 An I²C Master Writes to an I²C Slave with a 7-bit Address in One Command Sequence

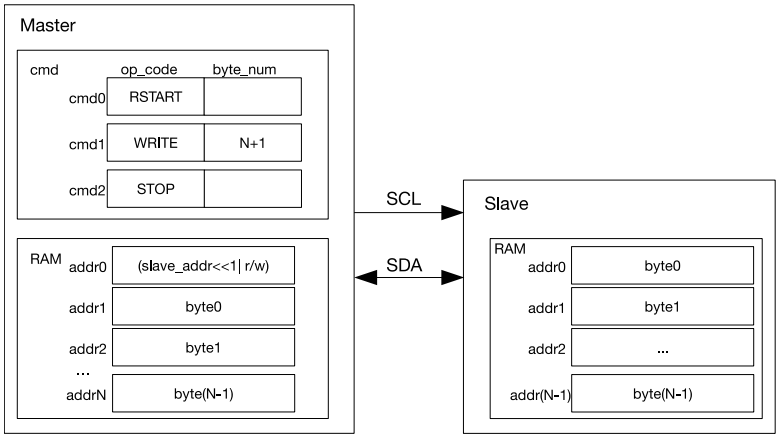


Figure 17-5. An I²C Master Writing to an I²C Slave with a 7-bit Address

Figure 17-5 shows how an I²C master writes N bytes of data using 7-bit addressing. As shown in figure 17-5, the first byte in the master's RAM is a 7-bit slave address followed by a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is zero, 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, the master enables the controller and initiates data transfer by setting I2C_TRANS_START bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL 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 the slave in the same order. The first byte is the address of the slave.
4. Send a STOP. Once the I²C master transfers a STOP bit, an I2C_TRANS_COMPLETE_INT interrupt is generated.

If data to be transferred is larger than 32 bytes, the RAM access will wrap around. While the controller sends data, software must replace data already sent in RAM. To assist with this process, the master will generate an I2C_TXFIFO_WM_INT interrupt when less than I2C_TXFIFO_WM_THRHD bytes of data remain to be sent.

After detecting this interrupt, software can refresh data in RAM. When the RAM is accessed in non-FIFO mode, in order to overwrite existing data in the RAM with new ones, software needs to first configure I2C_TX_UPDATE bit to latch the start address and the end address of data sent in the RAM, and then read I2C_TXFIFO_START_ADDR and I2C_TXFIFO_END_ADDR field in I2C_FIFO_ST_REG register to obtain these addresses. When the RAM is accessed in FIFO mode, new data can be written to the RAM directly via I2C_DATA_REG register.

The controller stops executing the command sequence after a STOP command, or when one of the following two events occurs:

1. When ack_check_en is set to 1, the I²C master checks the ACK value each time it sends a data byte. If the ACK value received does not match ack_exp (the expected ACK value) in the WRITE command, the master generates an I2C_NACK_INT interrupt and stops the transmission.

2. During the high period of SCL, if the input value and the output value of SDA do not match, the I²C master generates an I2C_ARBITRATION_LOST_INT interrupt, stops executing the command sequence, returns to IDLE state and releases SCL and SDA.

Once detecting a START bit sent by the I²C master, the I²C slave receives the address and compares it with its own address. If the received address does not match I2C_SLAVE_ADDR[6:0], the slave stops receiving data. If the received address does match I2C_SLAVE_ADDR[6:0], the slave receives data and stores them into the RAM in order.

If data to be transferred is larger than 32 bytes, the RAM may wrap around. While the controller receives data, software reclaim data already received by the slave. To assist with this process, the master will generate an I2C_RXFIFO_WM_INT after I2C_RXFIFO_WM_THRHD bytes are received in RAM.

Once detecting this interrupt, software can read data from the RAM registers. When the RAM is accessed in non-FIFO mode, in order to read data, software needs to configure I2C_RX_UPDATE bit to latch the start address and the end address of data to be reclaimed, and read I2C_RXFIFO_START_ADDR and I2C_RXFIFO_END_ADDR fields in I2C_FIFO_ST_REG register to obtain these addresses. When the RAM is accessed in FIFO mode, data can be read directly via I2C_DATA_REG register.

17.4.2 An I²C Master Writes to an I²C Slave with a 10-bit Address in One Command Sequence

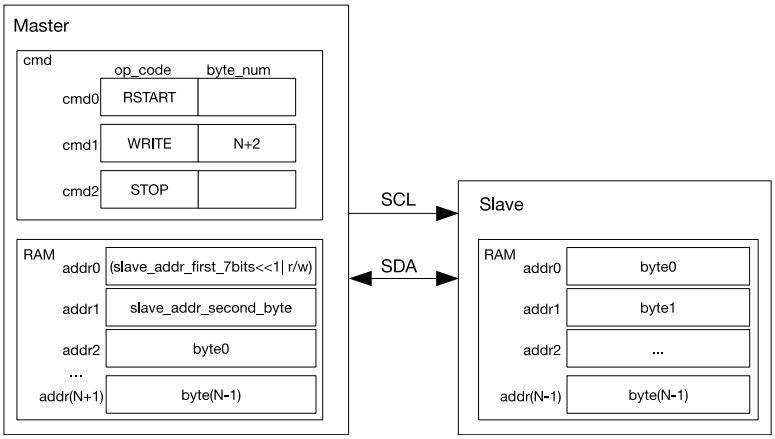


Figure 17-6. A Master Writing to a Slave with a 10-bit Address

Besides 7-bit addressing (SLV_ADDR[6:0]), the ESP32-S2 I²C controller also supports 10-bit addressing (SLV_ADDR[9:0]). In the following text, the slave address is referred to as SLV_ADDR.

Figure 17-6 shows how an I²C master writes N-bytes of data using 10-bit addressing. Unlike a 7-bit address, a 10-bit slave address is formed from two bytes. In master mode, the first byte of the slave address, which comprises slave_addr_first_7bits and a $R/\overline{W}$ bit, is stored into addr0 in the RAM. slave_addr_first_7bits should be configured as (0x78 | SLV_ADDR[9:8]). The second byte slave_addr_second_byte is stored into addr1 in the RAM. slave_addr_second_byte should be configured as SLV_ADDR[7:0].

In the slave, the 10-bit addressing mode is enabled by configuring I2C_ADDR_10BIT_EN bit. The address of the I²C slave is configured using I2C_SLAVE_ADDR. I2C_SLAVE_ADDR[14:7] should be configured as SLV_ADDR[7:0], and I2C_SLAVE_ADDR[6:0] should be configured as (0x78 | SLV_ADDR[9:8]). Since a 10-bit slave address has one more byte than a 7-bit address, byte_num of the WRITE command and the number of bytes in the RAM increase by one.

17.4.3 An I²C Master Writes to an I²C Slave with Two 7-bit Addresses in One Command Sequence

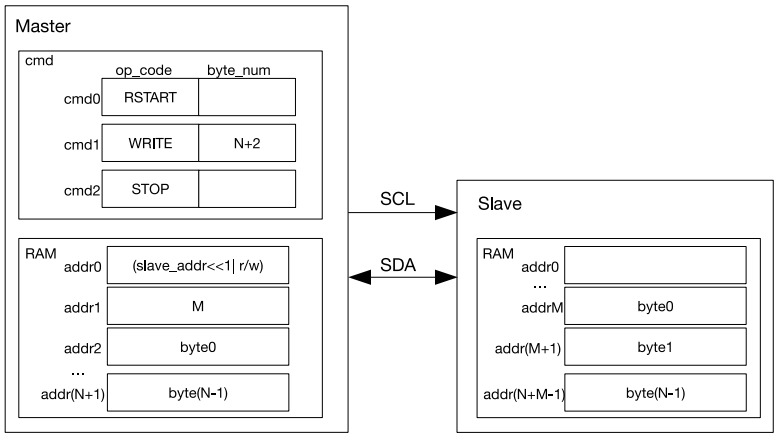


Figure 17-7. An I²C Master Writing Address M in the RAM to an I²C Slave with a 7-bit Address

When working in slave mode, the controller supports double addressing, where the first address is the address of an I²C slave, and the second one is the slave's memory address. Double addressing is enabled by configuring I2C_FIFO_ADDR_CFG_EN. When using double addressing, RAM must be accessed in non-FIFO mode. As figure 17-7 illustrates, the I²C slave put received byte0 ~ byte(N-1) into its RAM in an order staring from addrM. The RAM is overwritten every 32 bytes.

17.4.4 An I²C Master Writes to an I²C Slave with a 7-bit Address in Multiple Command Sequences

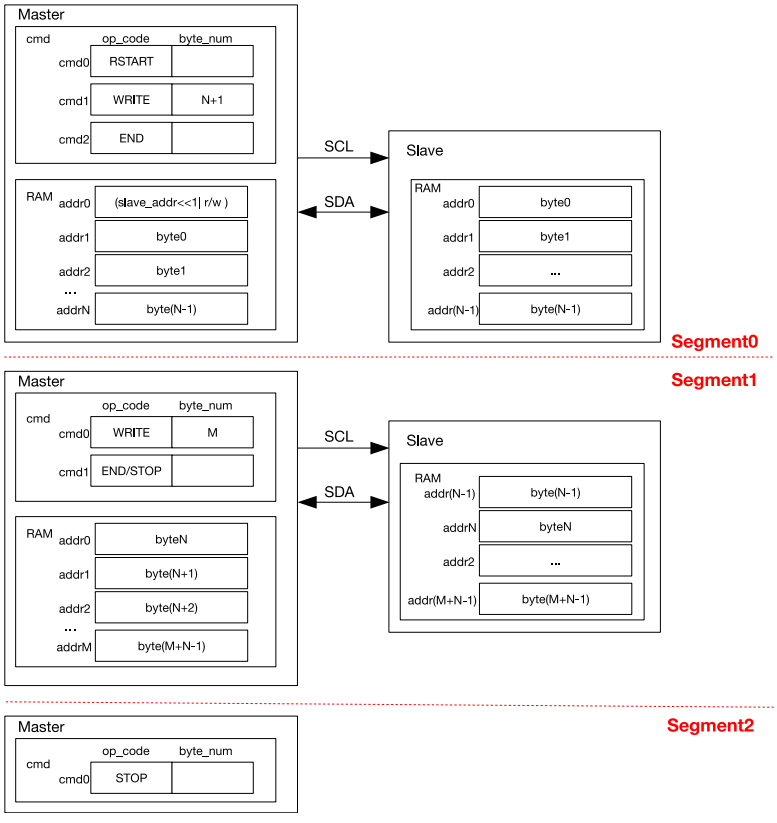


Figure 17-8. An I²C Master Writing to an I²C Slave with a 7-bit Address in Multiple Sequences

Given that the I²C Controller RAM holds only 32 bytes, when data are too large to be processed even by the wrapped RAM, it is advised to transmit them in multiple command sequences. At the end of every command sequence is an END command. When the controller executes this END command to pull SCL low, software refreshes command sequence registers and the RAM for next the transmission.

Figure 17-8 shows how an I²C master writes to an I²C slave in two or three segments. For the first segment, the CMD_Controller registers are configured as shown in Segment0. Once data in the master's RAM is ready and I2C_TRANS_START is set, the master initiates data transfer. After executing the END command, the master turns off the SCL clock and pulls the 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 and 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 the slave in two segments. The master resumes data transfer after I2C_TRANS_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C_END_DETECT_INT is detected, the CMD_Controller registers of the master are configured as shown in Segment2. Once I2C_TRANS_START is set, the master generates a STOP bit and terminates the transfer.

Please note that other I²C masters will not transact on the bus between two segments. The bus is only released after a STOP signal is sent. To interrupt the transfer, the I²C controller can be reset by setting I2C_FSM_RST at any time. This register will later be cleared automatically by hardware.

When the master is in IDLE state and I2C_SCL_RST_SLV_EN is set, hardware sends I2C_SCL_RST_SLV_NUM SCL pulses. I2C_SCL_RST_SLV_EN will later be cleared automatically by hardware.

Note that the operation of other I²C masters and I²C slaves may differ from that of the ESP32-S2 I²C devices. Please refer to datasheets of specific I²C devices.

17.4.5 An I²C Master Reads an I²C Slave with a 7-bit Address in One Command Sequence

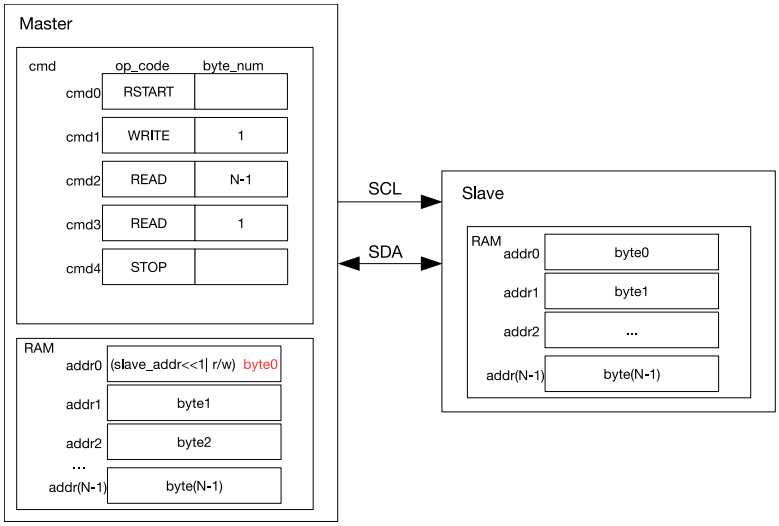


Figure 17-9. An I²C Master Reading an I²C Slave with a 7-bit Address

Figure 17-9 shows how an I²C master reads N bytes of data from an I²C slave using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed the master sends a slave address. The byte sent

comprises a 7-bit slave address and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 1, it indicates a READ operation. If the address of an I²C slave matches the sent address, this matching slave starts sending data to the master. The master generates acknowledgements according to ack_value defined in the READ command upon receiving every byte.

As illustrated in Figure 17-9, the master executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd 3. This configuration may be changed as required. The master writes received data into the controller RAM from addr0, whose original content (a slave address and a $R/\overline{W}$ bit) is overwritten by byte0 marked red in Figure 17-9.

17.4.6 An I²C Master Reads an I²C Slave with a 10-bit Address in One Command Sequence

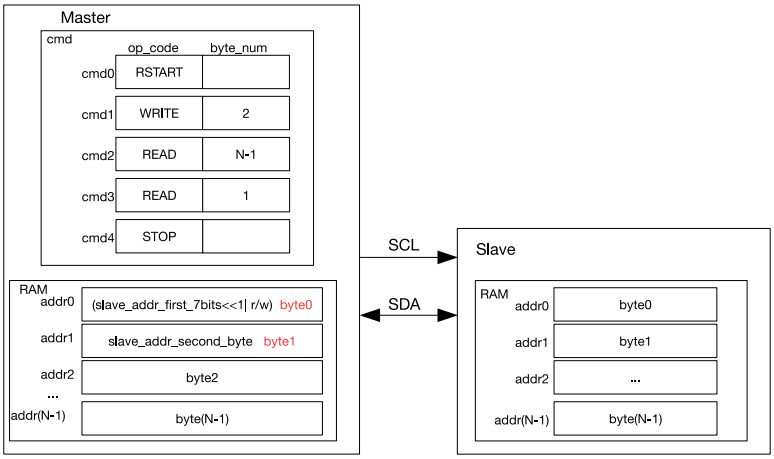


Figure 17-10. An I²C Master Reading an I²C Slave with a 10-bit Address

Figure 17-10 shows how an I²C master reads data from an I²C slave using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the I²C master is formed from two bytes, and correspondingly the RAM of this master stores a 10-bit address of two bytes. I2C_ADDR_10BIT_EN and I2C_SLAVE_ADDR[14:0] should be set. Please refer to 17.4.2 for how to set this register.

17.4.7 An I²C Master Reads an I²C Slave with Two 7-bit Addresses in One Command Sequence

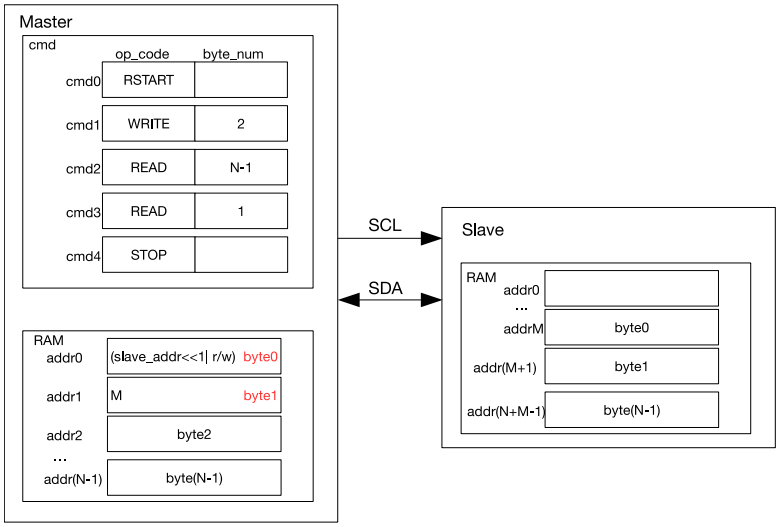


Figure 17-11. An I²C Master Reading N Bytes of Data from addrM of an I²C Slave with a 7-bit Address

Figure 17-11 shows how an I²C master reads data from specified addresses in an I²C slave. This procedure is as follows:

1. Set I2C_FIFO_ADDR_CFG_EN and prepare data to be sent in the RAM of the slave.
2. Prepare the slave address and register address M in the master.
3. Set I2C_TRANS_START and start transferring N bytes of data in the slave's RAM starting from address M to the master.

17.4.8 An I²C Master Reads an I²C Slave with a 7-bit Address in Multiple Command Sequences

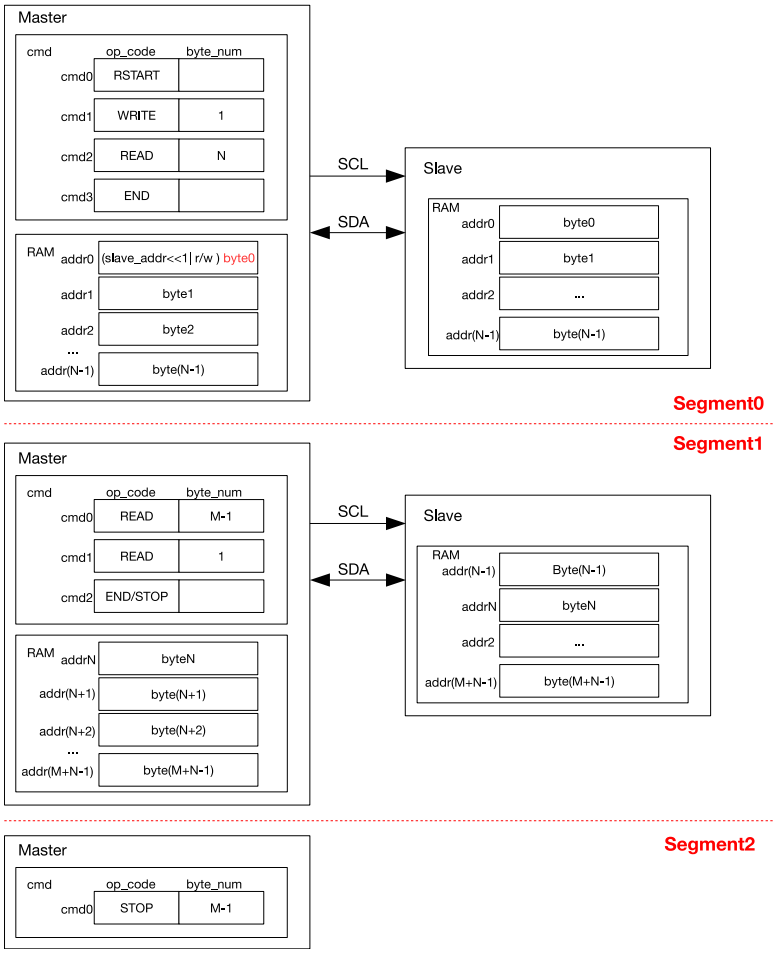


Figure 17-12. An I²C Master Reading an I²C Slave with a 7-bit Address in Segments

Figure 17-12 shows how an I²C master reads (N+M) bytes of data from an I²C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. Configure the command register and the RAM, as shown in Segment0.
2. Prepare data in the RAM of the slave, and set I2C_TRANS_START to start data transfer. After executing the END command, the master refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C_END_DETECT_INT interrupt. If cmd2 in the second segment is a STOP, then data is read from the slave in two segments. The master resumes data transfer by setting I2C_TRANS_START and terminates the transfer by sending a STOP bit.
3. If cmd2 in Segment1 is an END, then data is read from the slave in three segments. After the second data transfer finishes and an I2C_END_DETECT_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once I2C_TRANS_START is set, the master terminates the transfer by sending a STOP bit.

17.5 Clock Stretching

In slave mode, the I²C controller can hold the SCL line low in exchange for more processing time. This function called clock stretching is enabled by setting I2C_SLAVE_SCL_STRETCH_EN bit. The time period of clock

stretching is configured by setting I2C_STRETCH_PROTECT_NUM bit. The SCL line will be held low when one of the following three events occurs:

1. Finding a match: In slave mode, the address of the I²C controller matches the address sent via the SDA line.
2. RAM being full: In slave mode, RX RAM of the I²C controller is full.
3. Data all sent: In slave mode, TX RAM of the I²C controller is empty.

After SCL has been stretched low, the cause of stretching can be read from I2C_STRETCH_CAUSE bit. Clock stretching is disabled by setting I2C_SLAVE_SCL_STRETCH_CLR bit.

17.6 Interrupts

- I2C_SLAVE_STRETCH_INT: Generated when a slave holds SCL low.
- I2C_DET_START_INT: Triggered when a START bit is detected.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO[23:0] clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO[23:0] clock cycles.
- I2C_RXFIFO_UDF_INT: Triggered when the I²C controller receives I2C_NONFIFO_RX_THRES bytes of data in non-FIFO mode.
- I2C_TXFIFO_OVF_INT: Triggered when the I²C controller sends I2C_NONFIFO_TX_THRES bytes of data.
- I2C_NACK_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C_TRANS_START_INT: Triggered when the I²C controller sends a START bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than I2C_TIME_OUT clock cycles during data transfer.
- I2C_TRANS_COMPLETE_INT: Triggered when the I²C controller detects a STOP bit.
- I2C_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_ARBITRATION_LOST_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.
- I2C_BYTE_TRANS_DONE_INT: Triggered when the I²C controller sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_RXFIFO_OVF_INT: Triggered when Rx FIFO of the I²C controller overflows.
- I2C_TXFIFO_WM_INT: I²C TX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of TX FIFO are less than I2C_TXFIFO_WM_THRHD[4:0].
- I2C_RXFIFO_WM_INT: I²C Rx FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of Rx FIFO are greater than I2C_RXFIFO_WM_THRHD[4:0].

17.7 Base Address

Users can access the I²C Controller with base addresses in Table 63. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 63: I²C Controller Base Address

	Bus to Access Peripheral	Base Address
I²C0	PeriBUS1	0x3F413000
	PeriBUS2	0x60013000
I²C1	PeriBUS1	0x3F427000
	PeriBUS2	0x60027000

17.8 Register Summary

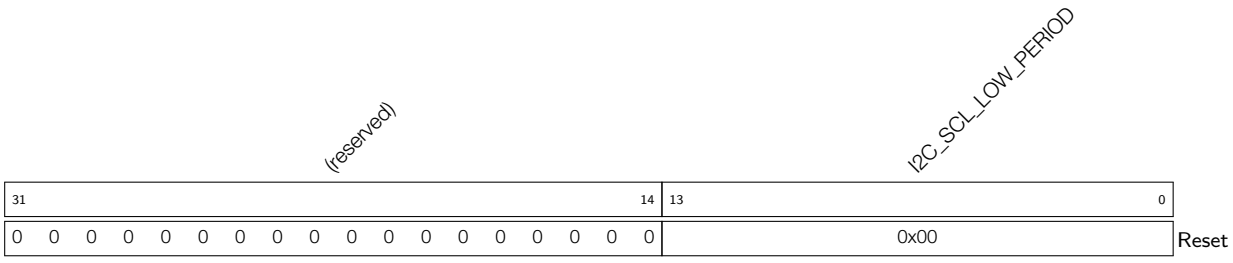
Addresses in the following table are relative to I²C base addresses provided in Section 17.7.

Name	Description	Address	Access
Timing Register			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL clock	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of the SCL clock	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the interval between pulling SDA low and pulling SCL low when the master generates a START condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the interval between the positive edge of SCL and the negative edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status time out register	0x0098	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status time out register	0x009C	R/W
Configuration Register			
I2C_CTR_REG	Transmission setting	0x0004	R/W
I2C_TO_REG	Setting time out control for receiving data	0x000C	R/W
I2C_SLAVE_ADDR_REG	Local slave address setting	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
I2C_SCL_SP_CONF_REG	Power configuration register	0x00A0	R/W
I2C_SCL_STRETCH_CONF_REG	Set SCL stretch of I2C slave	0x00A4	varies
Status Register			
I2C_SR_REG	Describe I2C work status	0x0008	RO
I2C_FIFO_ST_REG	FIFO status register	0x0014	varies
I2C_DATA_REG	RX FIFO read data	0x001C	RO

Name	Description	Address	Access
Interrupt Register			
I2C_INT_RAW_REG	Raw interrupt status	0x0020	RO
I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WO
I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W
I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Filter Register			
I2C_SCL_FILTER_CFG_REG	SCL filter configuration register	0x0050	R/W
I2C_SDA_FILTER_CFG_REG	SDA filter configuration register	0x0054	R/W
Command Register			
I2C_COMD0_REG	I2C command register 0	0x0058	R/W
I2C_COMD1_REG	I2C command register 1	0x005C	R/W
I2C_COMD2_REG	I2C command register 2	0x0060	R/W
I2C_COMD3_REG	I2C command register 3	0x0064	R/W
I2C_COMD4_REG	I2C command register 4	0x0068	R/W
I2C_COMD5_REG	I2C command register 5	0x006C	R/W
I2C_COMD6_REG	I2C command register 6	0x0070	R/W
I2C_COMD7_REG	I2C command register 7	0x0074	R/W
I2C_COMD8_REG	I2C command register 8	0x0078	R/W
I2C_COMD9_REG	I2C command register 9	0x007C	R/W
I2C_COMD10_REG	I2C command register 10	0x0080	R/W
I2C_COMD11_REG	I2C command register 11	0x0084	R/W
I2C_COMD12_REG	I2C command register 12	0x0088	R/W
I2C_COMD13_REG	I2C command register 13	0x008C	R/W
I2C_COMD14_REG	I2C command register 14	0x0090	R/W
I2C_COMD15_REG	I2C command register 15	0x0094	R/W
Version Register			
I2C_DATE_REG	Version control register	0x00F8	R/W

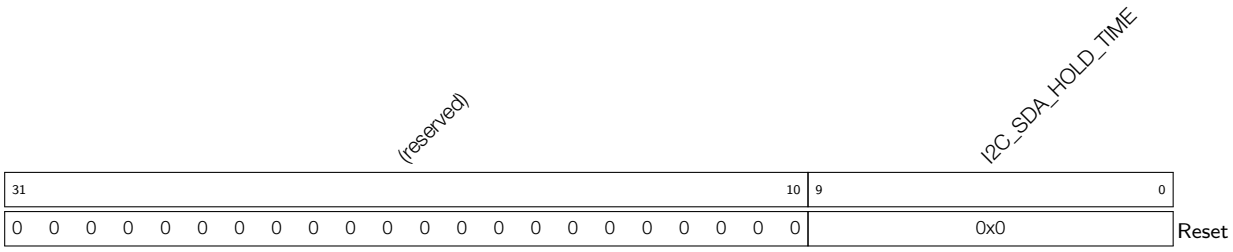
17.9 Registers

Register 17.1: I2C_SCL_LOW_PERIOD_REG (0x0000)



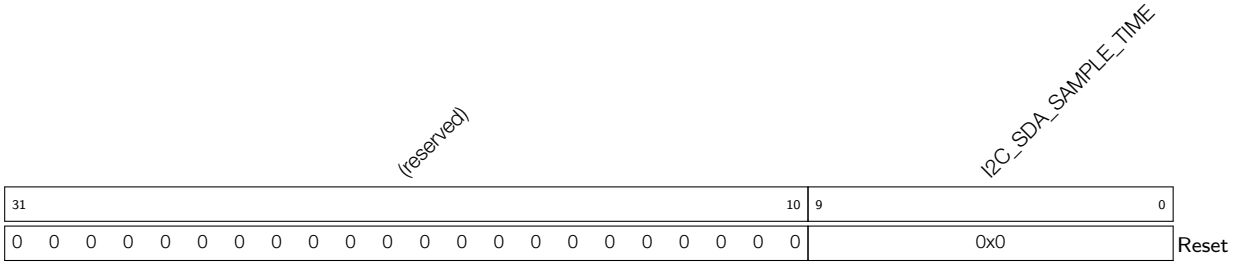
I2C_SCL_LOW_PERIOD This register is used to configure for how long SCL remains low in master mode, in I2C module clock cycles. (R/W)

Register 17.2: I2C_SDA_HOLD_REG (0x0030)



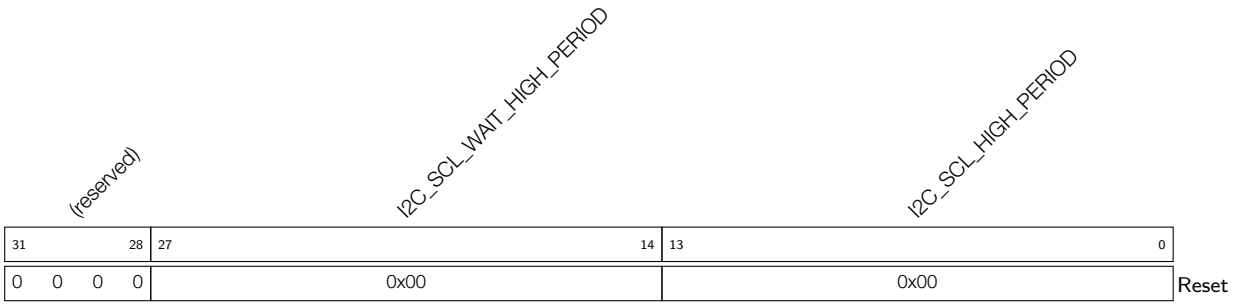
I2C_SDA_HOLD_TIME This register is used to configure the interval between changing the SDA output level and the falling edge of SCL, in I2C module clock cycles. (R/W)

Register 17.3: I2C_SDA_SAMPLE_REG (0x0034)



I2C_SDA_SAMPLE_TIME This register is used to configure the interval between the rising edge of SCL and the level sampling time of SDA, in I2C module clock cycles. (R/W)

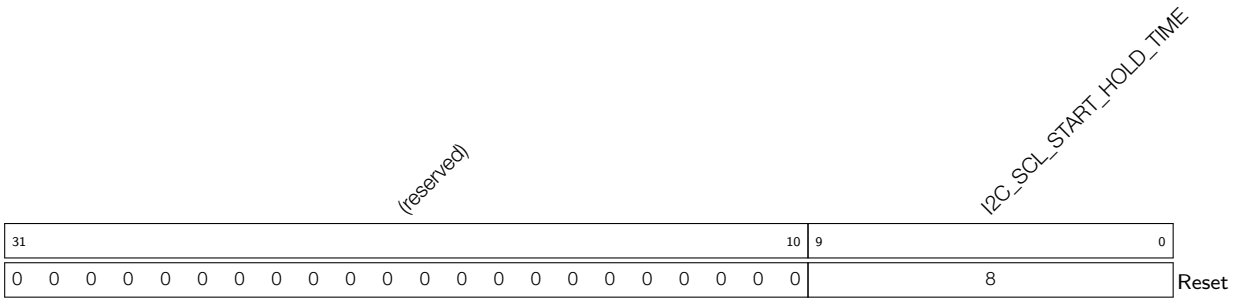
Register 17.4: I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD This register is used to configure for how long SCL remains high in master mode, in I2C module clock cycles. (R/W)

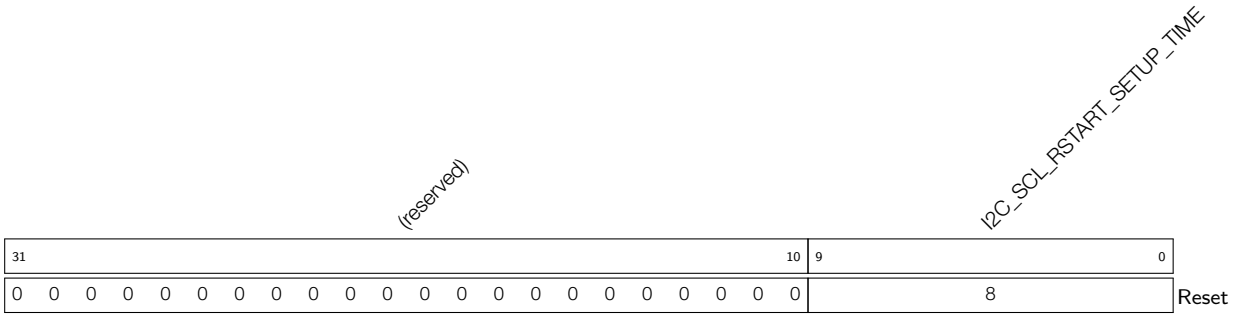
I2C_SCL_WAIT_HIGH_PERIOD This register is used to configure for the SCL_FSM's waiting period for SCL to go high in master mode, in I2C module clock cycles. (R/W)

Register 17.5: I2C_SCL_START_HOLD_REG (0x0040)



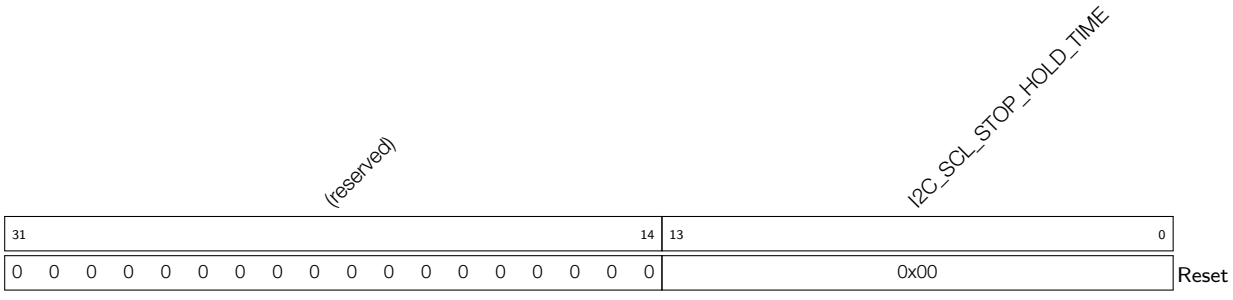
I2C_SCL_START_HOLD_TIME This register is used to configure interval between pulling SDA low and pulling SCL low when the master generates a START condition, in I2C module clock cycles. (R/W)

Register 17.6: I2C_SCL_RSTART_SETUP_REG (0x0044)



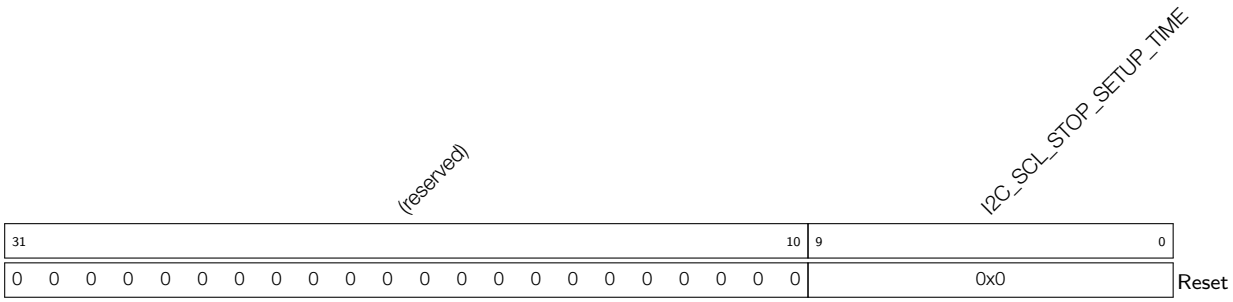
I2C_SCL_RSTART_SETUP_TIME This register is used to configure the interval between the positive edge of SCL and the negative edge of SDA for a RESTART condition, in I2C module clock cycles. (R/W)

Register 17.7: I2C_SCL_STOP_HOLD_REG (0x0048)



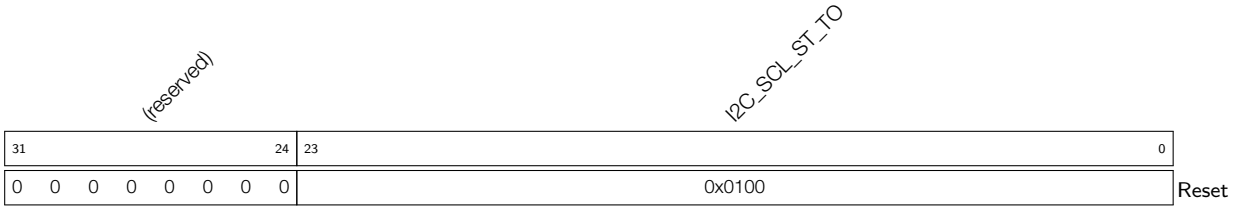
I2C_SCL_STOP_HOLD_TIME This register is used to configure the delay after the STOP condition, in I2C module clock cycles. (R/W)

Register 17.8: I2C_SCL_STOP_SETUP_REG (0x004C)



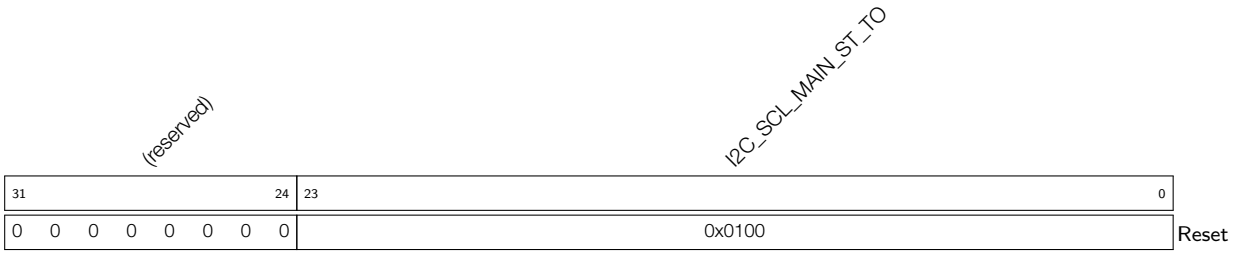
I2C_SCL_STOP_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the positive edge of SDA, in I2C module clock cycles. (R/W)

Register 17.9: I2C_SCL_ST_TIME_OUT_REG (0x0098)



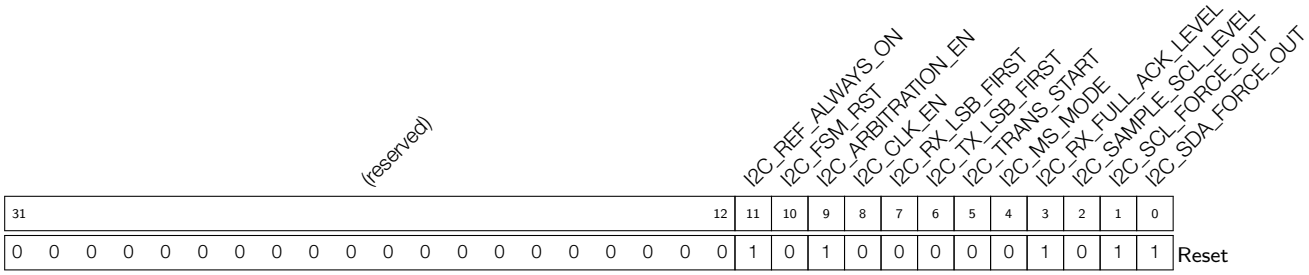
I2C_SCL_ST_TO The threshold value of SCL_FSM state unchanged period. (R/W)

Register 17.10: I2C_SCL_MAIN_ST_TIME_OUT_REG (0x009C)



I2C_SCL_MAIN_ST_TO The threshold value of SCL_MAIN_FSM state unchanged period. (R/W)

Register 17.11: I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SCL_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SAMPLE_SCL_LEVEL This register is used to select the sample mode. 1: sample SDA data on the SCL low level. 0: sample SDA data on the SCL high level. (R/W)

I2C_RX_FULL_ACK_LEVEL This register is used to configure the ACK value that need to sent by master when the rx_fifo_cnt has reached the threshold. (R/W)

I2C_MS_MODE Set this bit to configure the module as an I2C Master. Clear this bit to configure the module as an I2C Slave. (R/W)

I2C_TRANS_START Set this bit to start sending the data in TX FIFO. (R/W)

I2C_TX_LSB_FIRST This bit is used to control the sending mode for data needing to be sent. 1: send data from the least significant bit; 0: send data from the most significant bit. (R/W)

I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. 1: receive data from the least significant bit; 0: receive data from the most significant bit. (R/W)

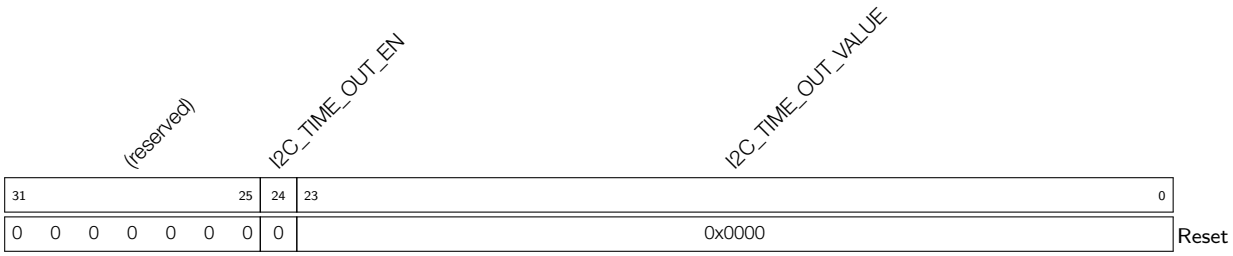
I2C_CLK_EN Reserved (R/W)

I2C_ARBITRATION_EN This is the enable bit for I2C bus arbitration function. (R/W)

I2C_FSM_RST This register is used to reset the SCL_FSM. (R/W)

I2C_REF_ALWAYS_ON This register is used to control the REF_TICK. (R/W)

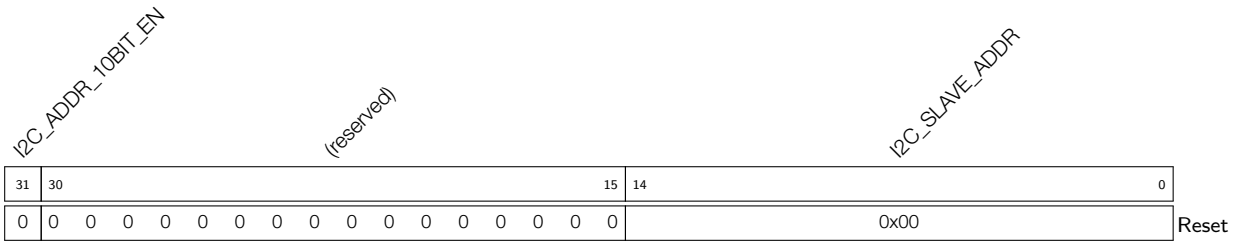
Register 17.12: I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE This register is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

I2C_TIME_OUT_EN This is the enable bit for time out control. (R/W)

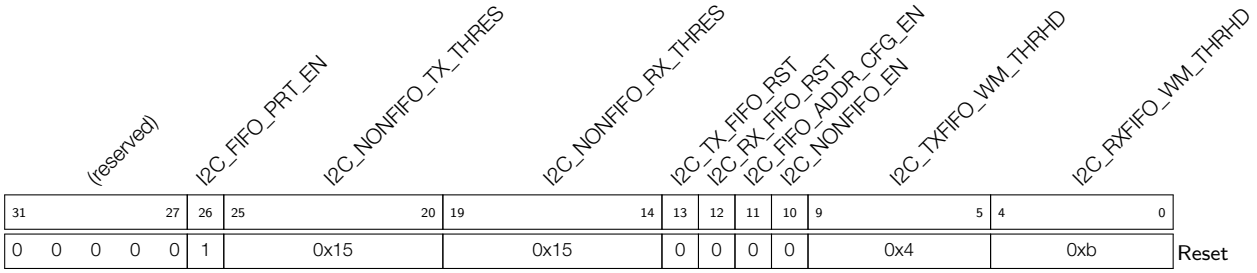
Register 17.13: I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR When configured as an I2C Slave, this field is used to configure the slave address. (R/W)

I2C_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode in master mode. (R/W)

Register 17.14: I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD The water mark 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[4:0], I2C_RXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_TXFIFO_WM_THRHD The water mark 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[4:0], I2C_TXFIFO_WM_INT_RAW bit will be valid. (R/W)

I2C_NONFIFO_EN Set this bit to enable APB non-FIFO mode. (R/W)

I2C_FIFO_ADDR_CFG_EN When this bit is set to 1, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM. (R/W)

I2C_RX_FIFO_RST Set this bit to reset RX FIFO. (R/W)

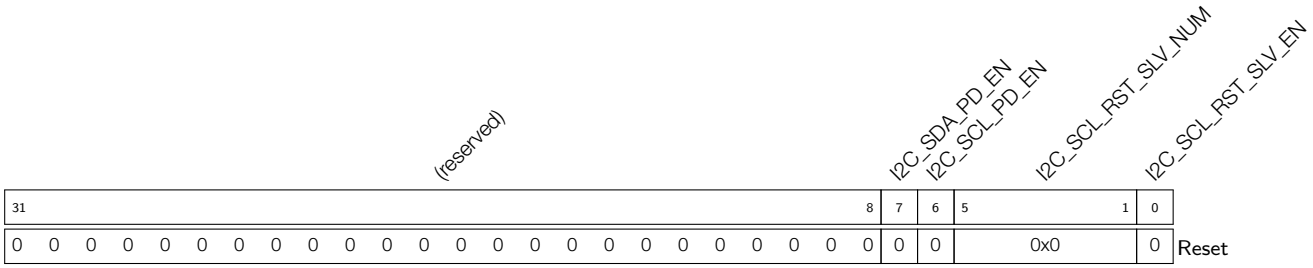
I2C_TX_FIFO_RST Set this bit to reset TX FIFO. (R/W)

I2C_NONFIFO_RX_THRES When I2C receives more than I2C_NONFIFO_RX_THRES bytes of data, it will generate an I2C_RXFIFO_UDF_INT interrupt and update the current offset address of the received data. (R/W)

I2C_NONFIFO_TX_THRES When I2C sends more than I2C_NONFIFO_TX_THRES bytes of data, it will generate an I2C_TXFIFO_OVF_INT interrupt and update the current offset address of the sent data. (R/W)

I2C_FIFO_PRT_EN The control enable bit of FIFO pointer in non-FIFO mode. This bit controls the valid bits and the interrupts of TX/RX FIFO overflow, underflow, full and empty. (R/W)

Register 17.15: I2C_SCL_SP_CONF_REG (0x00A0)



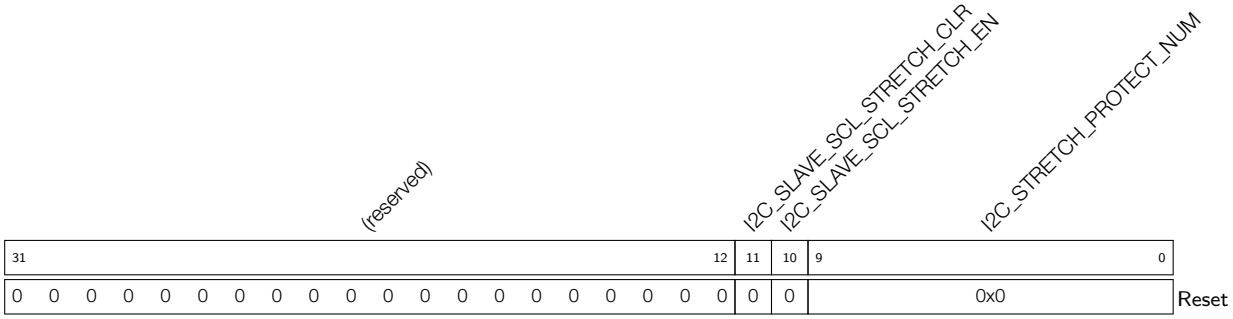
I2C_SCL_RST_SLV_EN When I2C master is IDLE, set this bit to send out SCL pulses. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0]. (R/W)

I2C_SCL_RST_SLV_NUM Configure the pulses of SCL generated in I2C master mode. 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. 1: Power down. 0: Not 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. 1: Power down. 0: Not power down. Set I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN to 1 to stretch SDA low. (R/W)

Register 17.16: I2C_SCL_STRETCH_CONF_REG (0x00A4)

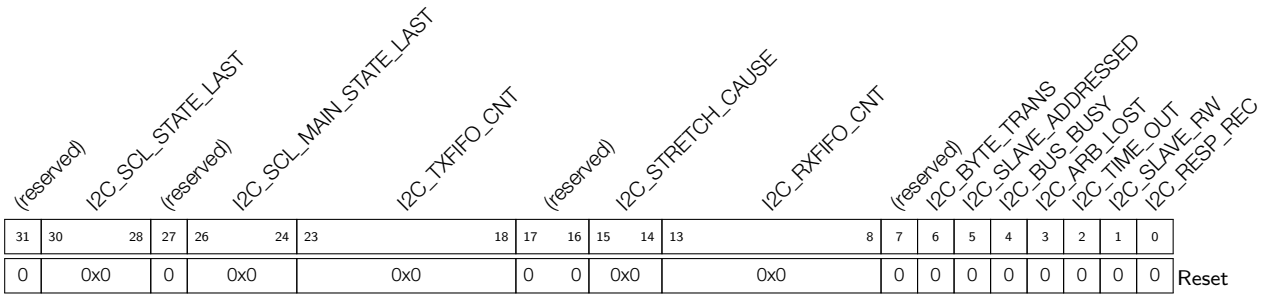


I2C_STRETCH_PROTECT_NUM Configure the period of I2C slave stretching SCL line. (R/W)

I2C_SLAVE_SCL_STRETCH_EN The enable bit for slave SCL stretch function. 1: Enable. 0: Disable. The SCL output line will be stretched low when I2C_SLAVE_SCL_STRETCH_EN is 1 and stretch event happens. The stretch cause can be seen in I2C_STRETCH_CAUSE. (R/W)

I2C_SLAVE_SCL_STRETCH_CLR Set this bit to clear the I2C slave SCL stretch function. (WO)

Register 17.17: I2C_SR_REG (0x0008)



I2C_RESP_REC The received ACK value in master mode or slave mode. 0: ACK; 1: NACK. (RO)

I2C_SLAVE_RW When in slave mode, 1: master reads from slave; 0: master writes to slave. (RO)

I2C_TIME_OUT When the I2C controller takes more than I2C_TIME_OUT clocks to receive a data bit, this field changes to 1. (RO)

I2C_ARB_LOST When the I2C controller loses control of SCL line, this register changes to 1. (RO)

I2C_BUS_BUSY 1: the I2C bus is busy transferring data; 0: the I2C bus is in idle state. (RO)

I2C_SLAVE_ADDRESSED When configured as an I2C Slave, and the address sent by the master is equal to the address of the slave, then this bit will be of high level. (RO)

I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

I2C_RXFIFO_CNT This field represents the amount of data needed to be sent. (RO)

I2C_STRETCH_CAUSE The cause of stretching SCL low in slave mode. 0: stretching SCL low at the beginning of I2C read data state. 1: stretching SCL low when I2C TX FIFO is empty in slave mode. 2: stretching SCL low when I2C RX FIFO is full in slave mode. (RO)

I2C_TXFIFO_CNT This field stores the amount of received data in RAM. (RO)

I2C_SCL_MAIN_STATE_LAST This field indicates the states of the I2C module state machine. 0: Idle; 1: Address shift; 2: ACK address; 3: RX data; 4: TX data; 5: Send ACK; 6: Wait ACK (RO)

I2C_SCL_STATE_LAST This field indicates the states of the state machine used to produce SCL. 0: Idle; 1: Start; 2: Negative edge; 3: Low; 4: Positive edge; 5: High; 6: Stop (RO)

Register 17.18: I2C_FIFO_ST_REG (0x0014)

(reserved)		I2C_SLAVE_RW_POINT	I2C_TX_UPDATE	I2C_RX_UPDATE	I2C_TXFIFO_END_ADDR	I2C_TXFIFO_START_ADDR	I2C_RXFIFO_END_ADDR	I2C_RXFIFO_START_ADDR	
31	30	29 … 22	21	20	19 … 15	14 … 10	9 … 5	4 … 0	
0	0	0x0	0	0	0x0	0x0	0x0	0x0	Reset

I2C_RXFIFO_START_ADDR This is the offset address of the last received data, as described in I2C_NONFIFO_RX_THRES. (RO)

I2C_RXFIFO_END_ADDR This is the offset address of the last received data, as described in I2C_NONFIFO_RX_THRES. This value refreshes when an I2C_RXFIFO_UDF_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_TXFIFO_START_ADDR This is the offset address of the first sent data, as described in I2C_NONFIFO_TX_THRES. (RO)

I2C_TXFIFO_END_ADDR This is the offset address of the last sent data, as described in I2C_NONFIFO_TX_THRES. The value refreshes when an I2C_TXFIFO_OVF_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_RX_UPDATE Write 0 or 1 to I2C_RX_UPDATE to update the value of I2C_RXFIFO_END_ADDR and I2C_RXFIFO_START_ADDR. (WO)

I2C_TX_UPDATE Write 0 or 1 to I2C_TX_UPDATE to update the value of I2C_TXFIFO_END_ADDR and I2C_TXFIFO_START_ADDR. (WO)

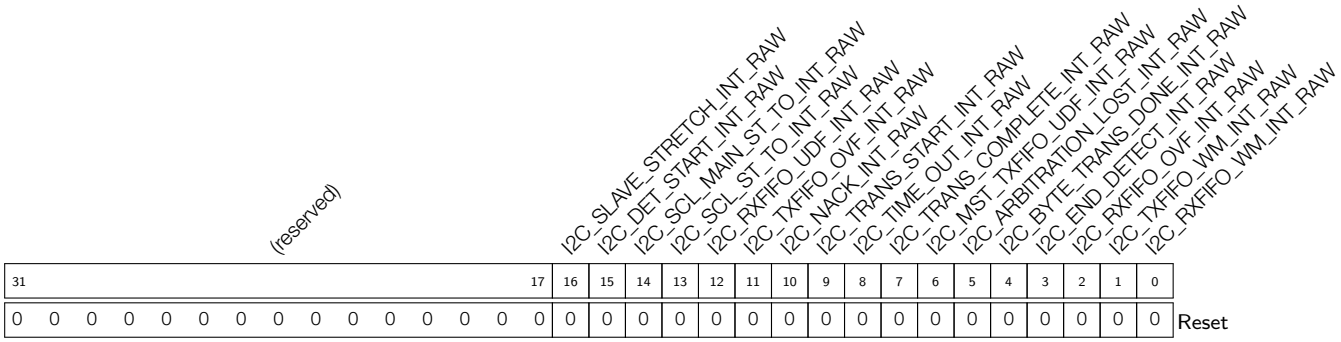
I2C_SLAVE_RW_POINT The received data in I2C slave mode. (RO)

Register 17.19: I2C_DATA_REG (0x001C)

(reserved)	I2C_FIFO_RDATA	
31 … 8	7 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

I2C_FIFO_RDATA The value of RX FIFO read data. (RO)

Register 17.20: I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW The raw interrupt bit for I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_RAW The raw interrupt bit for I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt bit for I2C_RXFIFO_OVF_INT interrupt. (RO)

I2C_END_DETECT_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt bit for I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TIME_OUT_INT_RAW The raw interrupt bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_START_INT_RAW The raw interrupt bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_NACK_INT_RAW The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt bit for I2C_TXFIFO_OVF_INT interrupt. (RO)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt bit for I2C_RXFIFO_UDF_INT interrupt. (RO)

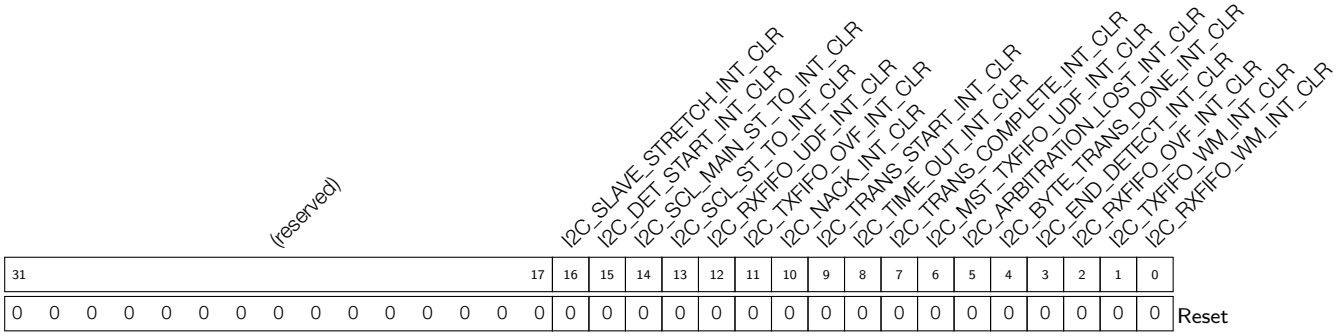
I2C_SCL_ST_TO_INT_RAW The raw interrupt bit for I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

I2C_DET_START_INT_RAW The raw interrupt bit for I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_RAW The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

Register 17.21: I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR Set this bit to clear I2C_RXFIFO_WM_INT interrupt. (WO)

I2C_TXFIFO_WM_INT_CLR Set this bit to clear I2C_TXFIFO_WM_INT interrupt. (WO)

I2C_RXFIFO_OVF_INT_CLR Set this bit to clear I2C_RXFIFO_OVF_INT interrupt. (WO)

I2C_END_DETECT_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

I2C_BYTE_TRANS_DONE_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

I2C_ARBITRATION_LOST_INT_CLR Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WO)

I2C_MST_TXFIFO_UDF_INT_CLR Set this bit to clear I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_TRANS_COMPLETE_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_TIME_OUT_INT_CLR Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WO)

I2C_TRANS_START_INT_CLR Set this bit to clear the I2C_TRANS_START_INT interrupt. (WO)

I2C_NACK_INT_CLR Set this bit to clear I2C_SLAVE_STRETCH_INT interrupt. (WO)

I2C_TXFIFO_OVF_INT_CLR Set this bit to clear I2C_TXFIFO_OVF_INT interrupt. (WO)

I2C_RXFIFO_UDF_INT_CLR Set this bit to clear I2C_RXFIFO_UDF_INT interrupt. (WO)

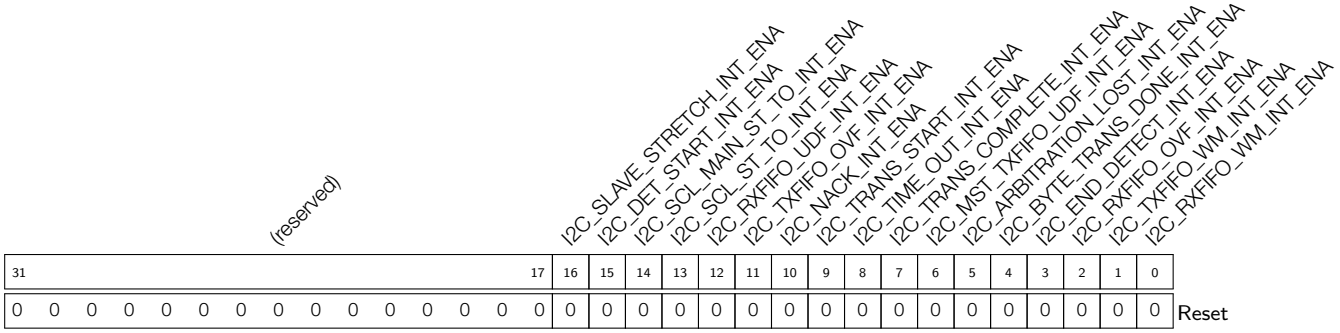
I2C_SCL_ST_TO_INT_CLR Set this bit to clear I2C_SCL_ST_TO_INT interrupt. (WO)

I2C_SCL_MAIN_ST_TO_INT_CLR Set this bit to clear I2C_SCL_MAIN_ST_TO_INT interrupt. (WO)

I2C_DET_START_INT_CLR Set this bit to clear I2C_DET_START_INT interrupt. (WO)

I2C_SLAVE_STRETCH_INT_CLR Set this bit to clear I2C_SLAVE_STRETCH_INT interrupt. (WO)

Register 17.22: I2C_INT_ENA_REG (0x0028)



I2C_RXFIFO_WM_INT_ENA The raw interrupt bit for I2C_RXFIFO_WM_INT interrupt. (R/W)

I2C_TXFIFO_WM_INT_ENA The raw interrupt bit for I2C_TXFIFO_WM_INT interrupt. (R/W)

I2C_RXFIFO_OVF_INT_ENA The raw interrupt bit for I2C_RXFIFO_OVF_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/W)

I2C_BYTE_TRANS_DONE_INT_ENA The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA The raw interrupt bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_MST_TXFIFO_UDF_INT_ENA The raw interrupt bit for I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TRANS_COMPLETE_INT_ENA The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA The raw interrupt bit for the I2C_TIME_OUT_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA The raw interrupt bit for the I2C_TRANS_START_INT interrupt. (R/W)

I2C_NACK_INT_ENA The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (R/W)

I2C_TXFIFO_OVF_INT_ENA The raw interrupt bit for I2C_TXFIFO_OVF_INT interrupt. (R/W)

I2C_RXFIFO_UDF_INT_ENA The raw interrupt bit for I2C_RXFIFO_UDF_INT interrupt. (R/W)

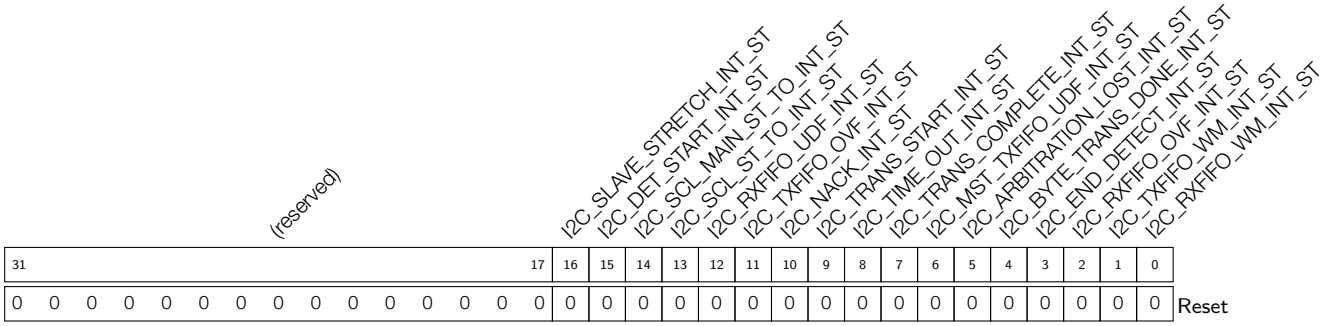
I2C_SCL_ST_TO_INT_ENA The raw interrupt bit for I2C_SCL_ST_TO_INT interrupt. (R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA The raw interrupt bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

I2C_DET_START_INT_ENA The raw interrupt bit for I2C_DET_START_INT interrupt. (R/W)

I2C_SLAVE_STRETCH_INT_ENA The raw interrupt bit for I2C_SLAVE_STRETCH_INT interrupt. (R/W)

Register 17.23: I2C_INT_STATUS_REG (0x002C)



I2C_RXFIFO_WM_INT_ST The masked interrupt status bit for I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_ST The masked interrupt status bit for I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_ST The masked interrupt status bit for 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_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status bit for I2C_TRANS_COMPLETE_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 I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_ST The masked interrupt status bit for I2C_TXFIFO_OVF_INT interrupt. (RO)

I2C_RXFIFO_UDF_INT_ST The masked interrupt status bit for I2C_RXFIFO_UDF_INT interrupt. (RO)

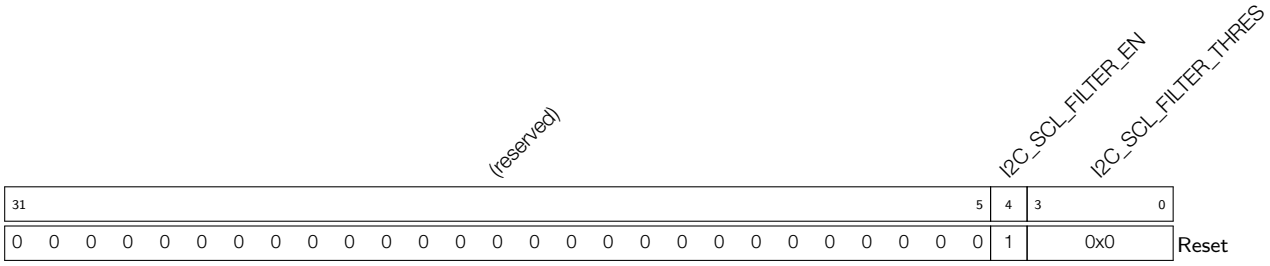
I2C_SCL_ST_TO_INT_ST The masked interrupt status bit for I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status bit for I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

I2C_DET_START_INT_ST The masked interrupt status bit for I2C_DET_START_INT interrupt. (RO)

I2C_SLAVE_STRETCH_INT_ST The masked interrupt status bit for I2C_SLAVE_STRETCH_INT interrupt. (RO)

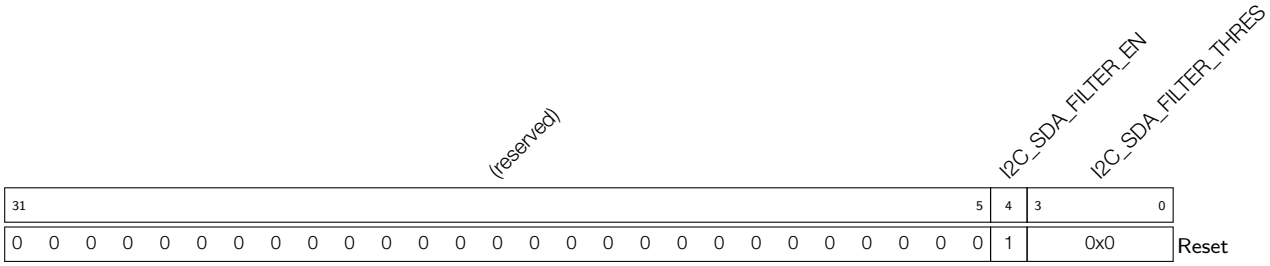
Register 17.24: I2C_SCL_FILTER_CFG_REG (0x0050)



I2C_SCL_FILTER_THRES When a pulse on the SCL input has smaller width than this register value in I2C module clock cycles, the I2C controller will ignore that pulse. (R/W)

I2C_SCL_FILTER_EN This is the filter enable bit for SCL. (R/W)

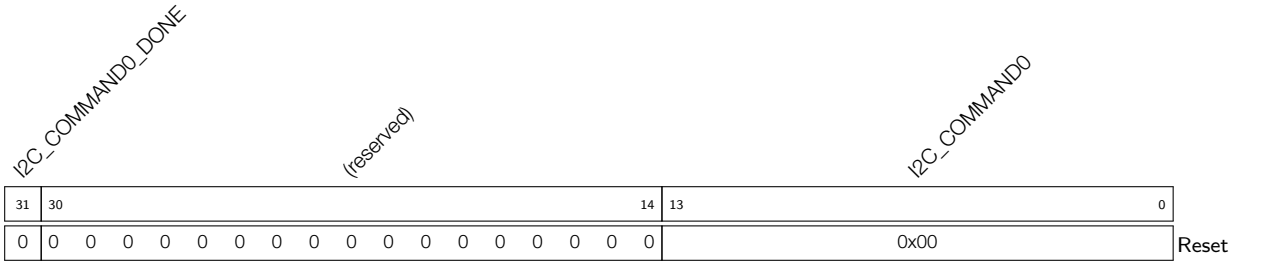
Register 17.25: I2C_SDA_FILTER_CFG_REG (0x0054)



I2C_SDA_FILTER_THRES When a pulse on the SDA input has smaller width than this register value in I2C module clock cycles, the I2C controller will ignore that pulse. (R/W)

I2C_SDA_FILTER_EN This is the filter enable bit for SDA. (R/W)

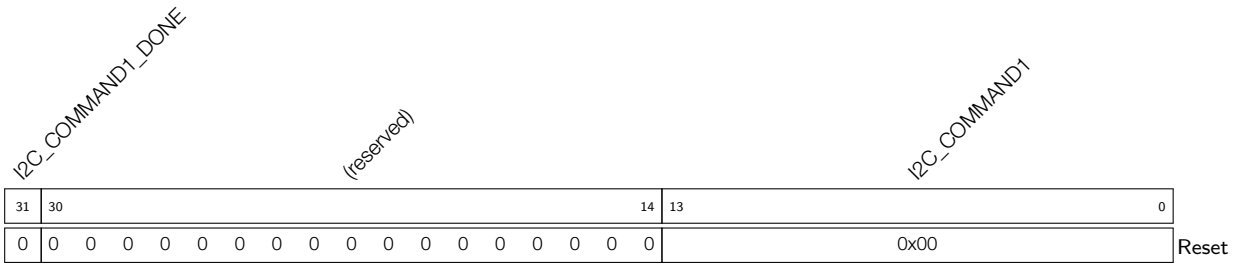
Register 17.26: I2C_COMD0_REG (0x0058)



I2C_COMMAND0 This is the content of command 0. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND0_DONE When command 0 is done in I2C Master mode, this bit changes to high level. (R/W)

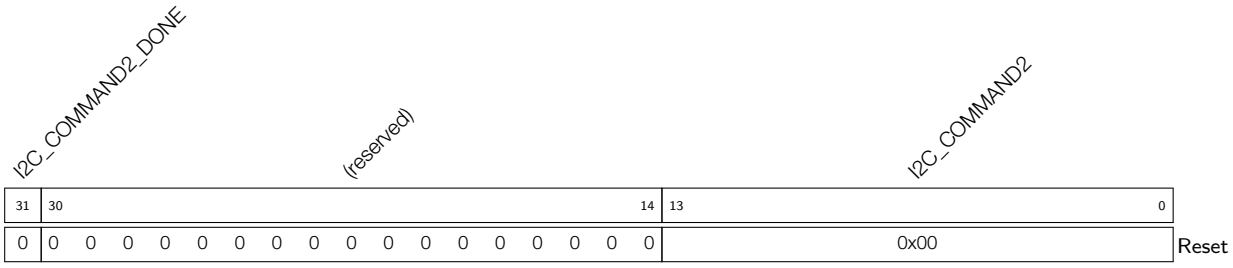
Register 17.27: I2C_COMD1_REG (0x005C)



I2C_COMMAND1 This is the content of command 1. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND1_DONE When command 1 is done in I2C Master mode, this bit changes to high level. (R/W)

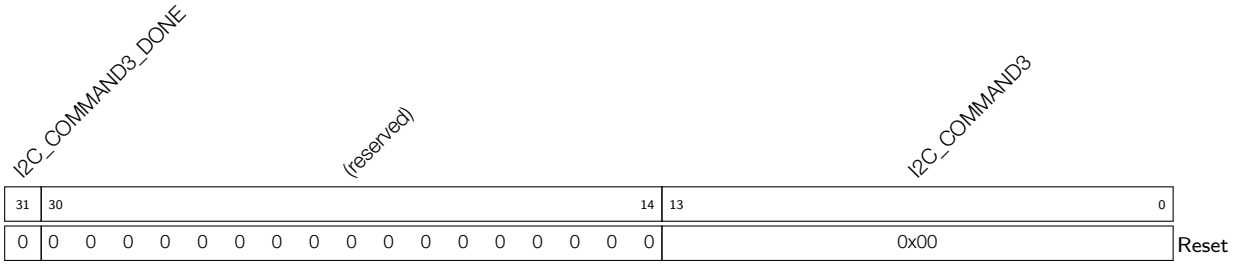
Register 17.28: I2C_COMD2_REG (0x0060)



I2C_COMMAND2 This is the content of command 2. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND2_DONE When command 2 is done in I2C Master mode, this bit changes to high Level. (R/W)

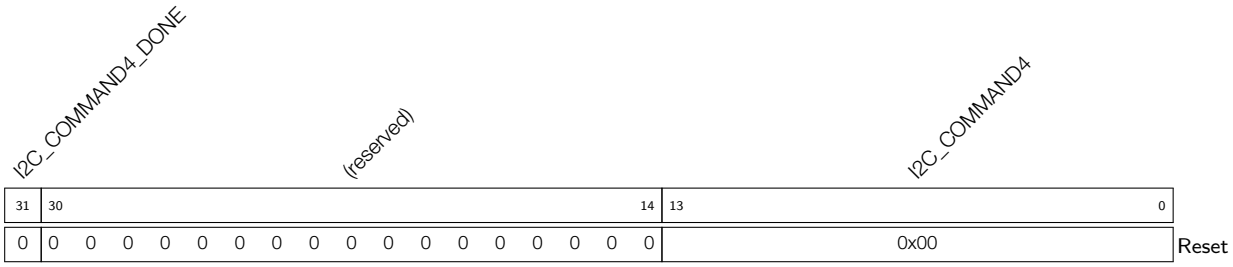
Register 17.29: I2C_COMD3_REG (0x0064)



I2C_COMMAND3 This is the content of command 3. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND3_DONE When command 3 is done in I2C Master mode, this bit changes to high level. (R/W)

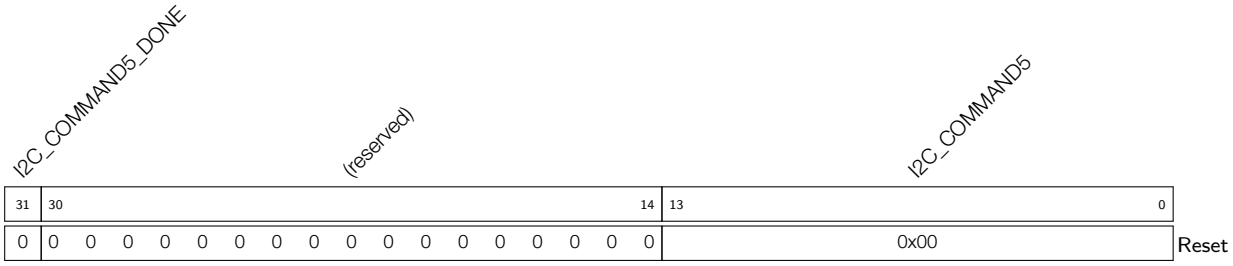
Register 17.30: I2C_COMD4_REG (0x0068)



I2C_COMMAND4 This is the content of command 4. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND4_DONE When command 4 is done in I2C Master mode, this bit changes to high level. (R/W)

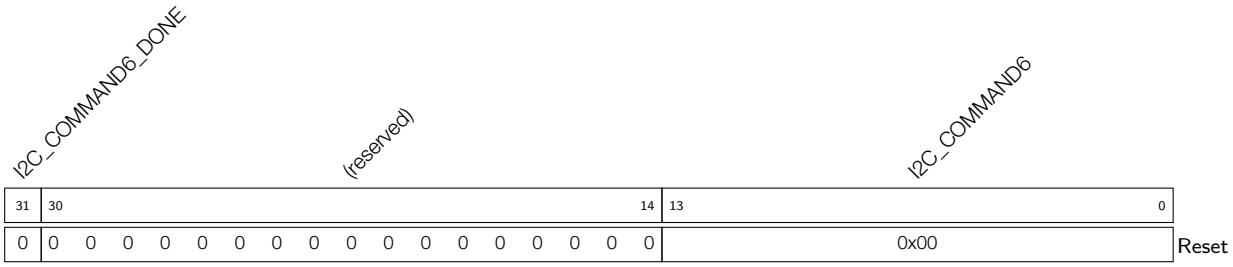
Register 17.31: I2C_COMD5_REG (0x006C)



I2C_COMMAND5 This is the content of command 5. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND5_DONE When command 5 is done in I2C Master mode, this bit changes to high level. (R/W)

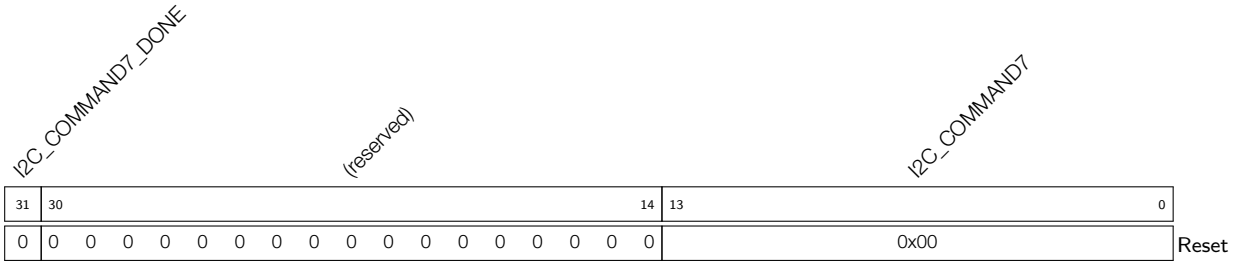
Register 17.32: I2C_COMD6_REG (0x0070)



I2C_COMMAND6 This is the content of command 6. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND6_DONE When command 6 is done in I2C Master mode, this bit changes to high level. (R/W)

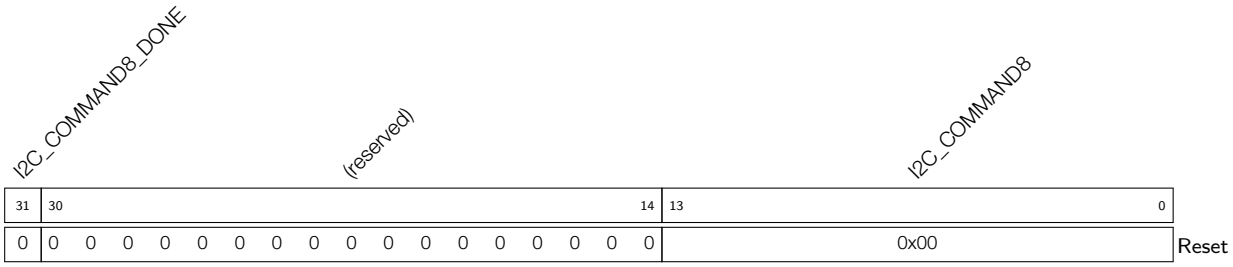
Register 17.33: I2C_COMD7_REG (0x0074)



I2C_COMMAND7 This is the content of command 7. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND7_DONE When command 7 is done in I2C Master mode, this bit changes to high level. (R/W)

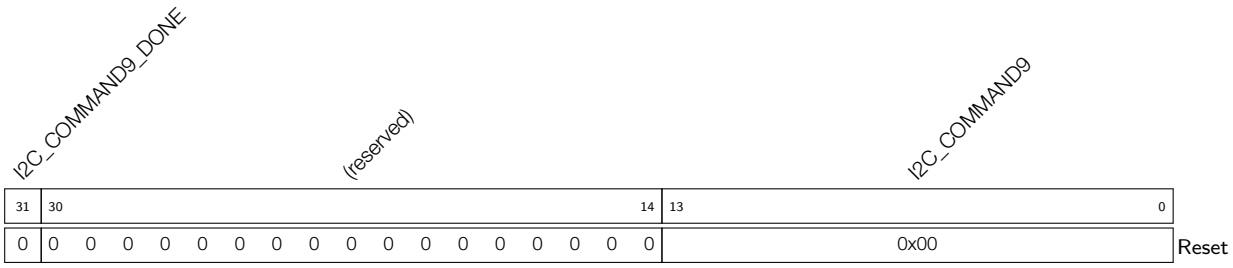
Register 17.34: I2C_COMD8_REG (0x0078)



I2C_COMMAND8 This is the content of command 8. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND8_DONE When command 8 is done in I2C Master mode, this bit changes to high level. (R/W)

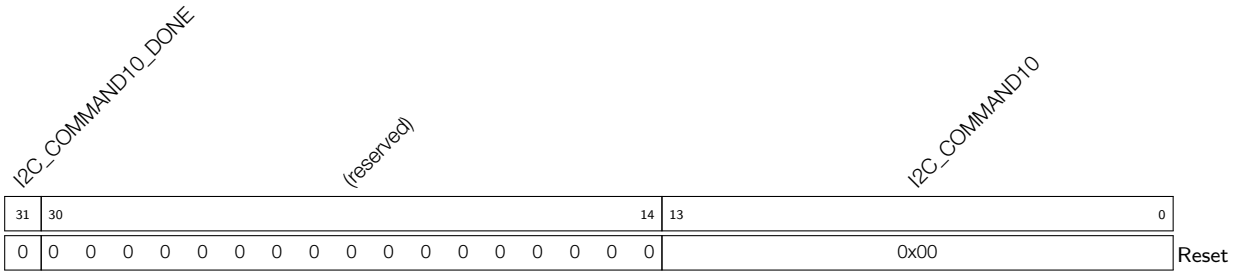
Register 17.35: I2C_COMD9_REG (0x007C)



I2C_COMMAND9 This is the content of command 9. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND9_DONE When command 9 is done in I2C Master mode, this bit changes to high level. (R/W)

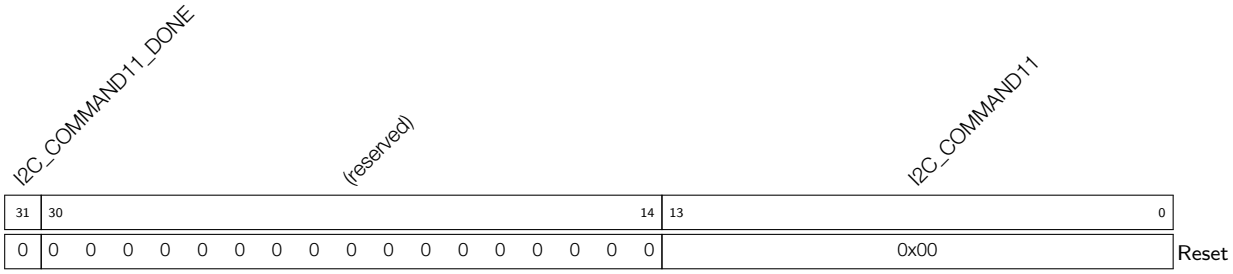
Register 17.36: I2C_COMD10_REG (0x0080)



I2C_COMMAND10 This is the content of command 10. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND10_DONE When command 10 is done in I2C Master mode, this bit changes to high level. (R/W)

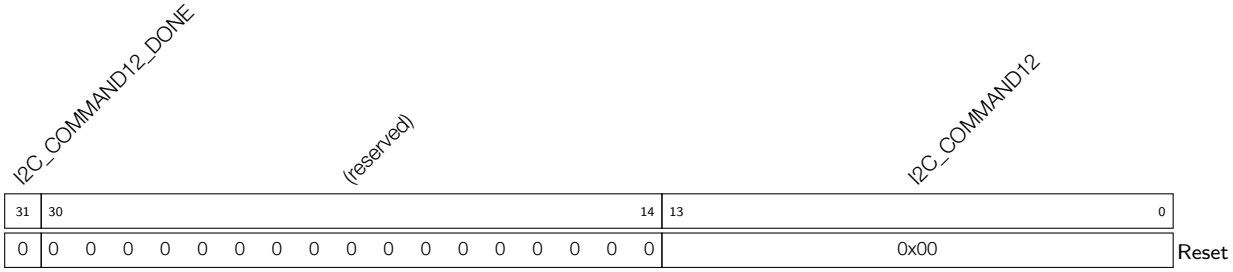
Register 17.37: I2C_COMD11_REG (0x0084)



I2C_COMMAND11 This is the content of command 11. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND11_DONE When command 11 is done in I2C Master mode, this bit changes to high level. (R/W)

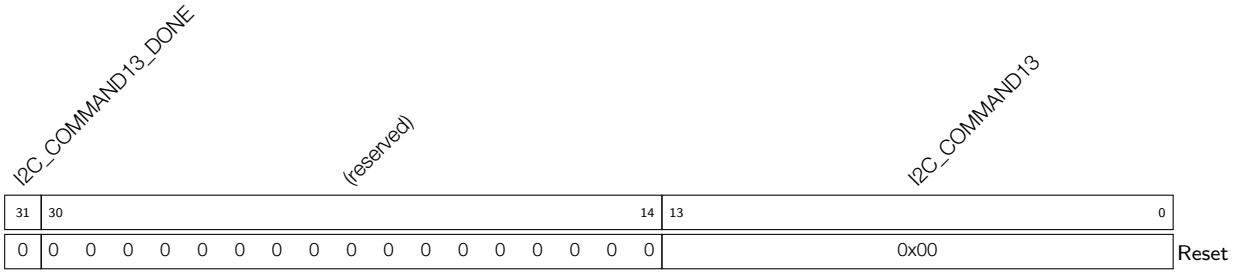
Register 17.38: I2C_COMD12_REG (0x0088)



I2C_COMMAND12 This is the content of command 12. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND12_DONE When command 12 is done in I2C Master mode, this bit changes to high level. (R/W)

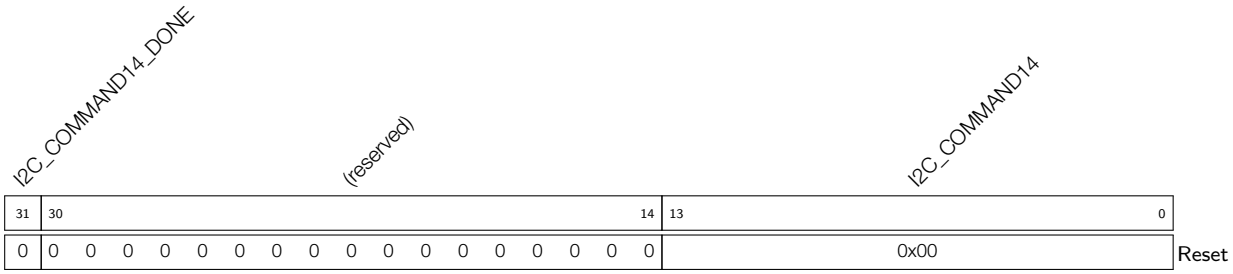
Register 17.39: I2C_COMD13_REG (0x008C)



I2C_COMMAND13 This is the content of command 13. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND13_DONE When command 13 is done in I2C Master mode, this bit changes to high level. (R/W)

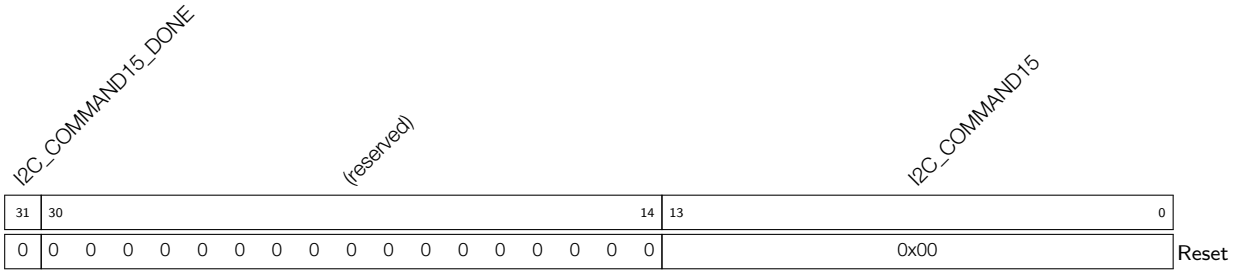
Register 17.40: I2C_COMD14_REG (0x0090)



I2C_COMMAND14 This is the content of command 14. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND14_DONE When command 14 is done in I2C Master mode, this bit changes to high level. (R/W)

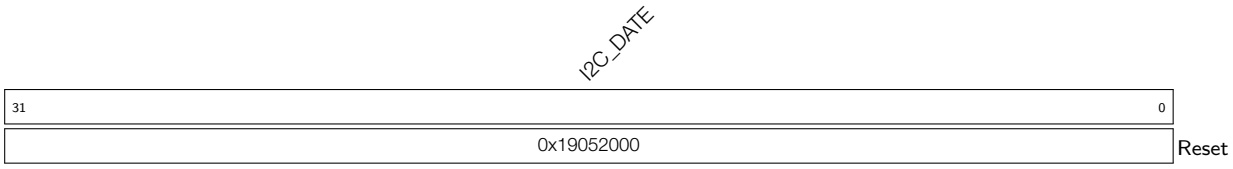
Register 17.41: I2C_COMD15_REG (0x0094)



I2C_COMMAND15 This is the content of command 15. It consists of three parts: op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END. byte_num represents the number of bytes that need to be sent or received. ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more Information. (R/W)

I2C_COMMAND15_DONE When command 15 is done in I2C Master mode, this bit changes to high level. (R/W)

Register 17.42: I2C_DATE_REG (0x00F8)



I2C_DATE This is the the version control register. (R/W)

18. TWAI

18.1 Overview

The Two-Wire Automobile Interface (TWAI) is a multi-master, multi-cast communication protocol with error detection and signaling and inbuilt message priorities and arbitration.The TWAI protocol is suited for automotive and industrial applications (Please see TWAI Protocol Description).

ESP32-S2 contains a TWAI controller that can be connected to a TWAI bus via an external transceiver. The TWAI controller contains numerous advanced features, and can be utilized in a wide range of use cases such as automotive products, industrial automation controls, building automation etc.

18.2 Features

ESP32-S2 TWAI controller supports the following features:

- compatible with ISO 11898-1 protocol
- Supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation
  - Normal
  - Listen Only (no influence on bus)
  - Self Test (transmissions do not require acknowledgment)
- 64-byte Receive FIFO
- Special transmissions
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self Reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports single and dual filter modes)
- Error detection and handling
  - Error counters
  - Configurable Error Warning Limit
  - Error Code Capture
  - Arbitration Lost Capture

18.3 Functional Protocol

18.3.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows for nodes to exchange messages in a latency bounded manner. A TWAI bus will has a following properties.

Single Channel and Non-Return-to-Zero: The bus consists of a single channel to carry bits, thus communication is half-duplex. Synchronization is also derived from this channel, thus extra channels (e.g., clock or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a Dominant or Recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting a Dominant state will always override a another node transmitting a Recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential wiring).

Bit-Stuffing: Certain fields of TWAI messages are bit-stuffed. A Transmitter that transmits five consecutive bits of the same value should automatically insert a complementary bit. Likewise, a Receiver that receives five consecutive bits should treat the next bit as a stuff bit. Bit stuffing is applied to the following fields: SOF, Arbitration Field, Control Field, Data Field, and CRC Sequence (see Section 18.3.2 for more details).

Multi-cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (See Section 18.3.3).

Multi-master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before beginning its own transmission.

Message-Priorities and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The Arbitration Field of the message transmitted by each node is used to determine which node will win arbitration.

Error Detection and Signaling: Each node will actively monitor the bus for errors, and signal the detection errors by transmitting an Error Frame.

Fault Confinement: Each node will maintain a set of error counts that are incremented/decremented according to a set of rules. When the error counts surpass a certain threshold, a node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes within the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a Transmitter or a Receiver.

- A node originating a message is a Transmitter. The node remains a Transmitter until the bus is idle or until the node loses arbitration. Note that multiple nodes can be Transmitter if they have yet to lose arbitration.
- All nodes that are not Transmitters are Receivers.

18.3.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes. Messages are split into various frame types, and some frame types will have different frame formats. The TWAI protocol has of the following frame types:

- Data Frames

- Remote Frames
- Error Frames
- Overload Frames
- Interframe Space

The TWAI protocol has the following frame formats:

- Standard Frame Format (SFF) that consists of a 11-bit identifier
- Extended Frame Format (EFF) that consists of a 29-bit identifier

18.3.2.1 Data Frames and Remote Frames

Data Frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote Frames are used to nodes to request a Data Frame with the same Identifier from another node, thus does not contain any data bytes. However, Data Frames and Remote Frames share many common fields. Figure 18-1 illustrates the fields and sub fields of the different frames and formats.

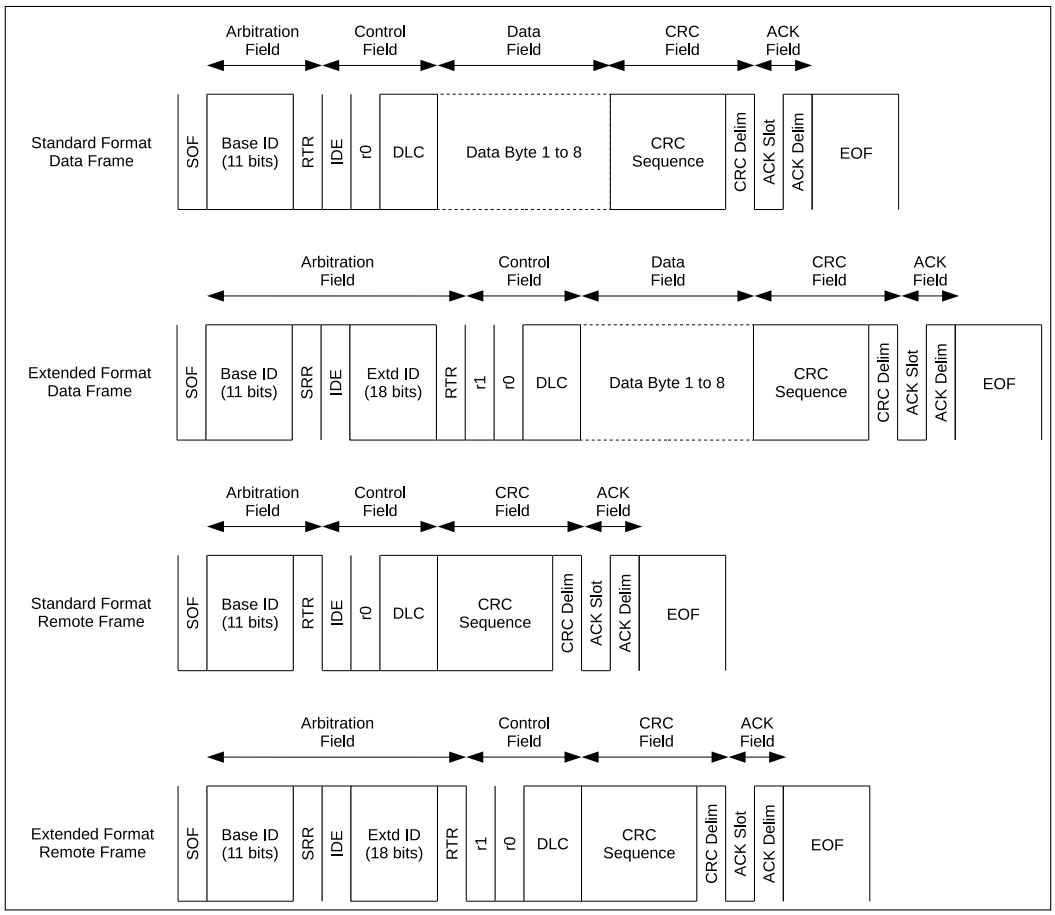


Figure 18-1. The bit fields of Data Frames and Remote Frames

Arbitration Field

When two or more nodes transmits a Data or Remote Frame simultaneously, the Arbitration Field is used to determine which node will win arbitration of the bus. During the Arbitration Field, if a node transmits a Recessive bit but observes a Dominant bit, this indicates that another node has overridden its Recessive bit. Therefore, the

node transmitting the Recessive bit has lost arbitration of the bus and should immediately become a Receiver.

The Arbitration Field primarily consists of the Frame Identifier that is transmitted most significant bit first. Given that a Dominant bit represents a logical 0, and a Recessive bit represents a logical 1:

- A frame with the smallest ID value will always win arbitration.
- Given the same ID and format, Data Frames will always prevail over RTR Frames.
- Given the same first 11 bits of ID, a Standard Format Data Frame will prevail over an Extended Format Data Frame due to the SRR being recessive.

Control Field

The control field primarily consists of the DLC (Data Length Code) which indicates the number of payload data bytes for a Data Frame, or the number of requested data bytes for a Remote Frame. The DLC is transmitted most significant bit first.

Data Field

The Data Field contains the actual payload data bytes of a Data Frame. Remote Frames do not contain a Data Field.

CRC Field

The CRC Field primarily consists of a a CRC Sequence. The CRC Sequence is a 15-bit cyclic redundancy code calculated form the de-stuffed contents (everything from the SOF to the end of the Data Field) of a Data or Remote Frame.

Table 65: Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single Dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit Identifier for SFF, or the first 11-bits of the 29-bit Identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a Data Frame (Dominant) or a Remote Frame (Recessive). This means that a Remote Frame will always lose arbitration to a Data Frame given they have the same ID.
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The IDE (Identifier Extension) bit indicates whether the message is SFF (Dominant) or EFF (Recessive). This means that a SFF frame will always win arbitration over an EFF frame given they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18-bits of the 29-bit identifier for EFF.
r1	The r1 (reserved bit 1) is always Dominant.
r0	The r0 (reserved bit 0) is always Dominant.
DLC	The DLC (Data Length Code) is 4-bits and should have a value from 0 to 8. Data Frames use the DLC to indicate the number data bytes in the Data Frame. Remote Frames used the DLC to indicate the number of data bytes to request from another node.

Data/Remote Frames	Description
Data Bytes	The data payload of Data Frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted most significant bit first.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.
CRC Delim	The CRC Delim (CRC Delimeter) is a single Recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) that intended for Receiver nodes to indicate that the Data or Remote Frame was received without issue. The Transmitter node will send a Recessive bit in the ACK Slot and Receiver nodes should override the ACK Slot with a Dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimeter) is a single Recessive bit.
EOF	The EOF (End of Frame) marks the end of a Data or Remote Frame, and consists of seven Recessive bits.

18.3.2.2 Error and Overload Frames

Error Frames

Error Frames are transmitted when a node detects a Bus Error. Error Frames notably consist of an Error Flag which is made up of 6 consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a Bus Error and transmits an Error Frame, all other nodes will then detect a Stuff Error and transmit their own Error Frames in response. This has the effect of propagating the detection of a Bus Error across all nodes on the bus. When a node detects a Bus Error, it will transmit an Error Frame starting on the next bit. However, if the type of Bus Error was a CRC Error, then the Error Frame will start at the bit following the ACK Delim (see Section 18.3.3). The following Figure 18-2 shows the various fields of an Error Frame:

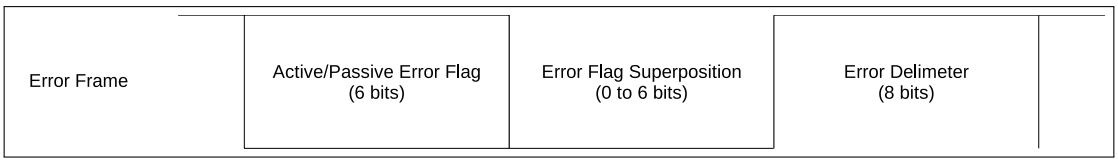


Figure 18-2. Various Fields of an Error Frame

Table 66: Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of 6 Dominant bits and the Passive Error Flag consisting of 6 Recessive bits (unless overridden by Dominant bits of other nodes). Active Error Flags are sent by Error Active nodes, whilst Passive Error Flags are sent by Error Passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first Recessive bit is detected (i.e., the first it of the Delimeter).
Error Delimeter	The Delimeter field marks the end of the Error/Overload Frame, and consists of 8 Recessive bits.

Overload Frames

An Overload Frame has the same bit fields as an Error Frame containing an Active Error Flag. The key difference is in the conditions that can trigger the transmission of an Overload Frame. Figure 18-3 below shows the bit fields of an Overload Frame.

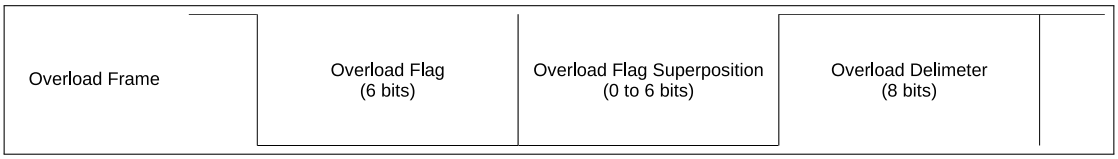


Figure 18-3. The Bit Fields of an Overload Frame

Table 67: Overload Frame

Overload Frame	Description
Overload Flag	Consists of 6 Dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimeter	Consists of 8 Recessive. Same as an Error Delimeter.

Overload Frames will be transmitted under the following conditions:

1. The internal conditions of a Receiver requires a delay of the next Data or Remote Frame.
2. Detection of a Dominant bit at the first and second bit of Intermission.
3. If a Dominant bit is detected at the eighth (last) bit of an Error Delimeter. Note that in this case, TEC and REC will not be incremented (See Section 18.3.3).

   Transmitting an overload frame due to one of the conditions must also satisfy the following rules:

   - Transmitting an Overload Frame due to condition 1 must only be started at the first bit of Intermission.
   - Transmitting an Overload Frame due to condition 2 and 3 must start one bit after the detecting the Dominant bit of the condition.
   - A maximum of two Overload frames may be generated in order to delay the next Data or Remote Frame.

18.3.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Data Frames and Remote Frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (Data Frame, Remote Frame, Error Frame, Overload Frame). However, Error Frames and Overload Frames do not need to be separated from preceding frames.

Figure 18-4 shows the fields within an Interframe Space:

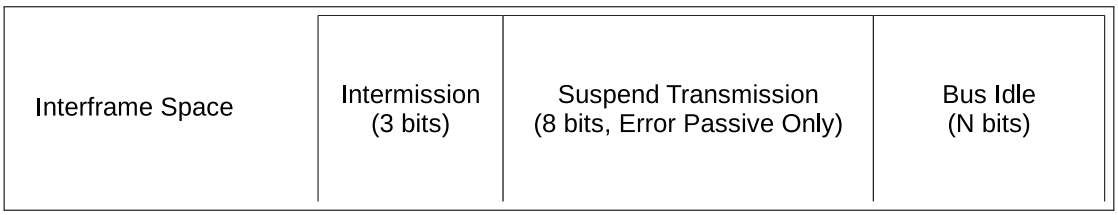


Figure 18-4. The Fields within an Interframe Space

Table 68: Interframe Space

Interface Space	Description
Intermission	The Intermission consists of 3 Recessive bits.
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of 8 Recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following Intermission.

18.3.3 TWAI Errors

18.3.3.1 Error Types

Bus Errors in TWAI are categorized into one of the following types:

Bit Error

A Bit Error occurs when a node transmits a bit value (i.e., Dominant or Recessive) but the opposite bit is detected (e.g., a Dominant bit is transmitted but a Recessive is detected). However, if the transmitted bit is Recessive and is located in the Arbitration Field or ACK Slot or Passive Error Flag, then detecting a Dominant bit will not be considered a Bit Error.

Stuff Error

A stuff error is detected when 6 consecutive bits of the same value are detected (thus violating the bit-stuffing encoding).

CRC Error

A Receiver of a Data or Remote Frame will calculate a CRC based on the bits it has received. A CRC error occurs when the CRC calculated by the Receiver does not match the CRC sequence in the received Data or Remote Frame.

Form Error

A Form Error is detected when a fixed-form bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be Dominant.

Acknowledgement Error

An Acknowledgment Error occurs when a Transmitter does not detect a Dominant bit at the ACK Slot.

18.3.3.2 Error States

TWAI nodes implement fault confinement by each maintaining two error counters, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states.

Error Active

An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication, but can only transmit an Passive Error Flag when it detects an error. Error Passive nodes that have transmitted a Data or Remote Frame must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit anything).

18.3.3.3 Error Counters

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply for a given message transfer.**

1. When a Receiver detects an error, the REC will be increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a Receiver detects a Dominant bit as the first bit after sending an Error Flag, the REC will be increased by 8.
3. When a Transmitter sends an Error Flag the TEC is increased by 8. However, the following scenarios are exempt form this rule:
    - If a Transmitter is Error Passive that detects an Acknowledgment Error due to not detecting a Dominant bit in the ACK slot, it should send a Passive Error Flag. If no Dominant bit is detected in that Passive Error Flag, the TEC should not be increased.
    - A Transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the offending bit should have been Recessive but was monitored as Dominant, then the TEC should not be increased.
4. If a Transmitter detects a Bit Error whilst sending an Active Error Flag or Overload Flag, the REC is increased by 8.
5. If a Receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
6. Any node tolerates up to 7 consecutive Dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive Dominant bit (when sending an Active Error Flag or Overload Flag), or the 8th consecutive Dominant bit following a Passive Error Flag, a Transmitter will increase its TEC by 8 and a Receiver will increase its REC by 8. Each additional eight consecutive Dominant bits will also increase the TEC (for Transmitters) or REC (for Receivers) by 8 as well.
7. When a Transmitter successfully transmits a message (getting ACK and no errors until the EOF is complete), the TEC is decremented by 1, unless the TEC is already at 0.
8. When a Receiver successfully receives a message (no errors before ACK Slot, and successful sending of ACK), the REC is decremented.
    - If the REC was between 1 and 127, the REC is decremented by 1.
    - If the REC was greater than 127, the REC is set to 127.

- If the REC was 0, the REC remains 0.

9. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. The error condition that causes a node to become Error Passive will cause the node to send an Active Error Flag. Note that once the REC has reached to 128, any further increases to its value are irrelevant until the REC returns to a value less than 128.
10. A node becomes Bus Off when its TEC is greater than or equal to 256.
11. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
12. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive Recessive bits on the bus.

18.3.4 TWAI Bit Timing

18.3.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- The **Nominal Bit Rate** is defined as number of bits transmitted per second from an ideal Transmitter and without any synchronization.
- The **Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a fixed unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 18-5 illustrates the segments within a single Nominal Bit Time.

TWAI Controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed at every Time Quanta. If two consecutive Time Quantas have different bus states (i.e., Recessive to Dominant or vice versa), this will be considered an edge. When the bus is analyzed at the intersection of PBS1 and PBS2, this is considered the Sample Point and the sampled bus value is considered the value of that bit.

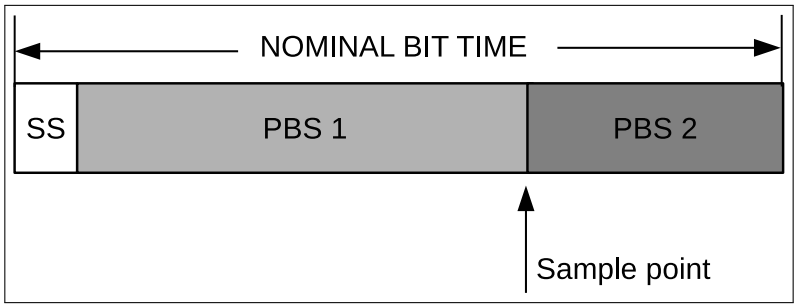


Figure 18-5. Layout of a Bit

Table 69: Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.

Segment	Description
PBS2	PBS2 (Phase Buffer Segment 2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

18.3.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct for Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules.

- Only one synchronization may occur in a single bit time.
- Synchronizations only occurs on Recessive to Dominant edges.

Hard Synchronization

Hard Synchronization occurs on the Recessive to Dominant edges during Bus Idle (i.e., the SOF bit). All nodes will restart their internal bit timings such that the Recessive to Dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on Recessive to Dominant edges not during Bus Idle. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is a programmable.

- When the magnitude of the Phase Error is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by **e** number of Time Quanta. This has a same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater to the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

18.4 Architectural Overview

The ESP32-S2 contains a TWAI Controller. Figure 18-6 show show the major functional blocks of the TWAI Controller.

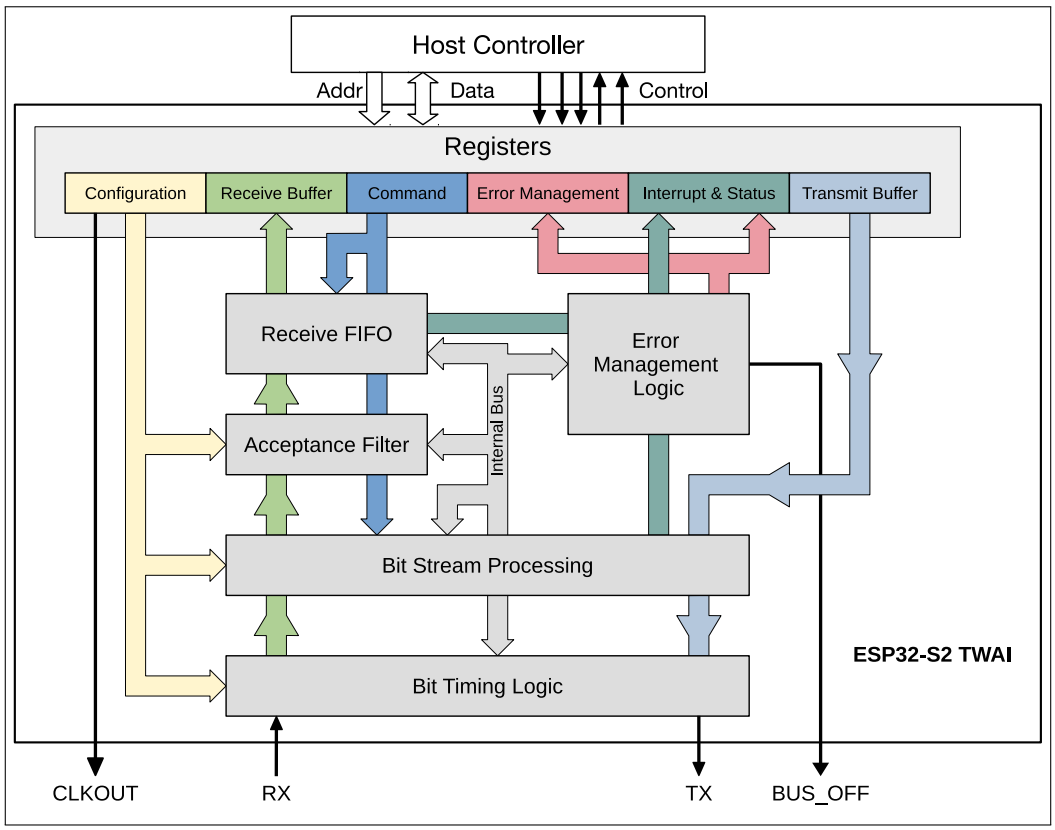


Figure 18-6. TWAI Overview Diagram

18.4.1 Registers Block

The ESP32-S2 CPU accesses peripherals as 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration options for the TWAI controller such as bit rates, operating mode, Acceptance Filter etc. Configuration registers can only be modified whilst the TWAI controller is in Reset Mode (See Section 18.5.1).

Command Register

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation Mode (see section 18.5.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management registers include error counters and capture registers. The error counter registers represent TEC and REC values. The capture registers will record information about instances where TWAI controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window into Receive FIFO mapping to the first received message in the Receive FIFO to the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset Mode, the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation Mode:
  - All reads to the address range maps to the Receive Buffer registers.
  - All writes to the address range maps to the Transmit Buffer registers.

18.4.2 Bit Stream Processor

The Bit Stream Processing (BSP) module is responsible for framing data from the Transmit Buffer (e.g. bit stuffing and additional CRC fields) and generating a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

18.4.3 Error Management Logic

The Error Management Logic (EML) module is responsible for updating the TEC and REC, recording error information like error types and positions, and updating the error state of the TWAI Controller such that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

18.4.4 Bit Timing Logic

The Bit Timing Logic (BTL) module is responsible for transmitting and receiving messages at the configured bit rate. The BTL module also handles synchronization of out of phase bits such that communication remains stable. A single bit time consists of multiple programmable segments that allows users to set the length of each segment to account for factors such as propagation delay and controller processing time etc.

18.4.5 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter's registers can be programmed to specify a single filter, or specify two separate filters (dual filter mode).

18.4.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (internal to the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13-bytes). When the Receive FIFO is full (or does not have enough space to store the next received message in its entirety), the Overrun Interrupt will be triggered, and any subsequent received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After clearing, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

18.5 Functional Description

18.5.1 Modes

The ESP32-S2 TWAI controller has two working modes: Reset Mode and Operation Mode. Reset Mode and Operation Mode are entered by setting the TWAI_RESET_MODE bit to 1 or 0 respectively.

18.5.1.1 Reset Mode

Entering Reset Mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset Mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset Mode, the TWAI controller will not be able to transmit any messages (including error signaling). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will also not be able to receive any messages.

18.5.1.2 Operation Mode

Entering Operation Mode essentially connects the TWAI controller to the TWAI bus, and write protects the TWAI controller's configuration registers ensuring the configuration stays consistent during operation. When in Operation Mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operating sub-mode the TWAI controller was configured with. The TWAI controller supports the following operating sub-modes:

- **Normal Mode:** The TWAI controller can transmit and receive messages including error signaling (such as Error and Overload Frames).
- **Self Test Mode:** Like Normal Mode, but the TWAI controller will consider the transmission of a Data or RTR Frame successful even if it was not acknowledged. This is commonly used when self testing the TWAI controller.
- **Listen Only Mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitors.

Note that when exiting Reset Mode (i.e., entering Operation Mode), the TWAI controller must wait for 11 consecutive Recessive bits to occur before being able to fully connect the TWAI bus (i.e., be able to transmit or receive).

18.5.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset Mode. The bit rate configuration is located in TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 70 illustrates the bit fields of TWAI_BUS_TIMING_0_REG.

Table 70: Bit Information of TWAI_CLOCK_DIVIDER_REG; TWAI Address 0x18

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

Notes:

- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1).
- BRP: The TWAI Time Quanta clock is derived from a prescaled version of the APB clock that is usually 80 MHz. The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock period and t_{CLK} is APB clock period :
  $t_{Tq} = 2 \times t_{CLK} \times (2^{13} \times BRP.13 + 2^{12} \times BRP.12 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$

The following Table 71 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 71: Bit Information of TWAI_BUS_TIMING_1_REG; TWAI Address 0x1c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined according to the following equation: (8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1).
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined according to the following equation: (4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1).
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses where filtering spikes on the bus line is beneficial.

18.5.3 Interrupt Management

The ESP32-S2 TWAI controller provides seven interrupts, each represented by a single bit in the TWAI_INT_RAW_REG. For a particular interrupt to be triggered (i.e., its bit in TWAI_INT_RAW_REG set to 1), the interrupt's corresponding enable bit in TWAI_INT ENA_REG must be set.

The TWAI controller provides the seven following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt
- Arbitration Lost Interrupt
- Bus Error Interrupt

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INT_RAW_REG, and deasserted when all bits in TWAI_INT_RAW_REG are cleared. The majority of interrupt bits in TWAI_INT_RAW_REG are automatically cleared when the register is read. However, the Receive Interrupt is an exception and can only be cleared the Receive FIFO is empty.

18.5.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_CNT_REG > 0). Pending received messages

includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUF command bit.

18.5.3.2 Transmit Interrupt (TXI)

The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer to be transmitted. The Transmit Buffer becomes free under the following scenarios:

- A message transmission has completed successfully (i.e., Acknowledged without any errors). Any failed messages will automatically be retried.
- A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_TX_COMPLETE bit).
- A message transmission was aborted using the TWAI_ABORT_TX command bit.

18.5.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST or TWAI_BUS_OFF_ST bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_ERR_ST or TWAI_BUS_OFF_ST at the moment the EWI is triggered.

- If TWAI_ERR_ST = 0 or TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
  - If the TWAI controller was previously in the Bus Recovery state, it indicates that Bus Recovery has completed successfully.
- If TWAI_ERR_ST = 1 or TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 or TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).
- If TWAI_ERR_ST = 0 or TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

18.5.3.4 Data Overrun Interrupt (DOI)

The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages.

The DOI is only triggered on the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI will only be able to trigger again when all received messages (valid or overrun) have been cleared.

18.5.3.5 Error Passive Interrupt (TXI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller transitions from Error Active to Error Passive, or vice versa.

18.5.3.6 Arbitration Lost Interrupt (ALI)

The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB LOST CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record new bit location until it is cleared (via a read from the CPU).

18.5.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever TWAI controller observes an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).

18.5.4 Transmit and Receive Buffers

18.5.4.1 Overview of Buffers

Table 72: Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
TWAI address	Content	TWAI address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 72 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation Mode. CPU write operations will access the Transmit Buffer registers, and CPU read operations will access the Receive Buffer registers. However, both buffers share the exact same register layout and fields to represent a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame

format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQ bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQ bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQ and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map to the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUF bit in TWAI_CMD_REG so that the next message in the Receive FIFO will be loaded in to the Receive Buffer registers.

18.5.4.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 73.

Table 73: TX/RX Frame Information (SFF/EFF) TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	XDLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

- FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when FF bit is 1, and SFF when FF bit is 0.
- RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a Data Frame or a Remote Frame. The message is a Remote Frame when the RTR bit is 1, and a Data Frame when the RTR bit is 0.
- DLC: The Data Length Code (DLC) field specifies the number of data bytes for a Data Frame, or the number of data bytes to request in a Remote Frame. TWAI Data Frames are limited to a maximum payload of 8 data bytes, thus the DLC should range anywhere from 0 to 8.
- X: Don't care, can be any value.

18.5.4.3 Frame Identifier

The Frame Identifier fields is 2 bytes (11-bits) if the message is SFF, and 4 bytes (29-bits) if the message is EFF.

The Frame Identifier fields for an SFF (11-bits) message is shown in Table 74-75.

Table 74: TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 75: TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	X[1]	X[2]	X[2]	X[2]	X[2]

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 76-79.

Table 76: TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 77: TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 78: TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 79: TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

18.5.4.4 Frame Data

The Frame Data fields contains the payload of transmitted or received a Data Frame, and can range from 0 to 8 bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than 8, the number of valid bytes would still be limited to 8. Remote Frames do not have data payloads, thus the Frame Data fields will be unused.

For example, when transmitting a Data Frame with 5 data bytes, the CPU should write a value of 5 to the DLC field, and then fill in data bytes 1 to 5 in the Frame Data fields. Likewise, when receiving a Data Frame with a DLC of 5, only data bytes 1 to 5 will contain valid payload data for the CPU to read.

18.5.5 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13-bytes of space in the Receive FIFO, and their byte layout is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO. When the TWAI controller receives a message, it will increment the value of TWAI_RX_MESSAGE_COUNTER up to a maximum of 64. If there is adequate space in the Receive FIFO, the message contents will be written into the Receive FIFO. Once a message has been read from the Receive Buffer, the TWAI_RELEASE_BUF bit should be set. This will decrement TWAI_RX_MESSAGE_COUNTER and free the space occupied by the first message in the Receive FIFO. The

Receive Buffer will then map to the next message in the Receive FIFO. A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks the adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs...

- Whatever free space is left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- On the first message that causes the Receive FIFO to overrun, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The RX FIFO will internally mark overrun messages as invalid. The TWAI_MISS_ST bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUF must be called repeatedly called until TWAI_RX_MESSAGE_COUNTER is 0. This has the effect of freeing all valid messages in the Receive FIFO and clearing all overrun messages.

18.5.6 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows for a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) due to the TWAI Controller only needing to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset Mode, due to those registers sharing the same address space as the Transmit Buffer and Receive Buffer registers.

The registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Code value specifies a bit pattern in which each filtered bit of the message must match in order for the message to be accepted. The Mask value is able to mask out certain bits of the Code value (i.e., set as "Don't Care" bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 18-7.

The TWAI Controller Acceptance Filter allows the 32-bit Code and Mask values to either define a single filter (i.e., Single Filter Mode), or two filters (i.e., Dual Filter Mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on whether Single Filter Mode is enabled, and the received message (i.e., SFF or EFF).

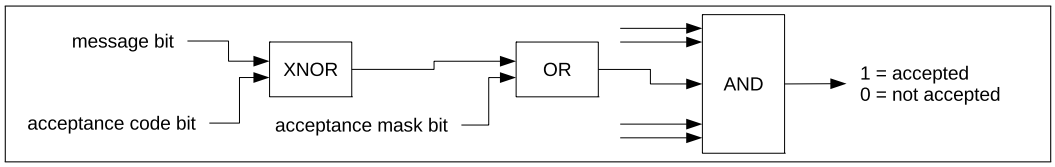


Figure 18-7. Acceptance Filter

18.5.6.1 Single Filter Mode

Single Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of a Data or Remote Frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 18-8 illustrates how the 32-bit code and mask values will be interpreted under Single Filter Mode.

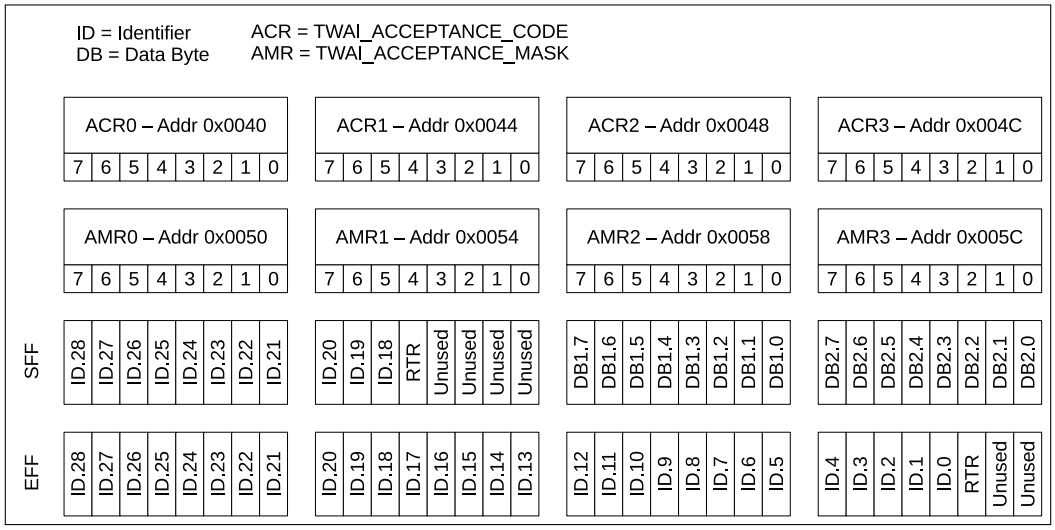


Figure 18-8. Single Filter Mode

18.5.6.2 Dual FIlter Mode

Dual Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define a two separate filters, referred to as filter 1 or two. Under Dual Filter Mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of a Data or Remote Frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)
- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 18-9 illustrates how the 32-bit code and mask values will be interpreted under Dual Filter Mode.

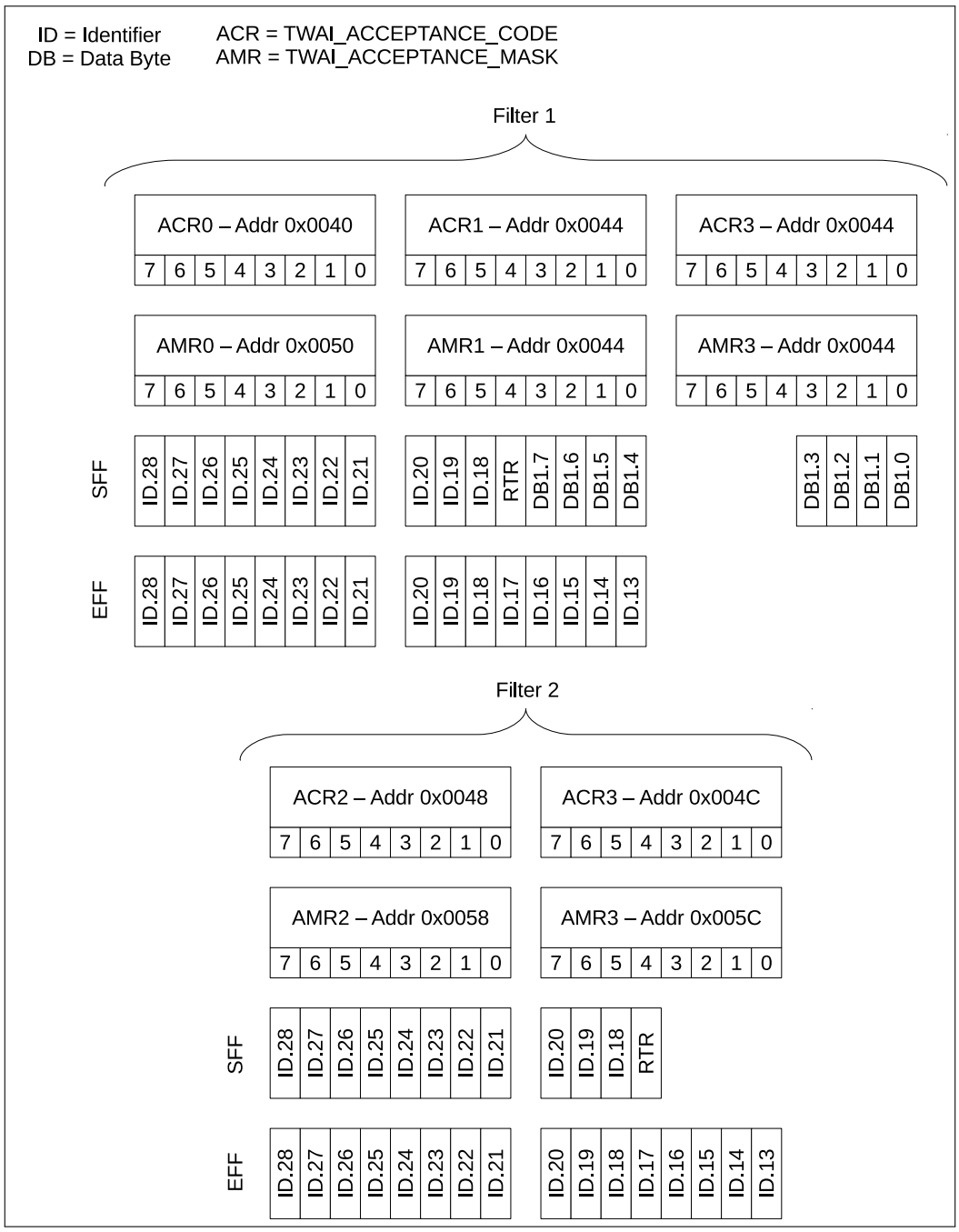


Figure 18-9. Dual Filter Mode

18.5.7 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Count (TEC) and Receive Error Count (REC). The value of both error counts determine the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in the TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and can be read by the CPU at anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn the user regarding the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_ERR_ST, and TWAI_BUS_OFF_ST. Certain changes to these values and bits will also trigger

interrupts, thus allowing the users to be notified of error state transitions (see section 18.5.3). The following figure 18-10 shows the relation between the error states, values and bits, and error state related interrupts.

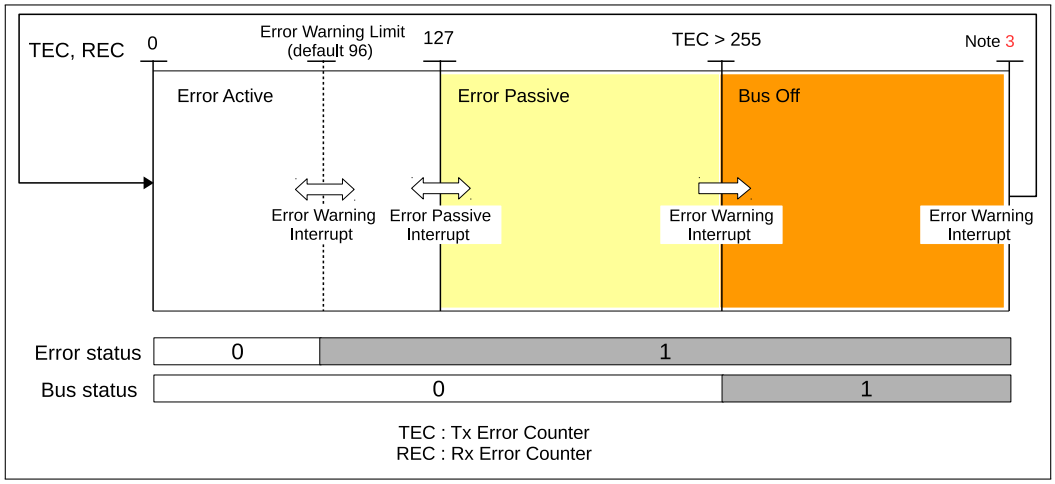


Figure 18-10. Error State Transition

18.5.7.1 Error Warning Limit

The Error Warning Limit (EWL) feature is a configurable threshold value for the TEC and REC, where if exceeded, will trigger an interrupt. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in the TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset Mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96. When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_ERR_ST bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_ERR_ST bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_ERR_ST bit (or the TWAI_BUS_OFF_ST) changes.

18.5.7.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

18.5.7.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_BUS_OFF_ST bit to 1
- Enter Reset Mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_BUS_OFF_ST bit (or the TWAI_ERR_ST bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off recovery. Bus-Off recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive Recessive bits on the bus. To initiate Bus-Off recovery (after entering the Bus-Off state), the TWAI controller should enter Operation Mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off recovery by decrementing the TEC each time the TWAI controller observes 11 consecutive Recessive bits. When Bus-Off recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

18.5.8 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in the TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from the TWAI_ERR_CODE_CAP_REG.

The following Table 80 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 80: Bit Information of TWAI_ERR_CODE_CAP_REG; TWAI Address 0x30

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error: 00 for bit error, 01 for form error, 10 for stuff error, 11 for other type of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error: 0 for Transmitter, 1 for Receiver.
- SEG: The Error Segment (SEG) indicates which segment of the TWAI message (i.e., bit position) the bus error occurred at.

The following Table 81 shows how to interpret the SEG.0 to SEG.4 bits.

Table 81: Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	bit SRTR[1]
0	0	1	0	1	bit IDE[2]
0	0	1	1	1	ID.17 to ID.13
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	acknowledge slot
1	1	0	1	1	acknowledge delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit RTR: under Standard Frame Format.
- Identifier Extension Bit: 0 for Standard Frame Format.

18.5.9 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in the TWAI_ARB LOST CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent loses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in the TWAI_ARB LOST CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 82 illustrates the bit fields of the TWAI_ERR_CODE_CAP_REG whilst Figure 18-11 illustrates the bit positions of a TWAI message.

Table 82: Bit Information of TWAI_ARB LOST CAP_REG; TWAI Address 0x2c

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

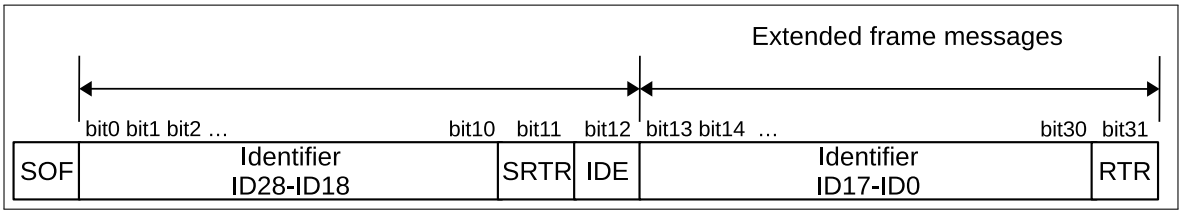


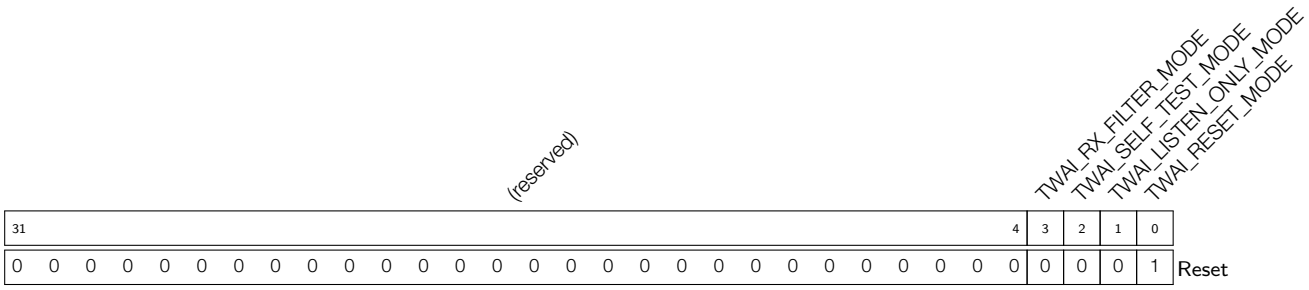
Figure 18-11. Positions of Arbitration Lost Bits

18.6 Register Summary

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	RO | R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	RO | R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	RO | R/W
TWAI_DATA_0_REG	Data Register 0	0x0040	WO | R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	WO | R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	WO | R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	WO | R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	WO | R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	WO | R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	WO | R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	WO | R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	WO | RO
TWAI_DATA_9_REG	Data Register 9	0x0064	WO | RO
TWAI_DATA_10_REG	Data Register 10	0x0068	WO | RO
TWAI_DATA_11_REG	Data Register 11	0x006C	WO | RO
TWAI_DATA_12_REG	Data Register 12	0x0070	WO | RO
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	varies
Control Registers			
TWAI_CMD_REG	Command Register	0x0004	WO
Status Registers			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB LOST CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	RO | R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	RO | R/W
TWAI_RX_MESSAGE_CNT_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INT_RAW_REG	Interrupt Register	0x000C	RO
TWAI_INT ENA_REG	Interrupt Enable Register	0x0010	R/W

18.7 Register Description

Register 18.1: TWAI_MODE_REG (0x0000)



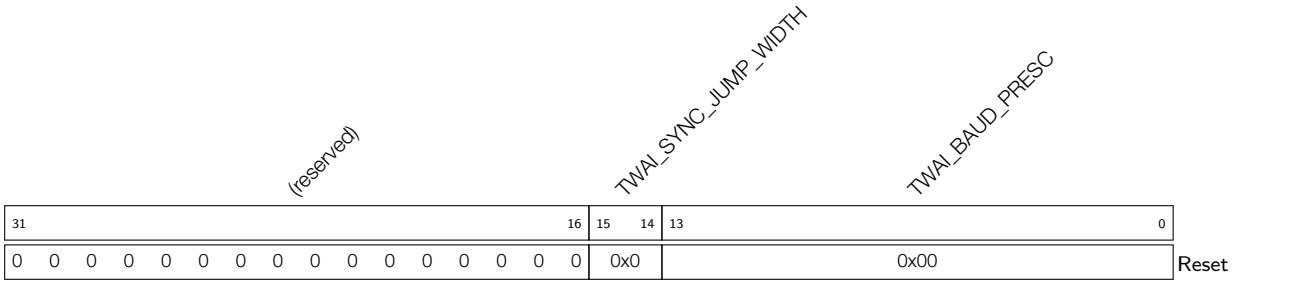
TWAI_RESET_MODE This bit is used to configure the operating mode of the TWAI Controller. 1: Reset mode; 0: Operating mode (R/W)

TWAI_LISTEN_ONLY_MODE 1: Listen only mode. In this mode the nodes will only receive messages from the bus, without generating the acknowledge signal nor updating the RX error counter. (R/W)

TWAI_SELF_TEST_MODE 1: Self test mode. In this mode the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self reception request command. (R/W)

TWAI_RX_FILTER_MODE This bit is used to configure the filter mode. 0: Dual filter mode; 1: Single filter mode (R/W)

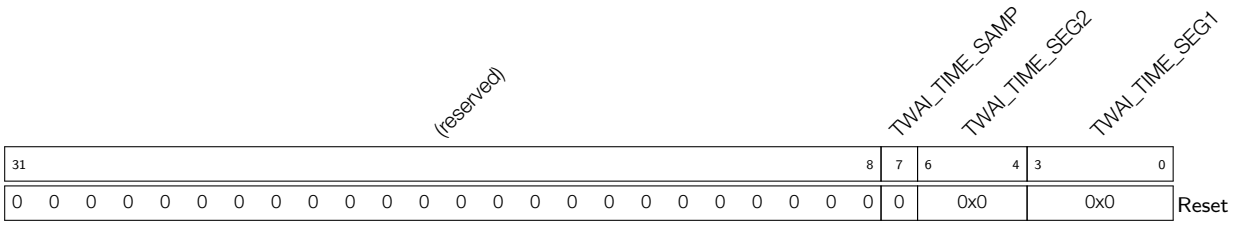
Register 18.2: TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Baud Rate Prescaler, determines the frequency dividing ratio. (RO | R/W)

TWAI_SYNC_JUMP_WIDTH Synchronization Jump Width (SJW), 1 ~ 14 Tq wide. (RO | R/W)

Register 18.3: TWAI_BUS_TIMING_1_REG (0x001C)

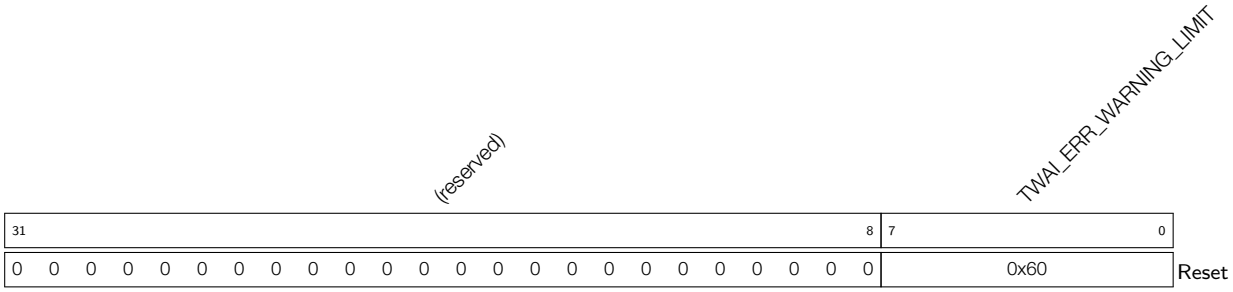


TWAI_TIME_SEG1 The width of PBS1. (RO | R/W)

TWAI_TIME_SEG2 The width of PBS2. (RO | R/W)

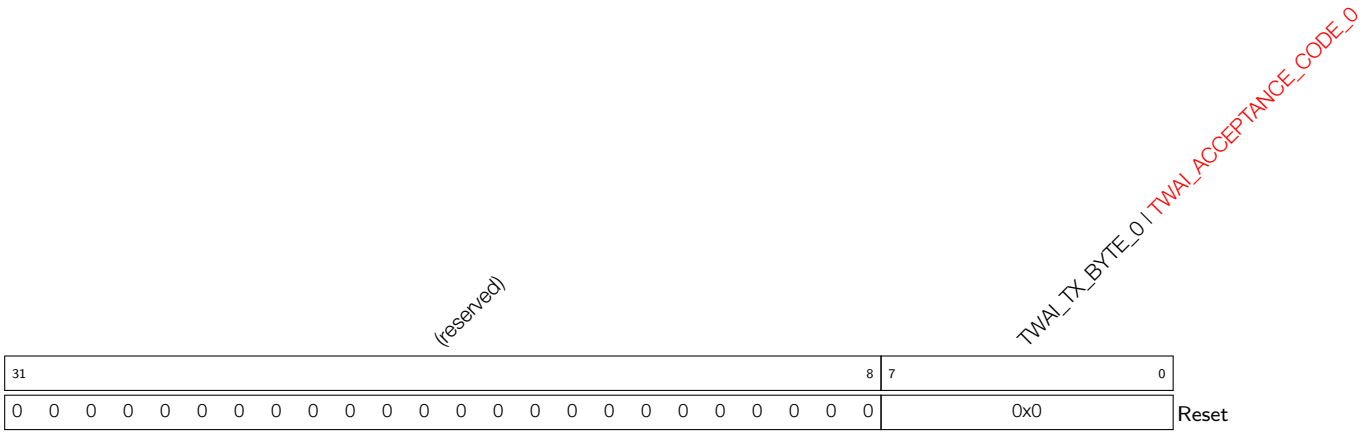
TWAI_TIME_SAMP The number of sample points. 0: the bus is sampled once; 1: the bus is sampled three times (RO | R/W)

Register 18.4: TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Error warning threshold. In the case when any of a error counter value exceeds the threshold, or all the error counter values are below the threshold, an error warning interrupt will be triggered (given the enable signal is valid). (RO | R/W)

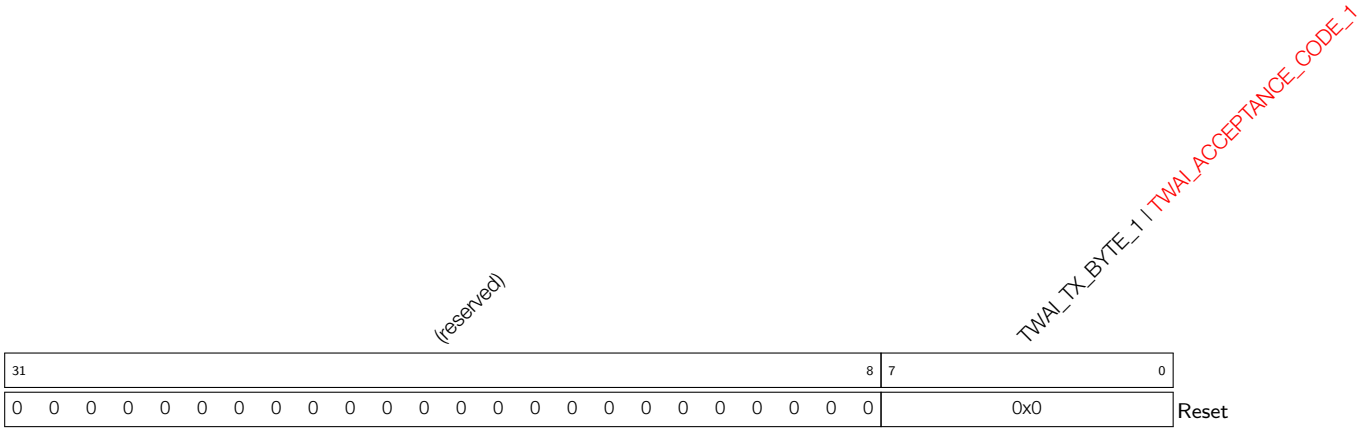
Register 18.5: TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 Stored the 0th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_0 Stored the 0th byte of the filter code under reset mode. (R/W)

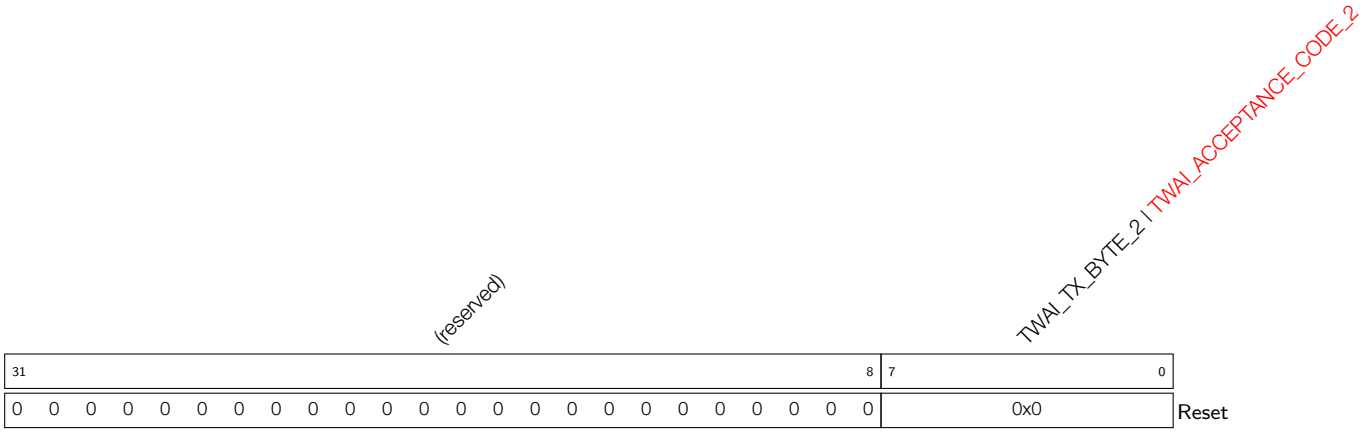
Register 18.6: TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 Stored the 1st byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_1 Stored the 1st byte of the filter code under reset mode. (R/W)

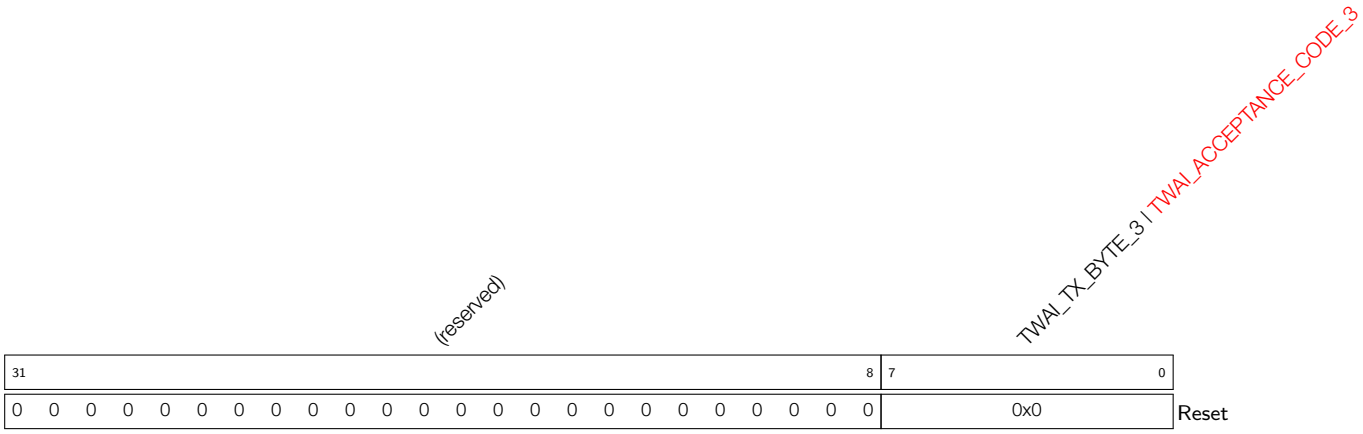
Register 18.7: TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 Stored the 2nd byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_2 Stored the 2nd byte of the filter code under reset mode. (R/W)

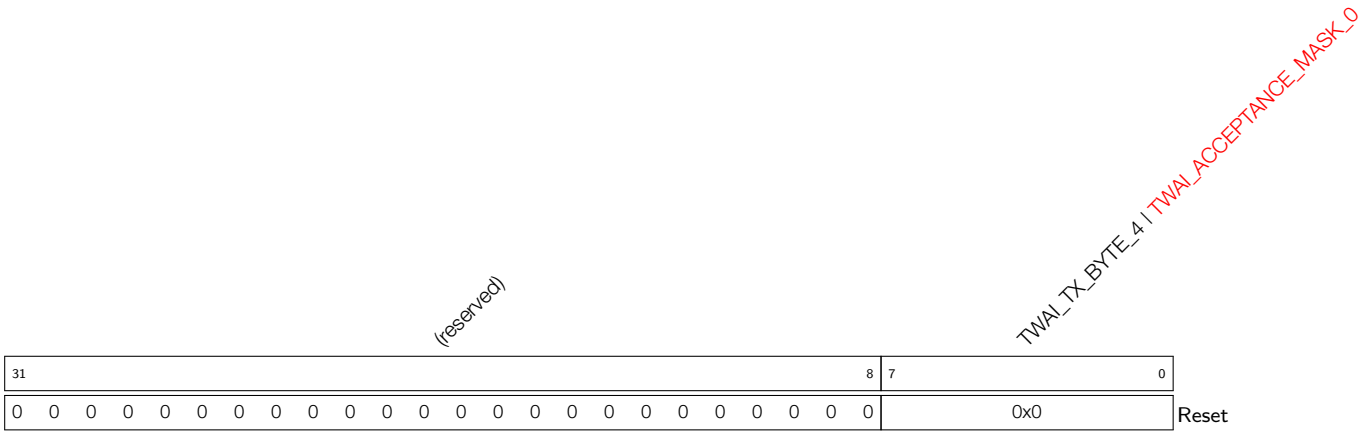
Register 18.8: TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 Stored the 3rd byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_CODE_3 Stored the 3rd byte of the filter code under reset mode. (R/W)

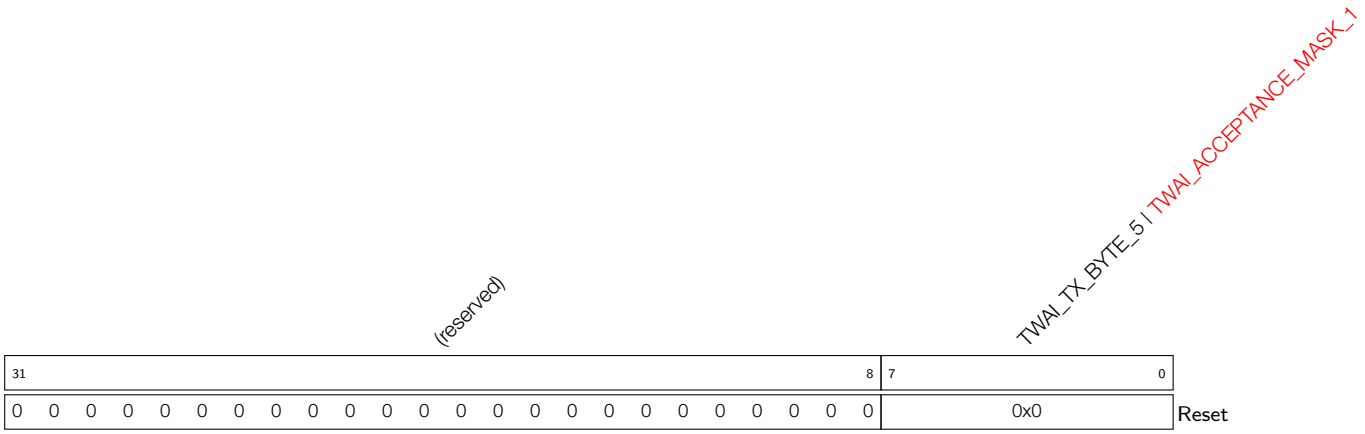
Register 18.9: TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 Stored the 4th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_0 Stored the 0th byte of the filter code under reset mode. (R/W)

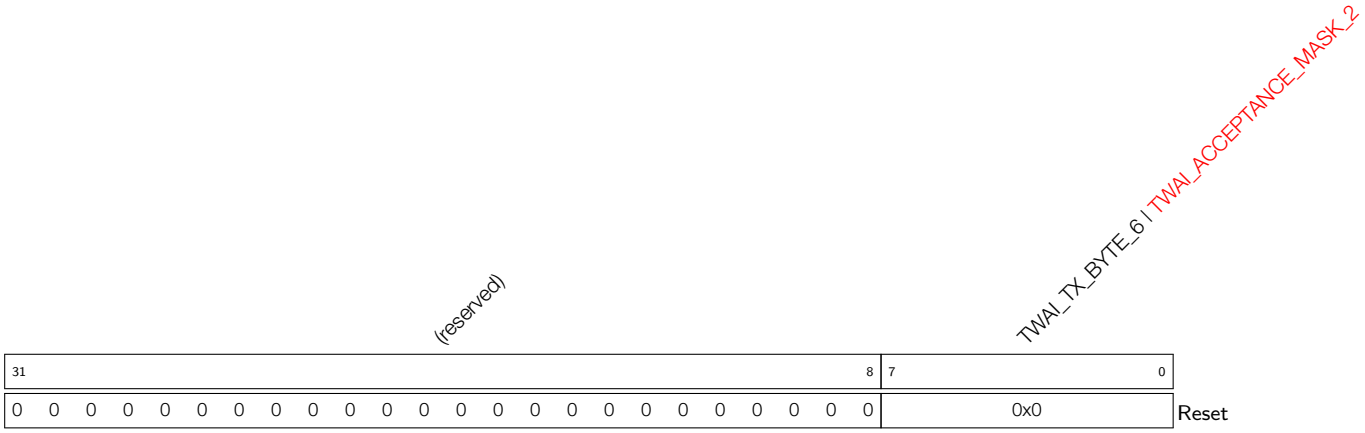
Register 18.10: TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 Stored the 5th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_1 Stored the 1st byte of the filter code under reset mode. (R/W)

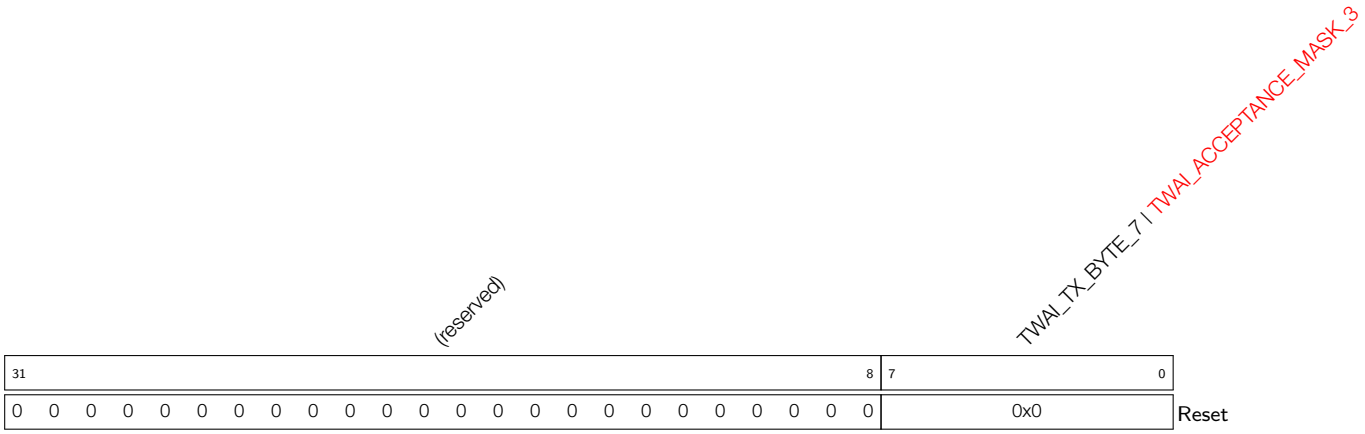
Register 18.11: TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 Stored the 6th byte information of the data to be transmitted under operating mode. (WO)

TWAI_ACCEPTANCE_MASK_2 Stored the 2nd byte of the filter code under reset mode. (R/W)

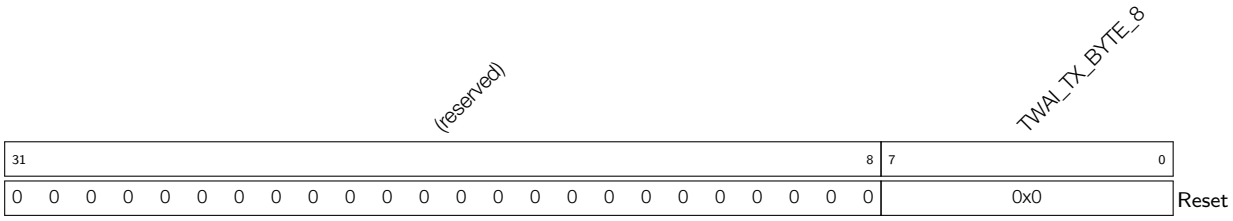
Register 18.12: TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 Stored the 7th byte information of the data to be transmitted under operating mode. (WO)

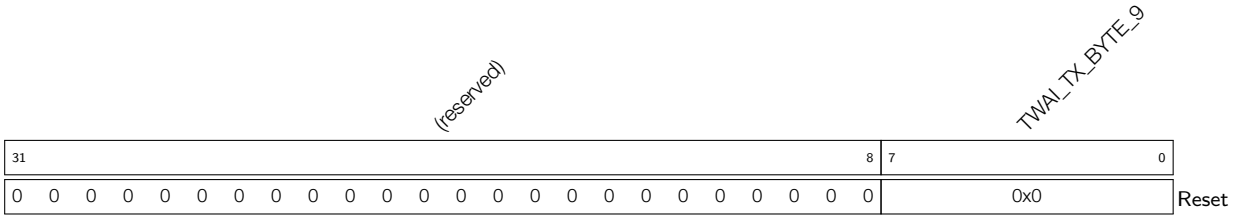
TWAI_ACCEPTANCE_MASK_3 Stored the 3rd byte of the filter code under reset mode. (R/W)

Register 18.13: TWAI_DATA_8_REG (0x0060)



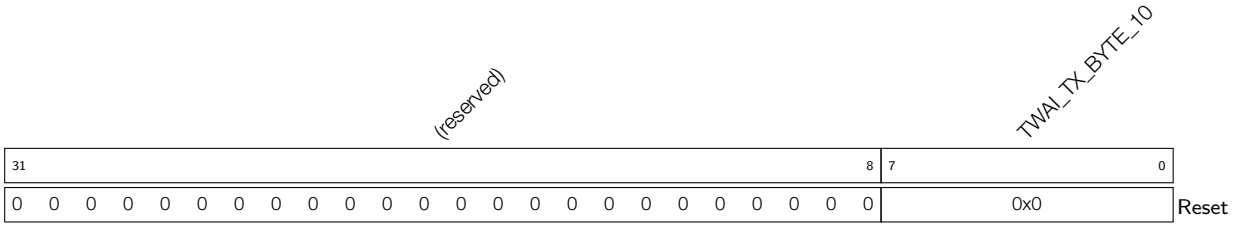
TWAI_TX_BYTE_8 Stored the 8th byte information of the data to be transmitted under operating mode. (WO)

Register 18.14: TWAI_DATA_9_REG (0x0064)



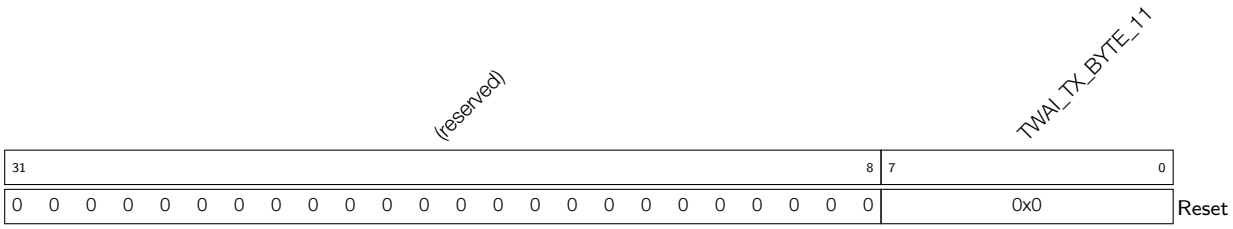
TWAI_TX_BYTE_9 Stored the 9th byte information of the data to be transmitted under operating mode. (WO)

Register 18.15: TWAI_DATA_10_REG (0x0068)



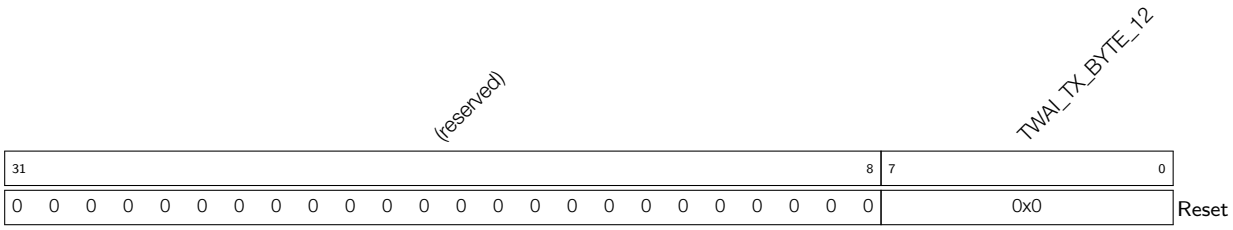
TWAI_TX_BYTE_10 Stored the 10th byte information of the data to be transmitted under operating mode. (WO)

Register 18.16: TWAI_DATA_11_REG (0x006C)



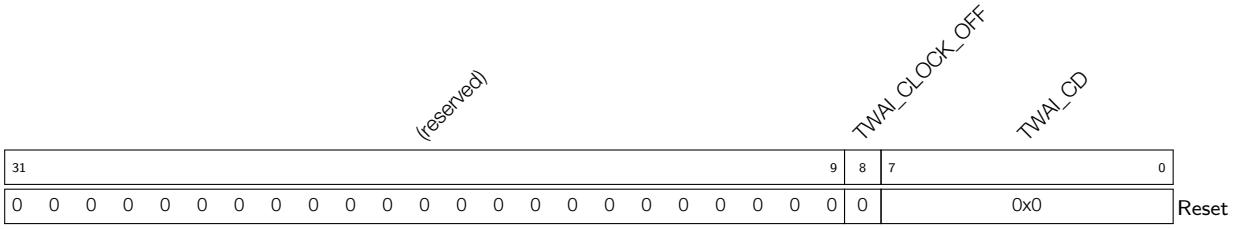
TWAI_TX_BYTE_11 Stored the 11th byte information of the data to be transmitted under operating mode. (WO)

Register 18.17: TWAI_DATA_12_REG (0x0070)



TWAI_TX_BYTE_12 Stored the 12th byte information of the data to be transmitted under operating mode. (WO)

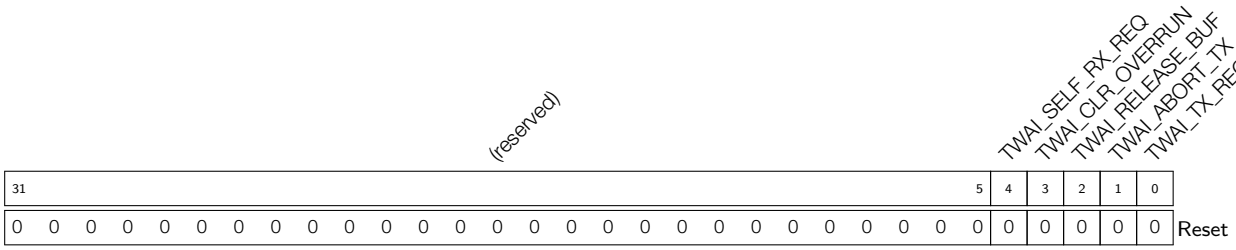
Register 18.18: TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD These bits are used to configure frequency dividing coefficients of the external CLKOUT pin. (R/W)

TWAI_CLOCK_OFF This bit can be configured under reset mode. 1: Disable the external CLKOUT pin; 0: Enable the external CLKOUT pin (RO | R/W)

Register 18.19: TWAI_CMD_REG (0x0004)



TWAI_TX_REQ Set the bit to 1 to allow the driving nodes start transmission. (WO)

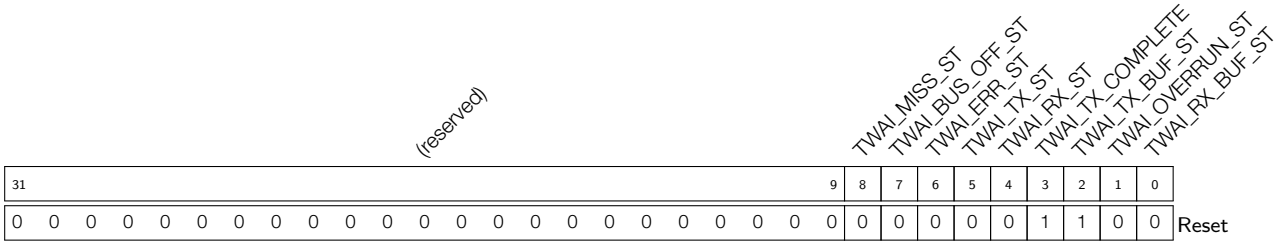
TWAI_ABORT_TX Set the bit to 1 to cancel a pending transmission request. (WO)

TWAI_RELEASE_BUF Set the bit to 1 to release the RX buffer. (WO)

TWAI_CLR_OVERRUN Set the bit to 1 to clear the data overrun status bit. (WO)

TWAI_SELF_RX_REQ Self reception request command. Set the bit to 1 to allow a message be transmitted and received simultaneously. (WO)

Register 18.20: TWAI_STATUS_REG (0x0008)



TWAI_RX_BUF_ST 1: The data in the RX buffer is not empty, with at least one received data packet. (RO)

TWAI_OVERRUN_ST 1: The RX FIFO is full and data overrun has occurred. (RO)

TWAI_TX_BUF_ST 1: The TX buffer is empty, the CPU may write a message into it. (RO)

TWAI_TX_COMPLETE 1: The TWAI controller has successfully received a packet from the bus. (RO)

TWAI_RX_ST 1: The TWAI Controller is receiving a message from the bus. (RO)

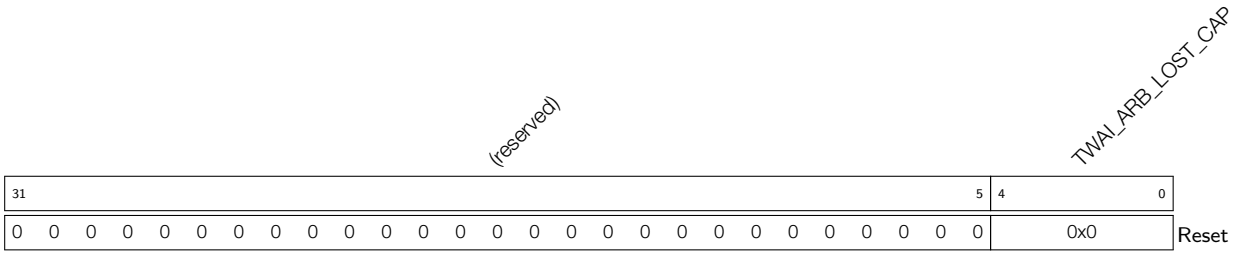
TWAI_TX_ST 1: The TWAI Controller is transmitting a message to the bus. (RO)

TWAI_ERR_ST 1: At least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

TWAI_BUS_OFF_ST 1: In bus-off status, the TWAI Controller is no longer involved in bus activities. (RO)

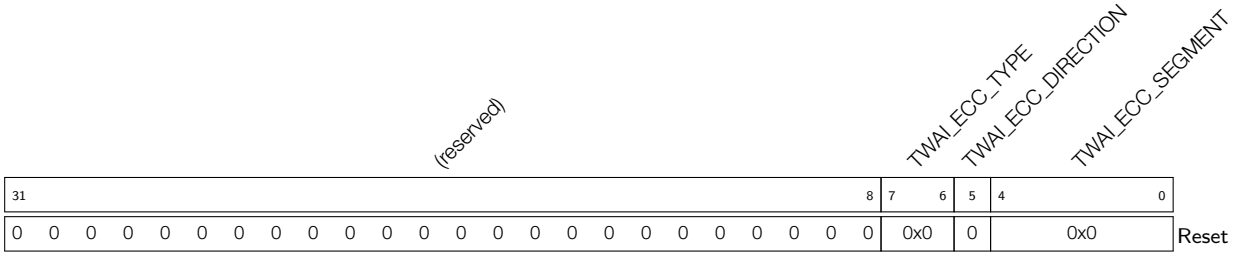
TWAI_MISS_ST This bit reflects whether the data packet in the RX FIFO is complete. 1: The current packet is missing; 0: The current packet is complete (RO)

Register 18.21: TWAI_ARB LOST CAP_REG (0x002C)



TWAI_ARB_LOST_CAP This register contains information about the bit position of lost arbitration. (RO)

Register 18.22: TWAI_ERR_CODE_CAP_REG (0x0030)

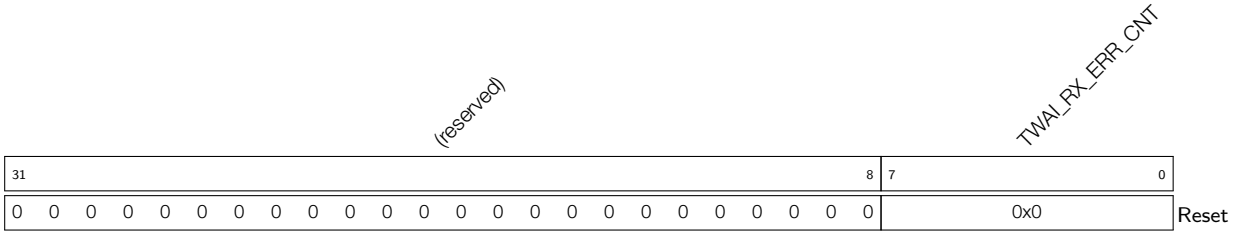


TWAI_ECC_SEGMENT This register contains information about the location of errors, see Table 80 for details. (RO)

TWAI_ECC_DIRECTION This register contains information about transmission direction of the node when error occurs. 1: Error occurs when receiving a message; 0: Error occurs when transmitting a message (RO)

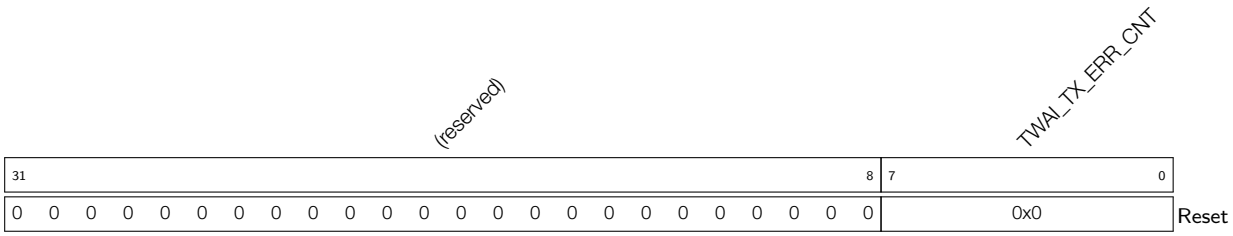
TWAI_ECC_TYPE This register contains information about error types: 00: bit error; 01: form error; 10: stuff error; 11: other type of error (RO)

Register 18.23: TWAI_RX_ERR_CNT_REG (0x0038)



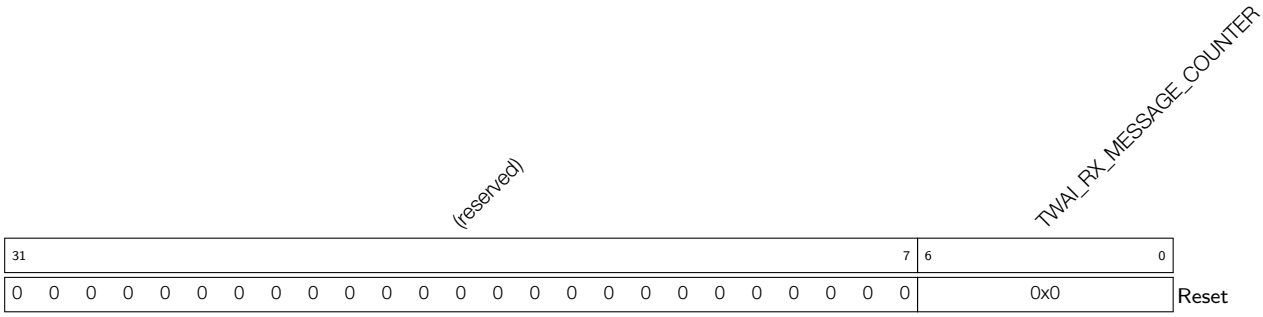
TWAI_RX_ERR_CNT The RX error counter register, reflects value changes under reception status. (RO | R/W)

Register 18.24: TWAI_TX_ERR_CNT_REG (0x003C)



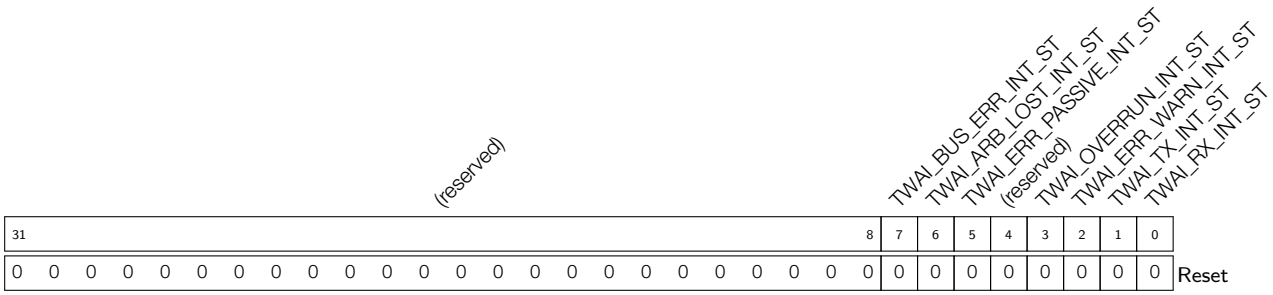
TWAI_TX_ERR_CNT The TX error counter register, reflects value changes under transmission status. (RO | R/W)

Register 18.25: TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER This register reflects the number of messages available within the RX FIFO. (RO)

Register 18.26: TWAI_INT_RAW_REG (0x000C)



TWAI_RX_INT_ST Receive interrupt. If this bit is set to 1, it indicates there are messages to be handled in the RX FIFO. (RO)

TWAI_TX_INT_ST Transmit interrupt. If this bit is set to 1, it indicates the message transmitting mission is finished and a new transmission is able to execute. (RO)

TWAI_ERR_WARN_INT_ST Error warning interrupt. If this bit is set to 1, it indicates the error status signal and the bus-off status signal of Status register have changed (e.g., switched from 0 to 1 or from 1 to 0). (RO)

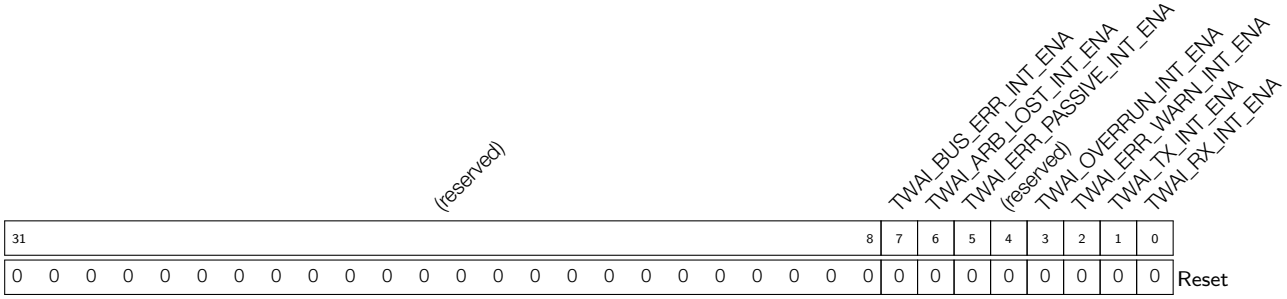
TWAI_OVERRUN_INT_ST Data overrun interrupt. If this bit is set to 1, it indicates a data overrun interrupt is generated in the RX FIFO. (RO)

TWAI_ERR_PASSIVE_INT_ST Error passive interrupt. If this bit is set to 1, it indicates the TWAI Controller is switched between error active status and error passive status due to the change of error counters. (RO)

TWAI_ARB_LOST_INT_ST Arbitration lost interrupt. If this bit is set to 1, it indicates an arbitration lost interrupt is generated. (RO)

TWAI_BUS_ERR_INT_ST Error interrupt. If this bit is set to 1, it indicates an error is detected on the bus. (RO)

Register 18.27: TWAI_INT ENA_REG (0x0010)



TWAI_RX_INT_ENA Set this bit to 1 to enable receive interrupt. (R/W)

TWAI_TX_INT_ENA Set this bit to 1 to enable transmit interrupt. (R/W)

TWAI_ERR_WARN_INT_ENA Set this bit to 1 to enable error warning interrupt. (R/W)

TWAI_OVERRUN_INT_ENA Set this bit to 1 to enable data overrun interrupt. (R/W)

TWAI_ERR_PASSIVE_INT_ENA Set this bit to 1 to enable error passive interrupt. (R/W)

TWAI_ARB_LOST_INT_ENA Set this bit to 1 to enable arbitration lost interrupt. (R/W)

TWAI_BUS_ERR_INT_ENA Set this bit to 1 to enable error interrupt. (R/W)

19. AES Accelerator

19.1 Introduction

ESP32-S2 integrates an Advanced Encryption Standard (AES) Accelerator, which is a hardware device that speeds up AES Algorithm significantly, compared to AES algorithms implemented solely in software. The AES Accelerator integrated in ESP32-S2 has two working modes, which are Typical AES and DMA-AES.

19.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-192/AES-256 encryption and decryption
  - Four variations of key endianness and four variations of text endianness
- DMA-AES working mode
  - Block mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - GCM (Galois/Counter Mode)
  - Interrupt on completion of computation

19.3 Working Modes

The AES Accelerator integrated in ESP32-S2 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode: supports AES-128/AES-192/AES-256 encryption and decryption under NIST FIPS 197. In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode: supports block cipher algorithms ECB/CBC/OFB/CTR/CFB8/CFB128 under NIST SP 800-38A, and GCM mode of operation under NIST SP 800-38D. In this working mode, the plaintext and ciphertext is written and read via crypto DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 84 below.

Table 84: AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode

0	Typical AES
1	DMA-AES

For detailed introduction on these two working modes, please refer to Section 19.4 and Section 19.5 below.

Notice:

ESP32-S2's Digital Signature and External Memory Manual Encryption modules also call the AES accelerator. Therefore, users cannot access the AES accelerator when these modules are working.

19.4 Typical AES Working Mode

In the Typical AES working mode, the AES accelerator is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data, i.e. AES-128/AES-192/AES-256 encryption and decryption. Users can choose the operation type for AES accelerator working in Typical AES working mode by configuring the AES_MODE_REG register according to Table 85 below.

Table 85: Operation Type under Typical AES Working Mode

AES_MODE_REG[2:0]	Operation Type
0	AES-128 encryption
1	AES-192 encryption
2	AES-256 encryption
4	AES-128 decryption
5	AES-192 decryption
6	AES-256 decryption

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 86 below.

Table 86: Working Status under Typical AES Working Mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle or completed operation.
1	WORK	The AES accelerator is in the middle of an operation.

In the Typical AES working mode, the AES accelerator requires 11 ~ 15 clock cycles to encrypt a message block, and 21 or 22 clock cycles to decrypt a message block.

19.4.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.
- For AES-192 encryption/decryption, the 192-bit key is stored in AES_KEY_0_REG ~ AES_KEY_5_REG.
- For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_*m*_REG and AES_TEXT_OUT_*m*_REG, which are two sets of four 32-bit registers.

- For AES-128/AES-192/AES-256 encryption, the AES_TEXT_IN_*m*_REG registers are initialized with plaintext. Then, the AES Accelerator stores the ciphertext into AES_TEXT_OUT_*m*_REG after operation.
- For AES-128/AES-192/AES-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES Accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

19.4.2 Endianness

Text Endianness

In Typical AES working mode, the AES Accelerator uses cryptograohic keys to encrypt and decrypt data in blocks of 128 bits. The Bit 2 and Bit 3 of the AES_ENDIAN_REG register define the endianness of input text, while the Bit 4 and Bit 5 define the endianness of output text. To be more specific, Bit 2 and Bit 4 control how the four bytes are stored in each word, and Bit 3 and Bit 5 control how the four words are stored in each message block.

Users can choose one of the four text endianness types provided by the AES Accelerator by configuring the AES_ENDIAN_REG register. Details can been seen in Table 87.

Table 87: Text Endianness Types for Typical AES

Word Endian Controlling Bit	Byte Endian Controlling Bit	Plaintext/Ciphertext[2]				
0	0	State[1]		c		
				0	1	2	3
		r	0	AES_TEXT_*x*_3_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_0_REG[31:24]
			1	AES_TEXT_*x*_3_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_0_REG[23:16]
			2	AES_TEXT_*x*_3_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_0_REG[15:8]
			3	AES_TEXT_*x*_3_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_0_REG[7:0]
0	1	State		c		
				0	1	2	3
		r	0	AES_TEXT_*x*_3_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_0_REG[7:0]
			1	AES_TEXT_*x*_3_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_0_REG[15:8]
			2	AES_TEXT_*x*_3_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_0_REG[23:16]
			3	AES_TEXT_*x*_3_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_0_REG[31:24]
1	0	State		c		
				0	1	2	3
		r	0	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]
			1	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
			2	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
			3	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
1	1	State		c		
				0	1	2	3
		r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
			1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
			2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
			3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

Note:

1. The definition of "State" is described in Section 3.4 The State in NIST FIPS 197.
2. Where,
   - When *x* = IN, the Word Endian and Byte Endian controlling bits of AES_TEXT_IN_*m*_REG are the Bit 2 and Bit 3 of AES_ENDIAN_REG, respectively;
   - When *x* = OUT, the Word Endian and Byte Endian controlling bits of AES_TEXT_OUT_*m*_REG are the Bit 4 and Bit 5 of AES_ENDIAN_REG, respectively.

Key Endianness

In Typical AES working mode, Bit 0 and bit 1 in AES_ENDIAN_REG define the key endianness.

Users can choose one of the four key endianness types provided by the AES accelerator by configuring the AES_ENDIAN_REG register. Details can been seen in Table 88, Table 89, and Table 90.

Table 88: Key Endianness Types for AES-128 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3][1]
0	0	[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

Note:

1. w[0] ~ w[3] are "the first Nk words of the expanded key" as specified in Section 5.2 Key Expansion in NIST FIPS 197.
2. "Column Bit" specifies the bytes of each word stored in w[0] ~ w[3].

Table 89: Key Endianness Types for AES-192 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5][1]
0	0	[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]

Note:

1. w[0] ~ w[5] are "the first Nk words of the expanded key" as specified in Chapter 5.2 Key Expansion in NIST FIPS 197.
2. "Column Bit" specifies the bytes of each word stored in w[0] ~ w[5].

Table 90: Key Endianness Types for AES-256 Encryption and Decryption

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit[2]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][1]
0	0	[31:24]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

Note:

1. w[0] ~ w[7] are “the first Nk words of the expanded key” as specified in Chapter 5.2 Key Expansion in NIST FIPS 197.
2. “Column Bit” specifies the bytes of each word stored in w[0] ~ w[7].

19.4.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_IN_*m*_REG, and AES_ENDIAN_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

19.5 DMA-AES Working Mode

In the DMA-AES working mode, the AES accelerator supports six block operation including ECB/CBC/OFB/CTR /CFB8/CFB128 as well as GCM operation. Users can choose the operation type for AES accelerator working in the DMA-AES working mode by configuring the AES_BLOCK_MODE_REG register according to Table 91 below.

Table 91: Operation Type under DMA-AES Working Mode

AES_BLOCK_MODE_REG[2:0]	Operation Type
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	GCM (Galois/Counter Mode)

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 92 below.

Table 92: Working Status under DMA-AES Working mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle.
1	WORK	The AES accelerator is in the middle of an operation.
2	DONE	The AES accelerator completed operations.

When working in the DMA-AES working mode, the AES accelerator supports interrupt on the completion of computation. To enable this function, write 1 to the AES_INT_ENA_REG register. By default, the interrupt function is enabled. Also, note that the interrupt should be cleared by software after use.

19.5.1 Key, Plaintext, and Cipertext

Block Operation

During the block operations, the AES Accelerator reads source data (in_stream) from DMA, and write result data (out_stream) to DMA as well, after the computation.

- For encryption, DMA reads plaintext from memory, then passes it to AES as source data. After computation, AES passes ciphertext as result data back to DMA to write into memory.
- For decryption, DMA reads ciphertext from memory, then passes it to AES as source data. After computation, AES passes plaintext as result data back to DMA to write into memory.

During block operations, the lengths of the source data and result data are the same. The total computation time is reduced because the DMA data operation and AES computation can happen concurrently.

The length of source data for AES Accelerator under DMA-AES working mode must be 128 bits or the integral multiples of 128 bits. Otherwise, trailing zeros will be added to the original source data, so the length of source

data equals to the nearest integral multiples of 128 bits. Please see details in Table 93 below.

Table 93: TEXT-PADDING

Function : TEXT-PADDING( )
Input : X, bit string.
Output : Y = **TEXT-PADDING**(X), whose length is the nearest integral multiples of 128 bits.
Steps Let us assume that X is a data-stream that can be split into n parts as following: $X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$ Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is t (0<=t<=127). If $t = 0$, then **TEXT-PADDING**(X) = X; If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n \| \| 0^{128-t}$, then **TEXT-PADDING**($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^* = X \| \| 0^{128-t}$

19.5.2 Endianness

Under the DMA-AES working mode, the transmission of source data and result data for AES Accelerator is solely controlled by DMA. Therefore, the AES Accelerator cannot control the Endianness of the source data and result data, but does have requirement on how these data should be stored in memory and on the length of the data.

For example, let us assume DMA needs to write the following data into memory at address 0x0280.

- Data represented in hexadecimal:
  - 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 94 below.

Table 94: Text Endianness for DMA-AES

Address	Byte	Address	Byte	Address	Byte	Address	Byte
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

DMA can access both internal memory and PSRAM outside ESP32-S2. When you use DMA to access external PSRAM, please use base addresses that meet the requirements for DMA. When you use DMA to access internal memory, base addresses do not have such requirements.

19.5.3 Standard Incrementing Function

AES accelerator provides two Standard Incrementing Functions for the CTR block operation, which are INC_{32} and INC_{128} Standard Incrementing Functions. By setting the AES_INC_SEL_REG register to 0 or 1, users can choose the INC_{32} or INC_{128} functions respectively. For details on the Standard Incrementing Function, please see Chapter B.1 The Standard Incrementing Function in NIST SP 800-38A.

19.5.4 Block Number

Register AES_BLOCK_NUM_REG stores the Block Number of plaintext P or cipertext C. The length of this register equals to length(**TEXT-PADDING**(P))/128 or length(**TEXT-PADDING**(C))/128. The AES Accelerator only uses this register when working in the DMA-AES mode.

19.5.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES Accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 $\cdots$ Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 94, i.e. the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

For more details on IV and ICB, please refer to NIST SP 800-38A.

19.5.6 Block Operation Process

1. Write 0 to the CRYPTO_DMA_AES_SHA_SELECT_REG register.
2. Configure Crypto DMA chained list and start DMA.
3. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, and AES_ENDIAN_REG.
   - Select operation type by configuring the AES_BLOCK_MODE_REG register. For details, see Table 91.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 19.5.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES Accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (only needed when AES Accelerator is working under ECB block operation).
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.
6. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly.

7. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occured during the computation.
8. Release the AES Accelerator by writing 0 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 5 is completed.

19.5.7 GCM Operation Process

1. Write 0 to the CRYPTO_DMA_AES_SHA_SELECT_REG register.
2. Configure Crypto DMA chained list and start DMA.
3. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG AES_ENDIAN_REG.
   - Write 6 to the AES_BLOCK_MODE_REG register.
   - Initialize the AES_BLOCK_NUM_REG register. Details about this register are described in Section 19.5.4.
   - Initialize the AES_AAD_BLOCK_NUM_REG register. Details about this register are described in Section 19.6.4.
   - Initialize the AES_REMAINDER_BIT_NUM_REG register. Details about this register is described in Section 19.6.5.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2. For details on the working status of AES Accelerator, please refer to Table 92. At this step, no interrupt occurs.
6. Obtain the H value from the AES_H_MEM memory.
7. Generate J_0 and write it to the AES_J0_MEM memory.
8. Continue operating by writing 1 to the AES_CONTINUE_REG register.
9. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs. For details on the working status of AES Accelerator, please refer to Table 92.
10. Obtain T_0 by reading AES_T0_MEM, which is already ready at this step.
11. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly.
12. Clear interrupt by writting 1 to the AES_INT_CLR_REG register, if any AES interrupt occured during the computation.
13. Exit DMA by writing 1 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG becomes 0. Note that, you can exit DMA earlier, but only after Step 9 is completed.

19.6 GCM Algorithm

ESP32-S2's AES accelerator fully supports GCM Algorithm. In reality, the AAD, C and P that are longer than 2^{32}-1 bits are seldom used. Therefore, we specify that the length of AAD, C and P should be no longer than 2^{32}-1 here. Figure 19-12 below demonstrates how GCM encryption is implemented in the AES Accelerator of ESP32-S2.

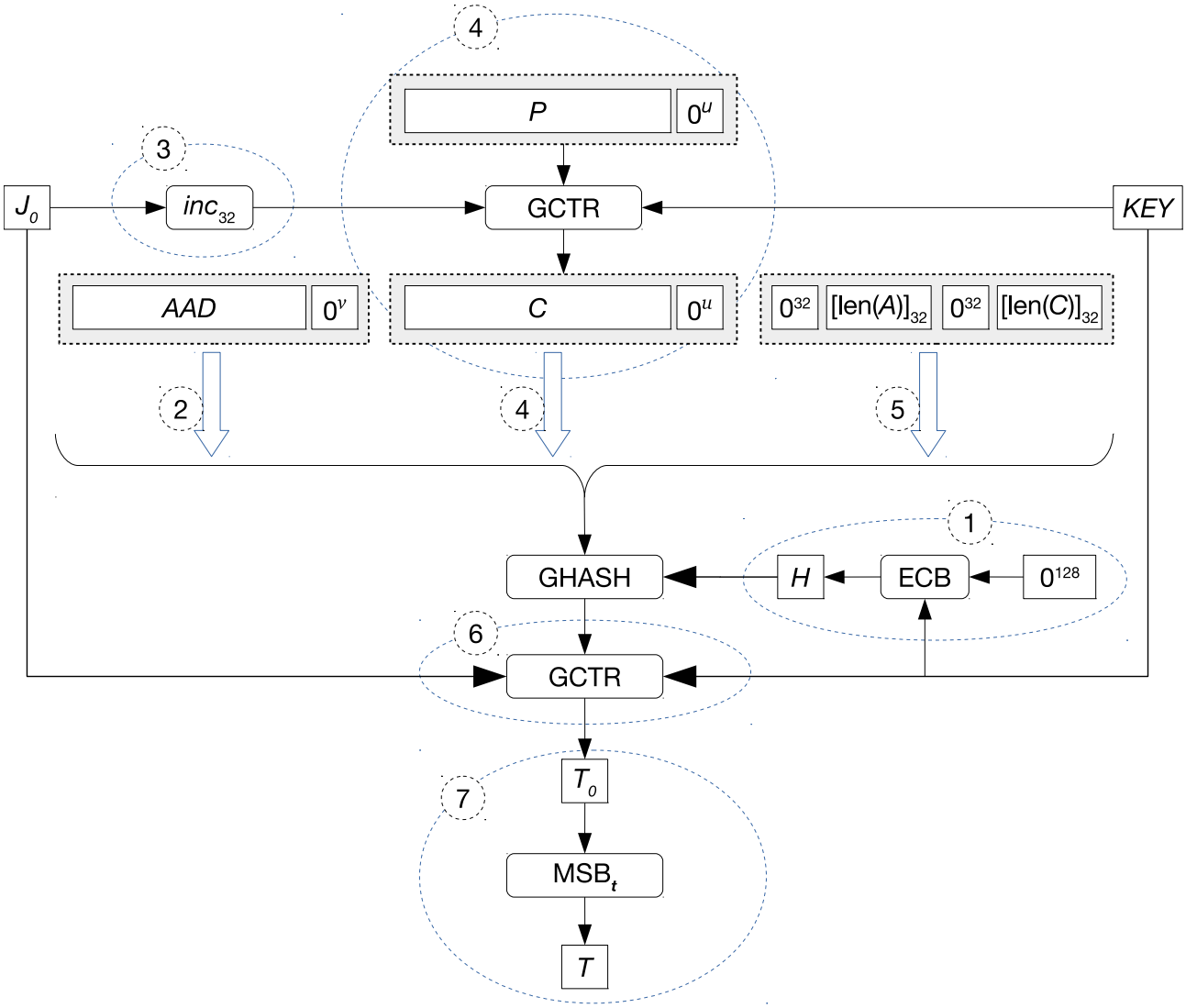


Figure 19-12. GCM Encryption Process

GCM encryption can be implemented as follows:

1. Hardware executes the ECB Algorithm to obtain the Hash subkey H, which is needed in the Hash computation.
2. Hardware executes the GHASH Algorithm to perform Hash computation with the padded AAD.
3. Hardware gets ready for CTR encryption by obtaining the result of applying Standard Incrementing Function INC_{32} to J_0.
4. Hardware executes the GCTR Algorithm to encrypt the padded plaintext P, then executes the GHASH Algorithm to perform Hash computation on the padded cipertext C.
5. Hardware executes the GHASH Algorithm to perform Hash computation on AAD Blocks, obtaining a

128-bit Hash result.

6. Hardware executes the GCTR Algorithm to encrypt J_0, obtaining T_0.
7. Software obtains the result T_0 from hardware, and execute MSB_t Algorithm to obtain the final result Authenticated Tag T.

The only difference between GCM decryption and GCM encryption lies in Step 4 in Figure 19-12. To be more specific, instead of executing GCTR Algorithm to encrypt the padded plaintext, the AES Accelerator executes the same Algorithm to decrypt the padded cipertext in GCM decryption. For details, please see NIST SP 800-38D.

19.6.1 Hash Subkey

During GCM operation, the Hash subkey H is a 128-bit value computed by hardware, which is demonstrated in Step 1 in Figure 19-12. Also you can find more information about Hash subkey at "Step 1. Let $H = \text{CIPH}_K(0^{128})$" in Chapter 7 GCM Specification of NIST SP 800-38D.
Just like all other Endianness, the Hash subkey H is stored in the AES_H_MEM memory with its most significant (i.e., left-most) byte Byte0 stored at the lowest address and least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 94.

19.6.2 J_0

The J_0 is a 128-bit value computed by hardware, which is required during Step 3 and Step 6 of the GCM process in Figure 19-12. For details on the generation of J_0, please see Chapter 7 GCM Specification in NIST SP 800-38D Specification.
The J_0 is stored in the AES_J0_MEM memory as Endianness. Just like all other Endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 94.

19.6.3 Authenticated Tag

Authenticated Tag (Tag for short) is one of the key results of GCM computation, which is demonstrated in Step 7 of Figure 19-12. The value of Tag is determined by the length of Authenticated Tag t ($1 <= t <= 128$):

- When $t = 128$, the value of Tag equals to T_0, a 128-bit string that is stored in the AES_T0_MEM as Endianness. Just like all other Endianness, its most significant (i.e., left-most) byte Byte0 is stored at the lowest address in the memory while least significant (i.e., right-most) byte Byte15 at the highest address. For details, see Table 94.
- When $1 <= t < 128$, the value of Tag equals to the t most significant (i.e., left-most) bits of T_0. In this case, Tag is represented as $\text{MSB}_t(T_0)$, which returns the t most significant bits of T_0. For example, $\text{MSB}_4(111011010) = 1110$ and $\text{MSB}_5(11010011010) = 11010$. For details on the $\text{MSB}_t()$ function, please refer to Chapter 6 Mathematical Components of GCM in the NIST SP 800-38D specification.

19.6.4 AAD Block Number

Register AES_AAD_BLOCK_NUM_REG stores the Block Number of Additional Authenticated Data (AAD). The length of this register equals to length(**TEXT-PADDING**(AAD))/128. AES Accelerator only uses this register when working in the DMA-AES mode.

19.6.5 Remainder Bit Number

Register AES_REMAINDER_BIT_NUM_REG stores the Remainder Bit Number, which indicates the number of effective bits of incomplete blocks in plaintext/cipertext. The value stored in this register equals to length(P)%128 or length(C)%128. AES Accelerator only uses this register when working in the DMA-AES mode.

Register AES_REMAINDER_BIT_NUM_REG does not affect the results of plaintext or cipertext, but does impact the value of T_0, therefore the Tag value too.

The GCM Algorithm can be viewed as the combination of GCTR operation and GHASH operation, among which, the GCTR performs the encryption and decryption, while the GHASH sloves the Tag.

Note that the AES_REMAINDER_BIT_NUM_REG register is only effective for GCM encryption. To be more specific:

- For GCM encryption, the Hardware firstly computes C, then passes it in the form of **TEXT-PADDING**(C) as the input of GHASH operation. In this case, hardware determines how many trailing "0" should be added based on the content of AES_REMAINDER_BIT_NUM_REG.
- For GCM decrpytion, the padding is completed with the **TEXT-PADDING**(C) function. In this case, the AES_REMAINDER_BIT_NUM_REG register is not effective.

19.7 Base Address

Users can access AES with two base addresses, which can be seen in Table 95. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 95: AES Accelerator Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

19.8 Memory Summary

Both the starting address and ending address in the following table are relative to AES base addresses provided in Section 19.7.

Name	Description	Length	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W
AES_H_MEM	Memory H	16 bytes	0x0060	0x006F	RO
AES_J0_MEM	Memory J0	16 bytes	0x0070	0x007F	R/W
AES_T0_MEM	Memory T0	16 bytes	0x0080	0x008F	RO

19.9 Register Summary

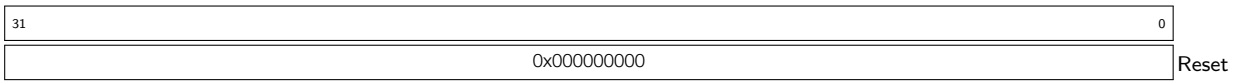
The addresses in the following table are relative to AES base addresses provided in Section 19.7.

Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key register 0	0x0000	R/W
AES_KEY_1_REG	AES key register 1	0x0004	R/W

Name	Description	Address	Access
AES_KEY_2_REG	AES key register 2	0x0008	R/W
AES_KEY_3_REG	AES key register 3	0x000C	R/W
AES_KEY_4_REG	AES key register 4	0x0010	R/W
AES_KEY_5_REG	AES key register 5	0x0014	R/W
AES_KEY_6_REG	AES key register 6	0x0018	R/W
AES_KEY_7_REG	AES key register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result data register 3	0x003C	RO
Configuration Registers			
AES_MODE_REG	AES working mode configuration register	0x0040	R/W
AES_ENDIAN_REG	Endian configuration register	0x0044	R/W
AES_DMA_ENABLE_REG	DMA enable register	0x0090	R/W
AES_BLOCK_MODE_REG	Block operation type register	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
AES_AAD_BLOCK_NUM_REG	AAD block number configuration register	0x00A0	R/W
AES_REMAINDER_BIT_NUM_REG	Remainder bit number of plaintext/ciphertext	0x00A4	R/W
Controlling / Status Registers			
AES_TRIGGER_REG	Operation start controlling register	0x0048	WO
AES_STATE_REG	Operation status register	0x004C	RO
AES_CONTINUE_REG	Operation continue controlling register	0x00A8	WO
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
Interruption Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WO
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

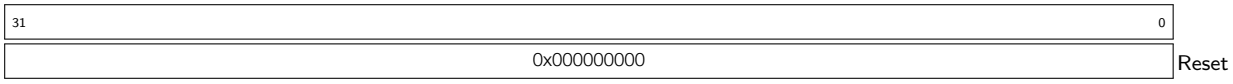
19.10 Registers

Register 19.1: AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



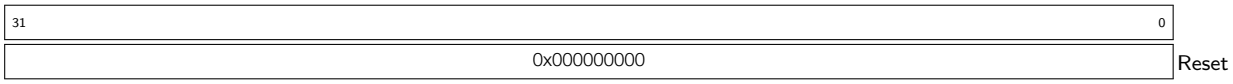
AES_KEY_*n*_REG (*n*: 0-7) Stores AES keys. (R/W)

Register 19.2: AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



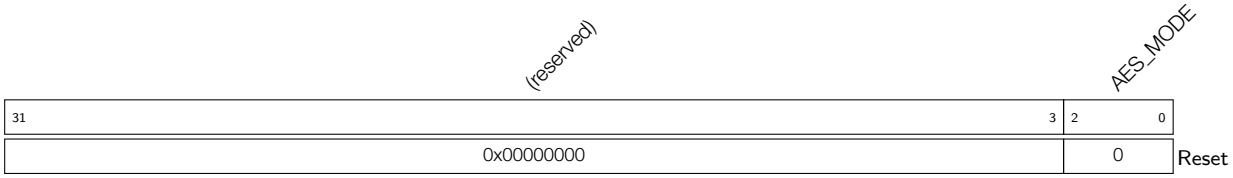
AES_TEXT_IN_*m*_REG (*m*: 0-3) Stores the source data when the AES Accelerator operates in the Typical AES working mode. (R/W)

Register 19.3: AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**



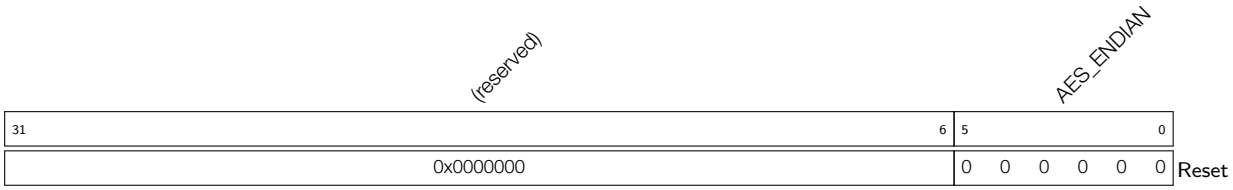
AES_TEXT_OUT_*m*_REG (*m*: 0-3) Stores the result data when the AES Accelerator operates in the Typical AES working mode. (RO)

Register 19.4: AES_MODE_REG (0x0040)



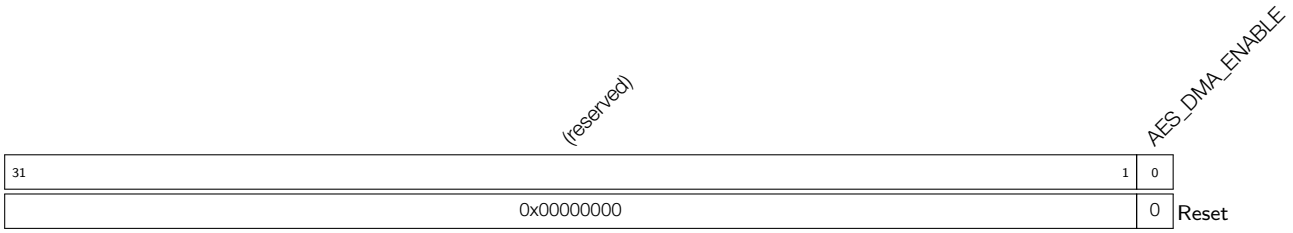
AES_MODE Defines the operation type of the AES Accelerator operating under the Typical AES working mode. For details, see Table 85. (R/W)

Register 19.5: AES_ENDIAN_REG (0x0044)



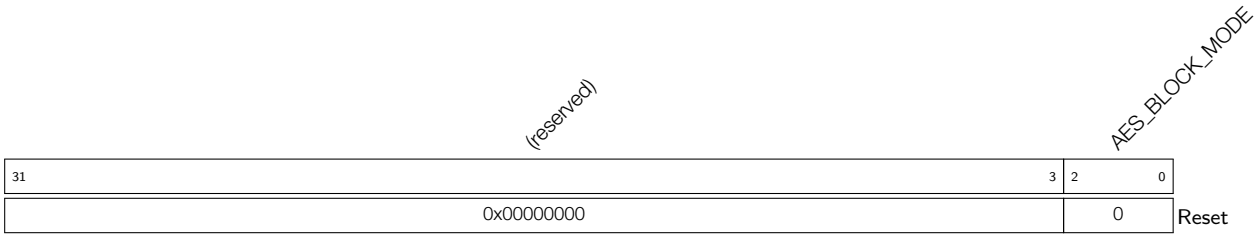
AES_ENDIAN Defines the endianness of input and output texts. For details, please see Table 87. (R/W)

Register 19.6: AES_DMA_ENABLE_REG (0x0090)



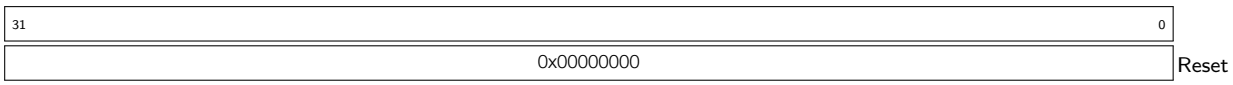
AES_DMA_ENABLE Defines the working mode of the AES Accelerator. For details, see Table 84. (R/W)

Register 19.7: AES_BLOCK_MODE_REG (0x0094)



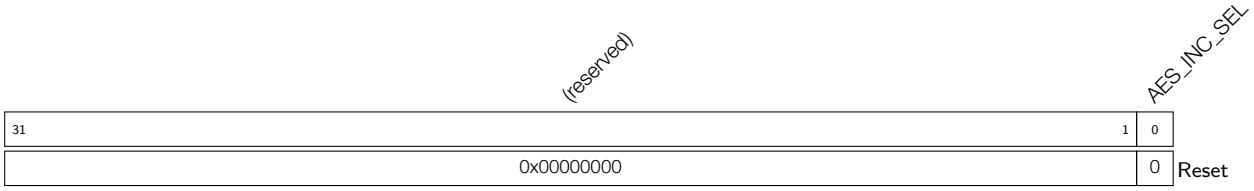
AES_BLOCK_MODE Defines the operation type of the AES Accelerator operating under the DMA-AES working mode. For details, see Table 91. (R/W)

Register 19.8: AES_BLOCK_NUM_REG (0x0098)



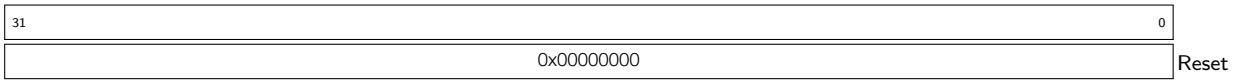
AES_BLOCK_NUM Stores the Block Number of plaintext or cipertext when the AES Accelerator operates under the DMA-AES working mode. For details, see Section 19.5.4. (R/W)

Register 19.9: AES_INC_SEL_REG (0x009C)



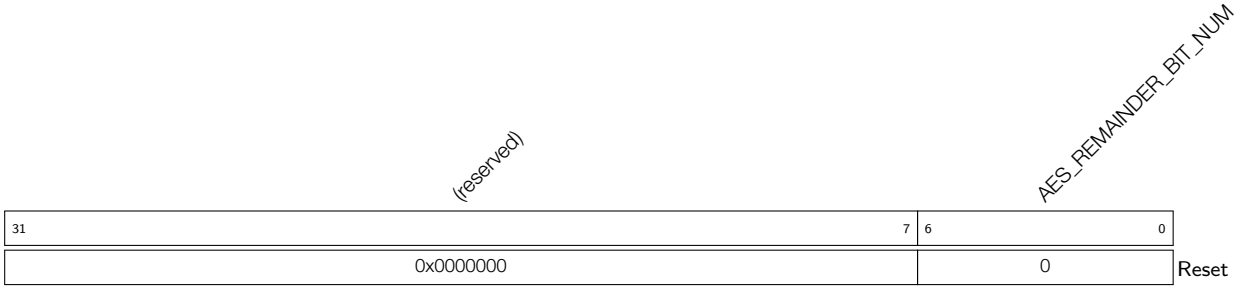
AES_INC_SEL Defines the Standard Incrementing Function for CTR block operation. Set this bit to 0 or 1 to choose INC_{32} or INC_{128}. (R/W)

Register 19.10: AES_AAD_BLOCK_NUM_REG (0x00A0)



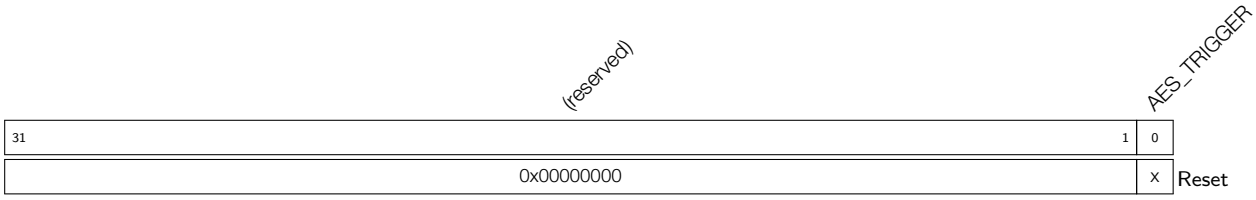
AES_AAD_BLOCK_NUM Stores the ADD Block Number for the GCM operation. (R/W) For details, see Section 19.6.4.

Register 19.11: AES_REMAINDER_BIT_NUM_REG (0x00A4)



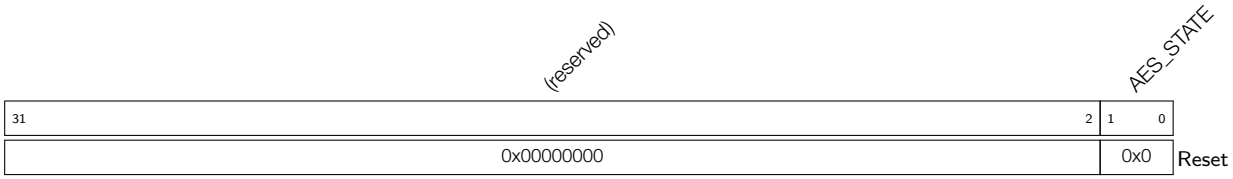
AES_REMAINDER_BIT_NUM Stores the Remainder Bit Number for the GCM operation. For details, see Section 19.6.5. (R/W)

Register 19.12: AES_TRIGGER_REG (0x0048)



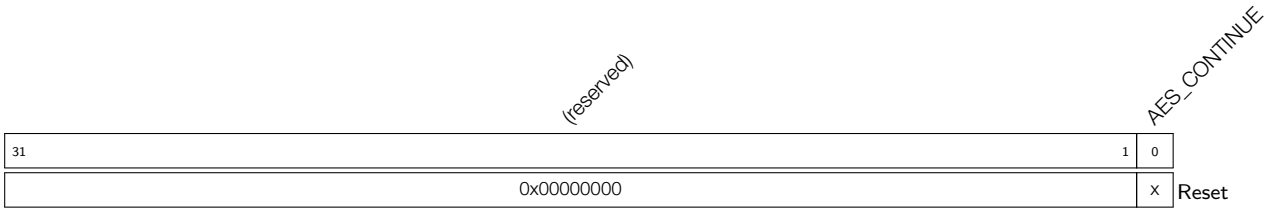
AES_TRIGGER Set this bit to 1 to start AES operation. (WO)

Register 19.13: AES_STATE_REG (0x004C)



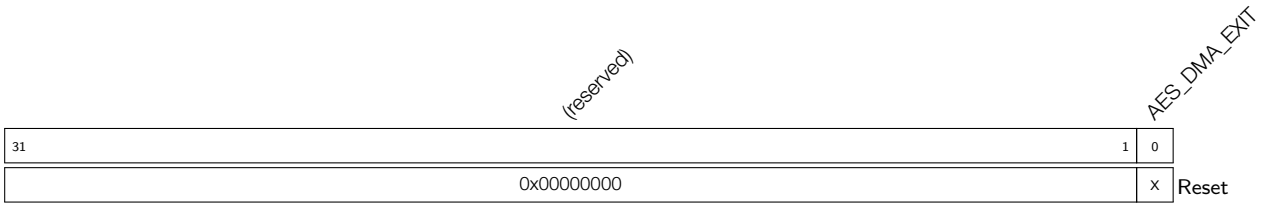
AES_STATE Stores the working status of the AES Accelerator. For details, see Table 86 for Typical AES working mode and Table 92 for DMA AES working mode. (RO)

Register 19.14: AES_CONTINUE_REG (0x00A8)



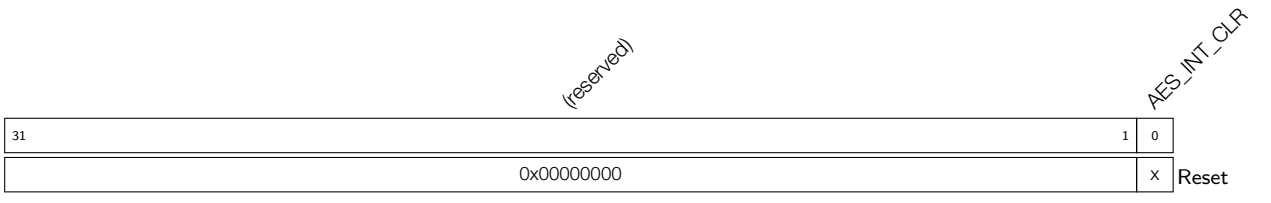
AES_CONTINUE Set this bit to 1 to continue AES operation. (WO)

Register 19.15: AES_DMA_EXIT_REG (0x00B8)



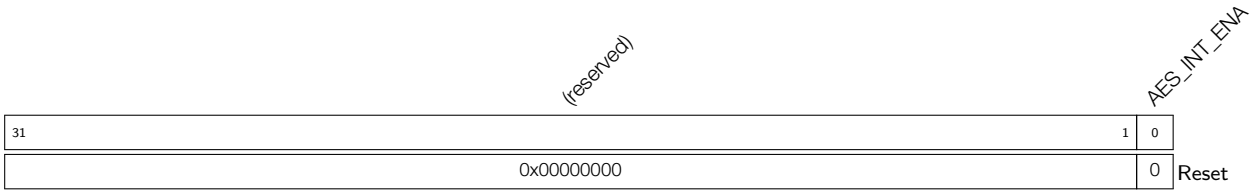
AES_DMA_EXIT Set this bit to 1 to exit AES operation. This register is only effective for DMA-AES operation. (WO)

Register 19.16: AES_INT_CLR_REG (0x00AC)



AES_INT_CLR Set this bit to 1 to clear AES interrupt. (WO)

Register 19.17: AES_INT_ENA_REG (0x00B0)



AES_INT_ENA Set this bit to 1 to enable AES interrupt and 0 to disable interrupt. (R/W)

20. SHA Accelerator

20.1 Introduction

ESP32-S2 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 ESP32-S2 has two working modes, which are Typical SHA and DMA-SHA.

20.2 Features

The following functionality is supported:

- All the hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
  - SHA-512/224
  - SHA-512/256
  - SHA-512/*t*
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleave function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

20.3 Working Modes

The SHA accelerator integrated in ESP32-S2 has two working modes: Typical SHA and DMA-SHA.

- Typical SHA Working Mode: all the data is written and read via CPU directly.
- DMA-SHA Working Mode: all the data is read via crypto 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 98.

Table 98: SHA Accelerator Working Mode

Working Mode	Configuration Method
Typical SHA	Set SHA_START_REG to 1
DMA-SHA	Set SHA_DMA_START_REG to 1

Users can choose hash algorithms by configuring the SHA_MODE_REG register. For details, please see Table 99.

Table 99: SHA Hash Algorithm

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/t	7

Notice:

ESP32-S2's Digital Signature and HMAC modules also call the SHA accelerator. Therefore, users cannot access the SHA accelerator when these modules are working.

20.4 Function Description

SHA accelerator can generate the message digest via two steps: the Preprocessing and Hash operation.

20.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

20.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 or 1024 bits, depending on the algorithm. Thus, all the messages should be padded to a multiple of 512 or 1024 bits before the hash computation.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

- **SHA-1**, **SHA-224** and **SHA-256**
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
  3. Last, append the 64-bit block that is equal to the number m expressed using a binary representation.

- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/**t
  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896 \ mod \ 1024$;
  3. Last, append the 128-bit block that is 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.

20.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit or 1024-bit blocks.

- For **SHA-1**, **SHA-224** and **SHA-256**: the message and its padding are parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.
- For **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/**t: the message and its padding are parsed into N 1024-bit blocks. Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

In Typical SHA working mode, all the message blocks are written into the SHA_M_*n*_REG, following the rules below:

- For **SHA-1, SHA-224 and SHA-256**: $M_0^{(i)}$ 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.
- For **SHA-384, SHA-512, SHA-512/224 and SHA-512/256**: the most significant 32 bits and the least significant 32 bits of $M_0^{(i)}$ are stored in SHA_M_0_REG and SHA_M_1_REG, respectively, ..., the most significant 32 bits and the least significant 32 bits of $M_{15}^{(i)}$ are stored in SHA_M_30_REG and SHA_M_31_REG, respectively.

> **Note:**
> For more information about "message block", please refer to Section "2.1 Glossary of Terms and Acronyms" in FIPS PUB 180-4 Spec.

In DMA-SHA working mode, please complete the following configuration:

1. Create an external linked list;
2. Configure this linked list based on the instruction described in Chapter *DMA Controller*, including but not limited to assigning the starting address of the input message to the buffer address pointer of the linked list;
3. Configure the CRYPTO_DMA_OUTLINK_ADDR to the first out-link linked list;
4. Write 1 to register CRYPTO_DMA_OUTLINK_START, so the DMA starts to move data;
5. Write 1 to register CRYPTO_DMA_AES_SHA_SELECT_REG, so the SHA accelerator gets to use the DMA resource shared by AES and SHA accelerators.

20.4.1.3 Initial Hash Value

Before hash computation begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t\_length = 7'h48$ and t_string is padded in the following format:

$8'h3t_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and $t\_string = 32'h38800000$.

   - If $10 <= t <= 99$, then $t\_length = 7'h50$ and t_string is padded in the following format:

$8'h3t_1$	$8'h3t_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and $t\_string = 32'h35368000$.

   - If $100 <= t < 512$, then $t\_length = 7'h58$ and t_string is padded in the following format:

$8'h3t_2$	$8'h3t_1$	$8'h3t_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and $t\_string = 32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, that is performing SHA-512 operation (with its initial hash value set to the result of 8-bitwise XOR operation of C and 0xa5) from the hexadecimal representation of the string "SHA-512/t".

20.4.2 Hash Computation Process

After the preprocessing, the ESP32-S2 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP32-S2 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.

20.4.2.1 Typical SHA Process

ESP32-S2 SHA accelerator supports "interleave" functionality when working under Typical SHA mode:

- **Type "alone"**: Users do not insert any new computation before the SHA accelerator completes all the message blocks.
- **Type "interleave"**: Users can insert new computations (both Typical SHA task and DMA-SHA task) every time the SHA accelerator completes one message block. To be more specific, users can store the message digest in registers SHA_H_*n*_REG after completing each message block, and assign the accelerator with other higher priority tasks. After the inserted task completes, users can put the message digest stored back to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused computation.

Typical SHA Process (except for SHA-512/*t*)

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 99.
2. Process the current message block.
   (a) Write the current message block in registers SHA_M_*n*_REG;
   (b) Start the SHA accelerator [1]:
      - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the computation;
      - If this is not the first time to execute this step, 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 computation.

   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the computation for the current message block and now is in the "idle" status. Then, go to step 3.
3. Decide if you want to insert other computations.
   - If yes, please get ready for handing over the SHA accelerator for the new task:
     (a) Read and store the hash algorithm selected for the current computation stored in the SHA_MODE_REG register;
     (b) Read and store the message digests stored in registers SHA_H_*n*_REG;
     (c) Last, please go to perform the inserted computation. For the detailed process of the inserted computation, please refer to Typical SHA or DMA-SHA, depending on the working mode.

- Otherwise, please continue to execute Step 4.

4. Decide if you have more message blocks following the previous computation:
   - If yes, please go back to 2.
   - Otherwise, go to Step 5.
5. Decide if you need to return the SHA accelerator to a previous computation (i.e., decide whether the current task is an inserted task or not):
   - If yes, please get ready to return the SHA accelerator for the previous computation:
     (a) Write the previously stored hash algorithm back to register SHA_MODE_REG;
     (b) Write the previously stored message digests back to registers SHA_H_*n*_REG;
     (c) Then, go to Step 2.
   - Otherwise, there is no need to return SHA Control. Therefore, please go to Step 6 directly.
6. Obtain the message digests:
   - Read the message digests from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/*t*)

1. Select a hash algorithm.
   - Select SHA-512/*t* algorithm by configuring the SHA_MODE_REG register to 7.
2. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 20.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
3. Process the current message block.
   (a) Write the current message block in registers SHA_M_*n*_REG;
   (b) Start the SHA accelerator [1]:
   - 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 computation.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the computation for the current message block and now is in the "idle" status. Then, go to step 4.
4. Decide if you want to insert other computations.
   - If yes, please get ready for handing over the SHA accelerator for the new task:
     (a) Read and store the hash algorithm selected for the current computation stored in the SHA_MODE_REG register;
     (b) Read and store the message digests stored in registers SHA_H_*n*_REG;

(c) Last, please go to perform the inserted computation. For the detailed process of the inserted computation, please refer to Typical SHA or DMA-SHA, depending on the working mode.

- Otherwise, please continue to execute Step 5.

5. Decide if you have more message blocks following the previous computation:
   - If yes, please go back to 3.
   - Otherwise, go to Step 6.
6. Decide if you need to return the SHA accelerator to a previous computation (i.e., decide whether the current task is an inserted task or not):
   - If yes, please get ready to return the SHA accelerator for the previous computation:
     (a) Write the previously stored hash algorithm back to register SHA_MODE_REG;
     (b) Write the previously stored message digests back to registers SHA_H_*n*_REG;
     (c) Then, go to Step 3.
   - Otherwise, there is no need to return SHA Control. Therefore, please go to Step 7 directly.
7. Obtain the message digests:
   - Read the message digests from registers SHA_H_*n*_REG.

Note:

1. In Step 2b, 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 computation, to save time.

20.4.2.2 DMA-SHA Process

ESP32-S2 SHA accelerator does not support type "interleave" computation, which means you cannot insert new computation before the whole DMA-SHA process completes. In this mode, users who need task insertion are recommended to divide your message blocks and perform several DMA-SHA computations, instead of trying to compute all the messages in one go.

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 crypto DMA.

Therefore, users are required to configure the DMA controller as instructed in Subsection 20.4.1.2.

DMA-SHA process (except SHA-512/*t*)

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 99.
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 computation.

- If the current DMA-SHA computation follows a previous computation, firstly write the message digest from the previous computation 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 computation, 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 digests:
   - Read the message digests from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/t

1. Select a hash algorithm.
   - Select SHA-512/t algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 20.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
4. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
5. Start the DMA-SHA computation.
   - Write 1 to register SHA_DMA_CONTINUE_REG to start the accelerator.
6. Wait till the completion of the DMA-SHA computation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
7. Obtain the message digests:
   - Read the message digests from registers SHA_H_*n*_REG.

20.4.3 Message Digest

After the hash computation completes, the SHA accelerator writes the message digests from the computation to registers SHA_H_*n*_REG(*n*: 0~15). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 103 below:

Table 103: The Storage and Length of Message Digests from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage [1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/t [2]	t	SHA_H_0_REG ~ SHA_H_x_REG

Note:

1. The message digests are 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 20.4.1.2.
2. The registers used for SHA-512/t algorithm depend on the value of t. x+1 indicates the number of 32-bit registers used to store t bits of message digest, so that $x = roundup(t/32)$-1. For example:
   - When t = 8, then x = 0, indicating that the 8-bit long message digest is stored in the most significant 8 bits of register SHA_H_0_REG;
   - When t = 32, then x = 0, indicating that the 32-bit long message digest is stored in register SHA_H_0_REG;
   - When t = 132, then x = 4, indicating that the 132-bit long message digest is stored in registers SHA_H_0_REG, SHA_H_1_REG, SHA_H_2_REG, SHA_H_3_REG, and SHA_H_4_REG.

20.4.4 Interrupt

SHA accelerator supports interrupt on the completion of computation 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.

20.5 Base Address

Users can access SHA with two base addresses, which can be seen in Table 104. For more information about accessing peripherals from different buses, please see Chapter 1 *System and Memory*.

Table 104: Base Address

Bus	Base Address
PeriBUS1	0x3F43B000
PeriBUS2	0x6003B000

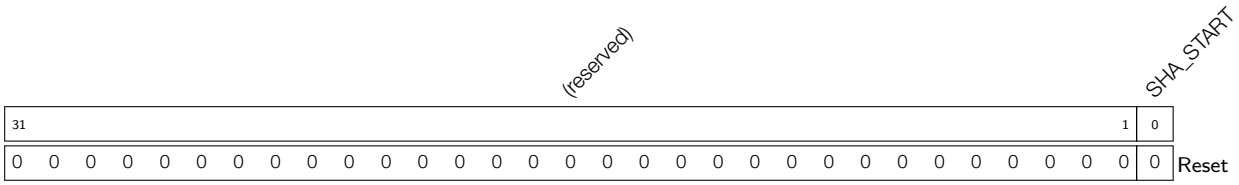
20.6 Register Summary

Name	Description	Address	Access
Control/Status registers			
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_BUSY_REG	Indicates if SHA Accelerator is busy or not	0x0018	RO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	Starts the SHA accelerator for Typical SHA operation	0x0010	WO
SHA_DMA_CONTINUE_REG	Continues SHA operation (only effective in DMA-SHA mode)	0x0020	WO
SHA_INT_CLEAR_REG	DMA-SHA interrupt clear register	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA interrupt enable register	0x0028	R/W
Version Register			
SHA_DATE_REG	Version control register	0x002C	R/W
Configuration Registers			
SHA_MODE_REG	Defines the algorithm of SHA accelerator	0x0000	R/W
SHA_T_STRING_REG	String content register for calculating initial Hash Value (only effective for SHA-512/t)	0x0004	R/W
SHA_T_LENGTH_REG	String length register for calculating initial Hash Value (only effective for SHA-512/t)	0x0008	R/W
Memories			
SHA_DMA_BLOCK_NUM_REG	Block number register (only effective for DMA-SHA)	0x000C	R/W
SHA_H_0_REG	Hash value	0x0040	R/W
SHA_H_1_REG	Hash value	0x0044	R/W
SHA_H_2_REG	Hash value	0x0048	R/W
SHA_H_3_REG	Hash value	0x004C	R/W
SHA_H_4_REG	Hash value	0x0050	R/W
SHA_H_5_REG	Hash value	0x0054	R/W
SHA_H_6_REG	Hash value	0x0058	R/W
SHA_H_7_REG	Hash value	0x005C	R/W
SHA_H_8_REG	Hash value	0x0060	R/W
SHA_H_9_REG	Hash value	0x0064	R/W

Name	Description	Address	Access
SHA_H_10_REG	Hash value	0x0068	R/W
SHA_H_11_REG	Hash value	0x006C	R/W
SHA_H_12_REG	Hash value	0x0070	R/W
SHA_H_13_REG	Hash value	0x0074	R/W
SHA_H_14_REG	Hash value	0x0078	R/W
SHA_H_15_REG	Hash value	0x007C	R/W
SHA_M_0_REG	Message	0x0080	R/W
SHA_M_1_REG	Message	0x0084	R/W
SHA_M_2_REG	Message	0x0088	R/W
SHA_M_3_REG	Message	0x008C	R/W
SHA_M_4_REG	Message	0x0090	R/W
SHA_M_5_REG	Message	0x0094	R/W
SHA_M_6_REG	Message	0x0098	R/W
SHA_M_7_REG	Message	0x009C	R/W
SHA_M_8_REG	Message	0x00A0	R/W
SHA_M_9_REG	Message	0x00A4	R/W
SHA_M_10_REG	Message	0x00A8	R/W
SHA_M_11_REG	Message	0x00AC	R/W
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W
SHA_M_16_REG	Message	0x00C0	R/W
SHA_M_17_REG	Message	0x00C4	R/W
SHA_M_18_REG	Message	0x00C8	R/W
SHA_M_19_REG	Message	0x00CC	R/W
SHA_M_20_REG	Message	0x00D0	R/W
SHA_M_21_REG	Message	0x00D4	R/W
SHA_M_22_REG	Message	0x00D8	R/W
SHA_M_23_REG	Message	0x00DC	R/W
SHA_M_24_REG	Message	0x00E0	R/W
SHA_M_25_REG	Message	0x00E4	R/W
SHA_M_26_REG	Message	0x00E8	R/W
SHA_M_27_REG	Message	0x00EC	R/W
SHA_M_28_REG	Message	0x00F0	R/W
SHA_M_29_REG	Message	0x00F4	R/W
SHA_M_30_REG	Message	0x00F8	R/W
SHA_M_31_REG	Message	0x00FC	R/W

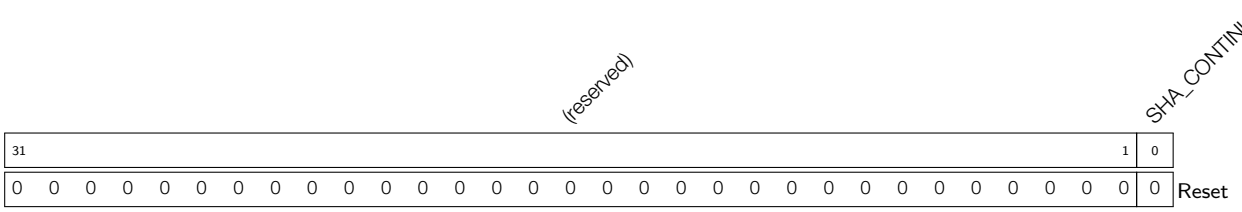
20.7 Registers

Register 20.1: SHA_START_REG (0x0010)



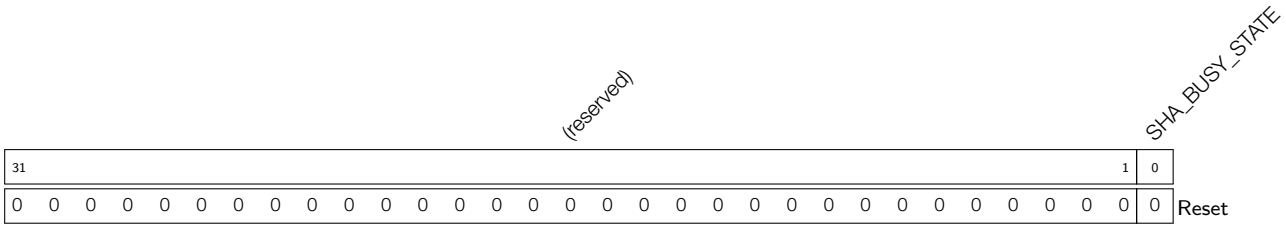
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 20.2: SHA_CONTINUE_REG (0x0014)



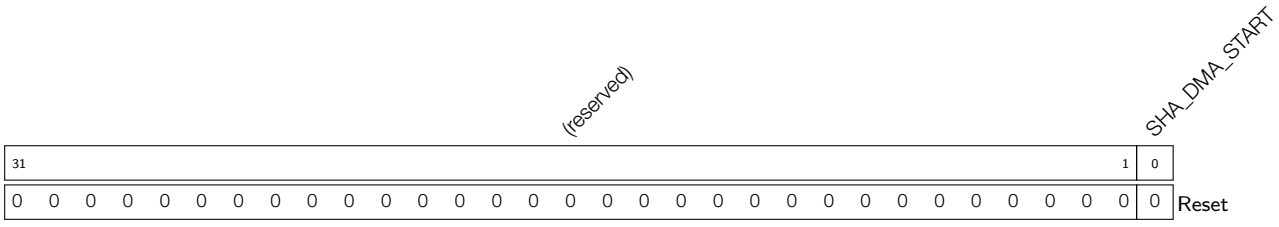
SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 20.3: SHA_BUSY_REG (0x0018)



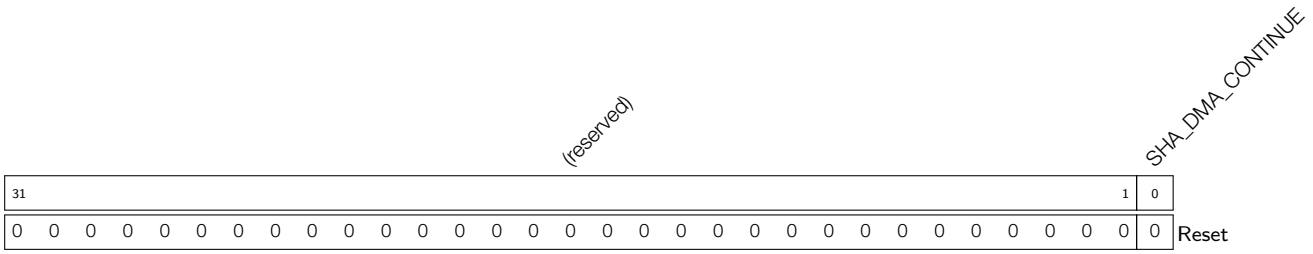
SHA_BUSY_STATE Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

Register 20.4: SHA_DMA_START_REG (0x001C)



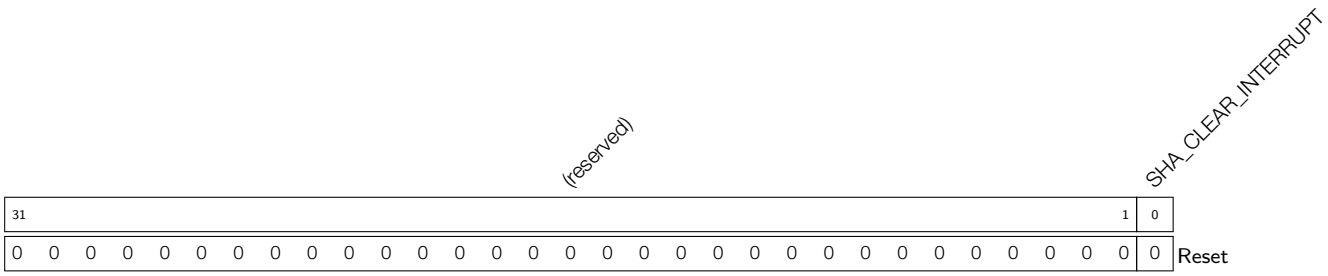
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 20.5: SHA_DMA_CONTINUE_REG (0x0020)



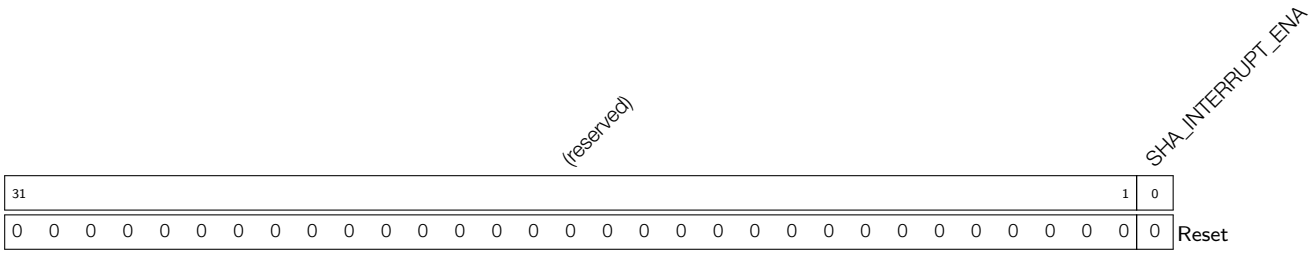
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 20.6: SHA_INT_CLEAR_REG (0x0024)



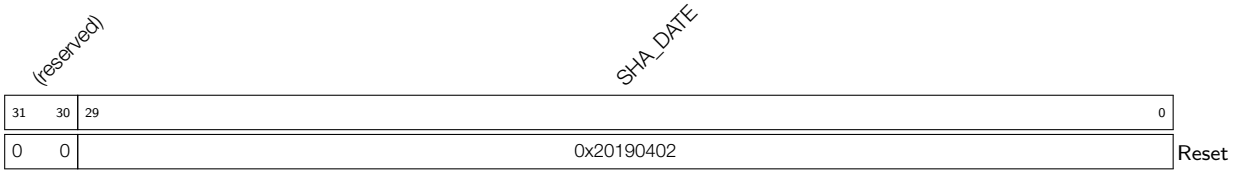
SHA_CLEAR_INTERRUPT Clears DMA-SHA interrupt. (WO)

Register 20.7: SHA_INT_ENA_REG (0x0028)



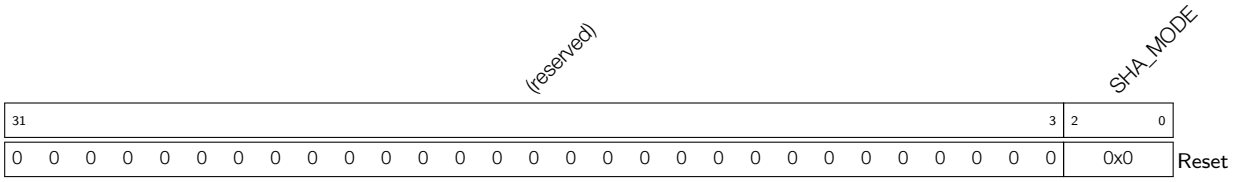
SHA_INTERRUPT_ENA Enables DMA-SHA interrupt. (R/W)

Register 20.8: SHA_DATE_REG (0x002C)



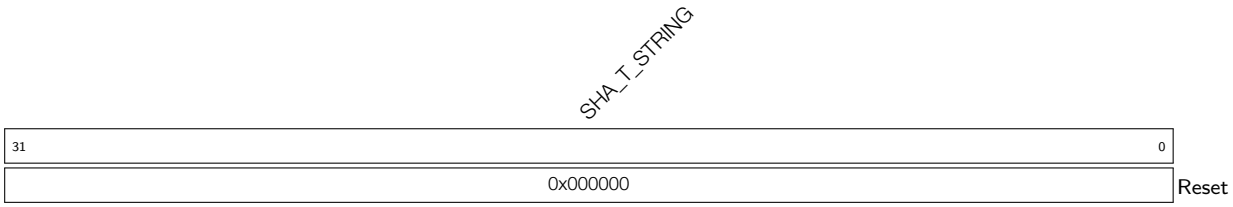
SHA_DATE Version control register. (R/W)

Register 20.9: SHA_MODE_REG (0x0000)



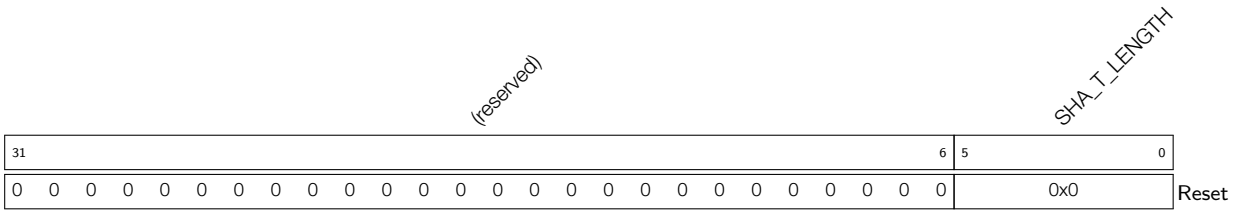
SHA_MODE Defines the SHA algorithm. For details, please see Table 99. (R/W)

Register 20.10: SHA_T_STRING_REG (0x0004)



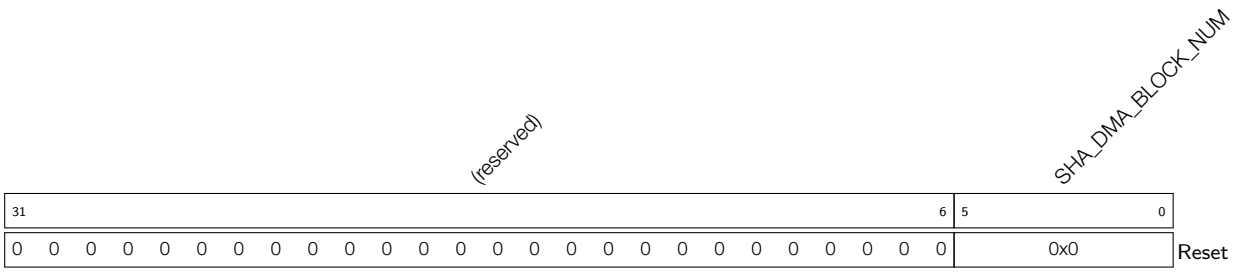
SHA_T_STRING Defines t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 20.11: SHA_T_LENGTH_REG (0x0008)



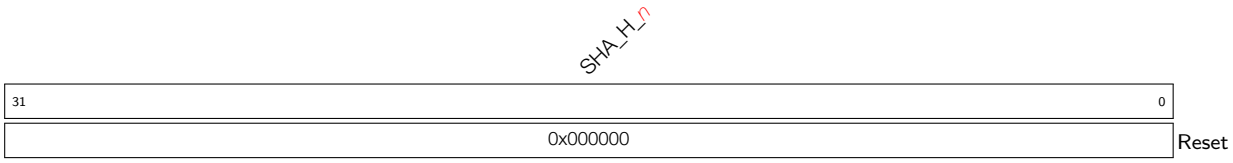
SHA_T_LENGTH Defines t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 20.12: SHA_DMA_BLOCK_NUM_REG (0x000C)



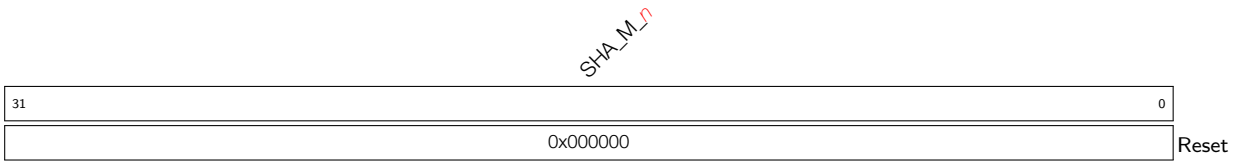
SHA_DMA_BLOCK_NUM Defines the DMA-SHA block number. (R/W)

Register 20.13: SHA_H_*n*_REG (*n*: 0-15) (0x0040+4n*)**



SHA_H_*n* Stores the *n*th 32-bit piece of the Hash value. (R/W)

Register 20.14: SHA_M_*n*_REG (*n*: 0-31) (0x0080+4n*)**



SHA_M_*n* Stores the *n*th 32-bit piece of the message. (R/W)

21. RSA Accelerator

21.1 Introduction

The RSA Accelerator provides hardware support for high precision computation used in various RSA asymmetric cipher algorithms by significantly reducing their software complexity. Compared with RSA algorithms implemented solely in software, this hardware accelerator can speed up RSA algorithms significantly. Besides, the RSA Accelerator also supports operands of different lengths, which provides more flexibility during the computation.

21.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two optional acceleration options
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

21.3 Functional Description

The RSA Accelerator is activated by setting the DPORT_PERIP_CLK_EN1_REG bit in the DPORT_CRYPTO_RSA_CLK_EN peripheral clock and clearing the DPORT_RSA_PD bit in the DPORT_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

The RSA Accelerator is only available after the RSA-related memories are initialized. The content of the RSA_CLEAN_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, it is advised to wait until RSA_CLEAN_REG becomes 1 before using the RSA Accelerator.

The RSA_INTERRUPT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this register to enable or disable interrupt. By default, the interrupt function of the RSA Accelerator is enabled.

Notice:

ESP32-S2's Digital Signature module also calls the RSA accelerator. Therefore, users cannot access the RSA accelerator when Digital Signature is working.

21.3.1 Large Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The computation is based on Montgomery multiplication. Therefore, aside from the X, Y, and M arguments, two additional ones are needed — $\overline{r}$ and M', which need to be calculated in advance by software.

RSA Accelerator supports operands of length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the n values in $Z_{n-1}\cdots Z_0$, $X_{n-1}\cdots X_0$, $Y_{n-1}\cdots Y_0$, $M_{n-1}\cdots M_0$, $\overline{r}_{n-1}\cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional arguments can be calculated as $\overline{r} = R^2 \bmod M$, where $R = b^n$.

The following equation in the form compatible with the extended binary GCD algorithm can be written as

$$M^{-1} \times M + 1 = R \times R^{-1}$$
$$M' = M^{-1} \bmod b$$

Large-number modular exponentiation can be implemented as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
   (c) Configure registers related to the acceleration options, which are described later in Section 21.3.4.
3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block

can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.

4. Write 1 to the RSA_MODEXP_START_REG register to start computation.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

21.3.2 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. The same values $\overline{r}$ and M' are derived by software.

The RSA Accelerator supports large-number modular multiplication with operands of 128 different lengths.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n\}$ to registers RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MODMULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

21.3.3 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA Accelerator only supports Large Number Multiplication with operand length $N = 32 \times x$, where $x \in \{0, 1, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e. $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n\}$ to registers RSA_X_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

   Write X_i for $i \in \{0, 1, \ldots, n\}$ to the address of the i words of the RSA_X_MEM register. Note that Y_i for $i \in \{0, 1, \ldots, n\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n + i$ words, i.e. the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n + i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

21.3.4 Acceleration Options

ESP32-S2 RSA provides two acceleration options for the large-number modular exponentiation, which are SEARCH Option and the CONSTANT_TIME Option. These two options are both disabled by default, but can be enabled at the same time.

When neither of these two options are enabled, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. However, when either one of these two options is enabled, the time required is also correlated with the 0/1 distribution of Y.

To better illustrate the acceleration options, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,

- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e. the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When the acceleration options are enabled, the RSA accelerator:

- SEARCH Option
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. The maximum value of α is N-1, which leads to the same result when this accleration option is disabled. The best acceleration performance can be achieved by setting α to t, in which case, all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
- CONSTANT_TIME Option
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better the acceleration performance is.

We provide an example to demonstrate the performance of the RSA Accelerator when different acceleration options are enabled. Here we perform $Z = X^Y \bmod M$ with N = 3072 and Y = 65537. Table 106 below demonstrates the time costs when different acceleration options are enabled. It's obvious that the time cost can be dramatically reduced when acceleration option(s) is enabled. Here, we should also mention that, α is set to 16 when the SEARCH option is enabled.

Table 106: Acceleration Performace

SEARCH Option	CONSTANT_TIME Option	Time Cost	Acceleration Performance by Percentage
Disabled	Disabled	376.405 ms	0%
Enabled	Disabled	2.260 ms	99.41%
Disabled	Enabled	1.203 ms	99.68%
Enabled	Enabled	1.165 ms	99.69%

21.4 Base Address

Users can access RSA with two base addresses, which can be seen in Table 107. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 107: RSA Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43C000
PeriBUS2	0x6003C000

21.5 Memory Summary

Both the starting address and ending address in the following table are relative to RSA base addresses provided in Section 21.4.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	512	0x0000	0x01FF	WO
RSA_Z_MEM	Memory Z	512	0x0200	0x03FF	R/W
RSA_Y_MEM	Memory Y	512	0x0400	0x05FF	WO
RSA_X_MEM	Memory X	512	0x0600	0x07FF	WO

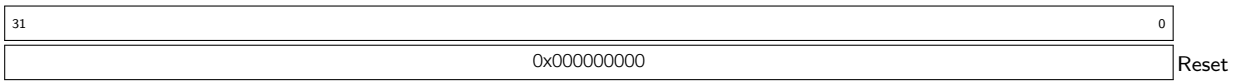
21.6 Register Summary

The addresses in the following table are relative to RSA base addresses provided in Section 21.4.

Name	Description	Address	Access
Configuration Registers			
RSA_M_PRIME_REG	Register to store M'	0x0800	R/W
RSA_MODE_REG	RSA length mode	0x0804	R/W
RSA_CONSTANT_TIME_REG	The constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	The search option	0x0824	R/W
RSA_SEARCH_POS_REG	The search position	0x0828	R/W
Status/Control Registers			
RSA_CLEAN_REG	RSA clean register	0x0808	RO
RSA_MODEXP_START_REG	Modular exponentiation starting bit	0x080C	WO
RSA_MODMULT_START_REG	Modular multiplication starting bit	0x0810	WO
RSA_MULT_START_REG	Normal multiplication starting bit	0x0814	WO
RSA_IDLE_REG	RSA idle register	0x0818	RO
Interupt Registers			
RSA_CLEAR_INTERRUPT_REG	RSA clear interrupt register	0x081C	WO
RSA_INTERRUPT_ENA_REG	RSA interrupt enable register	0x082C	R/W
Version Register			
RSA_DATE_REG	Version control register	0x0830	R/W

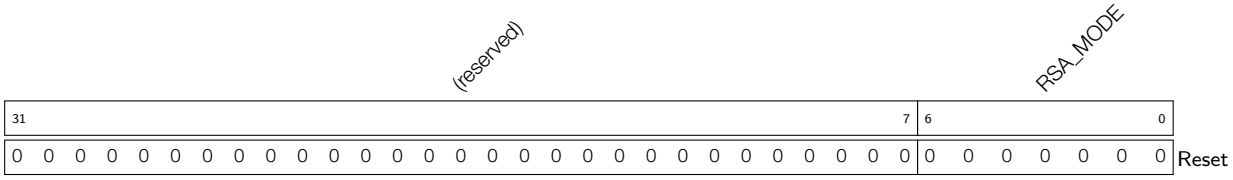
21.7 Registers

Register 21.1: RSA_M_PRIME_REG (0x800)



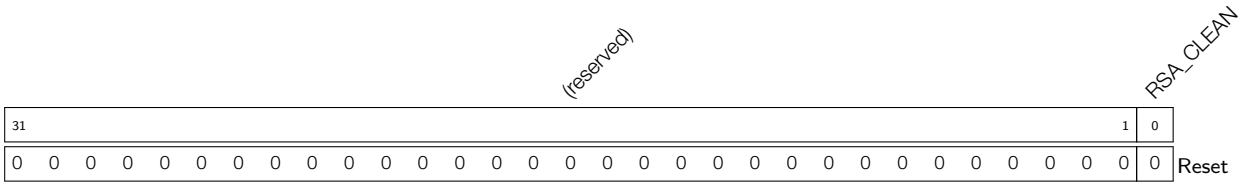
RSA_M_PRIME_REG Stores M'.(R/W)

Register 21.2: RSA_MODE_REG (0x804)



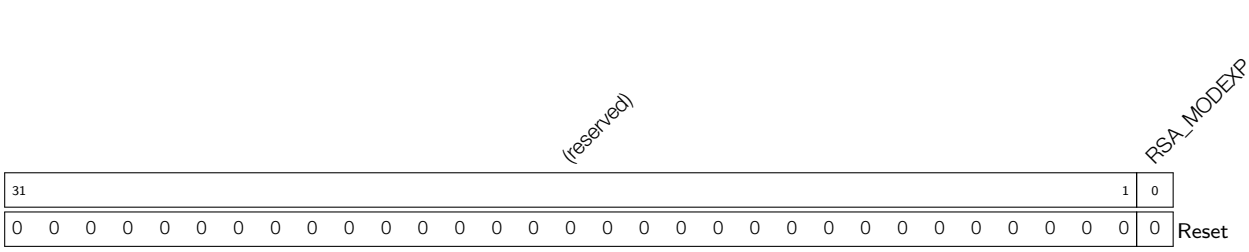
RSA_MODE Stores the mode of modular exponentiation. (R/W)

Register 21.3: RSA_CLEAN_REG (0x0808)



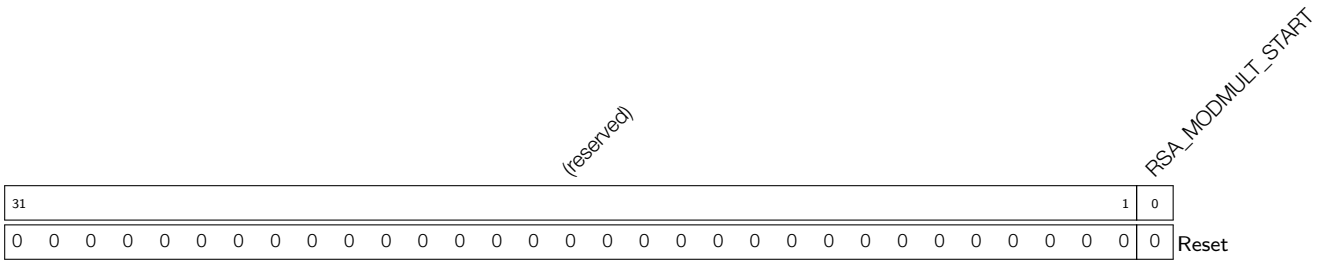
RSA_CLEAN The content of this bit is 1 when memories complete initialization. (RO)

Register 21.4: RSA_MODEXP_START_REG (0x080C)



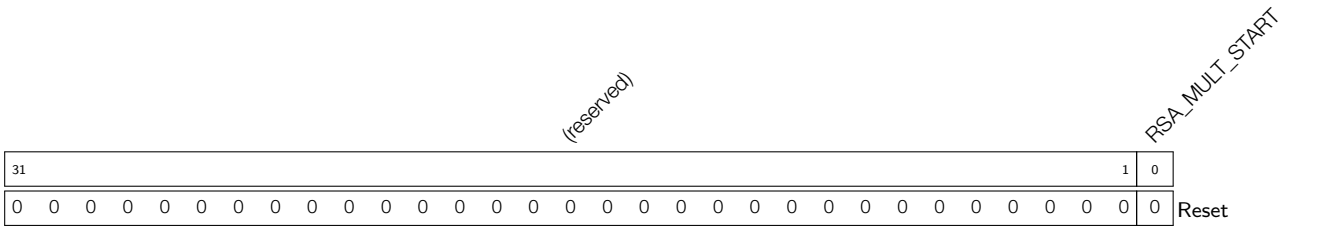
RSA_MODEXP_START Set this bit to 1 to start the modular exponentiation. (WO)

Register 21.5: RSA_MODMULT_START_REG (0x0810)



RSA_MODMULT_START Set this bit to 1 to start the modular multiplication. (WO)

Register 21.6: RSA_MULT_START_REG (0x0814)

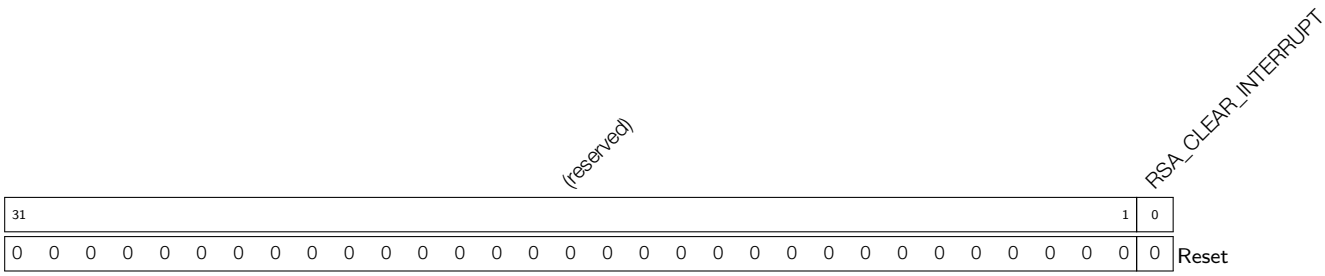


RSA_MULT_START Set this bit to 1 to start the multiplication. (WO)

Register 21.7: RSA_IDLE_REG (0x0818)

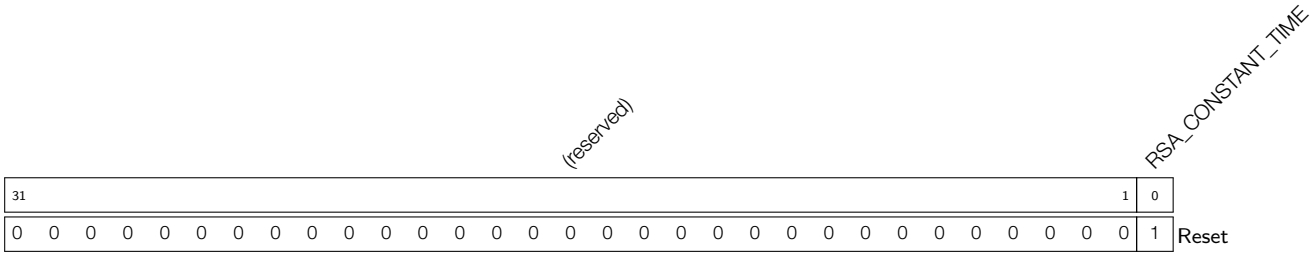
RSA_IDLE The content of this bit is 1 when the RSA accelerator is idle. (RO)

Register 21.8: RSA_CLEAR_INTERRUPT_REG (0x081C)



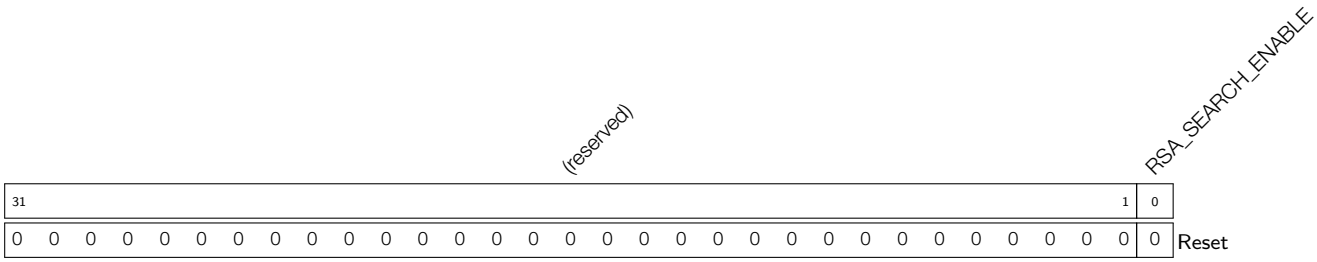
RSA_CLEAR_INTERRUPT Set this bit to 1 to clear the RSA interrupts. (WO)

Register 21.9: RSA_CONSTANT_TIME_REG (0x0820)



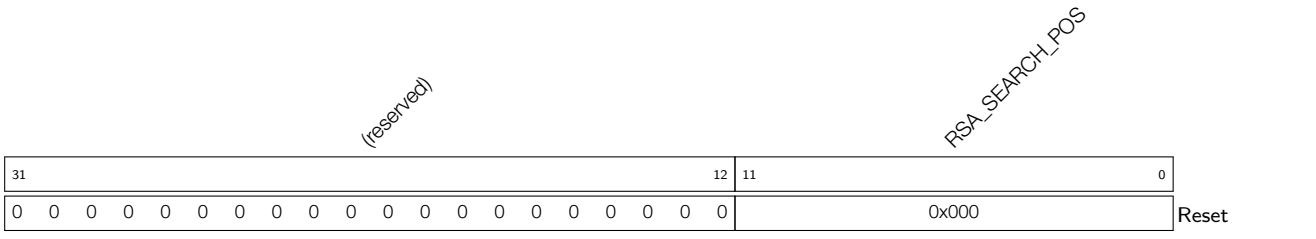
RSA_CONSTANT_TIME_REG Set this bit to 0 to enable the acceleration option of constant_time for modular exponentiation. Set to 1 to disable the acceleration (by default). (R/W)

Register 21.10: RSA_SEARCH_ENABLE_REG (0x0824)



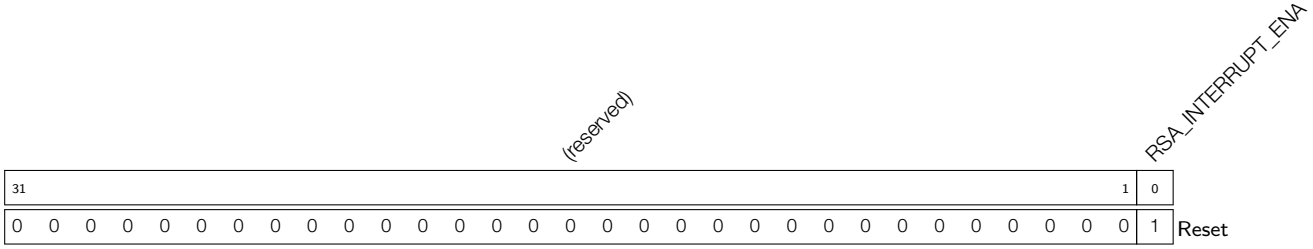
RSA_SEARCH_ENABLE Set this bit to 1 to enable the acceleration option of search for modular exponentiation. Set to 0 to disable the acceleration (by default). (R/W)

Register 21.11: RSA_SEARCH_POS_REG (0x0828)



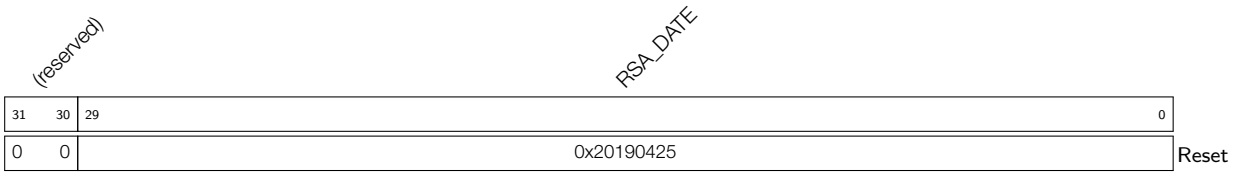
RSA_SEARCH_POS Is used to configure the starting address when the acceleration option of search is used. (R/W)

Register 21.12: RSA_INTERRUPT_ENA_REG (0x082C)



RSA_INTERRUPT_ENA Set this bit to 1 to enable the RSA interrupt. This option is enabled by default. (R/W)

Register 21.13: RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

22. Random Number Generator

22.1 Introduction

The ESP32-S2 contains a true random number generator, which generates random numbers that can be used for cryptographical operations, among other things.

22.2 Features

The random number generator generates true random numbers, which means no number generated within the specified range is more or less likely to appear than any other number.

22.3 Functional Description

When used correctly, 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.

Either the high-speed ADC, SAR ADC, or both can be used as the noise source for the random number generator.

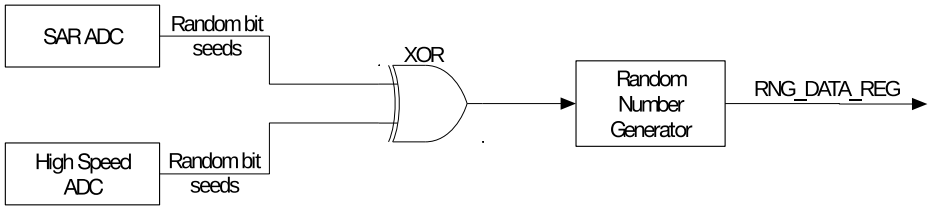


Figure 22-1. Noise Source

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 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.

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle of 8 MHz, which is generated from an internal RC oscillator (see Reset and Clock for details). Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 500 kHz to obtain the maximum entropy.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

Note:
When the Wi-Fi module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to enable the SAR ADC as the noise source for the random number generator when the Wi-Fi module is enabled.

To facilitate the generation of random numbers, a system API for generating random numbers will be provided. It's advisable to call this API directly.

22.4 Base Address

Users can access the random number generator with two base addresses, which can be seen in Table 110. For more information about accessing peripherals from different buses, please see Chapter 1 *System and Memory*.

Table 110: Random Number Generator Base Address

Bus	Base Address
PeriBUS1	0x3F435000
PeriBUS2	0x60035000

22.5 Register Summary

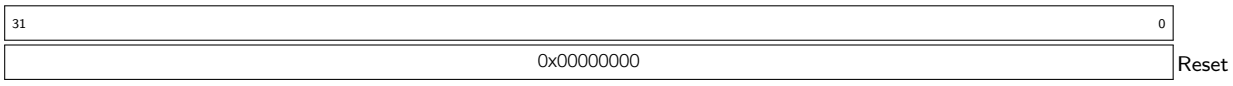
The addresses in the following table are relative to the random number generator base addresses provided in Section 22.4.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x0110	RO

22.6 Register

The address in this section is relative to the random number generator base addresses provided in Section 22.4.

Register 22.1: RNG_DATA_REG (0x0110)



RNG_DATA Random number source. (RO)

23. External Memory Encryption and Decryption

23.1 Overview

The ESP32-S2 SoC implements an External Memory Encryption and Decryption module that secures users' application code and data stored in the external memory (flash and external RAM). The encryption and decryption algorithm complies with the XTS-AES standard specified in IEEE Std 1619-2007. Users can store proprietary firmware and sensitive data (for example credentials for gaining access to a private network) to the external flash, and general data to the external RAM.

23.2 Features

- General XTS-AES algorithm, compliant with IEEE Std 1619-2007
- Software-based manual encryption
- High-speed hardware auto encryption
- High-speed hardware auto decryption
- Encryption and decryption functions jointly determined by register configuration, eFuse parameters, and boot mode

23.3 Functional Description

The External Memory Encryption and Decryption module consists of three blocks, namely the Manual Encryption block, Auto Encryption block, and Auto Decryption block. The module architecture is shown in Figure 23-1.

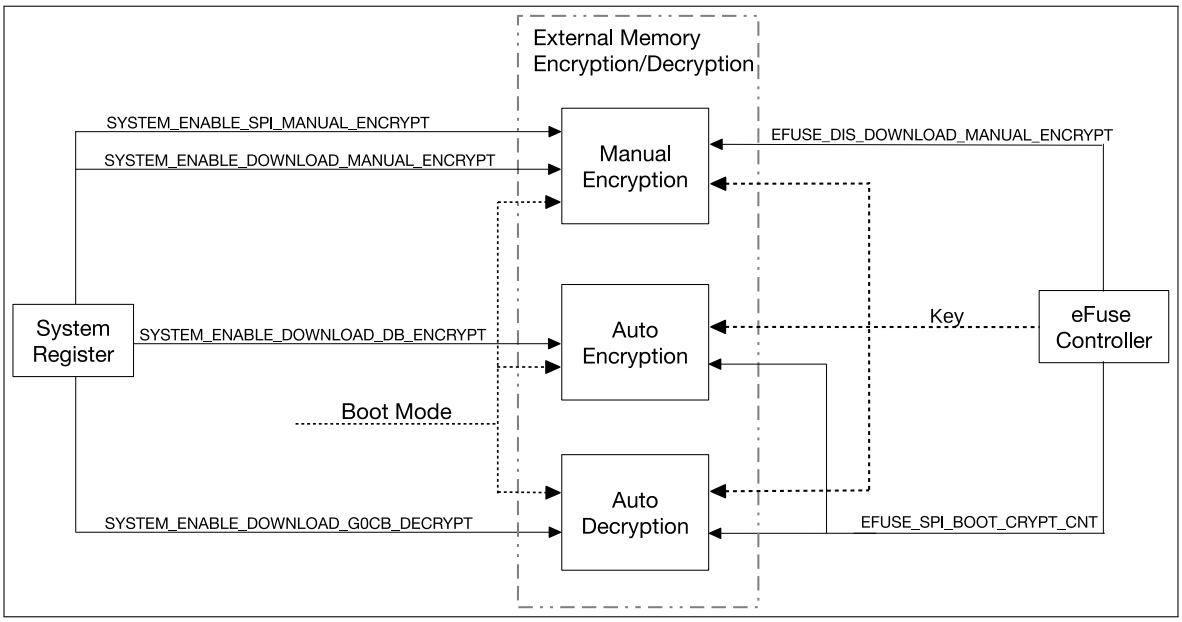


Figure 23-1. Architecture of the External Memory Encryption and Decryption Module

The Manual Encryption block can encrypt instructions and data which will then be written to the external flash as ciphertext through SPI1.

When the CPU writes the external RAM through cache, the Auto Encryption block automatically encrypts the data first, and the data is written to the external RAM as ciphertext.

When the CPU reads the external flash or RAM through cache, the Auto Decryption block automatically decrypts the ciphertext to retrieve instructions and data.

In the peripheral System Register, four bits in the SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG register are relevant to external memory encryption and decryption:

- SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

The External Memory Encryption and Decryption module fetches two parameters from the peripheral eFuse Controller. These parameters are: EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT and EFUSE_SPI_BOOT_CRYPT_CNT.

23.3.1 XTS Algorithm

The manual encryption, auto encryption, and auto decryption operations use the same algorithm, i.e., XTS algorithm. In real-life implementation, the XTS algorithm is characterized by "data unit" of 1024 bits. The "data unit" is defined in the *XTS-AES Tweakable Block Cipher* standard, section *XTS-AES encryption procedure*. More information on the XTS-AES algorithm can be found in IEEE Std 1619-2007.

23.3.2 Key

The Manual Encryption block, Auto Encryption block, and Auto Decryption block share the same key to perform XTS algorithm. The key is provided by the eFuse hardware and protected from user access.

The key can be either 256 bits or 512 bits long. The key is determined by the content in one or two eFuse blocks from BLOCK4 ~ BLOCK9. For easy description, define:

- Block$_A$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1. Block$_A$ contains 256-bit Key_A.
- Block$_B$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2. Block$_B$ contains 256-bit Key_B.
- Block$_C$, which refers to the block that has the key purpose set to EFUSE_KEY_PURPOSE_XTS_AES_128_KEY. Block$_C$ contains 256-bit Key_C.

Table 112 shows how the Key is generated, depending on whether Block$_A$, Block$_B$, and Block$_C$ exists or not.

Table 112: Key

Block$_A$	Block$_B$	Block$_C$	Key	Key Length (bit)
Yes	Yes	Don't care	$Key_A \| \| Key_B$	512
Yes	No	Don't care	$Key_A \| \| 0^{256}$	512
No	Yes	Don't care	$0^{256} \| \| Key_B$	512
No	No	Yes	Key_C	256

Block$_A$	Block$_B$	Block$_C$	Key	Key Length (bit)
No	No	No	0^{256}	256

"Yes" indicates that the block exists; "No" indicates that the block does not exist; "0^{256}" indicates a bit string that consists of 256-bit zeros; "||" is a bonding operator for joining one key string to another.

For more information on setting of key purposes, please refer to Chapter 16 *eFuse Controller*.

23.3.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory where the encrypted result is stored. The target memory space can be uniquely determined by three relevant parameters: $type$, $size$, and $base\_addr$. They are defined as follows:

- $type$: the type of the external memory, either flash or external RAM. Value 0 indicates flash, 1 indicates external RAM.
- $size$: the size of the target memory space, in unit of bytes. One single encryption operation supports either 16, 32, or 64 bytes of data.
- $base\_addr$: the base address of the target memory space. It is a physical address aligned to $size$, i.e., $base\_addr\%size == 0$.

Assume encrypted 16 bytes written to address 0x130 ~ 0x13F in the external flash, then, the target memory space is 0x130 ~ 0x13F, $type$ is 0 (flash), $size$ is 16 (bytes), and $base\_addr$ 0x130.

The encryption of any length (must be multiples of 16 bytes) of data can be completed separately in multiple operations. Each operation can have individual target memory space and the relevant parameters.

For auto encryption and auto decryption, these parameters are automatically defined by hardware. For manual encryption, these parameters should be configured manually by users.

23.3.4 Data Padding

For auto encryption and auto decryption, data padding is automatically completed by hardware. For manual encryption, data padding should be completed manually by users. The Manual Encryption block is equipped with 16 registers, i.e., XTS_AES_PLAIN_n_REG (n: 0-15), that are dedicated to data padding and can store up to 512 bits of plaintext at a time.

Actually, the Manual Encryption block does not care where the plaintext comes from, but only where the ciphertext is to be stored. Because of the strict correspondence between plaintext and ciphertext, in order to better describe how the plaintext is stored in the register heap, it is assumed 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. However, users should note that the plaintext can come from anywhere, and that they should understand how the plaintext is stored in the register heap.

How mapping works between target memory space and registers:

Assume a word is stored in $address$, define $offset = address\%64$, $n = \frac{offset}{4}$, then the word will be stored in register XTS_AES_PLAIN_n_REG.

For example, if the $size$ of the target memory space is 64, then all the 16 registers will be used for data storage. The mapping between $offset$ and registers is shown in Table 113.

Table 113: Mapping Between Offsets and Registers

$offset$	Register	$offset$	Register
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

23.3.5 Manual Encryption Block

The Manual Encryption block is a peripheral module. It is equipped with registers that can be accessed by the CPU directly. Registers embedded in this block, System registers, eFuse parameters, and boot mode jointly configure and control this block. Please note that currently the Manual Encryption block can only encrypt flash.

The manual encryption requires software participation. The steps are as follows:

1. Configure XTS_AES:
   - Set XTS_AES_DESTINATION_REG register to $type$ = 0.
   - Set XTS_AES_PHYSICAL_ADDRESS_REG register to $base\_addr$.
   - Set XTS_AES_LINESIZE_REG register to $\frac{size}{32}$.

   For definitions of $type$, $base\_addr$, $size$, please refer to Section 23.3.3.
2. Fill registers XTS_AES_PLAIN_n_REG (n: 0-15) in with plaintext (refer to Section 23.3.4). Registers that are not used can be written into any value.
3. Poll XTS_AES_STATE_REG until it reads 0 that indicates the Manual Encryption block is idle.
4. Activate encryption by writing 1 to XTS_AES_TRIGGER_REG register.
5. Wait for the encryption to complete. Poll register XTS_AES_STATE_REG until it reads 2.
   Steps 1 ~ 5 complete the encryption operation, where Key is used.
6. Grant SPI1 access to the encrypted result by writing 1 to XTS_AES_RELEASE_REG register. XTS_AES_STATE_REG will read 3 afterwards.
7. Call SPI1 and write the encrypted result to the external flash.
8. Destroy the encrypted result by writing 1 to XTS_AES_DESTROY_REG. XTS_AES_STATE_REG register will read 0 afterwards.

Repeat the steps above to complete multiple encryption operations.

The Manual Encryption block is operational only with granted permission. The operating conditions are:

- 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 is granted permission. 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 is granted permission. Otherwise, it is not operational.

Note:

- Even though the CPU can skip cache and get the encrypted result directly by reading the external memory, software can by no means access Key.
- The Manual Encryption block needs to call the AES accelerator to perform encryption. Therefore, users cannot access AES accelerator during the process.

23.3.6 Auto Encryption Block

The Auto Encryption block is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access this block. The System Register, eFuse parameters, and boot mode jointly control this block.

The Auto Encryption block is operational only with granted permission. The operating conditions are:

- In SPI Boot mode
  If the 3-bit parameter SPI_BOOT_CRYPT_CNT has 1 or 3 bits set to 1, then the Auto Encryption block is granted permission. Otherwise, it is not operational.
- In Download Boot mode
  If bit SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Encryption block is granted permission. Otherwise, it is not operational.

Note:

- When the Auto Encryption block is operational, the CPU will read data from the external RAM via the cache. The Auto Encryption block automatically encrypts the data and write it to the external RAM. The entire encryption process does not need software participation and is transparent to the cache. Software cannot access the encryption Key.
- When the Auto Encryption block is not operational, it will ignore the CPU's request to access cache and do not process the data. Therefore, data will be written to the external RAM as plaintext.

23.3.7 Auto Decryption Block

The Auto Decryption block is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access this block. The System Register, eFuse parameters, and boot mode jointly control and configure this block.

The Auto Decryption block is operational only with granted permission. The operating conditions are:

- In SPI Boot mode

If the 3-bit parameter SPI_BOOT_CRYPT_CNT has 1 or 3 bits set to 1, then the Auto Decryption block is granted permission. Otherwise, it is not operational.

- In Download Boot mode
  If bit SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Decryption block is granted permission. Otherwise, it is not operational.

Note:

- When the Auto Decryption block is operational, the CPU will read instructions and data from the external memory via cache. The Auto Decryption block automatically decrypts and retrieves the instructions and data. The entire decryption process does not need software participation and is transparent to the cache. Software cannot access the decryption Key.
- When the Auto Decryption block is not operational, it does not have any effect on the contents stored in the external memory, be they encrypted or unencrypted. What the CPU reads via cache is the original information stored in the external memory.

23.4 Base Address

Users can access the Manual Encryption block with two base addresses, which can be seen in the following table. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 114: Manual Encryption Block Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

23.5 Register Summary

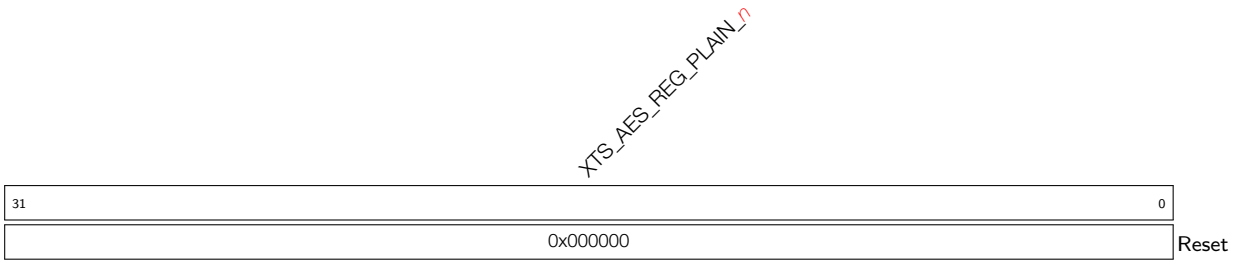
The addresses in the following table are relative to the Manual Encryption block's base addresses provided in Section 23.4.

Name	Description	Address	Access
Plaintext Register Heap			
XTS_AES_PLAIN_0_REG	Plaintext register 0	0x0100	R/W
XTS_AES_PLAIN_1_REG	Plaintext register 1	0x0104	R/W
XTS_AES_PLAIN_2_REG	Plaintext register 2	0x0108	R/W
XTS_AES_PLAIN_3_REG	Plaintext register 3	0x010C	R/W
XTS_AES_PLAIN_4_REG	Plaintext register 4	0x0110	R/W
XTS_AES_PLAIN_5_REG	Plaintext register 5	0x0114	R/W
XTS_AES_PLAIN_6_REG	Plaintext register 6	0x0118	R/W
XTS_AES_PLAIN_7_REG	Plaintext register 7	0x011C	R/W
XTS_AES_PLAIN_8_REG	Plaintext register 8	0x0120	R/W
XTS_AES_PLAIN_9_REG	Plaintext register 9	0x0124	R/W
XTS_AES_PLAIN_10_REG	Plaintext register 10	0x0128	R/W

Name	Description	Address	Access
XTS_AES_PLAIN_11_REG	Plaintext register 11	0x012C	R/W
XTS_AES_PLAIN_12_REG	Plaintext register 12	0x0130	R/W
XTS_AES_PLAIN_13_REG	Plaintext register 13	0x0134	R/W
XTS_AES_PLAIN_14_REG	Plaintext register 14	0x0138	R/W
XTS_AES_PLAIN_15_REG	Plaintext register 15	0x013C	R/W
Configuration Registers			
XTS_AES_LINESIZE_REG	Configures the size of target memory space	0x0140	R/W
XTS_AES_DESTINATION_REG	Configures the type of the external memory	0x0144	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	Physical address	0x0148	R/W
Control/Status Registers			
XTS_AES_TRIGGER_REG	Activates AES algorithm	0x014C	WO
XTS_AES_RELEASE_REG	Release control	0x0150	WO
XTS_AES_DESTROY_REG	Destroys control	0x0154	WO
XTS_AES_STATE_REG	Status register	0x0158	RO
Version Register			
XTS_AES_DATE_REG	Version control register	0x015C	RO

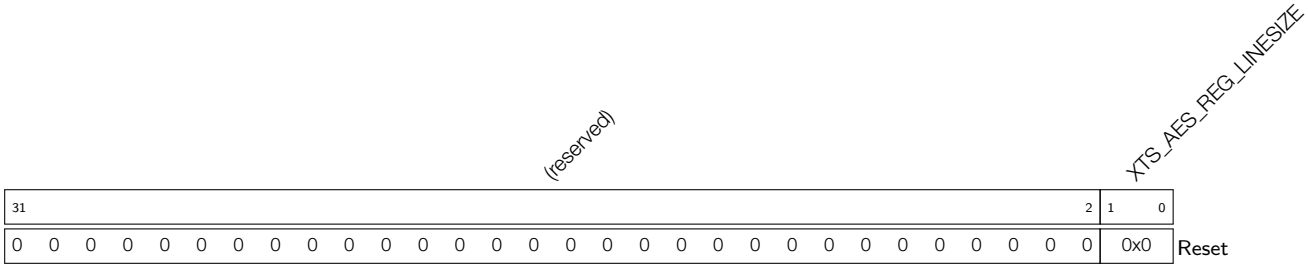
23.6 Registers

Register 23.1: XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0100+4n*)**



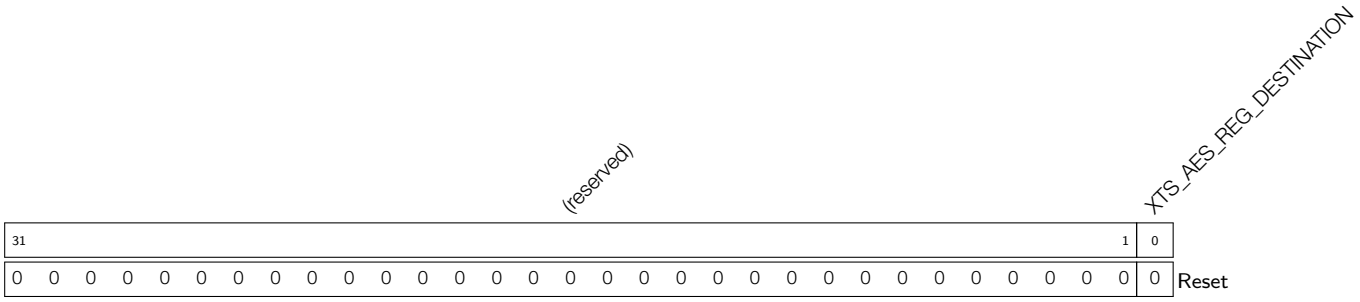
XTS_AES_REG_PLAIN_*n* This register stores *n*th 32-bit piece of plaintext. (R/W)

Register 23.2: XTS_AES_LINESIZE_REG (0x0140)



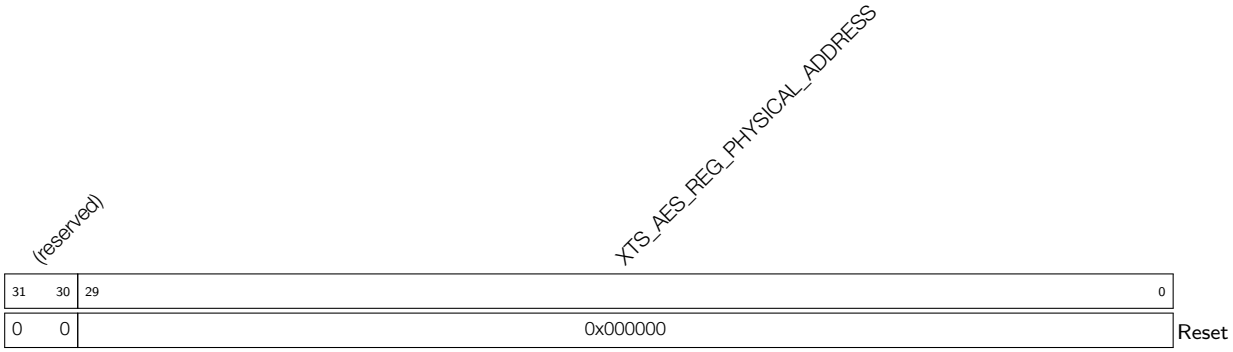
XTS_AES_REG_LINESIZE Configures the data size of a single encryption. 0: 128 bits; 1: 256 bits; 2: 512 bits. (R/W)

Register 23.3: XTS_AES_DESTINATION_REG (0x0144)



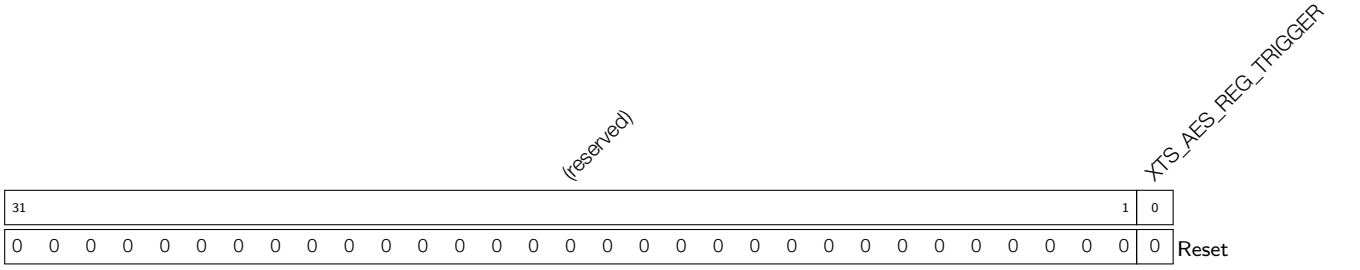
XTS_AES_REG_DESTINATION Configures the type of the external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Errors may occur if users write 1. 0: flash; 1: external RAM. (R/W)

Register 23.4: XTS_AES_PHYSICAL_ADDRESS_REG (0x0148)



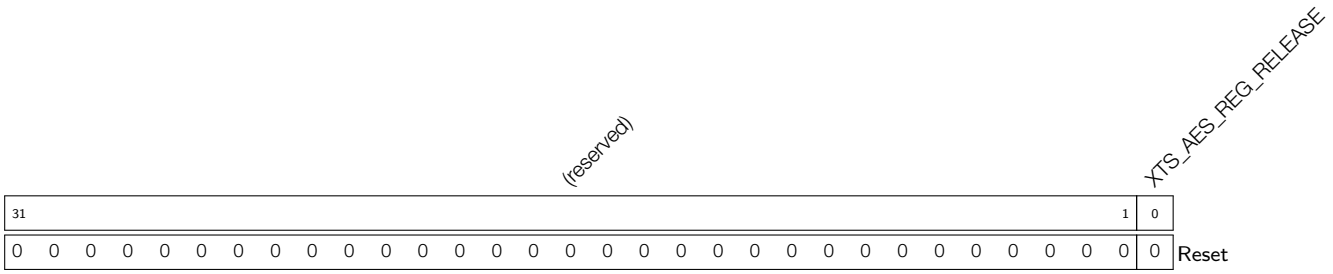
XTS_AES_REG_PHYSICAL_ADDRESS Physical address. (R/W)

Register 23.5: XTS_AES_TRIGGER_REG (0x014C)



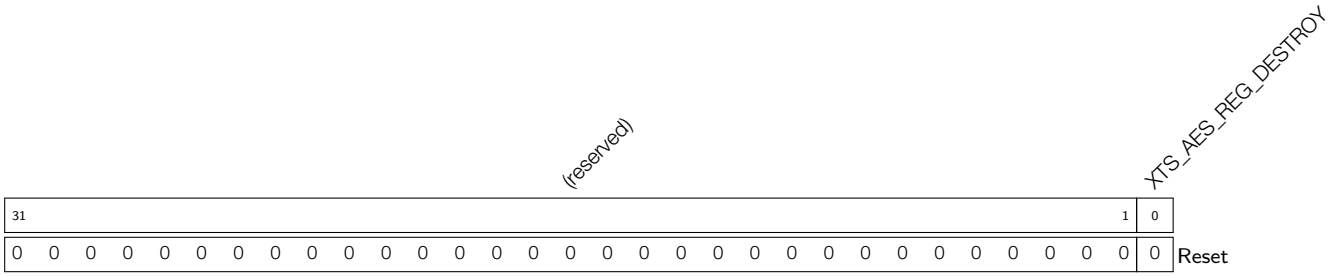
XTS_AES_REG_TRIGGER Set to enable manual encryption. (WO)

Register 23.6: XTS_AES_RELEASE_REG (0x0150)



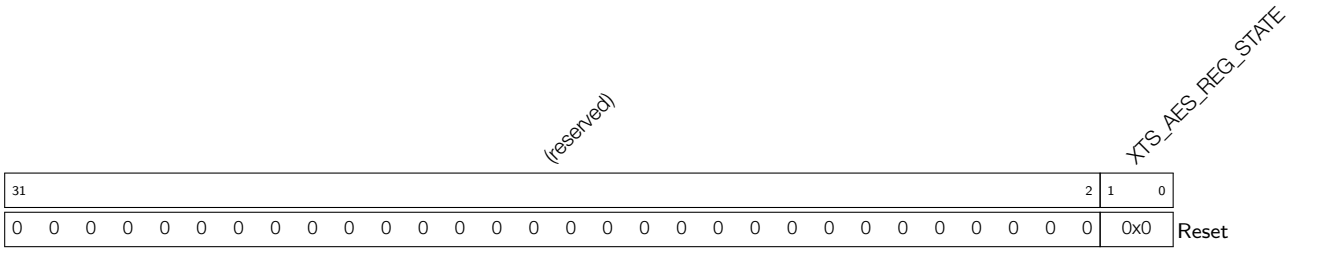
XTS_AES_REG_RELEASE Set to grant SPI1 access to encrypted result. (WO)

Register 23.7: XTS_AES_DESTROY_REG (0x0154)



XTS_AES_REG_DESTROY Set to destroy encrypted result. (WO)

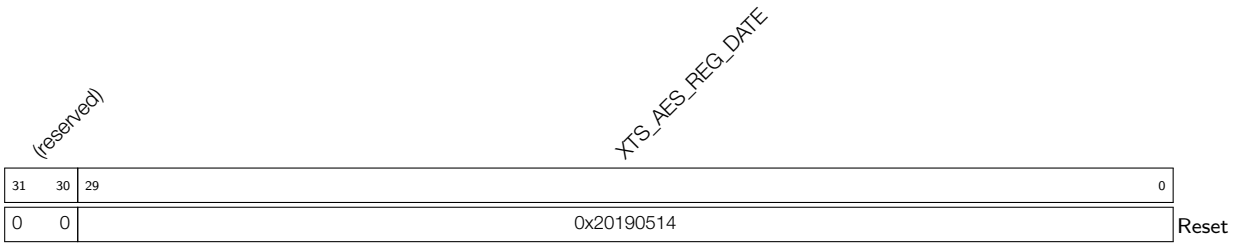
Register 23.8: XTS_AES_STATE_REG (0x0158)



XTS_AES_REG_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 23.9: XTS_AES_DATE_REG (0x015C)



XTS_AES_REG_DATE Version control register. (RO)

24. Permission Control

24.1 Overview

In ESP32-S2, different type of buses and modules can have different permissions to access the memory. The permission control module is used to manage all the access permissions through corresponding permission control registers.

24.2 Features

- Permission controls for the CPU buses to access the internal memory
  - SRAM partially adopts unified permission control and partially adopts split permission control.
  - RTC FAST SRAM adopts split permission control.
  - RTC SLOW SRAM adopts split permission control.
- Access to the external memory is managed by both the memory management unit (MMU) and the permission control module.

24.3 Functional Description

24.3.1 Internal Memory Permission Controls

The internal memory resources in ESP32-S2 include ROM, SRAM, RTC FAST Memory, and RTC SLOW Memory. The breakdown of each memory is as follows:

- The total ROM size is 128 KB and consists of two 64 KB blocks.
- The total SRAM size is 320 KB and consists of four 8 KB blocks and eighteen 16 KB blocks, numbered Block 0 ~ 21. The offset address range of each block is shown in Table 116. For the base addresses accessed by different buses please refer to Section 24.3.1.1, 24.3.1.2, and 24.3.1.3. Each of the four 8 KB blocks (Block 0 ~ 3) is equipped with an independent permission control register. The eighteen 16 KB blocks (Block 4 ~ 21) are regarded as a large storage space as a whole, which is managed in split parts (i.e., the storage space is split into high and low areas with separate permission controls). The split address cannot fall within the address range of Block 4 ~ 5.
- RTC FAST Memory adopts split permission control. The high and low areas have independent permission control registers.
- RTC SLOW Memory also adopts split permission control. The high and low areas have independent permission control registers.

The following sections describe the buses and modules in ESP32-S2 that can access the above-mentioned internal memory resources.

Table 116: Offset Address Range of Each SRAM Block

SRAM Block	Offset Start Address	Offset End Address
Block 0	0x0000	0x1FFF
Block 1	0x2000	0x3FFF
Block 2	0x4000	0x5FFF

SRAM Block	Offset Start Address	Offset End Address
Block 3	0x6000	0x7FFF
Block 4	0x8000	0xBFFF
Block 5	0xC000	0xFFFF
Block 6	0x10000	0x13FFF
Block 7	0x14000	0x17FFF
Block 8	0x18000	0x1BFFF
Block 9	0x1C000	0x1FFFF
Block 10	0x20000	0x23FFF
Block 11	0x24000	0x27FFF
Block 12	0x28000	0x2BFFF
Block 13	0x2C000	0x2FFFF
Block 14	0x30000	0x33FFF
Block 15	0x34000	0x37FFF
Block 16	0x38000	0x3BFFF
Block 17	0x3C000	0x3FFFF
Block 18	0x40000	0x43FFF
Block 19	0x44000	0x47FFF
Block 20	0x48000	0x4BFFF
Block 21	0x4C000	0x4FFFF

24.3.1.1 Permission Control for the Instruction Bus (IBUS)

The address range on IBUS that requires permission control is 0x4002_0000 ~ 0x4007_1FFF, where SRAM memory blocks and RTC FAST Memory are located. Access to different memory blocks is controlled by independent permission control registers which are configured by software.

The permission control registers related to IBUS are also controlled by the PMS_PRO_IRAM0_LOCK signal. When this signal is set to 1, the values configured in the permission control registers will be locked and cannot be changed. At the same time, the value of the PMS_PRO_IRAM0_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_IRAM0_LOCK signal is reset to 0 only when the CPU is reset.

SRAM Memory Blocks

Through IBUS, the CPU can Read (data) from SRAM, Write (data) to SRAM, and eXecute (an instruction), indicated as R/W/X, respectively. The base address of SRAM accessed by the CPU is 0x4002_0000. Software can configure valid access types to each SRAM block in advance. The configuration of the permission control registers is shown in Table 117.

Table 117: Permission Control for IBUS to Access SRAM

Register	Bit Positions	Description
PMS_PRO_IRAM0_1_REG	[11:9]	Configure the permission for IBUS to access SRAM Block 3 (from high to low as W/R/X)
	[8:6]	Configure the permission for IBUS to access SRAM Block 2 (from high to low as W/R/X)

Register	Bit Positions	Description
	[5:3]	Configure the permission for IBUS to access SRAM Block 1 (from high to low as W/R/X)
	[2:0]	Configure the permission for IBUS to access SRAM Block 0 (from high to low as W/R/X)
PMS_PRO_IRAM0_2_REG	[22:20]	Configure the permission for IBUS to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R/X)
	[19:17]	Configure the permission for IBUS to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R/X)
	[16:0]*	Configure the split address of IBUS in the SRAM Block 4 ~ 21 address range. The base address is 0x4000_0000. See below for instructions on how to configure split address.

Note:

* **Configuring split address:**

Split address is 32 bits in width and equal to base address plus the configured field value multiplied by 4. For example, if 0x10000 is written to PMS_PRO_IRAM0_2_REG [16:0], the split address of IBUS in the SRAM Block 4 ~ 21 address range is equal to 0x4004_0000. Note that the split address on IBUS and DBUS0 cannot fall within the address range of Block 0 ~ 5.

If the type of access initiated by the CPU to SRAM through IBUS does not match the configured access type, the permission control module will directly deny this access. For example, if software allows the IBUS bus to only read and write SRAM Block5, but the CPU initiates an instruction fetch, the permission control module will deny this fetch access.

An access denial does not block the access initiated by the CPU, which means:

- The write operation will not take effect and will not change the data in the memory block.
- Read and fetch operations will get invalid data.

If the interrupt that indicates an illegitimate access to SRAM via IBUS is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

RTC FAST Memory

The CPU can perform R/W/X access to the RTC FAST Memory through IBUS. As RTC FAST Memory adopts split permission control, software needs to configure access permissions for the high and low address ranges, respectively. The configuration of the permission control registers is shown in Table 118.

Table 118: Permission Control for IBUS to Access RTC FAST Memory

Register	Bit Positions	Description
PMS_PRO_IRAM0_3_REG	[16:14]	Configure the permission for IBUS to access the high address range in RTC FAST Memory (from high to low as W/R/X)

Register	Bit Positions	Description
	[13:11]	Configure the permission for IBUS to access the low address range in RTC FAST Memory (from high to low as W/R/X)
	[10:0]	Configure the user-configurable split address of IBUS in the RTC FAST address range. The base address is 0x4007_0000. For how to configure the user-configurable split address, please refer to the instructions above.

If the type of access initiated by the CPU to RTC Fast Memory through IBUS does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to the RTC Fast Memory via IBUS is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

24.3.1.2 Permission Control for the Data Bus (DBUS0)

The CPU can access two address ranges through DBUS0: 0x3FFB_0000 ~ 0x3FFF_FFFF and 0x3FF9_E000 ~ 0x3FF9_FFFF, where SRAM memory blocks and RTC FAST Memory are located. Access to different memory blocks is controlled by independent permission control registers which are configured by software.

The permission control registers related to DBUS0 are also controlled by the PMS_PRO_DRAM0_LOCK signal. When this signal is set to 1, the configured values of the permission control registers will be locked and cannot be changed. The value of the PMS_PRO_DRAM0_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_DRAM0_LOCK signal is reset to 0 only when the CPU is reset.

SRAM Memory Blocks

The CPU can perform R/W access to SRAM via DBUS0, with the based address 0x3FFB_0000. Software can configure allowed access types initiated by DBUS0 to each SRAM block in advance. The configuration of the permission control registers is shown in Table 119.

Table 119: Permission Control for DBUS0 to Access SRAM

Register	Bit Positions	Description
PMS_PRO_DRAM0_1_REG	[28:27]	Configure the permission for DBUS0 to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[26:25]	Configure the permission for DBUS0 to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[24:8]	Configure the user-configurable split address of DBUS0 in the SRAM Block 4 ~ 21 address range. The base address is 0x3FFB_0000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.
	[7:6]	Configure the permission for DBUS0 to access SRAM Block3 (from high to low as W/R)
	[5:4]	Configure the permission for DBUS0 to access SRAM Block2 (from high to low as W/R)

Register	Bit Positions	Description
	[3:2]	Configure the permission for DBUS0 to access SRAM Block1 (from high to low as W/R)
	[1:0]	Configure the permission for DBUS0 to access SRAM Block0 (from high to low as W/R)

If the type of access initiated by the CPU to SRAM through DBUS0 does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to SRAM via DBUS0 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

RTC FAST Memory

The CPU can perform R/W access to RTC FAST Memory through DBUS0. As RTC FAST Memory adopts split permission control, software needs to configure access permissions for the high and low address ranges, respectively. The configuration of the permission control registers is shown in Table 120.

Table 120: Permission Control for DBUS0 to Access RTC FAST Memory

Register	Bit Positions	Description
PMS_PRO_DRAM0_2_REG	[14:13]	Configure the permission for DBUS0 to access the high address range in RTC FAST Memory (from high to low as W/R)
	[12:11]	Configure the permission for DBUS0 to access the low address range in RTC FAST Memory (from high to low as W/R)
	[10:0]	Configure the user-configurable split address of DBUS0 within the RTC FAST Memory address range. The base address is 0x3FF9_E000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.

If the type of access initiated by the CPU to RTC FAST Memory through DBUS0 does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to RTC FAST Memory via DBUS0 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

24.3.1.3 Permission Control for On-chip DMA

The on-chip DMA can access the internal memory through Internal DMA, Copy DMA RX (receiving) channel, or Copy DMA TX (transmitting) channel, using the address range 0x3FFB_0000 ~ 0x3FFF_FFFF (0x3FFB_0000 is the base address). Software can configure the type of DMA access to each SRAM block in advance. The configuration of the permission control registers is shown in Table 121, where XX can be APB, TX, or RX,

corresponding to the configuration registers of Internal DMA, TX Copy DMA, and RX Copy DMA, respectively. More details about DMA can be found in Chapter 7 *DMA Controller*.

The permission control registers related to DMA are also controlled by the PMS_DMA_XX_I_LOCK signal. When this signal is set to 1, the configured values of the permission control registers will be locked and cannot be changed. The value of the PMS_DMA_XX_I_LOCK signal will also remain at 1 and cannot be changed. The PMS_DMA_XX_I_LOCK signal value is reset to 0 only when the CPU is reset.

Table 121: Permission Control for On-chip DMA to Access SRAM

Register	Bit Positions	Description
PMS_DMA_XX_I_1_REG	[28:27]	Configure the permission for DMA to access the high address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[26:25]	Configure the permission for DMA to access the low address range in SRAM Block 4 ~ 21 (from high to low as W/R)
	[24:8]	Configure the user-configurable split address of DMA in SRAM Block 4 ~ 21 address range. The base address is 0x3FFB_0000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.
	[7:6]	Configure the permission for DMA to access SRAM Block3 (from high to low as W/R)
	[5:4]	Configure the permission for DMA to access SRAM Block2 (from high to low as W/R)
	[3:2]	Configure the permission for DMA to access SRAM Block1 (from high to low as W/R)
	[1:0]	Configure the permission for DMA to access SRAM Block0 (from high to low as W/R)

If the type of access initiated by the on-chip DMA to SRAM does not match the configured access type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to SRAM via Internal DMA, TX Copy DMA, or RX Copy DMA is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

24.3.1.4 Permission Control for PeriBus1

The CPU can access the address range 0x3F40_0000 ~ 0x3F4F_FFFF through PeriBus1, where RTC SLOW SRAM is located. PeriBus1 can perform R/W operations. The read/write permission is controlled by software.

Software can also configure whether PeriBus1 is allowed to access peripherals that are within the address range 0x3F40_0000 ~ 0x3F4B_FFFF.

As the CPU can initiate predictive read operations through PeriBus1, it may cause peripheral FIFO read errors. To avoid FIFO read errors, the FIFO addresses of the corresponding peripherals have already been written in hardware and can no longer be changed, as Table 122 shows. In addition, four software-configurable address

registers PMS_PRO_DPORT_2 ~ 5_REG are reserved for users to write into addresses read-protected from PeriBus1 reading. The configuration of the permission control registers is shown in Table 123.

Table 122: Peripherals and FIFO Address

Peripherals	FIFO Address
ADDR_RTCSLOW	0x6002_1000
ADDR_FIFO_UART0	0x6000_0000
ADDR_FIFO_UART1	0x6001_0000
ADDR_FIFO_UART2	0x6002_E000
ADDR_FIFO_I2S0	0x6000_F004
ADDR_FIFO_I2S1	0x6002_D004
ADDR_FIFO_RMT_CH0	0x6001_6000
ADDR_FIFO_RMT_CH1	0x6001_6004
ADDR_FIFO_RMT_CH2	0x6001_6008
ADDR_FIFO_RMT_CH3	0x6001_600C
ADDR_FIFO_I2C_EXT0	0x6001_301C
ADDR_FIFO_I2C_EXT1	0x6002_701C
ADDR_FIFO_USB_0	0x6008_0020
ADDR_FIFO_USB_1_L	0x6008_1000
ADDR_FIFO_USB_1_H	0x6009_0FFF

The permission control registers related to PeriBus1 are also controlled by the PMS_PRO_DPORT_LOCK signal. When this signal is set to 1, the configured values of the permission registers will be locked and cannot be changed. The value of the PMS_PRO_DPORT_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_DPORT_LOCK signal is reset to 0 only when the CPU is reset.

Table 123: Permission Control for PeriBus1

Register	Bit Positions	Description
PMS_PRO_DPORT_1_REG	[19:16]	The value of each bit determines whether to enable the corresponding reserved FIFO address.
	[15:14]	Configure the permission for PeriBus1 to access the high address range in RTC SLOW Memory (from high to low as W/R)
	[13:12]	Configure the permission for PeriBus1 to access the low address range in RTC SLOW Memory (from high to low as W/R)
	[11:1]	Configure the user-configurable split address of PeriBus1 in the RTC SLOW Memory address range. The base address is 0x3F42_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.
	[0]	Configure whether PeriBus1 can access the peripherals within the address range 0x3F40_0000 ~ 0x3F4B_FFFF
PMS_PRO_DPORT_2_REG	[17:0]	The zeroth address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [16]
PMS_PRO_DPORT_3_REG	[17:0]	The first address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [17]

Register	Bit Positions	Description
PMS_PRO_DPORT_4_REG	[17:0]	The second address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [18]
PMS_PRO_DPORT_5_REG	[17:0]	The third address unreadable to PeriBus1, enabled by PMS_PRO_DPORT_RESERVE_FIFO_VALID [19]

If PeriBus1 initiates a read access to unreadable address, or if the type of the access to RTC SLOW Memory does not match the configured type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access to RTC SLOW Memory by PeriBus1 is enabled, the permission control module will record the current access address, access type, and access size (in bytes, halfwords, or words), and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

24.3.1.5 Permission Control for PeriBus2

The two address ranges on PeriBus2 that require permission control are 0x5000_0000 ~ 0x5000_1FFF and 0x6000_0000 ~ 0x600B_FFFF where RTC SLOW Memory and peripheral modules are located.

PeriBus2 supports R/W/X access to RTC SLOW Memory through two address ranges, namely RTCSlow_0 and RTCSlow_1. Software can configure the allowed access types. The configuration of the permission control registers is shown in Table 124.

PeriBus2 does not support fetch operations on peripherals. If a fetch operation is initiated on a peripheral, invalid data will be obtained.

The permission control registers related to PeriBus2 are also controlled by the PMS_PRO_AHB_LOCK signal. When this signal is configured to 1, the values of the permission control registers are locked and cannot be changed. The value of the PMS_PRO_AHB_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_AHB_LOCK signal is reset to 0 only when the CPU is reset.

Table 124: Permission Control for PeriBus2 to Access RTC SLOW Memory

Register	Bit Positions	Description
PMS_PRO_AHB_1_REG	[16:14]	Configure the permission for PeriBus2 to access the high address range in RTCSlow_0 (from high to low as W/R/X)
	[13:11]	Configure the permission for PeriBus2 to access the low address range in RTCSlow_0 (from high to low as W/R/X)
	[10:0]	Configure the user-configurable split address of PeriBus2 in the RTCSlow_0 address range. The base address is 0x5000_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.
PMS_PRO_AHB_2_REG	[16:14]	Configure the permission for PeriBus2 to access the high address range in RTCSlow_1 (from high to low as W/R/X)
	[13:11]	Configure the permission for PeriBus2 to access the low address range in RTCSlow_1 (from high to low as W/R/X)

Register	Bit Positions	Description
	[10:0]	Configure the user-configurable split address of PeriBus2 in the RTCSlow_1 address range. The base address is 0x6002_1000. For how to configure the user-configurable split address, please refer to the instructions in Section 24.3.1.1.

If the type of the access to RTCSlow_0 or RTCSlow_1 made by the CPU through PeriBus2 does not match the configured type, the permission control module will directly deny this access. If the interrupt that indicates an illegitimate access by PeriBus2 is enabled, the permission control module will record the current access address and access type, and give an interrupt signal at the same time. If the access is denied repeatedly, the hardware only records the first error message. The access error interrupt is a level signal and needs to be cleared by software. When the interrupt is cleared, the related error record is also cleared at the same time.

24.3.1.6 Permission Control for Cache

Among the SRAM blocks, only Block 0 ~ 3 can be assigned to Icache and Dcache. Icache or Dcache can access up to 16 KB internal memory, which means, they can access up to two blocks. The 16 KB address range accessed by Icache or Dcache is divided into two ranges: high address range (indicated by "_H" and low address range (indicated by "_L").

The user can allocate Block 0 ~ 3 to Icache_H, Icache_L, Dcache_H, or Dcache_L by configuring the field PMS_PRO_CACHE_CONNECT in register PMS_PRO_CACHE_1_REG. The detailed configuration is shown in Table 125, where *FIELD* indicates field PMS_PRO_CACHE_CONNECT.

Table 125: Configuration of Register PMS_PRO_CACHE_1_REG

SRAM Block 0-3	Dcache_H	Dcache_L	Icache_H	Icache_L
Block 0	*FIELD*[3]	*FIELD*[2]	*FIELD*[1]	*FIELD*[0]
Block 1	*FIELD*[7]	*FIELD*[6]	*FIELD*[5]	*FIELD*[4]
Block 2	*FIELD*[11]	*FIELD*[10]	*FIELD*[9]	*FIELD*[8]
Block 3	*FIELD*[15]	*FIELD*[14]	*FIELD*[13]	*FIELD*[12]

Notes:

- One block can be assigned to only one of Dcache_H, Dcache_L, Icache_H, and Icache_L, which means that one bit at most can be set to 1 in the same row in Table 125.
- One of Dcache_H, Dcache_L, Icache_H, and Icache_L can occupy only one block, which means that one bit at most can be set to 1 in the same column in Table 125.
- SRAM memory occupied by the cache cannot be accessed by the CPU. Non-occupied SRAM memory can still be accessed by the CPU.

24.3.1.7 Permission Control of Other Types of Internal Memory

In ESP32-S2, software can read Trace memory to get information about the runtime status of the CPU. Additionally, it can also be used as a debug channel to communicate with software running on the CPU. Software can enable the Trace memory function by configuring register PMS_PRO_TRACE_1. This register is also controlled by the PMS_PRO_TRACE_LOCK signal. When this signal is set to 1, the value configured in the

register will be locked and cannot be changed. At the same time, the value of the PMS_PRO_TRACE_LOCK signal will also remain at 1 and cannot be changed. The value of the PMS_PRO_TRACE_LOCK signal is reset to 0 only when the CPU is reset.

After the Trace memory function is enabled, register PMS_OCCUPY_3 needs to be configured to select one block from SRAM Block 4 ~ 21 as Trace memory. Only one block can be used as Trace memory, in which case, it can no longer be accessed by the CPU. Register PMS_OCCUPY_3 is controlled by the PMS_OCCUPY_LOCK signal. When this signal is configured as 1, the value configured in the register will be locked and cannot be changed. At the same time, the value of the PMS_OCCUPY_LOCK signal will also remain at 1 and cannot be changed. The PMS_OCCUPY_LOCK signal value is reset to 0 the CPU is reset.

When an SRAM block is selected as Trace Memory, neither IBUS, DBUS, nor DMA can access this block.

24.3.2 External Memory Permission Control

The CPU can access the external flash and SRAM by SPI1, EDMA or cache. The former two support direct access, and the cache supports indirect access.

24.3.2.1 Cache MMU

The Cache MMU is used to control the permission for cache and EDMA to access the external memory. Its primary function is to convert virtual addresses to physical addresses. The MMU needs to be configured before enabling cache and EDMA. When a cache miss occurs or the data is written back to the external memory, the cache controller will automatically access the MMU and generate a physical address to access the external memory. When EDMA reads and writes the external SRAM, the EDMA controller also automatically accesses the MMU and generates a physical address to access the external SRAM.

Table 126: MMU Entries

MMU Entries	Bit Positions	Description
SRAM	[16]	The external memory attribute indicates whether cache/EDMA accesses the external flash or SRAM. If bit 15 is set, cache/EDMA accesses flash. If bit 16 is set, cache/EDMA accesses SRAM. The two bits cannot be set at the same time.
Flash	[15]	
Invalid	[14]	Cleared if MMU entry is valid.
Page number	[13:0]	For MMU purposes, external memory is divided in memory blocks called 'pages'. The page size is fixed at 64 KB. The page number of the physical address space indicates which page is accessed by cache/EDMA.

When cache or EDMA accesses an invalid MMU page or the external memory attribute is not specified in the MMU, an MMU error interrupt will be triggered.

24.3.2.2 External Memory Permission Controls

The hardware divides the physical address of the external memory (flash + SRAM) into eight (4 + 4) areas. Each area can be individually configured for W/R/X access. Software needs to configure the size of each area and its access type in advance.

The eight areas correspond to eight sets of registers, and the flash and SRAM each has four sets. Each set of registers contains three parts as follows:

1. Attribute list: APB_CTRL_X_ACS_n_ATTR_REG has 3 bits in total, which are W/R/X bits from high to low;
2. Area start address: APB_CTRL_X_ACS_n_ADDR_REG, which represents the physical address;
3. Area length: APB_CTRL_X_ACS_n_SIZE_REG in multiples of 64 KB;

Wherein "X" represents flash or SRAM, "n" represents 0 ~ 3. Note that the total size of the four areas corresponding to flash must be 1 GB, so is the total size of the four areas corresponding to SRAM.

When the CPU directly accesses external memory, the permission control module only monitors the CPU's write requests, not the read requests. The control module checks whether the access type matches the configured type according to the accessed physical address. If the area does not allow write access, the permission module directly rejects write access, records the information about the current access, and triggers an access error interrupt.

Cache miss, cache write back, and cache prefetch operations will trigger W/R/X requests from the cache to the external memory. The permission control module will query for the access attributes of the area according to the accessed physical address, and compare the the cache request against the pre-configured attributes. Only when the two match, the permission control module will pass the request to the external memory. If the cache initiates an R/F request, the permission control module will send the access attributes of the area to the cache for storage. When the access request is inconsistent with the current access attributes, the hardware records the information about the current access and triggers an access error interrupt.

When the CPU accesses the cache, the permission control module must check the attributes of the CPU access by comparing it with the local access attribute list. Only if the two match, the CPU access is allowed. When the access request is inconsistent with the current access attributes, the hardware records the information about the current access and triggers an access error interrupt.

If the access attribute check fails repeatedly, the hardware only records the first error message, and the subsequent error messages will be discarded directly. The access error interrupt is a level signal. When software clears the access error interrupt flag, the error record will be cleared at the same time.

24.3.3 Non-Aligned Access Permission Control

ESP32-S2 supports monitoring of non-word-aligned access. When PeriBus1 or PeriBus2 initiates a non-word address alignment or non-word sized access to a peripheral device, an interrupt or system exception may be triggered. The following table lists all possible access types and their corresponding results, where *IN* means interrupt, *EX* means exceptions, and $\surd$ means no action is triggered.

Table 127: Non-Aligned Access to Peripherals

Accessed by	Access Address	Access Unit	Read	Write
PeriBus2	0xXXXX_XXX0	byte	IN	IN
		halfword	IN	IN
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	√	IN
		word	√	IN
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	√	IN
Address range 0x3F40_0000 - 0x3F4B_FFFF	0xXXXX_XXX0	byte	IN	IN
		halfword	IN	IN
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	EX	EX
		word	EX	EX
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	EX	EX
Address range 0x3F4C_0000 - 0x3F4F_FFFF	0xXXXX_XXX0	byte	√	√
		halfword	√	√
		word	√	√
	0xXXXX_XXX1	byte	IN	IN
		halfword	EX	EX
		word	EX	EX
	0xXXXX_XXX2	byte	IN	IN
		halfword	IN	IN
		word	EX	EX

24.4 Base Address

Users can access the permission control module with the base address as shown in Table 128. For more information, see Chapter *System and Memory*.

Table 128: Permission Control Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F4C1000

24.5 Register Summary

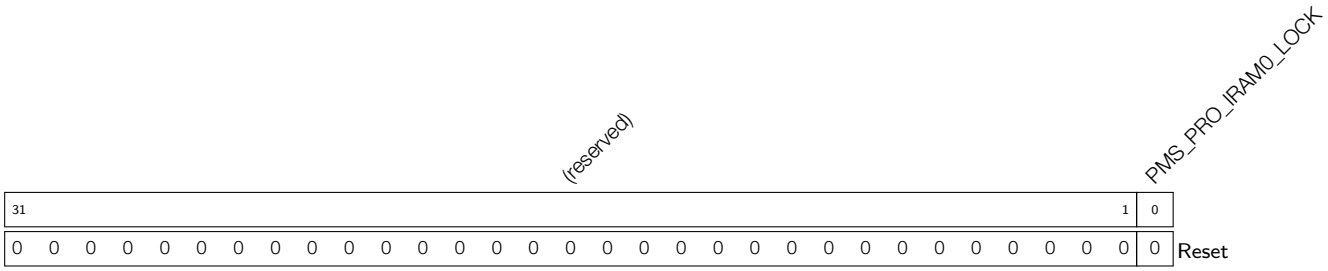
The address in the following table represents the address offset (relative address) with the respect to the peripheral base address, not the absolute address. For detailed information about the permission control base address, please refer to Section 24.4.

Name	Description	Address	Access
Control Registers			
PMS_PRO_IRAM0_0_REG	IBUS permission control register 0.	0x0010	R/W
PMS_PRO_DRAM0_0_REG	DBUS permission control register 0.	0x0028	R/W
PMS_PRO_DPORT_0_REG	PeriBus1 permission control register 0.	0x003C	R/W
PMS_PRO_AHB_0_REG	PeriBus2 permission control register 0.	0x005C	R/W
PMS_PRO_TRACE_0_REG	Trace memory permission control register 0.	0x0070	R/W
PMS_PRO_CACHE_0_REG	Cache permission control register 0.	0x0078	R/W
PMS_DMA_APB_I_0_REG	Internal DMA permission control register 0.	0x008C	R/W
PMS_DMA_RX_I_0_REG	RX Copy DMA permission control register 0.	0x009C	R/W
PMS_DMA_TX_I_0_REG	TX Copy DMA permission control register 0.	0x00AC	R/W
PMS_CACHE_SOURCE_0_REG	Cache access permission control register 0.	0x00C4	R/W
PMS_APB_PERIPHERAL_0_REG	Peripheral access permission control register 0.	0x00CC	R/W
PMS_OCCUPY_0_REG	Occupy permission control register 0.	0x00D4	R/W
PMS_CACHE_TAG_ACCESS_0_REG	Cache tag permission control register 0.	0x00E4	R/W
PMS_CACHE_MMU_ACCESS_0_REG	Cache MMU permission control register 0.	0x00EC	R/W
PMS_CLOCK_GATE_REG_REG	Clock gate register of permission control.	0x0104	R/W
Configuration Registers			
PMS_PRO_IRAM0_1_REG	IBUS permission control register 1.	0x0014	R/W
PMS_PRO_IRAM0_2_REG	IBUS permission control register 2.	0x0018	R/W
PMS_PRO_IRAM0_3_REG	IBUS permission control register 3.	0x001C	R/W
PMS_PRO_DRAM0_1_REG	DBUS permission control register 1.	0x002C	R/W
PMS_PRO_DRAM0_2_REG	DBUS permission control register 2.	0x0030	R/W
PMS_PRO_DPORT_1_REG	PeriBus1 permission control register 1.	0x0040	R/W
PMS_PRO_DPORT_2_REG	PeriBus1 permission control register 2.	0x0044	R/W
PMS_PRO_DPORT_3_REG	PeriBus1 permission control register 3.	0x0048	R/W
PMS_PRO_DPORT_4_REG	PeriBus1 permission control register 4.	0x004C	R/W
PMS_PRO_DPORT_5_REG	PeriBus1 permission control register 5.	0x0050	R/W
PMS_PRO_AHB_1_REG	PeriBus2 permission control register 1.	0x0060	R/W
PMS_PRO_AHB_2_REG	PeriBus2 permission control register 2.	0x0064	R/W
PMS_PRO_TRACE_1_REG	Trace memory permission control register 1.	0x0074	R/W
PMS_PRO_CACHE_1_REG	Cache permission control register 1.	0x007C	R/W
PMS_DMA_APB_I_1_REG	Internal DMA permission control register 1.	0x0090	R/W
PMS_DMA_RX_I_1_REG	RX Copy DMA permission control register 1.	0x00A0	R/W
PMS_DMA_TX_I_1_REG	TX Copy DMA permission control register 1.	0x00B0	R/W
PMS_APB_PERIPHERAL_1_REG	Peripheral access permission control register 1.	0x00D0	R/W
PMS_OCCUPY_1_REG	Occupy permission control register 1.	0x00D8	R/W
PMS_OCCUPY_3_REG	Occupy permission control register 3.	0x00E0	R/W
PMS_CACHE_TAG_ACCESS_1_REG	Cache tag permission control register 1.	0x00E8	R/W
PMS_CACHE_MMU_ACCESS_1_REG	Cache MMU permission control register 1.	0x00F0	R/W
Interrupt Registers			
PMS_PRO_IRAM0_4_REG	IBUS permission control register 4.	0x0020	varies
PMS_PRO_IRAM0_5_REG	IBUS status register.	0x0024	RO
PMS_PRO_DRAM0_3_REG	DBUS permission control register 3.	0x0034	varies

Name	Description	Address	Access
PMS_PRO_DRAM0_4_REG	DBUS status register.	0x0038	RO
PMS_PRO_DPORT_6_REG	PeriBus1 permission control register 6.	0x0054	varies
PMS_PRO_DPORT_7_REG	PeriBus1 status register.	0x0058	RO
PMS_PRO_AHB_3_REG	PeriBus2 permission control register 3.	0x0068	varies
PMS_PRO_AHB_4_REG	PeriBus2 status register.	0x006C	RO
PMS_PRO_CACHE_2_REG	Cache permission control register 2.	0x0080	varies
PMS_PRO_CACHE_3_REG	Icache status register.	0x0084	RO
PMS_PRO_CACHE_4_REG	Dcache status register.	0x0088	RO
PMS_DMA_APB_I_2_REG	Internal DMA permission control register 2.	0x0094	varies
PMS_DMA_APB_I_3_REG	Internal DMA status register.	0x0098	RO
PMS_DMA_RX_I_2_REG	RX Copy DMA permission control register 2.	0x00A4	varies
PMS_DMA_RX_I_3_REG	RX Copy DMA status register.	0x00A8	RO
PMS_DMA_TX_I_2_REG	TX Copy DMA permission control register 2.	0x00B4	varies
PMS_DMA_TX_I_3_REG	TX Copy DMA status register.	0x00B8	RO
PMS_APB_PERIPHERAL_INTR_REG	PeriBus2 permission control register.	0x00F4	varies
PMS_APB_PERIPHERAL_STATUS_REG	PeriBus2 peripheral access status register.	0x00F8	RO
PMS_CPU_PERIPHERAL_INTR_REG	PeriBus1 permission control register.	0x00FC	varies
PMS_CPU_PERIPHERAL_STATUS_REG	PeriBus1 peripheral access status register.	0x0100	RO
Version Control Register			
PMS_DATE	Version control register.	0x0FFC	R/W

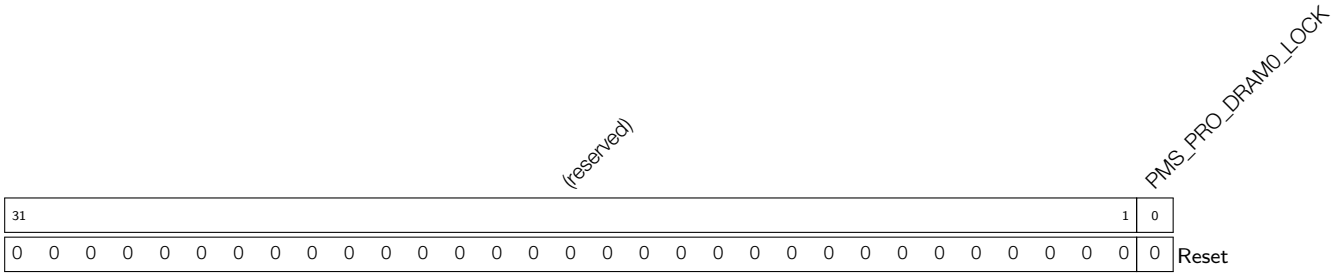
24.6 Registers

Register 24.1: PMS_PRO_IRAM0_0_REG (0x0010)



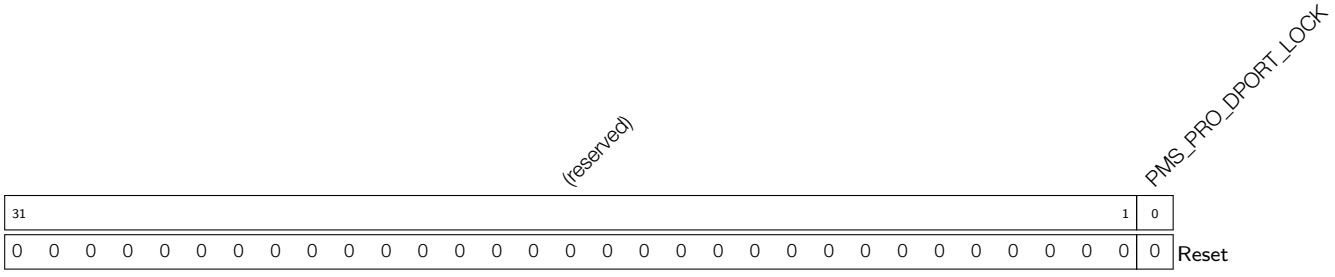
PMS_PRO_IRAM0_LOCK Lock register. Setting to 1 locks IBUS permission control registers. (R/W)

Register 24.2: PMS_PRO_DRAM0_0_REG (0x0028)



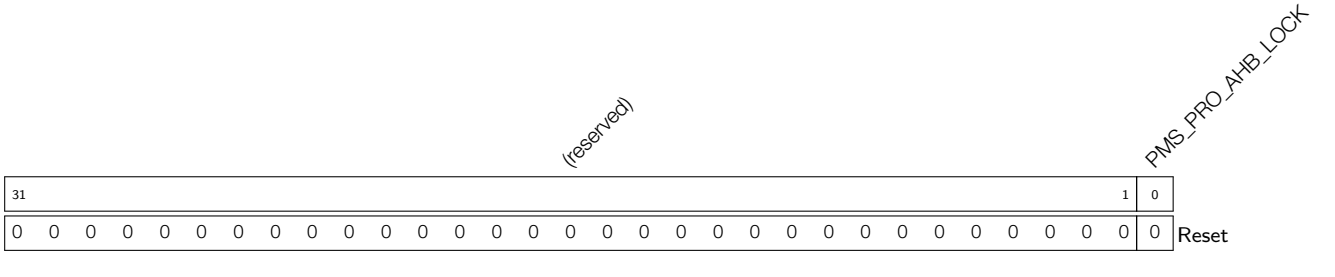
PMS_PRO_DRAM0_LOCK Lock register. Setting to 1 locks DBUS0 permission control registers. (R/W)

Register 24.3: PMS_PRO_DPORT_0_REG (0x003C)



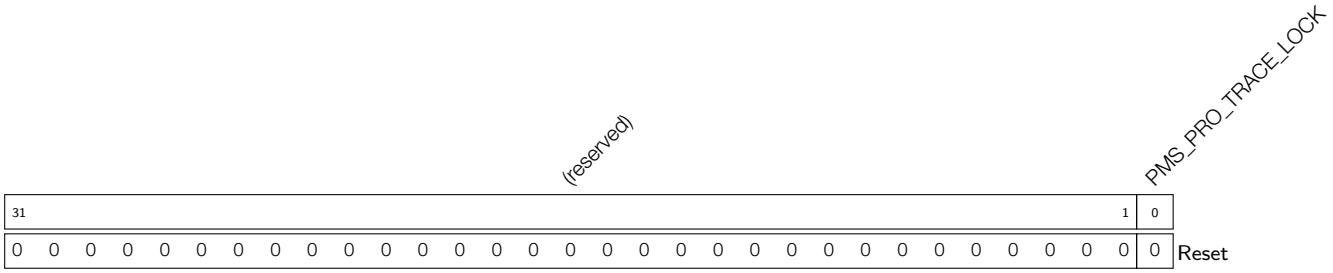
PMS_PRO_DPORT_LOCK Lock register. Setting to 1 locks PeriBus1 permission control registers. (R/W)

Register 24.4: PMS_PRO_AHB_0_REG (0x005C)



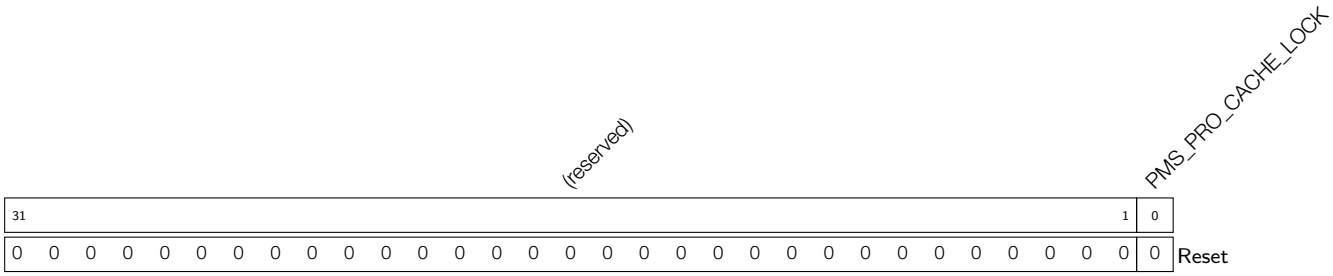
PMS_PRO_AHB_LOCK Lock register. Setting to 1 locks PeriBus2 permission control registers. (R/W)

Register 24.5: PMS_PRO_TRACE_0_REG (0x0070)



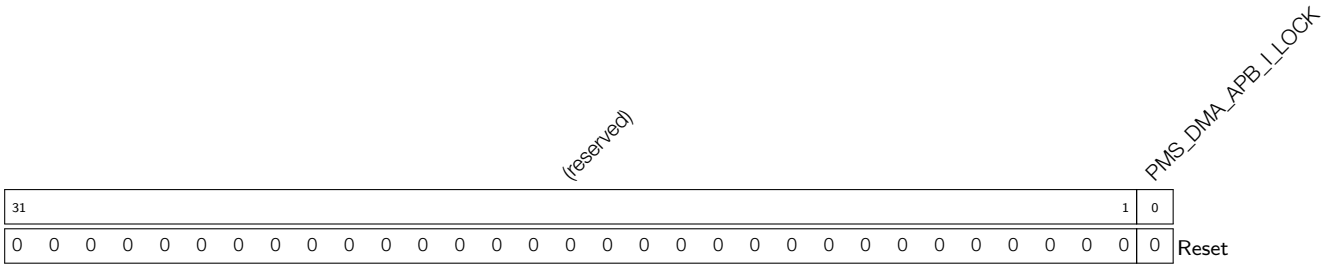
PMS_PRO_TRACE_LOCK Lock register. Setting to 1 locks trace function permission control registers. (R/W)

Register 24.6: PMS_PRO_CACHE_0_REG (0x0078)



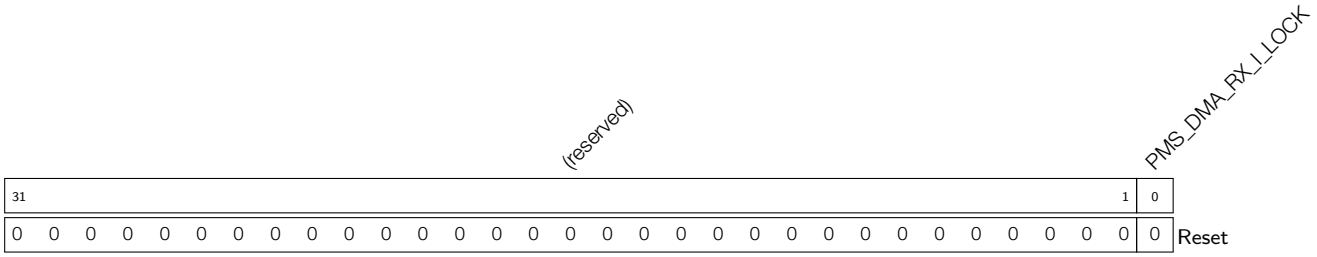
PMS_PRO_CACHE_LOCK Lock register. Setting to 1 locks cache permission control registers. (R/W)

Register 24.7: PMS_DMA_APB_I_0_REG (0x008C)



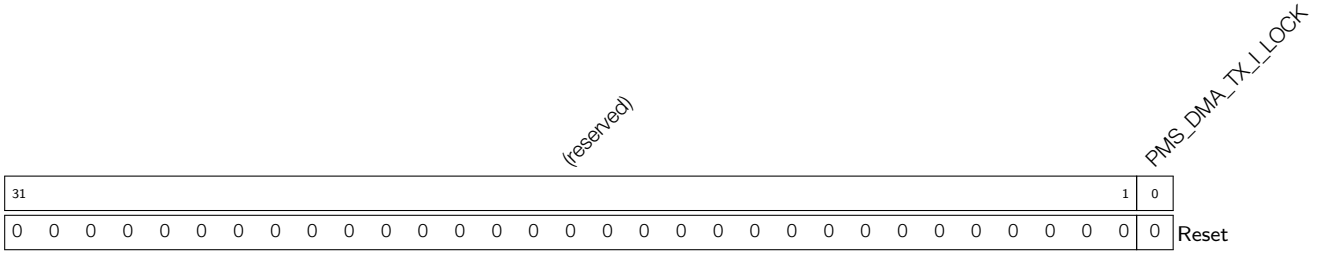
PMS_DMA_APB_I_LOCK Lock register. Setting to 1 locks internal DMA permission control registers. (R/W)

Register 24.8: PMS_DMA_RX_I_0_REG (0x009C)



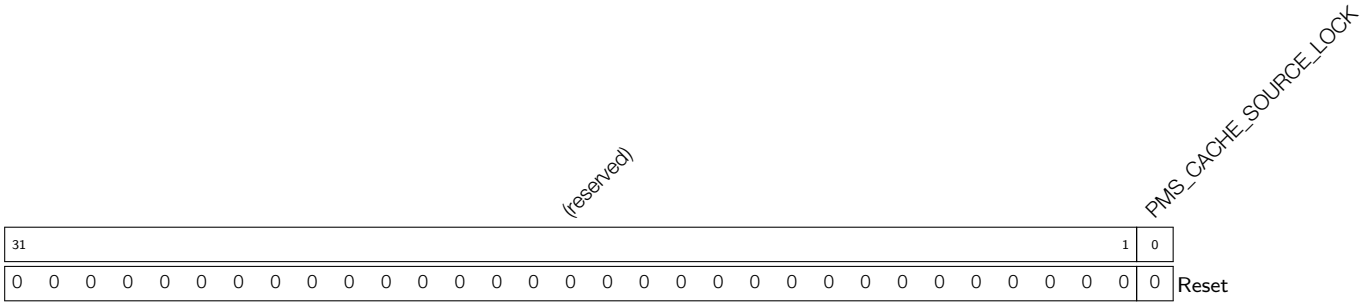
PMS_DMA_RX_I_LOCK Lock register. Setting to 1 locks RX Copy DMA permission control registers. (R/W)

Register 24.9: PMS_DMA_TX_I_0_REG (0x00AC)



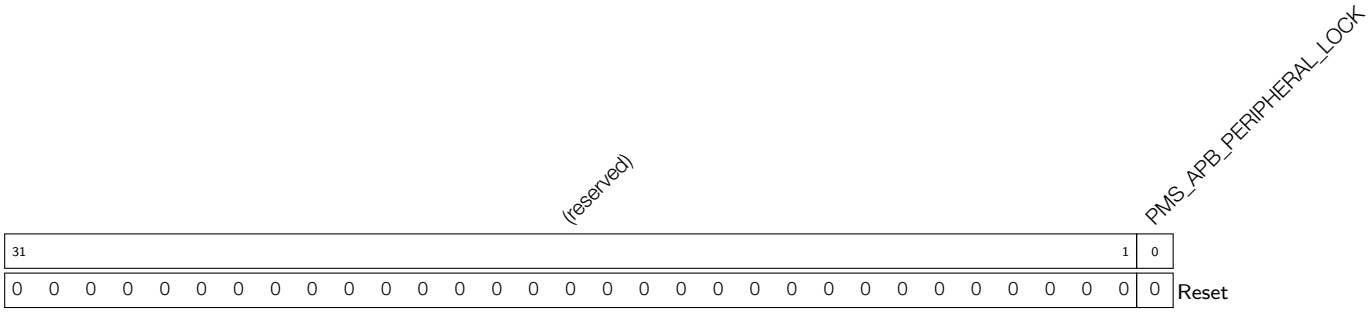
PMS_DMA_TX_I_LOCK Lock register. Setting to 1 locks TX Copy DMA permission control registers. (R/W)

Register 24.10: PMS_CACHE_SOURCE_0_REG (0x00C4)



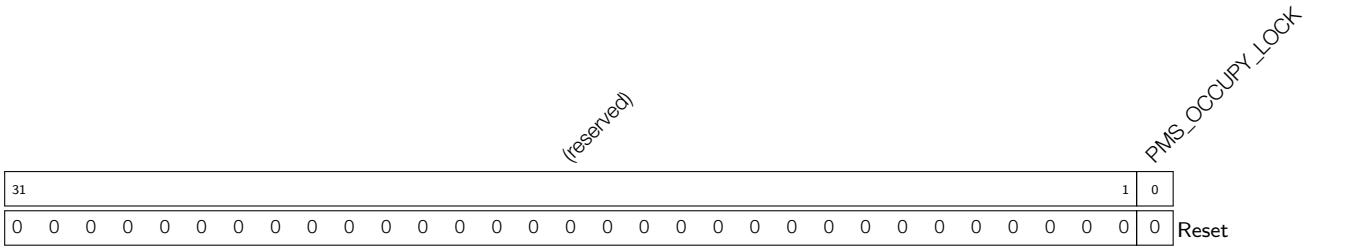
PMS_CACHE_SOURCE_LOCK Lock register. Setting to 1 locks cache access permission control registers. (R/W)

Register 24.11: PMS_APB_PERIPHERAL_0_REG (0x00CC)



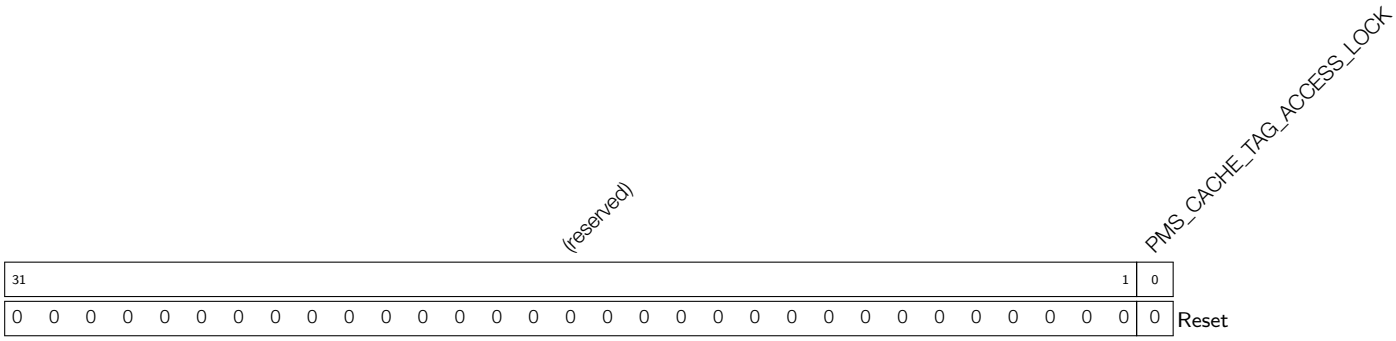
PMS_APB_PERIPHERAL_LOCK Lock register. Setting to 1 locks TX Copy DMA permission control registers. (R/W)

Register 24.12: PMS_OCCUPY_0_REG (0x00D4)



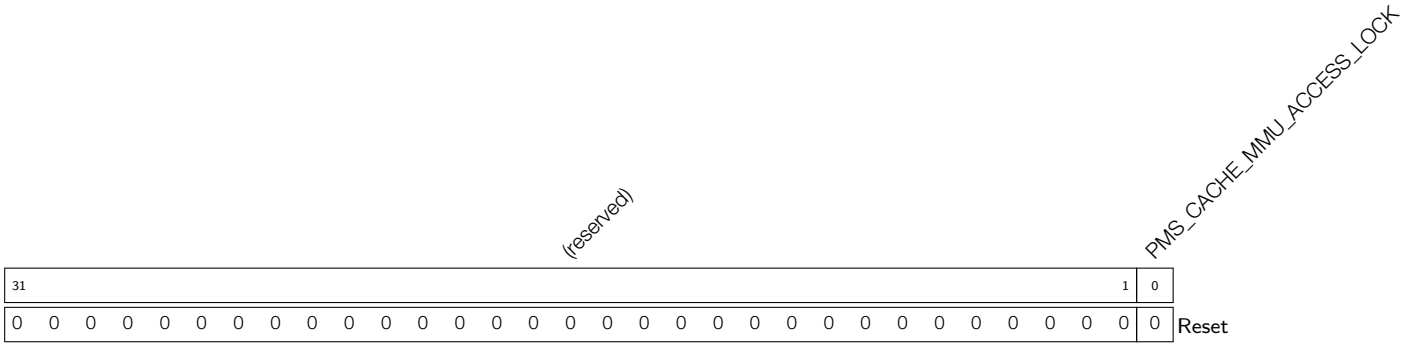
PMS_OCCUPY_LOCK Lock register. Setting to 1 locks occupy permission control registers. (R/W)

Register 24.13: PMS_CACHE_TAG_ACCESS_0_REG (0x00E4)



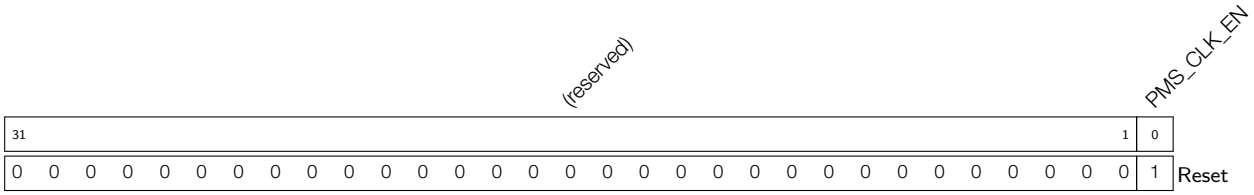
PMS_CACHE_TAG_ACCESS_LOCK Lock register. Setting to 1 locks cache tag permission control registers. (R/W)

Register 24.14: PMS_CACHE_MMU_ACCESS_0_REG (0x00EC)



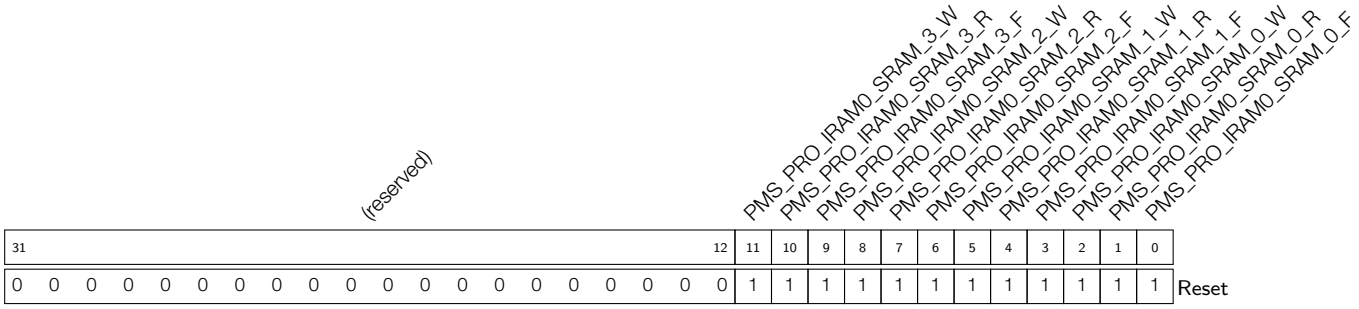
PMS_CACHE_MMU_ACCESS_LOCK Lock register. Setting to 1 locks cache MMU permission control registers. (R/W)

Register 24.15: PMS_CLOCK_GATE_REG_REG (0x0104)



PMS_CLK_EN Enable the clock of permission control module when set to 1. (R/W)

Register 24.16: PMS_PRO_IRAM0_1_REG (0x0014)



PMS_PRO_IRAM0_SRAM_0_F Setting to 1 grants IBUS permission to fetch SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_0_R Setting to 1 grants IBUS permission to read SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_0_W Setting to 1 grants IBUS permission to write SRAM Block 0. (R/W)

PMS_PRO_IRAM0_SRAM_1_F Setting to 1 grants IBUS permission to fetch SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_1_R Setting to 1 grants IBUS permission to read SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_1_W Setting to 1 grants IBUS permission to write SRAM Block 1. (R/W)

PMS_PRO_IRAM0_SRAM_2_F Setting to 1 grants IBUS permission to fetch SRAM Block 2. (R/W)

PMS_PRO_IRAM0_SRAM_2_R Setting to 1 grants IBUS permission to read SRAM Block 2. (R/W)

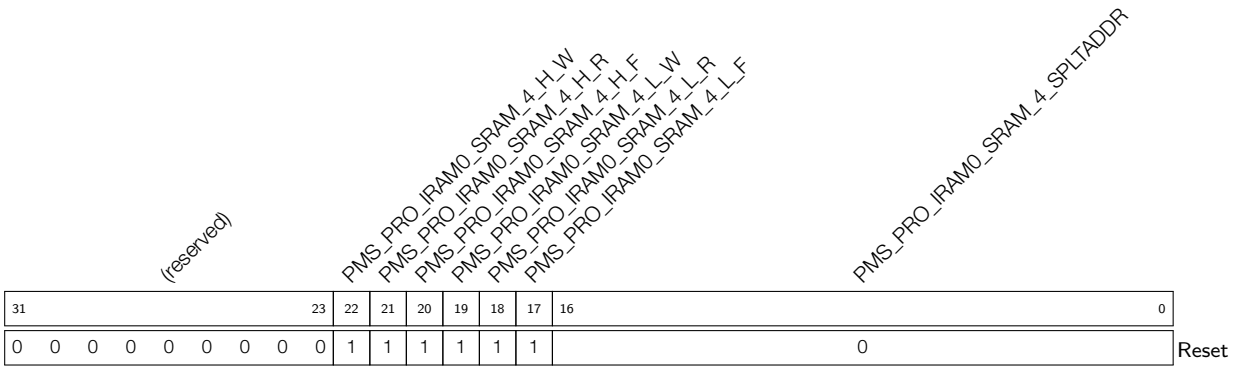
PMS_PRO_IRAM0_SRAM_2_W Setting to 1 grants IBUS permission to write SRAM Block 2. (R/W)

PMS_PRO_IRAM0_SRAM_3_F Setting to 1 grants IBUS permission to fetch SRAM Block 3. (R/W)

PMS_PRO_IRAM0_SRAM_3_R Setting to 1 grants IBUS permission to read SRAM Block 3. (R/W)

PMS_PRO_IRAM0_SRAM_3_W Setting to 1 grants IBUS permission to write SRAM Block 3. (R/W)

Register 24.17: PMS_PRO_IRAM0_2_REG (0x0018)



PMS_PRO_IRAM0_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for IBUS access. (R/W)

PMS_PRO_IRAM0_SRAM_4_L_F Setting to 1 grants IBUS permission to fetch SRAM Block 4-21 low address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_L_R Setting to 1 grants IBUS permission to read SRAM Block 4-21 low address region. (R/W)

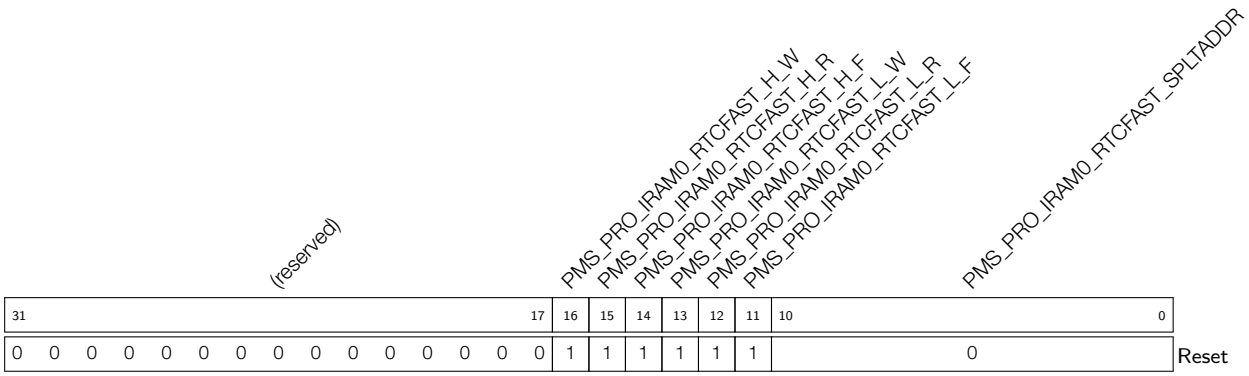
PMS_PRO_IRAM0_SRAM_4_L_W Setting to 1 grants IBUS permission to write SRAM Block 4-21 low address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_F Setting to 1 grants IBUS permission to fetch SRAM Block 4-21 high address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_R Setting to 1 grants IBUS permission to read SRAM Block 4-21 high address region. (R/W)

PMS_PRO_IRAM0_SRAM_4_H_W Setting to 1 grants IBUS permission to write SRAM Block 4-21 high address region. (R/W)

Register 24.18: PMS_PRO_IRAM0_3_REG (0x001C)



PMS_PRO_IRAM0_RTCFAST_SPLTADDR Configure the split address of RTC FAST for IBUS access. (R/W)

PMS_PRO_IRAM0_RTCFAST_L_F Setting to 1 grants IBUS permission to fetch RTC FAST low address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_L_R Setting to 1 grants IBUS permission to read RTC FAST low address region. (R/W)

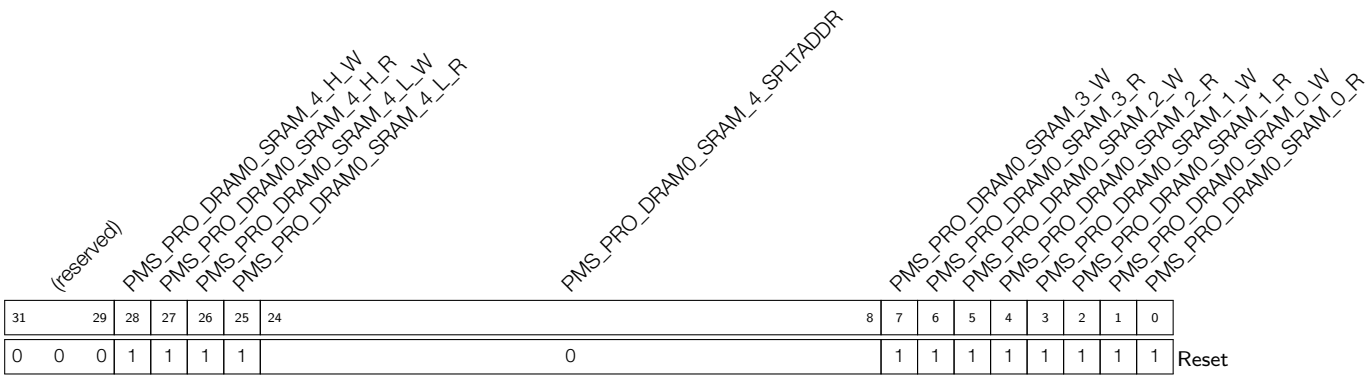
PMS_PRO_IRAM0_RTCFAST_L_W Setting to 1 grants IBUS permission to write RTC FAST low address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_F Setting to 1 grants IBUS permission to fetch RTC FAST high address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_R Setting to 1 grants IBUS permission to read RTC FAST high address region. (R/W)

PMS_PRO_IRAM0_RTCFAST_H_W Setting to 1 grants IBUS permission to write RTC FAST high address region. (R/W)

Register 24.19: PMS_PRO_DRAM0_1_REG (0x002C)



PMS_PRO_DRAM0_SRAM_0_R Setting to 1 grants DBUS0 permission to read SRAM Block 0. (R/W)

PMS_PRO_DRAM0_SRAM_0_W Setting to 1 grants DBUS0 permission to write SRAM Block 0. (R/W)

PMS_PRO_DRAM0_SRAM_1_R Setting to 1 grants DBUS0 permission to read SRAM Block 1. (R/W)

PMS_PRO_DRAM0_SRAM_1_W Setting to 1 grants DBUS0 permission to write SRAM Block 1. (R/W)

PMS_PRO_DRAM0_SRAM_2_R Setting to 1 grants DBUS0 permission to read SRAM Block 2. (R/W)

PMS_PRO_DRAM0_SRAM_2_W Setting to 1 grants DBUS0 permission to write SRAM Block 2. (R/W)

PMS_PRO_DRAM0_SRAM_3_R Setting to 1 grants DBUS0 permission to read SRAM Block 3. (R/W)

PMS_PRO_DRAM0_SRAM_3_W Setting to 1 grants DBUS0 permission to write SRAM Block 3. (R/W)

PMS_PRO_DRAM0_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for DBUS0 access. (R/W)

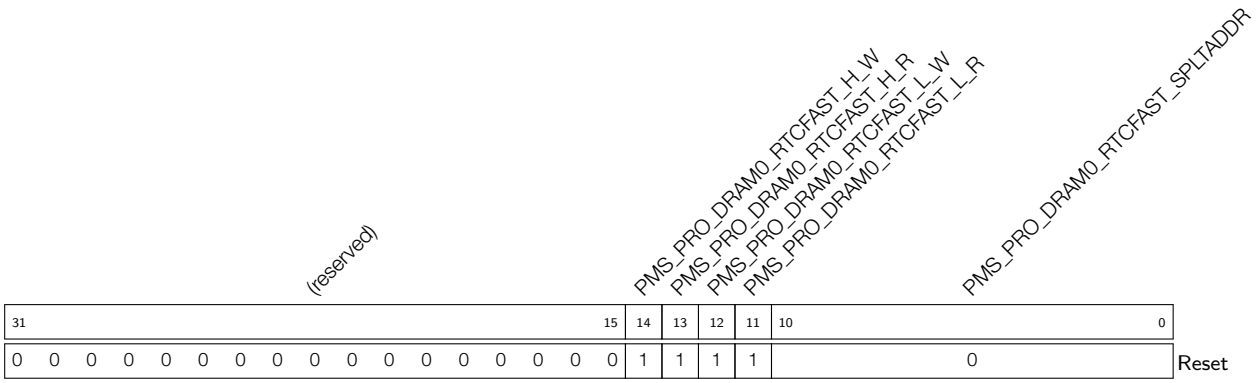
PMS_PRO_DRAM0_SRAM_4_L_R Setting to 1 grants DBUS0 permission to read SRAM Block 4-21 low address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_L_W Setting to 1 grants DBUS0 permission to write SRAM Block 4-21 low address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_H_R Setting to 1 grants DBUS0 permission to read SRAM Block 4-21 high address region. (R/W)

PMS_PRO_DRAM0_SRAM_4_H_W Setting to 1 grants DBUS0 permission to write SRAM Block 4-21 high address region. (R/W)

Register 24.20: PMS_PRO_DRAM0_2_REG (0x0030)



PMS_PRO_DRAM0_RTCFAST_SPLTADDR Configure the split address of RTC FAST for DBUS0 access. (R/W)

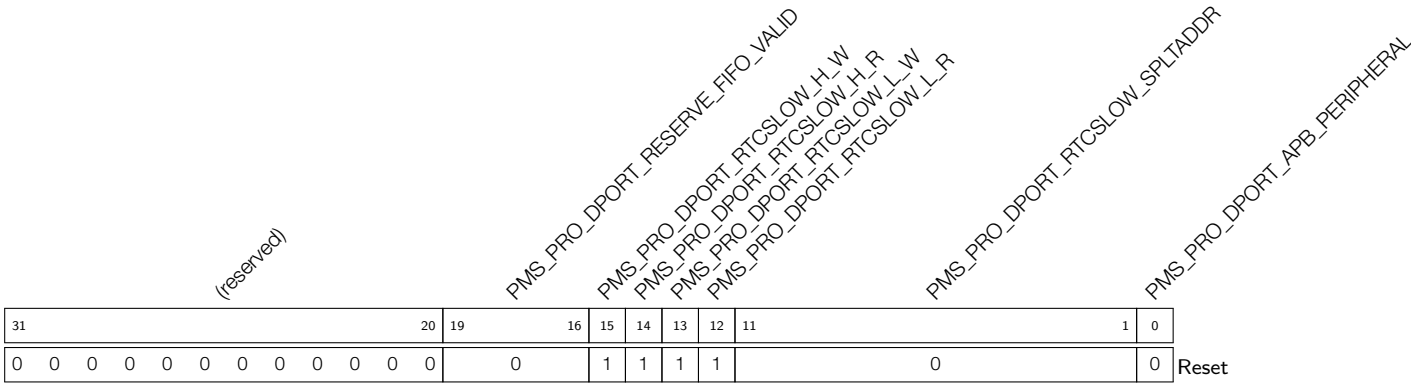
PMS_PRO_DRAM0_RTCFAST_L_R Setting to 1 grants DBUS0 permission to read RTC FAST low address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_L_W Setting to 1 grants DBUS0 permission to write RTC FAST low address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_H_R Setting to 1 grants DBUS0 permission to read RTC FAST high address region. (R/W)

PMS_PRO_DRAM0_RTCFAST_H_W Setting to 1 grants DBUS0 permission to write RTC FAST high address region. (R/W)

Register 24.21: PMS_PRO_DPORT_1_REG (0x0040)



PMS_PRO_DPORT_APB_PERIPHERAL_FORBID Setting to 1 denies PeriBus1 bus's access to APB peripheral. (R/W)

PMS_PRO_DPORT_RTCSLOW_SPLTADDR Configure the split address of RTC FAST for PeriBus1 access. (R/W)

PMS_PRO_DPORT_RTCSLOW_L_R Setting to 1 grants PeriBus1 permission to read RTC FAST low address region. (R/W)

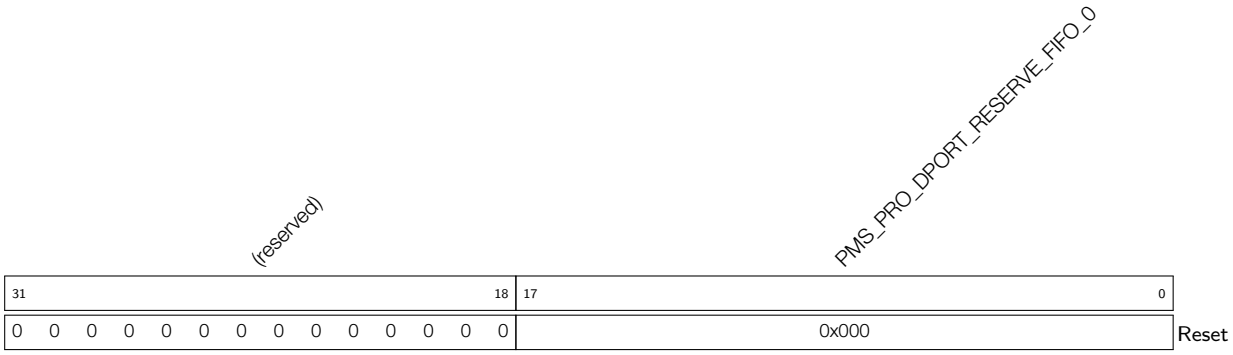
PMS_PRO_DPORT_RTCSLOW_L_W Setting to 1 grants PeriBus1 permission to write RTC FAST low address region. (R/W)

PMS_PRO_DPORT_RTCSLOW_H_R Setting to 1 grants PeriBus1 permission to read RTC FAST high address region. (R/W)

PMS_PRO_DPORT_RTCSLOW_H_W Setting to 1 grants PeriBus1 permission to write RTC FAST high address region. (R/W)

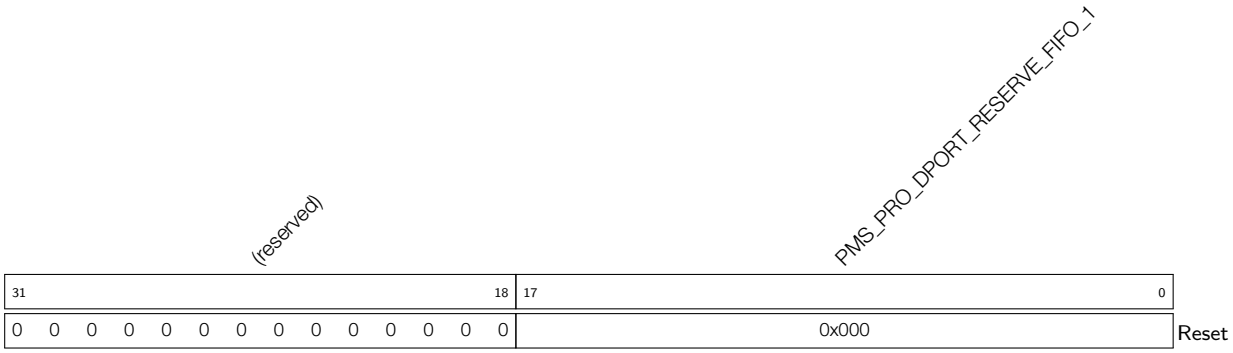
PMS_PRO_DPORT_RESERVE_FIFO_VALID Configure whether to enable read protection for user-configured FIFO address. (R/W)

Register 24.22: PMS_PRO_DPORT_2_REG (0x0044)



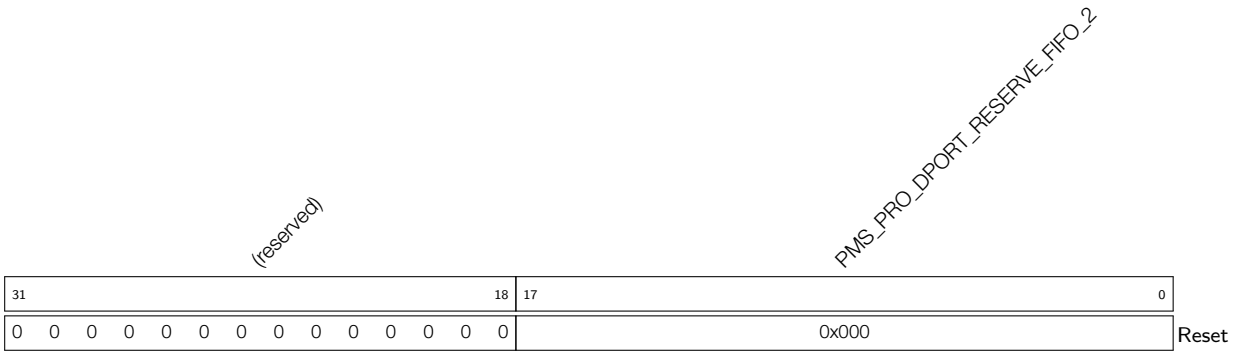
PMS_PRO_DPORT_RESERVE_FIFO_0 Configure read-protection address 0. (R/W)

Register 24.23: PMS_PRO_DPORT_3_REG (0x0048)



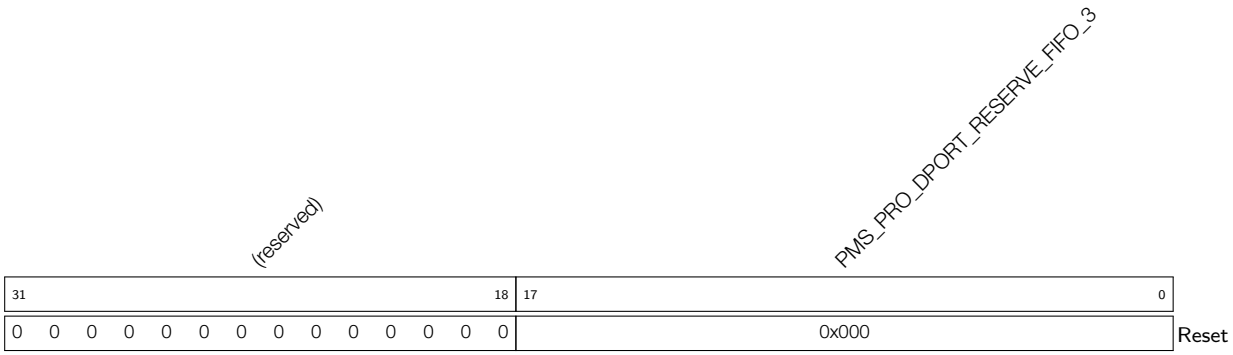
PMS_PRO_DPORT_RESERVE_FIFO_1 Configure read-protection address 1. (R/W)

Register 24.24: PMS_PRO_DPORT_4_REG (0x004C)



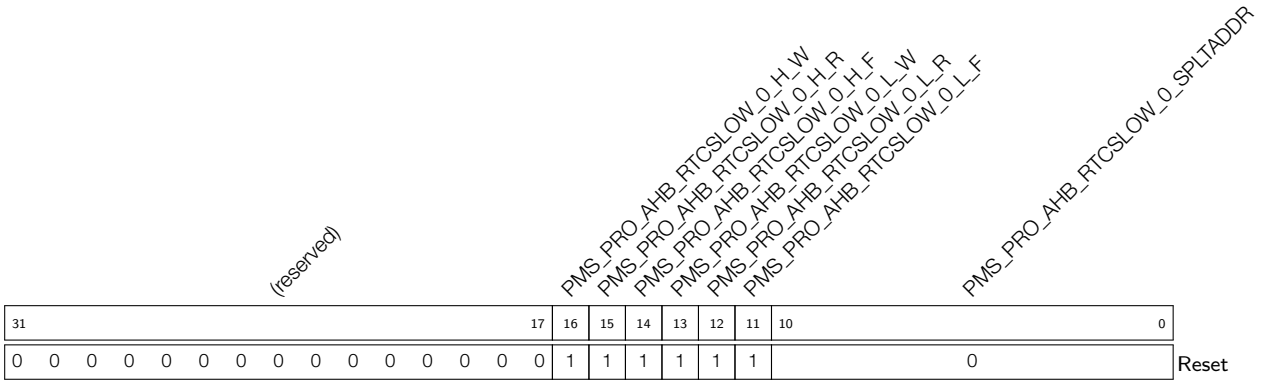
PMS_PRO_DPORT_RESERVE_FIFO_2 Configure read-protection address 2. (R/W)

Register 24.25: PMS_PRO_DPORT_5_REG (0x0050)



PMS_PRO_DPORT_RESERVE_FIFO_3 Configure read-protection address 3. (R/W)

Register 24.26: PMS_PRO_AHB_1_REG (0x0060)



PMS_PRO_AHB_RTCSLOW_0_SPLTADDR Configure the split address of RTCSlow_0 for PeriBus2 access. (R/W)

PMS_PRO_AHB_RTCSLOW_0_L_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_0 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_L_R Setting to 1 grants PeriBus2 permission to read RTCSlow_0 low address region. (R/W)

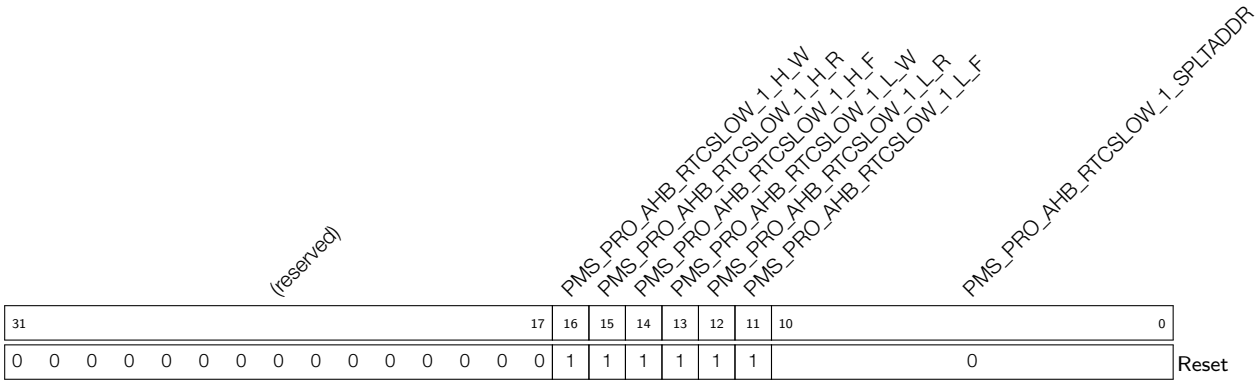
PMS_PRO_AHB_RTCSLOW_0_L_W Setting to 1 grants PeriBus2 permission to write RTCSlow_0 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_0 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_R Setting to 1 grants PeriBus2 permission to read RTCSlow_0 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_0_H_W Setting to 1 grants PeriBus2 permission to write RTCSlow_0 high address region. (R/W)

Register 24.27: PMS_PRO_AHB_2_REG (0x0064)



PMS_PRO_AHB_RTCSLOW_1_SPLTADDR Configure the split address of RTCSlow_1 for PeriBus2 access. (R/W)

PMS_PRO_AHB_RTCSLOW_1_L_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_1 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_L_R Setting to 1 grants PeriBus2 permission to read RTCSlow_1 low address region. (R/W)

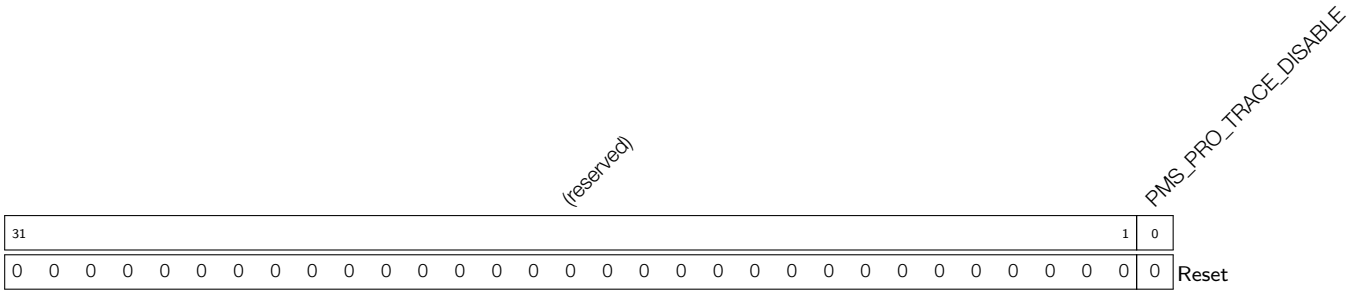
PMS_PRO_AHB_RTCSLOW_1_L_W Setting to 1 grants PeriBus2 permission to write RTCSlow_1 low address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_H_F Setting to 1 grants PeriBus2 permission to fetch RTCSlow_1 high address region. (R/W)

PMS_PRO_AHB_RTCSLOW_1_H_R Setting to 1 grants PeriBus2 permission to read RTCSlow_1 high address region. (R/W)

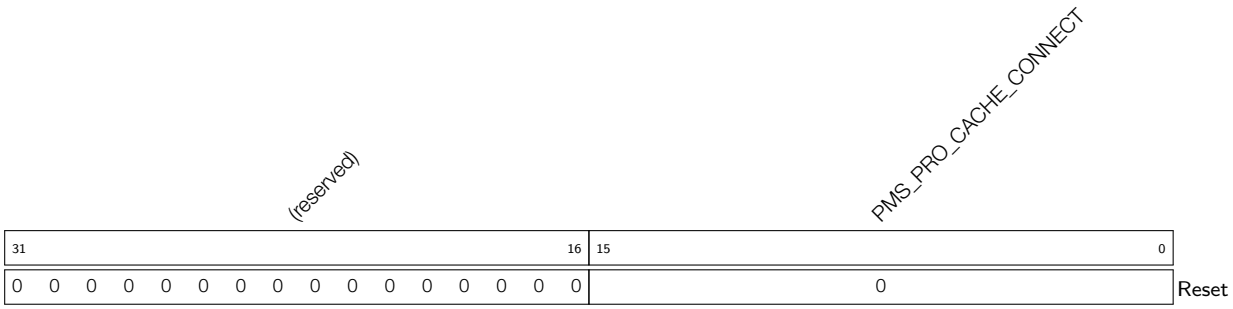
PMS_PRO_AHB_RTCSLOW_1_H_W Setting to 1 grants PeriBus2 permission to write RTCSlow_1 high address region. (R/W)

Register 24.28: PMS_PRO_TRACE_1_REG (0x0074)



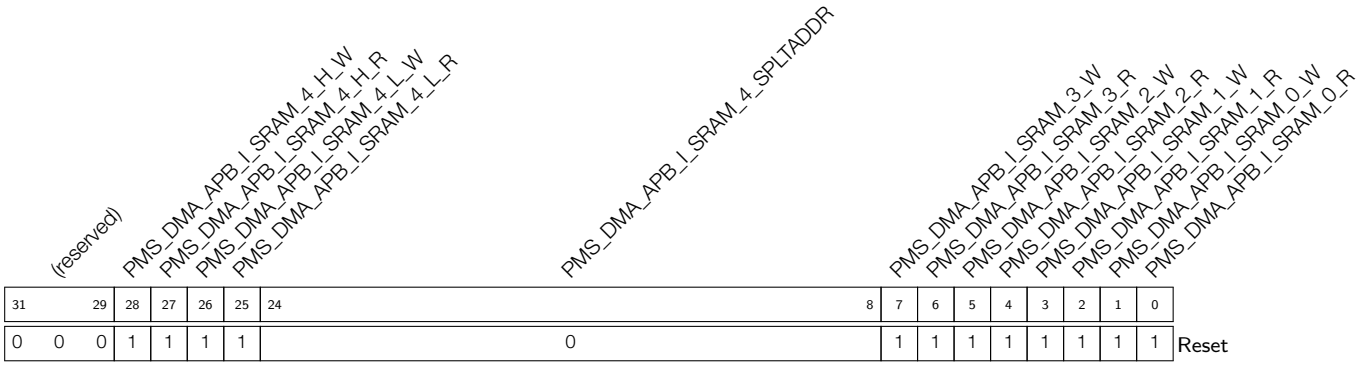
PMS_PRO_TRACE_DISABLE Setting to 1 disables the trace memory function. (R/W)

Register 24.29: PMS_PRO_CACHE_1_REG (0x007C)



PMS_PRO_CACHE_CONNECT Configure which SRAM Block will be occupied by Icache or Dcache. (R/W)

Register 24.30: PMS_DMA_APB_I_1_REG (0x0090)



PMS_DMA_APB_I_SRAM_0_R Setting to 1 grants internal DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_APB_I_SRAM_0_W Setting to 1 grants internal DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_APB_I_SRAM_1_R Setting to 1 grants internal DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_APB_I_SRAM_1_W Setting to 1 grants internal DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_APB_I_SRAM_2_R Setting to 1 grants internal DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_APB_I_SRAM_2_W Setting to 1 grants internal DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_APB_I_SRAM_3_R Setting to 1 grants internal DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_APB_I_SRAM_3_W Setting to 1 grants internal DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_APB_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for internal DMA access. (R/W)

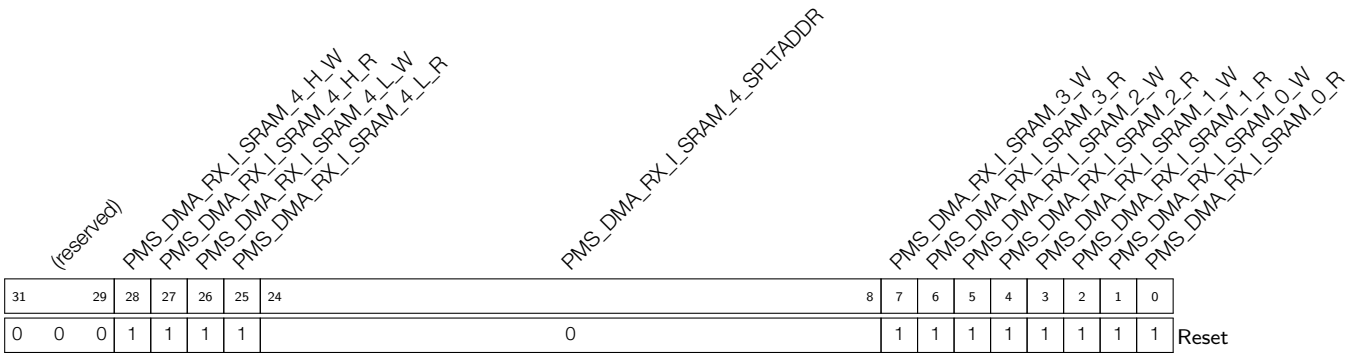
PMS_DMA_APB_I_SRAM_4_L_R Setting to 1 grants internal DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_APB_I_SRAM_4_L_W Setting to 1 grants internal DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_APB_I_SRAM_4_H_R Setting to 1 grants internal DMA permission to read SRAM Block 4-21 high address region. (R/W)

PMS_DMA_APB_I_SRAM_4_H_W Setting to 1 grants internal DMA permission to write SRAM Block 4-21 high address region. (R/W)

Register 24.31: PMS_DMA_RX_I_1_REG (0x00A0)



PMS_DMA_RX_I_SRAM_0_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_RX_I_SRAM_0_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_RX_I_SRAM_1_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_RX_I_SRAM_1_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_RX_I_SRAM_2_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_RX_I_SRAM_2_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_RX_I_SRAM_3_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_RX_I_SRAM_3_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_RX_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for RX Copy DMA access. (R/W)

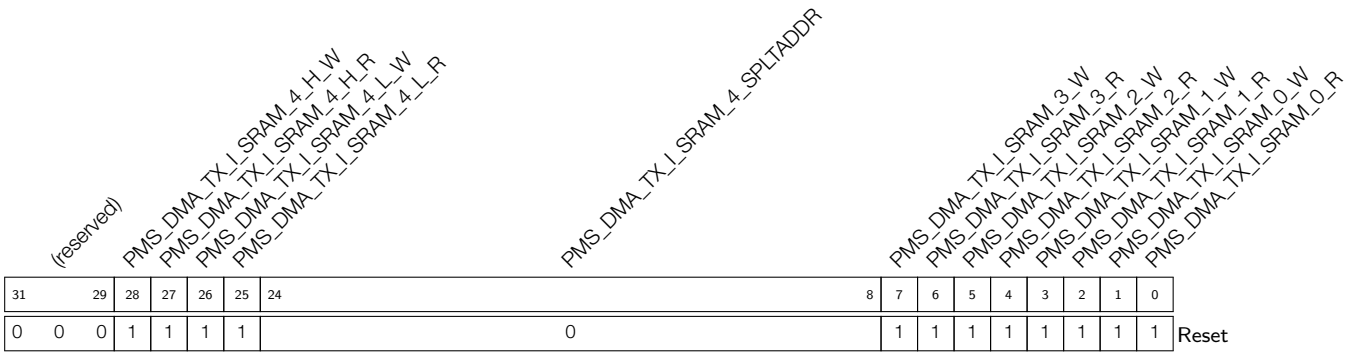
PMS_DMA_RX_I_SRAM_4_L_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_RX_I_SRAM_4_L_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_RX_I_SRAM_4_H_R Setting to 1 grants RX Copy DMA permission to read SRAM Block 4-21 high address region. (R/W)

PMS_DMA_RX_I_SRAM_4_H_W Setting to 1 grants RX Copy DMA permission to write SRAM Block 4 21 high address region. (R/W)

Register 24.32: PMS_DMA_TX_I_1_REG (0x00B0)



PMS_DMA_TX_I_SRAM_0_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 0. (R/W)

PMS_DMA_TX_I_SRAM_0_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 0. (R/W)

PMS_DMA_TX_I_SRAM_1_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 1. (R/W)

PMS_DMA_TX_I_SRAM_1_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 1. (R/W)

PMS_DMA_TX_I_SRAM_2_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 2. (R/W)

PMS_DMA_TX_I_SRAM_2_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 2. (R/W)

PMS_DMA_TX_I_SRAM_3_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 3. (R/W)

PMS_DMA_TX_I_SRAM_3_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 3. (R/W)

PMS_DMA_TX_I_SRAM_4_SPLTADDR Configure the split address of SRAM Block 4-21 for TX Copy DMA access. (R/W)

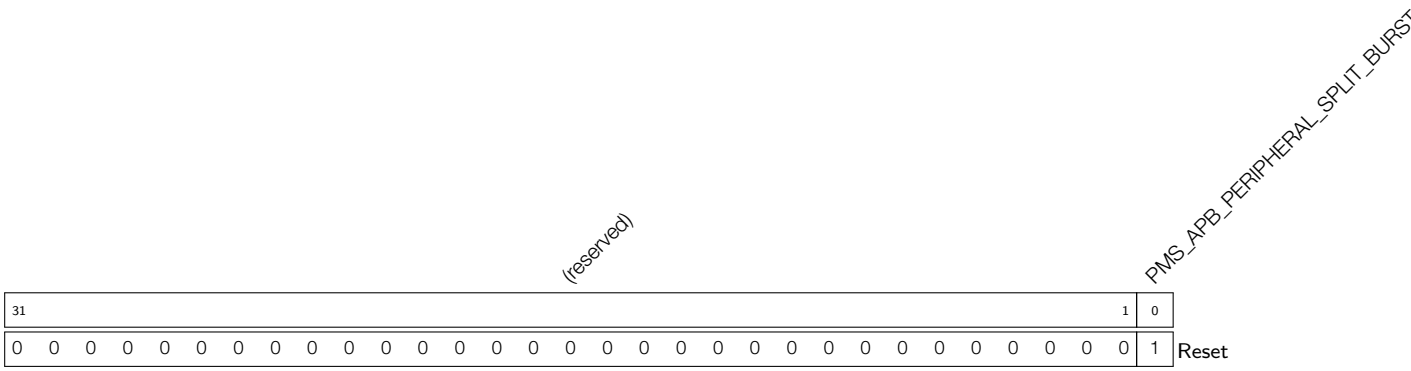
PMS_DMA_TX_I_SRAM_4_L_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 4-21 low address region. (R/W)

PMS_DMA_TX_I_SRAM_4_L_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 4-21 low address region. (R/W)

PMS_DMA_TX_I_SRAM_4_H_R Setting to 1 grants TX Copy DMA permission to read SRAM Block 4-21 high address region. (R/W)

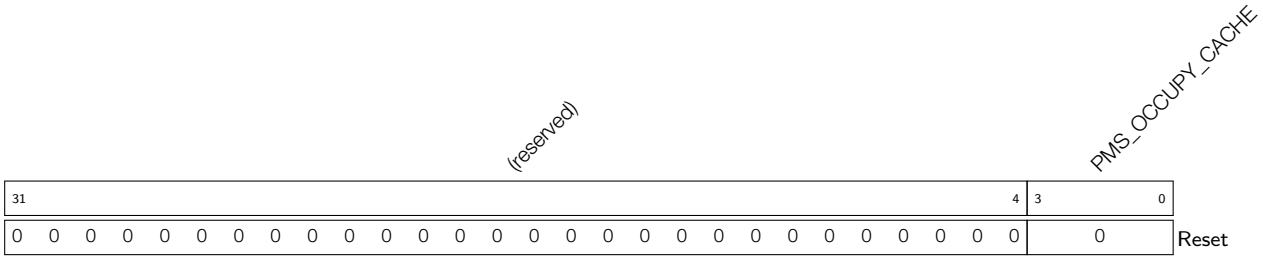
PMS_DMA_TX_I_SRAM_4_H_W Setting to 1 grants TX Copy DMA permission to write SRAM Block 4-21 high address region. (R/W)

Register 24.33: PMS_APB_PERIPHERAL_1_REG (0x00D0)



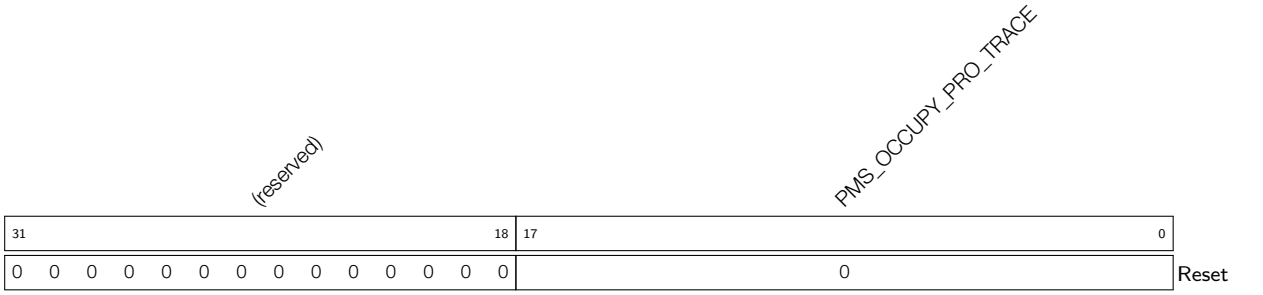
PMS_APB_PERIPHERAL_SPLIT_BURST Setting to 1 splits the data phase of the last access and the address phase of following access. (R/W)

Register 24.34: PMS_OCCUPY_1_REG (0x00D8)



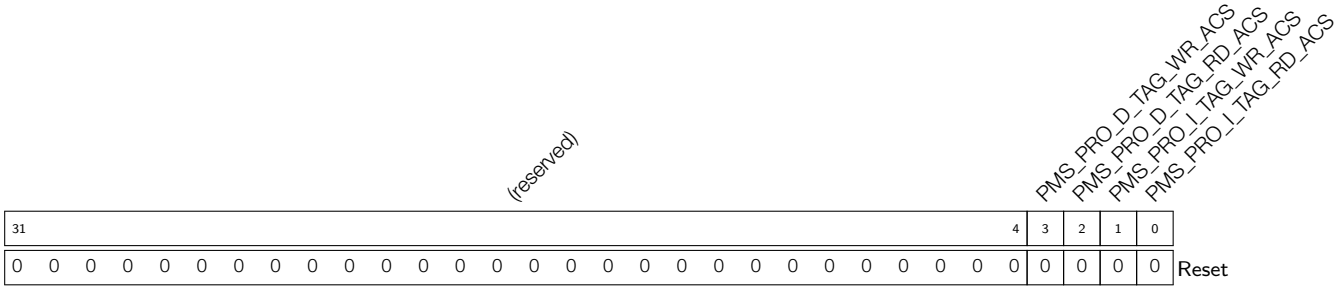
PMS_OCCUPY_CACHE Configure whether SRAM Block 0-3 is used as cache memory. (R/W)

Register 24.35: PMS_OCCUPY_3_REG (0x00E0)



PMS_OCCUPY_PRO_TRACE Configure one out block of Block 4-21 is used as trace memory. (R/W)

Register 24.36: PMS_CACHE_TAG_ACCESS_1_REG (0x00E8)



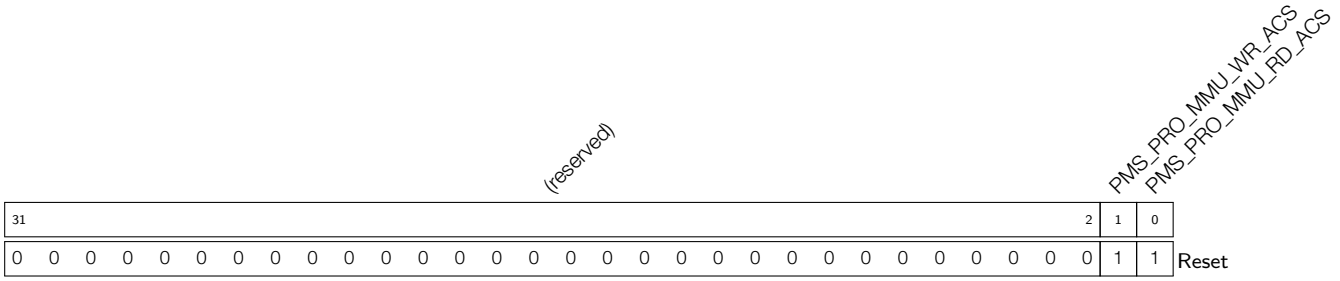
PMS_PRO_I_TAG_RD_ACS Setting to 1 permits read access to Icache tag memory. (R/W)

PMS_PRO_I_TAG_WR_ACS Setting to 1 permits write access to Icache tag memory. (R/W)

PMS_PRO_D_TAG_RD_ACS Setting to 1 permits read access to Dcache tag memory. (R/W)

PMS_PRO_D_TAG_WR_ACS Setting to 1 permits write access to Dcache tag memory. (R/W)

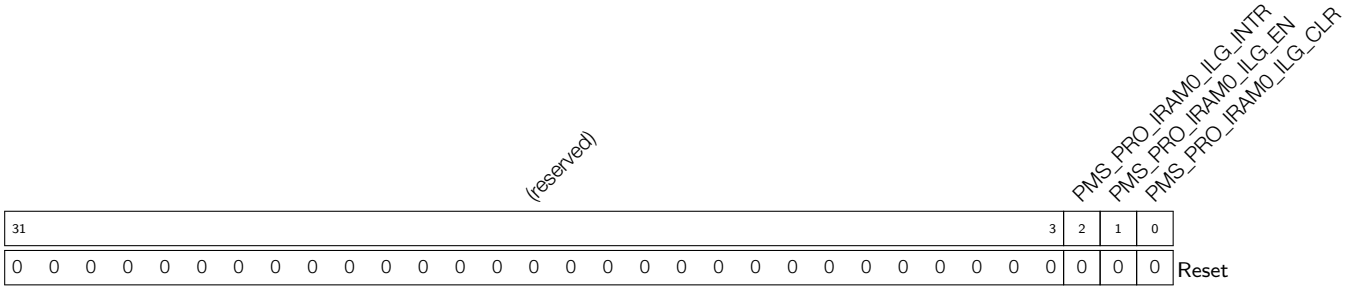
Register 24.37: PMS_CACHE_MMU_ACCESS_1_REG (0x00F0)



PMS_PRO_MMU_RD_ACS Setting to 1 permits read access to MMU memory. (R/W)

PMS_PRO_MMU_WR_ACS Setting to 1 permits write access to MMU memory. (R/W)

Register 24.38: PMS_PRO_IRAM0_4_REG (0x0020)

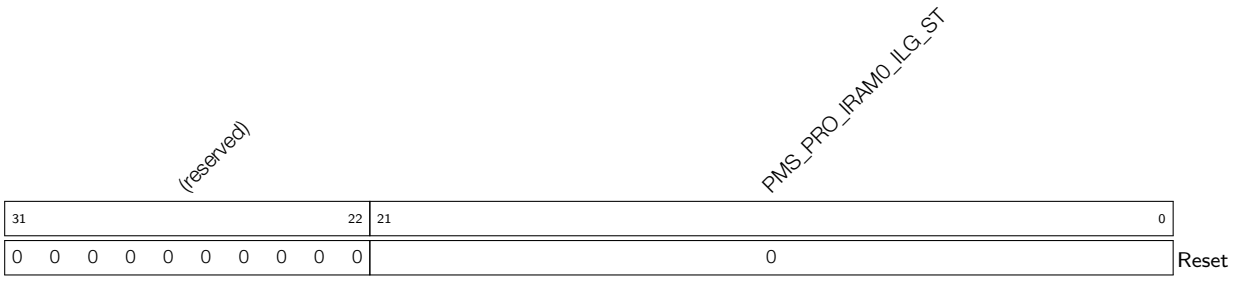


PMS_PRO_IRAM0_ILG_CLR The clear signal for IBUS access interrupt. (R/W)

PMS_PRO_IRAM0_ILG_EN The enable signal for IBUS access interrupt. (R/W)

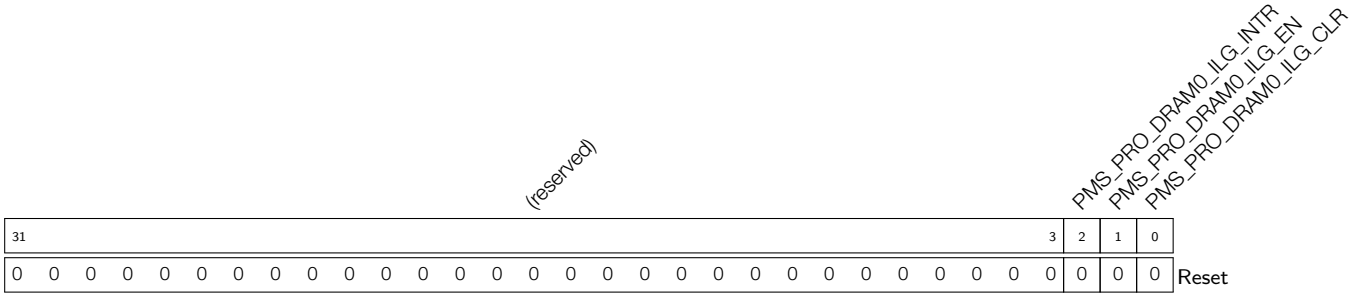
PMS_PRO_IRAM0_ILG_INTR IBUS access interrupt signal. (RO)

Register 24.39: PMS_PRO_IRAM0_5_REG (0x0024)



PMS_PRO_IRAM0_ILG_ST Record the illegitimate information of IBUS. [21:2]: store the bits [21:2] of IBUS address; [1]: 1 means data access, 0 means instruction access; [0]: 1 means write operation, 0 means read operation. (RO)

Register 24.40: PMS_PRO_DRAM0_3_REG (0x0034)

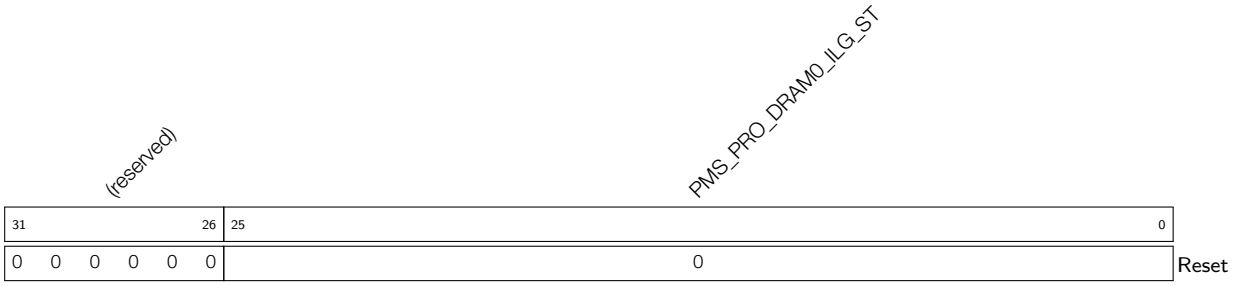


PMS_PRO_DRAM0_ILG_CLR The clear signal for DBUS0 access interrupt. (R/W)

PMS_PRO_DRAM0_ILG_EN The enable signal for DBUS0 access interrupt. (R/W)

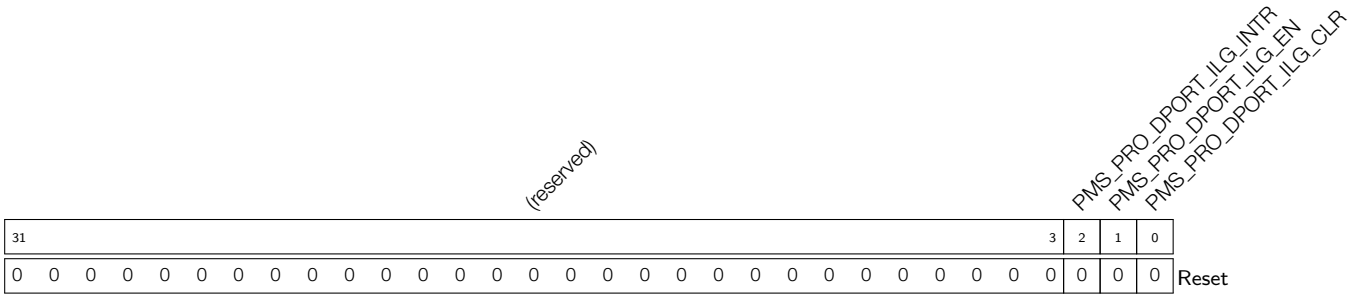
PMS_PRO_DRAM0_ILG_INTR DBUS0 access interrupt signal. (RO)

Register 24.41: PMS_PRO_DRAM0_4_REG (0x0038)



PMS_PRO_DRAM0_ILG_ST Record the illegitimate information of DBUS. [25:6]: store the bits [21:2] of DBUS address; [5]: 1 means atomic access, 0 means nonatomic access; [4]: 1 means write operation, 0 means read operation; [3:0]: DBUS0 bus byte enables. (RO)

Register 24.42: PMS_PRO_DPORT_6_REG (0x0054)

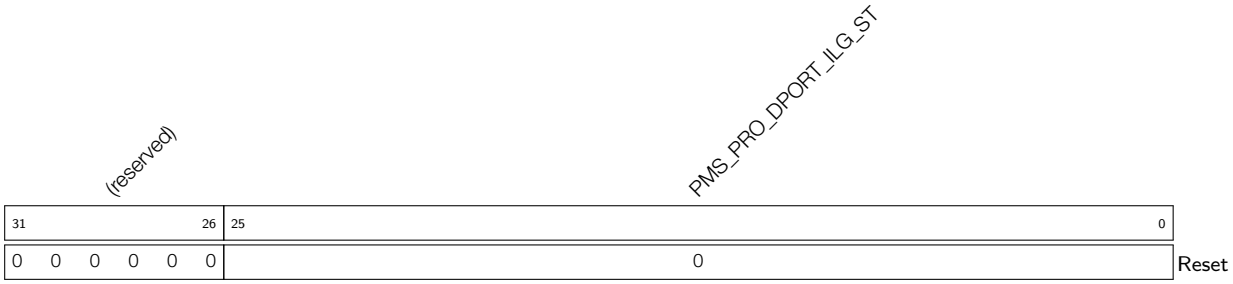


PMS_PRO_DPORT_ILG_CLR The clear signal for PeriBus1 access interrupt. (R/W)

PMS_PRO_DPORT_ILG_EN The enable signal for PeriBus1 access interrupt. (R/W)

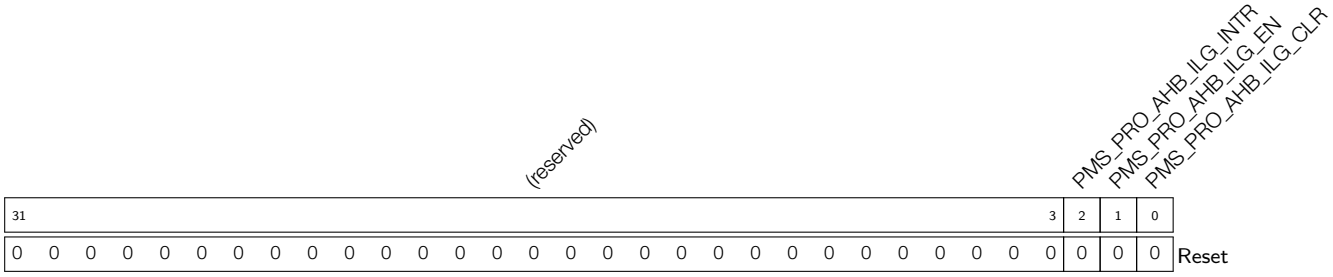
PMS_PRO_DPORT_ILG_INTR PeriBus1 access interrupt signal. (RO)

Register 24.43: PMS_PRO_DPORT_7_REG (0x0058)



PMS_PRO_DPORT_ILG_ST Record the illegitimate information of PeriBus1. [25:6]: store the bits [21:2] of PeriBus1 address; [5]: 1 means atomic access, 0 means nonatomic access; [4]: if bits [31:22] of PeriBus1 address are 0xfd, then the bit value is 1, otherwise it is 0; [3:0]: PeriBus 1 byte enables. (RO)

Register 24.44: PMS_PRO_AHB_3_REG (0x0068)

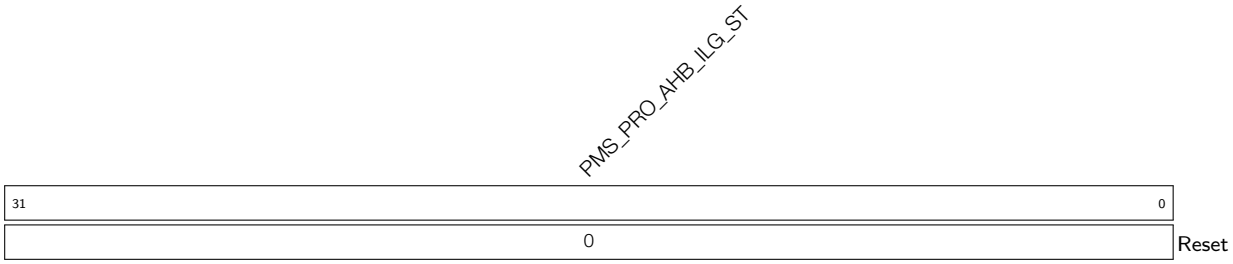


PMS_PRO_AHB_ILG_CLR The clear signal for PeriBus2 access interrupt. (R/W)

PMS_PRO_AHB_ILG_EN The enable signal for PeriBus2 access interrupt. (R/W)

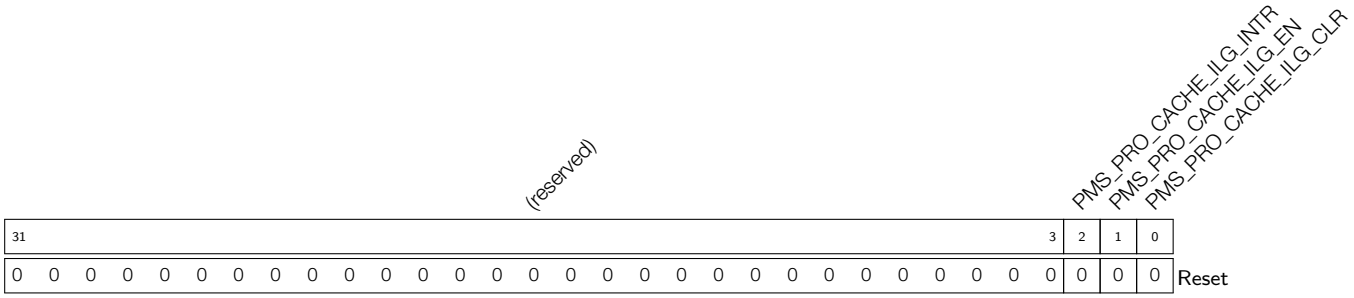
PMS_PRO_AHB_ILG_INTR PeriBus2 access interrupt signal. (RO)

Register 24.45: PMS_PRO_AHB_4_REG (0x006C)



PMS_PRO_AHB_ILG_ST Record the illegitimate information of PeriBus2. [31:2]: store the bits [31:2] of PeriBus2 address; [1]: 1 means data access, 0 means instruction access; [0]: 1 means write operation, 0 means read operation. (RO)

Register 24.46: PMS_PRO_CACHE_2_REG (0x0080)

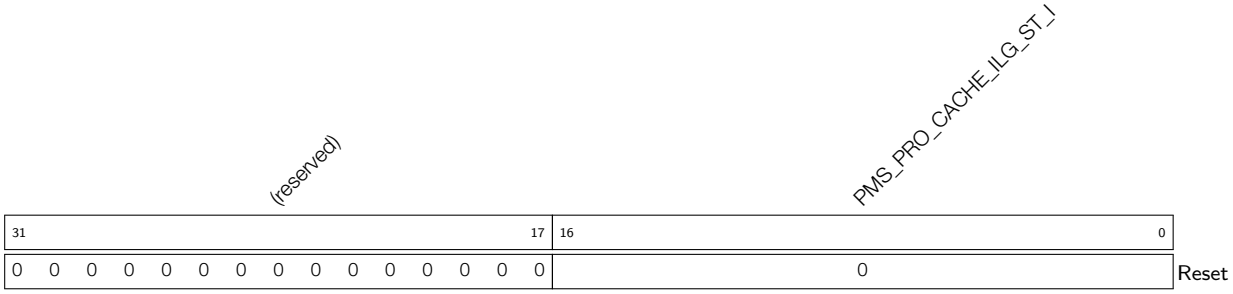


PMS_PRO_CACHE_ILG_CLR The clear signal for cache access interrupt. (R/W)

PMS_PRO_CACHE_ILG_EN The enable signal for cache access interrupt. (R/W)

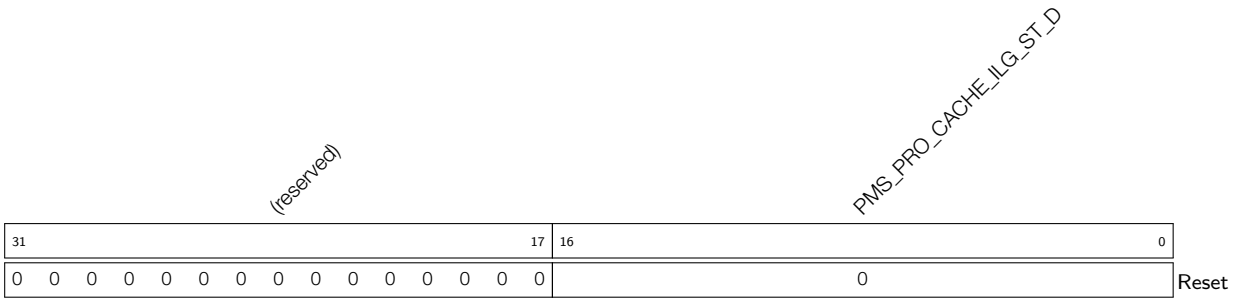
PMS_PRO_CACHE_ILG_INTR Cache access interrupt signal. (RO)

Register 24.47: PMS_PRO_CACHE_3_REG (0x0084)



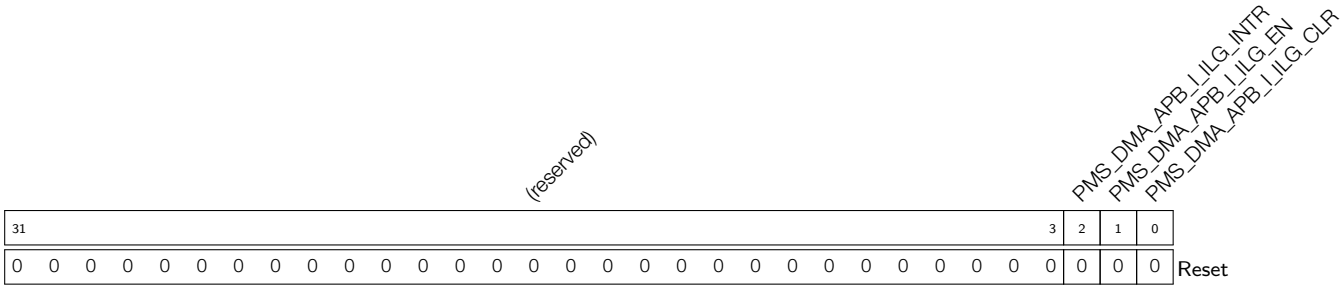
PMS_PRO_CACHE_ILG_ST_I Record the illegitimate information of Icache to access memory. [16]: access enable, active low; [15:4]: store the bits [11:0] of address; [3:0]: Icache bus byte enables, active low. (RO)

Register 24.48: PMS_PRO_CACHE_4_REG (0x0088)



PMS_PRO_CACHE_ILG_ST_D Record the illegitimate information of Dcache to access memory. [16]: access enable, active low; [15:4]: store the bits [11:0] of address; [3:0]: Dcache bus byte enables, active low. (RO)

Register 24.49: PMS_DMA_APB_I_2_REG (0x0094)

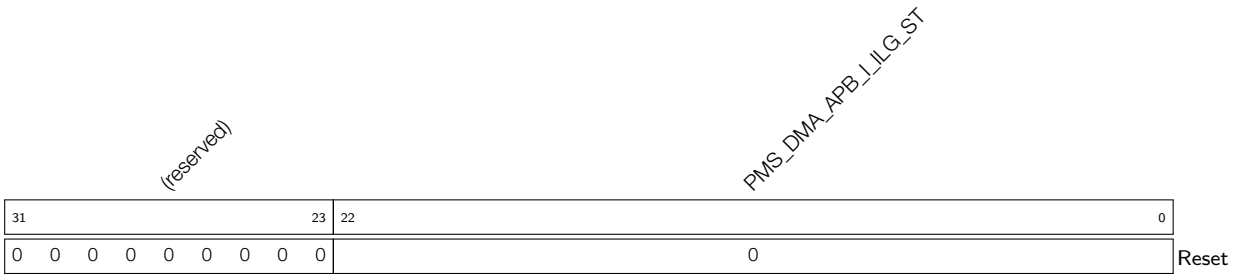


PMS_DMA_APB_I_ILG_CLR The clear signal for internal DMA access interrupt. (R/W)

PMS_DMA_APB_I_ILG_EN The enable signal for internal DMA access interrupt. (R/W)

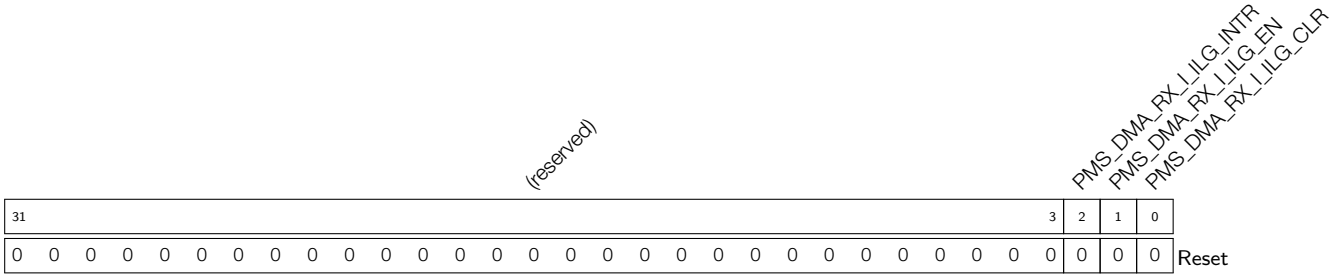
PMS_DMA_APB_I_ILG_INTR Internal DMA access interrupt signal. (RO)

Register 24.50: PMS_DMA_APB_I_3_REG (0x0098)



PMS_DMA_APB_I_ILG_ST Record the illegal information of Internal DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: Internal DMA bus byte enable; (RO)

Register 24.51: PMS_DMA_RX_I_2_REG (0x00A4)

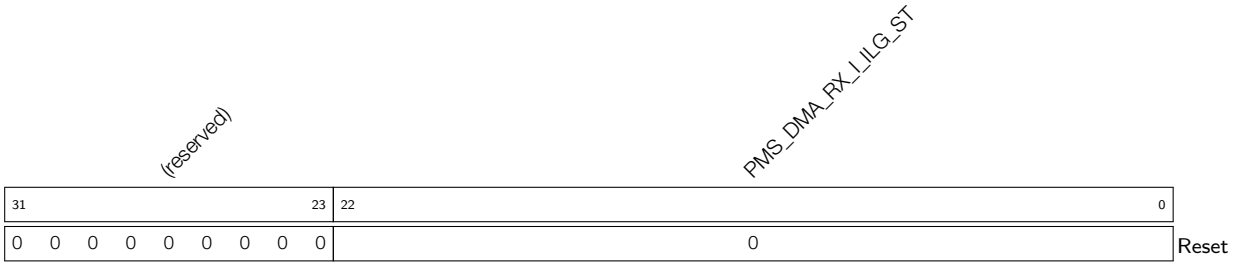


PMS_DMA_RX_I_ILG_CLR The clear signal for RX Copy DMA access interrupt. (R/W)

PMS_DMA_RX_I_ILG_EN The enable signal for RX Copy DMA access interrupt. (R/W)

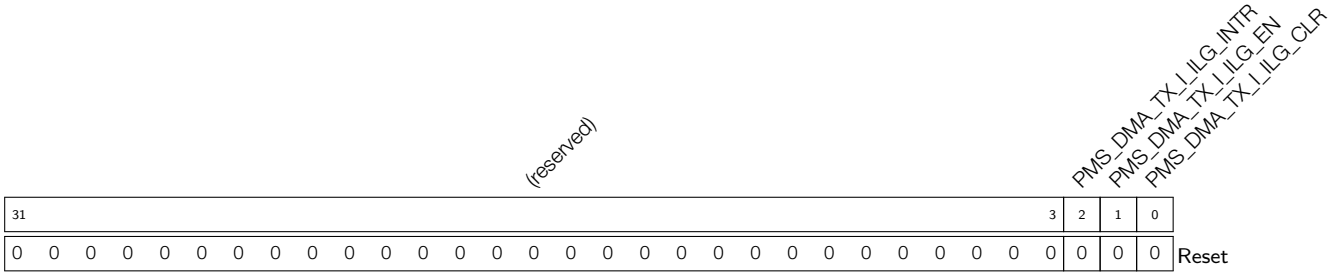
PMS_DMA_RX_I_ILG_INTR RX Copy DMA access interrupt signal. (RO)

Register 24.52: PMS_DMA_RX_I_3_REG (0x00A8)



PMS_DMA_RX_I_ILG_ST Record the illegitimate information of RX Copy DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: RX Copy DMA bus byte enables. (RO)

Register 24.53: PMS_DMA_TX_I_2_REG (0x00B4)

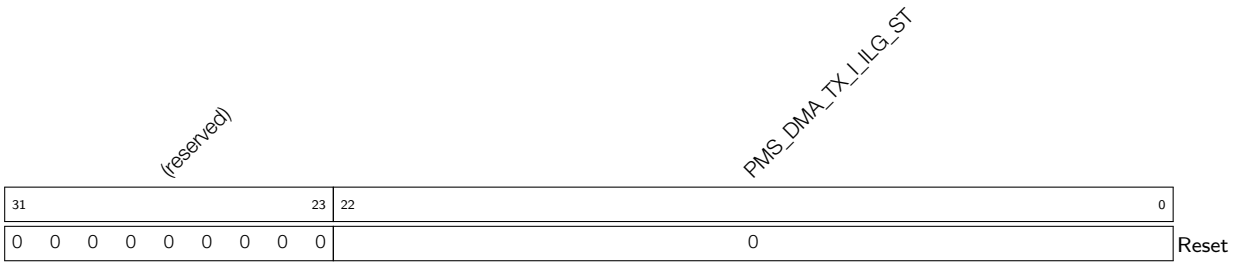


PMS_DMA_TX_I_ILG_CLR The clear signal for TX Copy DMA access interrupt. (R/W)

PMS_DMA_TX_I_ILG_EN The enable signal for TX Copy DMA access interrupt. (R/W)

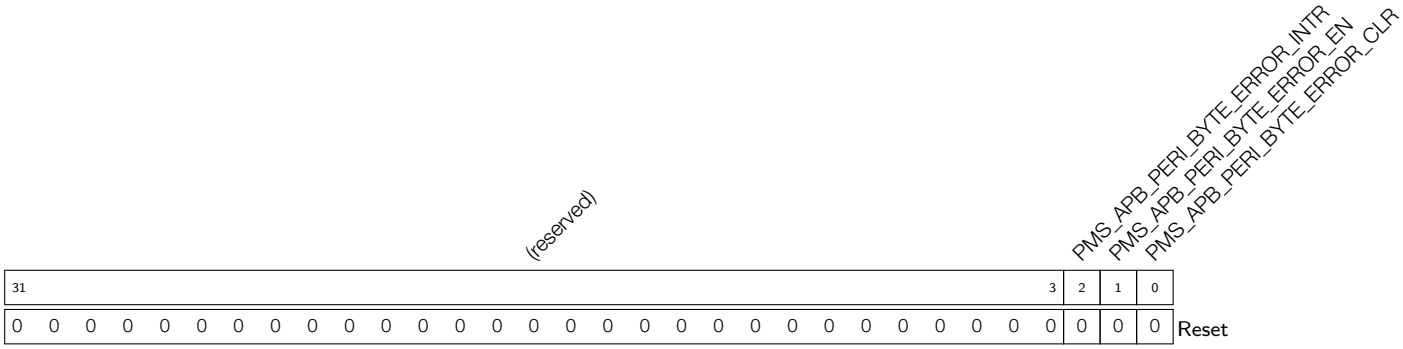
PMS_DMA_TX_I_ILG_INTR TX Copy DMA access interrupt signal. (RO)

Register 24.54: PMS_DMA_TX_I_3_REG (0x00B8)



PMS_DMA_TX_I_ILG_ST Record the illegitimate information of TX Copy DMA. [22:6]: store the bits [18:2] of address; [5]: if bits [31:19] of address are 0x7ff, then the bit value is 1, otherwise it is 0; [4]: 1 means write operation, 0 means read operation; [3:0]: TX Copy DMA bus byte enables. (RO)

Register 24.55: PMS_APB_PERIPHERAL_INTR_REG (0x00F4)

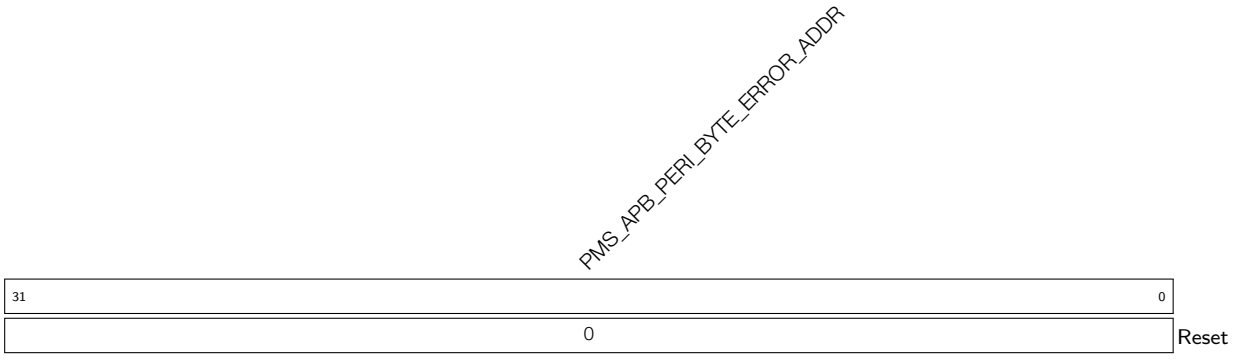


PMS_APB_PERI_BYTE_ERROR_CLR The clear signal for APB peripheral interrupt. (R/W)

PMS_APB_PERI_BYTE_ERROR_EN The enable signal for APB peripheral access interrupt. (R/W)

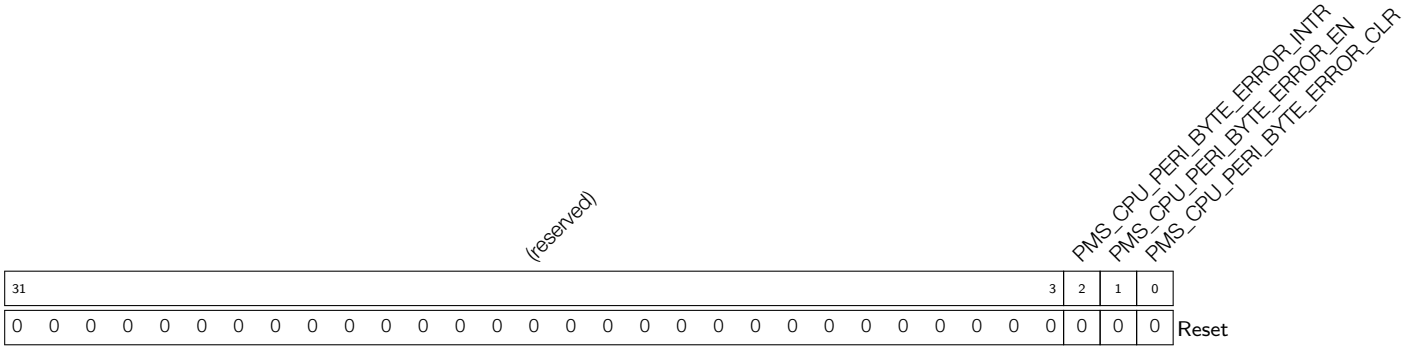
PMS_APB_PERI_BYTE_ERROR_INTR APB peripheral access interrupt signal. (RO)

Register 24.56: PMS_APB_PERIPHERAL_STATUS_REG (0x00F8)



PMS_APB_PERI_BYTE_ERROR_ADDR Record the illegitimate address of APB peripheral. (RO)

Register 24.57: PMS_CPU_PERIPHERAL_INTR_REG (0x00FC)

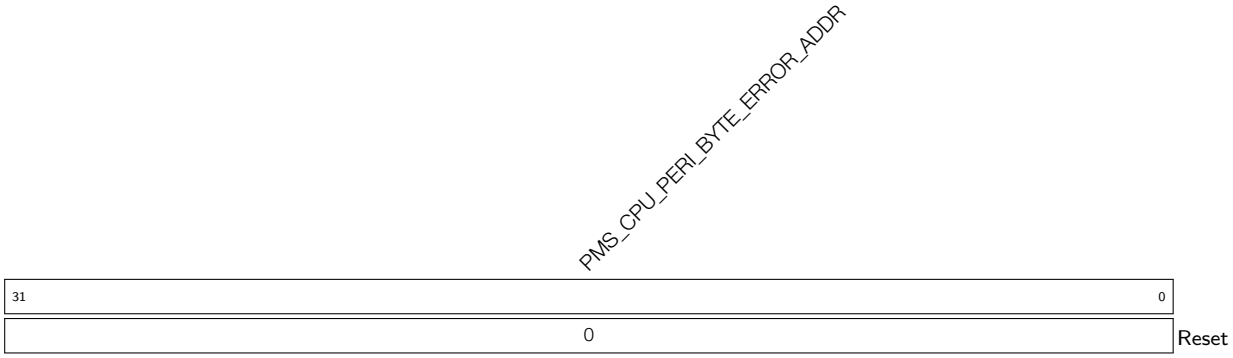


PMS_CPU_PERI_BYTE_ERROR_CLR The clear signal for CPU peripheral access interrupt. (R/W)

PMS_CPU_PERI_BYTE_ERROR_EN The enable signal for CPU peripheral access interrupt. (R/W)

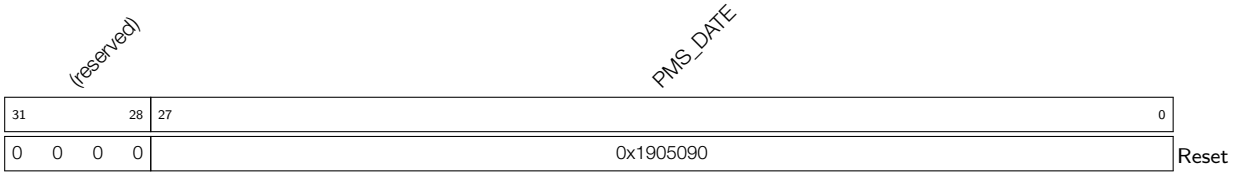
PMS_CPU_PERI_BYTE_ERROR_INTR CPU peripheral access interrupt signal. (RO)

Register 24.58: PMS_CPU_PERIPHERAL_STATUS_REG (0x0100)



PMS_CPU_PERI_BYTE_ERROR_ADDR Record the illegitimate address of CPU peripheral. (RO)

Register 24.59: PMS_DATE_REG (0x0FFC)



PMS_DATE Version control register. (R/W)

25. Digital Signature

25.1 Overview

Digital signatures provide a way to cryptographically authenticate a message using a private key, to be verified using the corresponding public key. This can be used to validate a device's identity to a server, or to authenticate the integrity of a message has not been tampered with.

ESP32-S2 includes a digital signature (DS) peripheral which produces hardware accelerated RSA digital signatures, without the RSA private key being accessible by software.

25.2 Features

- RSA Digital Signatures with key lengths up to 4096 bits
- Private key data is encrypted and only readable by DS peripheral
- SHA-256 digest is used to protect private key data against tampering by an attacker

25.3 Functional Description

25.3.1 Overview

The DS peripheral calculates the RSA encryption operation $Z = X^Y \bmod M$ where Z is the signature, X is the input message, Y and M are the RSA private key parameters.

Private key parameters are stored in flash or another form of storage, in an encrypted form. They are encrypted using a key which can only be read by the DS peripheral via the HMAC peripheral. The required inputs to generate the key are stored in eFuse and can only be accessed by the HMAC peripheral. This means that only the DS peripheral hardware can decrypt the private key, and the plaintext private key data is never accessed by software.

The input message X is input directly to the DS peripheral by software, each time a signature is needed. After the operation, the signature Z is read back by software.

25.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They will have a particular RSA key length (up to 4096 bits). A corresponding public key is also generated and stored separately, it can be used independently to verify DS signatures.

Two additional private key operands are needed — $\overline{r}$ and M'. These two operands are derived from Y and M, but they are calculated in advance by software.

Operands Y, M, $\overline{r}$, and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is input to the DS peripheral in this encrypted format, then the hardware decrypts C and uses the key data to generate the signature. Detailed description of the encryption process to prepare C is provided in Section 25.3.4.

The DS peripheral needs to activate RSA to perform $Z = X^Y \bmod M$. For detailed information on the RSA algorithm, please refer to Section 21.3.1 *Large Number Modular Exponentiation* in Chapter 21 *RSA Accelerator*.

25.3.3 Conventions

The following sections of this chapter will use the following symbols and functions:

- 1^s A bit string that consists of s "1" bits.
- $[x]_s$ A bit string of length s bits. If x is a number ($x < 2^s$), it is represented in little endian byte order in the bit string. x may be a variable value such as $[Y]_{4096}$ or as a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]$ is right-padded with 0s to reach s bits in length. For example: $[0x5]_4 = 0101$, $[0x5]_8 = 00000101$, $[0x5]_{16} = 0000010100000000$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$.
- $||$ A bit string concatenation operator for joining multiple bit strings into a longer bit string.

25.3.4 Software Storage of Private Key Data

To store a private key for use with the DS peripheral, users need to complete the following preparations:

- Generate the RSA private key (Y, M) and associated operands $\overline{r}$ and M', as described in Section 25.3.2.
- Generate a 256-bit HMAC key ($[HMAC\_KEY]_{256}$) that is stored in eFuse. This HMAC key is read by the HMAC peripheral to derive a key, as DS_KEY = HMAC-SHA256($[HMAC\_KEY]_{256}$, 1^{256}). This key is used to securely encrypt and decrypt the stored RSA private key data.
- Prepare encrypted private key parameters as ciphertext C, 1584 bytes in length.

Figure 25-1 below describes the preparations at the software level (the left part) and the DS peripheral operation at the hardware level (the right part).

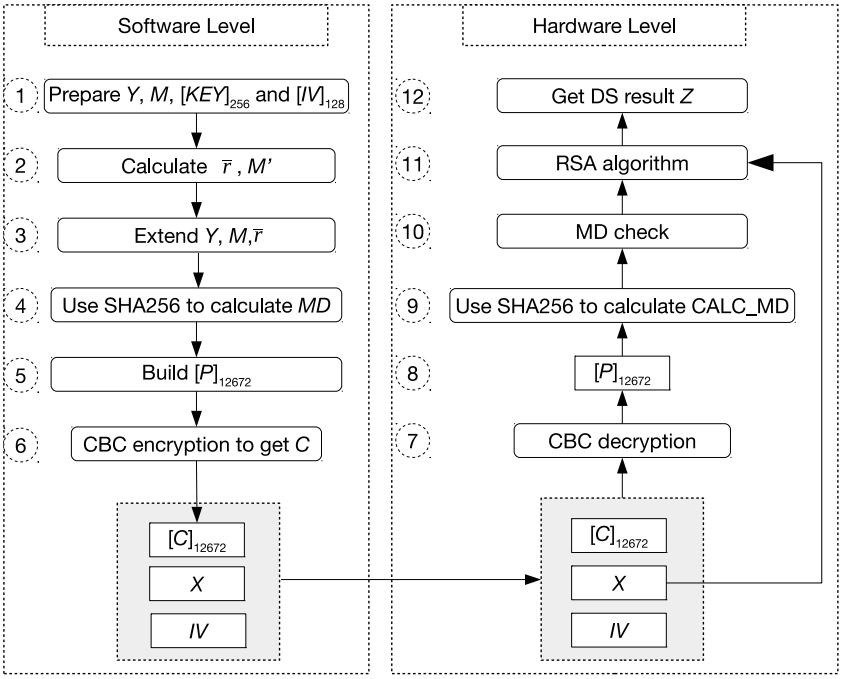


Figure 25-1. Preparations and DS Operation

Users need to follow the steps shown in the left part of Figure 25-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare Y and M whose lengths should meet the aforementioned requirements. Define $[L]_{32} = \frac{N}{32}$ (i.e., for RSA 4096, $[L]_{32}$ == $[0x80]_{32}$). Prepare $[DS\_KEY]_{256}$ and generate a random $[IV]_{128}$ which

should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 19 *AES Accelerator*.

- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M, and $\overline{r}$, in order to get $[Y]_{4096}$, $[M]_{4096}$, and $[\overline{r}]_{4096}$, respectively. Since the largest operand length for Y, M, and $\overline{r}$ is 4096 bits, this step is only required for lengths smaller than 4096 bits.
- **Step 4**: Calculate MD authentication code using the SHA-256 algorithm:

  $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$), where $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a 64-bit string $[0x0808080808080808]_{64}$ that is composed of eight bytes (value = 0x08). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.
- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[DS\_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext that includes RSA operands Y, M, $\overline{r}$, M', and L as well as the MD authentication code and $[\beta]_{64}$. DS_KEY is derived from the $HMAC\_KEY$ stored in eFuse, as described above in Section 25.3.4.

25.3.5 DS Operation at the Hardware Level

The hardware operation is triggered each time a Digital Signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV.

The DS operation at the hardware level is a reverse process of preparing C described in Section 25.3.4. The hardware operation can be divided into the following three stages.

1. **Decryption: Step 7 and 8**

   The decryption process is the reverse of Step 6. The DS peripheral will call AES accelerator to decrypt C in CBC block mode and get the resulted plaintext. The decryption process can be represented by P = AES-CBC-DEC (C, DS_KEY, IV), where IV (i.e., $[IV]_{128}$) is defined by users. $[DS\_KEY]_{256}$ is provided by HMAC module, derived from $HMAC\_KEY$ stored in eFuse. $[DS\_KEY]_{256}$ is not readable by software.

   With P, the DS peripheral can work out $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the reverse of Step 5.

2. **Check: Step 9 and 10**

   The DS peripheral will perform two check operations: MD check and padding check. Padding check is not shown in Figure 25-1, as it happens at the same time with MD check.

   - MD check: The DS peripheral calls SHA-256 to get the hash value $[CALC\_MD]_{256}$. This step is the reverse of Step 4. Then, $[CALC\_MD]_{256}$ is compared against $[MD]_{256}$. Only when the two match, MD check passes.
   - Padding check: The DS peripheral checks if $[\beta]_{64}$ complies with the aforementioned PKCS#7 format. Only when $[\beta]_{64}$ complies with the format, padding check passes.

   If MD check passes, the DS peripheral will perform subsequent operations, otherwise, it will not. If padding check fails, an error bit is set in the query register, but it does not affect the subsequent operations.

3. **Calculation: Step 11 and 12**

The DS peripheral treats X, Y, M, and $\overline{r}$ as big numbers. With M', all operands to perform X^Y mod M are in place. The operand length is defined by L. The DS peripheral will get the signed result Z by calling RSA to perform $Z = X^Y$ mod M.

25.3.6 DS Operation at the Software Level

The following software steps should be followed each time a Digital Signature needs to be calculated. The inputs are the pre-generated private key ciphertext C, a unique message X, and IV. These software steps trigger the hardware steps described in Section 25.3.5.

1. **Activate the DS peripheral**: Write 1 to DS_SET_START_REG.
2. **Check if DS_KEY is ready**: Poll DS_QUERY_BUSY_REG until it reads 0.

   If DS_QUERY_BUSY_REG does not read 0 after approximately 1 ms, it indicates a problem with HMAC initialization. In such case, software can read register DS_QUERY_KEY_WRONG_REG to get more information.

   - If DS_QUERY_KEY_WRONG_REG reads 0, it indicates that HMAC peripheral was not activated.
   - If DS_QUERY_KEY_WRONG_REG reads any value from 1 to 15, it indicates that HMAC was activated, but the DS peripheral did not successfully receive the DS_KEY value from the HMAC peripheral. This may indicate that the HMAC operation was interrupted due to a software concurrency problem.
3. **Configure register**: Write IV block to register DS_IV_*m*_REG (*m*: 0-3). For more information on IV block, please refer to Chapter 19 *AES Accelerator*.
4. **Write X to memory block DS_X_MEM**: Write X_i ($i \in [0, n) \cap \mathbb{N}$) to memory block DS_X_MEM whose capacity is 128 words. Each word can store one base-b digit. The memory block uses the little endian format for storage, i.e., the least significant digit of the operand is in the lowest address. Words in DS_X_MEM block after the configured length of X (N bits, as described in Section 25.3.2) are ignored.
5. **Write C to memory block DS_C_MEM**: Write C_i ($i \in [0, 396) \cap \mathbb{N}$) to memory block DS_C_MEM whose capacity is 396 words. Each word can store one base-b digit.
6. **Start DS operation**: Write 1 to register DS_SET_ME_REG.
7. **Wait for the operation to be completed**: Poll register DS_QUERY_BUSY_REG until it reads 0.
8. **Query check result**: Read register DS_QUERY_CHECK_REG and determine the subsequent operations based on the return value.
   - If the value is 0, it indicates that both padding check and MD check pass. Users can continue to get the signed result Z.
   - If the value is 1, it indicates that the padding check passes but MD check fails. The signed result Z is invalid. The operation would resume directly from Step 10.
   - If the value is 2, it indicates that the padding check fails but MD check passes. Users can continue to get the signed result Z.
   - If the value is 3, it indicates that both padding check and MD check fail. The signed result Z is invalid. The operation would resume directly from Step 10.
9. **Read the signed result**: Read the signed result Z_i ($i \in \{0, 1, 2..., n\}$) from memory block DS_Z_MEM. The memory block stores Z in little-endian byte order.

10. **Exit the operation**: Write 1 to DS_SET_FINISH_REG, then poll DS_QUERY_BUSY_REG until it reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

25.4 Base Address

Users can access the DS peripheral with two base addresses, which can be seen in Table 130. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 130: Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3F43D000
PeriBUS2	0x6003D000

25.5 Memory Blocks

Both the starting address and ending address in the following table are relative to the DS peripheral base addresses provided in Section 25.4.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
DS_C_MEM	Memory block C	1584	0x0000	0x062F	WO
DS_X_MEM	Memory block X	512	0x0800	0x09FF	WO
DS_Z_MEM	Memory block Z	512	0x0A00	0x0BFF	RO

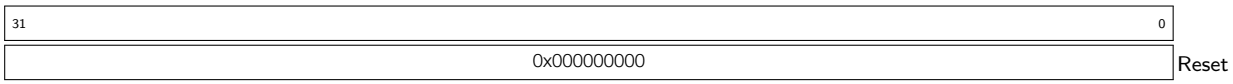
25.6 Register Summary

The addresses in the following table are relative to the DS peripheral base addresses provided in Section 25.4.

Name	Description	Address	Access
Configuration Registers			
DS_IV_0_REG	IV block data	0x0630	WO
DS_IV_1_REG	IV block data	0x0634	WO
DS_IV_2_REG	IV block data	0x0638	WO
DS_IV_3_REG	IV block data	0x063C	WO
Status/Control Registers			
DS_SET_START_REG	Activates the DS peripheral	0x0E00	WO
DS_SET_ME_REG	Starts DS operation	0x0E04	WO
DS_SET_FINISH_REG	Ends DS operation	0x0E08	WO
DS_QUERY_BUSY_REG	Status of the DS	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	Checks the reason why DS_KEY is not ready	0x0E10	RO
DS_QUERY_CHECK_REG	Queries DS check result	0x0814	RO
Version Register			
DS_DATE_REG	Version control egister	0x0820	W/R

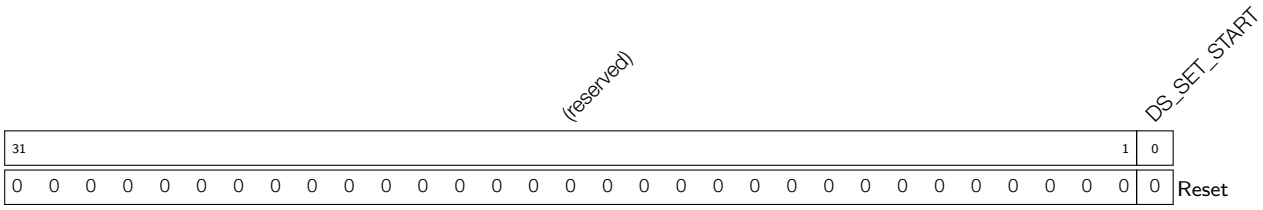
25.7 Registers

Register 25.1: DS_IV_*m*_REG (*m*: 0-3) (0x0630+4m*)**



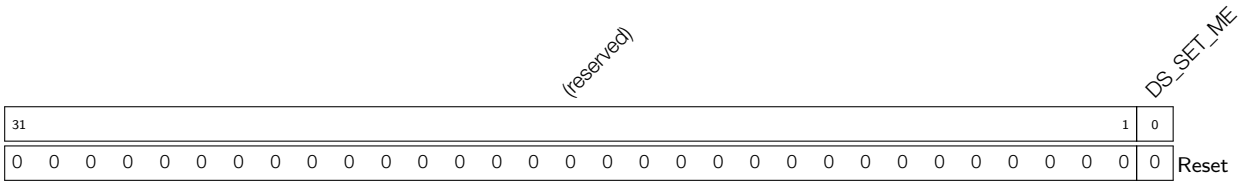
DS_IV_*m*_REG (*m*: 0-3) IV block data. (WO)

Register 25.2: DS_SET_START_REG (0x0E00)



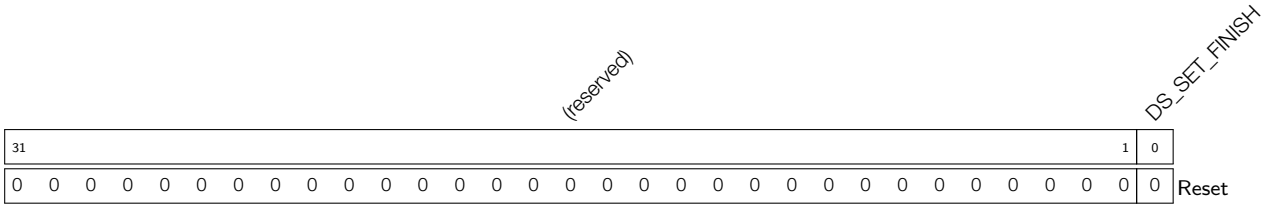
DS_SET_START Write 1 to this register to activate the DS peripheral. (WO)

Register 25.3: DS_SET_ME_REG (0x0E04)



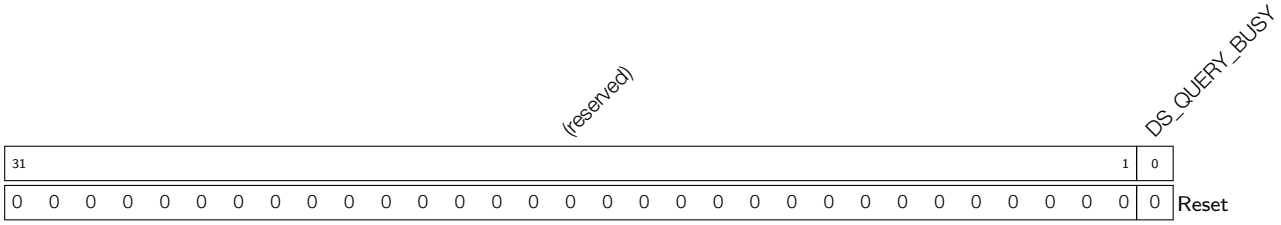
DS_SET_ME Write 1 to this register to start DS operation. (WO)

Register 25.4: DS_SET_FINISH_REG (0x0E08)



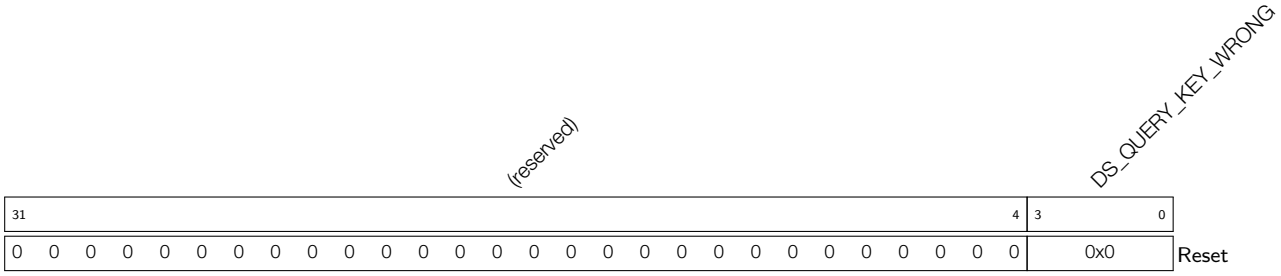
DS_SET_FINISH Write 1 to this register to end DS operation. (WO)

Register 25.5: DS_QUERY_BUSY_REG (0x0E0C)



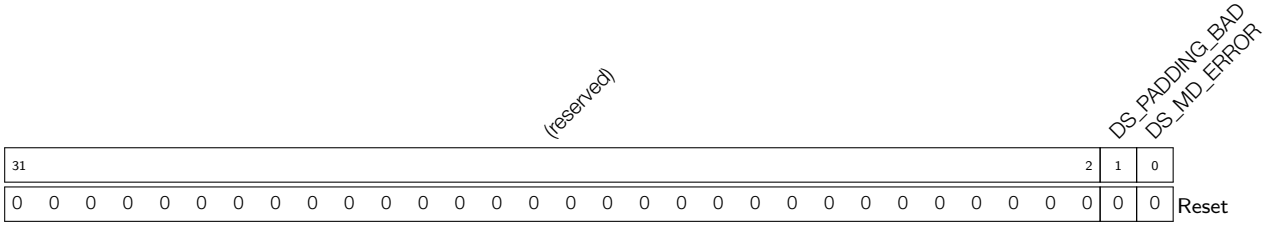
DS_QUERY_BUSY 1: The DS peripheral is busy; 0: The DS peripheral is idle. (RO)

Register 25.6: DS_QUERY_KEY_WRONG_REG (0x0E10)



DS_QUERY_KEY_WRONG 1-15: HMAC was activated, but the DS peripheral did not successfully receive the DS_KEY value from the HMAC peripheral. The biggest value is 15. 0: HMAC is not activated. (RO)

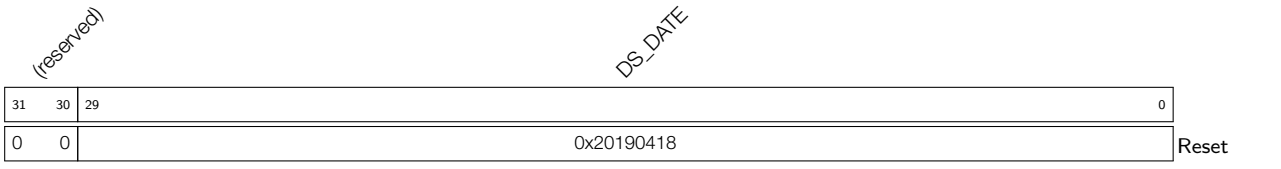
Register 25.7: DS_QUERY_CHECK_REG (0x0E14)



DS_PADDING_BAD 1: The padding check fails; 0: The padding check passes. (RO)

DS_MD_ERROR 1: The MD check fails; 0: The MD check passes. (RO)

Register 25.8: DS_DATE_REG (0x0E20)



DS_DATE Version control register. (R/W)

26. HMAC Module

26.1 Overview

The Hash-based Message Authentication Code (HMAC) module computes Message Authentication Codes (MACs) through Hash algorithms and keys as described in RFC 2104. The underlying hash algorithm is SHA-256, and the 256-bit HMAC key is stored in an eFuse key block and can be configured as not readable by software.

The HMAC module can be used in two modes - in ”upstream” mode the HMAC message is supplied by the user and the calculation result is read back by the user. In ”downstream” mode the HMAC module is used as a Key Derivation Function (KDF) for other internal hardwares.

26.2 Main Features

- Standard HMAC-SHA-256 algorithm
- The hash result is only accessible by the configurable hardware peripheral (downstream mode)
- Supports identity verification challenge-response algorithms
- Supports Digital Signature peripheral (downstream mode)
- Supports re-enabling of Soft-Disabled JTAG (downstream mode)

26.3 Functional Description

26.3.1 Upstream Mode

In Upstream mode, the HMAC message is provided by the user and the result is read back by the user.

This allows the key stored in eFuse (can be configured as not readable by software) to become a shared secret between the user and another party. Any challenge-response protocol that supports HMAC-SHA-256 can be used in this way.

The generalized form of these protocols is as follows:

- A calculates a unique nonce message M
- A sends M to B
- B calculates HMAC (M, KEY) and sends to A
- A also calculates HMAC (M, KEY) internally
- A compares both results, If the same, then the identity of B is verified

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with corresponding purpose eFuse set to EFUSE_KEY_PURPOSE_HMAC_UP. See Chapter 16 for details.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. A copy of this secret key should be kept by any other party that wants to authenticate this device.

To calculate an HMAC:

1. User initializes the HMAC module,and enter upstream mode.

2. User writes the correctly padded message to the peripheral, one block at a time.
3. User reads back the HMAC result from peripheral registers.

See Section 26.3.5 for detailed steps of this process.

26.3.2 Downstream JTAG Enable Mode

eFuse memory has two parameters to disable JTAG debugging: EFUSE_HARD_DIS_JTAG and EFUSE_SOFT_DIS_JTAG. JTAG will be disabled permanently if the former is programmed to 1, and it will be disabled temporarily if the latter is programmed to 1. See Chapter 16 for details.

The HMAC peripheral can be used to re-enable JTAG when EFUSE_SOFT_DIS_JTAG is programmed.

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with purpose set to either EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG or EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL. In the latter case, the same key can be used for both DS and JTAG re-enable functions.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. User stores the randomly generated HMAC key in the process securely elsewhere.
3. Program the eFuse EFUSE_SOFT_DIS_JTAG to 1.

To re-enable JTAG:

1. User performs HMAC calculation on the 32-byte 0x00 locally using SHA-256 and the known random key, and inputs this pre-calculated value into the registers SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 ~ SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7.
2. User enables the HMAC module, and enters downstream JTAG enable mode.
3. If the HMAC calculated result matches the value supplied in the registers SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_0 ~ SYSTEM_CANCEL_EFUSE_DISABLE_JTAG_TEMPORARY_7, JTAG is re-enabled. Otherwise, JTAG remains disabled.
4. JTAG remains the status in step 3 until the user writes 1 in register HMAC_SET_INVALIDATE_JTAG_REG, or restarts the system.

See Section 26.3.5 for detailed steps of this process.

26.3.3 Downstream Digital Signature Mode

The Digital Signature (DS) module encrypts its parameters using AES-CBC. The HMAC module is used as a Key Derivation Function (KDF) to derive the AES key used for this encryption.

To set up the key:

1. A 256-bit HMAC key is randomly generated and is programmed to an eFuse key block with purpose set to either EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE or EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL. In the latter case, the same key can be used for both DS and JTAG re-enable functions.
2. Configure the eFuse key block to be read protected, so software cannot read back the value. If necessary a copy of the key can also be stored in a secure location.

Before using the DS module, software needs to enable the calculation task of HMAC module's downstream DS mode. The above calculation result will be used as the key when the DS mode performs the calculation task. Consult the 25 chapter for details.

26.3.4 HMAC eFuse Configuration

The correct implementation of the HMAC module depends on whether the selected eFuse key block is consistent with the configured HMAC function.

Configure HMAC Function

Currently, HMAC module supports three functions: JTAG re-enable in downstream mode, DS Key Derivation in downstream mode, and HMAC calculation in upstream mode. Table 133 lists the configuration register value corresponding to each function. The values corresponding to the function in use should be written into the register HMAC_SET_PARA_PURPOSE_REG (see Section 26.3.5).

Table 133: HMAC Function and Configuration Value

Functions	Mode Type	Value	Description
JTAG Re-enable	Downstream	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
DS Key Derivation	Downstream	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC Calculation	Upstream	8	EFUSE_KEY_PURPOSE_HMAC_UP
Both JTAG Re-enable and DS KDF	Downstream	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

Select eFuse Key Blocks

The eFuse controller provides six key blocks, KEY0 ~ 5. To select a particular KEYn for HMAC module use at runtime, user writes the number n into register HMAC_SET_PARA_KEY_REG.

Note that the purpose of the key is also programmed into eFuse memory. Only when the configured HMAC purpose matches the purpose defined in KEYn, will the HMAC module execute the configured computation.

For more information see 16 chapter.

For example, suppose the user selects KEY3 for the computation, and the value programmed in KEY_PURPOSE_3 is 6 (EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG). Based on Table 133, KEY3 is the key used for JTAG restart. If the configured value of HMAC_SET_PARA_PURPOSE_REG is also 6, then the HMAC peripheral will allow the JTAG start the computation of JTAG re-enable.

26.3.5 HMAC Process (Detailed)

The process to call HMAC in ESP32-S2 is as follows:

1. Enable HMAC module
   (a) Enable the peripheral clock bits for HMAC and SHA peripherals, and clear the corresponding peripheral reset bits.
   (b) Write 1 into register HMAC_SET_START_REG.
2. Configure HMAC keys and key functions

(a) Write m representing key functions into register HMAC_SET_PARA_PURPOSE_REG. Correlation between value m and key functions is shown in Table 55. Refer to Section 26.3.4.

(b) Select KEYn of eFuse memory as the key by writing n into register HMAC_SET_PARA_KEY_REG (n in the range of 0 to 5). Refer to Section 26.3.4.

(c) Finish the configuration by writing 1 into register HMAC_SET_PARA_FINISH_REG.

(d) Read register HMAC_QUERY_ERROR_REG. Value of 1 means the selected key block does not match the configured key purpose, and computation will not proceed. Value of 0 means the selected key block matches the configured key purpose, and computation can proceed.

(e) Setting HMAC_SET_PARA_PURPOSE_REG to values other than 8 means HMAC module will operate in downstream mode, proceed with Step 3. Setting value 8 means HMAC module will operate in upstream mode, proceed with Step 4.

3. Downstream Mode

(a) Poll state register HMAC_QUERY_BUSY_REG. The register value of 0 means HMAC computation in downstream mode is finished.

(b) In downstream mode, the result is used by JTAG module or DS module in the hardware. Users can write 1 into register HMAC_SET_INVALIDATE_JTAG_REG to clean the result generated by JTAG key; or can write 1 into register HMAC_SET_INVALIDATE_DS_REG to clean the result generated by digital signature key.

(c) This is the end of the HMAC downstream operation.

4. Upstream Mode Transmit message block Block_n ($n >= 1$)

(a) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

(b) Write 512-bit message block Block_n into register range HMAC_WDATA0~15_REG. Then write 1 in register HMAC_SET_MESSAGE_ONE_REG, and HMAC module will compute this message block.

(c) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

(d) Subsequent message blocks are different, depending on whether the size of to-be-processed data is a multiple of 512 bits.

- If the bit length of the message is a multiple of 512, there are three possible options:
  i. If Block_n+1 exists, write 1 in register HMAC_SET_MESSAGE_ING_REG to make $n = n + 1$, then jump to step 4.(b).
  ii. If Block_n is the last block of the message, and user wishes to apply SHA padding through hardware, write 1 in register HMAC_SET_MESSAGE_END_REG, then jump to step 6.
  iii. If Block_n is the last block of the padded message, and the user has applied SHA padding in software, write 1 in register HMAC_SET_MESSAGE_PAD_REG, and jump to step 5.
- If the bit length of the message is not a multiple of 512, there are three possible options. Note that in this case the user is required to apply SHA padding to the message, after which the padded message length will be a multiple of 512 bits.
  i. If Block_n is the only message block, $n = 1$, and Block_1 has included all padding bits, write 1 in register HMAC_ONE_BLOCK_REG, and then jump to step 6.

ii. If Block_n is the second last block of the padded message, write 1 in register HMAC_SET_MESSAGE_PAD_REG, and jump to step 5.

iii. If Block_n is neither the last nor the second last message block, write 1 in register HMAC_SET_MESSAGE_ING_REG and make $n = n + 1$, then jump to step 4.(b).

5. Apply SHA Padding to Message

   (a) Users apply SHA padding to the final message block as described in Section 26.4.1, Write this block in register HMAC_WDATA0~15_REG, then write 1 in register HMAC_SET_MESSAGE_ONE_REG. HMAC module will compute the message block.

   (b) Jump to step 6.

6. Read hash result in upstream mode

   (a) Poll state register HMAC_QUERY_BUSY_REG. Go to the next step when the value of the register is 0.

   (b) Read hash result from register HMAC_RDATA0~7_REG.

   (c) Write 1 in register HMAC_SET_RESULT_FINISH_REG to finish the computation.

Note:
The SHA accelerator can be called directly, or used internally by the DS module and the HMAC module. However, they can not share the hardware resources simultaneously. Therefore, SHA module can not be called by CPU or DS module when HMAC module is in use.

26.4 HMAC Algorithm Details

26.4.1 Padding Bits

HMAC module uses the SHA-256 hash algorithm. If the input message is not a multiple of 512 bits, the user must apply SHA-256 padding algorithm in software. The SHA-256 padding algorithm is described here, and is the same as "5.1 Padding the Message" of "FIPS PUB 180-4".

As shown in Figure 26-1, suppose the length of the unpadded message is m bits. Padding steps are as follows:

1. Append a single 1-bit "1" to the end of unpadded message;
2. Append k bits of value "0", where k is the smallest non-negative number which satisfies $m + 1 + k \equiv 448 (mod 512)$;
3. Append a 64-bit integer value as a binary block. The content of this block is the length of the the unpadded message as a big-endian binary integer value m.

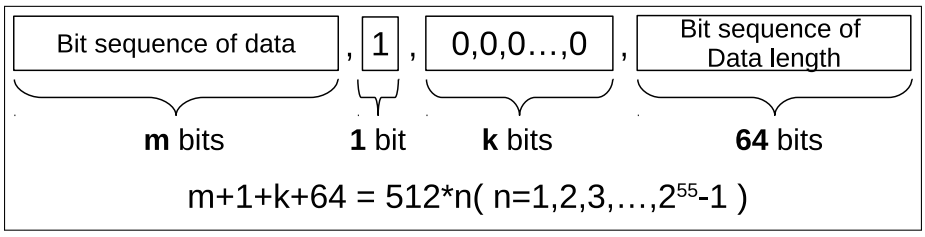


Figure 26-1. HMAC SHA-256 Padding Diagram

In downstream mode, there is no need for users to input any message data or apply padding. In upstream mode, if the total number of bits in the unpadded message is a multiple of 512, users can choose to configure hardware to apply SHA padding. If the total number of bits in the unpadded message is not a multiple of 512, SHA padding must be applied by the user. See Section 26.3.5 for the steps involved.

26.4.2 HMAC Algorithm Structure

The Structure of HMAC algorithm as implemented in the HMAC module is shown in Figure 26-2. This is the standard HMAC algorithm as described in RFC 2104.

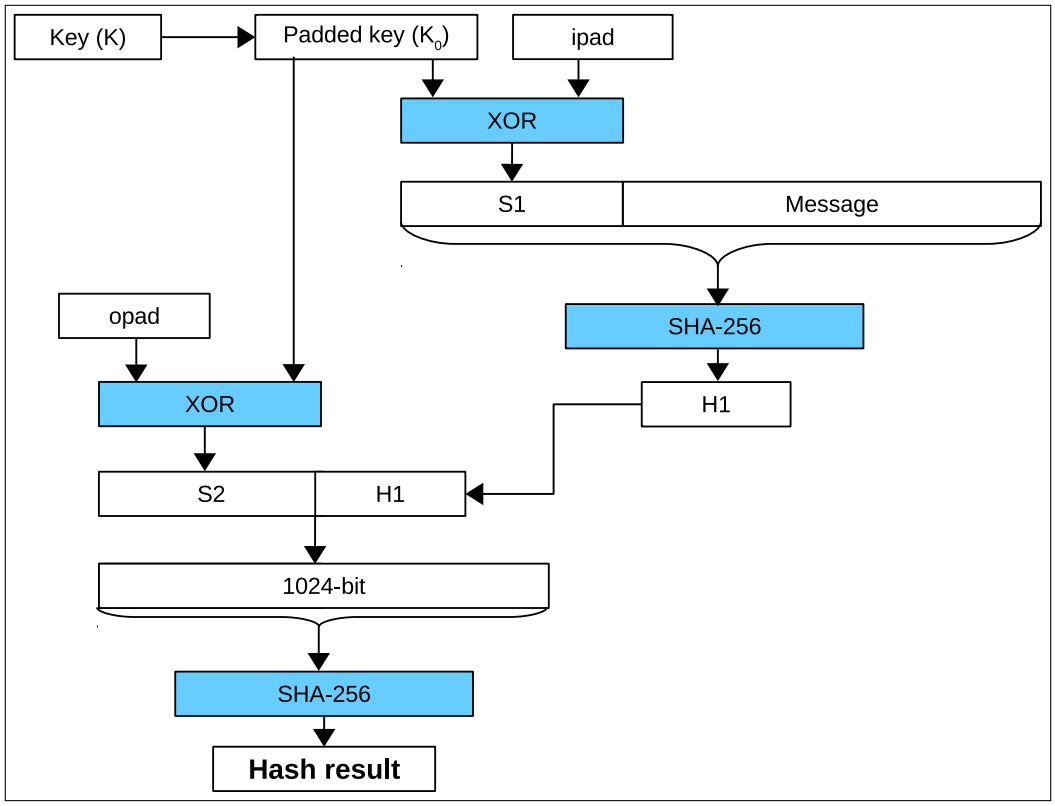


Figure 26-2. HMAC Structure Schematic Diagram

In Figure 26-2,

1. ipad is 512-bit message block composed of sixty-four 0x36 byte.
2. opad is 512-bit message block composed of sixty-four 0x5c byte.

HMAC module appends 256-bit 0 sequence after the bit sequence of 256-bit key k, and gets 512-bit K_0. Afterwards, HMAC module XORs K_0 with ipad to get 512-bit S1. Then, HMAC module appends the input message which is a multiple of 512 after 512-bit S1, and processes SHA-256 algorithm to get 256-bit H1.

HMAC module appends the 256-bit SHA-256 hash result to the 512-bit S2 value, which is calculated by XOR between K_0 and opad, this produces a 768-bit sequence. Then, HMAC module uses SHA padding algorithm described in Section 26.4.1 to pad 768-bit sequence into 1024-bit sequence, and applies SHA-256 algorithm to get the final hash result.

26.5 Base Address

Users can access HMAC module with base address, which can be seen in the following table.

Table 134: HMAC Base Address

Base Address	Address Value
PeriBUS1	0x3F43E000
PeriBUS2	0x6003E000

26.6 Register Summary

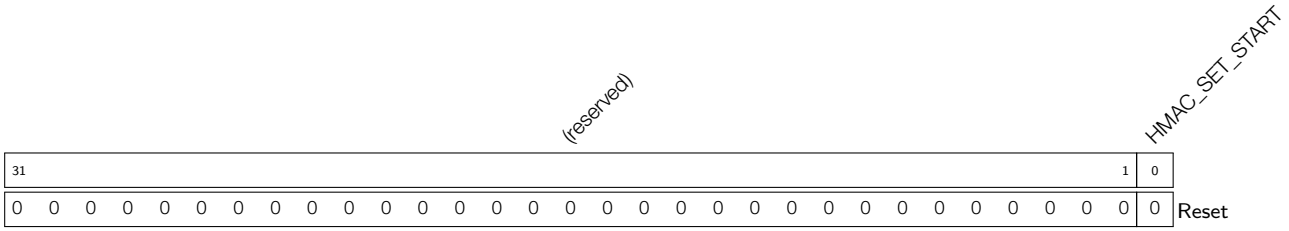
The addresses in the following table are relative to the system registers base addresses provided in Section 26.5.

Name	Description	Address	Access
Control/Status Registers			
HMAC_SET_START_REG	HMAC start control register	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC configuration completion register	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC one message control register	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC message continue register	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC message end register	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC read result completion register	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	Invalidate JTAG result register	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	Invalidate digital signature result register	0x0064	WO
HMAC_QUERY_ERROR_REG	The matching result between key and purpose user configured	0x0068	RO
HMAC_QUERY_BUSY_REG	The busy state of HMAC module	0x006C	RO
configuration Registers			
HMAC_SET_PARA_PURPOSE_REG	HMAC parameter configuration register	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC key configuration register	0x0048	WO
HMAC Message Block			
HMAC_WR_MESSAGE_0_REG	Message register 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	Message register 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	Message register 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	Message register 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	Message register 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	Message register 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	Message register 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	Message register 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	Message register 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	Message register 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	Message register 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	Message register 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	Message register 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	Message register 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	Message register 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	Message register 15	0x00BC	WO
HMAC Upstream Result			
HMAC_RD_RESULT_0_REG	Hash result register 0	0x00C0	RO

Name	Description	Address	Access
HMAC_RD_RESULT_1_REG	Hash result register 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash result register 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash result register 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash result register 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash result register 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash result register 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash result register 7	0x00DC	RO
Control/Status registers			
HMAC_SET_MESSAGE_PAD_REG	Software padding register	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block message register	0x00F4	WO
Version Register			
HMAC_DATE_REG	Version control register	0x00F8	R/W

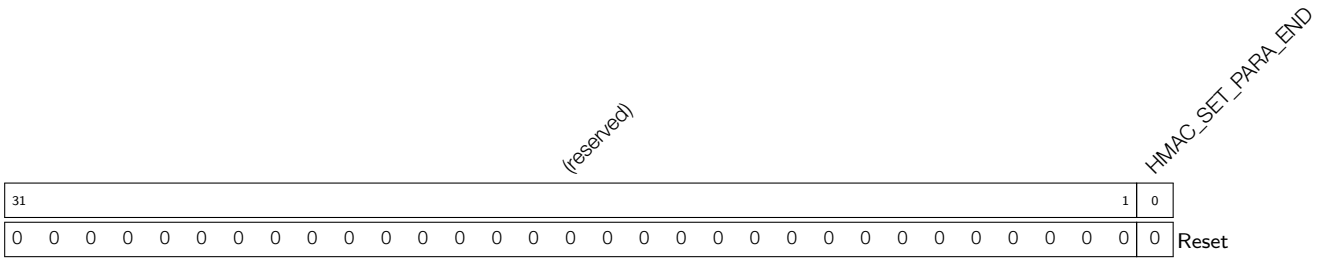
26.7 Registers

Register 26.1: HMAC_SET_START_REG (0x0040)



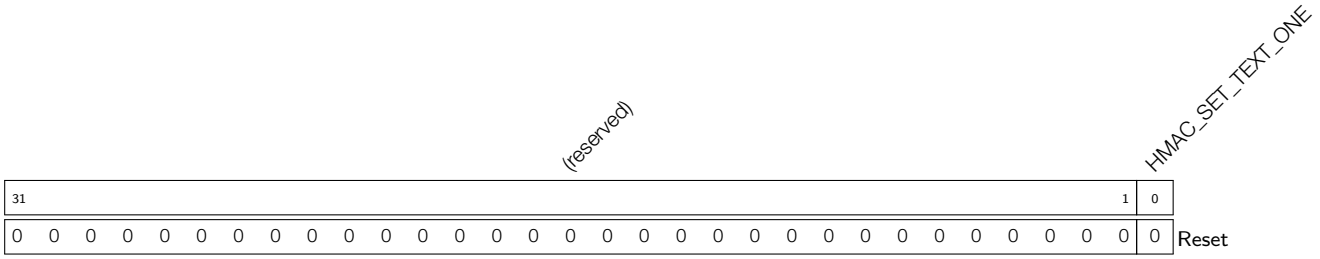
HMAC_SET_START Set this bit to enable HMAC. (WO)

Register 26.2: HMAC_SET_PARA_FINISH_REG (0x004C)



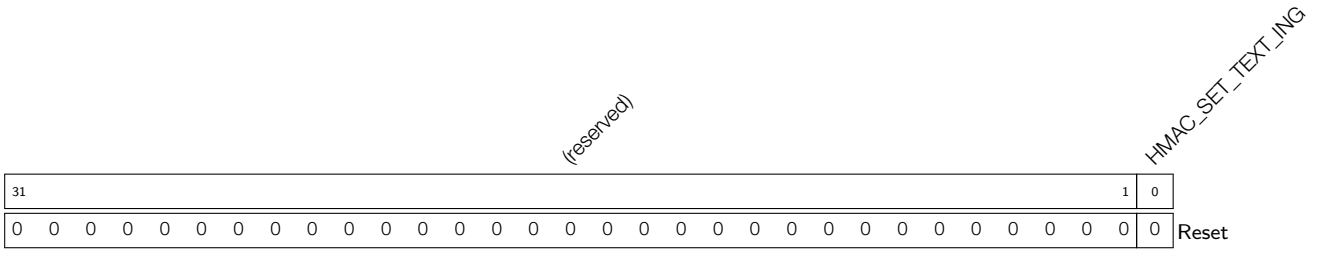
HMAC_SET_PARA_END Set this bit to finish HMAC configuration. (WO)

Register 26.3: HMAC_SET_MESSAGE_ONE_REG (0x0050)



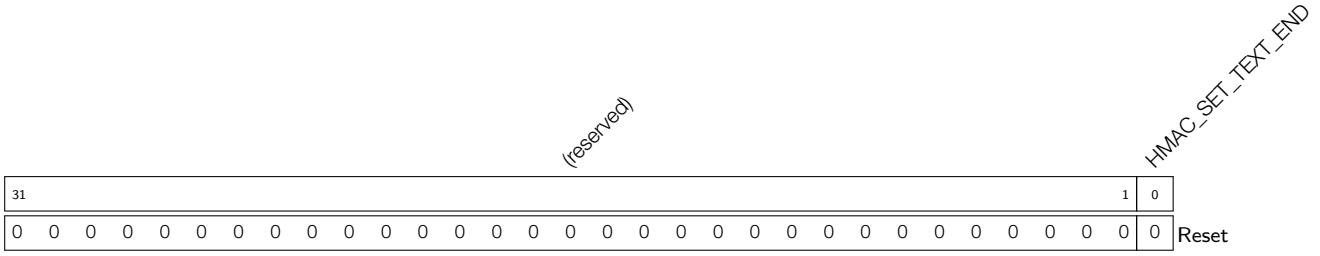
HMAC_SET_TEXT_ONE Call SHA to calculate one message block. (WO)

Register 26.4: HMAC_SET_MESSAGE_ING_REG (0x0054)



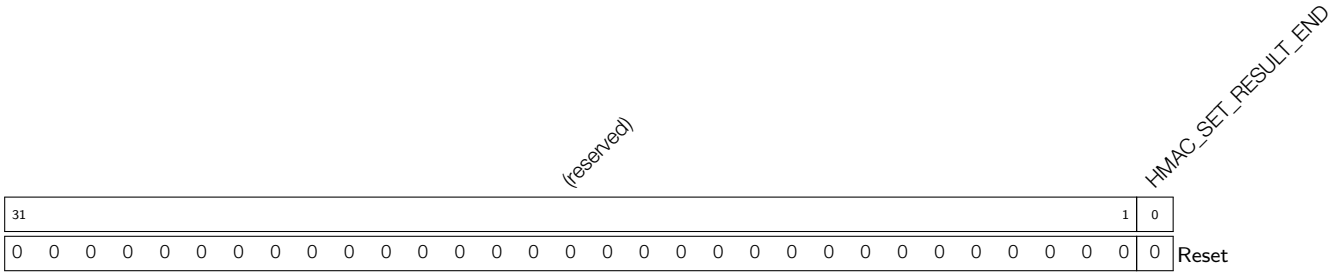
HMAC_SET_TEXT_ING Set this bit to show there are still some message blocks to be processed. (WO)

Register 26.5: HMAC_SET_MESSAGE_END_REG (0x0058)



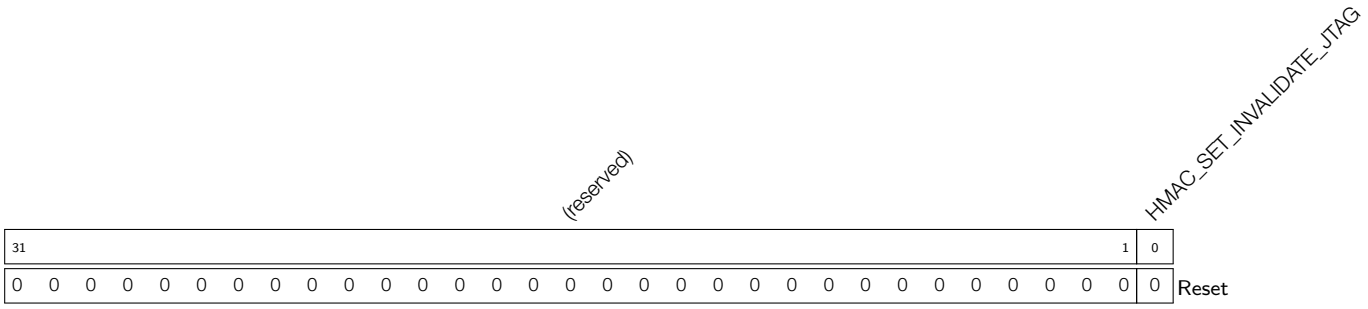
HMAC_SET_TEXT_END Set this bit to start hardware padding. (WO)

Register 26.6: HMAC_SET_RESULT_FINISH_REG (0x005C)



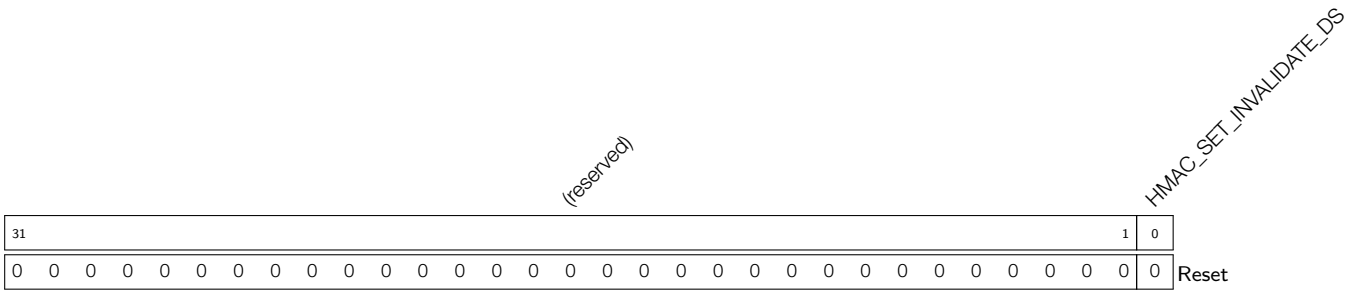
HMAC_SET_RESULT_END Set this bit to end upstream and clear the calculation result. (WO)

Register 26.7: HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



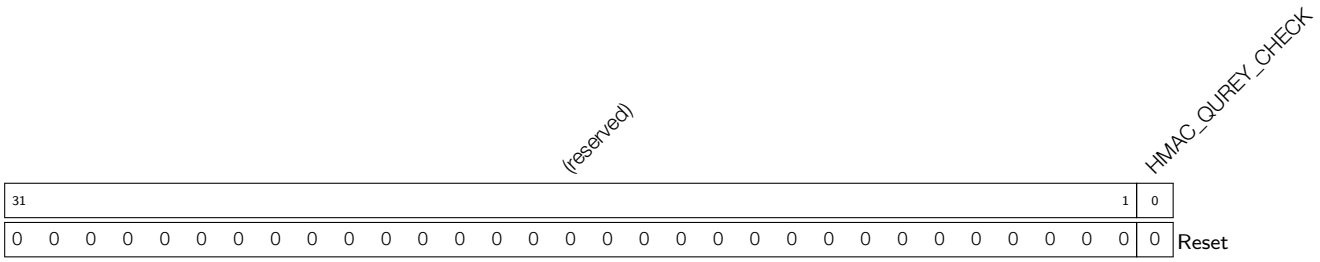
HMAC_SET_INVALIDATE_JTAG Set this bit to clear calculation results in JTAG re-enable function under downstream mode. (WO)

Register 26.8: HMAC_SET_INVALIDATE_DS_REG (0x0064)



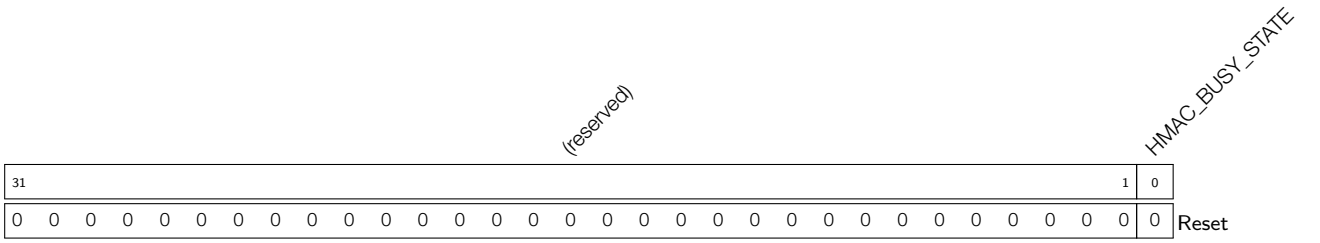
HMAC_SET_INVALIDATE_DS Set this bit to clear calculation results in DS function under downstream mode. (WO)

Register 26.9: HMAC_QUERY_ERROR_REG (0x0068)



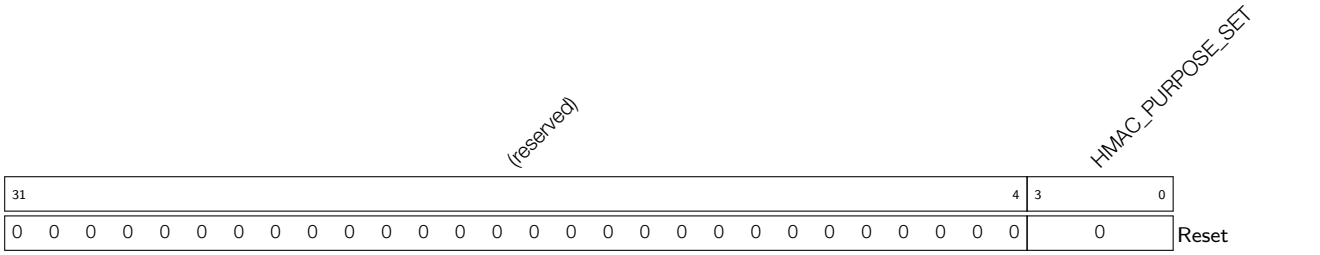
HMAC_QUREY_CHECK HMAC error status. 0: HMAC key and purpose match. 1: error. (RO)

Register 26.10: HMAC_QUERY_BUSY_REG (0x006C)



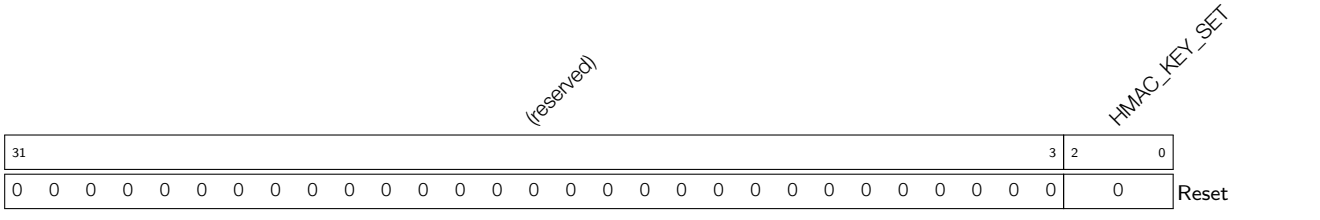
HMAC_BUSY_STATE The state of HMAC. 1'b0: idle. 1'b1: busy. (RO)

Register 26.11: HMAC_SET_PARA_PURPOSE_REG (0x0044)



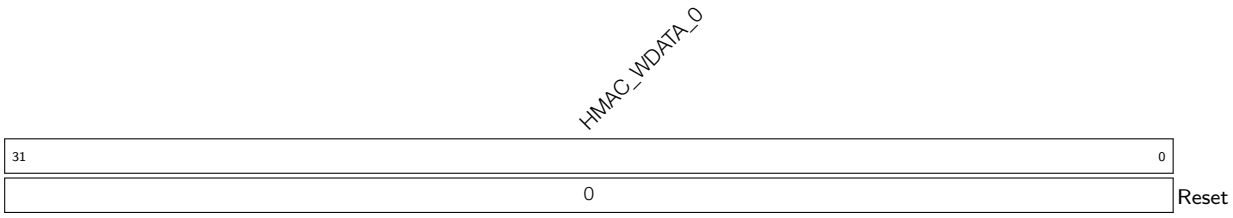
HMAC_PURPOSE_SET Set HMAC purpose. (WO)

Register 26.12: HMAC_SET_PARA_KEY_REG (0x0048)



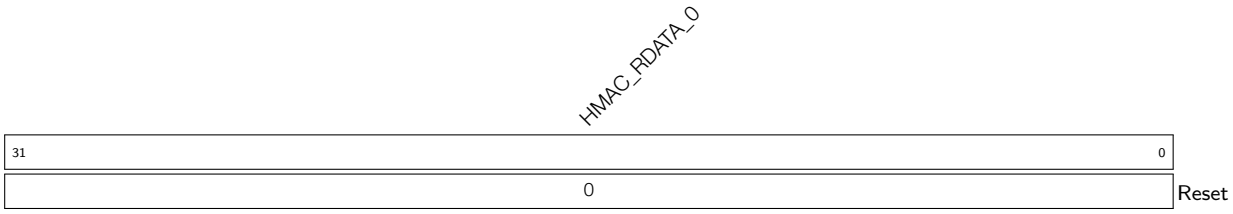
HMAC_KEY_SET Select HMAC key. (WO)

Register 26.13: HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4n*)**



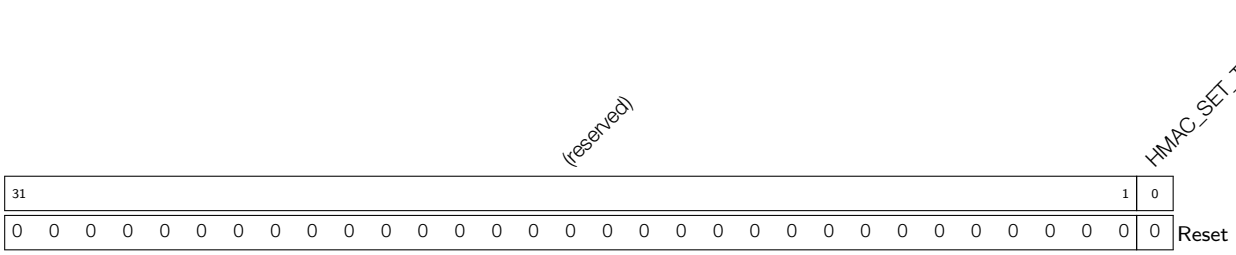
HMAC_WDATA_*n* Store the *n*th 32-bit of message. (WO)

Register 26.14: HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4n*)**



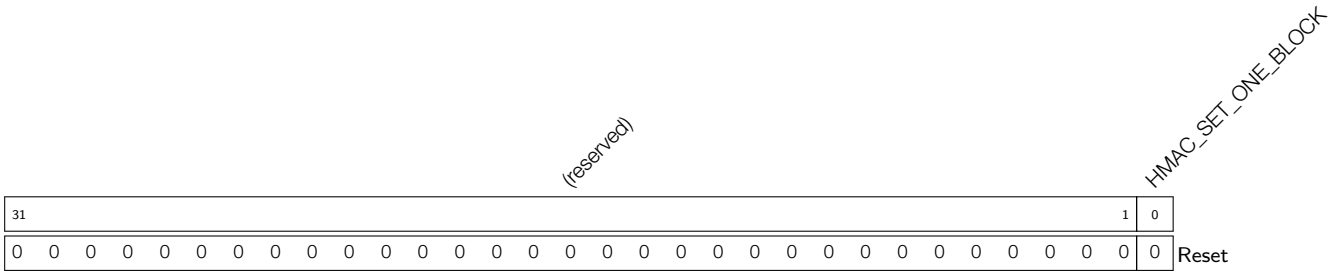
HMAC_RDATA_*n* Read the *n*th 32-bit of hash result. (RO)

Register 26.15: HMAC_SET_MESSAGE_PAD_REG (0x00F0)



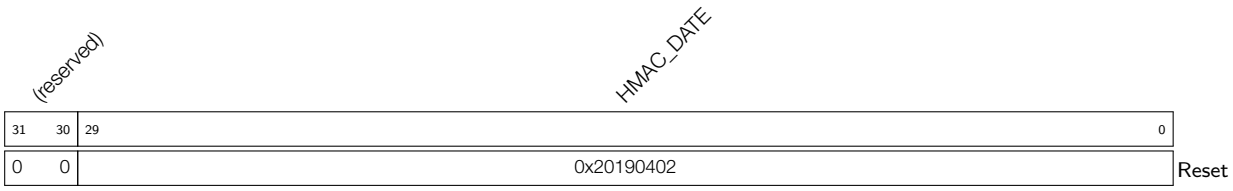
HMAC_SET_TEXT_PAD Set this bit to let software do padding job. (WO)

Register 26.16: HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK Set this bit to show no padding is required. (WO)

Register 26.17: HMAC_DATE_REG (0x00F8)



HMAC_DATE Store HMAC version information. (R/W)

27. ULP Coprocessor

27.1 Overview

The ULP coprocessor is an ultra-low-power processor that remains powered on when the chip is in Deep-sleep (see Chapter 28 *Low-power Management*). Hence, users can store in RTC memory a program for the ULP coprocessor to access peripheral devices, internal sensors and RTC registers during Deep-sleep.

In power-sensitive scenarios, the main CPU goes to sleep mode to lower power consumption. Meanwhile, the coprocessor is woken up by ULP timer, and then monitors the external environment or interacts with the external circuit by controlling peripheral devices such as RTCIO, RTC I²C, SAR ADC, or temperature sensor (TSENS). The coprocessor wakes the main CPU up once a wakeup condition is reached.

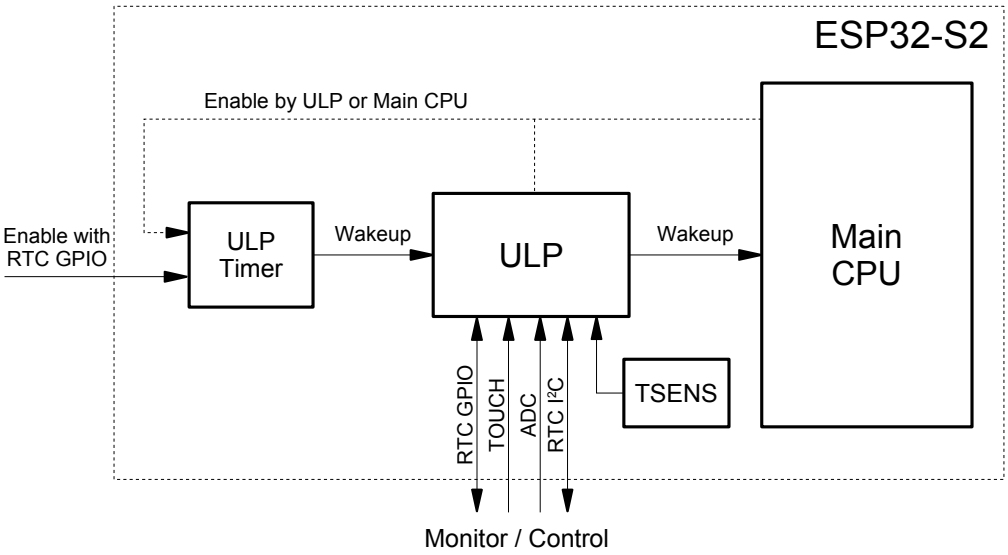


Figure 27-1. ULP Coprocessor Overview

ESP32-S2 has two ULP coprocessors, with one based on RISC-V instruction set architecture (ULP-RISC-V) and the other on finite state machine (ULP-FSM). Users can choose between the two coprocessors depending on their needs.

27.2 Features

- Access up to 8 KB of SRAM RTC slow memory for instructions and data
- Clocked with 8 MHz RTC_FAST_CLK
- Support working in normal mode and in monitor mode
- Wake up the CPU or send an interrupt to the CPU
- Access peripherals, internal sensors and RTC registers

ULP-FSM and ULP-RISC-V can not be used simultaneously. Users can only choose one of them as the ULP coprocessor of ESP32-S2. The differences between the two coprocessors are shown in the table below.

Feature		Coprocessors	
		ULP-FSM	ULP-RISC-V
Memory (RTC Slow Memory)		8 KB	
Work Clock Frequency		8 MHz	
Wakeup Source		ULP Timer	
Work Mode	Normal Mode	Assist the main CPU to complete some tasks after the system is woken up.	
	Monitor Mode	Control sensors to do tasks such as monitoring environment, when the system is in sleep.	
Control Low-Power Peripherals		ADC0/ADC1	
		DAC0/DAC1	
		RTC I^2C	
		RTC GPIO	
		Touch Sensor	
		Temperature Sensor	
Architecture		Programmable FSM	RISC-V
Development		Special instruction set	Standard C compiler

Table 136: Comparison of the Two Coprocessors

ULP coprocessor can access the modules in RTC domain via RTC registers. In many cases the ULP coprocessor can be a good supplement to, or replacement of, the main CPU, especially for power-sensitive applications. Figure 27-2 shows the overall layout of ESP32-S2 coprocessor.

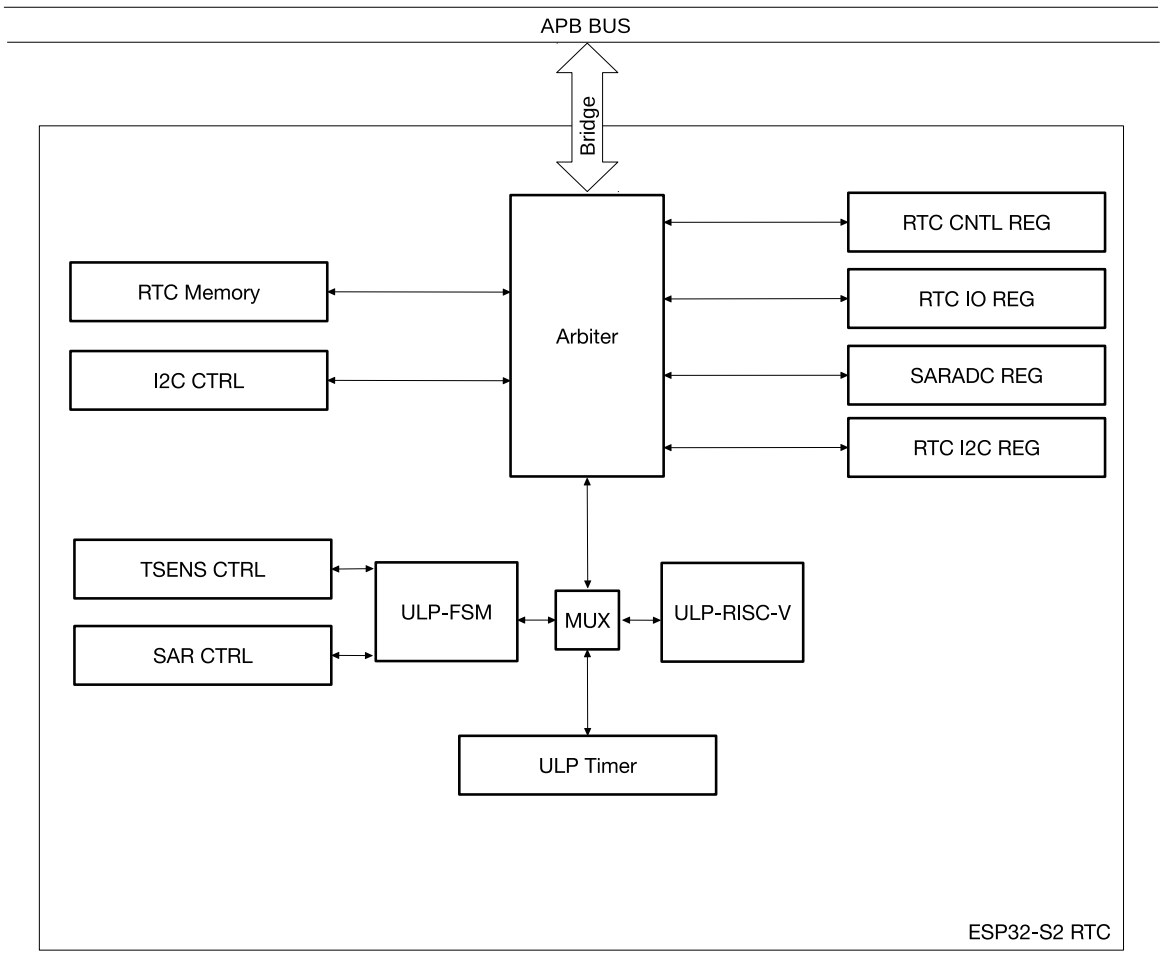


Figure 27-2. ULP Coprocessor Diagram

27.3 Programming Workflow

The ULP-RISC-V is intended for programming using C language. The program in C is then compiled to RV32IMC standard instruction code. The ULP-FSM is using custom instructions normally not supported by high-level programming language. Users develop their programs using ULP-FSM instructions (see Section 27.5.2).

27.4 ULP Coprocessor Workflow

ULP coprocessor is designed to operate independently of the CPU, while the CPU is either in sleep or running.

In a typical power-saving scenario, the chip goes to Deep-sleep mode to lower power consumption. Before setting the chip to sleep mode, users should complete the following operations.

1. Flash the program to be executed by ULP coprocessor into RTC slow memory.
2. Select the working ULP coprocessor by configuring the register RTC_CNTL_COCPU_SEL.
   - 0: select ULP-RISC-V
   - 1: select ULP-FSM
3. Set sleep cycles for the timer by configuring RTC_CNTL_ULP_CP_TIMER_1_REG.

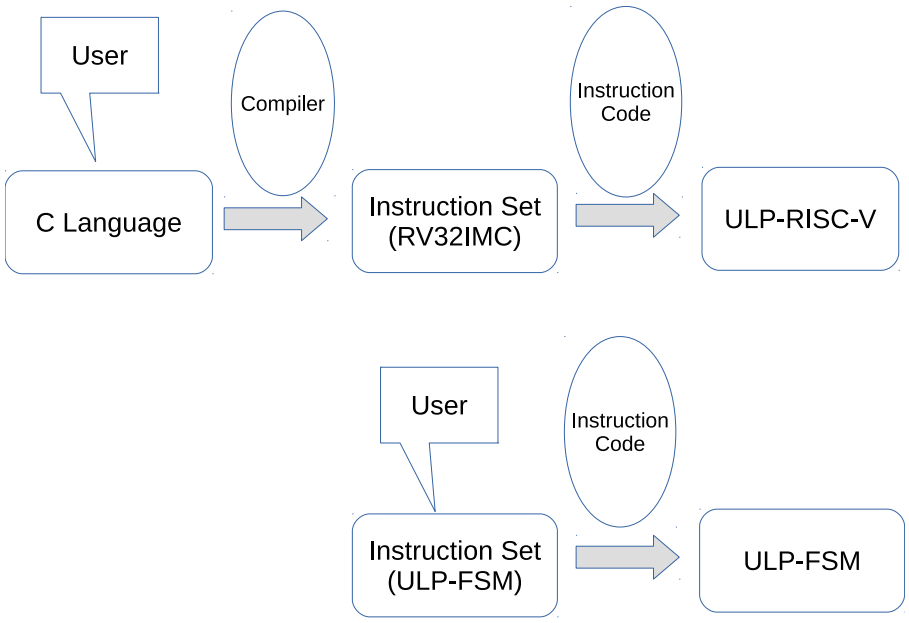


Figure 27-3. Programing Workflow

4. Enable the timer by software or by RTC GPIO;
   - By software: set the register RTC_CNTL_ULP_CP_SLP_TIMER_EN.
   - By RTC GPIO: set the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA.
5. Set the system into sleep mode.

When the system is in Deep-sleep mode:

1. The timer periodically sets the low-power controller (see Chapter 28 *Low-power Management*) to Monitor mode and then wakes up the coprocessor.
2. Coprocessor executes some necessary operations, such as monitoring external environment via low-power sensors.
3. After the operations are finished, the system goes back to Deep-sleep mode.
4. ULP coprocessor goes back to halt mode and waits for next wakeup.

In monitor mode, ULP coprocessor is woken up and goes to halt as shown in Figure 27-4.

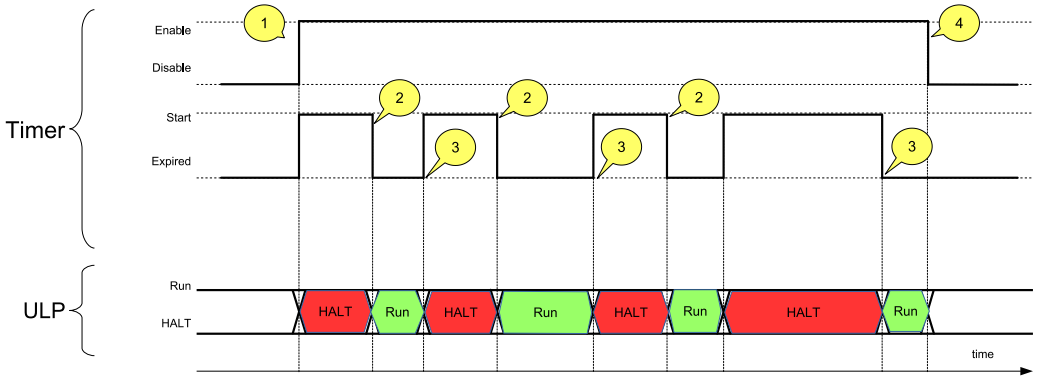


Figure 27-4. Sample of a ULP Operation Sequence

1. Enable the timer and the timer starts counting.

2. The timer expires and wakes up the ULP coprocessor. ULP coprocessor starts running and executes the program flashed in RTC slow memory.
3. ULP coprocessor goes to halt and the timer starts counting again.
    - Put ULP-RISC-V into HALT: set the register RTC_CNTL_COCPU_DONE,
    - Put ULP-FSM into HALT: execute HALT instruction.
4. Disable the timer by ULP program or by software. The system exits from monitor mode.
    - Disabled by software: clear the register RTC_CNTL_ULP_CP_SLP_TIMER_EN.
    - Disabled by RTC GPIO: clear the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA, and set the register RTC_CNTL_ULP_CP_GPIO_WAKEUP_CLR.

Note:

- If the timer is enabled by software (RTC GPIO), it should be disabled by software (RTC GPIO).
- Before setting ULP-RISC-V to HALT, users should configure the register RTC_CNTL_COCPU_DONE first, therefore, it is recommended to end the flashed program with the following pattern:
    - Set the register RTC_CNTL_COCPU_DONE to end the operation of ULP-RISC-V and put it into halt;
    - Set the register RTC_CNTL_COCPU_SHUT_RESET_EN to reset ULP-RISC-V.

  Enough time is reserved for the ULP-RISC-V to complete the operations above before it goes to halt.

The relationship between the signals and registers is shown in Figure 27-5.

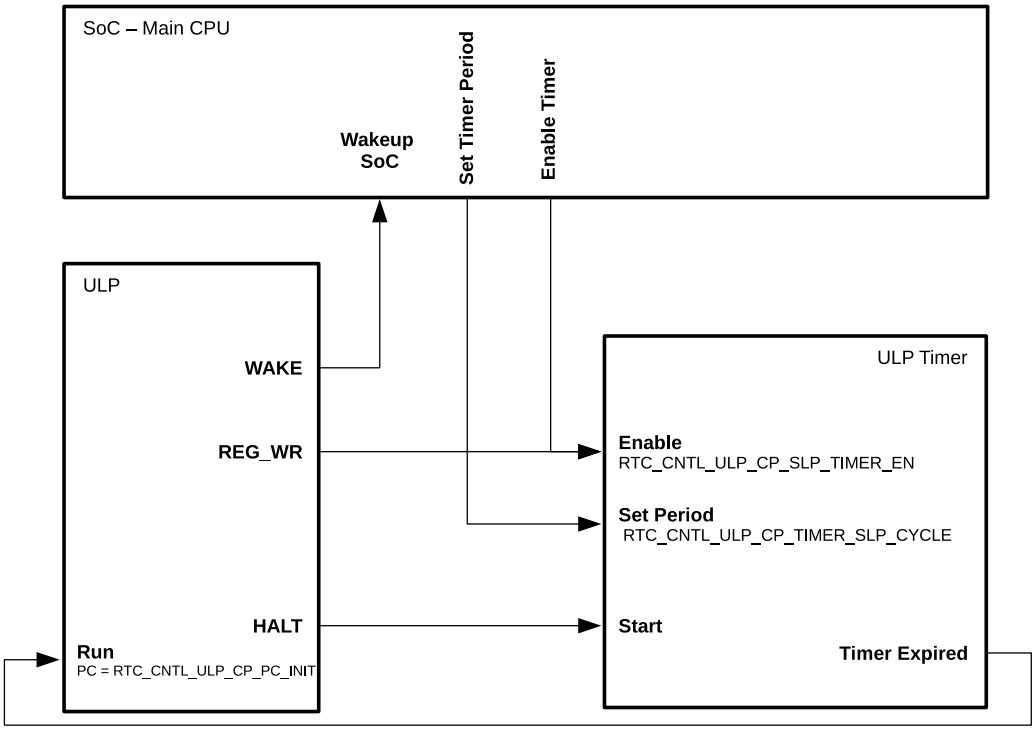


Figure 27-5. Control of ULP Program Execution

27.5 ULP-FSM

27.5.1 Features

ULP-FSM is a programmable finite state machine that can work while the main CPU is in Deep-sleep. ULP-FSM supports instructions for complex logic and arithmetic operations, and also provides dedicated instructions for RTC controllers or peripherals. ULP-FSM can access up to 8 KB of SRAM RTC slow memory (accessible by the CPU) for instructions and data. Hence, such memory is usually used to store instructions and share data between the ULP coprocessor and the CPU. ULP-FSM can be stopped by running HALT instruction.

ULP-FSM has the following features.

- Provide four 16-bit general-purpose registers (R0, R1, R2, and R3) for manipulating data and accessing memory.
- Provide one 8-bit stage count register (Stage_cnt) which can be manipulated by ALU and used in JUMP instructions.
- Support built-in instructions specially for direct control of low-power peripherals, such as SAR ADC and temperature sensor.

27.5.2 Instruction Set

ULP-FSM supports the following instructions.

- ALU: perform arithmetic and logic operations
- LD, ST, REG_RD and REG_WR: load and store data
- JUMP: jump to a certain address
- WAIT/HALT: manage program execution

- WAKE: wake up CPU or communicate with the CPU
- TSENS and ADC: take measurements

The format of ULP-FSM instructions is shown in Figure 27-6.

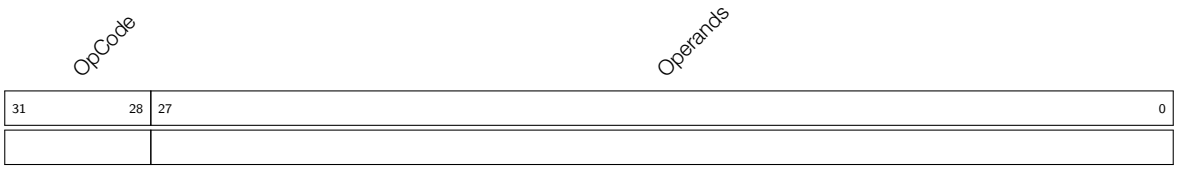


Figure 27-6. ULP-FSM Instruction Format

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

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

27.5.2.1 ALU - Perform Arithmetic and Logic Operations

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

- Arithmetic: ADD and SUB
- Logic: AND and OR
- Bit shifting: LSH and RSH
- Moving data to register: MOVE
- PC register operations - STAGE_RST, STAGE_INC, and STAGE_DEC

The ALU instruction, which has one OpCode (7), can perform various arithmetic and logic operations, depending on the setting of the instruction bits [27:21].

Operations Among Registers

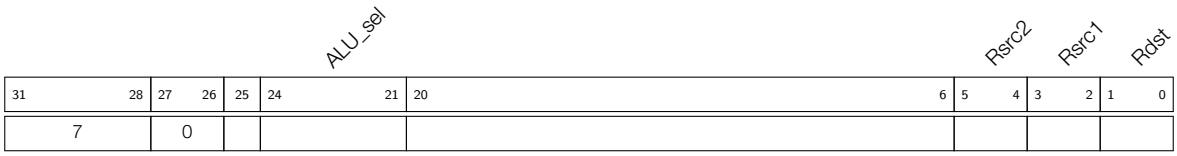


Figure 27-7. Instruction Type — ALU for Operations Among Registers

When bits [27:26] of the instruction in Figure 27-7 are set to 0, ALU performs operations on the data stored in ULP-FSM registers R[0-3]. The types of operations depend on the setting of the instruction bits ALU_sel [24:21] presented in Table 137.

Operand	**Description** - see Figure 27-7
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Rsrc2	Register R[0-3], source
ALU_sel	ALU Operation

ALU_sel	Instruction	Operation	Description
0	ADD	*Rdst = Rsrc1 + Rsrc2*	Add to register
1	SUB	*Rdst = Rsrc1 - Rsrc2*	Subtract from register
2	AND	*Rdst = Rsrc1 & Rsrc2*	Logical AND of two operands
3	OR	*Rdst = Rsrc1 | Rsrc2*	Logical OR of two operands
4	MOVE	*Rdst = Rsrc1*	Move to register
5	LSH	*Rdst = Rsrc1 << Rsrc2*	Logical shift left
6	RSH	*Rdst = Rsrc1 >> Rsrc2*	Logical shift right

Table 137: ALU Operations Among Registers

Note:

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

Operations with Immediate Value

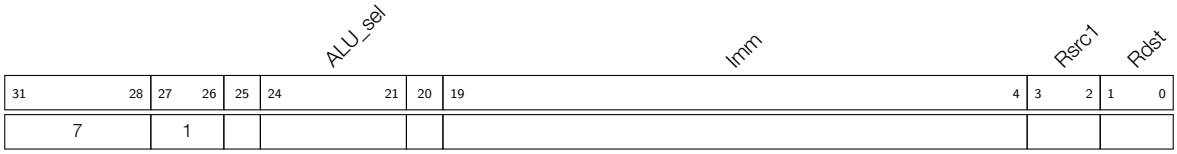


Figure 27-8. Instruction Type — ALU for Operations with Immediate Value

When bits [27:26] of the instruction in Figure 27-8 are set to 1, ALU performs operations using register R[0-3] and the immediate value stored in instruction bits [19:4]. The types of operations depend on the setting of the instruction bits ALU_sel[24:21] presented in Table 138.

Operand	**Description** - see Figure 27-8
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Imm	16-bit signed immediate value
ALU_sel	ALU Operation

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

Table 138: ALU Operations with Immediate Value

Note:

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

Operations with Stage Count Register

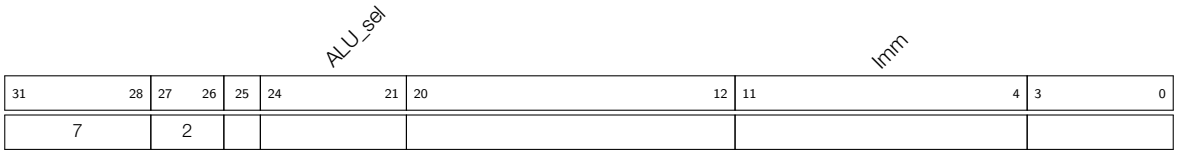


Figure 27-9. Instruction Type — ALU for Operations with Stage Count Register

ALU is also able to increment or decrement by a given value, or reset the 8-bit register Stage_cnt. To do so, bits [27:26] of instruction in Figure 27-9 should be set to 2. The type of operation depends on the setting of the instruction bits ALU_sel[24:21] presented in Table 27-9. The Stage_cnt is a separate register and is not a part of the instruction in Figure 27-9.

Operand	**Description** - see Figure 27-9
Imm	8-bit signed immediate value
ALU_sel	ALU Operation
Stage_cnt	Stage count register, a 8-bit separate register used to store variables, such as loop index

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

Table 139: ALU Operations with Stage Count Register

Note: This instruction is mainly used with JUMPS instruction based on the stage count register to form a stage count for-loop. For the usage, please refer to the following pseudocode:

STAGE_RST // clear stage count register
STAGE_INC // stage count register ++
{...} // loop body, containing n instructions
JUMPS (step = n, cond = 0, threshold = m) // If the value of stage count register is less than m, then jump to STAGE_INC, otherwise jump out of the loop. By such way, a cumulative for-loop with threshold m is implemented.

27.5.2.2 ST – Store Data in Memory

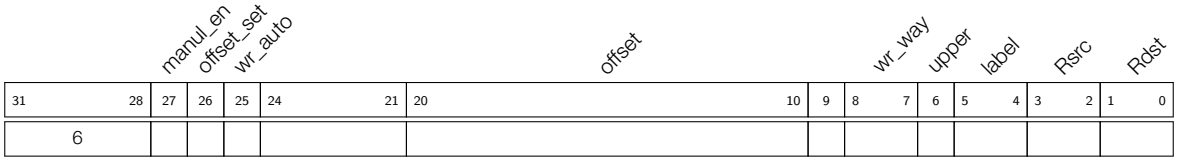


Figure 27-10. Instruction Type - ST

Operand	**Description** - see Figure 27-10
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
upper	0: write the low half-word; 1: write the high half-word
wr_way	0: write the full-word; 1: with the label; 3: without the label
offset	11-bit signed value, expressed in 32-bit words
wr_auto	Enable automatic storage mode
offset_set	Offset enable bit. 0: Do not configure the offset for automatic storage mode. 1: Configure the offset for automatic storage mode.
manul_en	Enable manual storage mode

Automatic Storage Mode

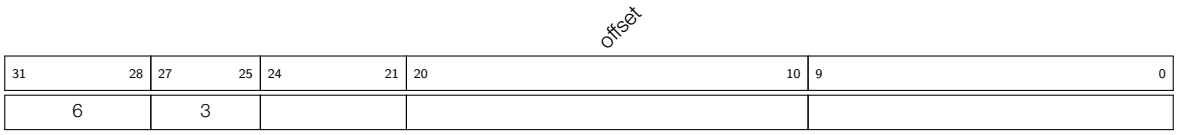


Figure 27-11. Instruction Type - Offset in Automatic Storage Mode (ST-OFFSET)

Operand	**Description** - see Figure 27-11
offset	Initial address offset, 11-bit signed value, expressed in 32-bit words

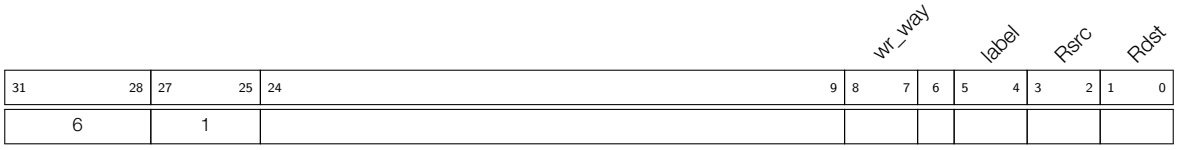


Figure 27-12. Instruction Type - Data Storage in Automatic Storage Mode (ST-AUTO-DATA)

Operand	**Description** - See Figure 27-12
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
wr_way	0: write the fullword; 1: with the label; 3: without the label

Description

This mode is used to access continuous addresses. Before using this mode for the first time, please configure the initial address using ST-OFFSET instruction. Executing the instruction ST-AUTO-DATA will store the 16-bit data in *Rsrc* into the memory address *Rdst* + *Offset*, see Table 140. Write_cnt here indicates the times of the instruction ST-AUTO-DATA executed.

wr_way	write_cnt	Store Data	Operation
0	*	Mem [Rdst + Offset]{31:0} ={PC[10:0], 5'b0, Rsrc[15:0]}	Write full-word, including the pointer and the data
1	odd	Mem [Rdst + Offset]{15:0} = {Label[1:0],Rscr[13:0]}	Store the data with label in the low half-word
1	even	Mem [Rdst + Offset]{31:16} = {Label[1:0],Rscr[13:0]}	Store the data with label in the high half-word
3	odd	Mem [Rdst + Offset]{15:0} = Rscr[15:0]}	Store the data without label in the low half-word
3	even	Mem [Rdst + Offset]{31:16} = Rscr[15:0]	Store the data without label in the high half-word

Table 140: Data Storage Type - Automatic Storage Mode

Note:

- When full-word is written, the offset will be automatically incremented by 1 after each ST-AUTO-DATA execution.
- When half-word is written (low half-word first), the offset will be automatically incremented by 1 after twice ST-AUTO-DATA execution.
- This instruction can only access 32-bit memory words.
- The "Mem" written is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPU.

Manual Storage Mode

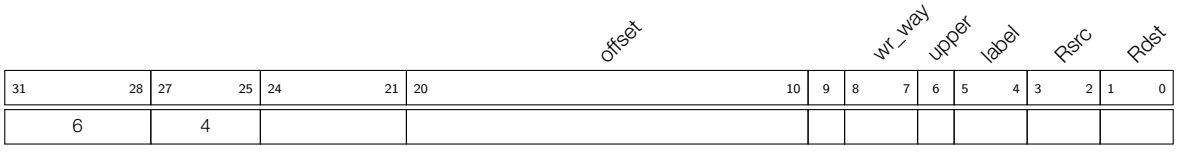


Figure 27-13. Instruction Type - Data Storage in Manual Storage Mode

Operand	Description - See Figure 27-13
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
label	Data label, 2-bit user defined unsigned value
upper	0: Write the low half-word; 1: write the high half-word
wr_way	0: Write the full-word; 1: with the label; 3: without the label
offset	11-bit signed value, expressed in 32-bit words

Description

Manual storage mode is mainly used for storing data into discontinuous addresses. Each instruction needs a storage address and offset. The detailed storage methods are shown in Table 141.

wr_way	upper	Data	Operation
0	*	Mem [Rdst + Offset]{31:0} ={PC[10:0], 5'b0, Rsrc[15:0]}	Write full-word, including the pointer and the data
1	0	Mem [Rdst + Offset]{15:0} = {Label[1:0],Rscr[13:0]}	Store the data with label in the low half-word
1	1	Mem [Rdst + Offset]{31:16} = {Label[1:0],Rscr[13:0]}	Store the data with label in the high half-word
3	0	Mem [Rdst + Offset]{15:0} = Rsrc[15:0]	Store the data without label in the low half-word
3	1	Mem [Rdst + Offset]{31:16} = Rsrc[15:0]	Store the data without label in the high half-word

Table 141: Data Storage - Manual Storage Mode

27.5.2.3 LD – Load Data from Memory

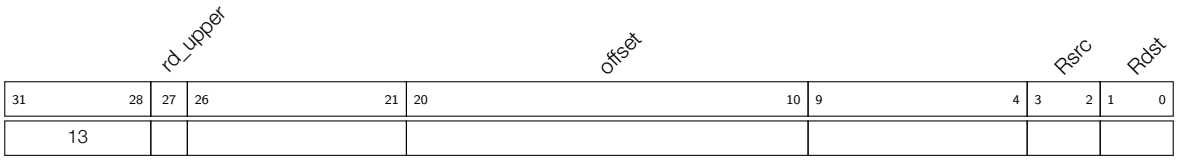


Figure 27-14. Instruction Type - LD

Operand	Description - see Figure 27-14
Rdst	Register R[0-3], destination
Rsrc	Register R[0-3], address of destination memory, expressed in 32-bit words
Offset	11-bit signed value, expressed in 32-bit words
rd_upper	Choose which half-word to read: 0 - read the high half-word 1 - read the low half-word

Description

This instruction loads the low or high 16-bit half-word, depending on *rd_upper*, from memory with address *Rsrc* + *offset* into the destination register *Rdst*:

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

Note:

- This instruction can only access 32-bit memory words.
- The "Mem" loaded is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP coprocessor, corresponds to address 0x50000000, as seen by the main CPU.

27.5.2.4 JUMP – Jump to an Absolute Address

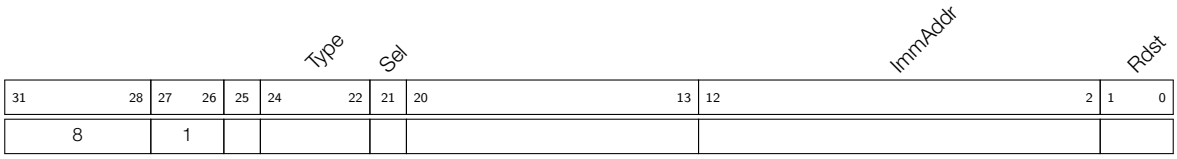


Figure 27-15. Instruction Type- JUMP

Operand	**Description** - see Figure 27-15
Rdst	Register R[0-3], containing address to jump to (expressed in 32-bit words)
ImmAddr	11-bit address, expressed in 32-bit words
Sel	Select the address to jump to:
	0 - jump to the address stored in *ImmAddr*
	1 - jump to the address stored in *Rdst*
Type	Jump type:
	0 - make an unconditional jump
	1 - jump only if the last ALU operation has set zero flag
	2 - jump only if the last ALU operation has set overflow flag

Note:

All jump addresses are expressed in 32-bit words.

Description

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

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

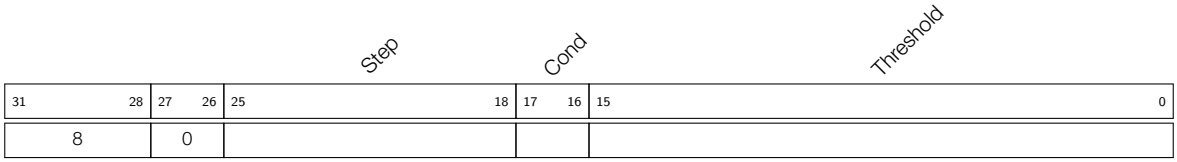


Figure 27-16. Instruction Type - JUMPR

Operand	**Description** - see Figure 27-16
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition to jump: 0 - jump if R0 < *Threshold* 1 - jump if R0 > *Threshold* 2 - jump if R0 = *Threshold*
Step	Relative shift from current position, expressed in 32-bit words: if Step[7] = 0, then PC = PC + Step[6:0] if Step[7] = 1, then PC = PC - Step[6:0]

Note:

All jump addresses are expressed in 32-bit words.

Description

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

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

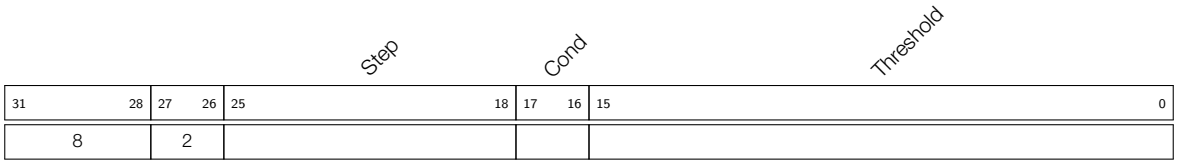


Figure 27-17. Instruction Type - JUMPS

Operand	**Description** - see Figure 27-17
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition to jump: 1X - jump if *Stage_cnt* <= *Threshold* 00 - jump if *Stage_cnt* < *Threshold* 01 - jump if *Stage_cnt* >= *Threshold*
Step	Relative shift from current position, expressed in 32-bit words: if Step[7] = 0, then PC = PC + Step[6:0] if Step[7] = 1, then PC = PC - Step[6:0]

Note:

- For more information about the stage count register, please refer to Section 27.5.2.1.
- All jump addresses are expresses in 32-bit words.

Description

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

27.5.2.7 HALT – End the Program

31 28	27 0
11	

Figure 27-18. Instruction Type- HALT

Description

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

Note:

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

27.5.2.8 WAKE – Wake up the Chip

31 28	27 26	25 1	0
9	0		1'b1

Figure 27-19. Instruction Type - WAKE

Description

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

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

27.5.2.9 WAIT – Wait for a Number of Cycles

		Cycles
31 28	27 16	15 0
4		

Figure 27-20. Instruction Type - WAIT

Operand	**Description** - see Figure 27-20
Cycles	The number of cycles to wait

Description

The instruction will delay the ULP-FSM for a given number of cycles.

27.5.2.10 TSENS – Take Measurement with Temperature Sensor

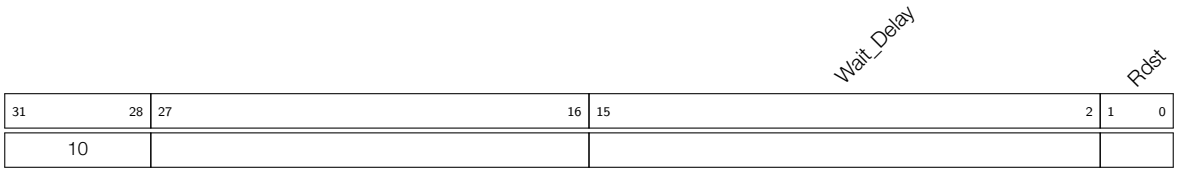


Figure 27-21. Instruction Type - TSENS

Operand	**Description** - see Figure 27-21
Rdst	Destination Register R[0-3], results will be stored in this register.
Wait_Delay	Number of cycles used to perform the measurement.

Description

Increasing the measurement cycles *Wait_Delay* helps improve the accuracy and optimize the result. The instruction performs measurement via temperature sensor and stores the result into a general purpose register.

27.5.2.11 ADC – Take Measurement with ADC

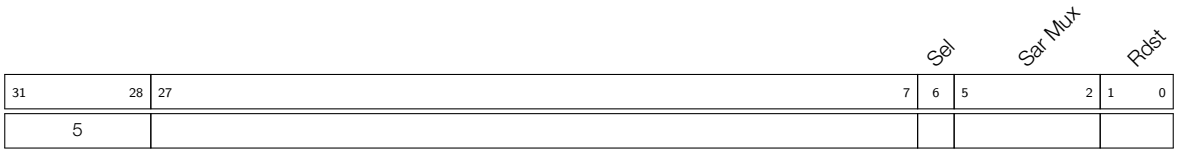


Figure 27-22. Instruction Type - ADC

Operand	**Description** - see Figure 27-22
Rdst	Destination Register R[0-3], results will be stored in this register.
Sar_Mux	Enable SAR ADC pad [Sar_Mux - 1], see Table 142.
Sel	Select ADC. 0: select SAR ADC1; 1: select SAR ADC2, see Table 142.

Table 142: Input Signals Measured Using the ADC Instruction

Pad / Signal / GPIO	*Sar_Mux*	ADC Selection *Sel*
GPIO1	1	*Sel* = 0, select SAR ADC1
GPIO2	2	
GPIO3	3	
GPIO4	4	
GPIO5	5	
GPIO6	6	
GPIO7	7	
GPIO8	8	
GPIO9	9	
GPIO10	10	
GPIO11	1	*Sel* = 1, select SAR ADC2
GPIO12	2	

Pad / Signal / GPIO	*Sar_Mux*	ADC Selection *Sel*
GPIO13	3	*Sel* = 1, select SAR ADC2
GPIO14	4	
XTAL_32k_P	5	
XTAL_32k_N	6	
DAC_1	7	
DAC_2	8	
GPIO19	9	
GPIO20	10	

27.5.2.12 REG_RD – Read from Peripheral Register

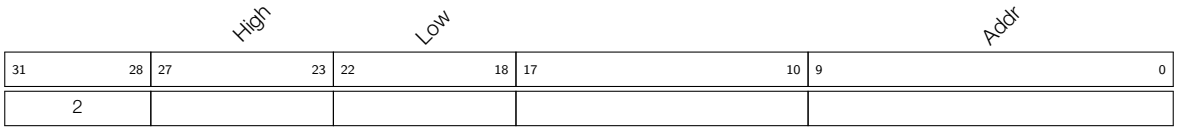


Figure 27-23. Instruction Type - REG_RD

Operand	**Description** - see Figure 27-23
Addr	Peripheral register address, in 32-bit words
Low	Register start bit number
High	Register end bit number

Description

The instruction reads up to 16 bits from a peripheral register into a general-purpose register:

R0 = REG[Addr][High:Low]

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

Note:

- This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the register, as seen from the ULP coprocessor, can be calculated from the address of the same register on PeriBUS1 (addr_peribus1), as follows:

  addr_ulp = (addr_peribus1 - DR_REG_RTCCNTL_BASE) / 4

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

27.5.2.13 REG_WR – Write to Peripheral Register

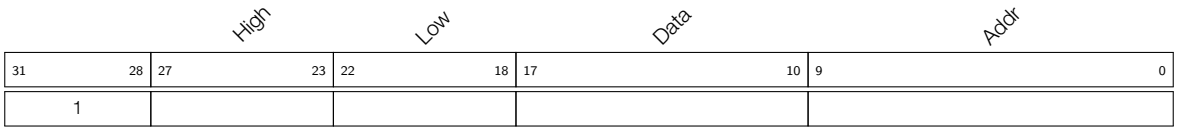


Figure 27-24. Instruction Type - REG_WR

Operand	**Description** - see Figure 27-24
Addr	Register address, expressed in 32-bit words
Data	Value to write, 8 bits
Low	Register start bit number
High	Register end bit number

Description

This instruction writes up to 8 bits from an immediate data value into a peripheral register.

REG[Addr][High:Low] = Data

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

Note:

See notes regarding *addr_ulp* in Section 27.5.2.12.

27.6 ULP-RISC-V

27.6.1 Features

- Support RV32IMC instruction set
- Thirty-two 32-bit general-purpose registers
- 32-bit multiplier and divider
- Support for interrupts

27.6.2 Multiplier and Divider

ULP-RISC-V has an independent multiplication and division unit. The efficiency of multiplication and division instructions is shown in the following table.

Table 143: Instruction Efficiency

Operation	Instruction	Execution Cycle	Instruction Description
Multiply	MUL	34	Multiply two 32-bit integers and return the lower 32-bit of the result
	MULH	66	Multiply two 32-bit signed integers and return the higher 32-bit of the result
	MULHU	66	Multiply two 32-bit unsigned integers and return the higher 32-bit of the result
	MULHSU	66	Multiply a 32-bit signed integer with a unsigned integer and return the higher 32-bit of the result
Divide	DIV	34	Divide a 32-bit integer by a 32-bit integer and return the quotient
	DIVU	34	Divide a 32-bit unsigned integer by a 32-bit unsigned integer and return the quotient
	REM	34	Divide a 32-bit signed integer by a 32-bit signed integer and return the remainder
	REMU	34	Divide a 32-bit unsigned integer by a 32-bit unsigned integer and return the remainder

27.6.3 ULP-RISC-V Interrupts

The interrupts from some sensors, software, and RTC I²C can be routed to ULP-RISC-V. To enable the interrupts, please set the register SENS_SAR_COCPU_INT_ENA_REG, see Table 144.

Enable bit	Interrupt	Description
0	TOUCH_DONE_INT	Triggered when the touch sensor completes the scan of a channel
1	TOUCH_INACTIVE_INT	Triggered when the touch pad is released
2	TOUCH_ACTIVE_INT	Triggered when the touch pad is touched
3	SARADC1_DONE_INT	Triggered when SAR ADC1 completes the conversion one time
4	SARADC2_DONE_INT	Triggered when SAR ADC2 completes the conversion one time
5	TSENS_DONE_INT	Triggered when the temperature sensor completes the dump of its data
6	RISCV_START_INT	Triggered when ULP-RISC-V powers on and starts working
7	SW_INT	Triggered by software
8	SWD_INT	Triggered by timeout of Super Watchdog (SWD)

Table 144: ULP-RISC-V Interrupt List

Note:

- Besides the above-mentioned interrupts, ULP-RISC-V can also handle the interrupt from RTC_IO by simply configuring RTC_IO as input mode. Users can configure RTCIO_GPIO_PIN*n*_INT_TYPE to select the interrupt trigger modes, but only level trigger modes are available. For more details about RTC_IO configuration, see Chapter IO MUX and GPIO Matrix.
- The interrupt from RTC_IO can be cleared by releasing RTC_IO and its source can be read from the register RTCIO_RTC_GPIO_STATUS_REG.
- The SW_INT interrupt is generated by configuring the register RTC_CNTL_COCPU_SW_INT_TRIGGER.
- For the information about RTC I²C interrupts, please refer to Section 27.7.4.

27.7 RTC I²C Controller

ULP coprocessor can use RTC I²C controller to read from or write to the external I²C slave devices.

27.7.1 Connecting RTC I²C Signals

SDA and SCL signals can be mapped onto two out of the four GPIO pins, which are identified in Table 24 in Chapter 5 *IO MUX and GPIO Matrix*, using the register RTCIO_SAR_I2C_IO_REG.

27.7.2 Configuring RTC I²C

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

1. Set the low and high SCL half-periods by configuring RTC_I2C_SCL_LOW_PERIOD_REG and RTC_I2C_SCL_HIGH_PERIOD_REG in RTC_FAST_CLK cycles (e.g. RTC_I2C_SCL_LOW_PERIOD_REG = 40, RTC_I2C_SCL_HIGH_PERIOD_REG = 40 for 100 kHz frequency).
2. Set the number of cycles between the SDA switch and the falling edge of SCL by using RTC_I2C_SDA_DUTY_REG in RTC_FAST_CLK (e.g. RTC_I2C_SDA_DUTY_REG = 16).
3. Set the waiting time after the START condition by using RTC_I2C_SCL_START_PERIOD_REG (e.g. RTC_I2C_SCL_START_PERIOD = 30).
4. Set the waiting time before the END condition by using RTC_I2C_SCL_STOP_PERIOD_REG (e.g. RTC_I2C_SCL_STOP_PERIOD = 44).
5. Set the transaction timeout by using RTC_I2C_TIME_OUT_REG (e.g. RTC_I2C_TIME_OUT_REG = 200).
6. Configure the RTC I²C controller into master mode by setting the RTC_I2C_MS_MODE bit in RTC_I2C_CTRL_REG.
7. Write the address(es) of external slave(s) to SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7). Up to eight slave addresses can be pre-programmed this way. One of these addresses can then be selected for each transaction as part of the RTC I²C instruction.

Once RTC I²C is configured, the main CPU or the ULP coprocessor can communicate with the external I²C devices.

27.7.3 Using RTC I²C

27.7.3.1 Instruction Format

The format of RTC I²C instruction is consistent with that of I²C0/I²C1, see Section 17.3.2.2 *CMD_Controller* in Chapter 17 *I²C Controller*. The only difference is that RTC I²C provides fixed instructions for different operations, as follows:

- Command 0 ~ Command 1: specially for I²C write operation
- Command 2 ~ Command 6: specially for I²C read operation

Note: All slave addresses are expressed in 7 bits.

27.7.3.2 I2C_RD - I²C Read Workflow

Preparation for RTC I²C read:

- Configure the instruction list of RTC I²C (see Section CMD_Controller in Chapter 17 *I²C Controller*), including instruction order, instruction code, read data number (byte_num), and other information.
- Configure the slave register address by setting the register SENS_SAR_I2C_CTRL[18:11].
- Start RTC I²C transmission by setting the registers SENS_SAR_I2C_START_FORCE and SENS_SAR_I2C_START.
- When an RTC_I2C_RX_DATA_INT interrupt is received, transfer the read data stored in RTC_I2C_RDATA to SRAM RTC slow memory, or use the data directly.

The I2C_RD instruction performs the following operations (see Figure 27-25):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 ("write"). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*.
3. Slave generates ACK.
4. Master sends slave register address.
5. Slave generates ACK.
6. Master generates a repeated START (RSTART) condition.
7. Master sends slave address, with r/w bit set to 1 ("read").
8. Slave sends one byte of data.
9. Master checks whether the number of transmitted bytes reaches the number set by the current instruction (byte_num). If yes, master jumps out of the read instruction and sends an NACK signal. Otherwise master repeats Step 8 and waits for the slave to send the next byte.
10. Master generates a STOP condition and stops reading.

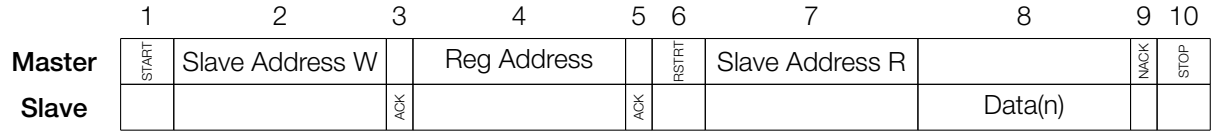


Figure 27-25. I²C Read Operation

Note:

The RTC I²C peripheral samples the SDA signals on the falling edge of SCL. If the slave changes SDA in less than 0.38 microsecond, the master may receive incorrect data.

27.7.3.3 I2C_WR - I²C Write Workflow

Preparation for RTC I²C write:

- Configure RTC I²C instruction list, including instruction order, instruction code, and the data to be written in byte (byte_num). See the configuration of I²C0/I²C1 in Section CMD_Controller in Chapter 17 *I²C Controller*.
- Configure the slave register address by setting the register SENS_SAR_I2C_CTRL[18:11], and the data to be transmitted in SENS_SAR_I2C_CTRL[26:19].
- Set the registers SENS_SAR_I2C_START_FORCE and SENS_SAR_I2C_START to start the transmission.

- Update the next data to be transmitted in the register SENS_SAR_I2C_CTRL[26:19], each time when an RTC_I2C_TX_DATA_INT interrupt is received.

The I2C_WR instruction performs the following operations, see Figure 27-26.

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 ("write"). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*.
3. Slave generates ACK.
4. Master sends slave register address.
5. Slave generates ACK.
6. Master generates a repeated START condition (RSTART).
7. Master sends slave address, with r/w bit set to 0 ("write").
8. Master sends one byte of data.
9. Slave generates ACK. Master checks whether the number of transmitted bytes reaches the number set by the current instruction (byte_num). If yes, master jumps out of the write instruction and starts the next instruction. Otherwise the master repeats step 8 and sends the next byte.
10. Master generates a STOP condition and stops the transmission.

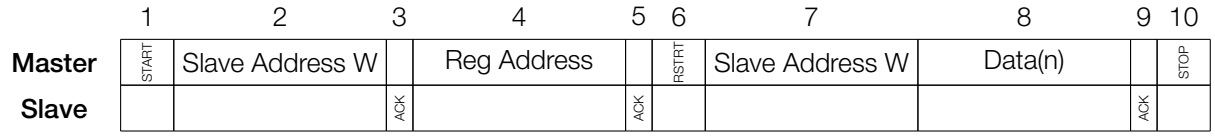


Figure 27-26. I²C Write Operation

27.7.3.4 Detecting Error Conditions

Applications can query specific bits in the RTC_I2C_INT_ST_REG register to check if the transaction is successful. To enable checking for specific communication events, their corresponding bits should be set in register RTC_I2C_INT_ENA_REG. Note that the bit map is shifted by 1. If a specific communication event is detected and its corresponding bit in register RTC_I2C_INT_ST_REG is set, the event can then be cleared using register RTC_I2C_INT_CLR_REG.

27.7.4 RTC I²C Interrupts

- RTC_I2C_SLAVE_TRAN_COMP_INT: Triggered when the slave finishes the transaction.
- RTC_I2C_ARBITRATION_LOST_INT: Triggered when the master loses control of the bus.
- RTC_I2C_MASTER_TRAN_COMP_INT: Triggered when the master completes the transaction.
- RTC_I2C_TRANS_COMPLETE_INT: Triggered when a STOP signal is detected.
- RTC_I2C_TIME_OUT_INT: Triggered by time out event.
- RTC_I2C_ACK_ERR_INT: Triggered by ACK error.
- RTC_I2C_RX_DATA_INT: Triggered when data is received.

- RTC_I2C_TX_DATA_INT: Triggered when data is transmitted.
- RTC_I2C_DETECT_START_INT: Triggered when a START signal is detected.

27.8 Base Address

27.8.1 ULP Coprocessor Base Address

Users can access ULP coprocessors with the following base addresses as shown in Table 145.

Table 145: ULP Coprocessor Base Address

Module	Bus to Access Peripheral	Base Address
ULP (ALWAYS_ON)	PeriBUS1	0x3F408000
	PeriBUS2	0x60008000
ULP (RTC_PERI)	PeriBUS1	0x3F408800
	PeriBUS2	0x60008800

Wherein:

- ULP (ALWAYS_ON) represents the registers, which will not be reset due to the power down of RTC_PERI domain. See Chapter Low Power Management.
- ULP (RTC_PERI) represents the registers in RTC_PERI domain, which will be reset due to the power down of RTC_PERI domain. See Chapter Low Power Management.

27.8.2 RTC I²C Base Address

Users can access the RTC I²C registers in RTC_PERI domain, including RTC_PERI registers and I²C registers, with the following base addresses as shown in Table 146.

Table 146: RTC I²C Base Address

Module	Bus to Access Peripheral	Base Address
RTC I²C (RTC_PERI)	PeriBUS1	0x3F408800
	PeriBUS2	0x60008800
RTC I²C (I²C)	PeriBUS1	0x3F408C00
	PeriBUS2	0x60008C00

27.9 Register Summary

The address in the following part represents the address offset (relative address) with respect to the base address, not the absolute address. For detailed information about the base address, please refer to Section 27.8.

27.9.1 ULP (ALWAYS_ON) Register Summary

Name	Description	Address	Access
ULP Timer Registers			
RTC_CNTL_ULP_CP_TIMER_REG	Configure the timer	0x00F8	varies
RTC_CNTL_ULP_CP_TIMER_1_REG	Configure sleep cycle of the timer	0x0130	R/W
ULP-FSM Register			
RTC_CNTL_ULP_CP_CTRL_REG	ULP-FSM configuration register	0x00FC	R/W

Name	Description	Address	Access
ULP-RISC-V Register			
RTC_CNTL_COCPU_CTRL_REG	ULP-RISC-V configuration register	0x0100	varies

27.9.2 ULP (RTC_PERI) Register Summary

Name	Description	Address	Access
ULP-RISC-V Registers			
SENS_ SAR_COCPU_INT_RAW_REG	Interrupt raw bit of ULP-RISC-V	0x0128	RO
SENS_SAR_COCPU_INT_ENA_REG	Interrupt enable bit of ULP-RISC-V	0x012C	R/W
SENS_SAR_COCPU_INT_ST_REG	Interrupt status bit of ULP-RISC-V	0x0130	RO
SENS_SAR_COCPU_INT_CLR_REG	Interrupt clear bit of ULP-RISC-V	0x0134	WO

27.9.3 RTC I²C (RTC_PERI) Register Summary

Name	Description	Address	Access
RTC I²C Controller Register			
SENS_SAR_I2C_CTRL_REG	Configure RTC I²C transmission	0x0058	R/W
RTC I²C Slave Address Registers			
SENS_SAR_SLAVE_ADDR1_REG	Configure slave addresses 0-1 of RTC I²C	0x0040	R/W
SENS_SAR_SLAVE_ADDR2_REG	Configure slave addresses 2-3 of RTC I²C	0x0044	R/W
SENS_SAR_SLAVE_ADDR3_REG	Configure slave addresses 4-5 of RTC I²C	0x0048	R/W
SENS_SAR_SLAVE_ADDR4_REG	Configure slave addresses 6-7 of RTC I²C	0x004C	R/W

27.9.4 RTC I²C (I²C) Register Summary

Name	Description	Address	Access
RTC I²C Signal Setting Registers			
RTC_I2C_SCL_LOW_REG	Configure the low level width of SCL	0x0000	R/W
RTC_I2C_SCL_HIGH_REG	Configure the high level width of SCL	0x0014	R/W
RTC_I2C_SDA_DUTY_REG	Configure the SDA hold time after a negative SCL edge	0x0018	R/W
RTC_I2C_SCL_START_PERIOD_REG	Configure the delay between the SDA and SCL negative edge for a start condition	0x001C	R/W
RTC_I2C_SCL_STOP_PERIOD_REG	Configure the delay between SDA and SCL positive edge for a stop condition	0x0020	R/W
RTC I²C Control Registers			
RTC_I2C_CTRL_REG	Transmission setting	0x0004	R/W
RTC_I2C_STATUS_REG	RTC I²C status	0x0008	RO
RTC_I2C_TO_REG	Configure RTC I²C timeout	0x000C	R/W
RTC_I2C_SLAVE_ADDR_REG	Configure slave address	0x0010	R/W
RTC I²C Interrupt Registers			
RTC_I2C_INT_CLR_REG	Clear RTC I²C interrupt	0x0024	WO
RTC_I2C_INT_RAW_REG	RTC I²C raw interrupt	0x0028	RO
RTC_I2C_INT_ST_REG	RTC I²C interrupt status	0x002C	RO

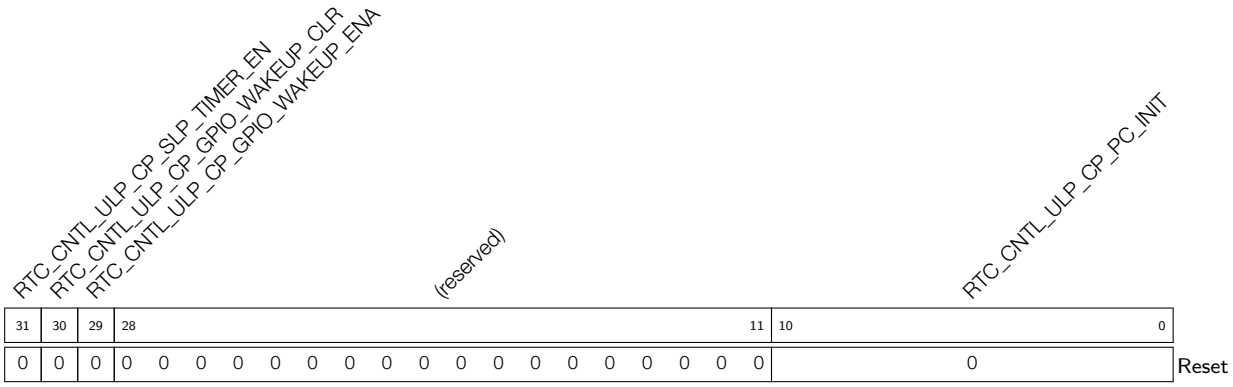
Name	Description	Address	Access
RTC_I2C_INT_ENA_REG	Enable RTC I²C interrupt	0x0030	R/W
RTC I²C Status Register			
RTC_I2C_DATA_REG	RTC I²C read data	0x0034	varies
RTC I²C Command Registers			
RTC_I2C_CMD0_REG	RTC I²C Command 0	0x0038	varies
RTC_I2C_CMD1_REG	RTC I²C Command 1	0x003C	varies
RTC_I2C_CMD2_REG	RTC I²C Command 2	0x0040	varies
RTC_I2C_CMD3_REG	RTC I²C Command 3	0x0044	varies
RTC_I2C_CMD4_REG	RTC I²C Command 4	0x0048	varies
RTC_I2C_CMD5_REG	RTC I²C Command 5	0x004C	varies
RTC_I2C_CMD6_REG	RTC I²C Command 6	0x0050	varies
RTC_I2C_CMD7_REG	RTC I²C Command 7	0x0054	varies
RTC_I2C_CMD8_REG	RTC I²C Command 8	0x0058	varies
RTC_I2C_CMD9_REG	RTC I²C Command 9	0x005C	varies
RTC_I2C_CMD10_REG	RTC I²C Command 10	0x0060	varies
RTC_I2C_CMD11_REG	RTC I²C Command 11	0x0064	varies
RTC_I2C_CMD12_REG	RTC I²C Command 12	0x0068	varies
RTC_I2C_CMD13_REG	RTC I²C Command 13	0x006C	varies
RTC_I2C_CMD14_REG	RTC I²C Command 14	0x0070	varies
RTC_I2C_CMD15_REG	RTC I²C Command 15	0x0074	varies
Version register			
RTC_I2C_DATE_REG	Version control register	0x00FC	R/W

27.10 Registers

The address in the following part represents the address offset (relative address) with respect to the base address, not the absolute address. For detailed information about the base address, please refer to Section 27.8.

27.10.1 ULP (ALWAYS_ON) Registers

Register 27.1: RTC_CNTL_ULP_CP_TIMER_REG (0x00F8)



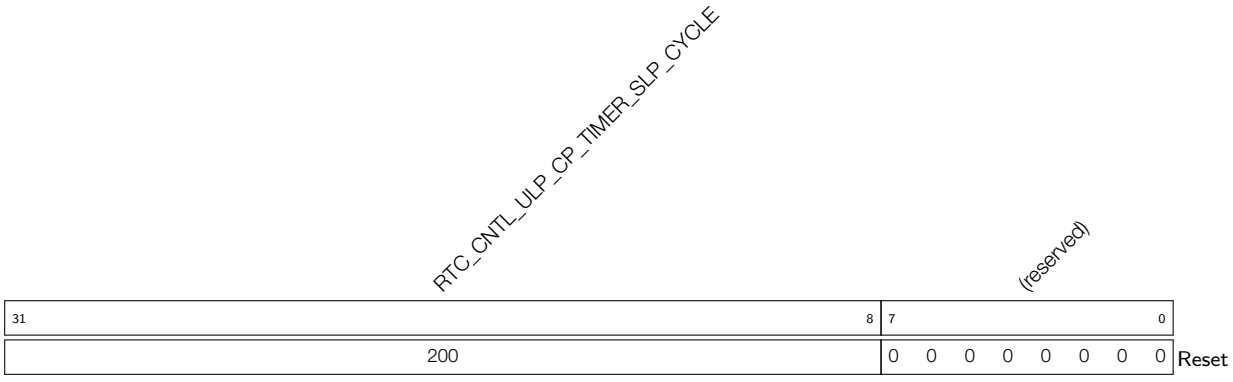
RTC_CNTL_ULP_CP_PC_INIT ULP coprocessor PC initial address. (R/W)

RTC_CNTL_ULP_CP_GPIO_WAKEUP_ENA Enable the option of ULP coprocessor woken up by RTC GPIO. (R/W)

RTC_CNTL_ULP_CP_GPIO_WAKEUP_CLR Disable the option of ULP coprocessor woken up by RTC GPIO. (WO)

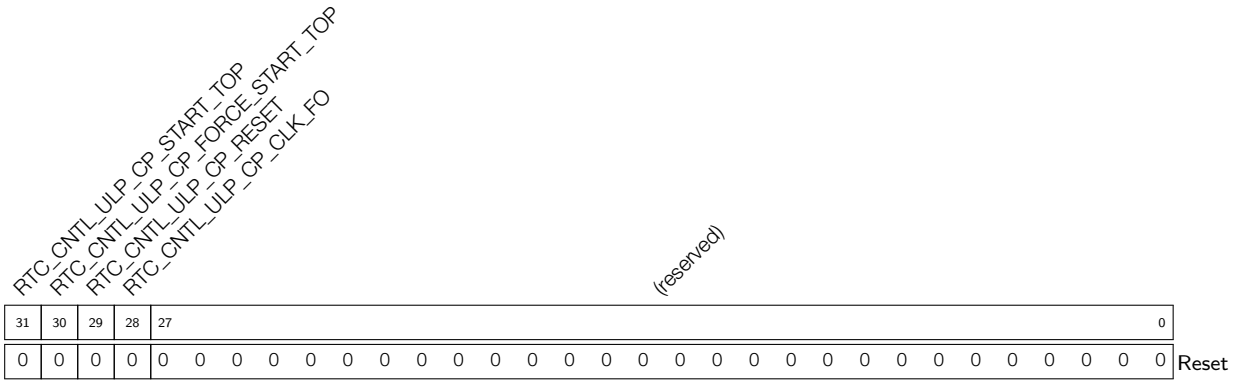
RTC_CNTL_ULP_CP_SLP_TIMER_EN ULP coprocessor timer enable bit. 0: Disable hardware timer; 1: Enable hardware timer. (R/W)

Register 27.2: RTC_CNTL_ULP_CP_TIMER_1_REG (0x0130)



RTC_CNTL_ULP_CP_TIMER_SLP_CYCLE Set sleep cycles for ULP coprocessor timer. (R/W)

Register 27.3: RTC_CNTL_ULP_CP_CTRL_REG (0x00FC)



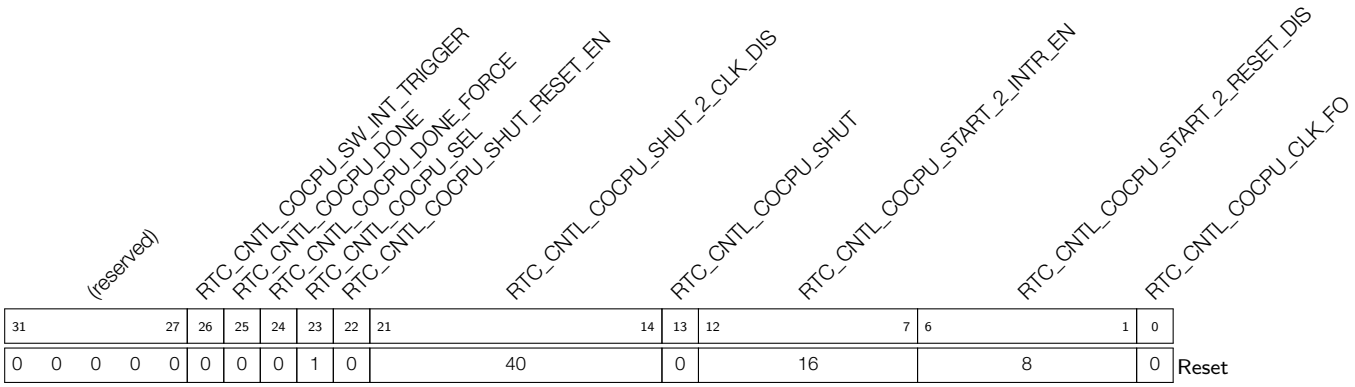
RTC_CNTL_ULP_CP_CLK_FO ULP-FSM clock force on. (R/W)

RTC_CNTL_ULP_CP_RESET ULP-FSM clock software reset. (R/W)

RTC_CNTL_ULP_CP_FORCE_START_TOP Write 1 to start ULP-FSM by software. (R/W)

RTC_CNTL_ULP_CP_START_TOP Write 1 to start ULP-FSM. (R/W)

Register 27.4: RTC_CNTL_COCPU_CTRL_REG (0x0100)



RTC_CNTL_COCPU_CLK_FO ULP-RISC-V clock force on. (R/W)

RTC_CNTL_COCPU_START_2_RESET_DIS Time from ULP-RISC-V startup to pull down reset. (R/W)

RTC_CNTL_COCPU_START_2_INTR_EN Time from ULP-RISC-V startup to send out RISCV_START_INT interrupt. (R/W)

RTC_CNTL_COCPU_SHUT Shut down ULP-RISC-V. (R/W)

RTC_CNTL_COCPU_SHUT_2_CLK_DIS Time from shut down ULP-RISC-V to disable clock. (R/W)

RTC_CNTL_COCPU_SHUT_RESET_EN This bit is used to reset ULP-RISC-V. (R/W)

RTC_CNTL_COCPU_SEL 0: select ULP-RISC-V; 1: select ULP-FSM. (R/W)

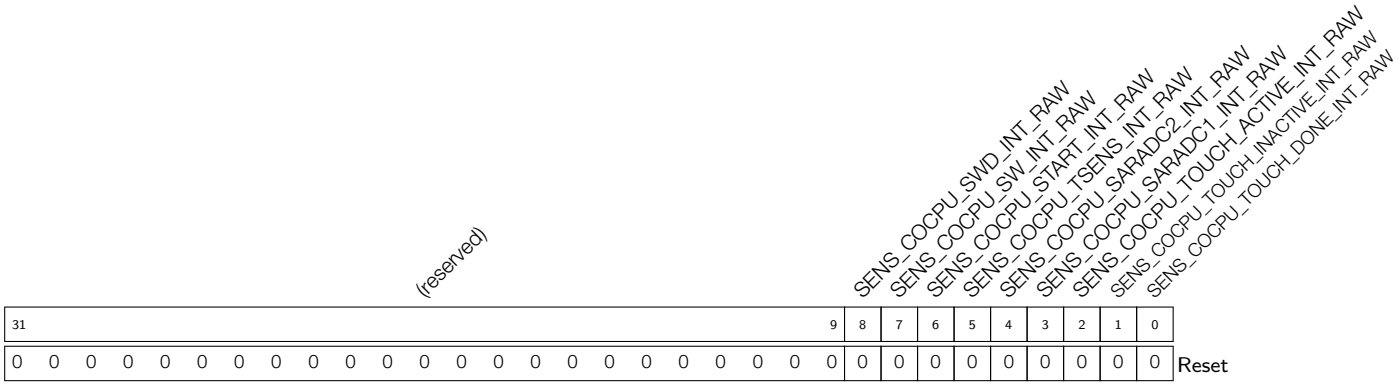
RTC_CNTL_COCPU_DONE_FORCE 0: select ULP-FSM DONE signal; 1: select ULP-RISC-V DONE signal. (R/W)

RTC_CNTL_COCPU_DONE DONE signal. Write 1 to this bit, ULP-RISC-V will go to HALT and the timer starts counting. (R/W)

RTC_CNTL_COCPU_SW_INT_TRIGGER Trigger ULP-RISC-V register interrupt. (WO)

27.10.2 ULP (RTC_PERI) Registers

Register 27.5: SENS_ SAR_COCPU_INT_RAW_REG (0x0128)



SENS_COCPU_TOUCH_DONE_INT_RAW TOUCH_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_TOUCH_INACTIVE_INT_RAW TOUCH_INACTIVE_INT interrupt raw bit. (RO)

SENS_COCPU_TOUCH_ACTIVE_INT_RAW TOUCH_ACTIVE_INT interrupt raw bit. (RO)

SENS_COCPU_SARADC1_INT_RAW SARADC1_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_SARADC2_INT_RAW SARADC2_DONE_INT interrupt raw bit. (RO)

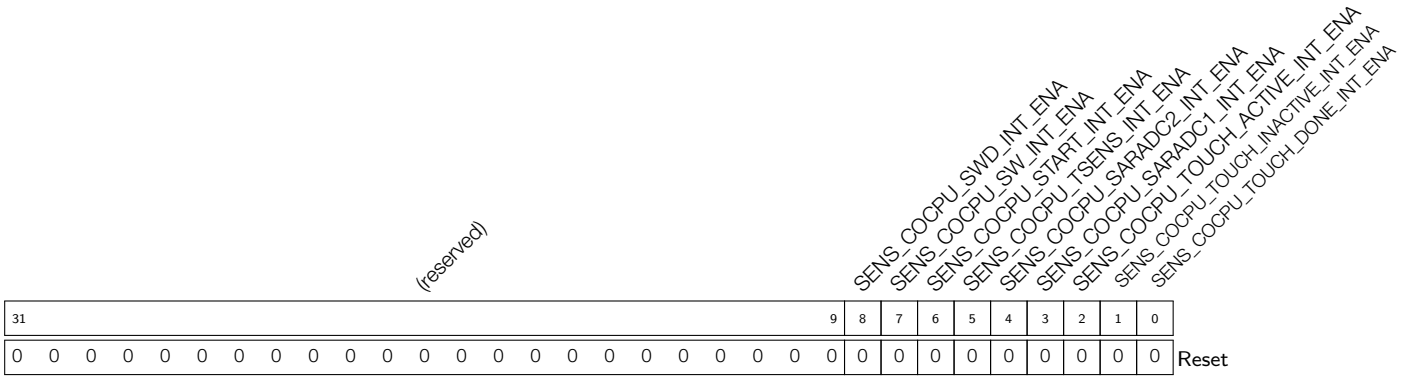
SENS_COCPU_TSENS_INT_RAW TSENS_DONE_INT interrupt raw bit. (RO)

SENS_COCPU_START_INT_RAW RISCV_START_INT interrupt raw bit. (RO)

SENS_COCPU_SW_INT_RAW SW_INT interrupt raw bit. (RO)

SENS_COCPU_SWD_INT_RAW SWD_INT interrupt raw bit. (RO)

Register 27.6: SENS_SAR_COCPU_INT_ENA_REG (0x012C)



SENS_COCPU_TOUCH_DONE_INT_ENA TOUCH_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_TOUCH_INACTIVE_INT_ENA TOUCH_INACTIVE_INT interrupt enable bit. (R/W)

SENS_COCPU_TOUCH_ACTIVE_INT_ENA TOUCH_ACTIVE_INT interrupt enable bit. (R/W)

SENS_COCPU_SARADC1_INT_ENA SARADC1_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_SARADC2_INT_ENA SARADC2_DONE_INT interrupt enable bit. (R/W)

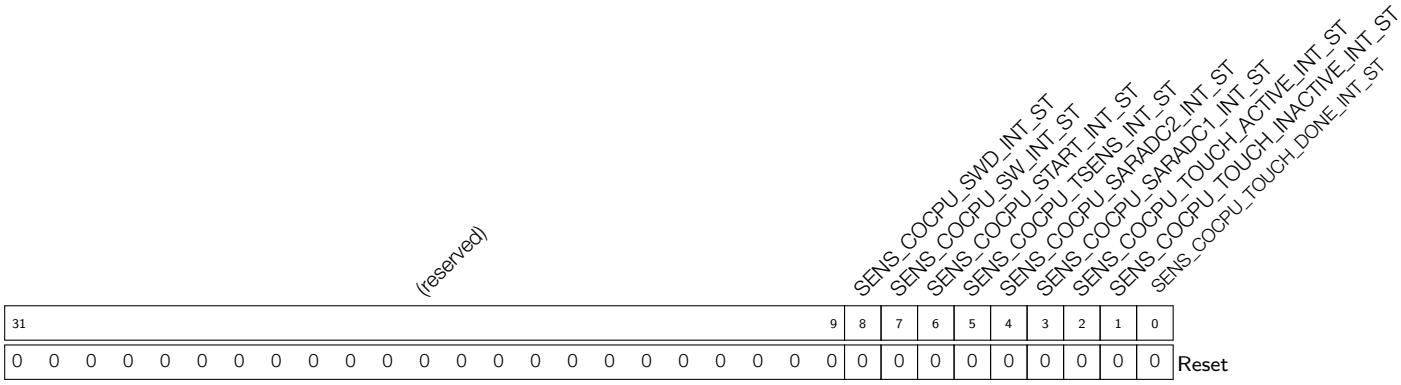
SENS_COCPU_TSENS_INT_ENA TSENS_DONE_INT interrupt enable bit. (R/W)

SENS_COCPU_START_INT_ENA RISCV_START_INT interrupt enable bit. (R/W)

SENS_COCPU_SW_INT_ENA SW_INT interrupt enable bit. (R/W)

SENS_COCPU_SWD_INT_ENA SWD_INT interrupt enable bit. (R/W)

Register 27.7: SENS_SAR_COCPU_INT_ST_REG (0x0130)



SENS_COCPU_TOUCH_DONE_INT_ST TOUCH_DONE_INT interrupt status bit. (RO)

SENS_COCPU_TOUCH_INACTIVE_INT_ST TOUCH_INACTIVE_INT interrupt status bit. (RO)

SENS_COCPU_TOUCH_ACTIVE_INT_ST TOUCH_ACTIVE_INT interrupt status bit. (RO)

SENS_COCPU_SARADC1_INT_ST SARADC1_DONE_INT interrupt status bit. (RO)

SENS_COCPU_SARADC2_INT_ST SARADC2_DONE_INT interrupt status bit. (RO)

SENS_COCPU_TSENS_INT_ST TSENS_DONE_INT interrupt status bit. (RO)

SENS_COCPU_START_INT_ST RISCV_START_INT interrupt status bit. (RO)

SENS_COCPU_SW_INT_ST SW_INT interrupt status bit. (RO)

SENS_COCPU_SWD_INT_ST SWD_INT interrupt status bit. (RO)

Register 27.8: SENS_SAR_COCPU_INT_CLR_REG (0x0134)

(reserved)	SENS_COCPU_SWD_INT_CLR	SENS_COCPU_SW_INT_CLR	SENS_COCPU_START_INT_CLR	SENS_COCPU_TSENS_INT_CLR	SENS_COCPU_SARADC2_INT_CLR	SENS_COCPU_SARADC1_INT_CLR	SENS_COCPU_TOUCH_ACTIVE_INT_CLR	SENS_COCPU_TOUCH_INACTIVE_INT_CLR	SENS_COCPU_TOUCH_DONE_INT_CLR	
31 … 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SENS_COCPU_TOUCH_DONE_INT_CLR TOUCH_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_TOUCH_INACTIVE_INT_CLR TOUCH_INACTIVE_INT interrupt clear bit. (WO)

SENS_COCPU_TOUCH_ACTIVE_INT_CLR TOUCH_ACTIVE_INT interrupt clear bit. (WO)

SENS_COCPU_SARADC1_INT_CLR SARADC1_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_SARADC2_INT_CLR SARADC2_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_TSENS_INT_CLR TSENS_DONE_INT interrupt clear bit. (WO)

SENS_COCPU_START_INT_CLR RISCV_START_INT interrupt clear bit. (WO)

SENS_COCPU_SW_INT_CLR SW_INT interrupt clear bit. (WO)

SENS_COCPU_SWD_INT_CLR SWD_INT interrupt clear bit. (WO)

27.10.3 RTC I2C (RTC_PERI) Registers

Register 27.9: SENS_SAR_I2C_CTRL_REG (0x0058)

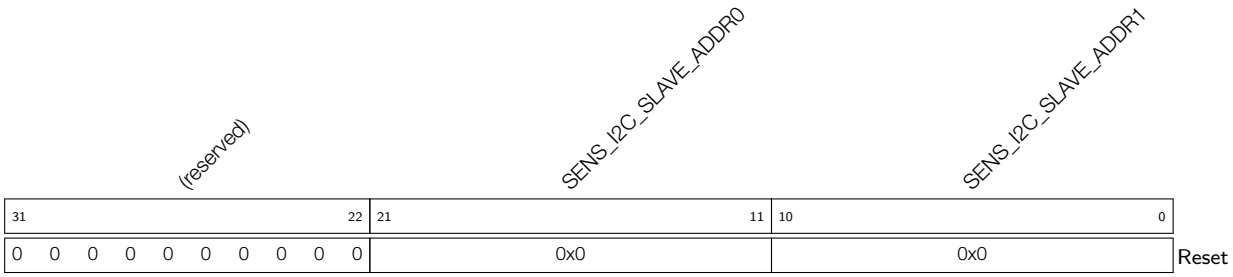
(reserved)	SENS_SAR_I2C_START_FORCE	SENS_SAR_I2C_START	SENS_SAR_I2C_CTRL	
31 30	29	28	27 … 0	
0 0	0	0	0	Reset

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

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

SENS_SAR_I2C_START_FORCE 0: RTC I²C started by FSM; 1: RTC I²C started by software. (R/W)

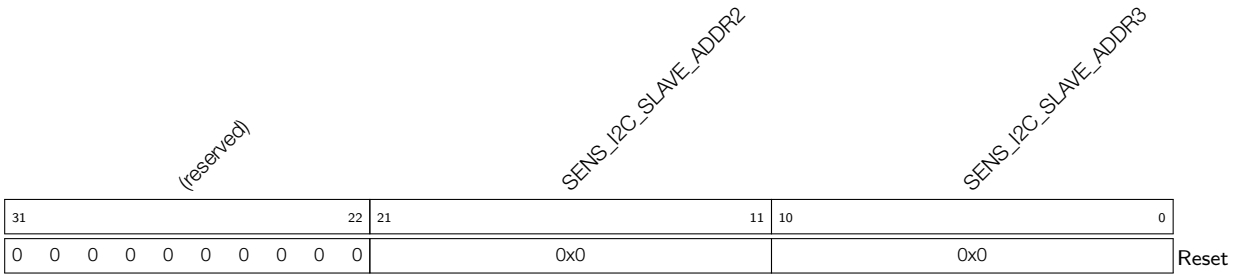
Register 27.10: SENS_SAR_SLAVE_ADDR1_REG (0x0040)



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

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

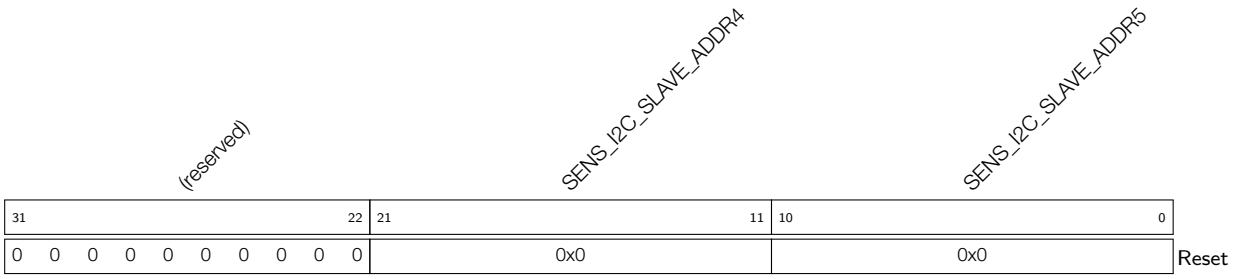
Register 27.11: SENS_SAR_SLAVE_ADDR2_REG (0x0044)



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

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

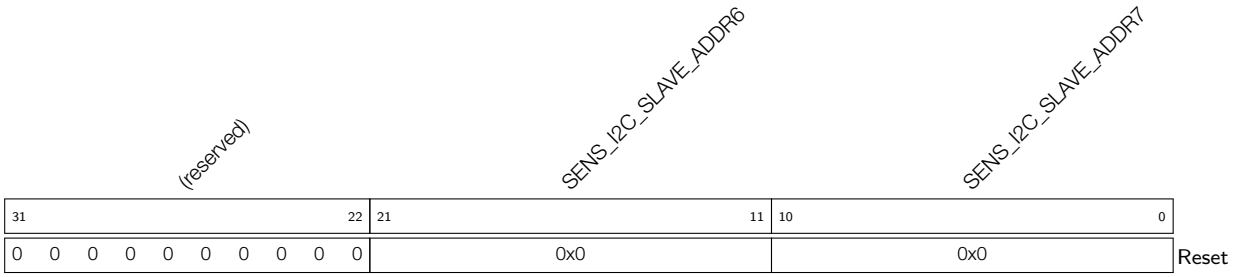
Register 27.12: SENS_SAR_SLAVE_ADDR3_REG (0x0048)



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

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

Register 27.13: SENS_SAR_SLAVE_ADDR4_REG (0x004C)

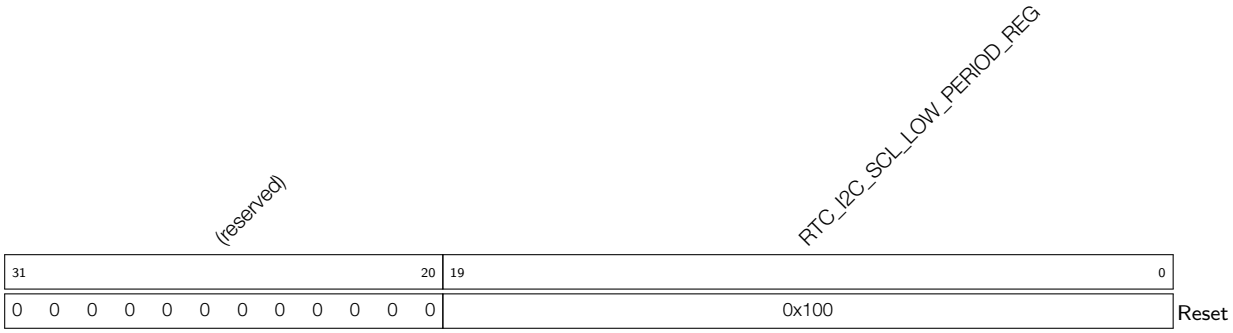


SENS_I2C_SLAVE_ADDR7 RTC I²C slave address 7. (R/W)

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

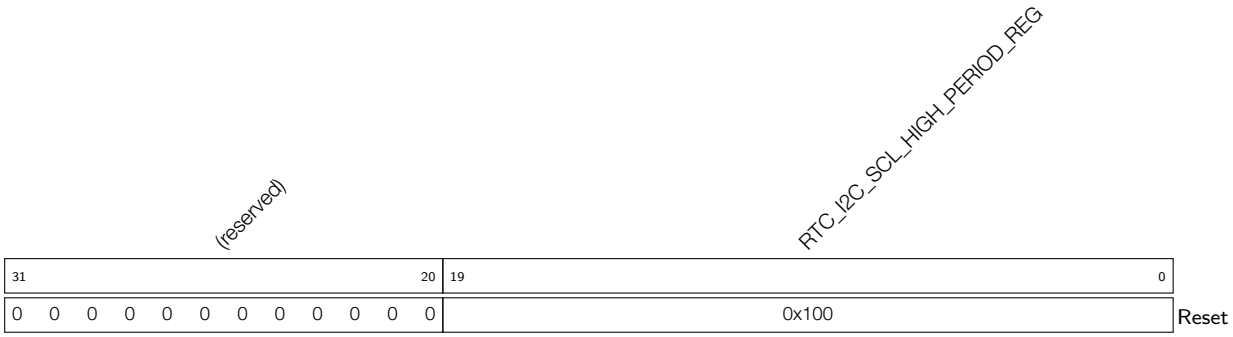
27.10.4 RTC I²C (I²C) Registers

Register 27.14: RTC_I2C_SCL_LOW_REG (0x0000)



RTC_I2C_SCL_LOW_PERIOD_REG This register is used to configure how many clock cycles SCL remains low. (R/W)

Register 27.15: RTC_I2C_SCL_HIGH_REG (0x0014)

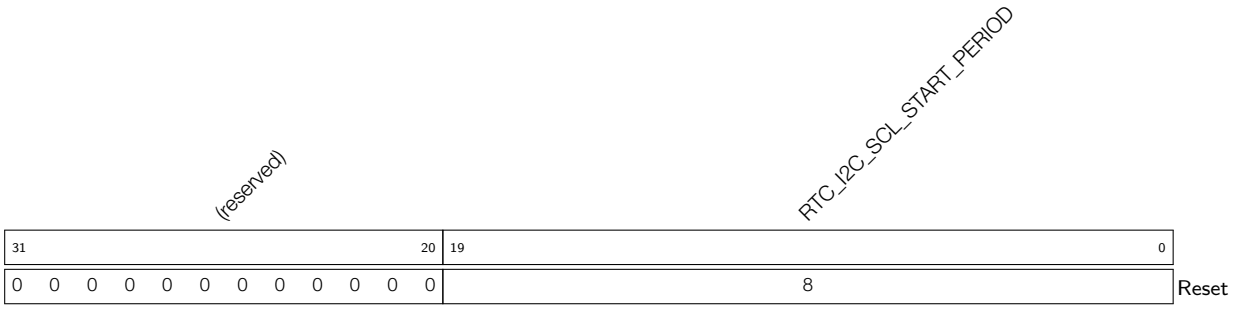


RTC_I2C_SCL_HIGH_PERIOD_REG This register is used to configure how many cycles SCL remains high. (R/W)

Register 27.16: RTC_I2C_SDA_DUTY_REG (0x0018)

RTC_I2C_SDA_DUTY_NUM The number of clock cycles between the SDA switch and the falling edge of SCL. (R/W)

Register 27.17: RTC_I2C_SCL_START_PERIOD_REG (0x001C)



RTC_I2C_SCL_START_PERIOD Number of clock cycles to wait after generating a start condition. (R/W)

Register 27.18: RTC_I2C_SCL_STOP_PERIOD_REG (0x0020)

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

Register 27.19: RTC_I2C_CTRL_REG (0x0004)

(reserved)	RTC_I2C_RESET	RTC_I2C_CTRL_CLK_GATE_EN	(reserved)	RTC_I2C_RX_LSB_FIRST	RTC_I2C_TX_LSB_FIRST	RTC_I2C_TRANS_START	RTC_I2C_MS_MODE	RTC_I2C_SCL_FORCE_OUT	RTC_I2C_SDA_FORCE_OUT	
31	30	29	28 … 6	5	4	3	2	1	0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

RTC_I2C_SDA_FORCE_OUT SDA output mode. 0: open drain; 1: push pull. (R/W)

RTC_I2C_SCL_FORCE_OUT SCL output mode. 0: open drain; 1: push pull. (R/W)

RTC_I2C_MS_MODE Set this bit to configure RTC I²C as a master. (R/W)

RTC_I2C_TRANS_START Set this bit to 1, RTC I²C starts sending data. (R/W)

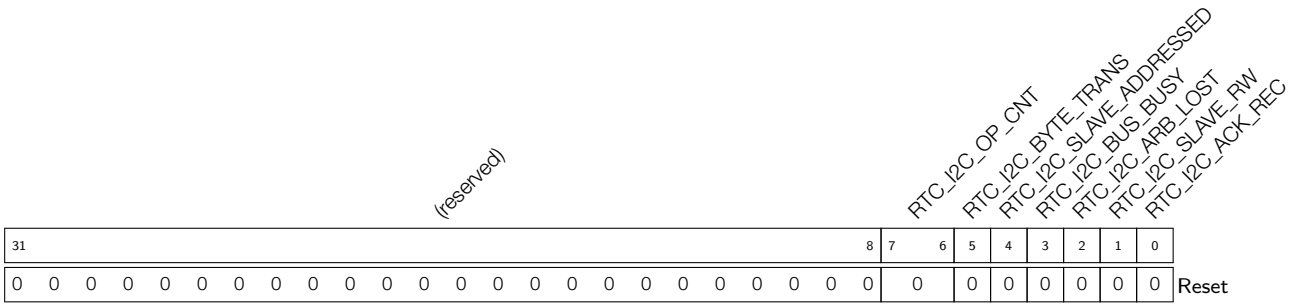
RTC_I2C_TX_LSB_FIRST This bit is used to control the sending mode. 0: send data from the most significant bit; 1: send data from the least significant bit. (R/W)

RTC_I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. 0: receive data from the most significant bit; 1: receive data from the least significant bit. (R/W)

RTC_I2C_CTRL_CLK_GATE_EN RTC I²C controller clock gate. (R/W)

RTC_I2C_RESET RTC I²C software reset. (R/W)

Register 27.20: RTC_I2C_STATUS_REG (0x0008)



RTC_I2C_ACK_REC The received ACK value. 0: ACK; 1: NACK. (RO)

RTC_I2C_SLAVE_RW 0: master writes to slave; 1: master reads from slave. (RO)

RTC_I2C_ARB_LOST When the RTC I²C loses control of SCL line, the register changes to 1. (RO)

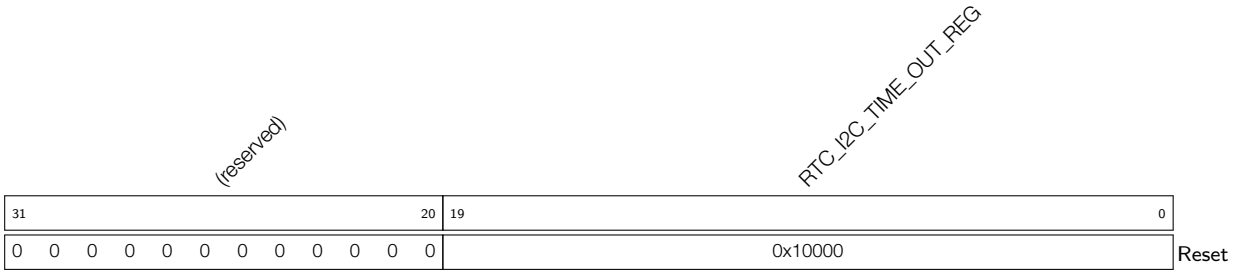
RTC_I2C_BUS_BUSY 0: RTC I²C bus is in idle state; 1: RTC I²C bus is busy transferring data. (RO)

RTC_I2C_SLAVE_ADDRESSED When the address sent by the master matches the address of the slave, then this bit will be set. (RO)

RTC_I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

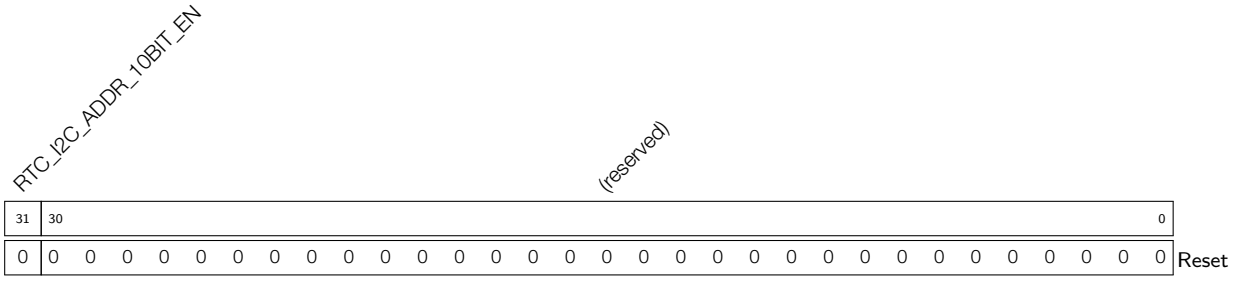
RTC_I2C_OP_CNT Indicate which operation is working. (RO)

Register 27.21: RTC_I2C_TO_REG (0x000C)



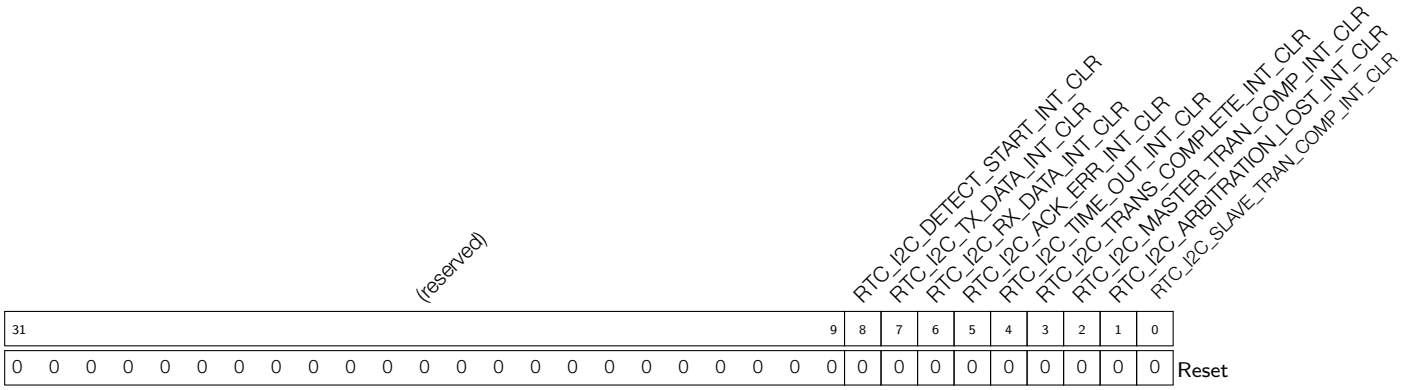
RTC_I2C_TIME_OUT_REG Timeout threshold. (R/W)

Register 27.22: RTC_I2C_SLAVE_ADDR_REG (0x0010)



RTC_I2C_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode. (R/W)

Register 27.23: RTC_I2C_INT_CLR_REG (0x0024)



RTC_I2C_SLAVE_TRAN_COMP_INT_CLR RTC_I2C_SLAVE_TRAN_COMP_INT interrupt clear bit. (WO)

RTC_I2C_ARBITRATION_LOST_INT_CLR RTC_I2C_ARBITRATION_LOST_INT interrupt clear bit. (WO)

RTC_I2C_MASTER_TRAN_COMP_INT_CLR RTC_I2C_MASTER_TRAN_COMP_INT interrupt clear bit. (WO)

RTC_I2C_TRANS_COMPLETE_INT_CLR RTC_I2C_TRANS_COMPLETE_INT interrupt clear bit. (WO)

RTC_I2C_TIME_OUT_INT_CLR RTC_I2C_TIME_OUT_INT interrupt clear bit. (WO)

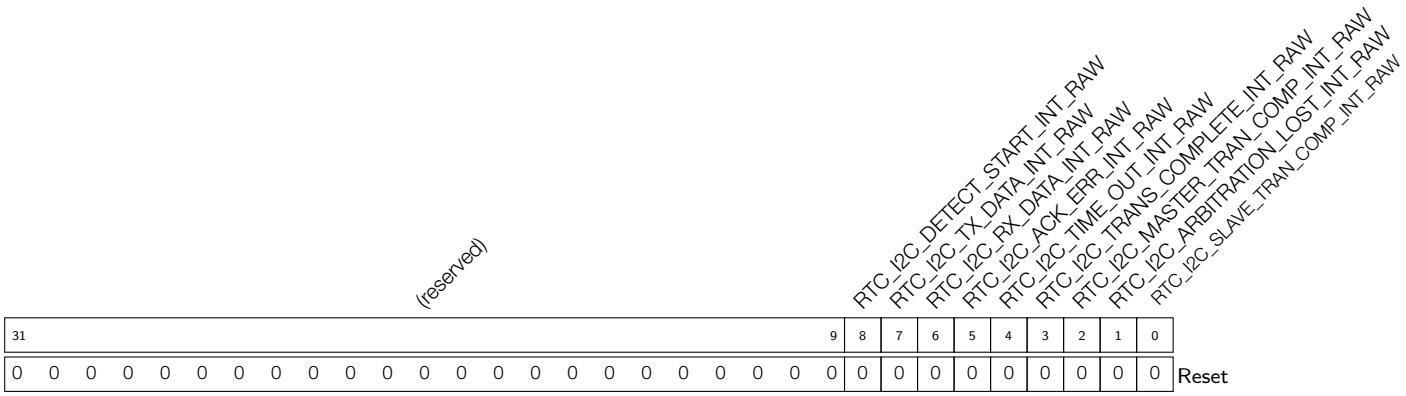
RTC_I2C_ACK_ERR_INT_CLR RTC_I2C_ACK_ERR_INT interrupt clear bit. (WO)

RTC_I2C_RX_DATA_INT_CLR RTC_I2C_RX_DATA_INT interrupt clear bit. (WO)

RTC_I2C_TX_DATA_INT_CLR RTC_I2C_TX_DATA_INT interrupt clear bit. (WO)

RTC_I2C_DETECT_START_INT_CLR RTC_I2C_DETECT_START_INT interrupt clear bit. (WO)

Register 27.24: RTC_I2C_INT_RAW_REG (0x0028)



RTC_I2C_SLAVE_TRAN_COMP_INT_RAW RTC_I2C_SLAVE_TRAN_COMP_INT interrupt raw bit. (RO)

RTC_I2C_ARBITRATION_LOST_INT_RAW RTC_I2C_ARBITRATION_LOST_INT interrupt raw bit. (RO)

RTC_I2C_MASTER_TRAN_COMP_INT_RAW RTC_I2C_MASTER_TRAN_COMP_INT interrupt raw bit. (RO)

RTC_I2C_TRANS_COMPLETE_INT_RAW RTC_I2C_TRANS_COMPLETE_INT interrupt raw bit. (RO)

RTC_I2C_TIME_OUT_INT_RAW RTC_I2C_TIME_OUT_INT interrupt raw bit. (RO)

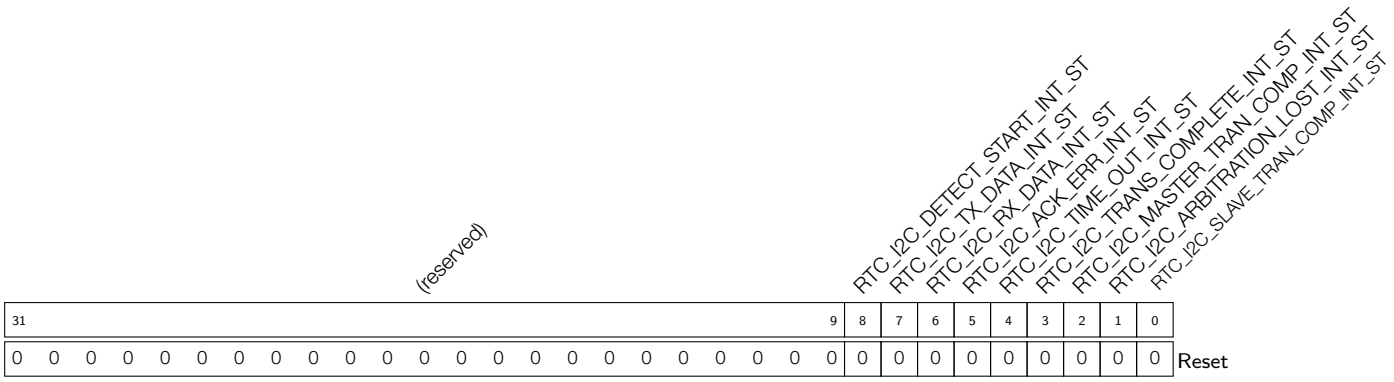
RTC_I2C_ACK_ERR_INT_RAW RTC_I2C_ACK_ERR_INT interrupt raw bit. (RO)

RTC_I2C_RX_DATA_INT_RAW RTC_I2C_RX_DATA_INT interrupt raw bit. (RO)

RTC_I2C_TX_DATA_INT_RAW RTC_I2C_TX_DATA_INT interrupt raw bit. (RO)

RTC_I2C_DETECT_START_INT_RAW RTC_I2C_DETECT_START_INT interrupt raw bit. (RO)

Register 27.25: RTC_I2C_INT_ST_REG (0x002C)



RTC_I2C_SLAVE_TRAN_COMP_INT_ST RTC_I2C_SLAVE_TRAN_COMP_INT interrupt status bit. (RO)

RTC_I2C_ARBITRATION_LOST_INT_ST RTC_I2C_ARBITRATION_LOST_INT interrupt status bit. (RO)

RTC_I2C_MASTER_TRAN_COMP_INT_ST RTC_I2C_MASTER_TRAN_COMP_INT interrupt status bit. (RO)

RTC_I2C_TRANS_COMPLETE_INT_ST RTC_I2C_TRANS_COMPLETE_INT interrupt status bit. (RO)

RTC_I2C_TIME_OUT_INT_ST RTC_I2C_TIME_OUT_INT interrupt status bit. (RO)

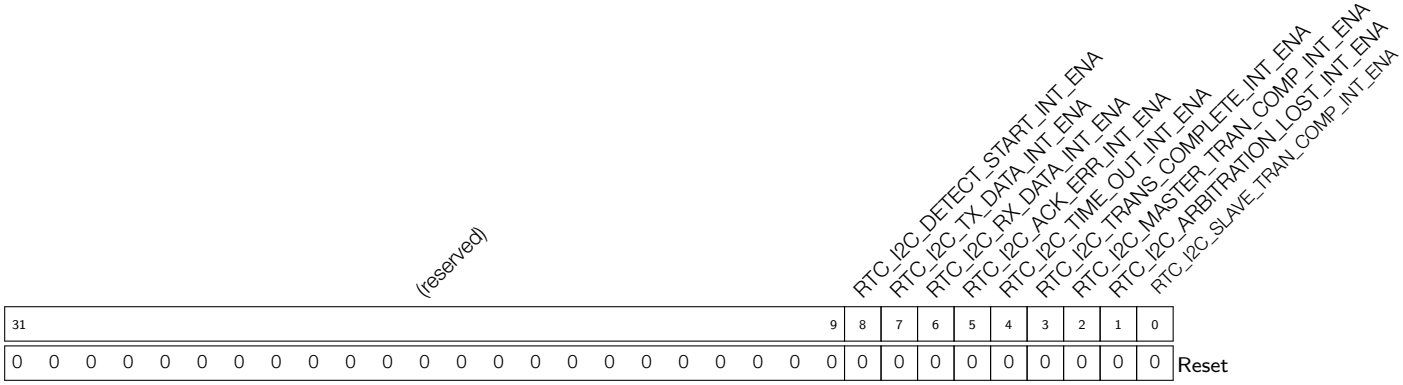
RTC_I2C_ACK_ERR_INT_ST RTC_I2C_ACK_ERR_INT interrupt status bit. (RO)

RTC_I2C_RX_DATA_INT_ST RTC_I2C_RX_DATA_INT interrupt status bit. (RO)

RTC_I2C_TX_DATA_INT_ST RTC_I2C_TX_DATA_INT interrupt status bit. (RO)

RTC_I2C_DETECT_START_INT_ST RTC_I2C_DETECT_START_INT interrupt status bit. (RO)

Register 27.26: RTC_I2C_INT_ENA_REG (0x0030)



RTC_I2C_SLAVE_TRAN_COMP_INT_ENA RTC_I2C_SLAVE_TRAN_COMP_INT interrupt enable bit. (R/W)

RTC_I2C_ARBITRATION_LOST_INT_ENA RTC_I2C_ARBITRATION_LOST_INT interrupt enable bit. (R/W)

RTC_I2C_MASTER_TRAN_COMP_INT_ENA RTC_I2C_MASTER_TRAN_COMP_INT interrupt enable bit. (R/W)

RTC_I2C_TRANS_COMPLETE_INT_ENA RTC_I2C_TRANS_COMPLETE_INT interrupt enable bit. (R/W)

RTC_I2C_TIME_OUT_INT_ENA RTC_I2C_TIME_OUT_INT interrupt enable bit. (R/W)

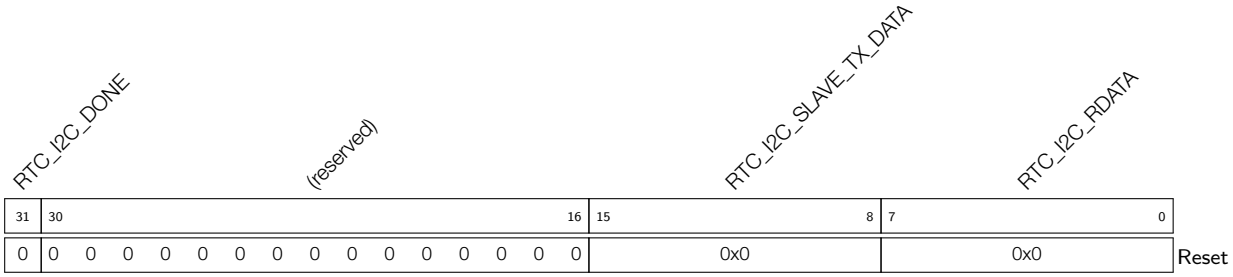
RTC_I2C_ACK_ERR_INT_ENA RTC_I2C_ACK_ERR_INT interrupt enable bit. (R/W)

RTC_I2C_RX_DATA_INT_ENA RTC_I2C_RX_DATA_INT interrupt enable bit. (R/W)

RTC_I2C_TX_DATA_INT_ENA RTC_I2C_TX_DATA_INT interrupt enable bit. (R/W)

RTC_I2C_DETECT_START_INT_ENA RTC_I2C_DETECT_START_INT interrupt enable bit. (R/W)

Register 27.27: RTC_I2C_DATA_REG (0x0034)

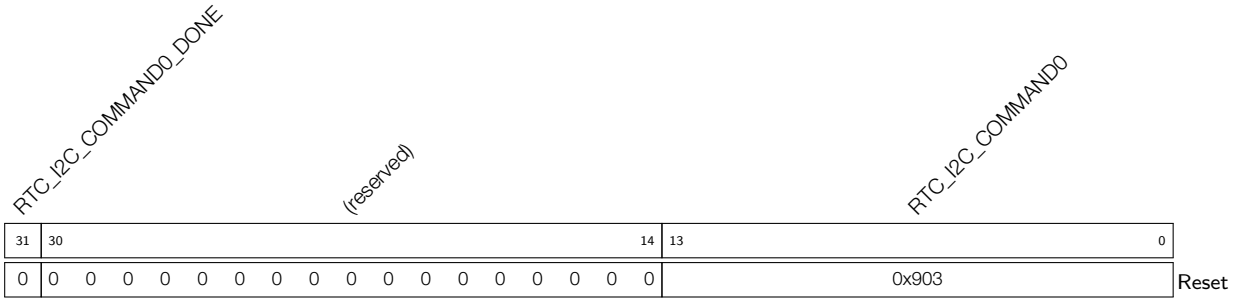


RTC_I2C_RDATA Data received. (RO)

RTC_I2C_SLAVE_TX_DATA The data sent by slave. (R/W)

RTC_I2C_DONE RTC I²C transmission is done. (RO)

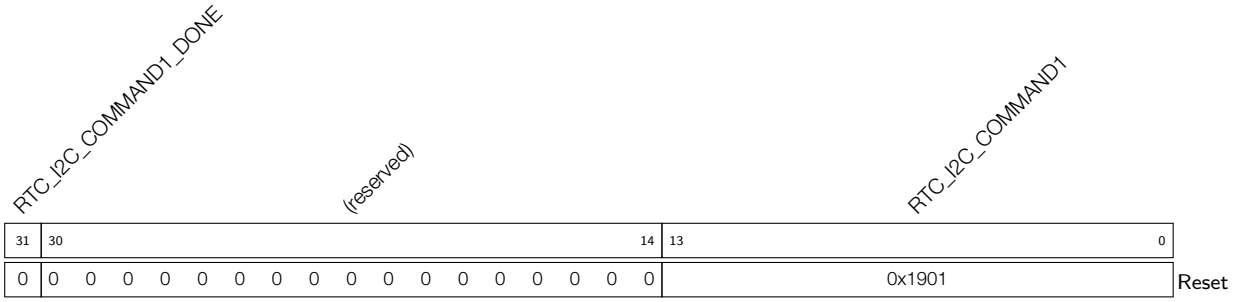
Register 27.28: RTC_I2C_CMD0_REG (0x0038)



RTC_I2C_COMMAND0 Content of command 0. For more information, please refer to the register I2C_COMD0_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND0_DONE When command 0 is done, this bit changes to 1. (RO)

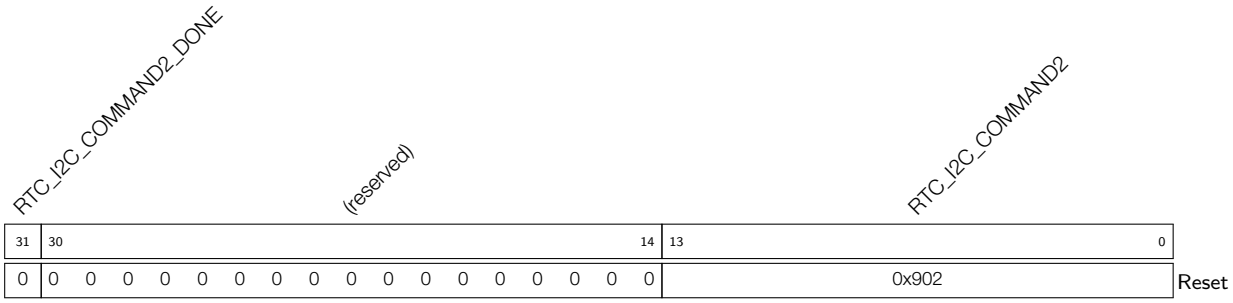
Register 27.29: RTC_I2C_CMD1_REG (0x003C)



RTC_I2C_COMMAND1 Content of command 1. For more information, please refer to the register I2C_COMD1_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND1_DONE When command 1 is done, this bit changes to 1. (RO)

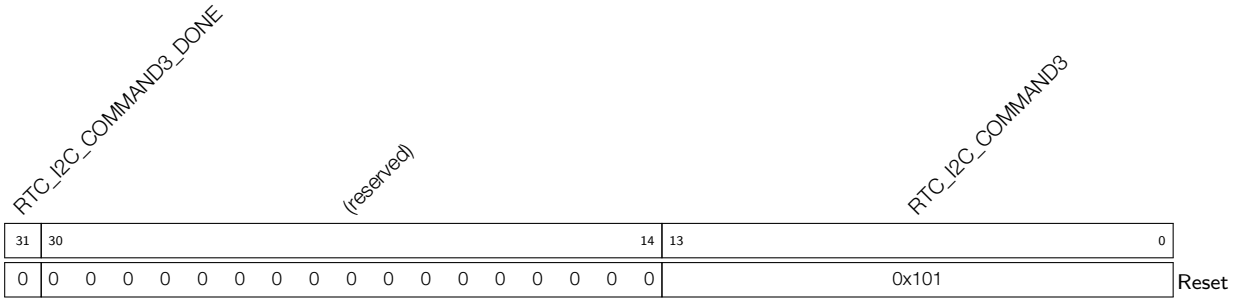
Register 27.30: RTC_I2C_CMD2_REG (0x0040)



RTC_I2C_COMMAND2 Content of command 2. For more information, please refer to the register I2C_COMD2_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND2_DONE When command 2 is done, this bit changes to 1. (RO)

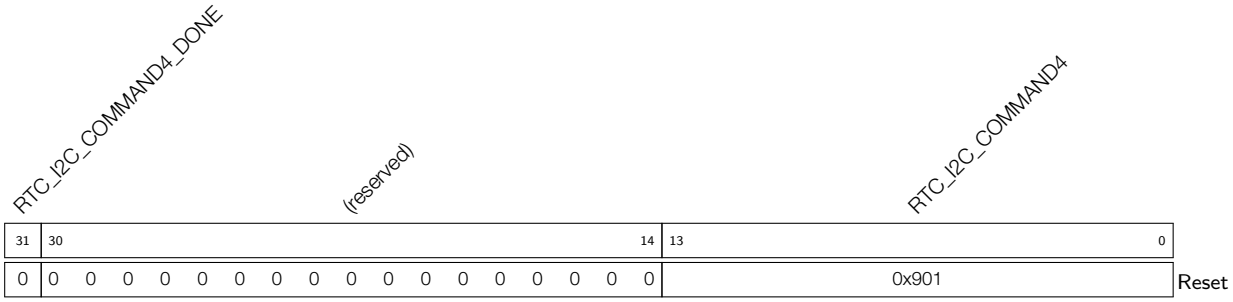
Register 27.31: RTC_I2C_CMD3_REG (0x0044)



RTC_I2C_COMMAND3 Content of command 3. For more information, please refer to the register I2C_COMD3_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND3_DONE When command 3 is done, this bit changes to 1. (RO)

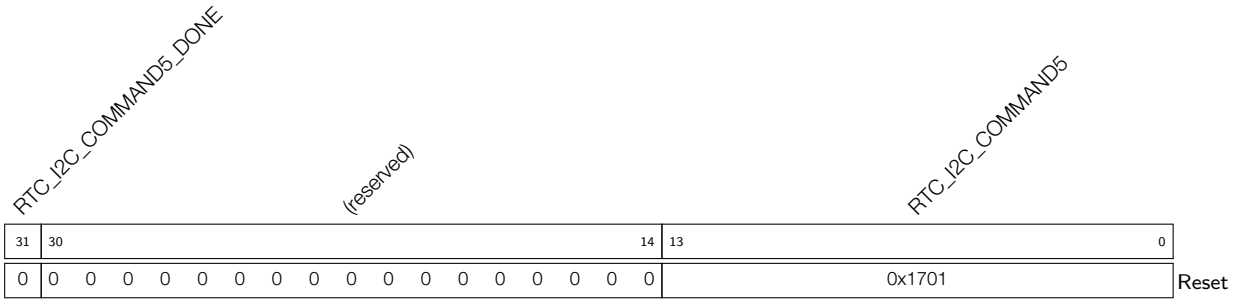
Register 27.32: RTC_I2C_CMD4_REG (0x0048)



RTC_I2C_COMMAND4 Content of command 4. For more information, please refer to the register I2C_COMD4_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND4_DONE When command 4 is done, this bit changes to 1. (RO)

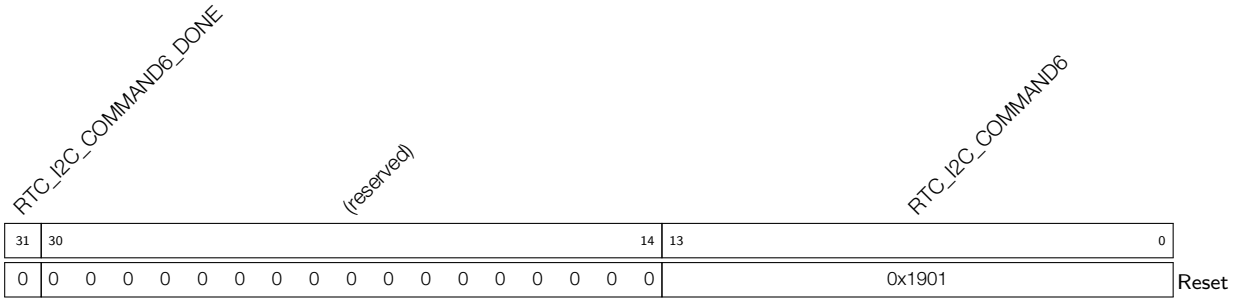
Register 27.33: RTC_I2C_CMD5_REG (0x004C)



RTC_I2C_COMMAND5 Content of command 5. For more information, please refer to the register I2C_COMD5_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND5_DONE When command 5 is done, this bit changes to 1. (RO)

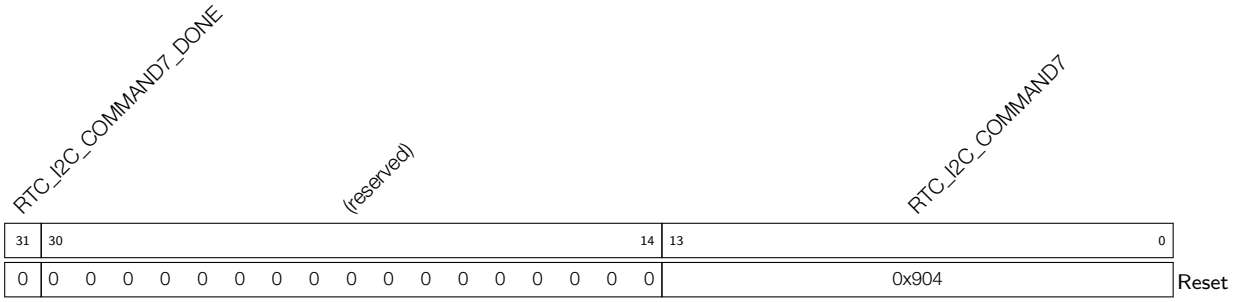
Register 27.34: RTC_I2C_CMD6_REG (0x0050)



RTC_I2C_COMMAND6 Content of command 6. For more information, please refer to the register I2C_COMD6_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND6_DONE When command 6 is done, this bit changes to 1. (RO)

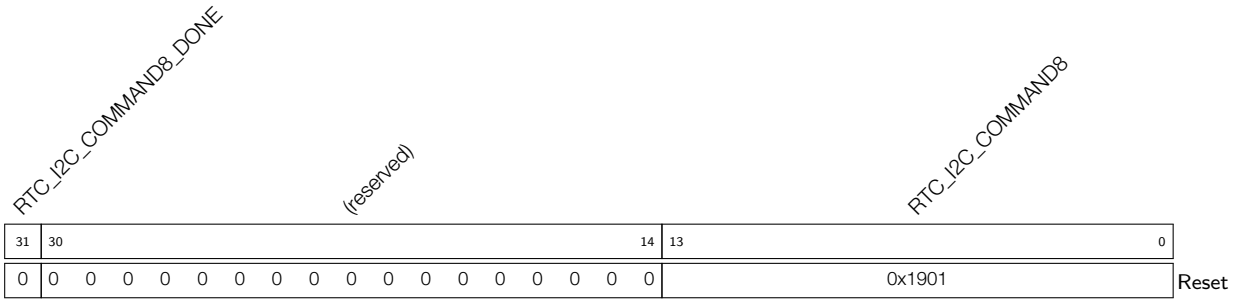
Register 27.35: RTC_I2C_CMD7_REG (0x0054)



RTC_I2C_COMMAND7 Content of command 7. For more information, please refer to the register I2C_COMD7_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND7_DONE When command 7 is done, this bit changes to 1. (RO)

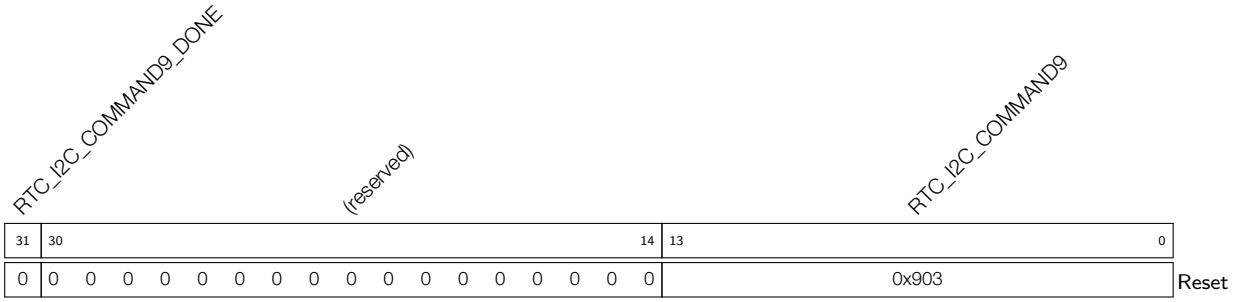
Register 27.36: RTC_I2C_CMD8_REG (0x0058)



RTC_I2C_COMMAND8 Content of command 8. For more information, please refer to the register I2C_COMD8_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND8_DONE When command 8 is done, this bit changes to 1. (RO)

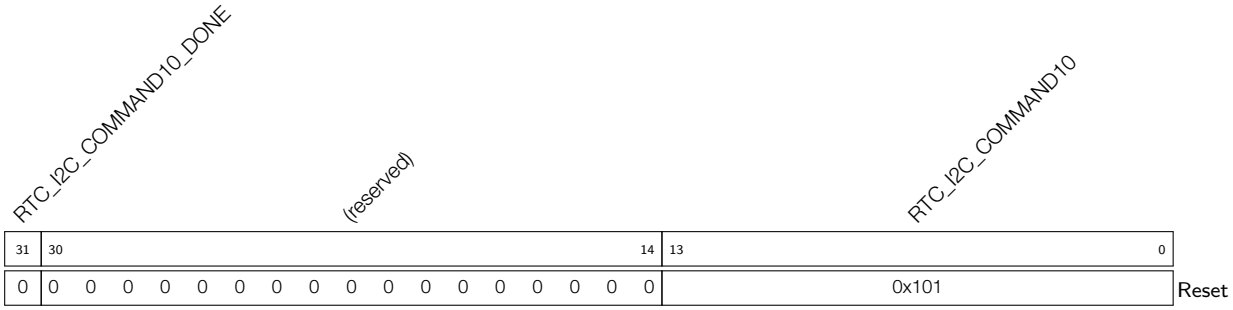
Register 27.37: RTC_I2C_CMD9_REG (0x005C)



RTC_I2C_COMMAND9 Content of command 9. For more information, please refer to the register I2C_COMD9_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND9_DONE When command 9 is done, this bit changes to 1. (RO)

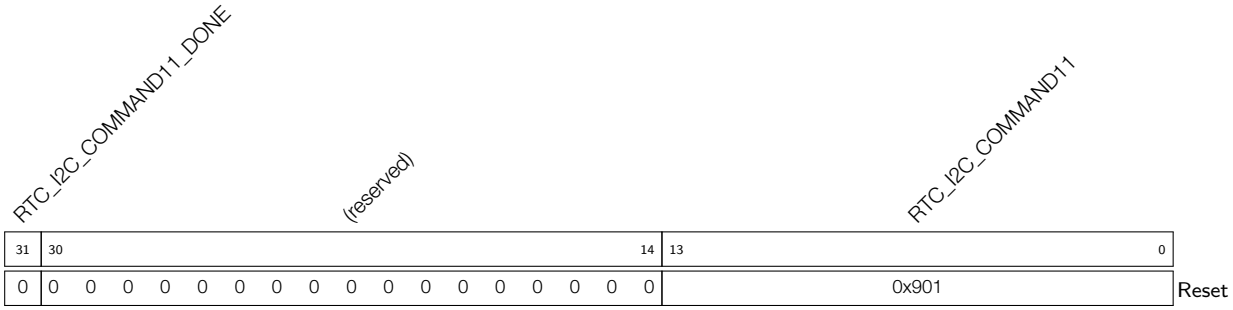
Register 27.38: RTC_I2C_CMD10_REG (0x0060)



RTC_I2C_COMMAND10 Content of command 10. For more information, please refer to the register I2C_COMD10_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND10_DONE When command 10 is done, this bit changes to 1. (RO)

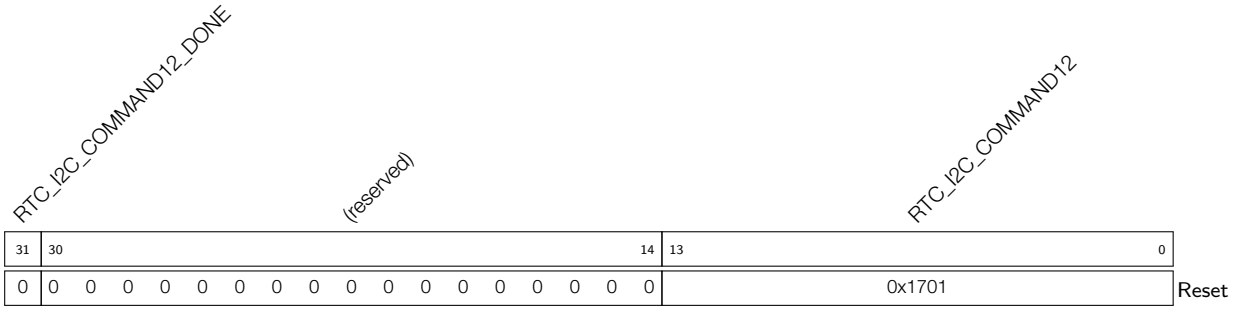
Register 27.39: RTC_I2C_CMD11_REG (0x0064)



RTC_I2C_COMMAND11 Content of command 11. For more information, please refer to the register I2C_COMD11_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND11_DONE When command 11 is done, this bit changes to 1. (RO)

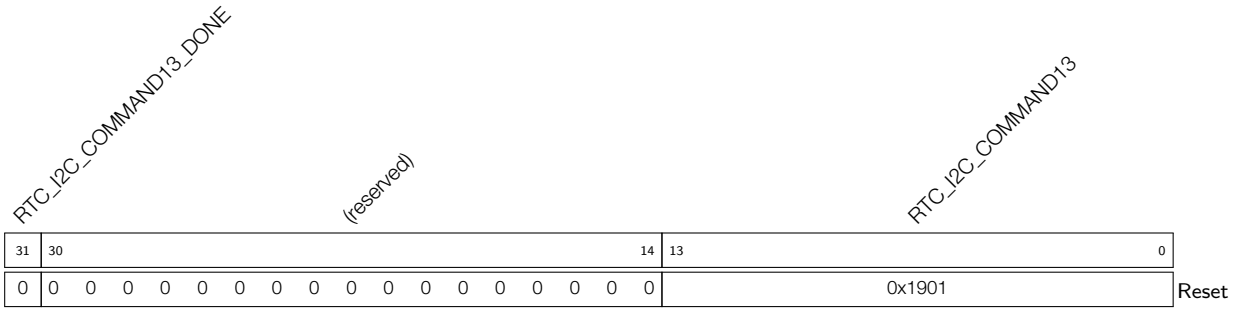
Register 27.40: RTC_I2C_CMD12_REG (0x0068)



RTC_I2C_COMMAND12 Content of command 12. For more information, please refer to the register I2C_COMD12_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND12_DONE When command 12 is done, this bit changes to 1. (RO)

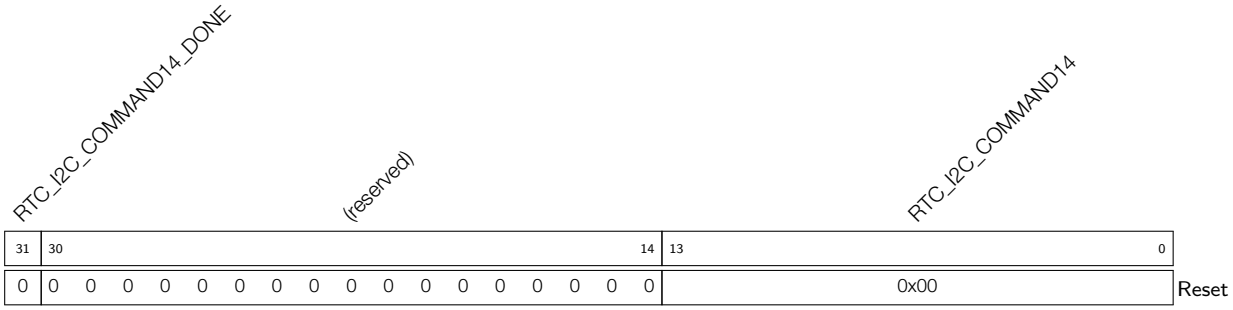
Register 27.41: RTC_I2C_CMD13_REG (0x006C)



RTC_I2C_COMMAND13 Content of command 13. For more information, please refer to the register I2C_COMD13_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND13_DONE When command 13 is done, this bit changes to 1. (RO)

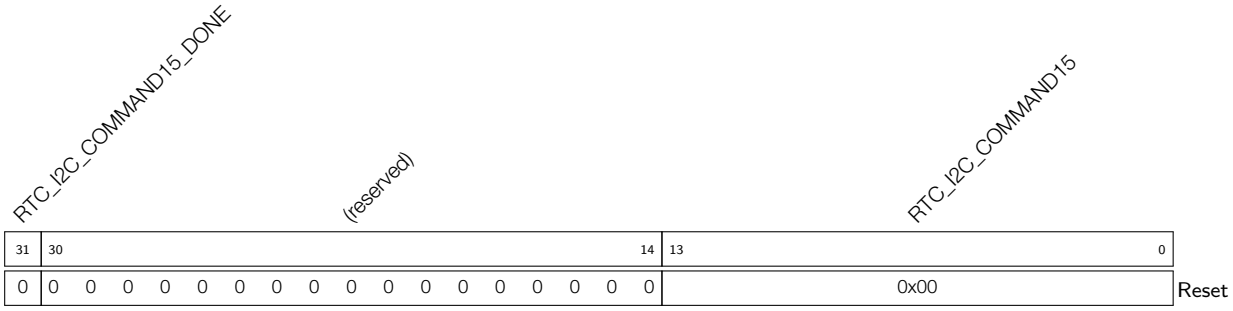
Register 27.42: RTC_I2C_CMD14_REG (0x0070)



RTC_I2C_COMMAND14 Content of command 14. For more information, please refer to the register I2C_COMD14_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND14_DONE When command 14 is done, this bit changes to 1. (RO)

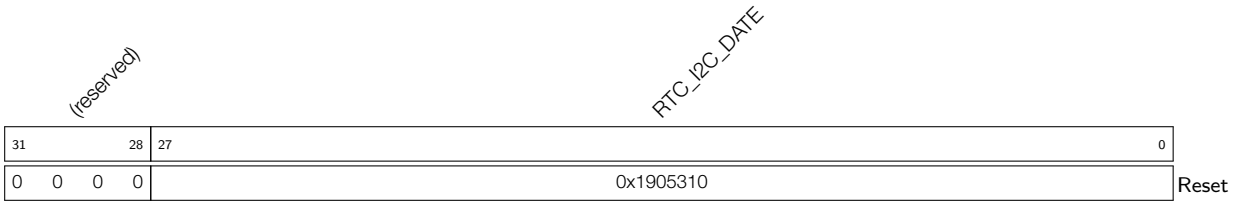
Register 27.43: RTC_I2C_CMD15_REG (0x0074)



RTC_I2C_COMMAND15 Content of command 15. For more information, please refer to the register I2C_COMD15_REG in Chapter I²C Controller. (R/W)

RTC_I2C_COMMAND15_DONE When command 15 is done, this bit changes to 1. (RO)

Register 27.44: RTC_I2C_DATE_REG (0x00FC)



RTC_I2C_DATE Version control register (R/W)

28. Low-power Management

28.1 Introduction

ESP32-S2 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, ESP32-S2 has predefined five 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. ESP32-S2 has integrated two Ultra-Low-Power co-processors (ULP co-processors), which allow the chip to work when most of the power domains are powered down, thus achieving extremely low-power consumption.

28.2 Features

ESP32-S2's low-power management supports the following features:

- ULP co-processors supported in all power modes
- Five predefined power modes to simplify power management for typical scenarios
- Up to 16 KB of retention memory
- 8 x 32-bit retention registers
- RTC Boot supported for reduced wakeup latency

In this chapter, we first introduce the working process of ESP32-S2's low-power management, then introduce the predefined power modes of the chip, and at last, introduce the RTC boot of the chip.

28.3 Functional Description

ESP32-S2's low-power management involves the following components:

- Power management unit: controls the power supply to Analog, RTC and Digital power domains.
- 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.
- Timers:
  - RTC timer: logs the status of the RTC main state machine in dedicated registers.
  - ULP timer: wakes up the ULP co-processors at a predefined time. For details, please refer to Chapter 27 *ULP Coprocessor*.
  - Touch sensor timer: wakes up the touch sensor at a predefined time.
- 8 x 32-bit "always-on" retention registers: These registers are always powered up and can be used for storing data that cannot be lost.
- 22 x "always-on" pins: These pins are always powered up and can be used as wakeup sources when the chip is working in the low-power modes (for details, please refer to Section 28.4.4), or can be used as regular GPIOs (for details, please refer to Chapter 5 *IO MUX and GPIO Matrix*).

- RTC slow memory: supports 8 KB SRAM, which can be used as retention memory or to store ULP directives and data.
- RTC fast memory: supports 8 KB SRAM, which can be used as retention memory.
- Regulators: regulate the power supply to different power domains.

The schematic diagram of ESP32-S2's low-power management is shown in Figure 28-1.

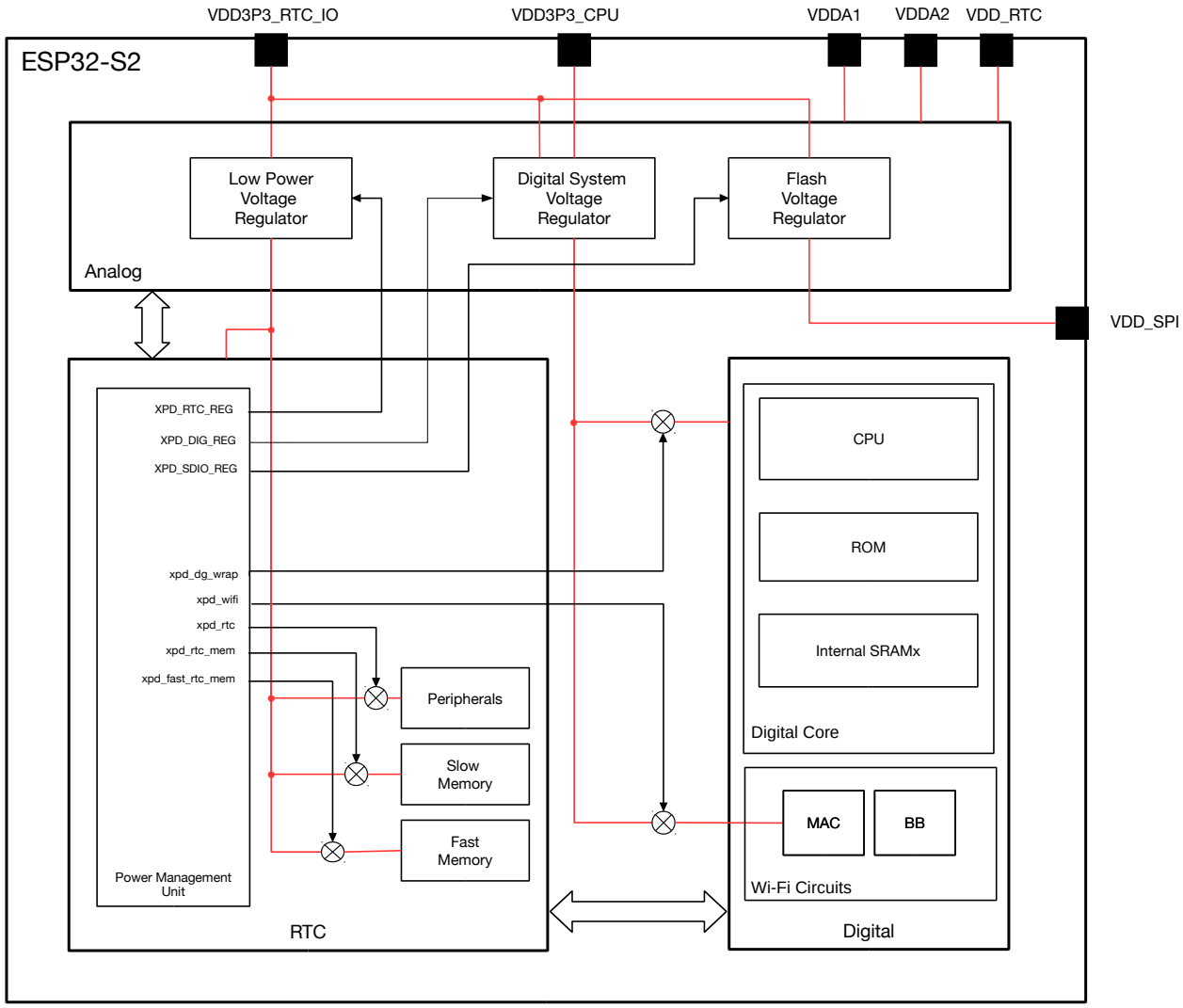


Figure 28-1. Low-power Management Schematics

28.3.1 Power Management Unit

ESP32-S2'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.

In ESP32-S2'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 28-2.

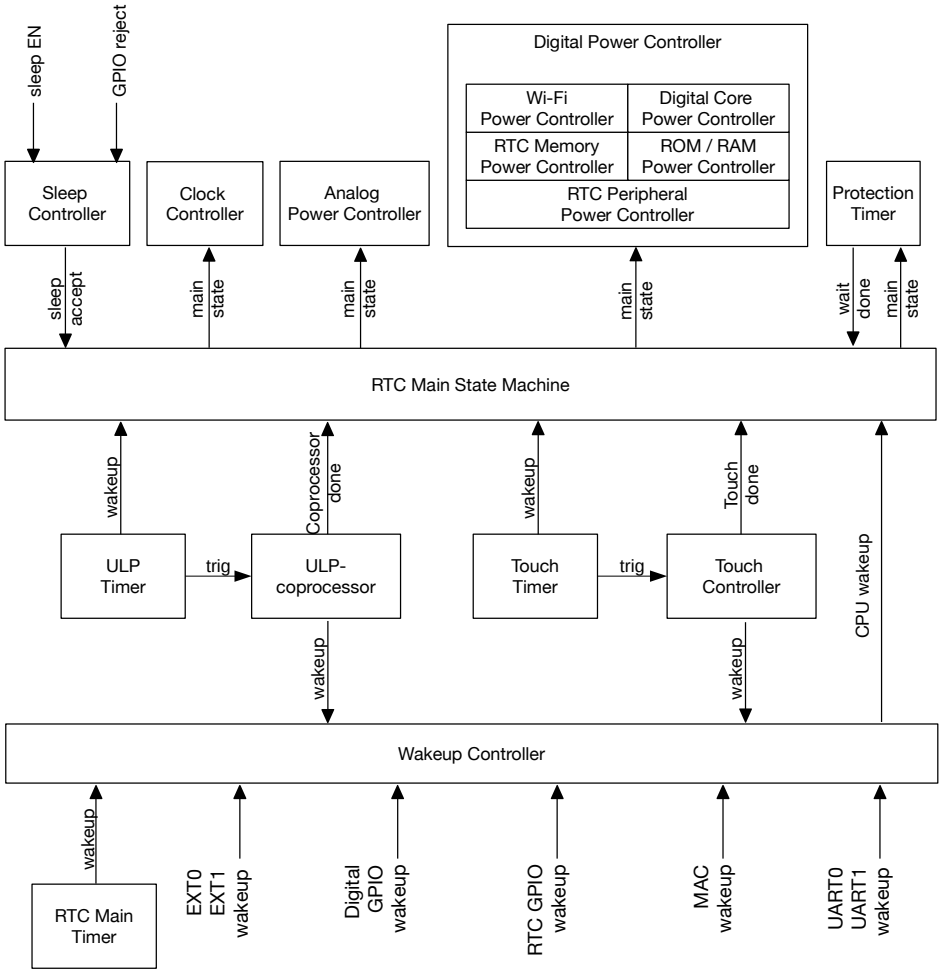


Figure 28-2. Power Management Unit Workflow

Note:
For more detailed description about power domains, please refer to 28.4.2.

28.3.2 Low-Power Clocks

In general, ESP32-S2 powers down its 40 MHz crystal oscillator and PLL 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 different power domains, such as the power management unit, RTC peripherals, RTC fast memory, RTC slow memory, and Wi-Fi digital circuits in the digital domain.

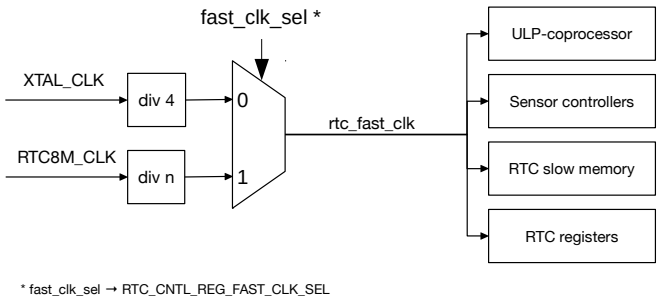


Figure 28-3. Low-Power Clocks for RTC Power Domains

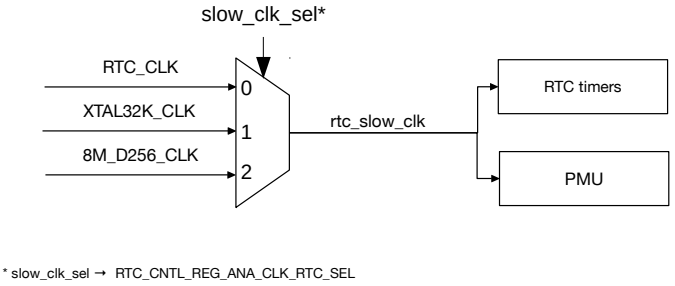


Figure 28-4. Low-Power Clocks for RTC Power Domains

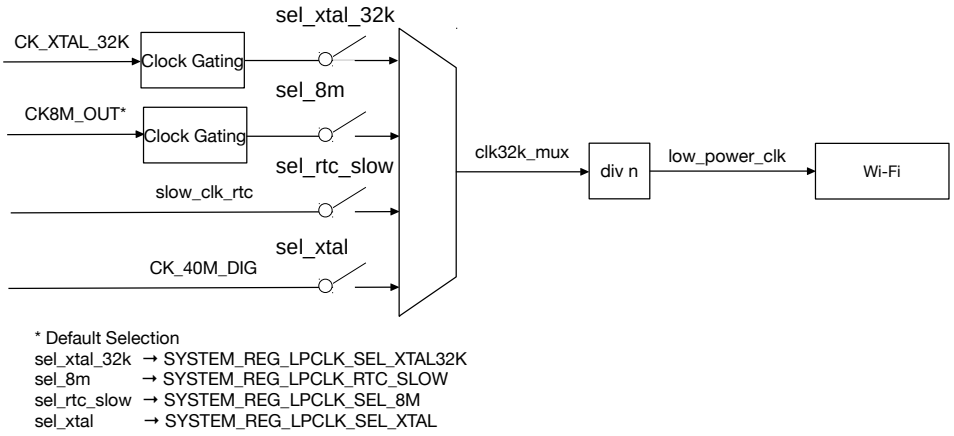


Figure 28-5. Low-Power Clocks for RTC Fast Memory and Slow Memory

Table 151: Low-Power Clocks

Clock Type	Clock Source	Selection Option	Power Domain
RTC fast clock	RTC8M_CLK divided by n [1]	RTC_CNTL_REG_FAST_CLK_RTC_SEL	RTC peripherals
	XTAL_CLK divided by 4		RTC fast memory
			RTC slow memory
RTC slow clock	XTAL32K_CLK	RTC_CNTL_REG_ANA_CLK_RTC_SEL	Power management unit
	RTC8M_D256_CLK		
	RTC_CLK [2]		
Low-Power clocks	XTAL32K_CLK	SYSTEM_LPCLK_SEL_XTAL32K	Digital system (Wi-Fi) in low-power modes
	RTC8M_CLK	SYSTEM_LPCLK_SEL_8M	
	RTC_CLK	SYSTEM_LPCLK_RTC_SLOW	
	XTAL_CLK	SYSTEM_LPCLK_SEL_XTAL	

Note:

1. The default RTC fast clock source.
2. The default RTC slow clock source.

For more detailed description about clocks, please refer to 2 *Reset and Clock*.

28.3.3 Timers

ESP32-S2's low-power management uses 3 timers:

- RTC timer
- ULP timer
- Touch sensor timer

This section only introduces the RTC timer. For detailed description of ULP timer and touch sensor timer, please refer to Chapters 27 *ULP Coprocessor*.

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 152.

Table 152: The Triggering Conditions for the RTC Timer

Enabling Options	Descriptions
RTC_CNTL_REG_TIMER_XTL_OFF	1. RTC main state machine powers down; 2. 40 MHz crystal powers up.
RTC_CNTL_REG_TIMER_SYS_STALL	CPU enters or exits the stall state. This is to ensure the SYS_TIMER is continuous in time.
RTC_CNTL_REG_TIMER_SYS_RST	Resetting digital system completes.
RTC_CNTL_REG_TIME_UPDATE	Register RTC_CNTL_RTC_TIME_UPDATE is configured by CPU (i.e. users).

The RTC timer updates two groups of registers upon any new trigger. The first group logs the information 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 overrode), 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 28.4.4.

28.3.4 Regulators

ESP32-S2 has three regulators to regulate the power supply to different power domains:

- Digital system voltage regulator for digital power domains;
- Low-power voltage regulator for RTC power domains;
- Flash voltage regulator for the rest of power domains.

Note:
For more detailed description about power domains, please refer to Section 28.4.2.

28.3.4.1 Digital System Voltage Regulator

ESP32-S2'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 has an adjustable output. For the architecture of the ESP32-S2 digital system voltage regulator, see Figure 28-6.

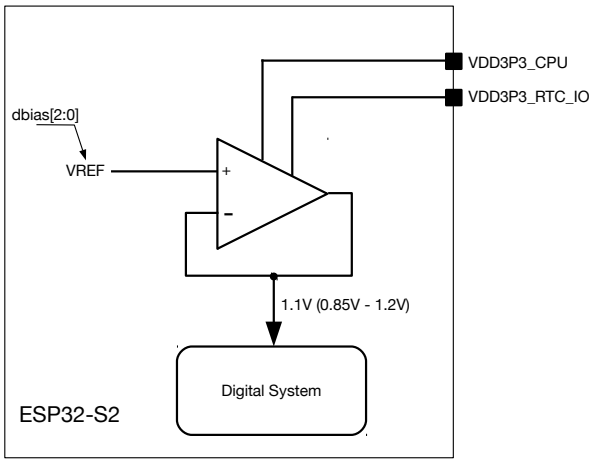


Figure 28-6. Digital System Regulator

1. When XPD_DIG_REG == 1, the regulator outputs a 1.1 V voltage and the power domains in digital system are able to run; when XPD_DIG_REG == 0, both the regulator and the power domains in digital system stop running.
2. DIG_REG_DBIAS[2:0] tunes the supply voltage of the digital system:

$$\text{VDD\_DIG} = 0.90 + \text{DBIAS} \times 0.05\text{V}$$

3. The current to power domains in digital system comes from pin VDD3P3_CPU and pin VDD3P3_RTC_IO.

28.3.4.2 Low-power Voltage Regulator

ESP32-S2's built-in low-power voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for RTC power domains. This regulator has an adjustable output to achieve lower power consumption and can output even lower voltage in Deep-sleep mode and Hibernation mode. For the architecture of the ESP32-S2 low-power voltage regulator, see Figure 28-1.

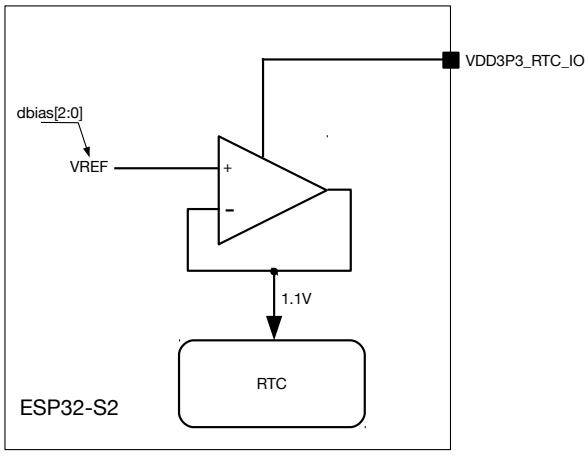


Figure 28-7. Low-power voltage regulator

1. When the pin CHIP_PU 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.
2. RTC_DBIAS[2:0] can be used to tune the output voltage:

$$VDD\_RTC = 0.90 + DBIAS \times 0.05V$$

3. The current to the RTC power domains comes from pin VDD3P3_RTC_IO.

28.3.4.3 Flash Voltage Regulator

ESP32-S2's built-in flash voltage regulator can supply a voltage of 3.3 V or 1.8 V to other components outside of digital system and RTC, such as flash. For the architecture of the ESP32-S2 flash voltage regulator, see Figure 28-8.

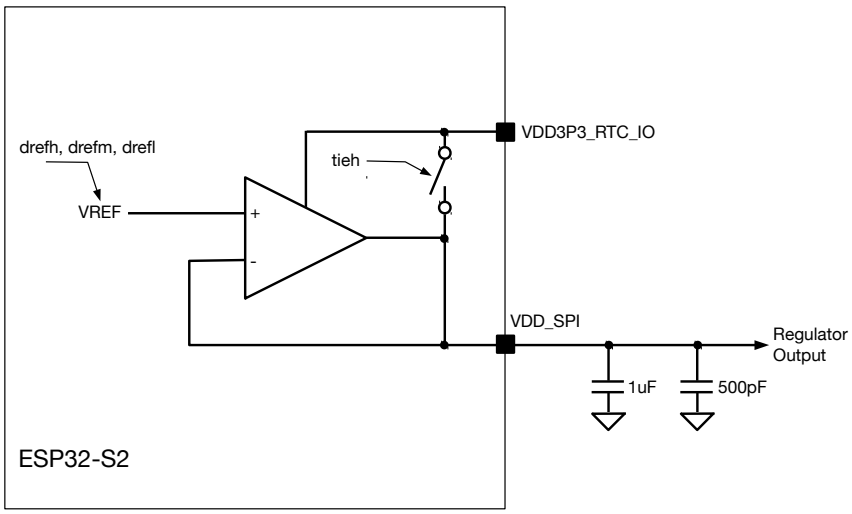


Figure 28-8. Flash voltage regulator

1. When XPD_SDIO_VREG == 1, the regulator outputs a voltage of 3.3 V or 1.8 V; when XPD_SDIO_VREG == 0, the output is high-impedance. In this case, the voltage is provided by the external power supply.
2. When SDIO_TIEH == 1, the regulator shorts pin VDD_SDIO to pin VDD3P3_RTC, and outputs a voltage of 3.3 V, which is the voltage of pin VDD3P3_RTC. When SDIO_TIEH == 0, the regulator outputs the reference voltage VREF, which is typically 1.8 V.
3. DREFH_SDIO, DREFM_SDIO and DREFL_SDIO could be used to fine tune the reference voltage VREF, but are not recommended because this fine tuning may jeopardize the stability of the inner loop.
4. When the regulator output is 3.3 V or 1.8 V, the output current comes from the pin VDD3P3_RTC_IO.

The flash voltage regulator can be configured via RTC registers or eFuse controller registers.

- The configuration of XPD_SDIO_VREG:
  - When the chip is in active mode, RTC_CNTL_SDIO_FORCE == 0 and VDD_SPI_FORCE(EFUSE) == 1, the XPD_SDIO_VREG voltage is defined by XPD_VDD_SPI_REG(EFUSE);
  - When the chip is in sleep modes and RTC_CNTL_SDIO_REG_PD_EN == 1, XPD_SDIO_VREG is 0;

  - When RTC_CNTL_SDIO_FORCE == 1, XPD_SDIO_VREG is defined by RTC_CNTL_XPD_SDIO_REG.

- The configuration of SDIO_TIEH:
  - When RTC_CNTL_SDIO_FORCE == 0 and VDD_SPI_FORCE(EFUSE) == 1, SDIO_TIEH = VDD_SDIO_TIEH(EFUSE)
  - Otherwise, SDIO_TIEH = RTC_CNTL_SDIO_TIEH.

28.3.4.4 Brownout Detector

The brownout detector checks the voltage of pins VDD3P3_RTC_IO, VDD3P3_CPU, VDDA1, and VDDA2. If the voltage of these pins drops rapidly and becomes too low, the detector would trigger a signal to shut down some power-consuming blocks (such as LNA, PA, etc.) to allow extra time for the digital 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 ESP32-S2 brownout detector, see Figure 28-9.

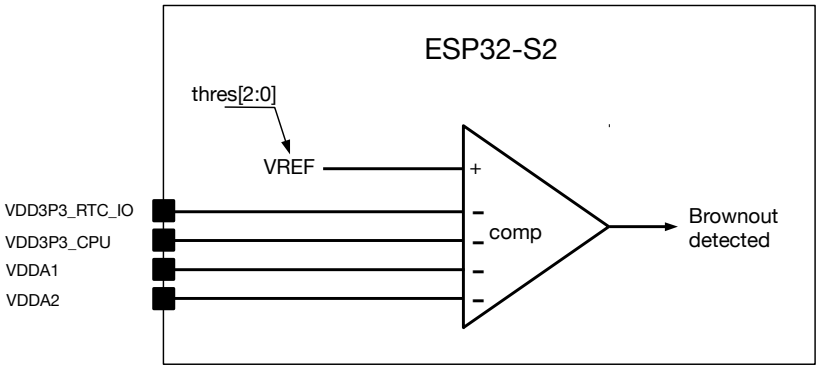


Figure 28-9. Brown-out detector

1. RTC_CNTL_RTC_BROWN_OUT_DET indicates the output level of brown-out detector. This register is low level by default, and outputs high level when the voltage of the detected pin drops below the predefined threshold;
2. I2C register ULP_CAL_REG5[2:0] configures the trigger threshold of the brown-out detector;

Table 153: Brown-out Detector Configuration

ULP_CAL_REG5[2:0]	VDD (Unit: V)
0	2.67
1	3.30
2	3.19
3	2.98
4	2.84
5	2.67
6	2.56
7	2.44

3. RTC_CNTL_REG_BROWN_OUT_RST_SEL configures the reset type.
    - 0: resets the chip
    - 1: resets the system

Note:
For more information regarding chip reset and system reset, please refer to 2 *Reset and Clock*.

28.4 Power Modes Management

28.4.1 Power Domain

ESP32-S2 has 10 power domains in three power domain categories:

- RTC
    - Power management unit
    - RTC peripherals
    - RTC slow memory
    - RTC fast memory
- Digital
    - Digital core
    - Wi-Fi digital circuits
- Analog
    - 8 MHz crystals
    - 40 MHz crystals
    - PLL
    - RF circuits

28.4.2 RTC States

ESP32-S2 has three main RTC states: Active, Monitor, and Sleep. The transition process among these states can be seen in Figure 28-10.

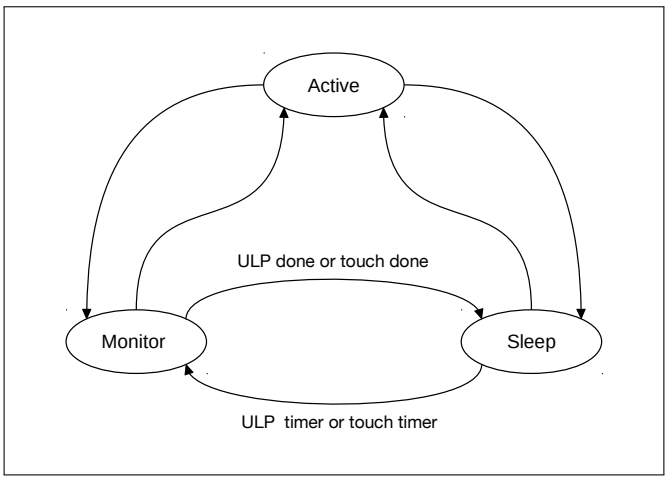


Figure 28-10. RTC States

Under different RTC states, different power domains are powered up or down by default, but can also be force-powered-up (FPU) or force-powered-down (FPD) individually based on actual requirements. For details, please refer to Table 154.

Table 154: RTC Statues Transition

Power Domain		RTC States			Notes
Category	Sub-category	Active	Monitor	Sleep	
RTC	Power Management Unit	ON	ON	ON	1
	RTC Peripherals	ON	ON	OFF	2
	RTC Slow Memory	ON	OFF	OFF	3
	RTC Fast Memory	ON	OFF	OFF	4
Digital	Digital Core	ON	OFF	OFF	5
	Wi-Fi Digital Circuits	ON	OFF	OFF	6
Analog	8 MHz Crystals	ON	ON	OFF	-
	40 MHz Crystals	ON	OFF	OFF	-
	PLL	ON	OFF	OFF	-
	RF Circuits	-	-	-	-

Note:

1. ESP32-S2's power management unit is specially designed to be "always-on", which means it is always on when the chip is powered up. Therefore, users cannot FPU or FPD the power management unit.
2. The RTC peripherals include 27 *ULP Coprocessor* and on-chip sensor (i.e. temperature sensor controller, touch sensor, SAR ADC controller).
3. RTC slow memory supports 8 KB SRAM, which can be used to reserve memory or to store ULP instructions and / or data. This memory can be accessed by CPU via PeriBUS2 (starting address is 0x50000000), and should be force-power-on.
4. RTC fast memory supports 8 KB SRAM, which can be used to reserve memory. This memory can be accessed by CPU via IRAM0 / DRAM0, and should be forced-power-up.
5. When the digital core of the digital system is powered down, all components in the digital system are turned off.

It's worth noting that, ESP32-S2's ROM and SRAM are no longer controlled as independent power domains, thus cannot be force-powered-up or force-powered-down when the digital core is powered down.

6. Power domain Wi-Fi digital circuits includes Wi-Fi MAC and BB (Base Band).

28.4.3 Pre-defined Power Modes

As mentioned earlier, ESP32-S2 has five power modes, which are predefined configurations that power up different combinations of power domains. For details, please refer to Table 155.

Table 155: Predefined Power Modes

Power Modes	Power Domain									
	PMU	RTC Peripherals	RTC Slow Memory	RTC Fast Memory	Digital Core	Wi-Fi Digital Circuits	8 MHz Crystals	40 MHz Crystals	PLL	RF Circuits
Active	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	ON *	ON	ON	ON	OFF
Light-sleep	ON	ON	ON	ON	ON	ON *	OFF	OFF	OFF	OFF
Deep-sleep	ON	ON	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF
Hibernation	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Note:

* Configurable

By default, ESP32-S2 first enters the Active mode after system resets, then enters different low-power modes (including Modem-sleep, Light-sleep, Deep-sleep, and Hibernation) to save power after the CPU stalls for a specific time (For example, when CPU is waiting to be wakened up by an external event). From modes Active to Hibernation, the performance[1] and power consumption[2] decreases and wakeup latency increases. Also, the supported wakeup sources for different power modes are different[3]. Users can choose a power mode based on their requirements of performance, power consumption, wakeup latency, and available wakeup sources.

Note:

1. For details, please refer to Table 155.
2. For details on power consumption, please refer to the Current Consumption Characteristics in *ESP32-S2 Datasheet*.
3. For details on the supported wakeup sources, please refer to Section 28.4.4.

28.4.4 Wakeup Source

The ESP32-S2 supports various wakeup sources, which could wake up the CPU in different sleep modes. The wakeup source is determined by RTC_CNTL_REG_RTC_WAKEUP_ENA as shown in Table 156.

Table 156: Wakeup Source

WAKEUP_ENA	Wakeup Source	Light-sleep	Deep-sleep	Hibernation	Notes*
0x1	EXT0	Y	Y	-	1
0x2	EXT1	Y	Y	Y	2
0x4	GPIO	Y	Y	-	3
0x8	RTC timer	Y	Y	Y	-
0x20	Wi-Fi	Y	-	-	4
0x40	UART0	Y	-	-	5
0x80	UART1	Y	-	-	5
0x100	TOUCH	Y	Y	-	6
0x800	ULP-FSM	Y	Y	-	7
0x1000	XTAL_32K	Y	Y	Y	8
0x2000	ULP-RISCV Trap	Y	Y	-	9
0x8000	USB	Y	-	-	10

Note:

1. EXT0 can only wake up the chip from Light-sleep / Deep-sleep modes. If RTC_CNTL_REG_EXT_WAKEUP0_LV is 1, it's triggered when the pin level is high. Otherwise, it's triggered when the pin level is low. Users can configure RTCIO_EXT_WAKEUP0_SEL to select an RTC pin as a wakeup source.
2. EXT1 is especially designed to wake up the chip from any sleep modes, and can be triggered by a combination of pins. Users should define the combination of wakeup sources by configuring RTC_CNTL_REG_EXT_WAKEUP1_SEL[17:0] according to the bitmap of selected wakeup source. When RTC_CNTL_REG_EXT_WAKEUP1_LV == 1, the chip is waken up if any pin in the combination is high level. When RTC_CNTL_REG_EXT_WAKEUP1_LV == 0, the chip is only waken up if all pins in the combination is low level.
3. In Deep-sleep mode, only the RTC GPIOs (not regular GPIOs) can work as a wakeup source.
4. 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.
5. A wakeup is triggered when the number of RX pulses received exceeds the setting in the threshold register.
6. A wakeup is triggered when any touch event is detected by the touch sensor.
7. A wakeup is triggered when RTC_CNTL_RTC_SW_CPU_INT is configured by the ULP co-processor.
8. When the 32 kHz crystal is working as RTC slow clock, a wakeup is triggered upon any detection of any crystal stops by the 32 kHz watchdog timer.
9. A wakeup is triggered when the ULP co-processor starts capturing exceptions (e.g., stack overflow).
10. A wakeup is triggered when the USB host sends USB into the RESUMING state.

28.5 RTC Boot

The wakeup time for Deep-sleep and Hibernation modes are much longer, compared to the Light-sleep and Modem-sleep, because the ROMs and RAMs are both powered down in this case, and the CPU needs more time for ROM unpacking and data-copying from the flash (SPI booting). However, it's worth noting that both RTC fast memory and RTC slow memory remain powered up in the Deep-sleep mode. Therefore, users can store

codes (so called “deep sleep wake stub” of up to 8 KB) either in RTC fast memory or RTC slow memory to avoid the above-mentioned ROM unpacking and SPI booting, thus speeding up the wakeup process.

Method one: Using RTC slow memory

1. Set RTC_CNTL_PROCPU_STAT_VECTOR_SEL to 0.
2. Send the chip into sleep.
3. After the CPU is powered up, the reset vector starts resetting from 0x50000000 instead of 0x40000400, which does not involve any ROM unpacking and SPI booting. The codes stored in RTC slow memory starts running immediately after the CPU reset. However, note that the content in the RTC slow memory should be initialized before sending the chip into sleep.

Method two: Using RTC fast memory

1. Set RTC_CNTL_PROCPU_STAT_VECTOR_SEL to 1.
2. Calculate CRC for the RTC fast memory, and save the result in RTC_CNTL_STORE6_REG[31:0].
3. Set RTC_CNTL_STORE7_REG[31:0] to the entry address of RTC fast memory.
4. Send the chip into sleep.
5. ROM unpacking and some of the initialization starts after the CPU is powered up. After that, calculate the CRC for the RTC fast memory again. If the result matches with register RTC_CNTL_STORE6_REG[31:0], the CPU jumps to the entry address.

The boot flow is shown in Figure 28-11.

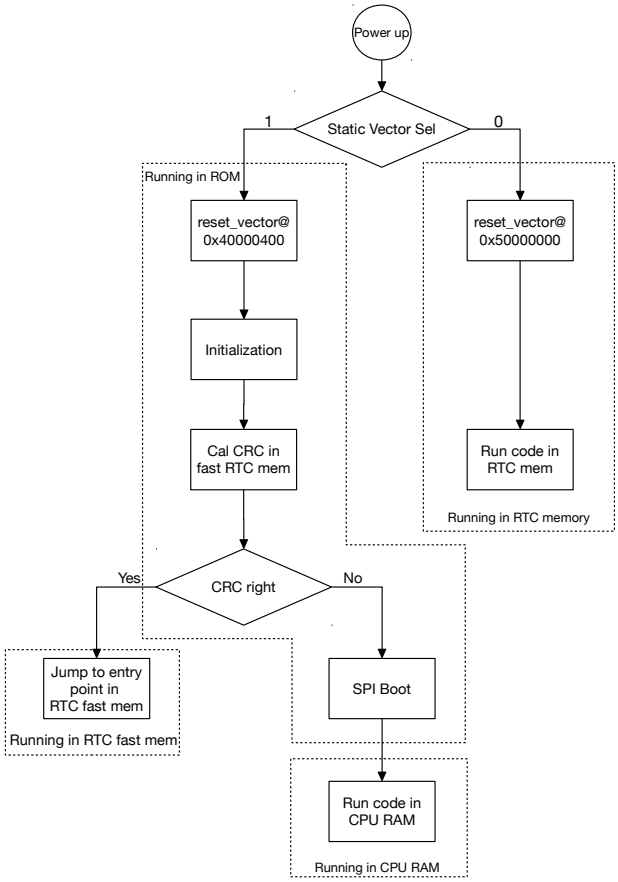


Figure 28-11. ESP32-S2 Boot Flow

When working under low-power modes, ESP32-S2's 40 MHz crystal oscillator and PLL are usually powered down to reduce power consumption. However, the low-power clock remains on so the chip can operate properly under low-power modes.

28.6 Base Address

Users can access the low-power management module of ESP32-S2 with two base addresses, which can be seen in Table 157. For more information about accessing peripherals from different buses please see Chapter 1 *System and Memory*.

Table 157: Low-power Management Base Address

Bus to Access Peripheral	Base Address
PeriBUS1	0x3f408000
PeriBUS2	0x60008000

28.7 Register Summary

The addresses in the following table are relative to the Low Power Management base addresses provided in Section 28.6.

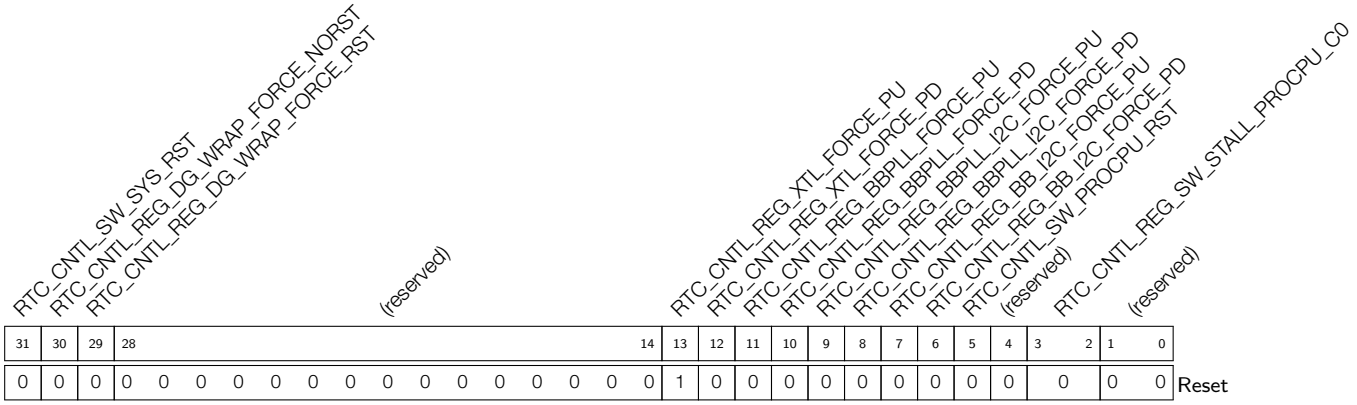
Name	Description	Address	Access
RTC Option Registers			
RTC_CNTL_OPTIONS0_REG	Sets the power options of crystal and PLL clocks, and initiates reset by software	0x0000	varies
RTC_CNTL_OPTION1_REG	RTC option register	0x0128	R/W
RTC Timer Registers			
RTC_CNTL_SLP_TIMER0_REG	RTC timer threshold register 0	0x0004	R/W
RTC_CNTL_SLP_TIMER1_REG	RTC timer threshold register 1	0x0008	varies
RTC_CNTL_TIME_UPDATE_REG	RTC timer update control register	0x000C	varies
RTC_CNTL_TIME_LOW0_REG	Stores the lower 32 bits of RTC timer 0	0x0010	RO
RTC_CNTL_TIME_HIGH0_REG	Stores the higher 16 bits of RTC timer 0	0x0014	RO
RTC_CNTL_STATE0_REG	Configures the sleep / reject / wakeup state	0x0018	varies
RTC_CNTL_TIMER1_REG	Configures CPU stall options	0x001C	R/W
RTC_CNTL_TIMER2_REG	Configures RTC slow clock and touch controller	0x0020	R/W
RTC_CNTL_TIMER5_REG	Configures the minimal sleep cycles	0x002C	R/W
RTC_CNTL_TIME_LOW1_REG	Stores the lower 32 bits of RTC timer 1	0x00E8	RO
RTC_CNTL_TIME_HIGH1_REG	Stores the higher 16 bits of RTC timer 1	0x00EC	RO
Internal Power Control Register			
RTC_CNTL_ANA_CONF_REG	Configures the power options for I2C and PLLA	0x0034	R/W
RTC_CNTL_REG_REG	Low-power voltage and digital voltage regulators configuration register	0x0084	R/W
RTC_CNTL_PWC_REG	RTC power configuration register	0x0088	R/W
RTC_CNTL_DIG_PWC_REG	Digital system power configuraiton register	0x008C	R/W
RTC_CNTL_DIG_ISO_REG	Digital system isolation configuration register	0x0090	varies
RTC_CNTL_LOW_POWER_ST_REG	RTC main state machine state register	0x00CC	RO
Reset Control Register			
RTC_CNTL_RESET_STATE_REG	Indicates the CPU reset source	0x0038	varies
Sleep and Wake-up Control Register			
RTC_CNTL_WAKEUP_STATE_REG	Wakeup bitmap enabling register	0x003C	R/W
RTC_CNTL_EXT_WAKEUP_CONF_REG	GPIO wakeup configuration register	0x0064	R/W
RTC_CNTL_SLP_REJECT_CONF_REG	Configures sleep / reject options	0x0068	R/W
RTC_CNTL_EXT_WAKEUP1_REG	EXT1 wakeup configuration register	0x00DC	varies
RTC_CNTL_EXT_WAKEUP1_STATUS_REG	EXT1 wakeup source register	0x00E0	RO
RTC_CNTL_SLP_REJECT_CAUSE_REG	Stores the reject-to-sleep cause	0x0124	RO
RTC_CNTL_SLP_WAKEUP_CAUSE_REG	Stores the sleep-to-wakeup cause	0x012C	RO
Interrupt Registers			
RTC_CNTL_INT_ENA_RTC_REG	RTC interrupt enabling register	0x0040	R/W
RTC_CNTL_INT_RAW_RTC_REG	RTC interrupt raw register	0x0044	RO
RTC_CNTL_INT_ST_RTC_REG	RTC interrupt state register	0x0048	RO
RTC_CNTL_INT_CLR_RTC_REG	RTC interrupt clear register	0x004C	WO
Reservation Registers			
RTC_CNTL_STORE0_REG	Reservation register 0	0x0050	R/W

Name	Description	Address	Access
RTC_CNTL_STORE1_REG	Reservation register 1	0x0054	R/W
RTC_CNTL_STORE2_REG	Reservation register 2	0x0058	R/W
RTC_CNTL_STORE3_REG	Reservation register 3	0x005C	R/W
RTC_CNTL_STORE4_REG	Reservation register 4	0x00BC	R/W
RTC_CNTL_STORE5_REG	Reservation register 5	0x00C0	R/W
RTC_CNTL_STORE6_REG	Reservation register 6	0x00C4	R/W
RTC_CNTL_STORE7_REG	Reservation register 7	0x00C8	R/W
Clock Control Registers			
RTC_CNTL_EXT_XTL_CONF_REG	32 kHz crystal oscillator configuration register	0x0060	varies
RTC_CNTL_CLK_CONF_REG	RTC clock configuration register	0x0074	R/W
RTC_CNTL_SLOW_CLK_CONF_REG	RTC slow clock configuration register	0x0078	R/W
RTC_CNTL_XTAL32K_CLK_FACTOR_REG	Configures the divider for the backup clock of 32 kHz crystal oscillator	0x00F0	R/W
RTC_CNTL_XTAL32K_CONF_REG	32 kHz crystal oscillator configuration register	0x00F4	R/W
RTC Watchdog Control Register			
RTC_CNTL_WDTCONFIG0_REG	RTC watchdog configuration register	0x0094	R/W
RTC_CNTL_WDTCONFIG1_REG	Configures the hold time of RTC watchdog at level 1	0x0098	R/W
RTC_CNTL_WDTCONFIG2_REG	Configures the hold time of RTC watchdog at level 2	0x009C	R/W
RTC_CNTL_WDTCONFIG3_REG	Configures the hold time of RTC watchdog at level 3	0x00A0	R/W
RTC_CNTL_WDTCONFIG4_REG	Configures the hold time of RTC watchdog at level 4	0x00A4	R/W
RTC_CNTL_WDTFEED_REG	RTC watchdog SW feed configuration register	0x00A8	WO
RTC_CNTL_WDTWPROTECT_REG	RTC watchdog write protection configuration register	0x00AC	R/W
RTC_CNTL_SWD_CONF_REG	Super watchdog configuration register	0x00B0	varies
RTC_CNTL_SWD_WPROTECT_REG	Super watchdog write protection configuration register	0x00B4	R/W
Other Registers			
RTC_CNTL_SW_CPU_STALL_REG	CPU stall configuration register	0x00B8	R/W
RTC_CNTL_PAD_HOLD_REG	Configures the hold options for RTC GPIOs	0x00D4	R/W
RTC_CNTL_DIG_PAD_HOLD_REG	Configures the hold options for digital GPIOs	0x00D8	R/W
RTC_CNTL_BROWN_OUT_REG	Brownout configuration register	0x00E4	varies

28.8 Registers

The addresses in this section are relative to the Low Power Management base addresses provided in Section 28.6.

Register 28.1: RTC_CNTL_OPTIONS0_REG (0x0000)



RTC_CNTL_REG_SW_STALL_PROCPU_C0 When RTC_CNTL_REG_SW_STALL_PROCPU_C1 is configured to 0x21, setting this bit to 0x2 stalls the CPU by SW.

RTC_CNTL_SW_PROCPU_RST Set this bit to reset the CPU by SW. (WO)

RTC_CNTL_REG_BB_I2C_FORCE_PD Set this bit to FPD BB_I2C. (R/W)

RTC_CNTL_REG_BB_I2C_FORCE_PU Set this bit to FPU BB_I2C. (R/W)

RTC_CNTL_REG_BBPLL_I2C_FORCE_PD Set this bit to FPD BB_PLL _I2C. (R/W)

RTC_CNTL_REG_BBPLL_I2C_FORCE_PU Set this bit to FPU BB_PLL _I2C. (R/W)

RTC_CNTL_REG_BBPLL_FORCE_PD Set this bit to FPD BB_PLL. (R/W)

RTC_CNTL_REG_BBPLL_FORCE_PU Set this bit to FPU BB_PLL. (R/W)

RTC_CNTL_REG_XTL_FORCE_PD Set this bit to FPD the crystal oscillator. (R/W)

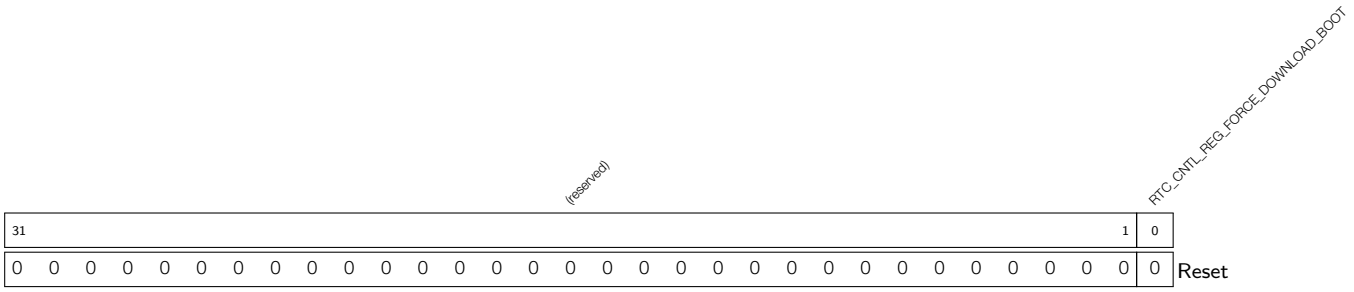
RTC_CNTL_REG_XTL_FORCE_PU Set this bit to FPU the crystal oscillator. (R/W)

RTC_CNTL_REG_DG_WRAP_FORCE_RST Set this bit to force reset the digital system in deep-sleep. (R/W)

RTC_CNTL_REG_DG_WRAP_FORCE_NORST Set this bit to disable force reset to digital system in deep-sleep. (R/W)

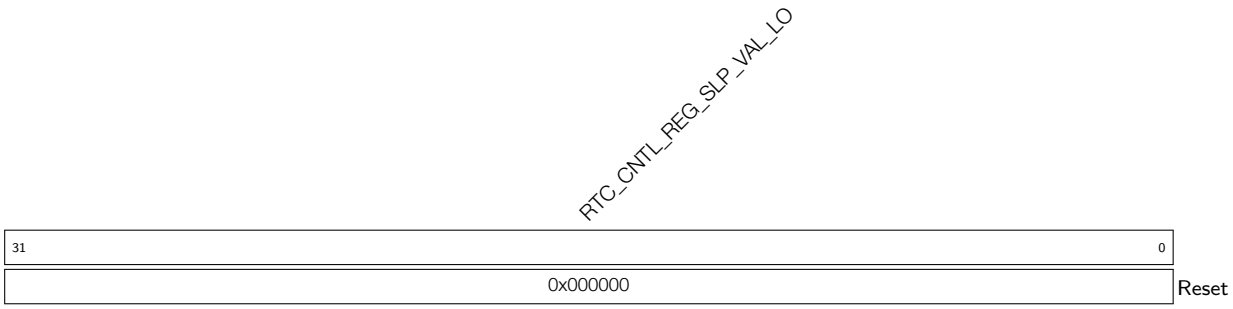
RTC_CNTL_SW_SYS_RST Set this bit to reset the system via SW. (WO)

Register 28.2: RTC_CNTL_OPTION1_REG (0x0128)



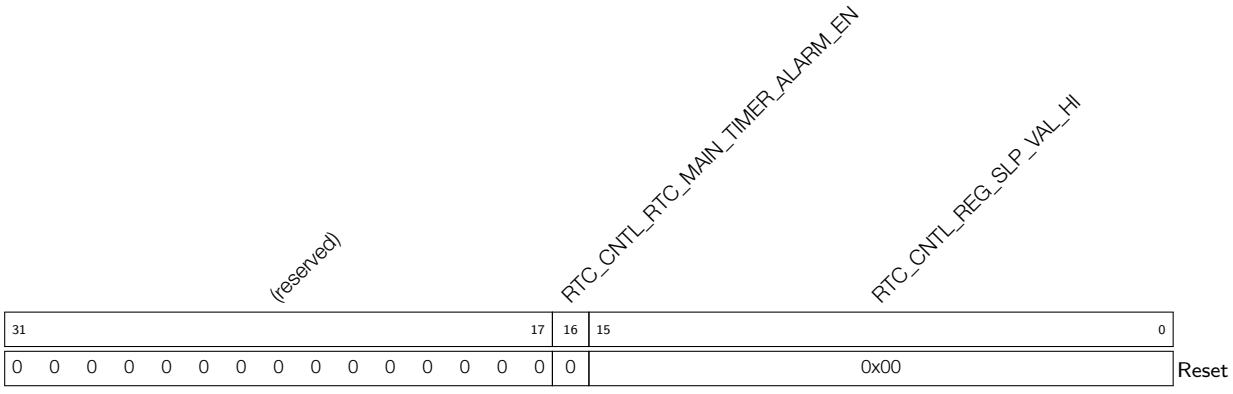
RTC_CNTL_REG_FORCE_DOWNLOAD_BOOT Set this bit to force the chip to boot from the download mode. (R/W)

Register 28.3: RTC_CNTL_SLP_TIMER0_REG (0x0004)



RTC_CNTL_REG_SLP_VAL_LO Sets the lower 32 bits of the trigger threshold for the RTC timer. (R/W)

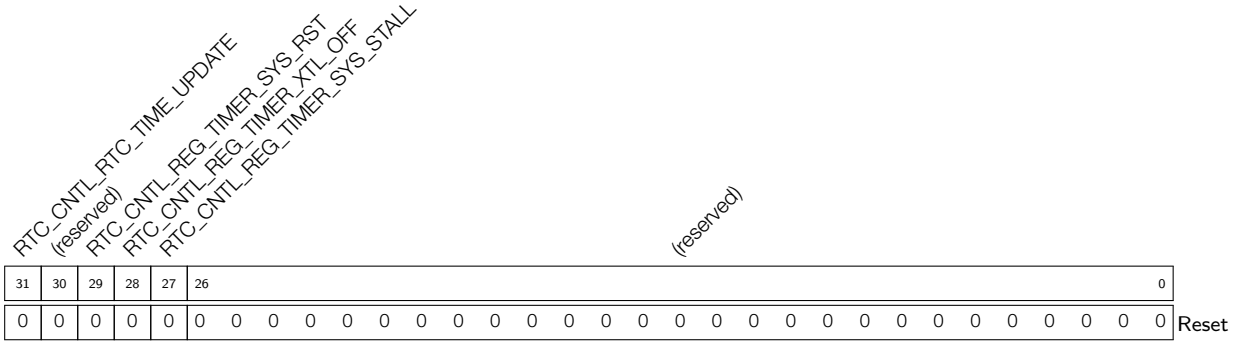
Register 28.4: RTC_CNTL_SLP_TIMER1_REG (0x0008)



RTC_CNTL_REG_SLP_VAL_HI Sets the higher 16 bits of the trigger threshold for the RTC timer. (R/W)

RTC_CNTL_RTC_MAIN_TIMER_ALARM_EN Sets this bit to enable the timer alarm. (WO)

Register 28.5: RTC_CNTL_TIME_UPDATE_REG (0x000C)



RTC_CNTL_REG_TIMER_SYS_STALL Selects the triggering condition for the RTC timer. (R/W)

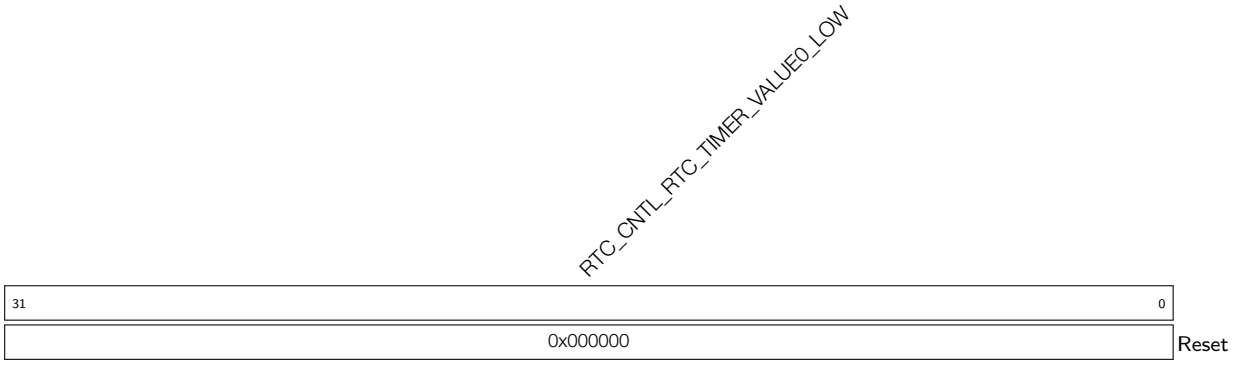
RTC_CNTL_REG_TIMER_XTL_OFF Selects the triggering condition for the RTC timer. (R/W)

RTC_CNTL_REG_TIMER_SYS_RST Selects the triggering condition for the RTC timer. (R/W)

RTC_CNTL_RTC_TIME_UPDATE Selects the triggering condition for the RTC timer. (WO)

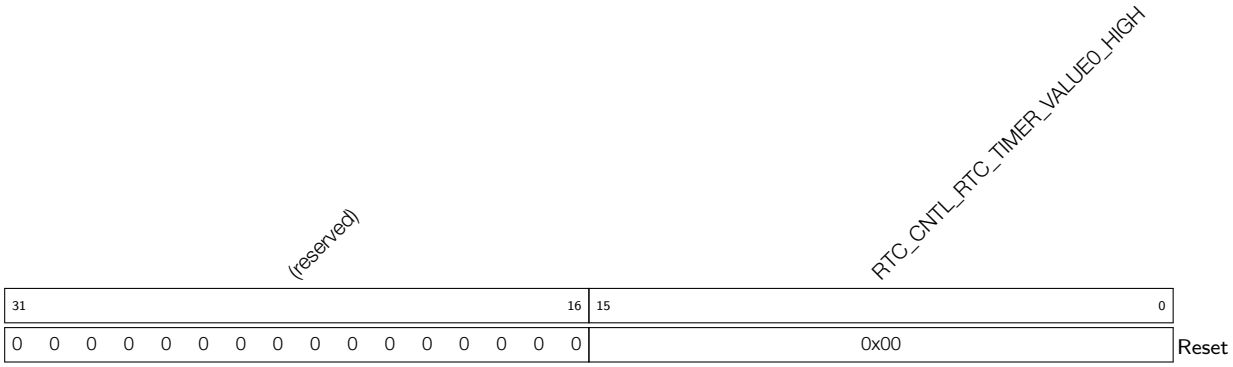
Note:
For details, please refer to Table 152.

Register 28.6: RTC_CNTL_TIME_LOW0_REG (0x0010)



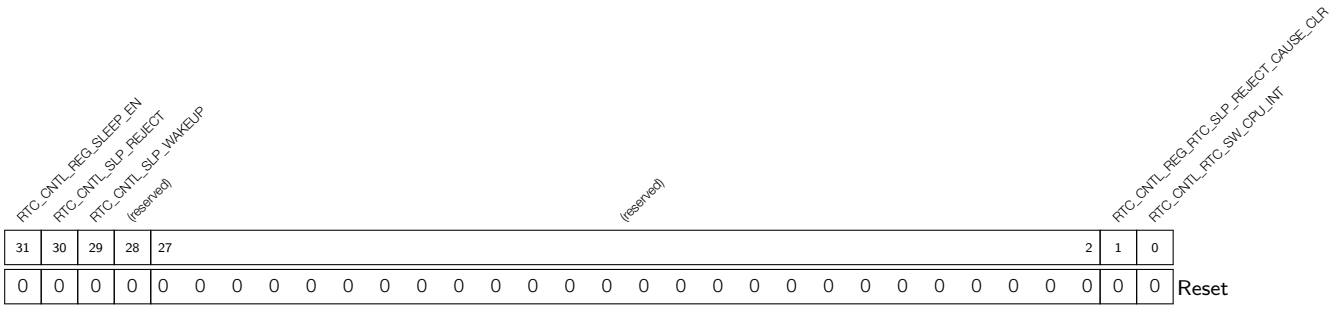
RTC_CNTL_RTC_TIMER_VALUE0_LOW Stores the lower 32 bits of RTC timer 0. (RO)

Register 28.7: RTC_CNTL_TIME_HIGH0_REG (0x0014)



RTC_CNTL_RTC_TIMER_VALUE0_HIGH Stores the higher 16 bits of RTC timer 0. (RO)

Register 28.8: RTC_CNTL_STATE0_REG (0x0018)



RTC_CNTL_RTC_SW_CPU_INT Sends a SW RTC interrupt to CPU. (WO)

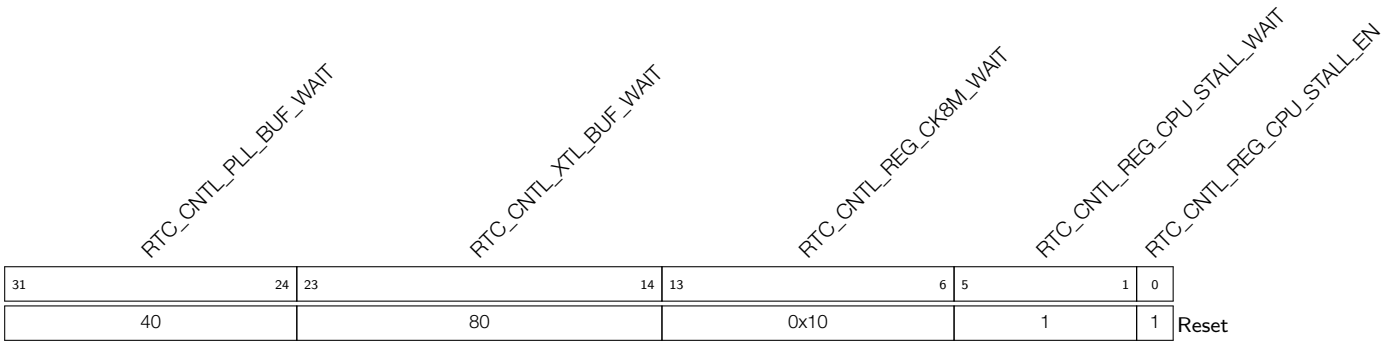
RTC_CNTL_REG_RTC_SLP_REJECT_CAUSE_CLR Clears the RTC reject-to-sleep cause. (WO)

RTC_CNTL_SLP_WAKEUP Sleep wakeup bit. (R/W)

RTC_CNTL_SLP_REJECT Sleep reject bit. (R/W)

RTC_CNTL_REG_SLEEP_EN Sends the chip to sleep. (R/W)

Register 28.9: RTC_CNTL_TIMER1_REG (0x001C)



RTC_CNTL_REG_CPU_STALL_EN Enables the CPU stalling. (R/W)

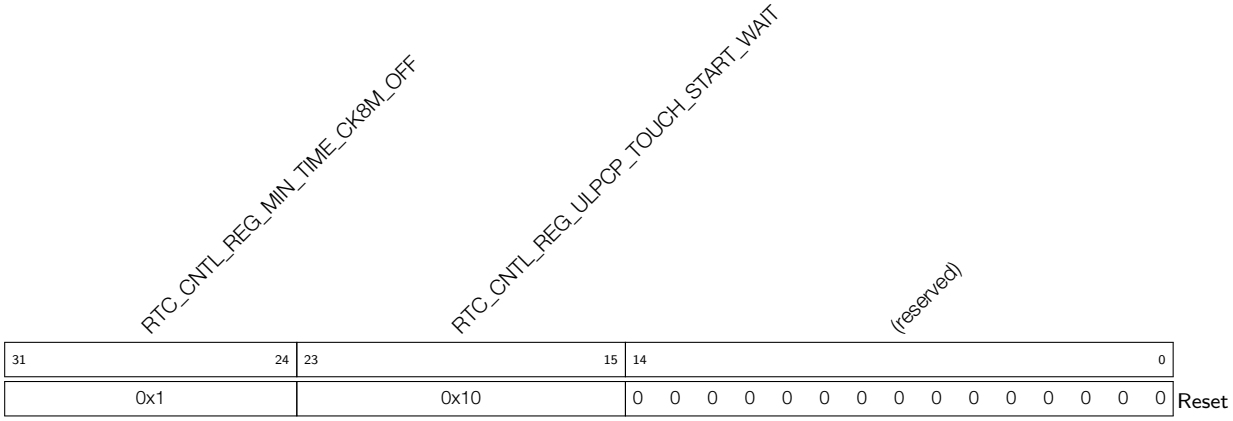
RTC_CNTL_REG_CPU_STALL_WAIT Sets the CPU stall waiting cycles (using the RTC fast clock). (R/W)

RTC_CNTL_REG_CK8M_WAIT Sets the 8 MHz clock waiting cycles (using the RTC slow clock). (R/W)

RTC_CNTL_XTL_BUF_WAIT Sets the XTAL waiting cycles (using the RTC slow clock). (R/W)

RTC_CNTL_PLL_BUF_WAIT Sets the PLL waiting cycles (using the RTC slow clock). (R/W)

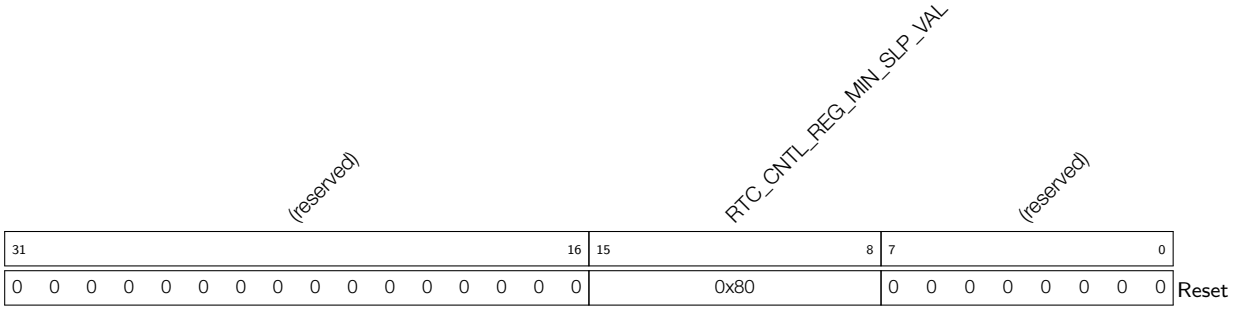
Register 28.10: RTC_CNTL_TIMER2_REG (0x0020)



RTC_CNTL_REG_ULPCP_TOUCH_START_WAIT Sets the waiting cycles (using the RTC slow clock) before the ULP co-processor or touch controller starts to work. (R/W)

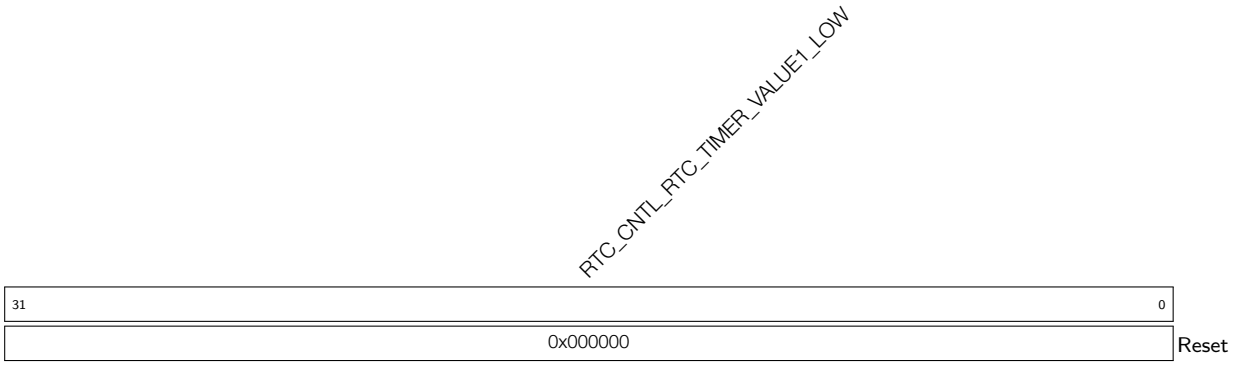
RTC_CNTL_REG_MIN_TIME_CK8M_OFF Sets the minimal cycles for 8 MHz clock (using the RTC slow clock) when powered down. (R/W)

Register 28.11: RTC_CNTL_TIMER5_REG (0x002C)



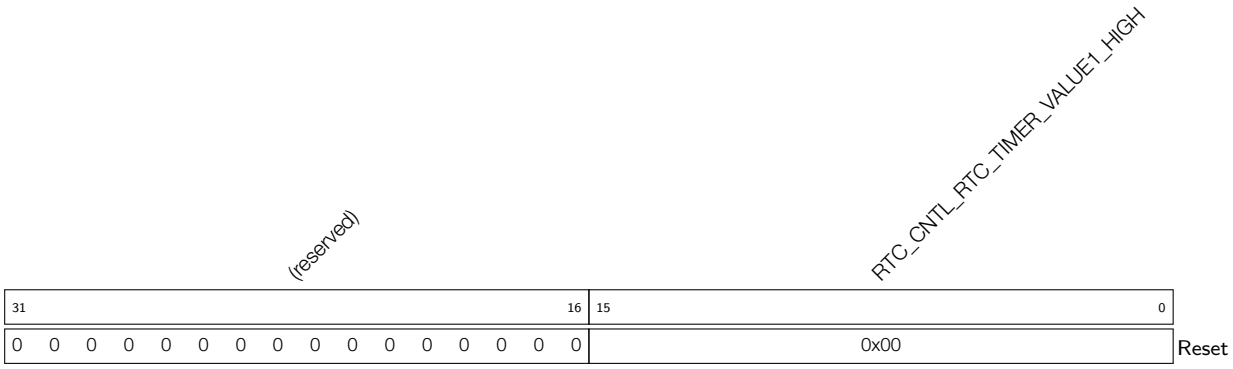
RTC_CNTL_REG_MIN_SLP_VAL Sets the minimal sleep cycles (using the RTC slow clock). (R/W)

Register 28.12: RTC_CNTL_TIME_LOW1_REG (0x00E8)



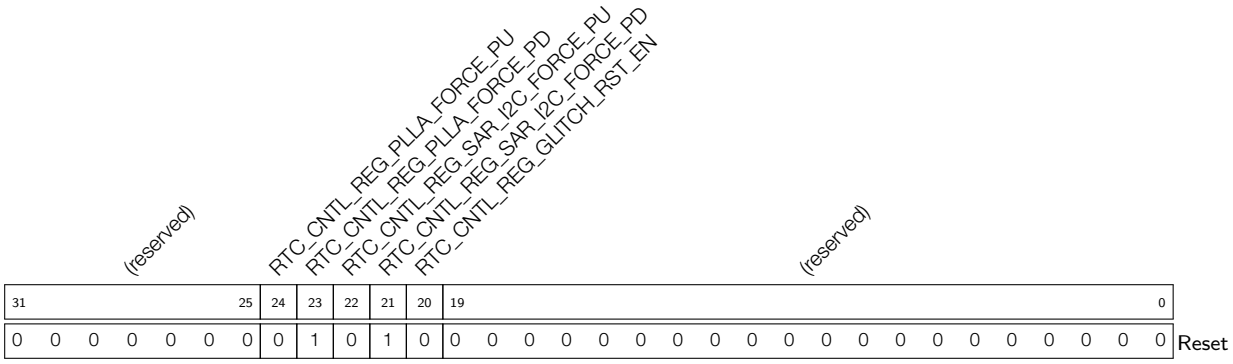
RTC_CNTL_RTC_TIMER_VALUE1_LOW Stores the lower 32 bits of RTC timer 1. (RO)

Register 28.13: RTC_CNTL_TIME_HIGH1_REG (0x00EC)



RTC_CNTL_RTC_TIMER_VALUE1_HIGH Stores the higher 16 bits of RTC timer. (RO)

Register 28.14: RTC_CNTL_ANA_CONF_REG (0x0034)



RTC_CNTL_REG_GLITCH_RST_EN Set this bit to enable a reset when the system detects a glitch. (R/W)

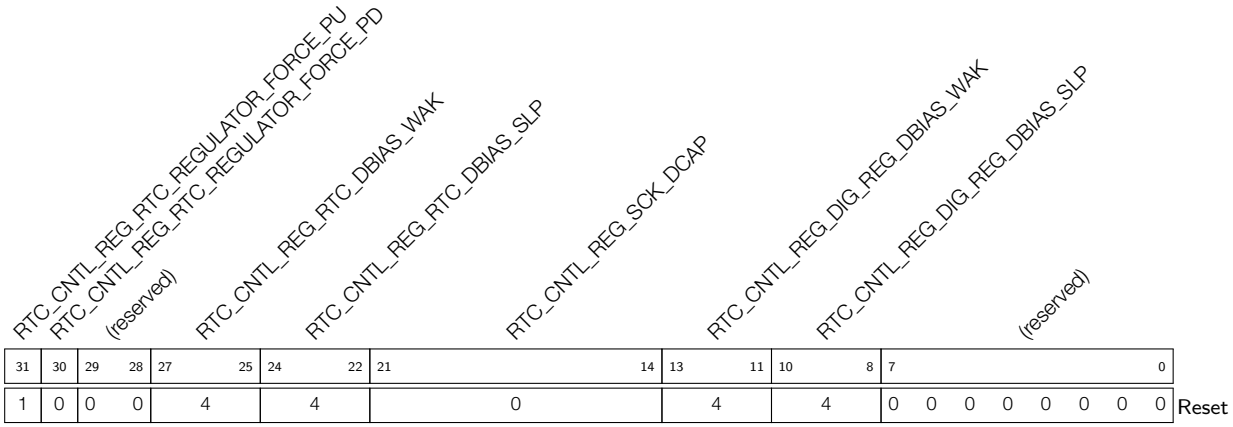
RTC_CNTL_REG_SAR_I2C_FORCE_PD Set this bit to FPD the SAR_I2C. (R/W)

RTC_CNTL_REG_SAR_I2C_FORCE_PU Set this bit to FPU the SAR_I2C. (R/W)

RTC_CNTL_REG_PLLA_FORCE_PD Set this bit to FPD the PLLA. (R/W)

RTC_CNTL_REG_PLLA_FORCE_PU Sets this bit to FPU the PLLA. (R/W)

Register 28.15: RTC_CNTL_REG_REG (0x0084)



RTC_CNTL_REG_DIG_REG_DBIAS_SLP Configures the regulation factor for the digital system voltage regulator when the CPU is in sleep state. (R/W)

RTC_CNTL_REG_DIG_REG_DBIAS_WAK Configures the regulation factor for the digital system voltage regulator when the CPU is in active status. (R/W)

RTC_CNTL_REG_SCK_DCAP Configures the frequency of the RTC clocks. (R/W)

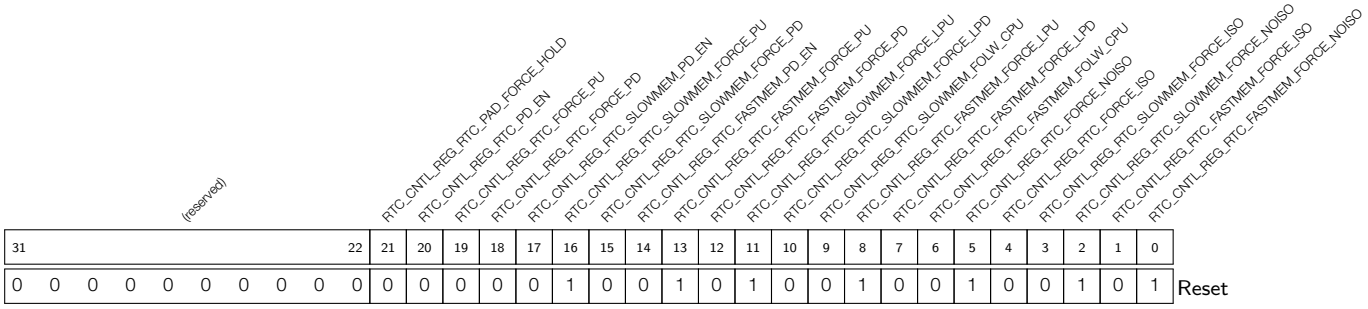
RTC_CNTL_REG_RTC_DBIAS_SLP Configures the regulation factor for the low-power voltage regulator when the CPU is in sleep status. (R/W)

RTC_CNTL_REG_RTC_DBIAS_WAK Configures the regulation factor for the low-power voltage regulator when the CPU is in active status. (R/W)

RTC_CNTL_REG_RTC_REGULATOR_FORCE_PD Set this bit to FPD the low-power voltage regulator, which means decreasing its voltage to 0.8 V or lower. (R/W)

RTC_CNTL_REG_RTC_REGULATOR_FORCE_PU Set this bit to FPU the low-power voltage regulator, which means increasing its voltage to higher than 0.8 V. (R/W)

Register 28.16: RTC_CNTL_PWC_REG (0x0088)



RTC_CNTL_REG_RTC_FASTMEM_FORCE_NOISO Set this bit to disable the force isolation of the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FORCE_ISO Set this bit to force isolation of the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_NOISO Set this bit to disable the force isolation of the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_ISO Set this bit to force isolation of the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_FORCE_ISO Set this bit to force isolation of the RTC peripherals. (R/W)

RTC_CNTL_REG_RTC_FORCE_NOISO Set this bit to disable the force isolation of the RTC peripherals. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FOLW_CPU Set this bit to FPD the RTC fast memory when the CPU is powered down. Reset this bit to FPD the RTC fast memory when the RTC main state machine is powered down. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FORCE_LPD Set this bit to force not retain the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FORCE_LPU Set this bit to force retain the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FOLW_CPU Set this bit to FPD the RTC slow memory when the CPU is powered down. Reset this bit to FPD the RTC slow memory when the RTC main state machine is powered down. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_LPD Set this bit to force not retain the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_LPU Set this bit to force retain the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FORCE_PD Set this bit to FPD the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_FORCE_PU Set this bit to FPU the RTC fast memory. (R/W)

RTC_CNTL_REG_RTC_FASTMEM_PD_EN Set this bit to enable PD for the RTC fast memory in sleep. (R/W)

Continued on the next page...

Register 28.16: RTC_CNTL_PWC_REG (0x0088)

Continued from the previous page...

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_PD Set this bit to FPD the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_FORCE_PU Set this bit to FPU the RTC slow memory. (R/W)

RTC_CNTL_REG_RTC_SLOWMEM_PD_EN Set this bit to enable PD for the RTC slow memory in sleep. (R/W)

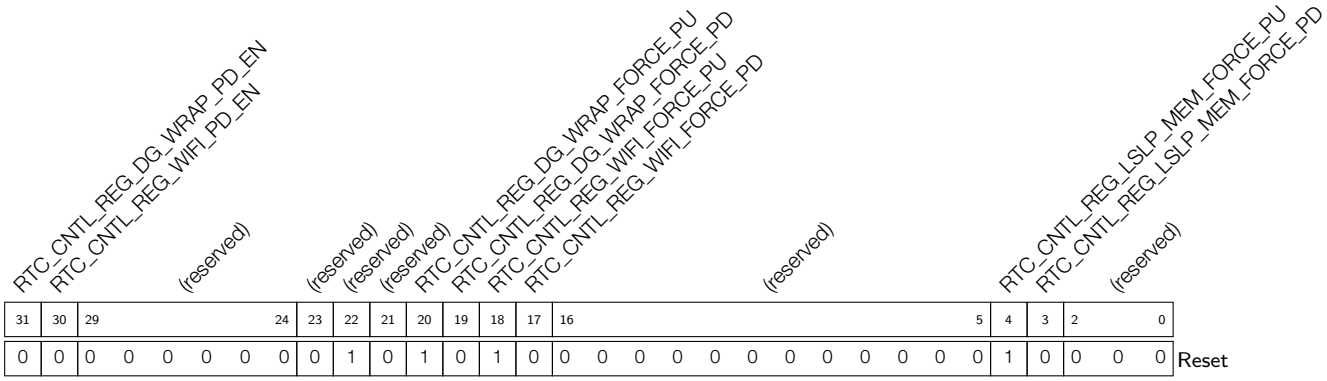
RTC_CNTL_REG_RTC_FORCE_PD Set this bit to FPD the RTC peripherals. (R/W)

RTC_CNTL_REG_RTC_FORCE_PU Set this bit to FPU the RTC peripherals. (R/W)

RTC_CNTL_REG_RTC_PD_EN Set this bit to enable PD for the RTC peripherals in sleep. (R/W)

RTC_CNTL_REG_RTC_PAD_FORCE_HOLD Set this bit the force hold the RTC GPIOs. (R/W)

Register 28.17: RTC_CNTL_DIG_PWC_REG (0x008C)



RTC_CNTL_REG_LSLP_MEM_FORCE_PD Set this bit to FPD the memories in the digital system in sleep. (R/W)

RTC_CNTL_REG_LSLP_MEM_FORCE_PU Set this bit to FPU the memories in the digital system. (R/W)

RTC_CNTL_REG_WIFI_FORCE_PD Set this bit to FPD the Wi-Fi digital circuit in the digital system. (R/W)

RTC_CNTL_REG_WIFI_FORCE_PU Set this bit to FPU the Wi-Fi digital circuit in the digital system. (R/W)

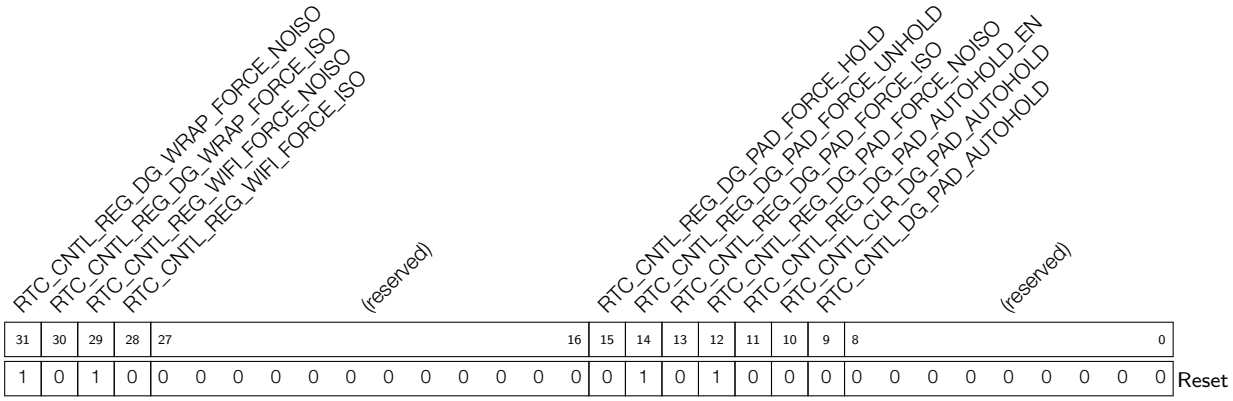
RTC_CNTL_REG_DG_WRAP_FORCE_PD Set this bit to FPD the digital system. (R/W)

RTC_CNTL_REG_DG_WRAP_FORCE_PU Set this bit to FPU the digital system. (R/W)

RTC_CNTL_REG_WIFI_PD_EN Set this bit to enable PD for the Wi-Fi digital circuit in sleep. (R/W)

RTC_CNTL_REG_DG_WRAP_PD_EN Set this bit to enable PD for the digital system in sleep. (R/W)

Register 28.18: RTC_CNTL_DIG_ISO_REG (0x0090)



RTC_CNTL_DG_PAD_AUTOHOLD Indicates the auto-hold status of the digital GPIOs. (RO)

RTC_CNTL_CLR_DG_PAD_AUTOHOLD Set this bit to clear the auto-hold enabler for the digital GPIOs. (WO)

RTC_CNTL_REG_DG_PAD_AUTOHOLD_EN Set this bit to allow the digital GPIOs to enter the auto-hold status. (R/W)

RTC_CNTL_REG_DG_PAD_FORCE_NOISO Set this bit to disable the force isolation of the digital GPIOs. (R/W)

RTC_CNTL_REG_DG_PAD_FORCE_ISO Set this bit to force isolation of the digital GPIOs. (R/W)

RTC_CNTL_REG_DG_PAD_FORCE_UNHOLD Set this bit the force unhold the digital GPIOs. (R/W)

RTC_CNTL_REG_DG_PAD_FORCE_HOLD Set this bit the force hold the digital GPIOs. (R/W)

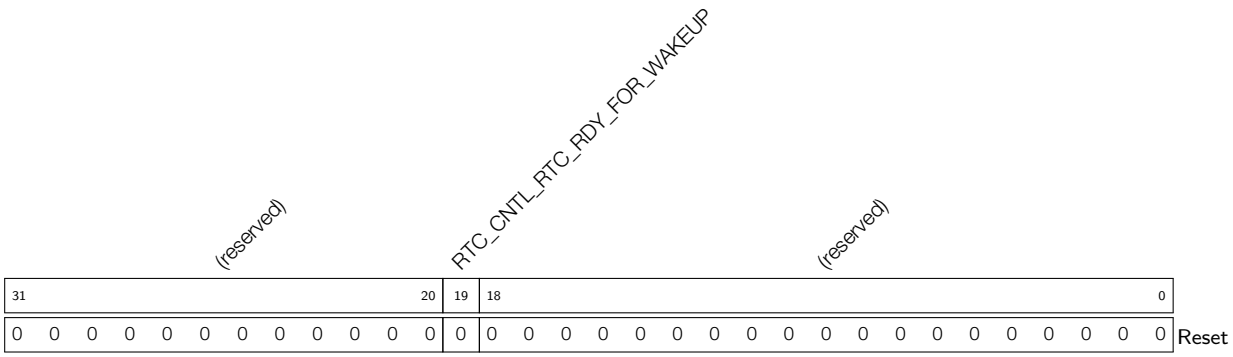
RTC_CNTL_REG_WIFI_FORCE_ISO Set this bit to force isolation of the Wi-Fi digital circuits. (R/W)

RTC_CNTL_REG_WIFI_FORCE_NOISO Set this bit to disable the force isolation of the Wi-Fi digital circuits. (R/W)

RTC_CNTL_REG_DG_WRAP_FORCE_ISO Set this bit to force isolation of the digital system. (R/W)

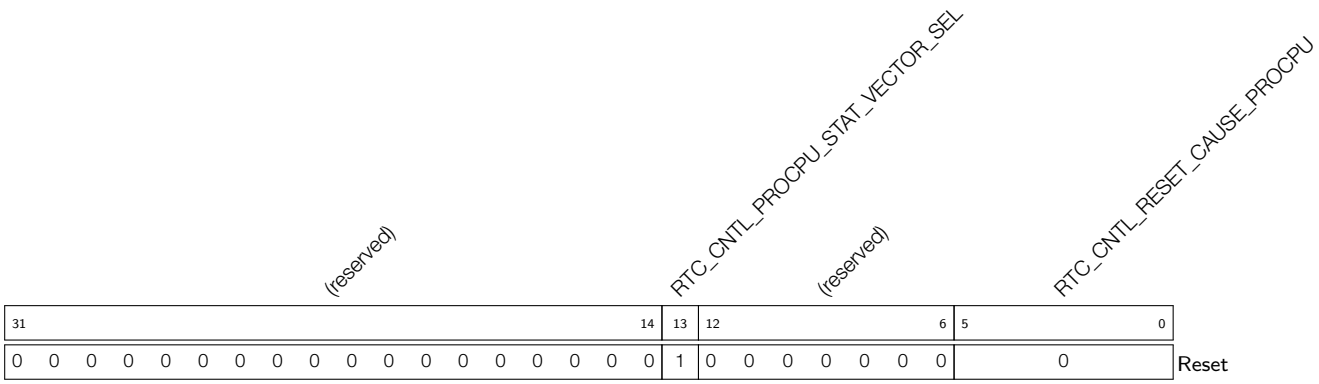
RTC_CNTL_REG_DG_WRAP_FORCE_NOISO Set this bit to disable the force isolation of the digital system. (R/W)

Register 28.19: RTC_CNTL_LOW_POWER_ST_REG (0x00CC)



RTC_CNTL_RTC_RDY_FOR_WAKEUP Indicates the RTC is ready to be triggered by any wakeup source. (RO)

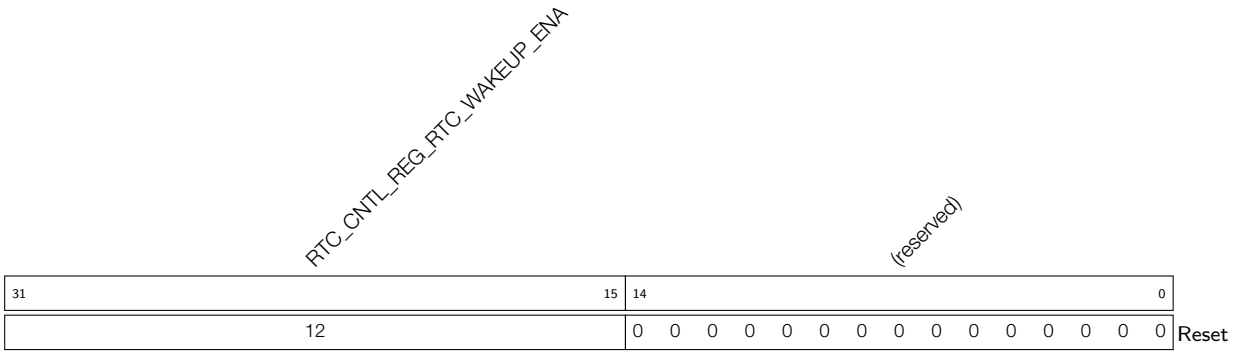
Register 28.20: RTC_CNTL_RESET_STATE_REG (0x0038)



RTC_CNTL_RESET_CAUSE_PROCPU Stores the CPU reset cause. (RO)

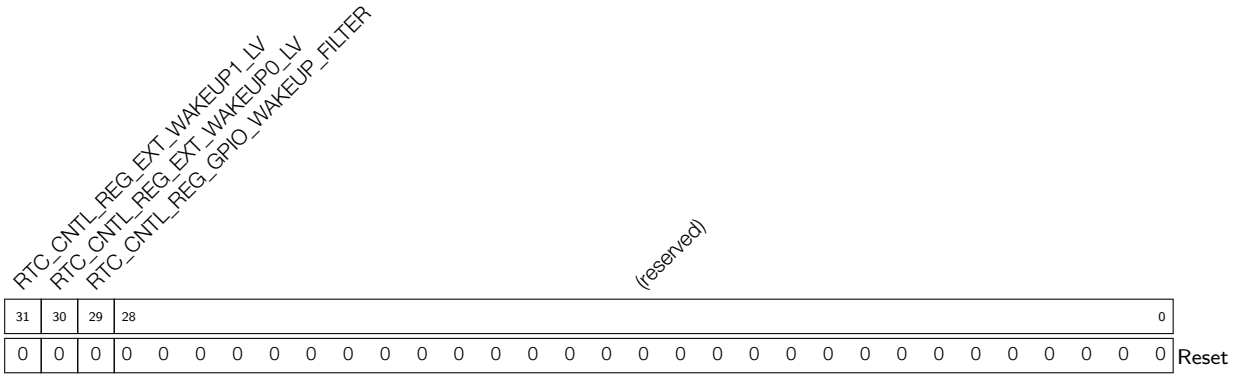
RTC_CNTL_PROCPU_STAT_VECTOR_SEL Selects the CPU static vector. (R/W)

Register 28.21: RTC_CNTL_WAKEUP_STATE_REG (0x003C)



RTC_CNTL_REG_RTC_WAKEUP_ENA Selects the wakeup source. For details, please refer to Table 156. (R/W)

Register 28.22: RTC_CNTL_EXT_WAKEUP_CONF_REG (0x0064)



RTC_CNTL_REG_GPIO_WAKEUP_FILTER Set this bit to enable the GPIO wakeup event filter. (R/W)

RTC_CNTL_REG_EXT_WAKEUP0_LV 0: EXT0 at low level, 1: EXT0 at high level. (R/W)

RTC_CNTL_REG_EXT_WAKEUP1_LV 0: EXT1 at low level, 1: EXT1 at high level. (R/W)

Register 28.23: RTC_CNTL_SLP_REJECT_CONF_REG (0x0068)

RTC_CNTL_REG_RTC_SLEEP_REJECT_ENA Set this bit to enable reject-to-sleep. (R/W)

RTC_CNTL_REG_LIGHT_SLP_REJECT_EN Set this bit to enable reject-to-light-sleep. (R/W)

RTC_CNTL_REG_DEEP_SLP_REJECT_EN Set this bit to enable reject-to-deep-sleep. (R/W)

Register 28.24: RTC_CNTL_EXT_WAKEUP1_REG (0x00DC)

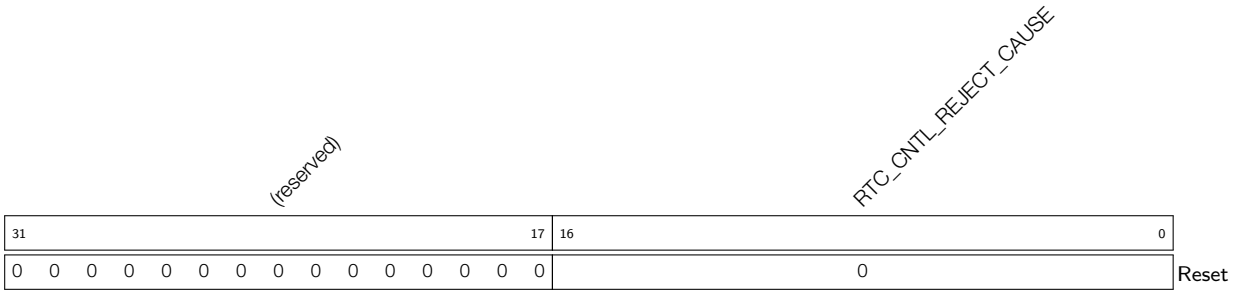
RTC_CNTL_REG_EXT_WAKEUP1_SEL Selects a RTC GPIO to be the EXT1 wakeup source. (R/W)

RTC_CNTL_REG_EXT_WAKEUP1_STATUS_CLR Clears the EXT1 wakeup status. (WO)

Register 28.25: RTC_CNTL_EXT_WAKEUP1_STATUS_REG (0x00E0)

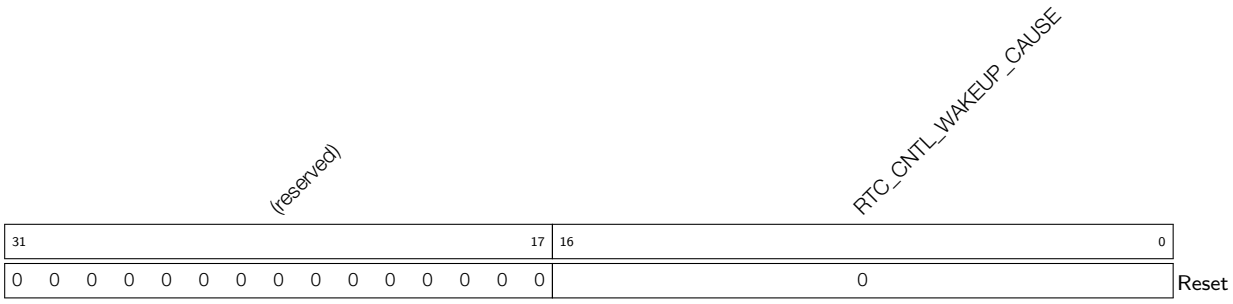
RTC_CNTL_REG_EXT_WAKEUP1_STATUS Indicates the EXT1 wakeup status. (RO)

Register 28.26: RTC_CNTL_SLP_REJECT_CAUSE_REG (0x0124)



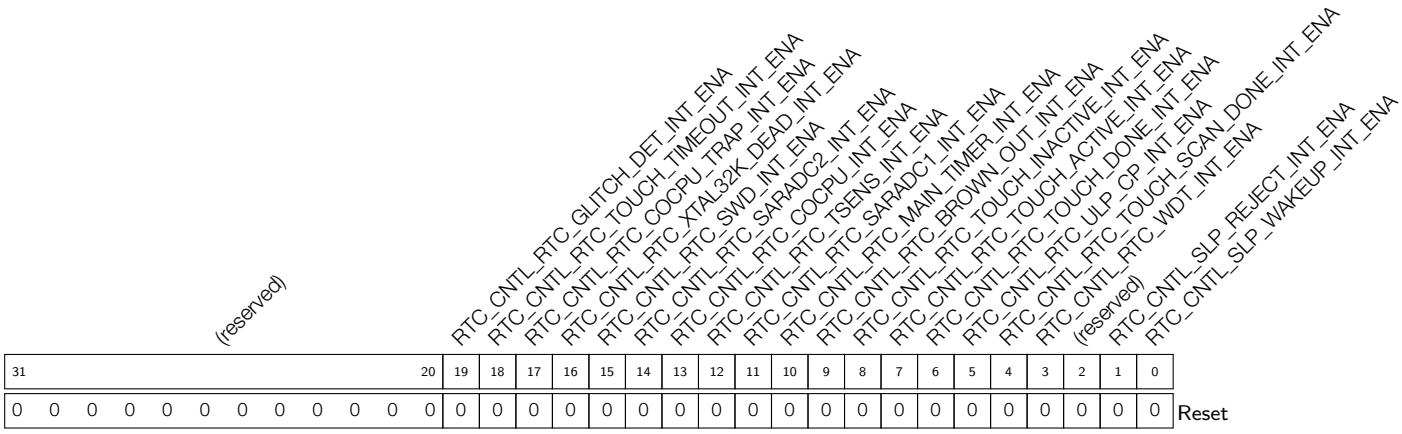
RTC_CNTL_REJECT_CAUSE Stores the reject-to-sleep cause. (RO)

Register 28.27: RTC_CNTL_SLP_WAKEUP_CAUSE_REG (0x012C)



RTC_CNTL_WAKEUP_CAUSE Stores the wakeup cause. (RO)

Register 28.28: RTC_CNTL_INT_ENA_RTC_REG (0x0040)



RTC_CNTL_SLP_WAKEUP_INT_ENA Enables interrupts when the chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA Enables interrupts when the chip rejects to go to sleep. (R/W)

RTC_CNTL_RTC_WDT_INT_ENA Enables the RTC watchdog interrupt. (R/W)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_ENA Enables interrupts upon the completion of a touch scanning. (R/W)

RTC_CNTL_RTC_ULP_CP_INT_ENA Enables the ULP co-processor interrupt. (R/W)

Continued on the next page...

Register 28.28: RTC_CNTL_INT_ENA_RTC_REG (0x0040)

Continued from the previous page...

RTC_CNTL_RTC_TOUCH_DONE_INT_ENA Enables interrupts upon the completion of a single touch. (R/W)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_ENA Enables interrupts when a touch is detected. (R/W)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_ENA Enables interrupts when a touch is released. (R/W)

RTC_CNTL_RTC_BROWN_OUT_INT_ENA Enables the brownout interrupt. (R/W)

RTC_CNTL_RTC_MAIN_TIMER_INT_ENA Enables the RTC main timer interrupt. (R/W)

RTC_CNTL_RTC_SARADC1_INT_ENA Enables the SAR ADC 1 interrupt. (R/W)

RTC_CNTL_RTC_TSENS_INT_ENA Enables the temperature sensor interrupt. (R/W)

RTC_CNTL_RTC_COCPU_INT_ENA Enables the ULP-RISCV interrupt. (R/W)

RTC_CNTL_RTC_SARADC2_INT_ENA Enables the SAR ADC 2 interrupt. (R/W)

RTC_CNTL_RTC_SWD_INT_ENA Enables the super watchdog interrupt. (R/W)

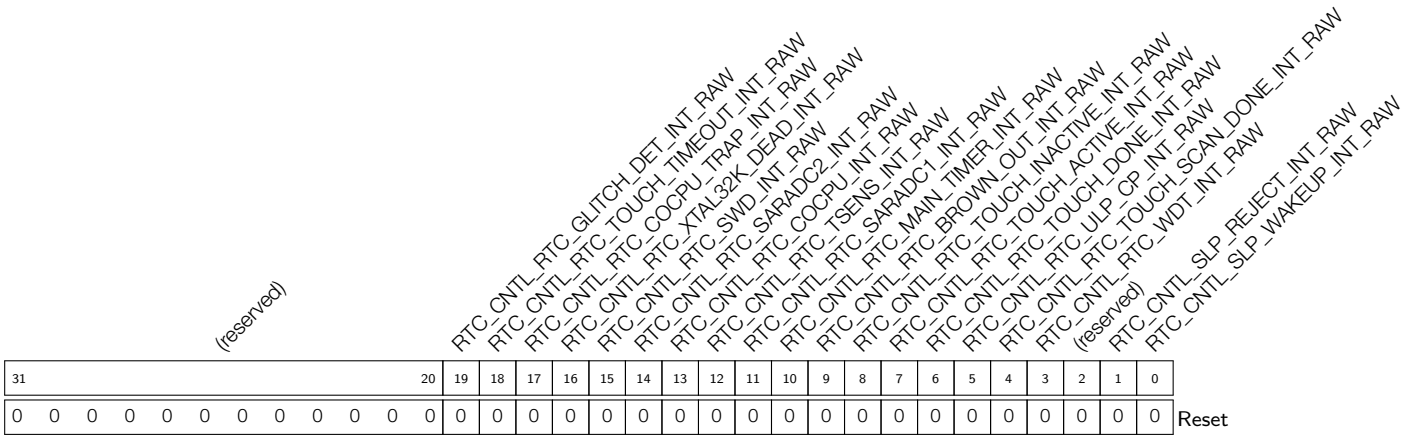
RTC_CNTL_RTC_XTAL32K_DEAD_INT_ENA Enables interrupts when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_RTC_COCPU_TRAP_INT_ENA Enables interrupts when the ULP-RISCV is trapped. (R/W)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_ENA Enables interrupts when touch sensor times out. (R/W)

RTC_CNTL_RTC_GLITCH_DET_INT_ENA Enables interrupts when a glitch is detected. (R/W)

Register 28.29: RTC_CNTL_INT_RAW_RTC_REG (0x0044)



RTC_CNTL_SLP_WAKEUP_INT_RAW Stores the raw interrupt triggered when the chip wakes up from sleep. (RO)

RTC_CNTL_SLP_REJECT_INT_RAW Stores the raw interrupt triggered when the chip rejects to go to sleep. (RO)

RTC_CNTL_RTC_WDT_INT_RAW Stores the raw RTC watchdog interrupt. (RO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_RAW Stores the raw interrupt triggered upon the completion of a touch scanning. (RO)

RTC_CNTL_RTC_ULP_CP_INT_RAW Stores the raw ULP co-processor interrupt. (RO)

RTC_CNTL_RTC_TOUCH_DONE_INT_RAW Stores the raw interrupt triggered upon the completion of a single touch. (RO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_RAW Stores the raw interrupt triggered when a touch is detected. (RO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_RAW Stores the raw interrupt triggered when a touch is released. (RO)

RTC_CNTL_RTC_BROWN_OUT_INT_RAW Stores the raw brownout interrupt. (RO)

RTC_CNTL_RTC_MAIN_TIMER_INT_RAW Stores the raw RTC main timer interrupt. (RO)

RTC_CNTL_RTC_SARADC1_INT_RAW Stores the raw SAR ADC 1 interrupt. (RO)

RTC_CNTL_RTC_TSENS_INT_RAW Stores the raw temperature sensor interrupt. (RO)

RTC_CNTL_RTC_COCPU_INT_RAW Stores the raw ULP-RISCV interrupt. (RO)

RTC_CNTL_RTC_SARADC2_INT_RAW Stores the raw SAR ADC 2 interrupt. (RO)

RTC_CNTL_RTC_SWD_INT_RAW Stores the raw super watchdog interrupt. (RO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_RAW Stores the raw interrupt triggered when the 32 kHz crystal is dead. (RO)

Continued on the next page...

Register 28.29: RTC_CNTL_INT_RAW_RTC_REG (0x0044)

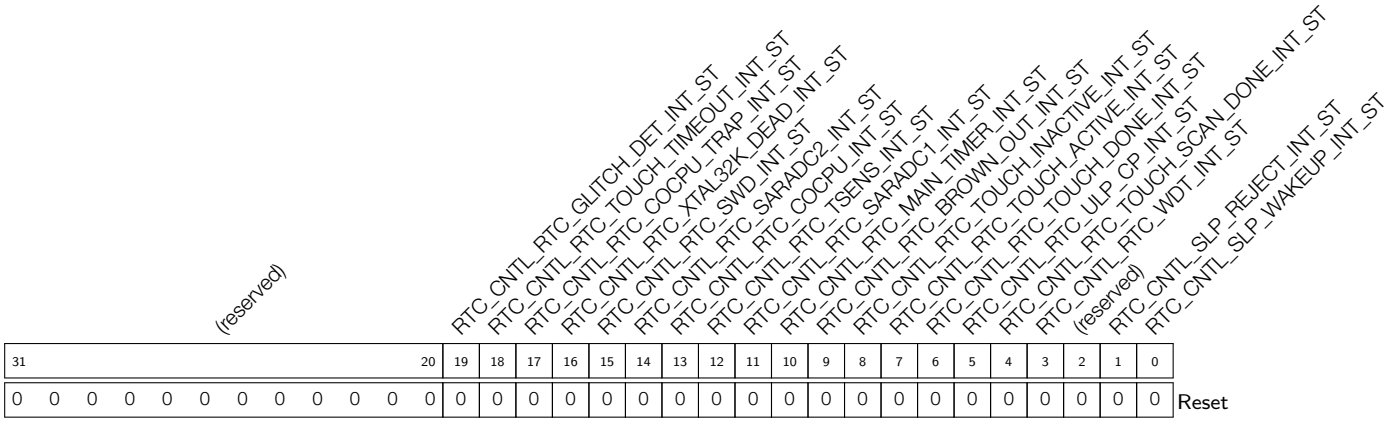
Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_RAW Stores the raw interrupt triggered when the ULP-RISCV is trapped. (RO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_RAW Stores the raw interrupt triggered when touch sensor times out. (RO)

RTC_CNTL_RTC_GLITCH_DET_INT_RAW Stores the raw interrupt triggered when a glitch is detected. (RO)

Register 28.30: RTC_CNTL_INT_ST_RTC_REG (0x0048)



RTC_CNTL_SLP_WAKEUP_INT_ST Stores the status of the interrupt triggered when the chip wakes up from sleep. (RO)

RTC_CNTL_SLP_REJECT_INT_ST Stores the status of the interrupt triggered when the chip rejects to go to sleep. (RO)

RTC_CNTL_RTC_WDT_INT_ST Stores the status of the RTC watchdog interrupt. (RO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_ST Stores the status of the interrupt triggered upon the completion of a touch scanning. (RO)

RTC_CNTL_RTC_ULP_CP_INT_ST Stores the status of the ULP co-processor interrupt. (RO)

RTC_CNTL_RTC_TOUCH_DONE_INT_ST Stores the status of the interrupt triggered upon the completion of a single touch. (RO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_ST Stores the status of the interrupt triggered when a touch is detected. (RO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_ST Stores the status of the interrupt triggered when a touch is released. (RO)

RTC_CNTL_RTC_BROWN_OUT_INT_ST Stores the status of the brownout interrupt. (RO)

RTC_CNTL_RTC_MAIN_TIMER_INT_ST Stores the status of the RTC main timer interrupt. (RO)

RTC_CNTL_RTC_SARADC1_INT_ST Stores the status of the SAR ADC 1 interrupt. (RO)

RTC_CNTL_RTC_TSENS_INT_ST Stores the status of the temperature sensor interrupt. (RO)

RTC_CNTL_RTC_COCPU_INT_ST Stores the status of the ULP-RISCV interrupt. (RO)

RTC_CNTL_RTC_SARADC2_INT_ST Stores the status of the SAR ADC 2 interrupt. (RO)

RTC_CNTL_RTC_SWD_INT_ST Stores the status of the super watchdog interrupt. (RO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_ST Stores the status of the interrupt triggered when the 32 kHz crystal is dead. (RO)

Continued on the next page...

Register 28.30: RTC_CNTL_INT_ST_RTC_REG (0x0048)

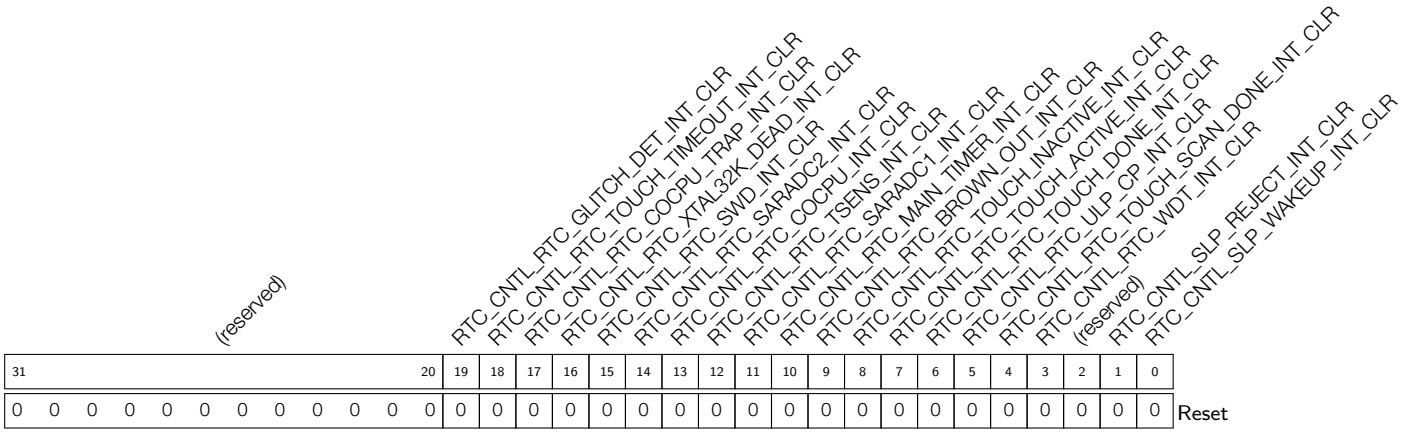
Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_ST Stores the status of the interrupt triggered when the ULP-RISCV is trapped. (RO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_ST Stores the status of the interrupt triggered when touch sensor times out. (RO)

RTC_CNTL_RTC_GLITCH_DET_INT_ST Stores the status of the interrupt triggered when a glitch is detected. (RO)

Register 28.31: RTC_CNTL_INT_CLR_RTC_REG (0x004C)



RTC_CNTL_SLP_WAKEUP_INT_CLR Clears the interrupt triggered when the chip wakes up from sleep. (WO)

RTC_CNTL_SLP_REJECT_INT_CLR Clears the interrupt triggered when the chip rejects to go to sleep. (WO)

RTC_CNTL_RTC_WDT_INT_CLR Clears the RTC watchdog interrupt. (WO)

RTC_CNTL_RTC_TOUCH_SCAN_DONE_INT_CLR Clears the interrupt triggered upon the completion of a touch scanning. (WO)

RTC_CNTL_RTC_ULP_CP_INT_CLR Clears the ULP co-processor interrupt. (WO)

RTC_CNTL_RTC_TOUCH_DONE_INT_CLR Clears the interrupt triggered upon the completion of a single touch. (WO)

RTC_CNTL_RTC_TOUCH_ACTIVE_INT_CLR Clears the interrupt triggered when a touch is detected. (WO)

RTC_CNTL_RTC_TOUCH_INACTIVE_INT_CLR Clears the interrupt triggered when a touch is released. (WO)

RTC_CNTL_RTC_BROWN_OUT_INT_CLR Clears the brownout interrupt. (WO)

RTC_CNTL_RTC_MAIN_TIMER_INT_CLR Clears the RTC main timer interrupt. (WO)

RTC_CNTL_RTC_SARADC1_INT_CLR Clears the SAR ADC 1 interrupt. (WO)

RTC_CNTL_RTC_TSENS_INT_CLR Clears the temperature sensor interrupt. (WO)

RTC_CNTL_RTC_COCPU_INT_CLR Clears the ULP-RISCV interrupt. (WO)

RTC_CNTL_RTC_SARADC2_INT_CLR Clears the SAR ADC 2 interrupt. (WO)

RTC_CNTL_RTC_SWD_INT_CLR Clears the super watchdog interrupt. (WO)

RTC_CNTL_RTC_XTAL32K_DEAD_INT_CLR Clears the interrupt triggered when the 32 kHz crystal is dead. (WO)

Continued on the next page...

Register 28.31: RTC_CNTL_INT_CLR_RTC_REG (0x004C)

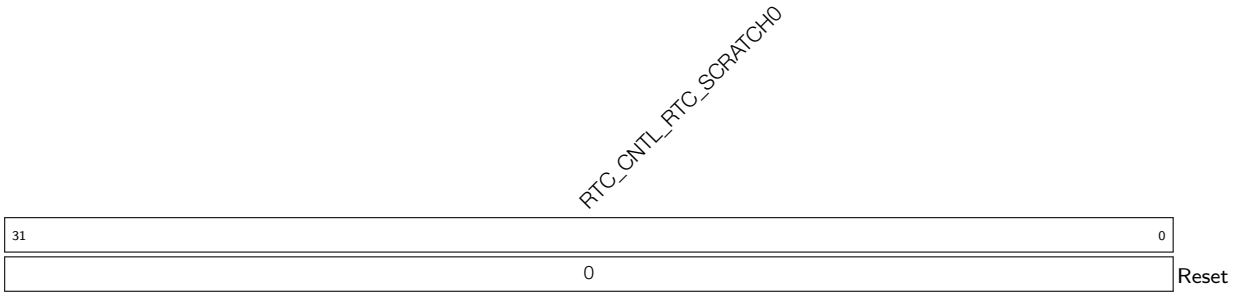
Continued from the previous page...

RTC_CNTL_RTC_COCPU_TRAP_INT_CLR Clears the interrupt triggered when the ULP-RISCV is trapped. (WO)

RTC_CNTL_RTC_TOUCH_TIMEOUT_INT_CLR Clears the interrupt triggered when touch sensor times out. (WO)

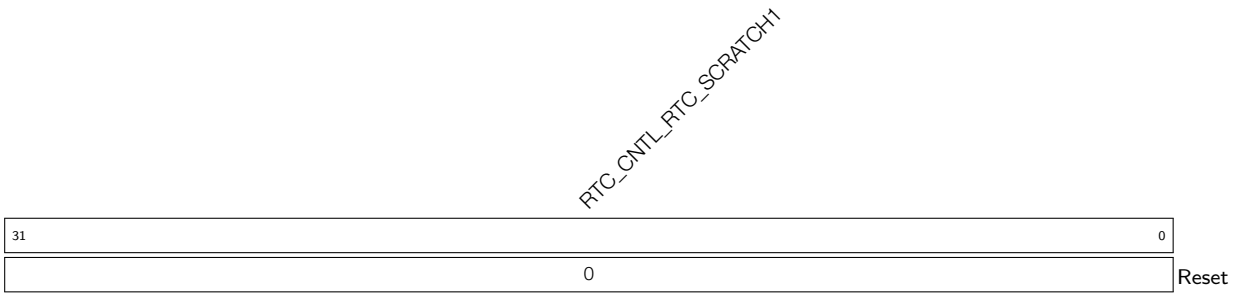
RTC_CNTL_RTC_GLITCH_DET_INT_CLR Clears the interrupt triggered when a glitch is detected. (WO)

Register 28.32: RTC_CNTL_STORE0_REG (0x0050)



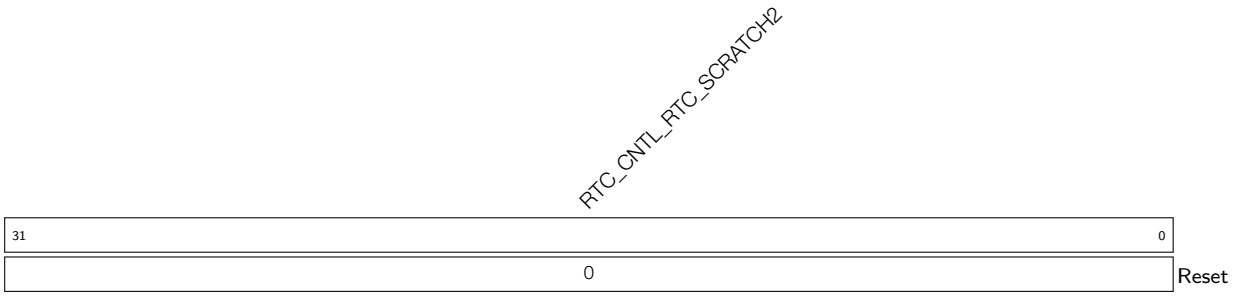
RTC_CNTL_RTC_SCRATCH0 Reservation register 0. (R/W)

Register 28.33: RTC_CNTL_STORE1_REG (0x0054)



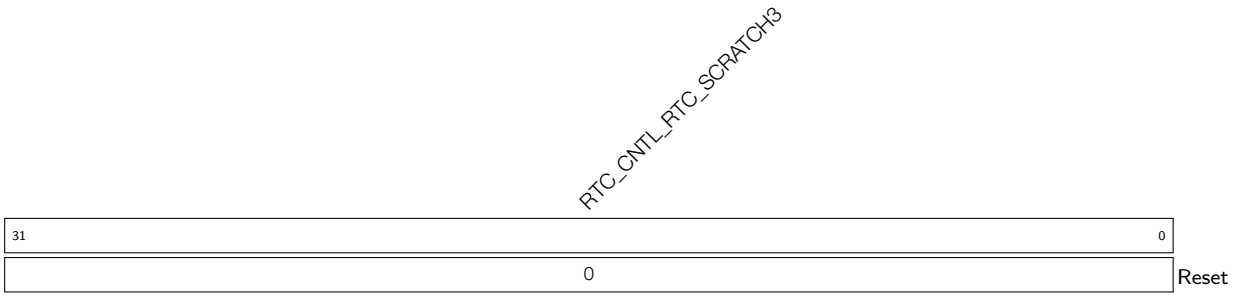
RTC_CNTL_RTC_SCRATCH1 Reservation register 1. (R/W)

Register 28.34: RTC_CNTL_STORE2_REG (0x0058)



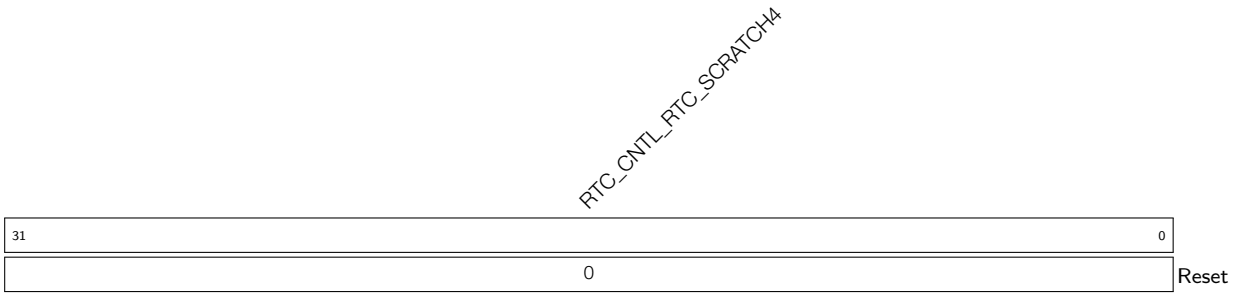
RTC_CNTL_RTC_SCRATCH2 Reservation register 2. (R/W)

Register 28.35: RTC_CNTL_STORE3_REG (0x005C)



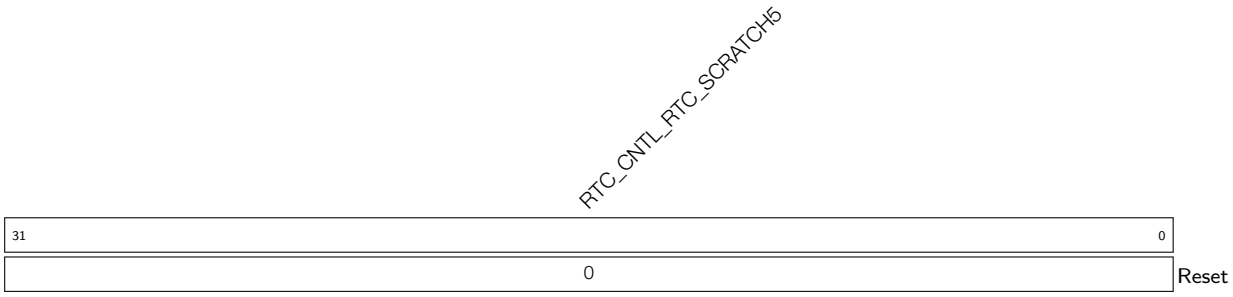
RTC_CNTL_RTC_SCRATCH3 Reservation register 3. (R/W)

Register 28.36: RTC_CNTL_STORE4_REG (0x00BC)



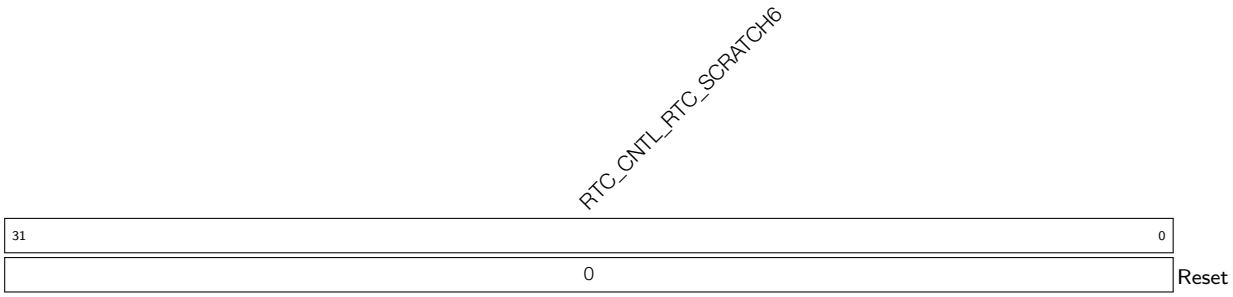
RTC_CNTL_RTC_SCRATCH4 Reservation register 4. (R/W)

Register 28.37: RTC_CNTL_STORE5_REG (0x00C0)



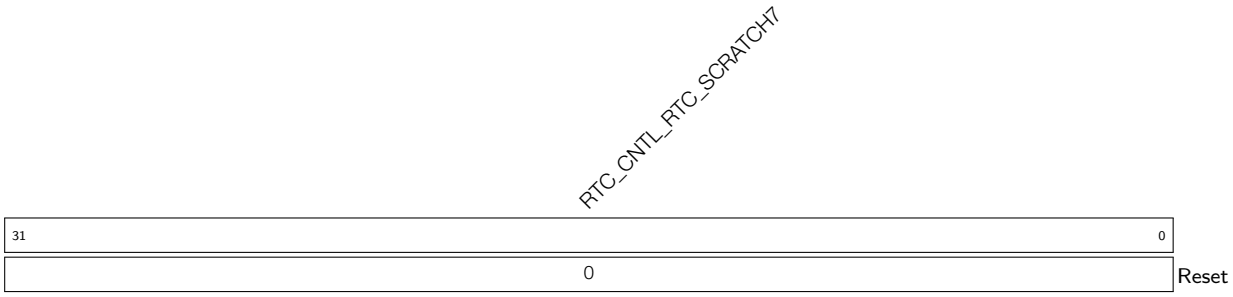
RTC_CNTL_RTC_SCRATCH5 Reservation register 5. (R/W)

Register 28.38: RTC_CNTL_STORE6_REG (0x00C4)



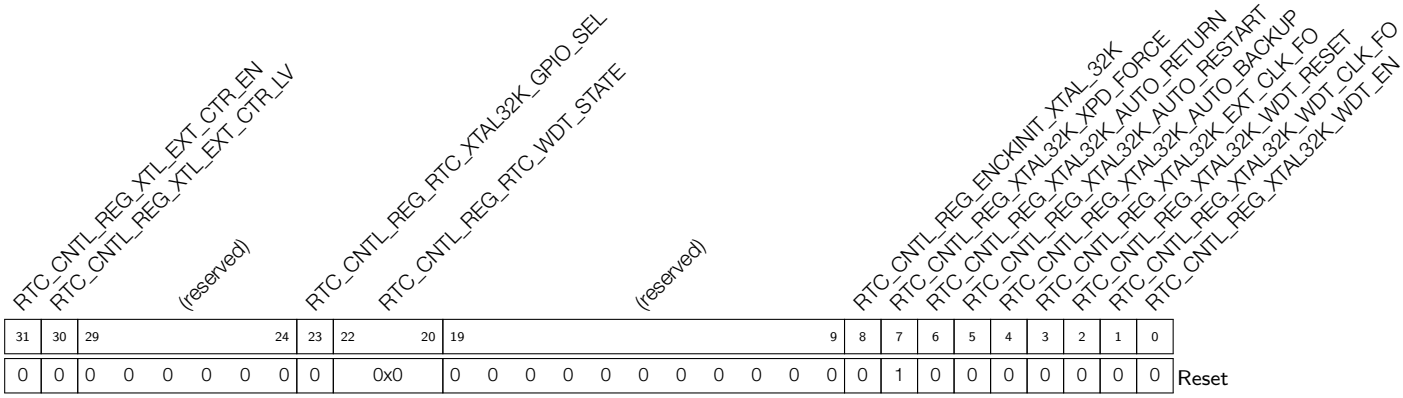
RTC_CNTL_RTC_SCRATCH6 Reservation register 6. (R/W)

Register 28.39: RTC_CNTL_STORE7_REG (0x00C8)



RTC_CNTL_RTC_SCRATCH7 Reservation register 7. (R/W)

Register 28.40: RTC_CNTL_EXT_XTL_CONF_REG (0x0060)



RTC_CNTL_REG_XTAL32K_WDT_EN Set this bit to enable the 32 kHz crystal watchdog. (R/W)

RTC_CNTL_REG_XTAL32K_WDT_CLK_FO Set this bit to FPU the 32 kHz crystal watchdog clock. (R/W)

RTC_CNTL_REG_XTAL32K_WDT_RESET Set this bit to reset the 32 kHz crystal watchdog by SW. (R/W)

RTC_CNTL_REG_XTAL32K_EXT_CLK_FO Set this bit to FPU the external clock of 32 kHz crystal. (R/W)

RTC_CNTL_REG_XTAL32K_AUTO_BACKUP Set this bit to switch to the backup clock when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_REG_XTAL32K_AUTO_RESTART Set this bit to restart the 32 kHz crystal automatically when the 32 kHz crystal is dead. (R/W)

RTC_CNTL_REG_XTAL32K_AUTO_RETURN Set this bit to switch back to 32 kHz crystal when the 32 kHz crystal is restarted. (R/W)

RTC_CNTL_REG_XTAL32K_XPD_FORCE Set this bit to allow the software to FPD the 32 kHz crystal; Reset this bit to allow the FSM to FPD the 32 kHz crystal. (R/W)

RTC_CNTL_REG_ENCKINIT_XTAL_32K Set 1 to apply an internal clock to help the 32 kHz crystal to start. (R/W)

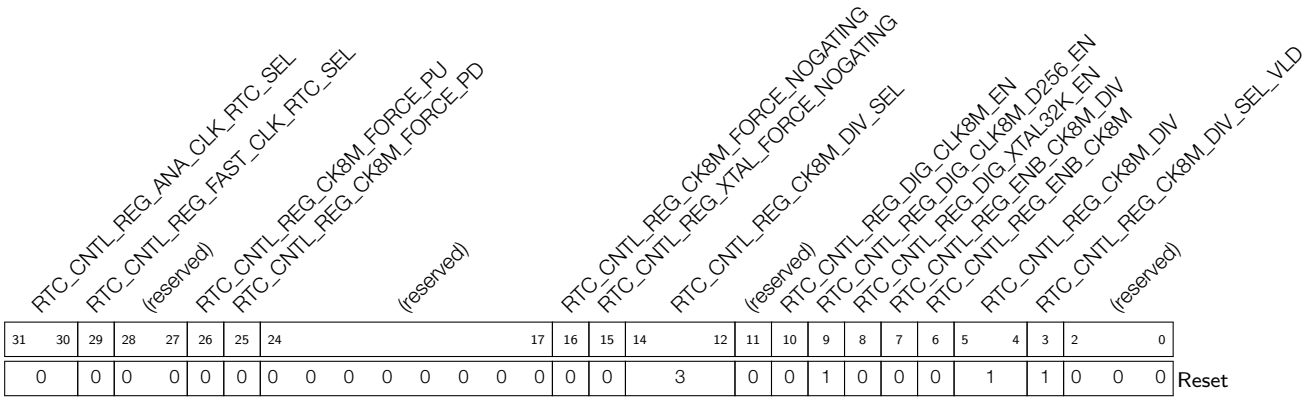
RTC_CNTL_REG_RTC_WDT_STATE Stores the status of the 32 kHz watchdog. (RO)

RTC_CNTL_REG_RTC_XTAL32K_GPIO_SEL Selects the 32 kHz crystal clock. 0: selects the external 32 kHz clock; 1: selects clock from the RTC GPIO X32P_C. (R/W)

RTC_CNTL_REG_XTL_EXT_CTR_LV 0: powers down XTAL at high level; 1: powers down XTAL at low level. (R/W)

RTC_CNTL_REG_XTL_EXT_CTR_EN Enables the GPIO to power down the crystal oscillator. (R/W)

Register 28.41: RTC_CNTL_CLK_CONF_REG (0x0074)



RTC_CNTL_REG_CK8M_DIV_SEL_VLD Synchronizes the reg_ck8m_div_sel. Note that you have to invalidate the bus before modifying the frequency divider, and then validate the new divider clock. (R/W)

RTC_CNTL_REG_CK8M_DIV Set the CK8M_D256_OUT divider. 00: divided by 128, 01: divided by 256, 10: divided by 512, 11: divided by 1024. (R/W)

RTC_CNTL_REG_ENB_CK8M Set this bit to disable CK8M and CK8M_D256_OUT. (R/W)

RTC_CNTL_REG_ENB_CK8M_DIV Selects the CK8M_D256_OUT. 1: CK8M, 0: CK8M divided by 256. (R/W)

RTC_CNTL_REG_DIG_XTAL32K_EN Set this bit to enable CK_XTAL_32K clock for the digital core. (R/W)

RTC_CNTL_REG_DIG_CLK8M_D256_EN Set this bit to enable CK8M_D256_OUT clock for the digital core. (R/W)

RTC_CNTL_REG_DIG_CLK8M_EN Set this bit to enable 8 MHz clock for the digital core. (R/W)

RTC_CNTL_REG_CK8M_DIV_SEL Stores the 8 MHz divider, which is reg_ck8m_div_sel + 1. (R/W)

RTC_CNTL_REG_XTAL_FORCE_NOGATING Set this bit to force no gating to crystal during sleep. (R/W)

RTC_CNTL_REG_CK8M_FORCE_NOGATING Set this bit to disable force gating to 8 MHz crystal during sleep. (R/W)

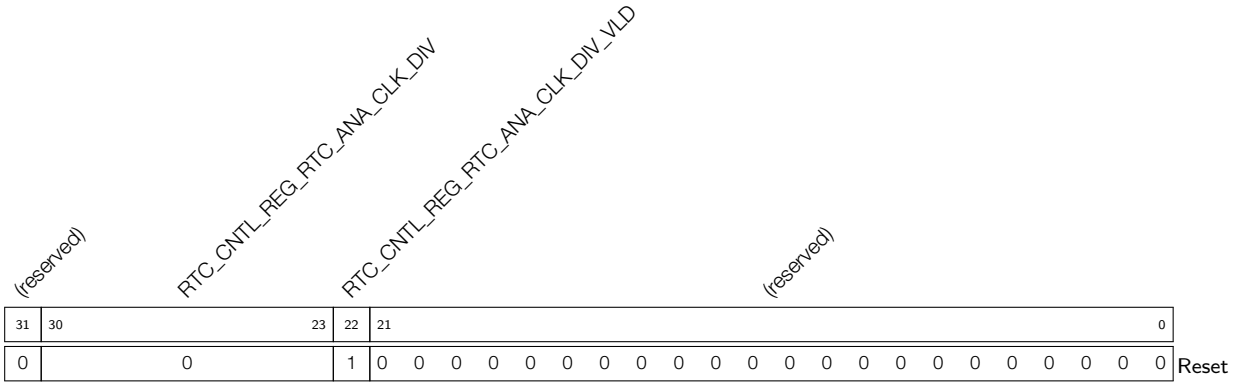
RTC_CNTL_REG_CK8M_FORCE_PD Set this bit to FPD the 8 MHz clock. (R/W)

RTC_CNTL_REG_CK8M_FORCE_PU Set this bit to FPU the 8 MHz clock. (R/W)

RTC_CNTL_REG_FAST_CLK_RTC_SEL Set this bit to select the RTC fast clock. 0: XTAL div 4, 1: CK8M. (R/W)

RTC_CNTL_REG_ANA_CLK_RTC_SEL Set this bit to select the RTC slow clock. 0: 90K rtc_clk, 1: 32k XTAL, 2: 8md256. (R/W)

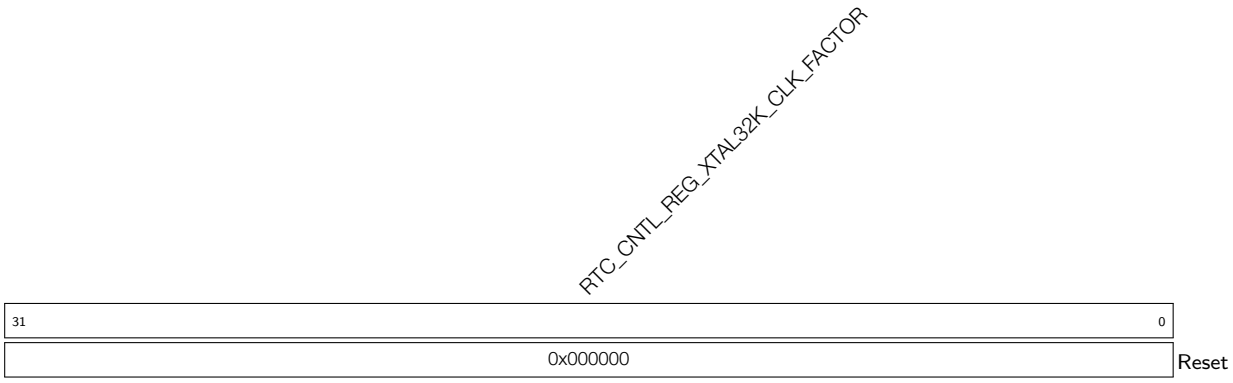
Register 28.42: RTC_CNTL_SLOW_CLK_CONF_REG (0x0078)



RTC_CNTL_REG_RTC_ANA_CLK_DIV_VLD Synchronizes the reg_rtc_ana_clk_div. Note that you have to invalidate the bus before modifying the frequency divider, and then validate the new divider clock. (R/W)

RTC_CNTL_REG_RTC_ANA_CLK_DIV Set the divider for the RTC clock. (R/W)

Register 28.43: RTC_CNTL_XTAL32K_CLK_FACTOR_REG (0x00F0)



RTC_CNTL_REG_XTAL32K_CLK_FACTOR Configures the divider factor for the 32 kHz crystal oscillator. (R/W)

Register 28.44: RTC_CNTL_XTAL32K_CONF_REG (0x00F4)

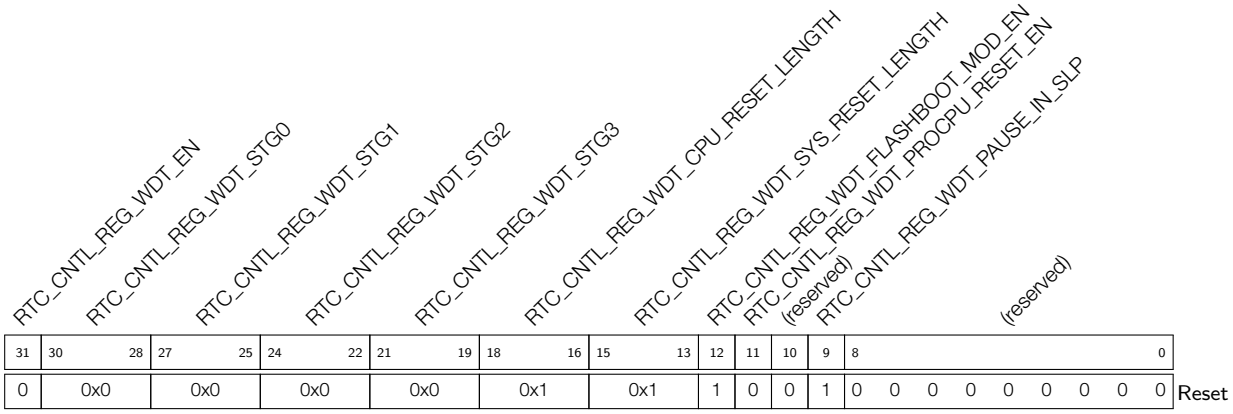
RTC_CNTL_REG_XTAL32K_RETURN_WAIT Defines the waiting cycles before returning to the normal 32 kHz crystal oscillator. (R/W)

RTC_CNTL_REG_XTAL32K_RESTART_WAIT Defines the waiting cycles before restarting the 32 kHz crystal oscillator. (R/W)

RTC_CNTL_REG_XTAL32K_WDT_TIMEOUT Defines the waiting period for clock detection. If no clock is detected after this period, the 32 kHz crystal oscillator can be regarded as dead. (R/W)

RTC_CNTL_REG_XTAL32K_STABLE_THRES Defines the allowed restarting period, within which the 32 kHz crystal oscillator can be regarded as stable. (R/W)

Register 28.45: RTC_CNTL_WDTCONFIG0_REG (0x0094)



RTC_CNTL_REG_WDT_PAUSE_IN_SLP Set this bit to pause the watchdog in sleep. (R/W)

RTC_CNTL_REG_WDT_PROCPU_RESET_EN Set this bit to allow the watchdog to be able to reset CPU. (R/W)

RTC_CNTL_REG_WDT_FLASHBOOT_MOD_EN Set this bit to enable watchdog when the chip boots from flash. (R/W)

RTC_CNTL_REG_WDT_SYS_RESET_LENGTH Sets the length of the system reset counter. (R/W)

RTC_CNTL_REG_WDT_CPU_RESET_LENGTH Sets the length of the CPU reset counter. (R/W)

RTC_CNTL_REG_WDT_STG3 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_REG_WDT_STG2 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

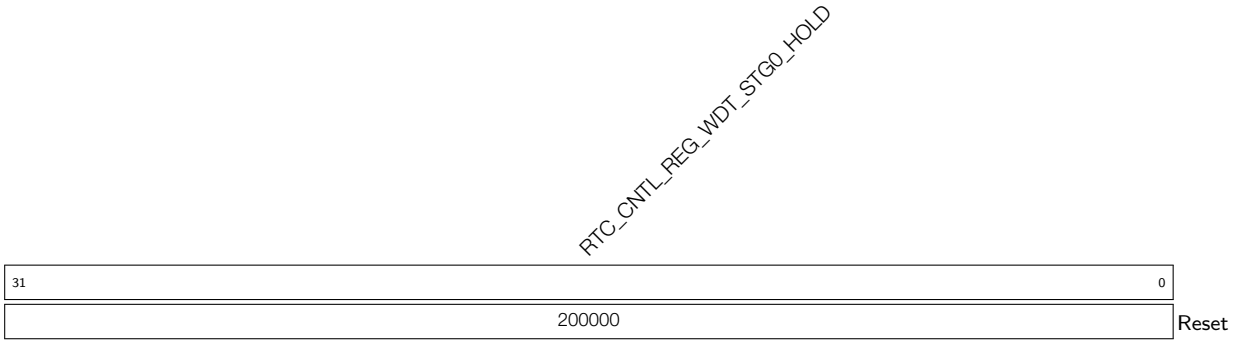
RTC_CNTL_REG_WDT_STG1 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_REG_WDT_STG0 1: enable at the interrupt stage, 2: enable at the CPU stage, 3: enable at the system stage, 4: enable at the system and RTC stage. (R/W)

RTC_CNTL_REG_WDT_EN Set this bit to enable the RTC watchdog. (R/W)

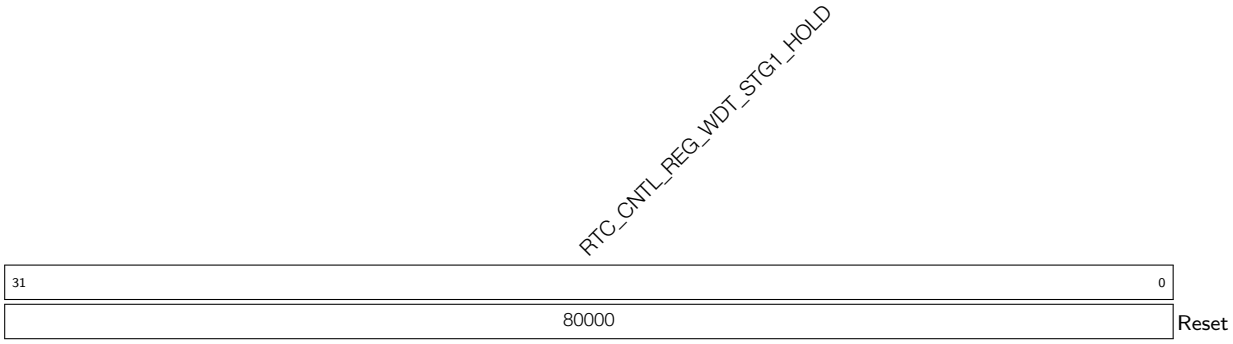
Note:
For details, please refer to Chapter 14 *XTAL32K Watchdog*.

Register 28.46: RTC_CNTL_WDTCONFIG1_REG (0x0098)



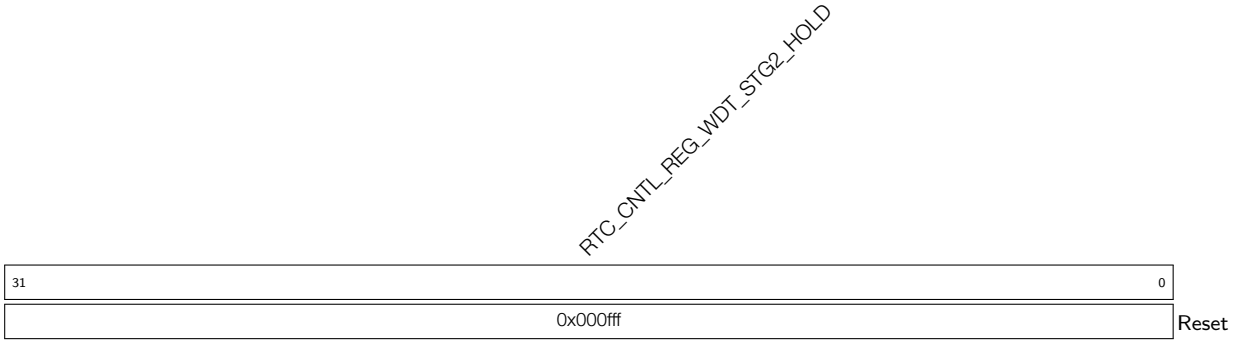
RTC_CNTL_REG_WDT_STG0_HOLD Configures the hold time of RTC watchdog at level 1. (R/W)

Register 28.47: RTC_CNTL_WDTCONFIG2_REG (0x009C)



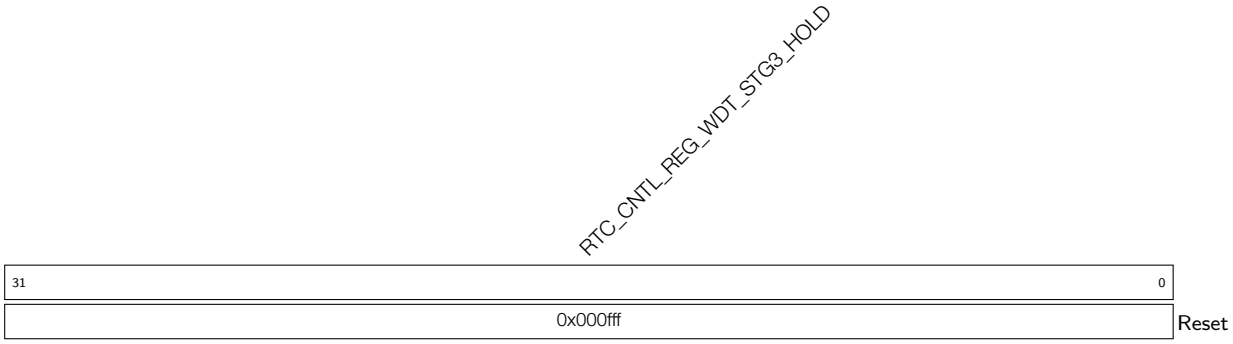
RTC_CNTL_REG_WDT_STG1_HOLD Configures the hold time of RTC watchdog at level 2. (R/W)

Register 28.48: RTC_CNTL_WDTCONFIG3_REG (0x00A0)



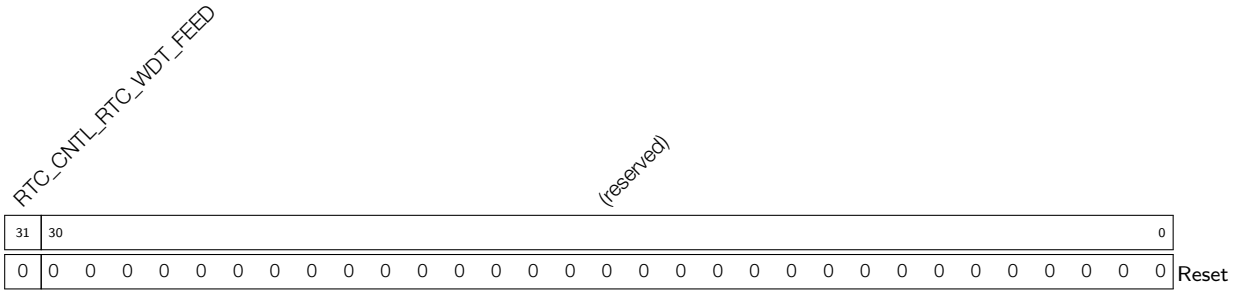
RTC_CNTL_REG_WDT_STG2_HOLD Configures the hold time of RTC watchdog at level 3. (R/W)

Register 28.49: RTC_CNTL_WDTCONFIG4_REG (0x00A4)



RTC_CNTL_REG_WDT_STG3_HOLD Configures the hold time of RTC watchdog at level 4. (R/W)

Register 28.50: RTC_CNTL_WDTFEED_REG (0x00A8)

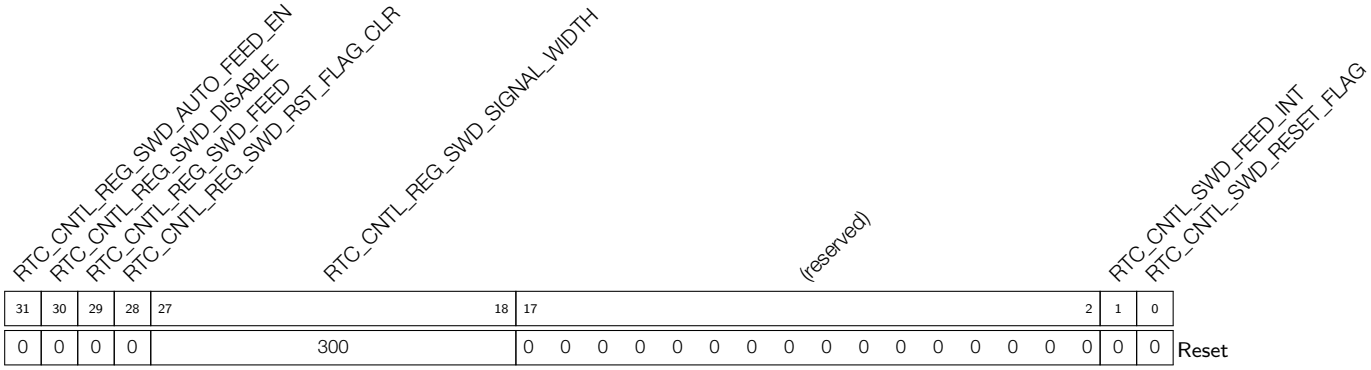


RTC_CNTL_RTC_WDT_FEED Set this bit to feed the RTC watchdog. (WO)

Register 28.51: RTC_CNTL_WDTWPROTECT_REG (0x00AC)

RTC_CNTL_REG_WDT_WKEY Sets the write protection key of the watchdog. (R/W)

Register 28.52: RTC_CNTL_SWD_CONF_REG (0x00B0)



RTC_CNTL_SWD_RESET_FLAG Indicates the super watchdog reset flag. (RO)

RTC_CNTL_SWD_FEED_INT Receiving this interrupt leads to feeding the super watchdog via SW. (RO)

RTC_CNTL_REG_SWD_SIGNAL_WIDTH Adjusts the signal width sent to the super watchdog. (R/W)

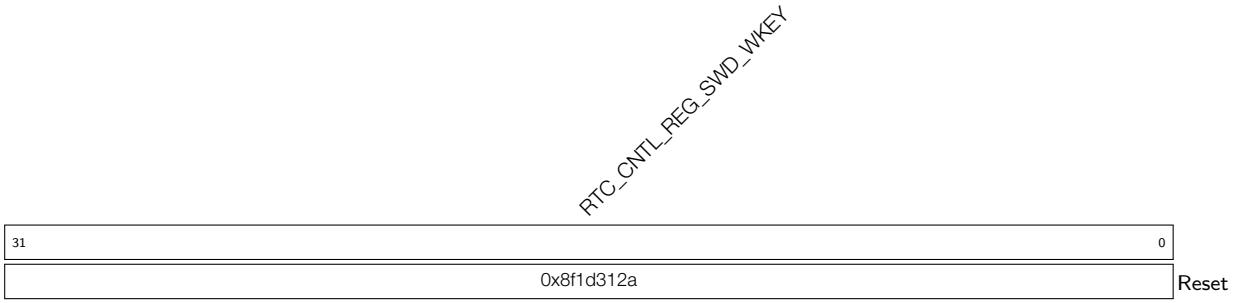
RTC_CNTL_REG_SWD_RST_FLAG_CLR Set to reset the super watchdog reset flag. (WO)

RTC_CNTL_REG_SWD_FEED Set to feed the super watchdog via SW. (WO)

RTC_CNTL_REG_SWD_DISABLE Set this bit to disable super watchdog. (R/W)

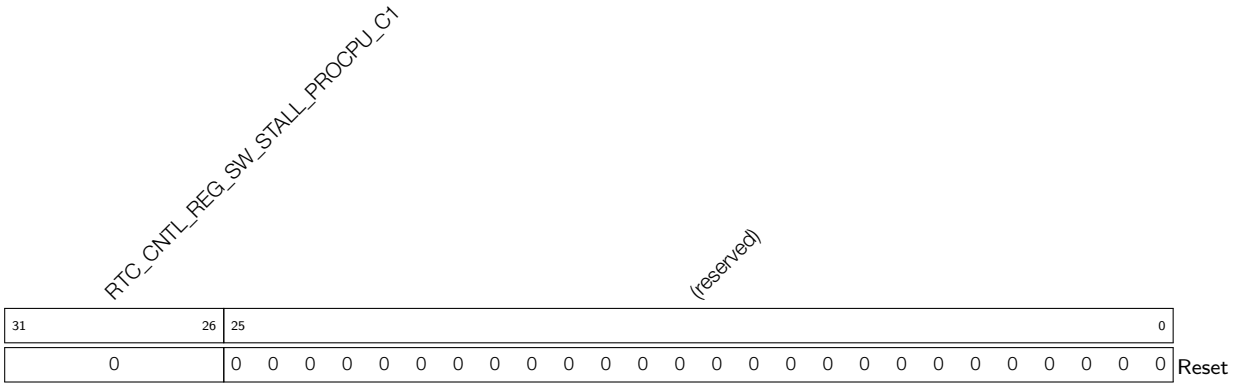
RTC_CNTL_REG_SWD_AUTO_FEED_EN Set this bit to enable automatic watchdog feeding upon interrupts. (R/W)

Register 28.53: RTC_CNTL_SWD_WPROTECT_REG (0x00B4)



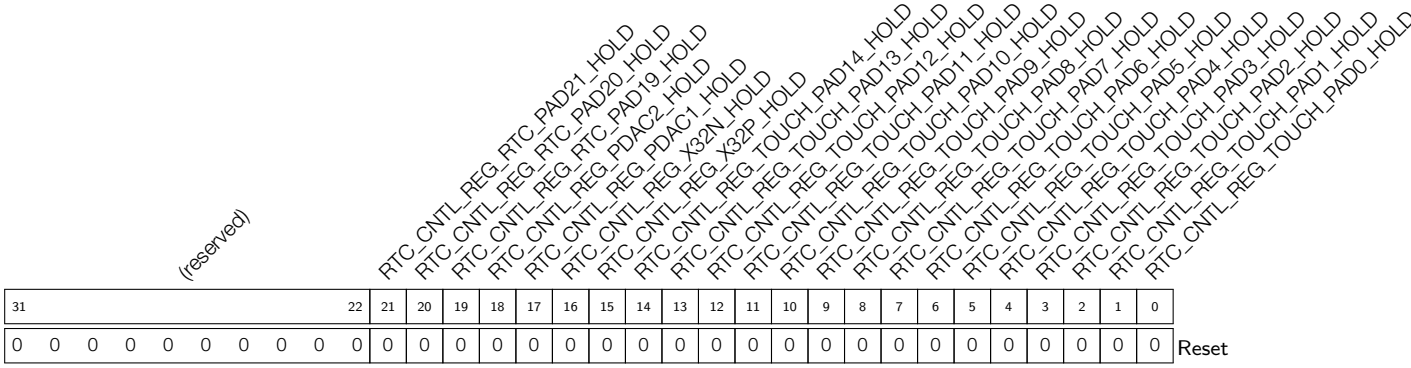
RTC_CNTL_REG_SWD_WKEY Sets the write protection key of the super watchdog. (R/W)

Register 28.54: RTC_CNTL_SW_CPU_STALL_REG (0x00B8)



RTC_CNTL_REG_SW_STALL_PROCPU_C1 When RTC_CNTL_REG_SW_STALL_PROCPU_C0 is configured to 0x2, setting this bit to 0x21 stalls the CPU by SW. (R/W)

Register 28.55: RTC_CNTL_PAD_HOLD_REG (0x00D4)



RTC_CNTL_REG_TOUCH_PAD0_HOLD Sets the touch GPIO 0 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD1_HOLD Sets the touch GPIO 1 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD2_HOLD Sets the touch GPIO 2 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD3_HOLD Sets the touch GPIO 3 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD4_HOLD Sets the touch GPIO 4 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD5_HOLD Sets the touch GPIO 5 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD6_HOLD Sets the touch GPIO 6 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD7_HOLD Sets the touch GPIO 7 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD8_HOLD Sets the touch GPIO 8 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD9_HOLD Sets the touch GPIO 9 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD10_HOLD Sets the touch GPIO 10 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD11_HOLD Sets the touch GPIO 11 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD12_HOLD Sets the touch GPIO 12 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD13_HOLD Sets the touch GPIO 13 to the holding state. (R/W)

RTC_CNTL_REG_TOUCH_PAD14_HOLD Sets the touch GPIO 14 to the holding state. (R/W)

RTC_CNTL_REG_X32P_HOLD Sets the x32p to the holding state. (R/W)

RTC_CNTL_REG_X32N_HOLD Sets the x32n to the holding state. (R/W)

RTC_CNTL_REG_PDAC1_HOLD Sets the pdac1 to the holding state. (R/W)

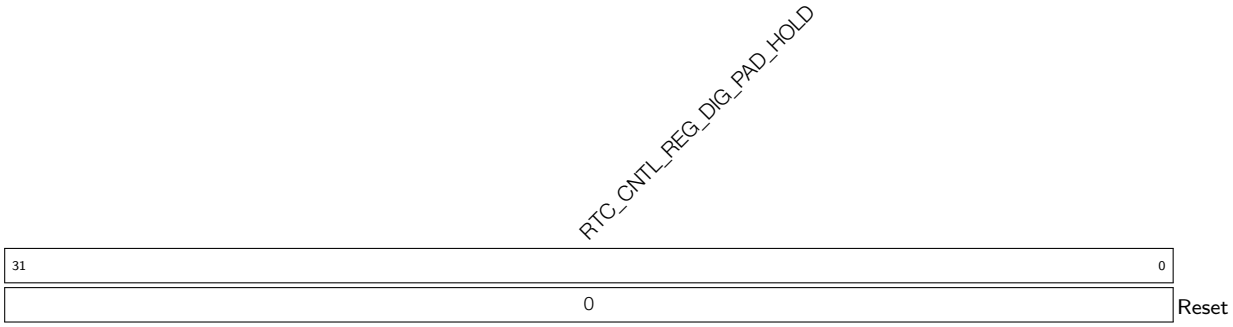
RTC_CNTL_REG_PDAC2_HOLD Sets the pdac2 to the holding state. (R/W)

RTC_CNTL_REG_RTC_PAD19_HOLD Sets the RTG GPIO 19 to the holding state. (R/W)

RTC_CNTL_REG_RTC_PAD20_HOLD Sets the RTG GPIO 20 to the holding state. (R/W)

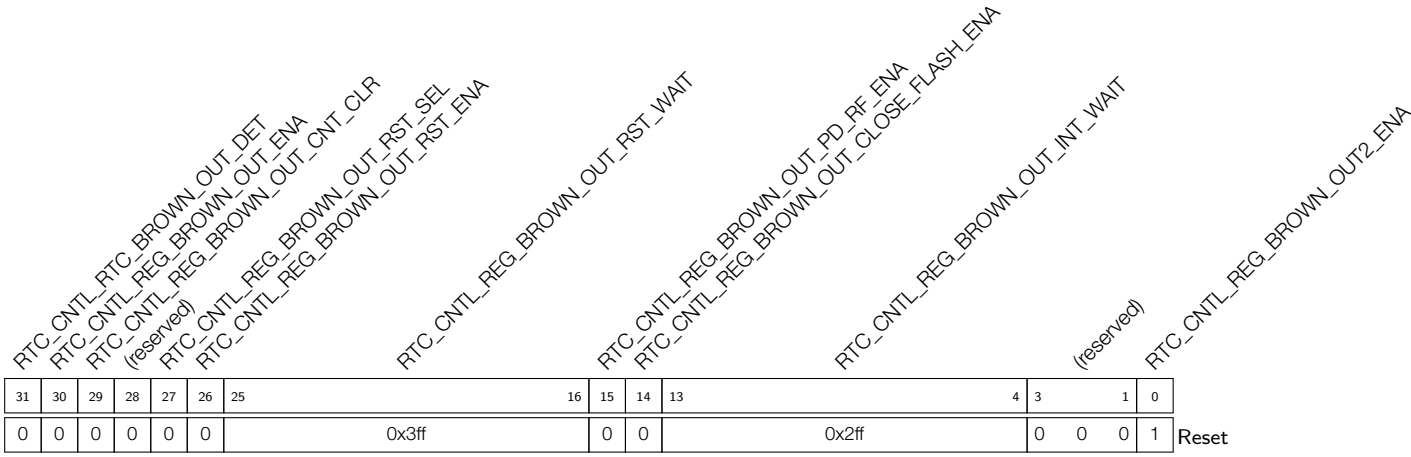
RTC_CNTL_REG_RTC_PAD21_HOLD Sets the RTG GPIO 21 to the holding state. (R/W)

Register 28.56: RTC_CNTL_DIG_PAD_HOLD_REG (0x00D8)



RTC_CNTL_REG_DIG_PAD_HOLD Set GPIO 21 to GPIO 45 to the holding state. (See bitmap to locate any GPIO). (R/W)

Register 28.57: RTC_CNTL_BROWN_OUT_REG (0x00E4)



RTC_CNTL_REG_BROWN_OUT2_ENA Enables the brown_out2 to initiate a chip reset. (R/W)

RTC_CNTL_REG_BROWN_OUT_INT_WAIT Configures the waiting cycles before sending an interrupt. (R/W)

RTC_CNTL_REG_BROWN_OUT_CLOSE_FLASH_ENA Set this bit to enable PD the flash when a brown-out happens. (R/W)

RTC_CNTL_REG_BROWN_OUT_PD_RF_ENA Set this bit to enable PD the RF circuits when a brown-out happens. (R/W)

RTC_CNTL_REG_BROWN_OUT_RST_WAIT Configures the waiting cycles before the reset after a brown-out. (R/W)

RTC_CNTL_REG_BROWN_OUT_RST_ENA Enables to reset brown-out. (R/W)

RTC_CNTL_REG_BROWN_OUT_RST_SEL Selects the reset type when a brown-out happens. 1: chip reset, 0: system reset. (R/W)

RTC_CNTL_REG_BROWN_OUT_CNT_CLR Clears the brown-out counter. (WO)

RTC_CNTL_REG_BROWN_OUT_ENA Set this bit to enable brown-out detection. (R/W)

RTC_CNTL_RTC_BROWN_OUT_DET Indicates the status of the brown-out signal. (RO)

Revision History

Date	Version	Release notes
2020-05-21	V0.4	Added the following chapters: • Chapter 18 *TWAI* • Chapter 28 *Low-power Management* Updated the following chapters: • Added a new section 5.4 *Dedicated GPIO* in Chapter 5 *IO MUX and GPIO Matrix* • Removed bit EFUSE_PGM_DATA1_REG[16] from the list of bits that are reserved and can only be used by software; added bit EFUSE_RPT4_RESERVED5 and EFUSE_RPT4_RESERVED5_ERR in Chapter 16 *eFuse Controller* • Updated the base address in Chapter 22 *Random Number Generator*
2020-04-01	V0.3	Added the following chapters: • Chapter 14 *XTAL32K Watchdog* • Chapter 15 *System Timer* • Chapter 24 *Permission Control* • Chapter 26 *HMAC Module* • Chapter 27 *ULP Coprocessor* Updated RTC_CLK frequency in Chapter 2 *Reset and Clock*
2020-01-20	V0.2	Added the following chapters: • Chapter 5 *IO MUX and GPIO Matrix* • Chapter 7 *DMA Controller* • Chapter 8 *UART Controller* Updated the configurations of eFuse-programming and eFuse-read timing parameters in Chapter 16 *eFuse Controller*
2019-11-27	V0.1	Preliminary release