ESP32-S3

Technical Reference Manual

Pre-release v0.1
Espressif Systems

About This Manual

The **ESP32-S3 Technical Reference Manual** is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32-S3 memory and peripherals.

For pin definition, electrical characteristics, and package information, please see *ESP32-S3 Datasheet*.

Document Updates

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

Revision History

For revision history of this document, 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.

Contents

List of Tables

List of Figures

1 System and Memory

1.1 Overview

The ESP32-S3 is a dual-core system with two Harvard Architecture Xtensa® LX7 CPUs. All internal memory, external memory, and peripherals are located on the CPU buses.

1.2 Features

- **Address Space**
  - 848 KB of internal memory address space accessed from the instruction bus
  - 560 KB of internal memory address space accessed from the data bus
  - 836 KB of peripheral address space
  - 32 MB of external memory virtual address space accessed from the instruction bus
  - 32 MB external memory virtual address space accessed from the data bus
  - 480 KB of internal DMA address space
  - 32 MB of external DMA address space
- **Internal Memory**
  - 384 KB Internal ROM
  - 512 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- **External Memory**
  - Supports up to 1 GB external flash
  - Supports up to 1 GB external RAM
- **Peripheral Space**
  - 45 modules/peripherals in total
- **GDMA**
  - 11 GDMA-supported modules/peripherals

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

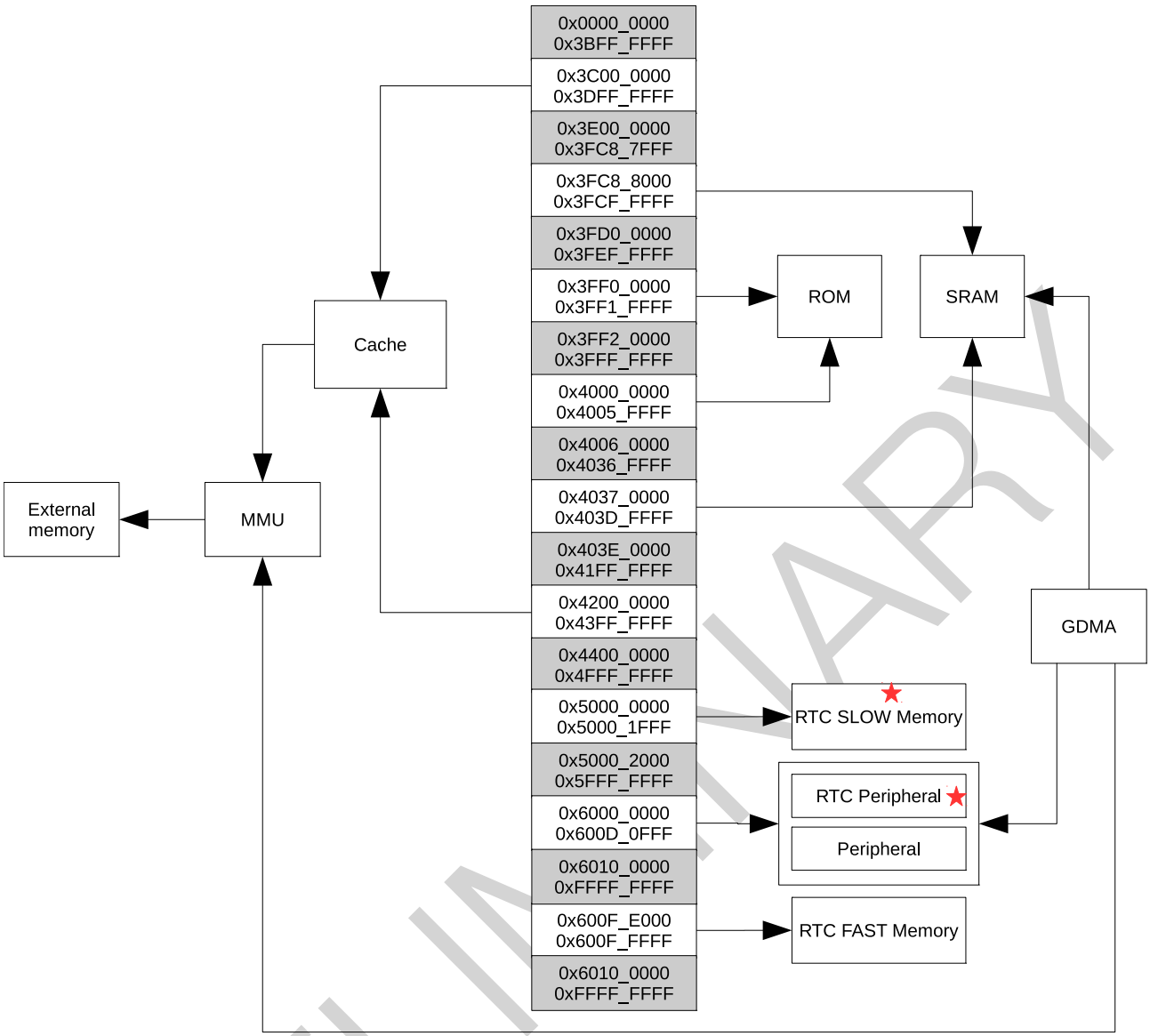


Figure 1-1. System Structure and Address Mapping

Note:

- The address space with gray background is not available to users.
- The memory or peripheral marked with a red pentagram can be accessed by the ULP co-processor.
- The range of addresses available in the address space may be larger than the actual available memory of a particular type.

1.3 Functional Description

1.3.1 Address Mapping

The system contains two Harvard Architecture Xtensa® LX7 CPUs, and both can access the same range of address space.

Addresses below 0x4000_0000 are accessed using the data bus. Addresses in the range of 0x4000_0000 ~

0x4FFF_FFFF are accessed using the instruction bus. Addresses over and including 0x5000_0000 are shared by both data bus and instruction bus.

Both data bus and instruction bus are little-endian. The CPU can access data via the data bus using single-byte, double-byte, 4-byte and 16-byte alignment. The CPU can also access data via the instruction bus, but only in 4-byte aligned manner; non-aligned data access will cause a CPU exception.

The CPU can:

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

Table 1-1 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 CPU can access the same memory using multiple addresses.

Table 1-1. Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3BFF_FFFF		Reserved
Data bus	0x3C00_0000	0x3DFF_FFFF	32 MB	External memory
	0x3E00_0000	0x3FC8_7FFF		Reserved
Data bus	0x3FC8_8000	0x3FCF_FFFF	480 KB	Internal memory
	0x3FD0_0000	0x3FEF_FFFF		Reserved
Data bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	Internal memory
	0x3FF2_0000	0x3FFF_FFFF		Reserved
Instruction bus	0x4000_0000	0x4005_FFFF	384 KB	Internal memory
	0x4006_0000	0x4036_FFFF		Reserved
Instruction bus	0x4037_0000	0x403D_FFFF	448 KB	Internal memory
	0x403E_0000	0x41FF_FFFF		Reserved
Instruction bus	0x4200_0000	0x43FF_FFFF	32 MB	External memory
	0x4400_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	0x600D_0FFF	836 KB	Peripherals
	0x600D_1000	0x600F_DFFF		Reserved
	0x600F_E000	0x600F_FFFF	8 KB	Internal memory
	0x6010_0000	0xFFFF_FFFF		Reserved

1.3.2 Internal Memory

The ESP32-S3 consists of the following three types of internal memory:

- Internal ROM (384 KB): The internal ROM is a read-only memory and cannot be programmed. Internal ROM contains the ROM code (software instructions and some software read-only data) of some low level system software.
- Internal SRAM (512 KB): The Internal Static RAM (SRAM) is a volatile memory that can be quickly accessed by the CPU (generally within a single CPU clock cycle).
  - A part of the SRAM can be configured to operate as a cache for external memory access, which cannot be accessed by CPU in such case.
  - Some parts of the SRAM can only be accessed via the CPU's instruction bus.
  - Some parts of the SRAM can only be accessed via the CPU's data bus.
  - Some parts of the SRAM can be accessed via both the CPU's instruction bus and the CPU's data bus.
- RTC Memory (16 KB): The RTC (Real Time Clock) memory implemented as Static RAM (SRAM) and thus is volatile. However, RTC memory has the added feature of being persistent throughout deep sleep (i.e., the RTC memory retains its values throughout deep sleep).
  - RTC FAST Memory (8 KB): RTC FAST memory can only be accessed by the CPU, and cannot be accessed by the ULP co-processor. It is generally used to store instructions and data that needs to persist across a deep sleep.
  - RTC SLOW Memory (8 KB): The RTC SLOW memory can be accessed by both the CPU and the ULP co-processor, and thus is generally used to store instructions and share data between the CPU and the ULP co-processor.

Based on the three different types of internal memory described above, the internal memory of the ESP32-S3 is split into four segments: Internal ROM (384 KB), Internal SRAM (512 KB), RTC FAST Memory (8 KB) and RTC SLOW Memory (8 KB). However, within each segment, there may be different bus access restrictions (e.g., some parts of the segment may only be accessible by the CPU's instruction bus). Therefore, some segments are also further divided down into parts. Table 1-2 describes each part of internal memory and their address ranges on the data bus and/or instruction bus.

Table 1-2. Internal Memory Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
Data bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	Internal ROM 1
	0x3FC8_0000	0x3FCE_FFFF	416 KB	Internal SRAM 1
	0x3FCF_0000	0x3FCF_FFFF	64 KB	Internal SRAM 2
Instruction bus	0x4000_0000	0x4003_FFFF	256 KB	Internal ROM 0
	0x4004_0000	0x4005_FFFF	128 KB	Internal ROM 1
	0x4037_0000	0x4037_7FFF	32 KB	Internal SRAM 0
	0x4037_8000	0x403D_FFFF	416 KB	Internal SRAM 1
Data/Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory
	0x600F_E000	0x600F_FFFF	8 KB	RTC FAST Memory

Note:

All of the internal memories are managed by Permission Control module. An internal memory can only be accessed when it is allowed by Permission Control, then the internal memory can be available to the CPU. For more information about Permission Control, please refer to Chapter 7 *Permission Control (PMS) [to be added later]*.

1. Internal ROM 0

Internal ROM 0 is a 256 KB, read-only memory space, addressed by the CPU only through the instruction bus, as shown in Table 1-2.

2. Internal ROM 1

Internal ROM 1 is a 128 KB, read-only memory space, addressed by the CPU through the instruction bus via 0x4004_0000 ~ 0x4005_FFFF or through the data bus via 0x3FF0_0000 ~ 0x3FF1_FFFF in the same order, as shown in Table 1-2.

This means, for example, address 0x4005_0000 and 0x3FF0_0000 correspond to the same word, 0x4005_0004 and 0x3FF0_0004 correspond to the same word, 0x4005_0008 and 0x3FF0_0008 correspond to the same word, etc (same below).

3. Internal SRAM 0

Internal SRAM 0 is a 32 KB, read-and-write memory space, addressed by the CPU through the instruction bus, as shown in Table 1-2.

A 16 KB or the total 32 KB of this memory space can be configured as instruction cache (ICache) to store instructions or read-only data of the external memory. In this case, the occupied memory space cannot be accessed by the CPU, while the remaining can still can be accessed by the CPU.

4. Internal SRAM 1

Internal SRAM 1 is a 416 KB, read-and-write memory space, addressed by the CPU through the data bus or instruction bus in the same order, as shown in Table 1-2.

The total 416 KB memory space comprises multiple 8 KB and 16 KB memory (sub-memory) blocks. A memory block (up to 16 KB) can be used as a Trace Memory, in which case this block can still be accessed by the CPU.

5. Internal SRAM 2

Internal SRAM 2 is a 64 KB, read-and-write memory space, addressed by the CPU through the data bus, as shown in Table 1-2.

A 32 KB or the total 64 KB can be configured as data cache (DCache) to cache data of the external memory. The space used as DCache cannot be accessed by the CPU, while the remaining space can still be accessed by the CPU.

6. RTC FAST Memory

RTC FAST Memory is a 8 KB, read-and-write SRAM, addressed by the CPU through the data/instruction bus via the shared address 0x600F_E000 ~ 0x600F_FFFF, as described in Table 1-2.

7. RTC SLOW Memory

RTC SLOW Memory is a 8 KB, read-and-write SRAM, addressed by the CPU through the data/instruction bus via shared address 0x5000_E000 ~ 0x5001_FFFF, as described in Table 1-2.

RTC SLOW Memory can also be used as a peripheral addressable to the CPU via 0x6002_1000 ~ 0x6002_2FFF.

1.3.3 External Memory

ESP32-S3 supports SPI, Dual SPI, Quad SPI, Octal SPI, QPI, and OPI interfaces that allow connection to external flash and RAM. It also supports hardware encryption and decryption based on XTS_AES algorithm to protect users' programs and data in the external flash and RAM.

1.3.3.1 External Memory Address Mapping

The CPU accesses the external memory via the cache. According to information inside the MMU (Memory Management Unit), the cache maps the CPU's instruction/data bus address into a physical address of the external flash and RAM. Due to this address mapping, ESP32-S3 can address up to 1 GB external flash and 1 GB external RAM.

Using the cache, ESP32-S3 is able to support the following address space mappings at a time:

- Up to 32 MB instruction bus address space can be mapped to the external flash or RAM as individual 64 KB blocks via the ICache. 4-byte aligned reads and fetches are supported.
- Up to 32 MB data bus address space can be mapped to the external RAM as individual 64 KB blocks via the DCache. Single-byte, double-byte, 4-byte, 16-byte aligned reads and writes are supported. This address space can also be mapped to the external flash or RAM for read operations only.

Table 1-3 lists the mapping between the cache and the corresponding address ranges on the data bus and instruction bus.

Table 1-3. External Memory Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
Data bus	0x3C00_0000	0x3DFF_FFFF	32 MB	DCache
Instruction bus	0x4200_0000	0x43FF_FFFF	32 MB	ICache

Note:

Only if the CPU obtains permission for accessing the external memory, can it be responded for memory access. For more detailed information about permission control, please refer to Chapter 7 *Permission Control (PMS) [to be added later]*.

1.3.3.2 Cache

As shown in Figure 1-2, ESP32-S3 has a dual-core-shared ICache and DCache structure, which allows prompt response upon simultaneous requests from the instruction bus and data bus. Some internal memory space can be used as cache (see Internal SRAM 0 and Internal SRAM 2 in Section 1.3.2).

When the instruction bus of two cores initiate a request on ICache simultaneously, the arbiter determines which core gets the access to the ICache first; when the data bus of two cores initiate a request on DCache

simultaneously, the arbiter determines which gets the access to the DCache first. When a cache miss occurs, the cache controller will initiate a request to the external memory. When ICache and DCache initiate requests on the external memory simultaneously, the arbiter determines which gets the access to the external memory first. The size of ICache can be configured to 16 KB or 32 KB, while its block size can be configured to 16 B or 32 B. When an ICache is configured to 32 KB, its block cannot be 16 B. The size of DCache can be configured to 32 KB or 64 KB, while its block size can be configured to 16 B, 32 B or 64 B. When a DCache is configured to 64 KB, its block cannot be 16 B.

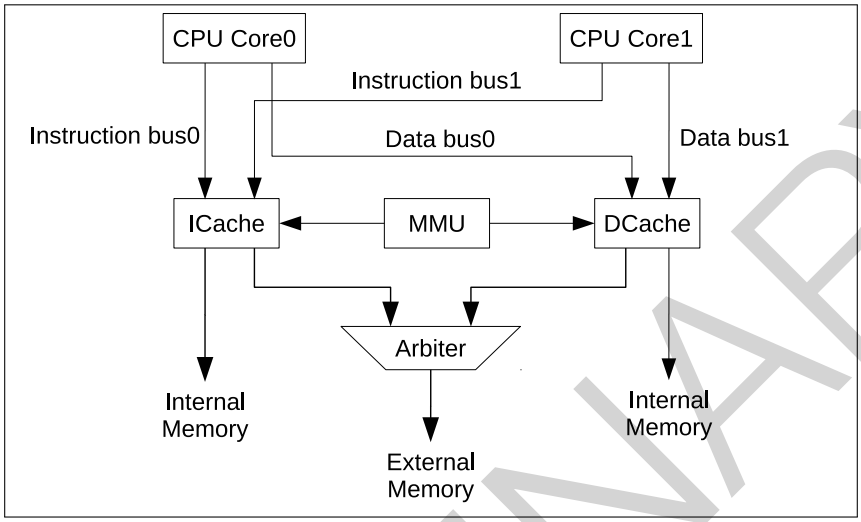


Figure 1-2. Cache Structure

1.3.3.3 Cache Operations

ESP32-S3 caches support the following operations:

1. **Write-Back**: This operation is used to clear the dirty bits in dirty blocks and update the new data to the external memory. After the write-back operation finished, both the external memory and the cache are bearing the new data. The CPU can then read/write the data directly from the cache. Only DCache has this function.

   If the data in the cache is newer than the one stored in the external memory, then the new data will be considered as a dirty block. The cache tracks these dirty blocks through their dirty bits. When the dirty bits of a data are cleared, the cache will consider the data as new.

2. **Clean**: This operation is used to clear dirty bits in the dirty block, without updating data to the external memory. After the clean operation finish, there will still be old data stored in the external memory, while the cache keeps the new one (but the cache does not know about this). The CPU can then read/write the data directly from the cache. Only DCache has this function.

3. **Invalidate**: This operation is used to remove valid data in the cache. Even if the data is a dirty block mentioned above, it will not be updated to the external memory. But for the non-dirty data, it will be only stored in the external memory after this operation. The CPU needs to access the external memory in order to read/write this data. As for the dirty blocks, they will be totally lost with only old data in the external memory after this operation. There are two types of invalidate operation: automatic invalidation (Auto-Invalidate) and manual invalidation (Manual-Invalidate). Manual-Invalidate is performed only on data in the specified area in the cache, while Auto-Invalidate is performed on all data in the cache. Both ICache and DCache have this function.

4. **Preload**: This operation is to load instructions and data into the cache in advance. The minimum unit of preload-operation is one block. There are two types of preload-operation: manual preload (Manual-Preload) and automatic preload (Auto-Preload). Manual-Preload means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Auto-Preload means the hardware prefetches a piece of continuous data according to the current address where the cache hits or misses (depending on configuration). Both ICache and DCache have this function.
5. **Lock/Unlock**: The lock operation is used to prevent the data in the cache from being easily replaced. There are two types of lock: prelock and manual lock. When prelock is enabled, the cache locks the data in the specified area when filling the missing data to cache memory, while the data outside the specified area will not be locked. When manual lock is enabled, the cache checks the data that is already in the cache memory and locks the data only if it falls in the specified area, and leaves the data outside the specified area unlocked. When there are missing data, the cache will replace the data in the unlocked way first, so the data in the locked way is always stored in the cache and will not be replaced. But when all ways within the cache are locked, the cache will replace data, as if it was not locked. Unlocking is the reverse of locking, except that it only can be done manually. Both ICache and DCache have this function.

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

1.3.4 GDMA Address Space

The GDMA (General Direct Memory Access) peripheral in ESP32-S3 can provide DMA (Direct Memory Access) services including:

- Data transfers between different locations of internal memory;
- Data transfers between internal memory and external memory;
- Data transfers between different locations of external memory.

GDMA uses the same addresses as the CPU's data bus to access Internal SRAM 1 and Internal SRAM 2. Specifically, GDMA uses address range 0x3FC8_8000 ~ 0x3FCE_FFFF to access Internal SRAM 1 and 0x3FCF_0000 ~ 0x3FCF_FFFF to access Internal SRAM 2. Note that GDMA cannot access the internal memory occupied by cache.

In addition, GDMA can access the external memory (only RAM) via the same address as CPU accessing DCache (0x3C00_0000 ~ 0x3DFF_FFFF). When DCache and GDMA access the external memory simultaneously, the software needs to make sure the data is consistent.

Besides, some peripherals/modules of the ESP32-S3 can work together with GDMA. In these cases, GDMA can provide the following powerful services for them:

- Data transfers between modules/peripherals and internal memory;
- Data transfers between modules/peripherals and external memory.

There are 11 peripherals/modules that can work together with GDMA. As shown in Figure 1-3, these 11 vertical lines in turn correspond to these 11 peripherals/modules with GDMA function, the horizontal line represents a certain channel of GDMA (can be any channel), and the intersection of the vertical line and the horizontal line indicates that a peripheral/module has the ability to access the corresponding channel of GDMA. If there are multiple intersections on the same line, it means that these peripherals/modules cannot enable the GDMA function at the same time.

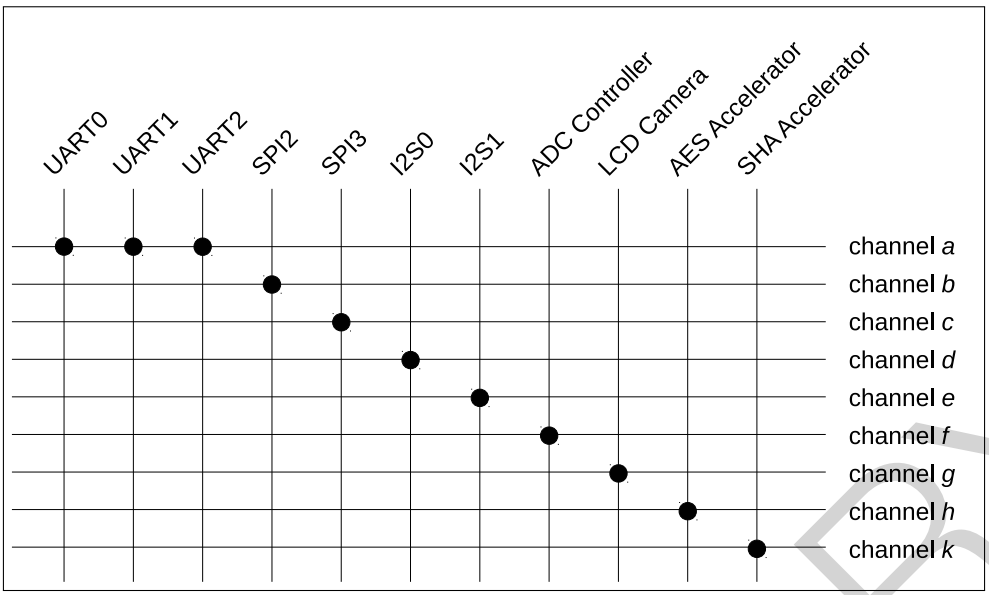


Figure 1-3. Peripherals/modules that can work with GDMA

These peripherals/modules can access any memory available to GDMA. For more information, please refer to Chapter 9 *GDMA Controller (DMA) [to be added later]*.

Note:

When accessing a memory via GDMA, a corresponding access permission is needed, otherwise this access may fail. For more information about permission control, please refer to Chapter 7 *Permission Control (PMS) [to be added later]*.

1.3.5 Modules/Peripherals

The CPU can access modules/peripherals via 0x6000_0000 ~ 0x600D_0FFF shared by the data/instruction bus.

1.3.5.1 Module/Peripheral Address Mapping

Table 1-4 lists all the modules/peripherals and their respective address ranges. Note that the address space of specific modules/peripherals is defined by "Boundary Address" (including both Low Address and High Address).

Table 1-4. Module/Peripheral Address Mapping

Target	Boundary Address		Size	Notes
	Low Address	High Address		
UART Controller 0	0x6000_0000	0x6000_0FFF	4 KB	
Reserved	0x6000_1000	0x6000_1FFF		
SPI Controller 1	0x6000_2000	0x6000_2FFF	4 KB	
SPI Controller 0	0x6000_3000	0x6000_3FFF	4 KB	
GPIO	0x6000_4000	0x6000_4FFF	4 KB	
Reserved	0x6000_5000	0x6000_6FFF		

Target	Boundary Address		Size	Notes
	Low Address	High Address		
eFuse Controller	0x6000_7000	0x6000_7FFF	4 KB	
Low-Power Management	0x6000_8000	0x6000_8FFF	4 KB	
IO MUX	0x6000_9000	0x6000_9FFF	4 KB	
Reserved	0x6000_A000	0x6000_EFFF		
I2S Controller 0	0x6000_F000	0x6000_FFFF	4 KB	
UART Controller 1	0x6001_0000	0x6001_0FFF	4 KB	
Reserved	0x6001_1000	0x6001_2FFF		
I2C Controller 0	0x6001_3000	0x6001_3FFF	4 KB	
UHCI0	0x6001_4000	0x6001_4FFF	4 KB	
Reserved	0x6001_5000	0x6001_5FFF		
Remote Control Peripheral	0x6001_6000	0x6001_6FFF	4 KB	
Pulse Count Controller	0x6001_7000	0x6001_7FFF	4 KB	
Reserved	0x6001_8000	0x6001_8FFF		
LED PWM Controller	0x6001_9000	0x6001_9FFF	4 KB	
Reserved	0x6001_A000	0x6001_DFFF		
Motor Control PWM 0	0x6001_E000	0x6001_EFFF	4 KB	
Timer Group 0	0x6001_F000	0x6001_FFFF	4 KB	
Timer Group 1	0x6002_0000	0x6002_0FFF	4 KB	
RTC SLOW Memory	0x6002_1000	0x6002_2FFF	8 KB	
System Timer	0x6002_3000	0x6002_3FFF	4 KB	
SPI Controller 2	0x6002_4000	0x6002_4FFF	4 KB	
SPI Controller 3	0x6002_5000	0x6002_5FFF	4 KB	
APB Controller	0x6002_6000	0x6002_6FFF	4 KB	
I2C Controller 1	0x6002_7000	0x6002_7FFF	4 KB	
SD/MMC Host Controller	0x6002_8000	0x6002_8FFF	4 KB	
Reserved	0x6002_9000	0x6002_AFFF		
Two-wire Automotive Interface	0x6002_B000	0x6002_BFFF	4 KB	
Motor Control PWM 1	0x6002_C000	0x6002_CFFF	4 KB	
I2S Controller 1	0x6002_D000	0x6002_DFFF	4 KB	
UART controller 2	0x6002_E000	0x6002_EFFF	4 KB	
Reserved	0x6002_F000	0x6003_7FFF		
USB Serial/JTAG Controller	0x6003_8000	0x6003_8FFF	4 KB	
USB External Control registers	0x6003_9000	0x6003_9FFF	4 KB	1
AES Accelerator	0x6003_A000	0x6003_AFFF	4 KB	
SHA Accelerator	0x6003_B000	0x6003_BFFF	4 KB	
RSA Accelerator	0x6003_C000	0x6003_CFFF	4 KB	
Digital Signature	0x6003_D000	0x6003_DFFF	4 KB	
HMAC Accelerator	0x6003_E000	0x6003_EFFF	4 KB	
GDMA Controller	0x6003_F000	0x6003_FFFF	4 KB	
ADC Controller	0x6004_0000	0x6004_0FFF	4 KB	
Camera-LCD Controller	0x6004_1000	0x6004_1FFF	4 KB	
Reserved	0x6004_2000	0x6007_FFFF		

Target	Boundary Address		Size	Notes
	Low Address	High Address		
USB core registers	0x6008_0000	0x600B_FFFF	256 KB	1
System Registers	0x600C_0000	0x600C_0FFF	4 KB	
Sensitive Register	0x600C_1000	0x600C_1FFF	4 KB	
Interrupt Matrix	0x600C_2000	0x600C_2FFF	4 KB	
Reserved	0x600C_3000	0x600C_3FFF		
Configure Cache	0x600C_4000	0x600C_BFFF	32 KB	
External Memory Encryption and Decryption	0x600C_C000	0x600C_CFFF	4 KB	
Reserved	0x600C_D000	0x600C_DFFF		
Debug Assist	0x600C_E000	0x600C_EFFF	4 KB	
Reserved	0x600C_F000	0x600C_FFFF		
World Controller	0x600D_0000	0x600D_0FFF	4 KB	

Note:

1. The address space in this module/peripheral is not continuous.
2. The CPU needs to obtain the access permission to a certain module/peripheral when initiating a request to access it, otherwise it may fail. For more information of permission control, please see Chapter 7 *Permission Control (PMS) [to be added later]*.

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

2.1 Overview

The ESP32-S3 chip features 45 physical GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Through GPIO matrix, IO MUX, and RTC IO MUX, peripheral input signals can be from any GPIO pin, and peripheral output signals can be routed to any GPIO pin. Together these modules provide highly configurable I/O.

Note that the 45 GPIO pins are numbered from 0 ~ 21 and 26 ~ 48. All these pins can be configured either as input or output.

2.2 Features

GPIO Matrix Features

- A full-switching matrix between the peripheral input/output signals and the GPIO pins.
- 175 digital peripheral input signals can be sourced from the input of any GPIO pins.
- The output of any GPIO pins can be from any of the 184 digital peripheral output signals.
- Supports signal synchronization for peripheral inputs based on APB clock bus.
- Provides input signal filter.
- Supports sigma delta modulated output.
- Supports GPIO simple input and output.

IO MUX Features

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

RTC IO MUX Features

- Controls low power feature of 22 RTC GPIO pins.
- Controls analog functions of 22 RTC GPIO pins.
- Redirects 22 RTC input/output signals to RTC system.

2.3 Architectural Overview

Figure 2-1 shows in details how IO MUX, RTC IO MUX, and GPIO matrix route signals from pins to peripherals, and from peripherals to pins.

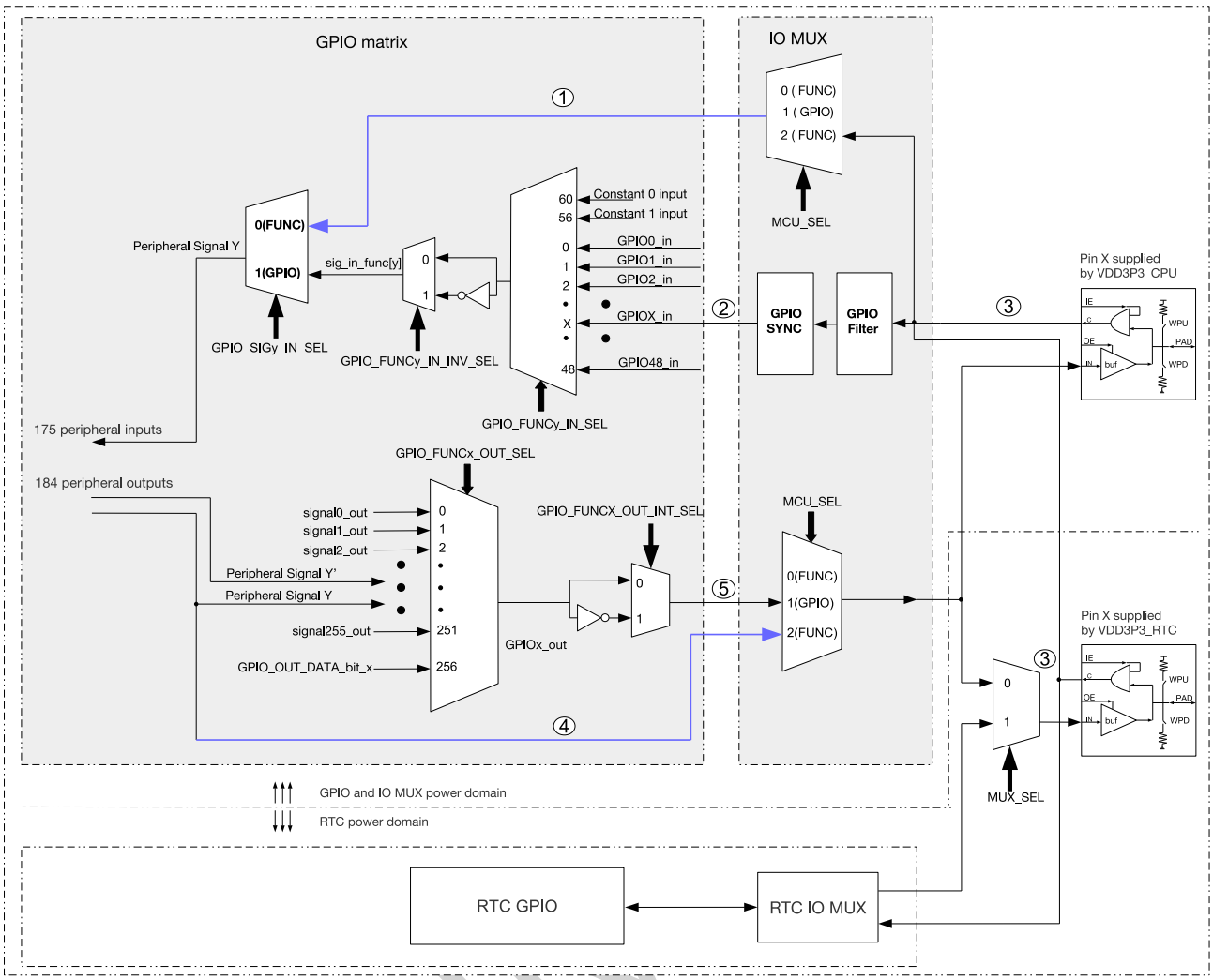


Figure 2-1. Architecture of IO MUX, RTC IO MUX, and GPIO Matrix

1. Only part of peripheral input signals (Y: 0 ~ 3, 7 ~ 13, 15 ~ 16, 101 ~ 110, 120 ~ 123, 155 ~ 159) can bypass GPIO matrix. The other input signals can only be routed to peripherals via GPIO matrix.
2. There are only 45 inputs from GPIO SYNC to GPIO matrix, since ESP32-S3 provides 45 GPIO pins in total.
3. The pins supplied by VDD3P3_CPU or by VDD3P3_RTC are controlled by the signals: IE, OE, WPU, and WPD.
4. Only part of peripheral outputs (0 ~ 13, 15 ~ 16, 101 ~ 110, 120 ~ 126) can be routed to pins bypassing GPIO matrix. See Table 2-2.
5. There are only 45 outputs (GPIO pin X: 0 ~ 21, 26 ~ 48) from GPIO matrix to IO MUX.

Figure 2-2 shows the internal structure of a pad, which is an electrical interface between the chip logic and the GPIO pin. The structure is applicable to all 45 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals.

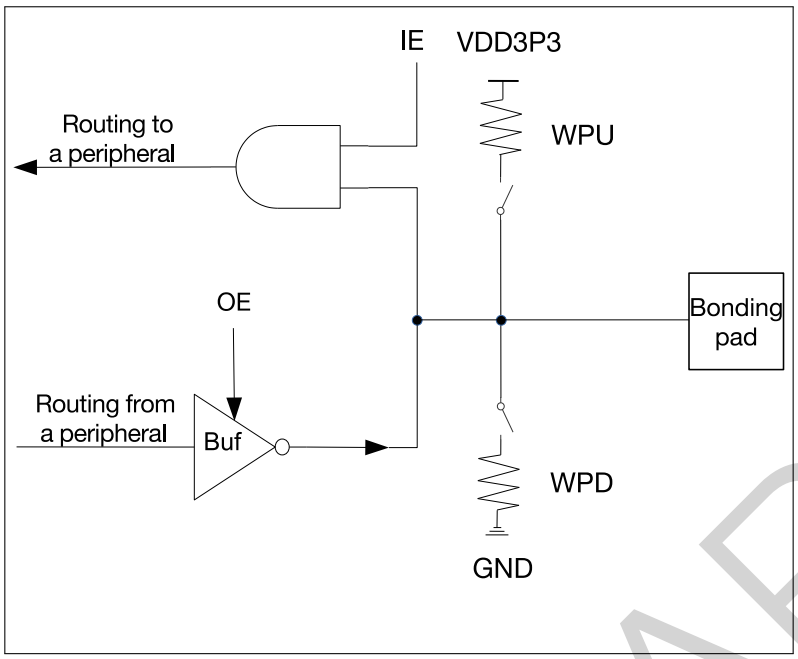


Figure 2-2. Internal Structure of a Pad

Note:

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

2.4 Peripheral Input via GPIO Matrix

2.4.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 45 GPIOs (0 ~ 21, 26 ~ 48), see Table 2-2. Meanwhile, register corresponding to the peripheral signal should be set to receive input signal via GPIO matrix.

2.4.2 Signal Synchronization

When signals are directed from pins using the GPIO matrix, the signals will be synchronized to the APB bus clock by the GPIO SYNC hardware, then go to GPIO matrix. This synchronization applies to all GPIO matrix signals but does not apply when using the IO MUX, see Figure 2-1.

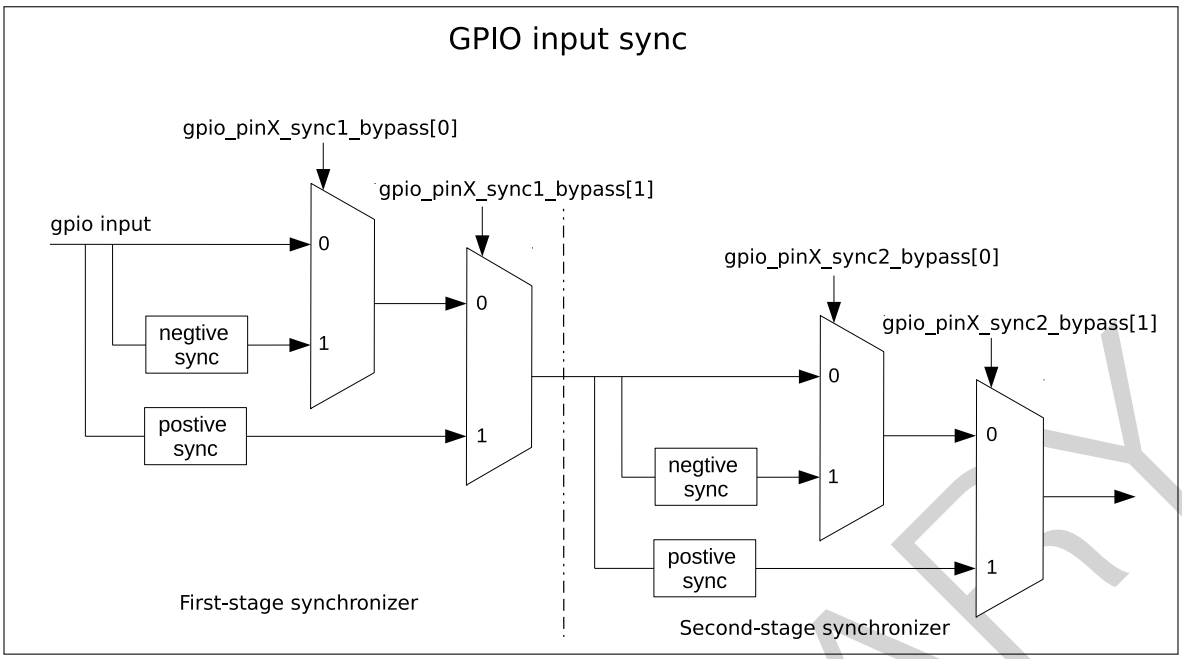


Figure 2-3. GPIO Input Synchronized on APB Clock Rising Edge or on Falling Edge

Figure 2-3 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.

2.4.3 Functional Description

To read GPIO pin *X*[1] 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 desired GPIO pin, i.e. *X* here.

   Note that some peripheral signals have no valid GPIO_SIG*y*_IN_SEL bit, namely, these peripherals can only receive input signals via GPIO matrix.
2. Optionally enable the filter for pin input signals by setting the register IO_MUX_FILTER_EN. Only the signals with a valid width of more than two APB clock cycles can be sampled, see Figure 2-4.

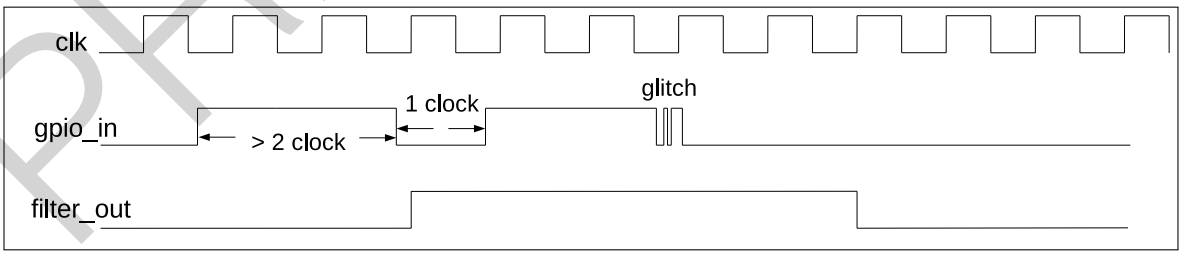


Figure 2-4. Filter Timing of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pin *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 2-3.

- Set GPIO_PIN*x*_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second clock, see Figure 2-3.

4. Configure IO MUX register to enable pin input. For this end, please set IO_MUX_*X*_REG corresponding to GPIO pin *x* as follows:
   - Set IO_MUX_FUN_IE to enable input[2].
   - 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[3] (rmt_sig_in0, signal index 81) to GPIO40, please follow the steps below. Note that GPIO40 is also named as MTDO pin.

1. Set GPIO_SIG81_IN_SEL in register GPIO_FUNC81_IN_SEL_CFG_REG to enable peripheral signal input via GPIO matrix.
2. Set GPIO_FUNC81_IN_SEL in register GPIO_FUNC81_IN_SEL_CFG_REG to 40, i.e. select GPIO40.
3. Set IO_MUX_FUN_IE in register IO_MUX_GPIO40_REG to enable pin input.

Note:

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

2.4.4 Simple GPIO Input

GPIO_IN_REG/GPIO_IN1_REG holds the input values of each GPIO pin. The input value of any GPIO pin can be read at any time without configuring GPIO matrix for a particular peripheral signal. However, it is necessary to enable pin input by setting IO_MUX_FUN_IE in register IO_MUX_*x*_REG corresponding to pin *X*, as described in Section 2.4.2.

2.5 Peripheral Output via GPIO Matrix

2.5.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (0 ~ 255) to one of the 45 GPIOs (0 ~ 21, 26 ~ 48). See Table 2-2.

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

Note:

There is a range of peripheral output signals (208 ~ 212) which are not connected to any peripheral, but to the input signals (208 ~ 212) directly. These can be used to input a signal from one GPIO pin and output directly to another GPIO

pin.

2.5.2 Functional Description

Some of the 256 output signals can be set to go through GPIO matrix into IO MUX and then to a pin. Figure 2-1 illustrates the configuration.

To output peripheral signal *Y* to a particular GPIO pin *X*[1, 2], follow these steps:

1. Configure GPIO_FUNC*x*_OUT_SEL_CFG_REG and GPIO_ENABLE_REG[*x*] corresponding to GPIO pin *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 bit GPIO_FUNC*x*_OEN_SEL in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE/ENABLE1_W1TS_REG, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the SPIQ_oe in column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" in Table 2-2), clear the bit GPIO_FUNC*x*_OEN_SEL instead.
   - Set the corresponding bit in register GPIO_ENABLE/ENABLE1_W1TC_REG to disable the output from the GPIO pin.
2. For an open drain output, set the bit GPIO_PIN*x*_PAD_DRIVER in register GPIO_PIN*x*_REG corresponding to GPIO pin *X*.
3. Configure IO MUX register to enable output via GPIO matrix. Set the IO_MUX_*x*_REG corresponding to GPIO pin *X* as follows:
   - Set the field IO_MUX_MCU_SEL to desired IO MUX function corresponding to GPIO pin *X*. This is Function 1 (GPIO function), numeric value 1, for all pins.
   - Set the field IO_MUX_FUN_DRV 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 IO_MUX_FUN_WPU and IO_MUX_FUN_WPD to enable/disable the internal pull-up/pull-down resistors.

Note:

1. The output signal from a single peripheral can be sent to multiple pins simultaneously.
2. The output signal can be inverted by setting GPIO_FUNC*x*_OUT_INV_SEL.

2.5.3 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] to the desired GPIO output value.

Note:

- GPIO_OUT_REG[21:0] and GPIO_OUT_REG[31:26] correspond to GPIO0 ~ 21 and GPIO26 ~ 31, respectively. GPIO_OUT_REG[25:22] are invalid.
- GPIO_OUT1_REG[16:0] correspond to GPIO32 ~ 48, and GPIO_OUT1_REG[21:17] 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.

2.5.4 Sigma Delta Modulated Output

2.5.4.1 Functional Description

Eight out of the 256 peripheral outputs (index: 93 ~ 100) support 1-bit second-order sigma delta modulation. By default output is enabled for these eight channels. This Sigma Delta modulator can also output PDM (pulse density modulation) signal with configurable duty cycle. The transfer function is:

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

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

- Set GPIO_FUNCTION_CLK_EN to enable the modulator clock.
- Configure GPIO_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.

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

- GPIO_SD*n*_IN = -128, the duty cycle of the output signal is 0%.
- GPIO_SD*n*_IN = 0, the duty cycle of the output signal is near 50%.
- GPIO_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:

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

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).

2.5.4.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pin via GPIO matrix, see Section 2.5.2.
- Enable the modulator clock by setting GPIO_FUNCTION_CLK_EN.
- Configure the divider value by setting GPIO_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting GPIO_SD*n*_IN.

2.6 Dedicated GPIO

2.7 Direct Input and Output via IO MUX

2.7.1 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 pins directly to the peripherals.

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

2.7.2 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 pin must be set to the required pin function. For the list of pin functions, please refer to Section 2.13.
2. Clear GPIO_SIG*n*_IN_SEL to route the input directly to the peripheral.

To bypass GPIO matrix for peripheral output signals, IO_MUX_MCU_SEL for the GPIO pin must be set to the required pin function. For the list of pin functions, please refer to Section 2.13.

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

2.8 RTC IO MUX for Low Power and Analog Input/Output

2.8.1 Overview

ESP32-S3 provides 22 GPIO pins with low power capabilities (RTC) and analog functions, which are handled by the RTC subsystem of ESP32-S3. IO MUX and GPIO matrix are not used for these functions, rather, RTC IO MUX is used to redirect 22 RTC input/output signals to the RTC subsystem.

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

2.8.2 Low Power Capabilities

The pins with RTC functions are controlled by RTC_IO_TOUCH/RTC_PAD*n*_MUX_SEL bit in register RTC_IO_TOUCH/RTC_PAD*n*_REG. By default all bits in these registers are set to 0, routing all input/output signals via IO MUX.

If RTC_IO_TOUCH/RTC_PAD*n*_MUX_SEL is set to 1, then input/output signals to and from that pin is routed to the RTC subsystem. In this mode, RTC_IO_TOUCH/RTC_PAD*n*_REG is used to control RTC low power pins. Note that RTC_IO_TOUCH/RTC_PAD*n*_REG applies the RTC GPIO pin numbering, not the GPIO pin numbering. See Table 2-4 for RTC functions of RTC IO MUX pins.

2.8.3 Analog Functions

When the pin is used for analog purpose, make sure this pin is left floating by configuring the register RTC_IO_TOUCH/RTC_PAD*n*_REG. By such way, external analog signal is connected to internal analog signal via GPIO pin. The configuration is as follows:

- Set RTC_IO_TOUCH/RTC_PAD*n*_MUX_SEL, to select RTC IO MUX to route input and output signals.
- Clear RTC_IO_TOUCH/RTC_PAD*n*_FUN_IE, RTC_IO_TOUCH/RTC_PAD*n*_FUN_RUE, and RTC_IO_TOUCH/RTC_PAD*n*_FUN_RDE, to set this pin floating.
- Configure RTC_IO_TOUCH/RTC_PAD*n*_FUN_SEL to 0, to enable analog function 0.
- Write 1 to RTC_GPIO_ENABLE_W1TC, to clear output enable.

See Table 2-5 for analog functions of RTC IO MUX pins.

2.9 Pin Functions in Light-sleep

Pins may provide different functions when ESP32-S3 is in Light-sleep mode. If IO_MUX_SLP_SEL in register IO_MUX_*n*_REG for a GPIO pin is set to 1, a different set of bits will be used to control the pin when the chip is in Light-sleep mode.

Table 2-1. Bits Used to Control IO MUX Functions in Light-sleep Mode

IO MUX Functions	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
Pull-up Resistor	IO_MUX_FUN_WPU	IO_MUX_MCU_WPU
Pull-down Resistor	IO_MUX_FUN_WPD	IO_MUX_MCU_WPD
Output Enable	OEN_SEL from GPIO matrix *	IO_MUX_MCU_OE

Note:
If IO_MUX_SLP_SEL is set to 0, pin functions remain the same in both normal execution and Light-sleep mode. Please refer to Section 2.5.2 for how to enable output in normal execution.

2.10 Pin Hold Feature

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

Note:

- For digital pins, to maintain pin input/output status in Deep-sleep mode, users can set RTC_CNTL_DG_PAD_FORCE_UNHOLD to 0 before powering down. For RTC pins, 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.
- 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 pin, users can set the corresponding bit in register RTC_CNTL_PAD_HOLD_REG to 1.

2.11 Power Supply and Management of GPIO Pins

2.11.1 Power Supply of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-S3 Datasheet*.

2.11.2 Power Supply Management

Each ESP32-S3 pin is connected to one of the three different power domains.

- VDD3P3_RTC: the input power supply for both RTC and CPU
- VDD3P3_CPU: the input power supply for CPU
- VDD_SPI: configurable input/output power supply

VDD_SPI can be configured to use an internal LDO. The LDO input and output both are 1.8 V. If the LDO is not enabled, VDD_SPI is connected directly to the same power supply as VDD3P3_RTC.

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-S3 Datasheet* sections Power Scheme and Strapping Pins for more details.

Note that GPIO33 ~ GPIO37 can be powered either by VDD_SPI or VDD3P3_CPU.

2.12 Peripheral Signals via GPIO Matrix

Table 2-2 shows the peripheral input/output signals via GPIO matrix.

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

- GPIO_FUNC*n*_OEN_SEL = 1: the output enable is controlled by the corresponding bit *n* of GPIO_ENABLE_REG:
  - GPIO_ENABLE_REG = 0: output is disabled;

  - GPIO_ENABLE_REG = 1: output is enabled;

- GPIO_FUNC*n*_OEN_SEL = 0: use the output enable signal from peripheral, for example SPIQ_oe in the column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" of Table 2-2. Note that the signals such as SPIQ_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in the "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0", it indicates that once the register GPIO_FUNC*n*_OEN_SEL is cleared, the output signal is always enabled by default.

Note:

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

- For input signals, only 0 ~ 3, 7 ~ 48, 51 ~ 54, 58 ~ 62, 66 ~ 71, 73, 81 ~ 84, 89 ~ 92, 101 ~ 110, 116, 120 ~ 123, 129 ~ 131, 133 ~ 152, 155 ~ 187, 192 ~ 199, 208 ~ 228, and 251 ~ 255 are valid.
- For output signals, only 0 ~ 32, 54, 60 ~ 84, 89 ~ 187, 208 ~ 228, and 251 ~ 250 are valid.

Table 2-2. Peripheral Signals via GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe	yes
1	SPID_in	0	yes	SPID_out	SPID_oe	yes
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe	yes
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe	yes
4	-	-	-	SPICLK_out_mux	SPICLK_oe	yes
5	-	-	-	SPICS0_out	SPICS0_oe	yes
6	-	-	-	SPICS1_out	SPICS1_oe	yes
7	SPID4_in	0	yes	SPID4_out	SPID4_oe	yes
8	SPID5_in	0	yes	SPID5_out	SPID5_oe	yes
9	SPID6_in	0	yes	SPID6_out	SPID6_oe	yes
10	SPID7_in	0	yes	SPID7_out	SPID7_oe	yes
11	SPIDQS_in	0	yes	SPIDQS_out	SPIDQS_oe	yes
12	U0RXD_in	0	yes	U0TXD_out	1'd1	yes
13	U0CTS_in	0	yes	U0RTS_out	1'd1	yes
14	U0DSR_in	0	no	U0DTR_out	1'd1	no
15	U1RXD_in	0	yes	U1TXD_out	1'd1	yes
16	U1CTS_in	0	yes	U1RTS_out	1'd1	yes
17	U1DSR_in	0	no	U1DTR_out	1'd1	no
18	U2RXD_in	0	no	U2TXD_out	1'd1	no
19	U2CTS_in	0	no	U2RTS_out	1'd1	no
20	U2DSR_in	0	no	U2DTR_out	1'd1	no
21	I2S1_MCLK_in	0	no	I2S1_MCLK_out	1'd1	no
22	I2S0O_BCK_in	0	no	I2S0O_BCK_out	1'd1	no
23	I2S0_MCLK_in	0	no	I2S0_MCLK_out	1'd1	no
24	I2S0O_WS_in	0	no	I2S0O_WS_out	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
25	I2S0I_SD_in	0	no	I2S0O_SD_out	1'd1	no
26	I2S0I_BCK_in	0	no	I2S0I_BCK_out	1'd1	no
27	I2S0I_WS_in	0	no	I2S0I_WS_out	1'd1	no
28	I2S1O_BCK_in	0	no	I2S1O_BCK_out	1'd1	no
29	I2S1O_WS_in	0	no	I2S1O_WS_out	1'd1	no
30	I2S1I_SD_in	0	no	I2S1O_SD_out	1'd1	no
31	I2S1I_BCK_in	0	no	I2S1I_BCK_out	1'd1	no
32	I2S1I_WS_in	0	no	I2S1I_WS_out	1'd1	no
33	pcnt_sig_ch0_in0	0	no	-	1'd1	no
34	pcnt_sig_ch1_in0	0	no	-	1'd1	no
35	pcnt_ctrl_ch0_in0	0	no	-	1'd1	-
36	pcnt_ctrl_ch1_in0	0	no	-	1'd1	-
37	pcnt_sig_ch0_in1	0	no	-	1'd1	-
38	pcnt_sig_ch1_in1	0	no	-	1'd1	-
39	pcnt_ctrl_ch0_in1	0	no	-	1'd1	-
40	pcnt_ctrl_ch1_in1	0	no	-	1'd1	-
41	pcnt_sig_ch0_in2	0	no	-	1'd1	-
42	pcnt_sig_ch1_in2	0	no	-	1'd1	-
43	pcnt_ctrl_ch0_in2	0	no	-	1'd1	-
44	pcnt_ctrl_ch1_in2	0	no	-	1'd1	-
45	pcnt_sig_ch0_in3	0	no	-	1'd1	-
46	pcnt_sig_ch1_in3	0	no	-	1'd1	-
47	pcnt_ctrl_ch0_in3	0	no	-	1'd1	-
48	pcnt_ctrl_ch1_in3	0	no	-	1'd1	-
49	-	-	-	-	1'd1	-
50	-	-	-	-	1'd1	-
51	I2S0I_SD1_in	0	no	-	1'd1	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
52	I2S0I_SD2_in	0	no	-	1'd1	-
53	I2S0I_SD3_in	0	no	-	1'd1	-
54	Core1_gpio_in7	0	no	Core1_gpio_out7	1'd1	no
55	-	-	-	-	1'd1	-
56	-	-	-	-	1'd1	-
57	-	-	-	-	1'd1	-
58	usb_otg_iddig_in	0	no	-	1'd1	-
59	usb_otg_avalid_in	0	no	-	1'd1	-
60	usb_srp_bvalid_in	0	no	usb_otg_idpullup	1'd1	no
61	usb_otg_vbusvalid_in	0	no	usb_otg_dppulldown	1'd1	no
62	usb_srp_sessend_in	0	no	usb_otg_dmpulldown	1'd1	no
63	-	-	-	usb_otg_drvvbus	1'd1	no
64	-	-	-	usb_srp_chrgvbus	1'd1	no
65	-	-	-	usb_srp_dischrgvbus	1'd1	no
66	SPI3_CLK_in	0	no	SPI3_CLK_out_mux	SPI3_CLK_oe	no
67	SPI3_Q_in	0	no	SPI3_Q_out	SPI3_Q_oe	no
68	SPI3_D_in	0	no	SPI3_D_out	SPI3_D_oe	no
69	SPI3_HD_in	0	no	SPI3_HD_out	SPI3_HD_oe	no
70	SPI3_WP_in	0	no	SPI3_WP_out	SPI3_WP_oe	no
71	SPI3_CS0_in	0	no	SPI3_CS0_out	SPI3_CS0_oe	no
72	-	-	-	SPI3_CS1_out	SPI3_CS1_oe	no
73	ext_adc_start	0	no	ledc_ls_sig_out0	1'd1	no
74	-	-	-	ledc_ls_sig_out1	1'd1	no
75	-	-	-	ledc_ls_sig_out2	1'd1	no
76	-	-	-	ledc_ls_sig_out3	1'd1	no
77	-	-	-	ledc_ls_sig_out4	1'd1	no
78	-	-	-	ledc_ls_sig_out5	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
79	-	-	-	ledc_ls_sig_out6	1'd1	no
80	-	-	-	ledc_ls_sig_out7	1'd1	no
81	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
82	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
83	rmt_sig_in2	0	no	rmt_sig_out2	1'd1	no
84	rmt_sig_in3	0	no	rmt_sig_out3	1'd1	no
85	-	-	-	-	1'd1	-
86	-	-	-	-	1'd1	-
87	-	-	-	-	1'd1	-
88	-	-	-	-	1'd1	-
89	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
90	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
91	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe	no
92	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe	no
93	-	-	-	gpio_sd0_out	1'd1	no
94	-	-	-	gpio_sd1_out	1'd1	no
95	-	-	-	gpio_sd2_out	1'd1	no
96	-	-	-	gpio_sd3_out	1'd1	no
97	-	-	-	gpio_sd4_out	1'd1	no
98	-	-	-	gpio_sd5_out	1'd1	no
99	-	-	-	gpio_sd6_out	1'd1	no
100	-	-	-	gpio_sd7_out	1'd1	no
101	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
102	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
103	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes
104	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
105	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
106	FSPIIO4_in	0	yes	FSPIIO4_out	FSPIIO4_oe	yes
107	FSPIIO5_in	0	yes	FSPIIO5_out	FSPIIO5_oe	yes
108	FSPIIO6_in	0	yes	FSPIIO6_out	FSPIIO6_oe	yes
109	FSPIIO7_in	0	yes	FSPIIO7_out	FSPIIO7_oe	yes
110	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
111	-	-	-	FSPICS1_out	FSPICS1_oe	no
112	-	-	-	FSPICS2_out	FSPICS2_oe	no
113	-	-	-	FSPICS3_out	FSPICS3_oe	no
114	-	-	-	FSPICS4_out	FSPICS4_oe	no
115	-	-	-	FSPICS5_out	FSPICS5_oe	no
116	twai_rx	1	no	twai_tx	1'd1	no
117	-	-	-	twai_bus_off_on	1'd1	no
118	-	-	-	twai_clkout	1'd1	no
119	-	-	-	SUBSPICLK_out_mux	SUBSPICLK_oe	no
120	SUBSPIQ_in	0	yes	SUBSPIQ_out	SUBSPIQ_oe	yes
121	SUBSPID_in	0	yes	SUBSPID_out	SUBSPID_oe	yes
122	SUBSPIHD_in	0	yes	SUBSPIHD_out	SUBSPIHD_oe	yes
123	SUBSPIWP_in	0	yes	SUBSPIWP_out	SUBSPIWP_oe	yes
124	-	-	-	SUBSPICS0_out	SUBSPICS0_oe	yes
125	-	-	-	SUBSPICS1_out	SUBSPICS1_oe	yes
126	-	-	-	FSPIDQS_out	FSPIDQS_oe	yes
127	-	-	-	SPI3_CS2_out	SPI3_CS2_oe	no
128	-	-	-	I2S0O_SD1_out	1'd1	no
129	Core1_gpio_in0	0	no	Core1_gpio_out0	1'd1	no
130	Core1_gpio_in1	0	no	Core1_gpio_out1	1'd1	no
131	Core1_gpio_in2	0	no	Core1_gpio_out2	1'd1	no
132	-	-	-	LCD_CS	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
133	CAM_DATA_in0	0	no	LCD_DATA_out0	1'd1	no
134	CAM_DATA_in1	0	no	LCD_DATA_out1	1'd1	no
135	CAM_DATA_in2	0	no	LCD_DATA_out2	1'd1	no
136	CAM_DATA_in3	0	no	LCD_DATA_out3	1'd1	no
137	CAM_DATA_in4	0	no	LCD_DATA_out4	1'd1	no
138	CAM_DATA_in5	0	no	LCD_DATA_out5	1'd1	no
139	CAM_DATA_in6	0	no	LCD_DATA_out6	1'd1	no
140	CAM_DATA_in7	0	no	LCD_DATA_out7	1'd1	no
141	CAM_DATA_in8	0	no	LCD_DATA_out8	1'd1	no
142	CAM_DATA_in9	0	no	LCD_DATA_out9	1'd1	no
143	CAM_DATA_in10	0	no	LCD_DATA_out10	1'd1	no
144	CAM_DATA_in11	0	no	LCD_DATA_out11	1'd1	no
145	CAM_DATA_in12	0	no	LCD_DATA_out12	1'd1	no
146	CAM_DATA_in13	0	no	LCD_DATA_out13	1'd1	no
147	CAM_DATA_in14	0	no	LCD_DATA_out14	1'd1	no
148	CAM_DATA_in15	0	no	LCD_DATA_out15	1'd1	no
149	CAM_PCLK	0	no	CAM_CLK	1'd1	no
150	CAM_H_ENABLE	0	no	LCD_H_ENABLE	1'd1	no
151	CAM_H_SYNC	0	no	LCD_H_SYNC	1'd1	no
152	CAM_V_SYNC	0	no	LCD_V_SYNC	1'd1	no
153	-	-	-	LCD_DC	1'd1	no
154	-	-	-	LCD_PCLK	1'd1	no
155	SUBSPID4_in	0	yes	SUBSPID4_out	SUBSPID4_oe	no
156	SUBSPID5_in	0	yes	SUBSPID5_out	SUBSPID5_oe	no
157	SUBSPID6_in	0	yes	SUBSPID6_out	SUBSPID6_oe	no
158	SUBSPID7_in	0	yes	SUBSPID7_out	SUBSPID7_oe	no
159	SUBSPIDQS_in	0	yes	SUBSPIDQS_out	SUBSPIDQS_oe	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
160	pwm0_sync0_in	0	no	pwm0_out0a	1'd1	no
161	pwm0_sync1_in	0	no	pwm0_out0b	1'd1	no
162	pwm0_sync2_in	0	no	pwm0_out1a	1'd1	no
163	pwm0_f0_in	0	no	pwm0_out1b	1'd1	no
164	pwm0_f1_in	0	no	pwm0_out2a	1'd1	no
165	pwm0_f2_in	0	no	pwm0_out2b	1'd1	no
166	pwm0_cap0_in	0	no	pwm1_out0a	1'd1	no
167	pwm0_cap1_in	0	no	pwm1_out0b	1'd1	no
168	pwm0_cap2_in	0	no	pwm1_out1a	1'd1	no
169	pwm1_sync0_in	0	no	pwm1_out1b	1'd1	no
170	pwm1_sync1_in	0	no	pwm1_out2a	1'd1	no
171	pwm1_sync2_in	0	no	pwm1_out2b	1'd1	no
172	pwm1_f0_in	0	no	sdhost_cclk_out_1	1'd1	no
173	pwm1_f1_in	0	no	sdhost_cclk_out_2	1'd1	no
174	pwm1_f2_in	0	no	sdhost_rst_n_1	1'd1	no
175	pwm1_cap0_in	0	no	sdhost_rst_n_2	1'd1	no
176	pwm1_cap1_in	0	no	sdhost_ccmd_od_pullup_en_n	1'd1	no
177	pwm1_cap2_in	0	no	sdio_tohost_int_out	1'd1	no
178	sdhost_ccmd_in_1	1	no	sdhost_ccmd_out_1	sdhost_ccmd_out_en_1	no
179	sdhost_ccmd_in_2	1	no	sdhost_ccmd_out_2	sdhost_ccmd_out_en_2	no
180	sdhost_cdata_in_10	1	no	sdhost_cdata_out_10	sdhost_cdata_out_en_10	no
181	sdhost_cdata_in_11	1	no	sdhost_cdata_out_11	sdhost_cdata_out_en_11	no
182	sdhost_cdata_in_12	1	no	sdhost_cdata_out_12	sdhost_cdata_out_en_12	no
183	sdhost_cdata_in_13	1	no	sdhost_cdata_out_13	sdhost_cdata_out_en_13	no
184	sdhost_cdata_in_14	1	no	sdhost_cdata_out_14	sdhost_cdata_out_en_14	no
185	sdhost_cdata_in_15	1	no	sdhost_cdata_out_15	sdhost_cdata_out_en_15	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
186	sdhost_cdata_in_16	1	no	sdhost_cdata_out_16	sdhost_cdata_out_en_16	no
187	sdhost_cdata_in_17	1	no	sdhost_cdata_out_17	sdhost_cdata_out_en_17	no
188	-	-	-	-	1'd1	-
189	-	-	-	-	1'd1	-
190	-	-	-	-	1'd1	-
191	-	-	-	-	1'd1	-
192	sdhost_data_strobe_1	0	no	-	1'd1	-
193	sdhost_data_strobe_2	0	no	-	1'd1	-
194	sdhost_card_detect_n_1	0	no	-	1'd1	-
195	sdhost_card_detect_n_2	0	no	-	1'd1	-
196	sdhost_card_write_prt_1	0	no	-	1'd1	-
197	sdhost_card_write_prt_2	0	no	-	1'd1	-
198	sdhost_card_int_n_1	0	no	-	1'd1	-
199	sdhost_card_int_n_2	0	no	-	1'd1	-
200	-	-	-	-	1'd1	no
201	-	-	-	-	1'd1	no
202	-	-	-	-	1'd1	no
203	-	-	-	-	1'd1	no
204	-	-	-	-	1'd1	no
205	-	-	-	-	1'd1	no
206	-	-	-	-	1'd1	no
207	-	-	-	-	1'd1	no
208	sig_in_func_208	0	no	sig_in_func208	1'd1	no
209	sig_in_func_209	0	no	sig_in_func209	1'd1	no
210	sig_in_func_210	0	no	sig_in_func210	1'd1	no
211	sig_in_func_211	0	no	sig_in_func211	1'd1	no
212	sig_in_func_212	0	no	sig_in_func212	1'd1	no

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
213	sdhost_cdata_in_20	1	no	sdhost_cdata_out_20	sdhost_cdata_out_en_20	no
214	sdhost_cdata_in_21	1	no	sdhost_cdata_out_21	sdhost_cdata_out_en_21	no
215	sdhost_cdata_in_22	1	no	sdhost_cdata_out_22	sdhost_cdata_out_en_22	no
216	sdhost_cdata_in_23	1	no	sdhost_cdata_out_23	sdhost_cdata_out_en_23	no
217	sdhost_cdata_in_24	1	no	sdhost_cdata_out_24	sdhost_cdata_out_en_24	no
218	sdhost_cdata_in_25	1	no	sdhost_cdata_out_25	sdhost_cdata_out_en_25	no
219	sdhost_cdata_in_26	1	no	sdhost_cdata_out_26	sdhost_cdata_out_en_26	no
220	sdhost_cdata_in_27	1	no	sdhost_cdata_out_27	sdhost_cdata_out_en_27	no
221	pro_alonegpio_in0	0	no	pro_alonegpio_out0	1'd1	no
222	pro_alonegpio_in1	0	no	pro_alonegpio_out1	1'd1	no
223	pro_alonegpio_in2	0	no	pro_alonegpio_out2	1'd1	no
224	pro_alonegpio_in3	0	no	pro_alonegpio_out3	1'd1	no
225	pro_alonegpio_in4	0	no	pro_alonegpio_out4	1'd1	no
226	pro_alonegpio_in5	0	no	pro_alonegpio_out5	1'd1	no
227	pro_alonegpio_in6	0	no	pro_alonegpio_out6	1'd1	no
228	pro_alonegpio_in7	0	no	pro_alonegpio_out7	1'd1	no
229	-	-	-	-	1'd1	-
230	-	-	-	-	1'd1	-
231	-	-	-	-	1'd1	-
232	-	-	-	-	1'd1	-
233	-	-	-	-	1'd1	-
234	-	-	-	-	1'd1	-
235	-	-	-	-	1'd1	-
236	-	-	-	-	1'd1	-
237	-	-	-	-	1'd1	-
238	-	-	-	-	1'd1	-
239	-	-	-	-	1'd1	-

Signal No.	Input Signal	Default value	Direct Input via IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output via IO MUX
240	-	-	-	-	1'd1	-
241	-	-	-	-	1'd1	-
242	-	-	-	-	1'd1	-
243	-	-	-	-	1'd1	-
244	-	-	-	-	1'd1	-
245	-	-	-	-	1'd1	-
246	-	-	-	-	1'd1	-
247	-	-	-	-	1'd1	-
248	-	-	-	-	1'd1	-
249	-	-	-	-	1'd1	-
250	-	-	-	-	1'd1	-
251	usb_jtag_tdo_bridge	0	no	usb_jtag_trst	1'd1	no
252	Core1_gpio_in3	0	no	Core1_gpio_out3	1'd1	no
253	Core1_gpio_in4	0	no	Core1_gpio_out4	1'd1	no
254	Core1_gpio_in5	0	no	Core1_gpio_out5	1'd1	no
255	Core1_gpio_in6	0	no	Core1_gpio_out6	1'd1	no

2.13 IO MUX Function List

Table 2-3 shows the IO MUX functions of each GPIO pin.

Table 2-3. IO MUX Pin Functions

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

GPIO	Pin Name	Function 0	Function 1	Function 2	Function 3	Function 4	DRV	RST	Notes
43	U0TXD	U0TXD	GPIO43	CLK_OUT1	-	-	2	4	-
44	U0RXD	U0RXD	GPIO44	CLK_OUT2	-	-	2	3	-
45	GPIO45	GPIO45	GPIO45	-	-	-	2	2	-
46	GPIO46	GPIO46	GPIO46	-	-	-	2	2	-
47	SPICLK_P	SPICLK_DIFF	GPIO47	SUBSPICLK_P_DIFF	-	-	2	1	-
48	SPICLK_N	SPICLK_DIFF	GPIO48	SUBSPICLK_N_DIFF	-	-	2	1	-

Drive Strength

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

- **0** - Drive current = ~5 mA
- **1** - Drive current = ~10 mA
- **2** - Drive current = ~20 mA
- **3** - Drive current = ~40 mA

Reset Configurations

"RST" column shows the default configuration of each pin after reset:

- **0** - IE = 0 (input disabled)
- **1** - IE = 1 (input enabled)
- **2** - IE = 1, WPD = 1 (input enabled, pull-down resistor enabled)
- **3** - IE = 1, WPU = 1 (input enabled, pull-up resistor enabled)
- **4** - OE = 1, WPU = 1 (output enabled, pull-up resistor enabled)
- **1*** - If EFUSE_DIS_JTAG = 1, the pin MTCK is left floating after reset, i.e. IE = 1. If EFUSE_DIS_JTAG = 0, the pin MTCK is connected to internal pull-up resistor, i.e. IE = 1, WPU = 1.

Note:

- **R** - Pin has RTC/analog functions via RTC IO MUX.

Please refer to Appendix A – ESP32-S3 Pin Lists in *ESP32-S3 Datasheet* for more details.

2.14 RTC IO MUX Pin List

Table 2-4 shows the RTC pins, their corresponding GPIO pins and RTC functions.

Table 2-4. RTC Functions of RTC IO MUX Pins

RTC GPIO Num	GPIO Num	Pin Name	RTC Function			
			0	1	2	3
0	0	GPIO0	RTC_GPIO0	-	-	sar_i2c_scl_0[a]
1	1	GPIO1	RTC_GPIO1	-	-	sar_i2c_sda_0[a]
2	2	GPIO2	RTC_GPIO2	-	-	sar_i2c_scl_1[a]
3	3	GPIO3	RTC_GPIO3	-	-	sar_i2c_sda_1[a]

Cont'd on next page

Table 2-4 – cont'd from previous page

RTC GPIO Num	GPIO Num	Pin Name	RTC Function			
			0	1	2	3
4	4	GPIO4	RTC_GPIO4	-	-	-
5	5	GPIO5	RTC_GPIO5	-	-	-
6	6	GPIO6	RTC_GPIO6	-	-	-
7	7	GPIO7	RTC_GPIO7	-	-	-
8	8	GPIO8	RTC_GPIO8	-	-	-
9	9	GPIO9	RTC_GPIO9	-	-	-
10	10	GPIO10	RTC_GPIO10	-	-	-
11	11	GPIO11	RTC_GPIO11	-	-	-
12	12	GPIO12	RTC_GPIO12	-	-	-
13	13	GPIO13	RTC_GPIO13	-	-	-
14	14	GPIO14	RTC_GPIO14	-	-	-
15	15	XTAL_32K_P	RTC_GPIO15	-	-	-
16	16	XTAL_32K_N	RTC_GPIO16	-	-	-
17	17	GPIO17	RTC_GPIO17	-	-	-
18	18	GPIO18	RTC_GPIO18	-	-	-
19	19	GPIO19	RTC_GPIO19	-	-	-
20	20	GPIO20	RTC_GPIO20	-	-	-
21	21	GPIO21	RTC_GPIO21	-	-	-

[a] For more information on the configuration of sar_i2c_xx, see Section RTC I2C Controller in Chapter 14 *ULP Coprocessor (ULP-FSM, ULP-RISC-V) [to be added later]*.

Table 2-5 shows the RTC pins, their corresponding GPIO pins and analog functions.

Table 2-5. Analog Functions of RTC IO MUX Pins

RTC GPIO Num	GPIO Num	Pin Name	Analog Function	
			0	1
0	0	GPIO0	-	-
1	1	GPIO1	TOUCH1	ADC1_CH0
2	2	GPIO2	TOUCH2	ADC1_CH1
3	3	GPIO3	TOUCH3	ADC1_CH2
4	4	GPIO4	TOUCH4	ADC1_CH3
5	5	GPIO5	TOUCH5	ADC1_CH4
6	6	GPIO6	TOUCH6	ADC1_CH5
7	7	GPIO7	TOUCH7	ADC1_CH6
8	8	GPIO8	TOUCH8	ADC1_CH7
9	9	GPIO9	TOUCH9	ADC1_CH8
10	10	GPIO10	TOUCH10	ADC1_CH9
11	11	GPIO11	TOUCH11	ADC2_CH0
12	12	GPIO12	TOUCH12	ADC2_CH1
13	13	GPIO13	TOUCH13	ADC2_CH2
14	14	GPIO14	TOUCH14	ADC2_CH3

RTC GPIO Num	GPIO Num	Pin Name	Analog Function	
			0	1
15	15	XTAL_32K_P	XTAL_32K_P	ADC2_CH4
16	16	XTAL_32K_N	XTAL_32K_N	ADC2_CH5
17	17	GPIO17	-	ADC2_CH6
18	18	GPIO18	-	ADC2_CH7
19	19	GPIO19	USB_D-	ADC2_CH8
20	20	GPIO20	USB_D+	ADC2_CH9
21	21	GPIO21	-	-

2.15 Register Summary

2.15.1 GPIO Matrix Register Summary

The addresses in this section are relative to the GPIO base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Address	Access
GPIO Configuration Registers			
GPIO_BT_SELECT_REG	GPIO bit select 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 ~ 48 output register	0x0010	R/W
GPIO_OUT1_W1TS_REG	GPIO32 ~ 48 output bit set register	0x0014	WO
GPIO_OUT1_W1TC_REG	GPIO32 ~ 48 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
GPIO_ENABLE1_REG	GPIO32 ~ 48 output enable register	0x002C	R/W
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ 48 output enable bit set register	0x0030	WO
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ 48 output enable bit clear register	0x0034	WO
GPIO_STRAP_REG	Strapping pin value register	0x0038	RO
GPIO_IN_REG	GPIO0 ~ 31 input register	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ 48 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_PIN46_REG	Configuration for GPIO pin 46	0x012C	R/W
GPIO_PIN47_REG	Configuration for GPIO pin 47	0x0130	R/W
GPIO_PIN48_REG	Configuration for GPIO pin 48	0x0134	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

Name	Description	Address	Access
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_FUNC47_OUT_SEL_CFG_REG	Peripheral output selection for GPIO47	0x0610	R/W
GPIO_FUNC48_OUT_SEL_CFG_REG	Peripheral output selection for GPIO47	0x0614	R/W
GPIO_CLOCK_GATE_REG	GPIO clock gating register	0x062C	R/W
Interrupt Status Registers			
GPIO_STATUS_REG	GPIO0 ~ 31 interrupt status register	0x0044	R/W
GPIO_STATUS1_REG	GPIO32 ~ 48 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 ~ 48 PRO_CPU interrupt status register	0x0068	RO
GPIO_PCPU_NMI_INT1_REG	GPIO32 ~ 48 PRO_CPU non-maskable interrupt status register	0x006C	RO
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 ~ 48 interrupt status bit set register	0x0054	WO
GPIO_STATUS1_W1TC_REG	GPIO32 ~ 48 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 ~ 48 interrupt source register	0x0150	RO
Version Register			
GPIO_DATE_REG	Version control register	0x06FC	R/W

2.15.2 IO MUX Register Summary

The addresses in this section are relative to the IO MUX base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Address	Access
IO_MUX_PIN_CTRL	Clock output configuration register	0x0000	R/W
IO_MUX_GPIO0_REG	Configuration register for pin GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	Configuration register for pin GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	Configuration register for pin GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	Configuration register for pin GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	Configuration register for pin GPIO4	0x0014	R/W

Name	Description	Address	Access
IO_MUX_GPIO5_REG	Configuration register for pin GPIO5	0x0018	R/W
IO_MUX_GPIO6_REG	Configuration register for pin GPIO6	0x001C	R/W
IO_MUX_GPIO7_REG	Configuration register for pin GPIO7	0x0020	R/W
IO_MUX_GPIO8_REG	Configuration register for pin GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	Configuration register for pin GPIO9	0x0028	R/W
IO_MUX_GPIO10_REG	Configuration register for pin GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	Configuration register for pin GPIO11	0x0030	R/W
IO_MUX_GPIO12_REG	Configuration register for pin GPIO12	0x0034	R/W
IO_MUX_GPIO13_REG	Configuration register for pin GPIO13	0x0038	R/W
IO_MUX_GPIO14_REG	Configuration register for pin GPIO14	0x003C	R/W
IO_MUX_GPIO15_REG	Configuration register for pad XTAL_32K_P	0x0040	R/W
IO_MUX_GPIO16_REG	Configuration register for pad XTAL_32K_N	0x0044	R/W
IO_MUX_GPIO17_REG	Configuration register for pad DAC_1	0x0048	R/W
IO_MUX_GPIO18_REG	Configuration register for pad DAC_2	0x004C	R/W
IO_MUX_GPIO19_REG	Configuration register for pin GPIO19	0x0050	R/W
IO_MUX_GPIO20_REG	Configuration register for pin GPIO20	0x0054	R/W
IO_MUX_GPIO21_REG	Configuration register for pin GPIO21	0x0058	R/W
IO_MUX_GPIO26_REG	Configuration register for pad SPICS1	0x006C	R/W
IO_MUX_GPIO27_REG	Configuration register for pad SPIHD	0x0070	R/W
IO_MUX_GPIO28_REG	Configuration register for pad SPIWP	0x0074	R/W
IO_MUX_GPIO29_REG	Configuration register for pad SPICS0	0x0078	R/W
IO_MUX_GPIO30_REG	Configuration register for pad SPICLK	0x007C	R/W
IO_MUX_GPIO31_REG	Configuration register for pad SPIQ	0x0080	R/W
IO_MUX_GPIO32_REG	Configuration register for pad SPID	0x0084	R/W
IO_MUX_GPIO33_REG	Configuration register for pin GPIO33	0x0088	R/W
IO_MUX_GPIO34_REG	Configuration register for pin GPIO34	0x008C	R/W
IO_MUX_GPIO35_REG	Configuration register for pin GPIO35	0x0090	R/W
IO_MUX_GPIO36_REG	Configuration register for pin GPIO36	0x0094	R/W
IO_MUX_GPIO37_REG	Configuration register for pin GPIO37	0x0098	R/W
IO_MUX_GPIO38_REG	Configuration register for pin GPIO38	0x009C	R/W
IO_MUX_GPIO39_REG	Configuration register for pad MTCK	0x00A0	R/W
IO_MUX_GPIO40_REG	Configuration register for pad MTDO	0x00A4	R/W
IO_MUX_GPIO41_REG	Configuration register for pad MTDI	0x00A8	R/W
IO_MUX_GPIO42_REG	Configuration register for pad MTMS	0x00AC	R/W
IO_MUX_GPIO43_REG	Configuration register for pad U0TXD	0x00B0	R/W
IO_MUX_GPIO44_REG	Configuration register for pad U0RXD	0x00B4	R/W
IO_MUX_GPIO45_REG	Configuration register for pin GPIO45	0x00B8	R/W
IO_MUX_GPIO46_REG	Configuration register for pin GPIO46	0x00BC	R/W
IO_MUX_GPIO47_REG	Configuration register for pin GPIO47	0x00C0	R/W
IO_MUX_GPIO48_REG	Configuration register for pin GPIO48	0x00C4	R/W

2.15.3 SDM Output Register Summary

The addresses in this section are relative to (GPIO base address provided in Table 1-4 in Chapter 1 *System and Memory* + 0x0F00).

Name	Description	Address	Access
Configuration Registers			
GPIO_SIGMADELTA0_REG	Duty Cycle Configure Register of SDM0	0x0000	R/W
GPIO_SIGMADELTA1_REG	Duty Cycle Configure Register of SDM1	0x0004	R/W
GPIO_SIGMADELTA2_REG	Duty Cycle Configure Register of SDM2	0x0008	R/W
GPIO_SIGMADELTA3_REG	Duty Cycle Configure Register of SDM3	0x000C	R/W
GPIO_SIGMADELTA4_REG	Duty Cycle Configure Register of SDM4	0x0010	R/W
GPIO_SIGMADELTA5_REG	Duty Cycle Configure Register of SDM5	0x0014	R/W
GPIO_SIGMADELTA6_REG	Duty Cycle Configure Register of SDM6	0x0018	R/W
GPIO_SIGMADELTA7_REG	Duty Cycle Configure Register of SDM7	0x001C	R/W
GPIO_SIGMADELTA_CG_REG	Clock Gating Configure Register	0x0020	R/W
GPIO_SIGMADELTA_MISC_REG	MISC Register	0x0024	R/W
GPIO_SIGMADELTA_VERSION_REG	Version Control Register	0x0028	R/W

2.15.4 RTC IO MUX Register Summary

The addresses in this section are relative to (Low-Power Management base address provided in Table 1-4 in Chapter 1 *System and Memory* + 0x0400).

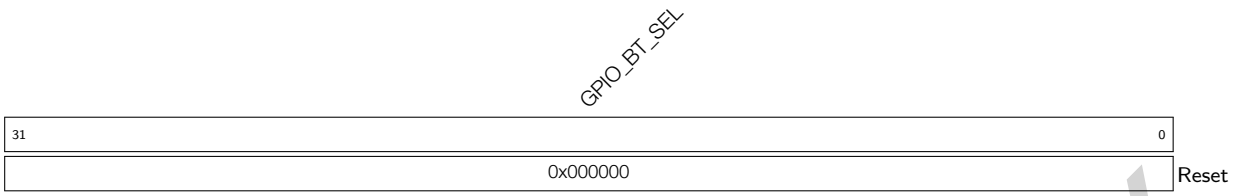
Name	Description	Address	Access
GPIO configuration/data registers			
RTC_GPIO_OUT_REG	RTC GPIO output register	0x0000	R/W
RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register	0x0004	WO
RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register	0x0008	WO
RTC_GPIO_ENABLE_REG	RTC GPIO output enable register	0x000C	R/W
RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit set register	0x0010	WO
RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register	0x0014	WO
RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register	0x0018	R/W
RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register	0x001C	WO
RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register	0x0020	WO
RTC_GPIO_IN_REG	RTC GPIO input register	0x0024	RO
RTC_GPIO_PIN0_REG	RTC configuration for pin 0	0x0028	R/W
RTC_GPIO_PIN1_REG	RTC configuration for pin 1	0x002C	R/W
RTC_GPIO_PIN2_REG	RTC configuration for pin 2	0x0030	R/W
RTC_GPIO_PIN3_REG	RTC configuration for pin 3	0x0034	R/W
RTC_GPIO_PIN4_REG	RTC configuration for pin 4	0x0038	R/W
RTC_GPIO_PIN5_REG	RTC configuration for pin 5	0x003C	R/W
RTC_GPIO_PIN6_REG	RTC configuration for pin 6	0x0040	R/W
RTC_GPIO_PIN7_REG	RTC configuration for pin 7	0x0044	R/W
RTC_GPIO_PIN8_REG	RTC configuration for pin 8	0x0048	R/W
RTC_GPIO_PIN9_REG	RTC configuration for pin 9	0x004C	R/W

Name	Description	Address	Access
RTC_GPIO_PIN10_REG	RTC configuration for pin 10	0x0050	R/W
RTC_GPIO_PIN11_REG	RTC configuration for pin 11	0x0054	R/W
RTC_GPIO_PIN12_REG	RTC configuration for pin 12	0x0058	R/W
RTC_GPIO_PIN13_REG	RTC configuration for pin 13	0x005C	R/W
RTC_GPIO_PIN14_REG	RTC configuration for pin 14	0x0060	R/W
RTC_GPIO_PIN15_REG	RTC configuration for pin 15	0x0064	R/W
RTC_GPIO_PIN16_REG	RTC configuration for pin 16	0x0068	R/W
RTC_GPIO_PIN17_REG	RTC configuration for pin 17	0x006C	R/W
RTC_GPIO_PIN18_REG	RTC configuration for pin 18	0x0070	R/W
RTC_GPIO_PIN19_REG	RTC configuration for pin 19	0x0074	R/W
RTC_GPIO_PIN20_REG	RTC configuration for pin 20	0x0078	R/W
RTC_GPIO_PIN21_REG	RTC configuration for pin 21	0x007C	R/W
GPIO RTC function configuration registers			
RTC_IO_TOUCH_PAD0_REG	Touch pin 0 configuration register	0x0084	R/W
RTC_IO_TOUCH_PAD1_REG	Touch pin 1 configuration register	0x0088	R/W
RTC_IO_TOUCH_PAD2_REG	Touch pin 2 configuration register	0x008C	R/W
RTC_IO_TOUCH_PAD3_REG	Touch pin 3 configuration register	0x0090	R/W
RTC_IO_TOUCH_PAD4_REG	Touch pin 4 configuration register	0x0094	R/W
RTC_IO_TOUCH_PAD5_REG	Touch pin 5 configuration register	0x0098	R/W
RTC_IO_TOUCH_PAD6_REG	Touch pin 6 configuration register	0x009C	R/W
RTC_IO_TOUCH_PAD7_REG	Touch pin 7 configuration register	0x00A0	R/W
RTC_IO_TOUCH_PAD8_REG	Touch pin 8 configuration register	0x00A4	R/W
RTC_IO_TOUCH_PAD9_REG	Touch pin 9 configuration register	0x00A8	R/W
RTC_IO_TOUCH_PAD10_REG	Touch pin 10 configuration register	0x00AC	R/W
RTC_IO_TOUCH_PAD11_REG	Touch pin 11 configuration register	0x00B0	R/W
RTC_IO_TOUCH_PAD12_REG	Touch pin 12 configuration register	0x00B4	R/W
RTC_IO_TOUCH_PAD13_REG	Touch pin 13 configuration register	0x00B8	R/W
RTC_IO_TOUCH_PAD14_REG	Touch pin 14 configuration register	0x00BC	R/W
RTC_IO_XTAL_32P_PAD_REG	32 kHz crystal P-pin configuration register	0x00C0	R/W
RTC_IO_XTAL_32N_PAD_REG	32 kHz crystal N-pin configuration register	0x00C4	R/W
RTC_IO_RTC_PAD17_REG	RTC pin 17 configuration register	0x00C8	R/W
RTC_IO_RTC_PAD18_REG	RTC pin 18 configuration register	0x00CC	R/W
RTC_IO_RTC_PAD19_REG	RTC pin 19 configuration register	0x00D0	R/W
RTC_IO_RTC_PAD20_REG	RTC pin 20 configuration register	0x00D4	R/W
RTC_IO_RTC_PAD21_REG	RTC pin 21 configuration register	0x00D8	R/W
RTC_IO_XTL_EXT_CTR_REG	Crystal power down enable GPIO source	0x00E0	R/W
RTC_IO_SAR_I2C_IO_REG	RTC I2C pin selection	0x00E4	R/W
Version Register			
RTC_IO_DATE_REG	Version control register	0x01FC	R/W

2.16 Registers

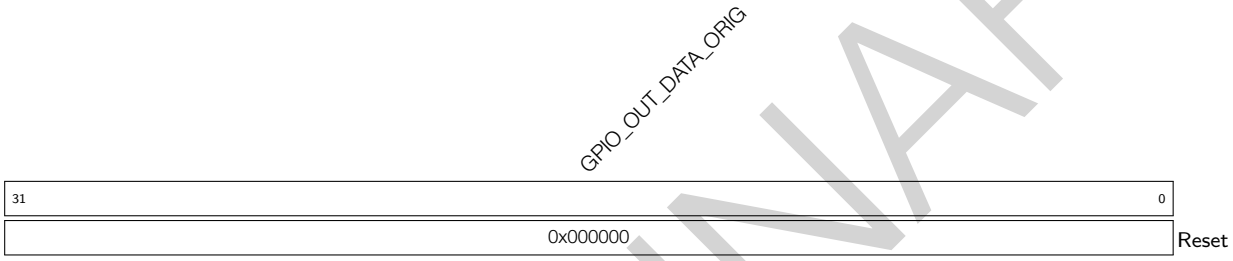
2.16.1 GPIO Matrix Registers

Register 2.1. GPIO_BT_SELECT_REG (0x0000)



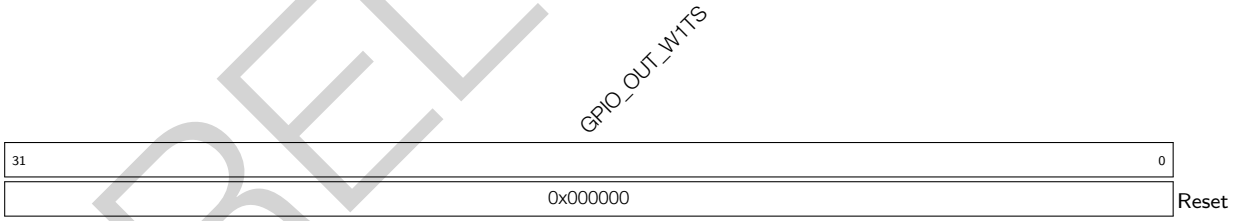
GPIO_BT_SEL Reserved (R/W)

Register 2.2. GPIO_OUT_REG (0x0004)



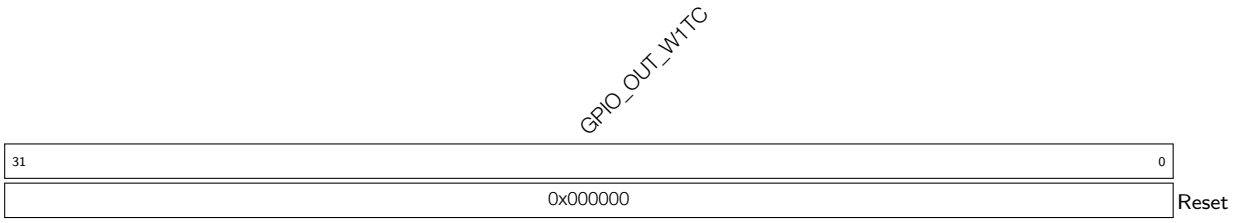
GPIO_OUT_DATA_ORIG GPIO0 ~ 21 and GPIO26 ~ 31 output values in simple GPIO output mode. The values of bit0 ~ bit21 correspond to the output values of GPIO0 ~ 21, and bit26 ~ bit31 to GPIO26 ~ 31. Bit22 ~ bit25 are invalid. (R/W)

Register 2.3. GPIO_OUT_W1TS_REG (0x0008)



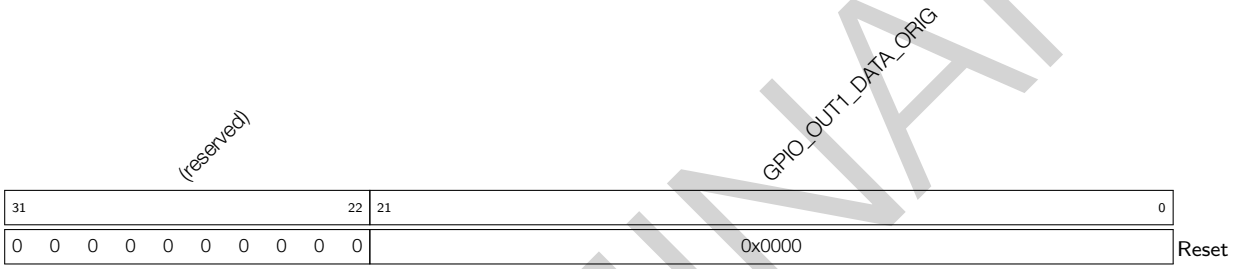
GPIO_OUT_W1TS GPIO0 ~ 31 output set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT_REG. (WO)

Register 2.4. GPIO_OUT_W1TC_REG (0x000C)



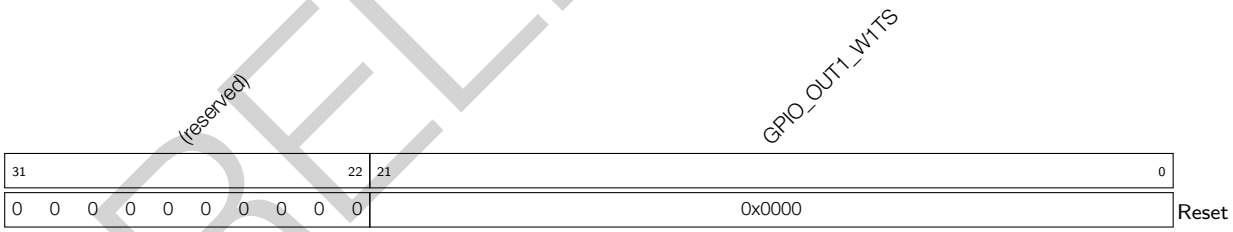
GPIO_OUT_W1TC GPIO0 ~ 31 output clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT_REG. (WO)

Register 2.5. GPIO_OUT1_REG (0x0010)



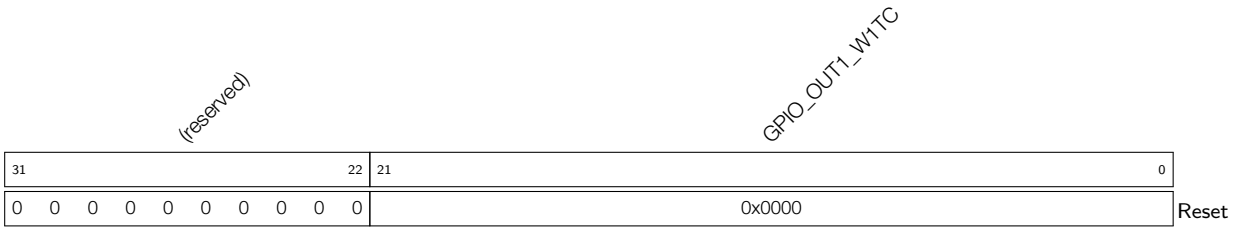
GPIO_OUT1_DATA_ORIG GPIO32 ~ 48 output value in simple GPIO output mode. The values of bit0 ~ bit16 correspond to GPIO32 ~ GPIO48. Bit17 ~ bit21 are invalid. (R/W)

Register 2.6. GPIO_OUT1_W1TS_REG (0x0014)



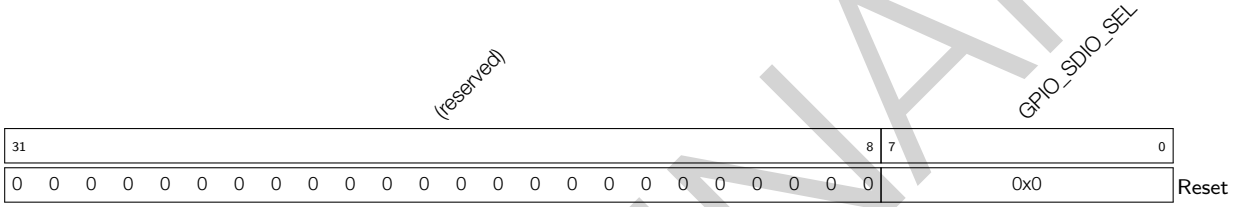
GPIO_OUT1_W1TS GPIO32 ~ 48 output value set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT1_REG. (WO)

Register 2.7. GPIO_OUT1_W1TC_REG (0x0018)



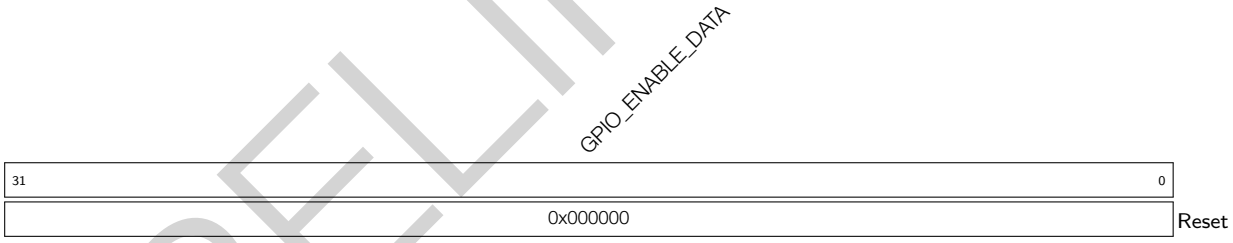
GPIO_OUT1_W1TC GPIO32 ~ 48 output value clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT1_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT1_REG. (WO)

Register 2.8. GPIO_SDIO_SELECT_REG (0x001C)



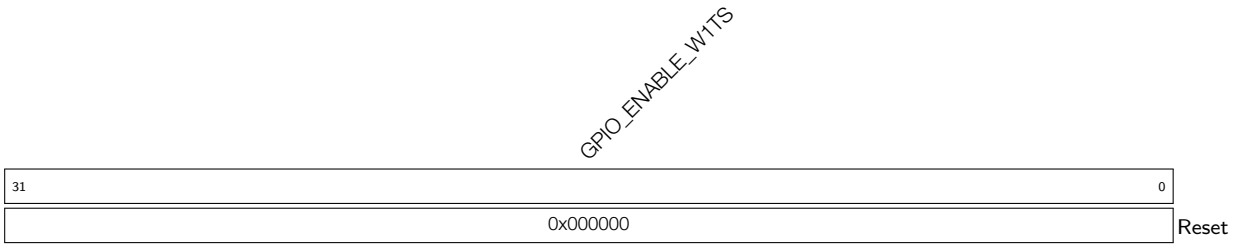
GPIO_SDIO_SEL Reserved (R/W)

Register 2.9. GPIO_ENABLE_REG (0x0020)



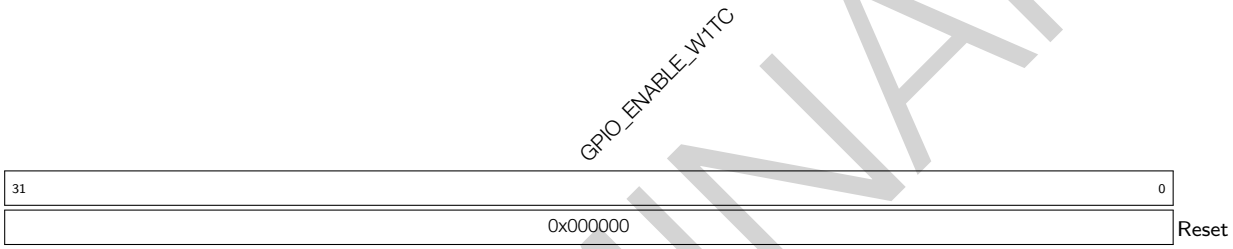
GPIO_ENABLE_DATA GPIO0~31 output enable register. (R/W)

Register 2.10. GPIO_ENABLE_W1TS_REG (0x0024)



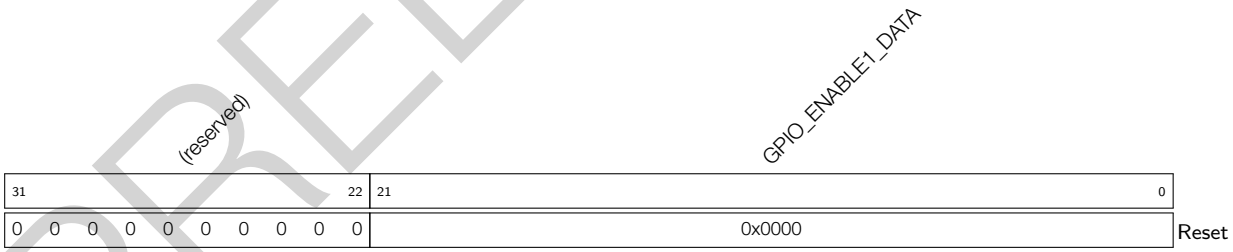
GPIO_ENABLE_W1TS GPIO0 ~ 31 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE_REG. (WO)

Register 2.11. GPIO_ENABLE_W1TC_REG (0x0028)



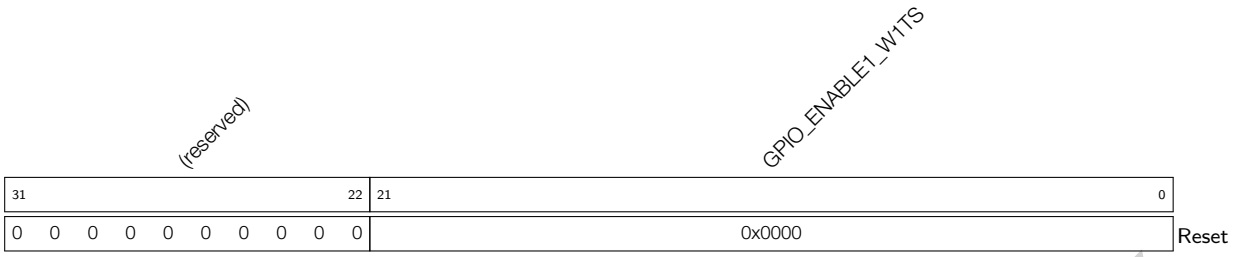
GPIO_ENABLE_W1TC GPIO0 ~ 31 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE_REG. (WO)

Register 2.12. GPIO_ENABLE1_REG (0x002C)



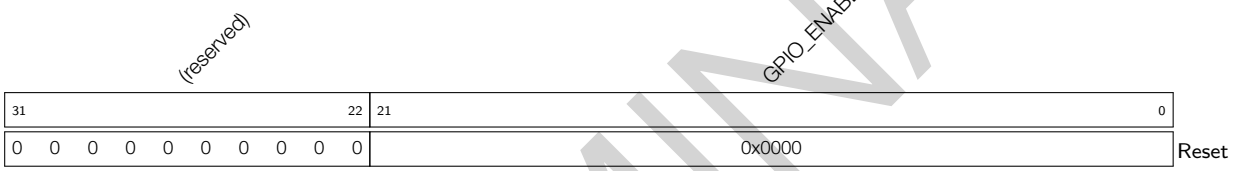
GPIO_ENABLE1_DATA GPIO32 ~ 48 output enable register. (R/W)

Register 2.13. GPIO_ENABLE1_W1TS_REG (0x0030)



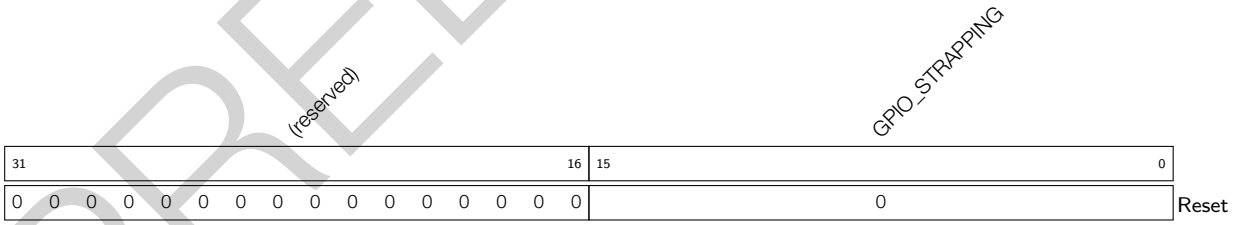
GPIO_ENABLE1_W1TS GPIO32 ~ 48 output enable set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE1_REG. (WO)

Register 2.14. GPIO_ENABLE1_W1TC_REG (0x0034)



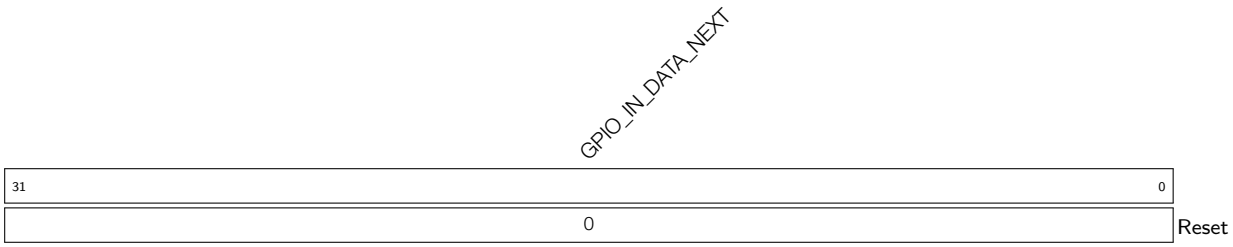
GPIO_ENABLE1_W1TC GPIO32 ~ 48 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE1_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE1_REG. (WO)

Register 2.15. GPIO_STRAP_REG (0x0038)



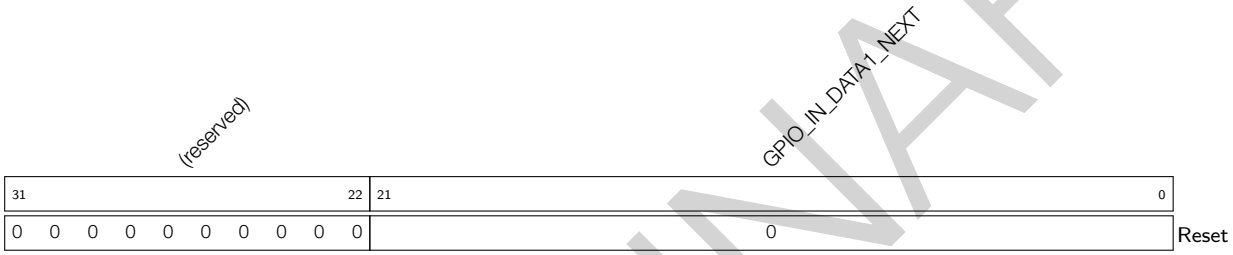
GPIO_STRAPPING GPIO strapping values: bit5 ~ bit2 correspond to stripping pins GPIO3, GPIO45, GPIO0, and GPIO46 respectively. (RO)

Register 2.16. GPIO_IN_REG (0x003C)



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

Register 2.17. GPIO_IN1_REG (0x0040)



GPIO_IN_DATA1_NEXT GPIO32 ~ 48 input value. Each bit represents a pin input value. (RO)

Register 2.18. GPIO_PIN*n*_REG (*n*: 0-48) (0x0074+0x4n*)**

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 Pin 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)

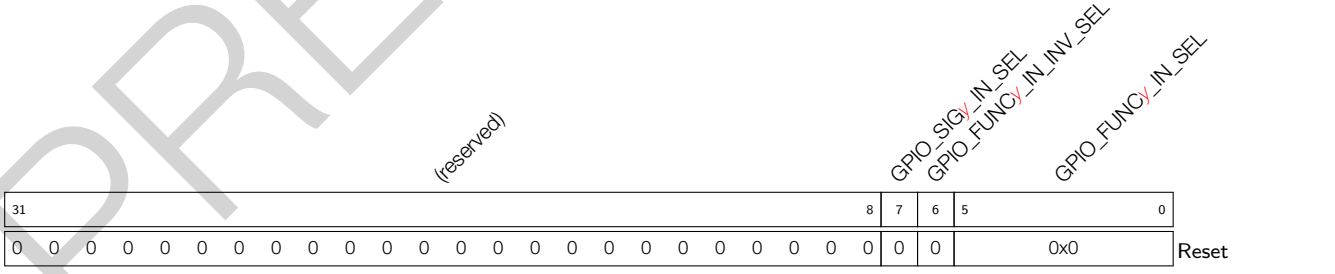
GPIO_PIN*n*_INT_TYPE Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

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 2.19. GPIO_FUNC*y*_IN_SEL_CFG_REG (*y*: 0-255) (0x0154+0x4y*)**

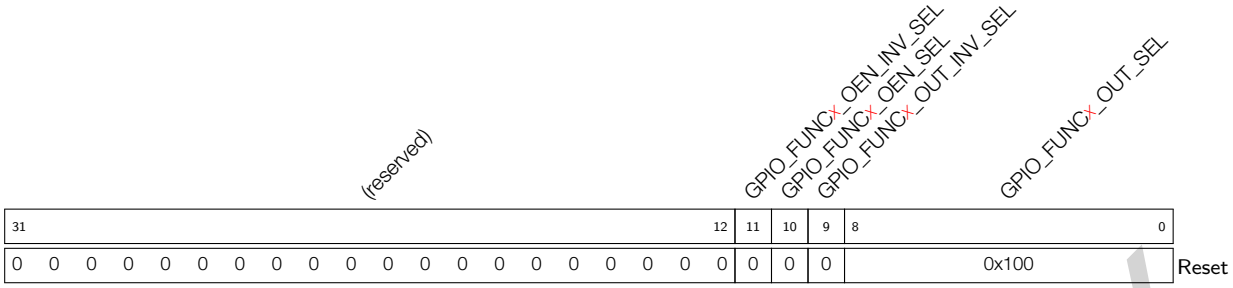


GPIO_FUNC*y*_IN_SEL Selection control for peripheral input signal *Y*, selects a pin from the 48 GPIO matrix pins to connect this input signal. Or selects 0x38 for a constantly high input or 0x3C for a constantly low input. (R/W)

GPIO_FUNC*y*_IN_INV_SEL 1: Invert the input value; 0: Do not invert the input value. (R/W)

GPIO_SIG*y*_IN_SEL Bypass GPIO matrix. 1: route signals via GPIO matrix, 0: connect signals directly to peripheral configured in IO MUX. (R/W)

Register 2.20. GPIO_FUNC*x*_OUT_SEL_CFG_REG (*x*: 0-48) (0x0554+0x4x*)**



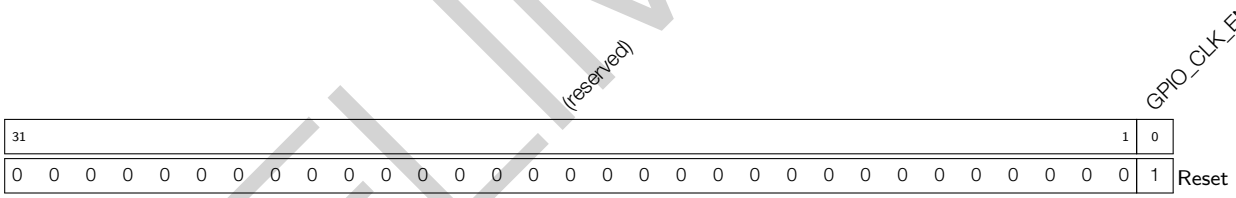
GPIO_FUNC*x*_OUT_SEL Selection control for GPIO output *X*. If a value *Y* (0<=Y<256) is written to this field, the peripheral output signal *Y* will be connected to GPIO output *X*. If a value 256 is written to this field, bit *X* of GPIO_OUT_REG/GPIO_OUT1_REG and GPIO_ENABLE_REG/GPIO_ENABLE1_REG will be selected as the output value and output enable. (R/W)

GPIO_FUNC*x*_OUT_INV_SEL 0: Do not invert the output value; 1: Invert the output value. (R/W)

GPIO_FUNC*x*_OEN_SEL 0: Use output enable signal from peripheral; 1: Force the output enable signal to be sourced from GPIO_ENABLE_REG[*x*]. (R/W)

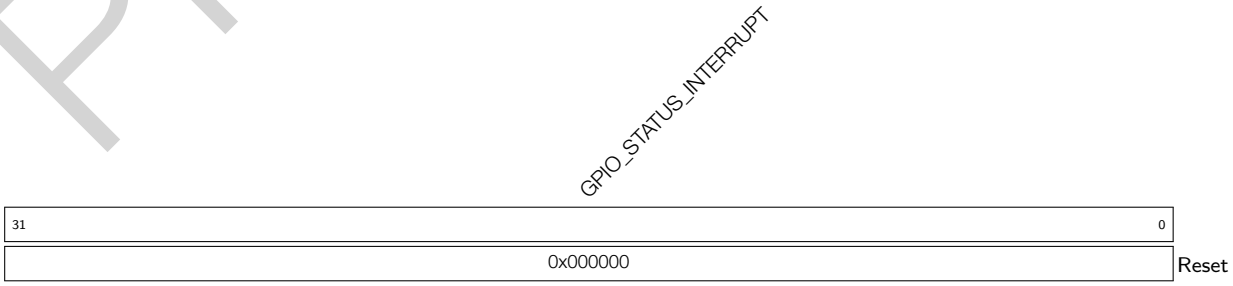
GPIO_FUNC*n*_OEN_INV_SEL 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

Register 2.21. GPIO_CLOCK_GATE_REG (0x062C)



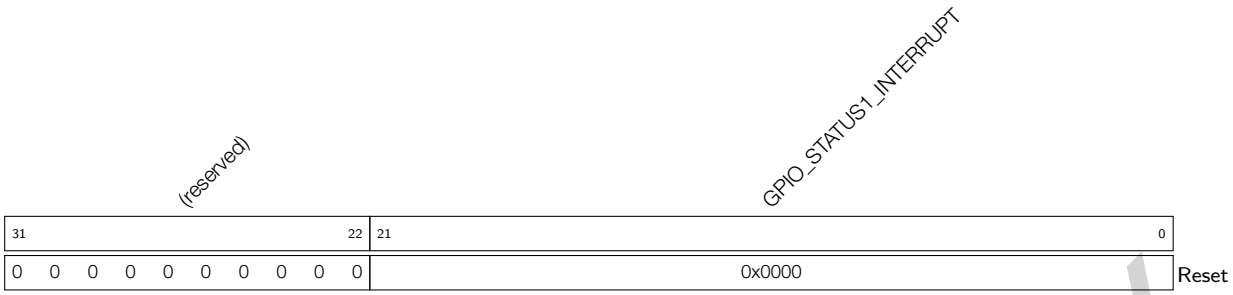
GPIO_CLK_EN Clock gating enable bit. If set to 1, the clock is free running. (R/W)

Register 2.22. GPIO_STATUS_REG (0x0044)



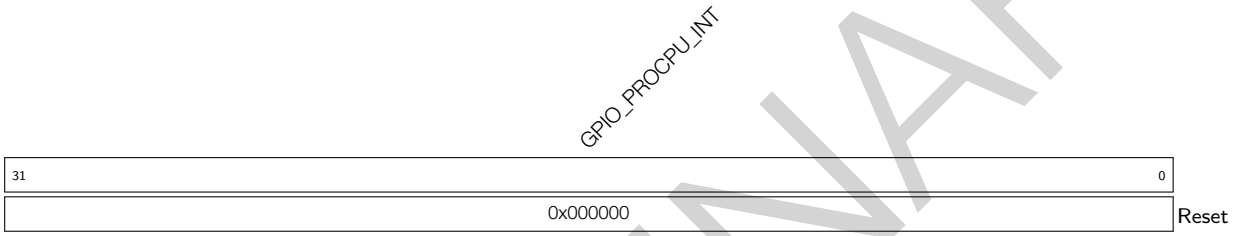
GPIO_STATUS_INTERRUPT GPIO0 ~ 31 interrupt status register. (R/W)

Register 2.23. GPIO_STATUS1_REG (0x0050)



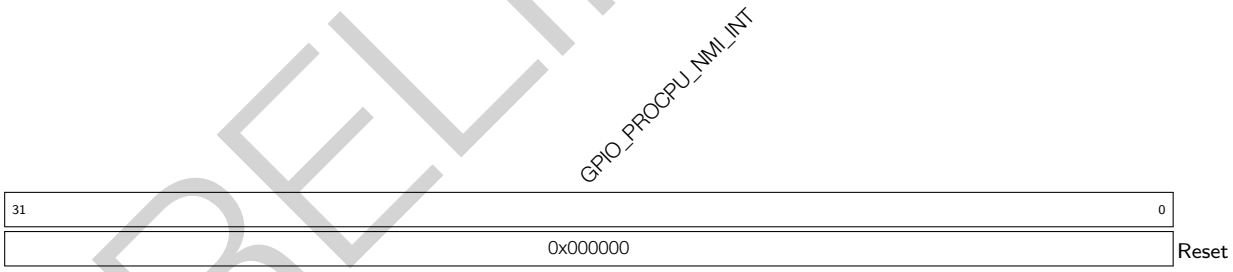
GPIO_STATUS1_INTERRUPT GPIO32 ~ 48 interrupt status register. (R/W)

Register 2.24. GPIO_PCPU_INT_REG (0x005C)



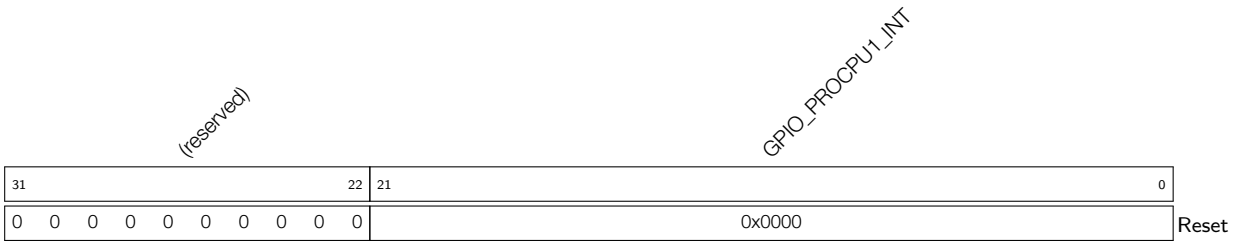
GPIO_PROCPU_INT GPIO0 ~ 31 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 2.25. GPIO_PCPU_NMI_INT_REG (0x0060)



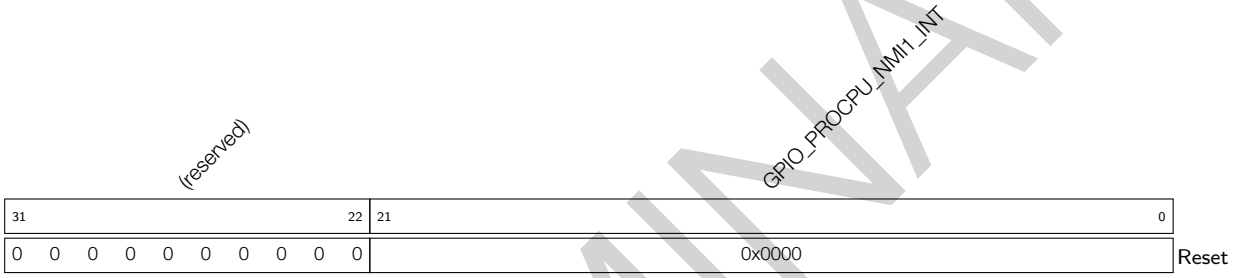
GPIO_PROCPU_NMI_INT GPIO0 ~ 31 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

Register 2.26. GPIO_PCPU_INT1_REG (0x0068)



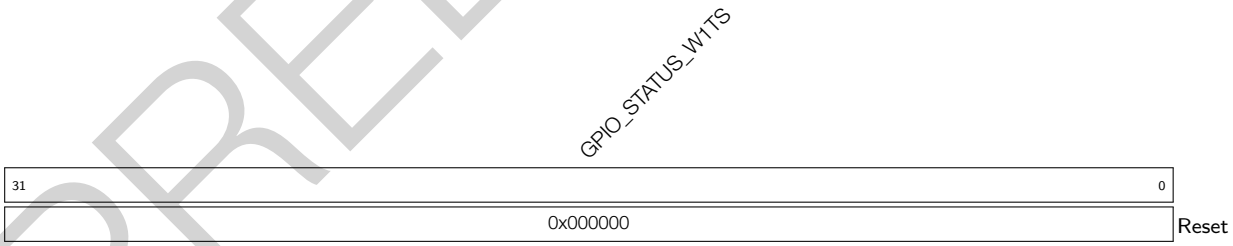
GPIO_PROCPU1_INT GPIO32 ~ 48 PRO_CPU interrupt status. This interrupt status is corresponding to the bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 13 of GPIO_PIN*n*_REG). (RO)

Register 2.27. GPIO_PCPU_NMI_INT1_REG (0x006C)



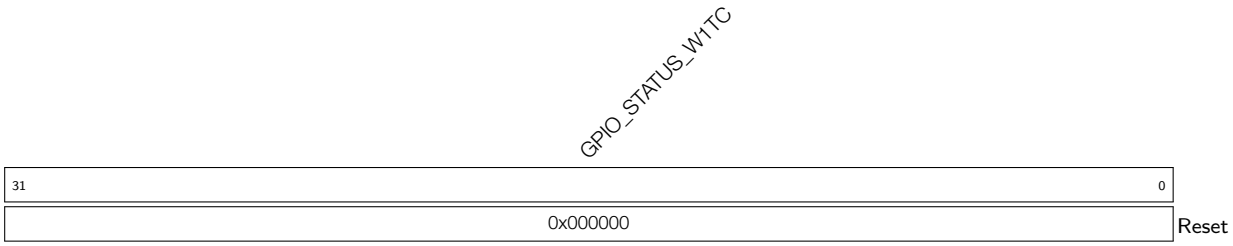
GPIO_PROCPU_NMI1_INT GPIO32 ~ 48 PRO_CPU non-maskable interrupt status. This interrupt status is corresponding to bit in GPIO_STATUS1_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

Register 2.28. GPIO_STATUS_W1TS_REG (0x0048)



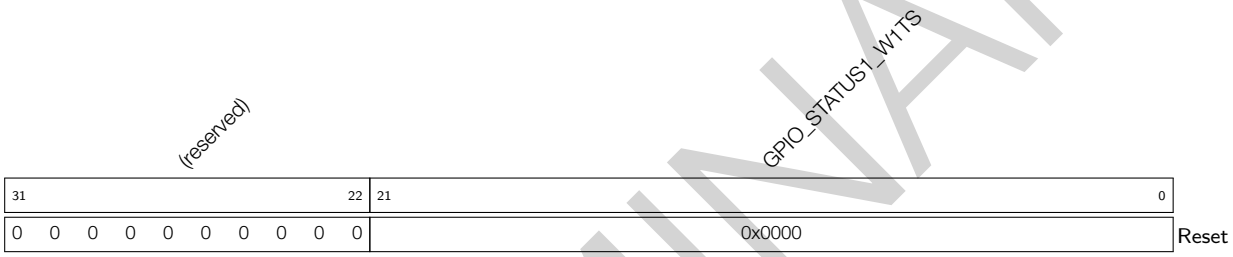
GPIO_STATUS_W1TS GPIO0 ~ 31 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1. Recommended operation: use this register to set GPIO_STATUS_INTERRUPT. (WO)

Register 2.29. GPIO_STATUS_W1TC_REG (0x004C)



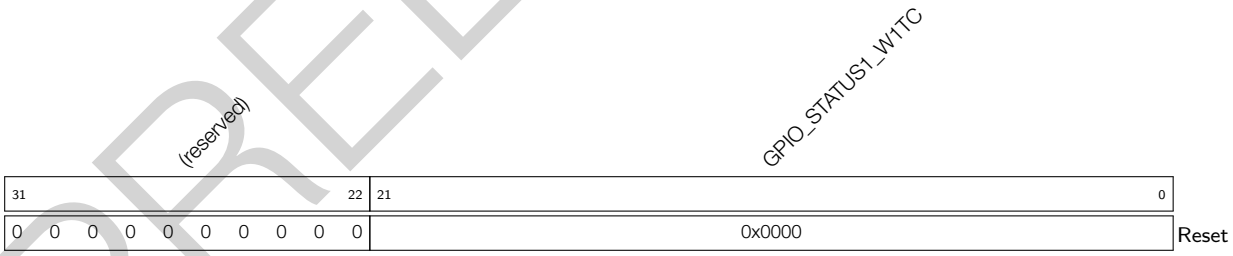
GPIO_STATUS_W1TC GPIO0 ~ 31 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT. (WO)

Register 2.30. GPIO_STATUS1_W1TS_REG (0x0054)



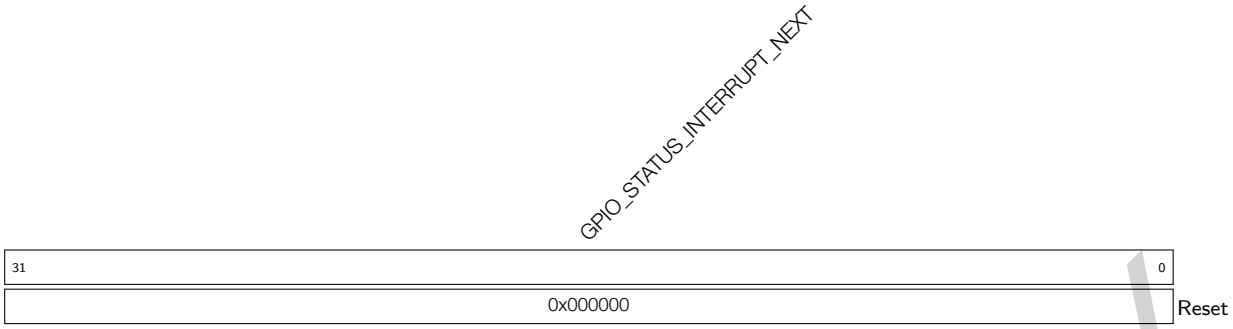
GPIO_STATUS1_W1TS GPIO32 ~ 48 interrupt status set register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be set to 1. Recommended operation: use this register to set GPIO_STATUS1_REG. (WO)

Register 2.31. GPIO_STATUS1_W1TC_REG (0x0058)



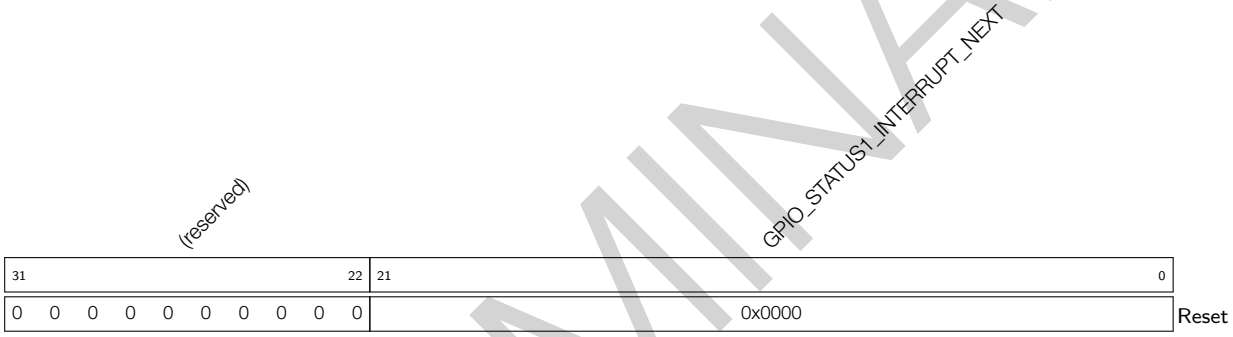
GPIO_STATUS1_W1TC GPIO32 ~ 48 interrupt status clear register. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS1_REG will be cleared. Recommended operation: use this register to clear GPIO_STATUS1_REG. (WO)

Register 2.32. GPIO_STATUS_NEXT_REG (0x014C)



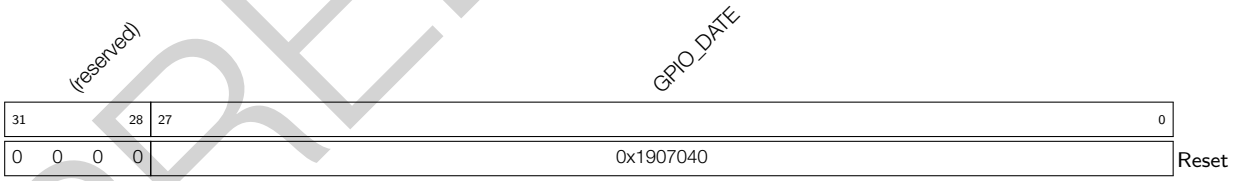
GPIO_STATUS_INTERRUPT_NEXT Interrupt source signal of GPIO0 ~ 31, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. (RO)

Register 2.33. GPIO_STATUS_NEXT1_REG (0x0150)



GPIO_STATUS1_INTERRUPT_NEXT Interrupt source signal of GPIO32 ~ 48. (RO)

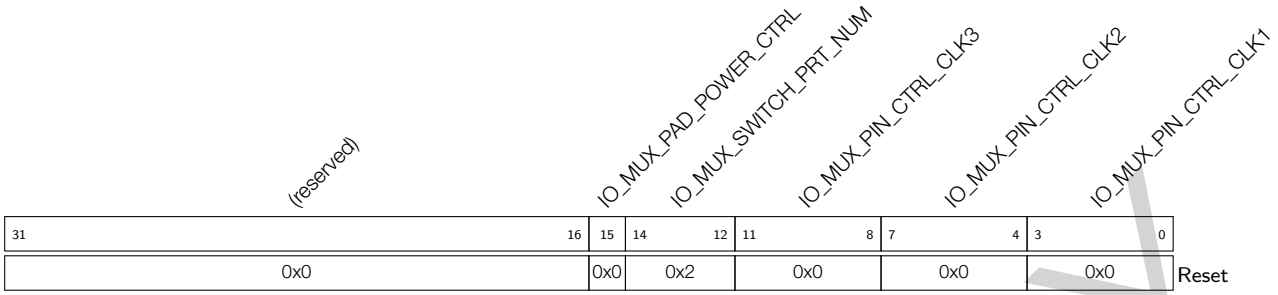
Register 2.34. GPIO_DATE_REG (0x06FC)



GPIO_DATE Version control register (R/W)

2.16.2 IO MUX Registers

Register 2.35. IO_MUX_PIN_CTRL (0x0000)



IO_MUX_PIN_CTRL_CLKx If you want to output clock for I2S0 to:

CLK_OUT1 then set IO_MUX_PIN_CTRL_CLK1 = 0x0

CLK_OUT2 then set IO_MUX_PIN_CTRL_CLK2 = 0x0;

CLK_OUT3 then set IO_MUX_PIN_CTRL_CLK3 = 0x0.

If you want to output clock for I2S1 to:

CLK_OUT1 then set IO_MUX_PIN_CTRL_CLK1 = 0xF

CLK_OUT2 then set IO_MUX_PIN_CTRL_CLK2 = 0xF;

CLK_OUT3 then set IO_MUX_PIN_CTRL_CLK3 = 0xF.

Note:

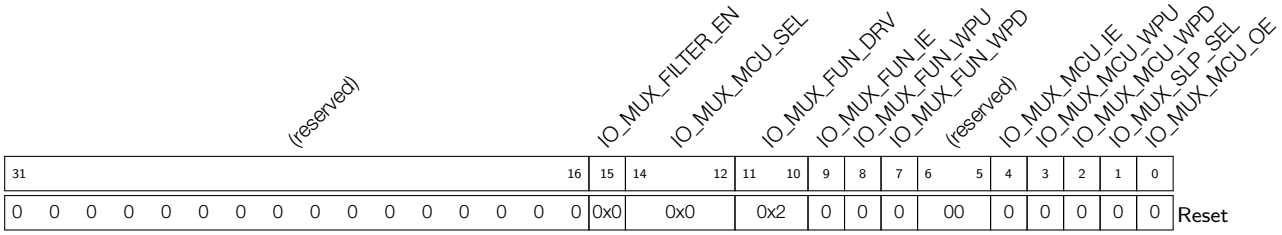
Only the above mentioned combinations of clock source and clock output pins are possible.

The CLK_OUT1 ~ 3 can be found in IO_MUX Pin Function List.

IO_MUX_SWITCH_PRT_NUM GPIO pin 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 2.36. IO_MUX_*n*_REG (*n*: GPIO0-GPIO21, GPIO26-GPIO48) (0x0010+4n*)**



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

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

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

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

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

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

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

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

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

IO_MUX_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)

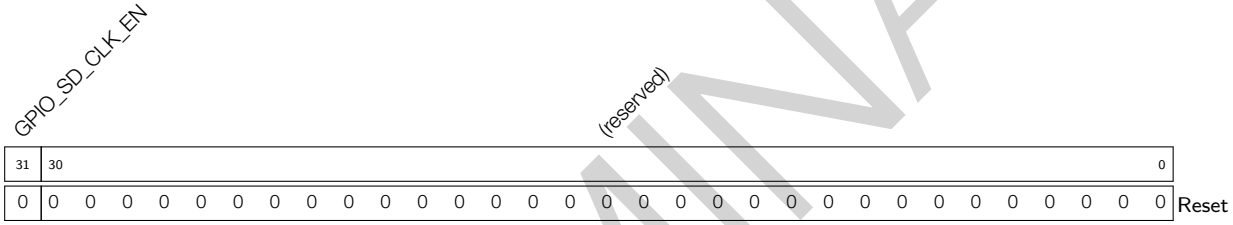
2.16.3 SDM Output Registers

Register 2.37. GPIO_SIGMADELTA*n*_REG (*n*: 0-7) (0x0000+4n*)**

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

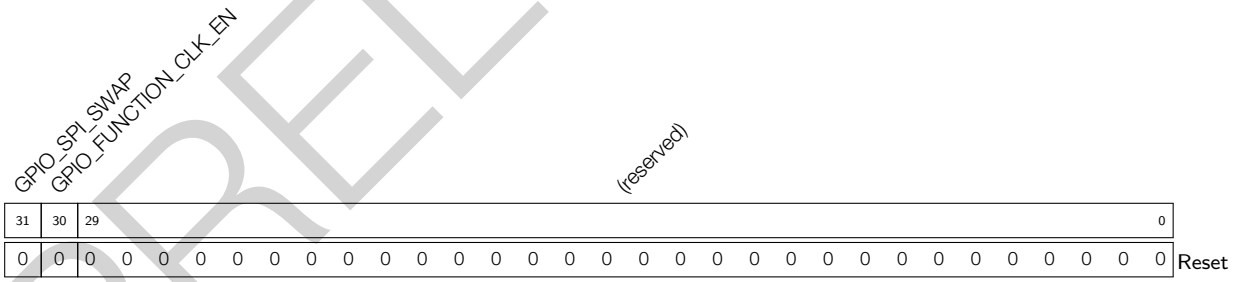
GPIO_SD*n*_PRESCALE This field is used to set a divider value to divide APB clock. (R/W)

Register 2.38. GPIO_SIGMADELTA_CG_REG (0x0020)



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

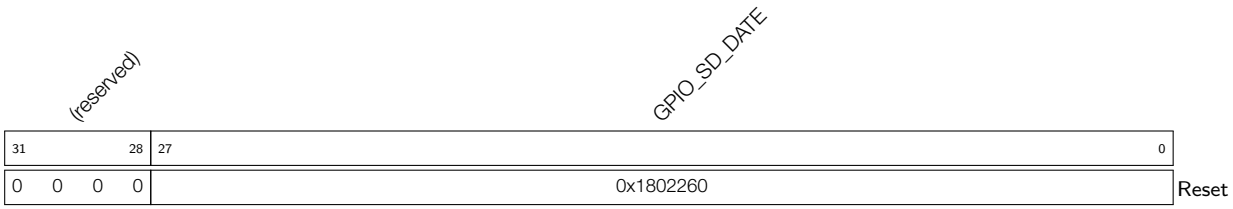
Register 2.39. GPIO_SIGMADELTA_MISC_REG (0x0024)



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

GPIO_SPI_SWAP Reserved. (R/W)

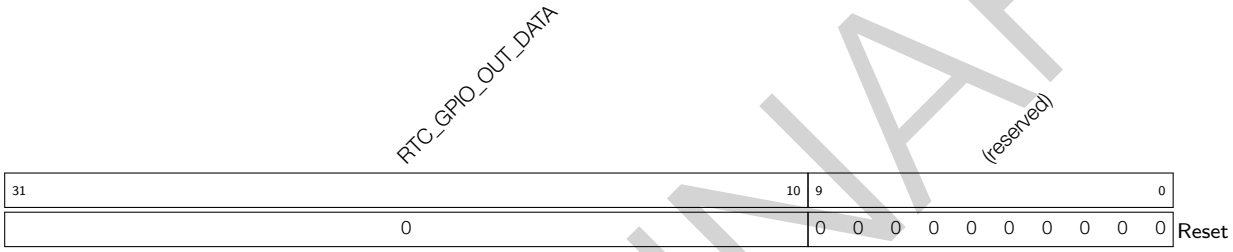
Register 2.40. GPIOSD_SIGMADELTA_VERSION_REG (0x0028)



GPIO_SD_DATE Version control register. (R/W)

2.16.4 RTC IO MUX Registers

Register 2.41. RTC_GPIO_OUT_REG (0x0000)

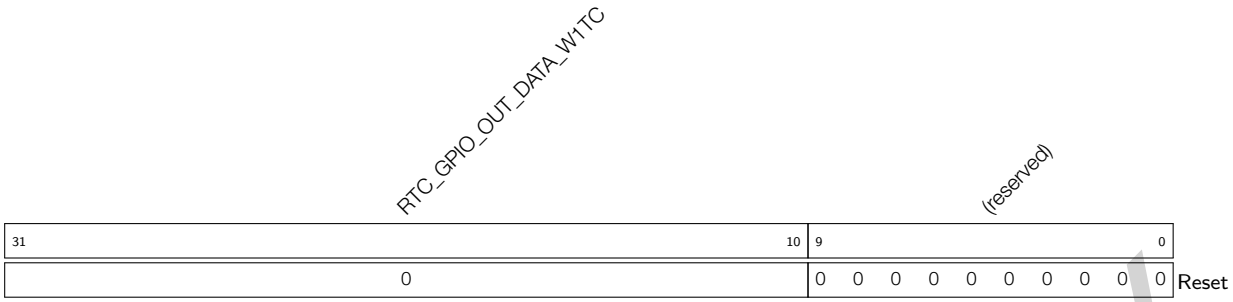


RTC_GPIO_OUT_DATA GPIO0 ~ 21 output register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. (R/W)

Register 2.42. RTC_GPIO_OUT_W1TS_REG (0x0004)

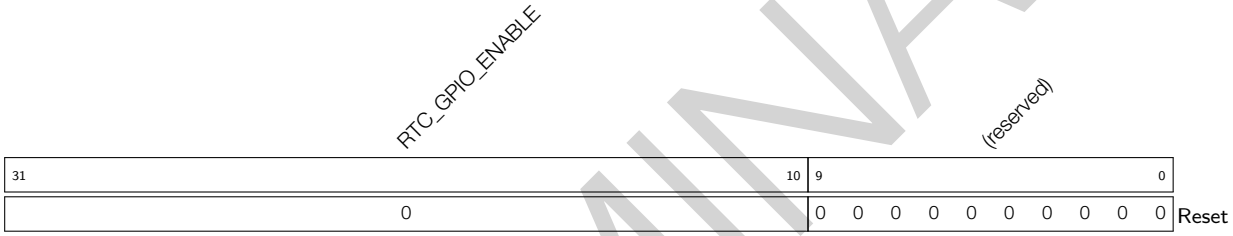
RTC_GPIO_OUT_DATA_W1TS GPIO0 ~ 21 output set register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set RTC_GPIO_OUT_REG. (WO)

Register 2.43. RTC_GPIO_OUT_W1TC_REG (0x0008)



RTC_GPIO_OUT_DATA_W1TC GPIO0 ~ 21 output clear register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear RTC_GPIO_OUT_REG. (WO)

Register 2.44. RTC_GPIO_ENABLE_REG (0x000C)

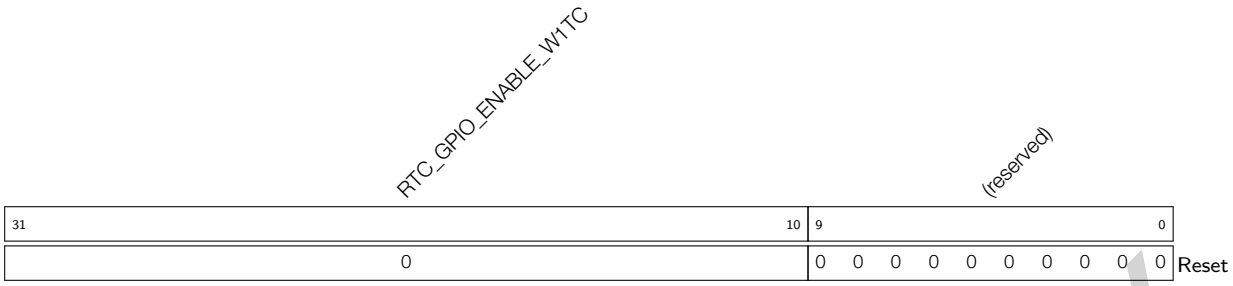


RTC_GPIO_ENABLE GPIO0 ~ 21 output enable. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. If the bit is set to 1, it means this GPIO pin is output. (R/W)

Register 2.45. RTC_GPIO_ENABLE_W1TS_REG (0x0010)

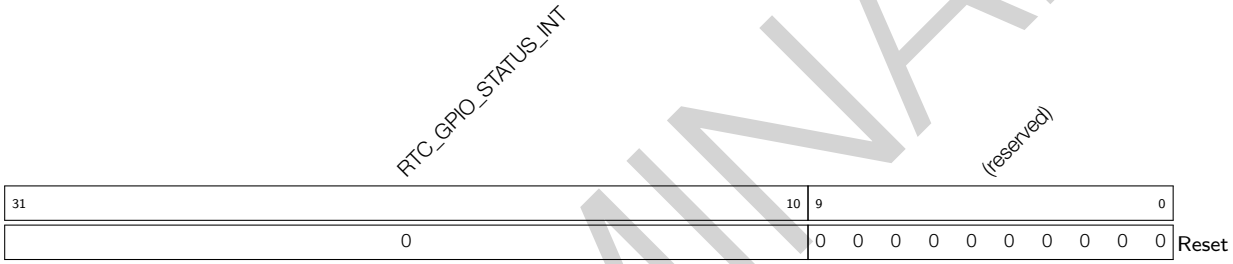
RTC_GPIO_ENABLE_W1TS GPIO0 ~ 21 output enable set register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set RTC_GPIO_ENABLE_REG. (WO)

Register 2.46. RTC_GPIO_ENABLE_W1TC_REG (0x0014)



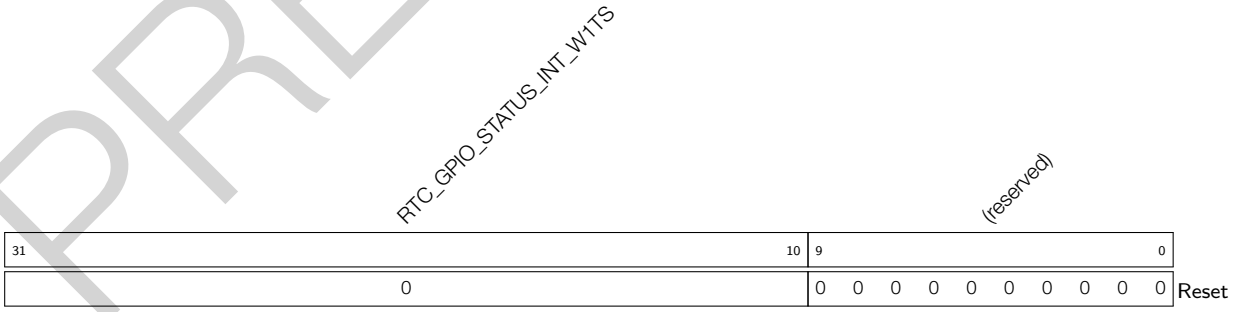
RTC_GPIO_ENABLE_W1TC GPIO0 ~ 21 output enable clear register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear RTC_GPIO_ENABLE_REG. (WO)

Register 2.47. RTC_GPIO_STATUS_REG (0x0018)



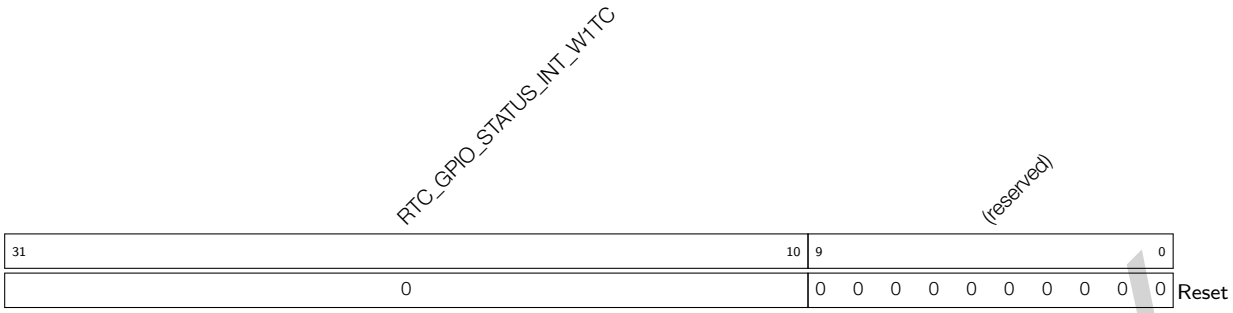
RTC_GPIO_STATUS_INT GPIO0 ~ 21 interrupt status register. Bit10 corresponds to GPIO0, bit11 corresponds to GPIO1, etc. This register should be used together with RTC_GPIO_PIN*n*_INT_TYPE in RTC_GPIO_PIN*n*_REG. 0: no interrupt; 1: corresponding interrupt. (R/W)

Register 2.48. RTC_GPIO_STATUS_W1TS_REG (0x001C)



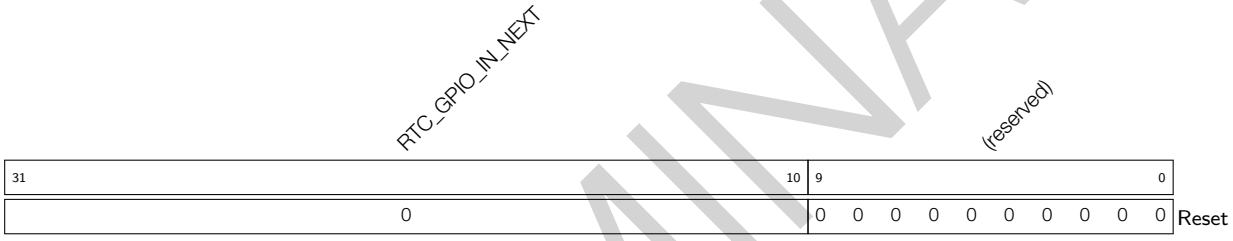
RTC_GPIO_STATUS_INT_W1TS GPIO0 ~ 21 interrupt set register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_STATUS_INT will be set to 1. Recommended operation: use this register to set RTC_GPIO_STATUS_INT. (WO)

Register 2.49. RTC_GPIO_STATUS_W1TC_REG (0x0020)



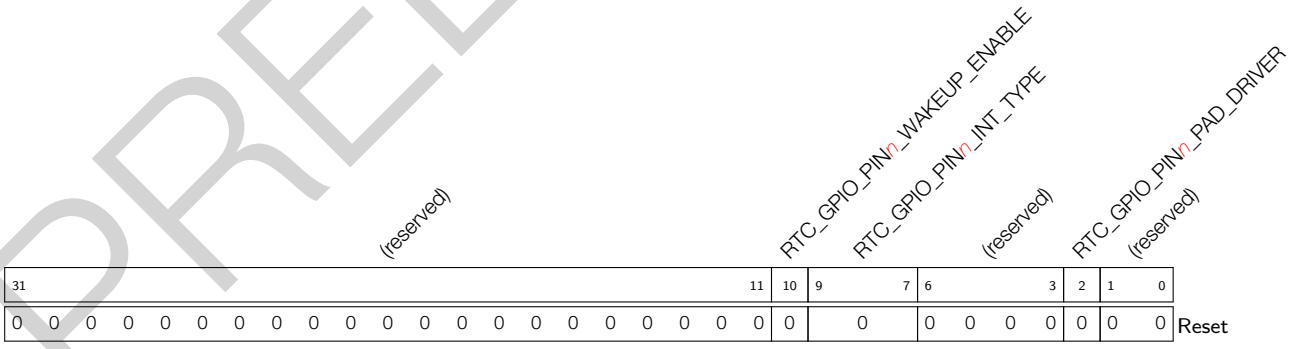
RTC_GPIO_STATUS_INT_W1TC GPIO0 ~ 21 interrupt clear register. If the value 1 is written to a bit here, the corresponding bit in RTC_GPIO_STATUS_INT will be cleared. Recommended operation: use this register to clear RTC_GPIO_STATUS_INT. (WO)

Register 2.50. RTC_GPIO_IN_REG (0x0024)



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

Register 2.51. RTC_GPIO_PIN*n*_REG (*n*: 0-21) (0x0028+0x4n*)**

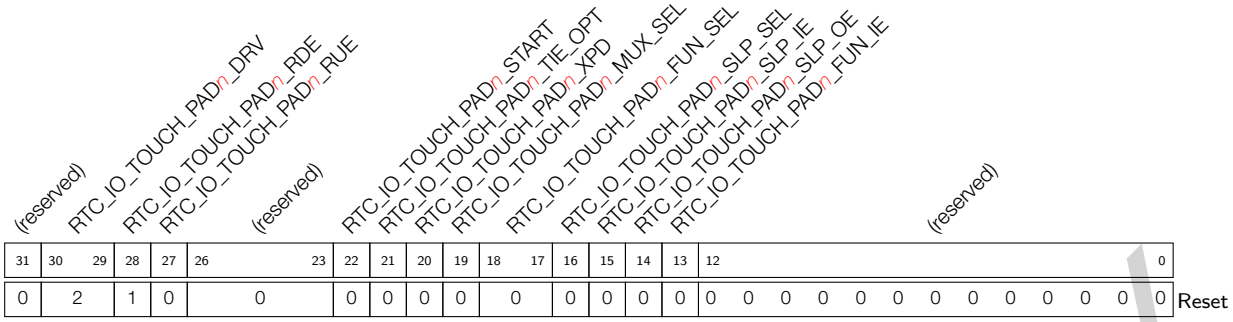


RTC_GPIO_PIN*n*_PAD_DRIVER Pin driver selection. 0: normal output; 1: open drain. (R/W)

RTC_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)

RTC_GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable. This will only wake up the chip from Light-sleep. (R/W)

Register 2.52. RTC_IO_TOUCH_PADn_REG (n: 0-14) (0x0084+0x4*n)



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

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

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

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

RTC_IO_TOUCH_PADn_FUN_SEL Function selection. (R/W)

RTC_IO_TOUCH_PADn_MUX_SEL Connect the RTC pin input or digital pin input. 0 is available, i.e. select digital pin input. (R/W)

RTC_IO_TOUCH_PADn_XPD Touch sensor power on. (R/W)

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

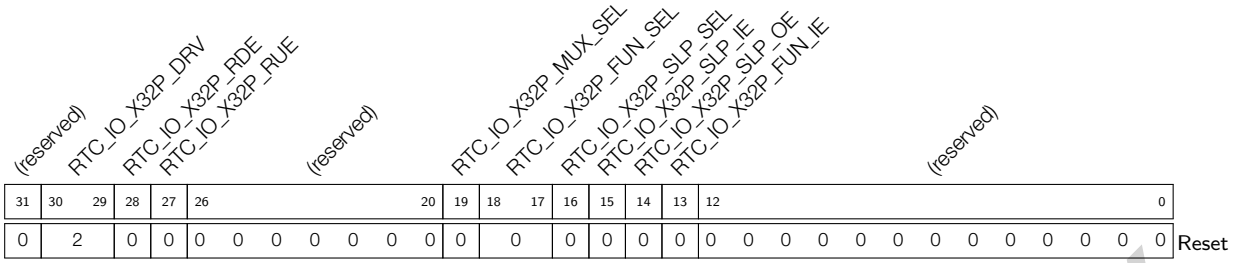
RTC_IO_TOUCH_PADn_START Start touch sensor. (R/W)

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

RTC_IO_TOUCH_PADn_RDE Pull-down enable of the pin. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

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

Register 2.53. RTC_IO_XTAL_32P_PAD_REG (0x00C0)



RTC_IO_X32P_FUN_IE Input enable in normal execution. (R/W)

RTC_IO_X32P_SLP_OE Output enable in sleep mode. (R/W)

RTC_IO_X32P_SLP_IE Input enable in sleep mode. (R/W)

RTC_IO_X32P_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTC_IO_X32P_FUN_SEL Function selection. (R/W)

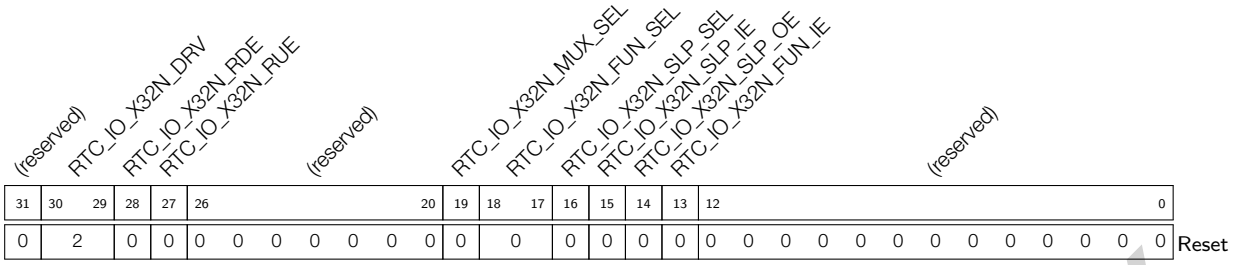
RTC_IO_X32P_MUX_SEL 1: use RTC GPIO; 0: use digital GPIO. (R/W)

RTC_IO_X32P_RUE Pull-up enable of the pin. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTC_IO_X32P_RDE Pull-down enable of the pin. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

RTC_IO_X32P_DRV Select the drive strength of the pin. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 2.54. RTC_IO_XTAL_32N_PAD_REG (0x00C4)



RTC_IO_X32N_FUN_IE Input enable in normal execution. (R/W)

RTC_IO_X32N_SLP_OE Output enable in sleep mode. (R/W)

RTC_IO_X32N_SLP_IE Input enable in sleep mode. (R/W)

RTC_IO_X32N_SLP_SEL 1: enable sleep mode; 0: no sleep mode. (R/W)

RTC_IO_X32N_FUN_SEL Function selection. (R/W)

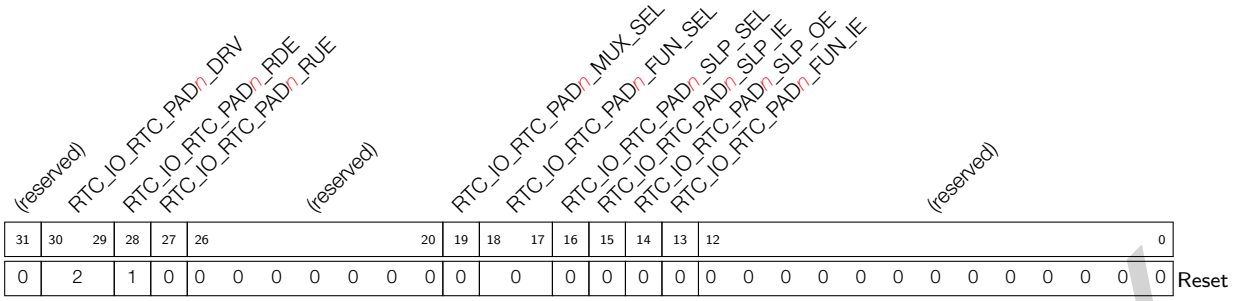
RTC_IO_X32N_MUX_SEL 1: use RTC GPIO; 0: use digital GPIO. (R/W)

RTC_IO_X32N_RUE Pull-up enable of the pin. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

RTC_IO_X32N_RDE Pull-down enable of the pin. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

RTC_IO_X32N_DRV Select the drive strength of the pin. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 2.55. RTC_IO_RTC_PADn_REG (n: 17-21) (0x00C8, 0x00CC, 0x00D0, 0x00D4, 0x00D8)



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

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

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

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

RTC_IO_RTC_PADn_FUN_SEL Function selection. (R/W)

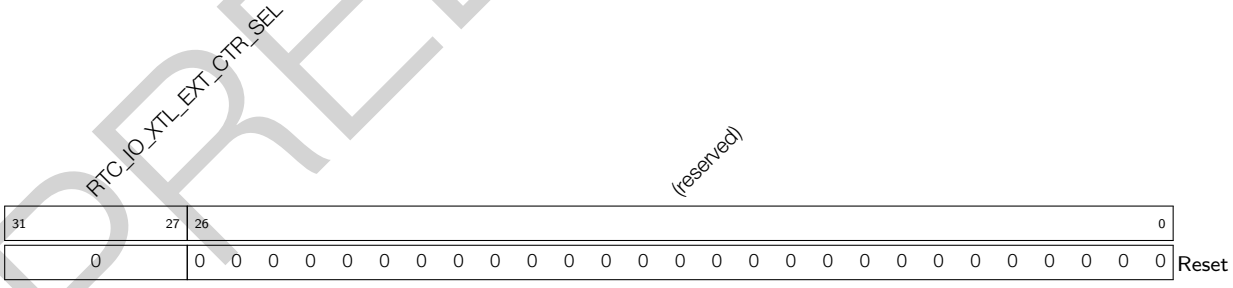
RTC_IO_RTC_PADn_MUX_SEL 1: use RTC GPIO; 0: use digital GPIO. (R/W)

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

RTC_IO_RTC_PADn_RDE Pull-down enable of the pin. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

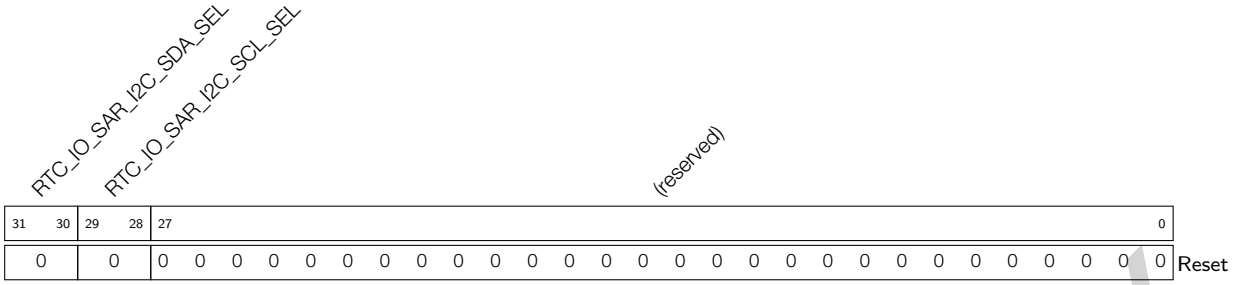
RTC_IO_RTC_PADn_DRV Select the drive strength of the pin. 0: ~5 mA: 1: ~10 mA: 2: ~20 mA; 3: ~40 mA. (R/W)

Register 2.56. RTC_IO_XTL_EXT_CTR_REG (0x00E0)



RTC_IO_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)

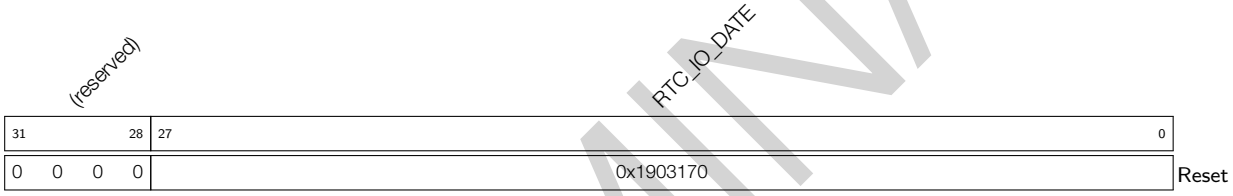
Register 2.57. RTC_IO_SAR_I2C_IO_REG (0x00E4)



RTC_IO_SAR_I2C_SCL_SEL Selects a pin the RTC I2C SCL signal connects to. 0: use RTC GPIO0; 1: use RTC GPIO2. (R/W)

RTC_IO_SAR_I2C_SDA_SEL Selects a pin the RTC I2C SDA signal connects to. 0: use RTC GPIO1; 1: use RTC GPIO3. (R/W)

Register 2.58. RTC_IO_DATE_REG (0x01FC)



RTC_IO_DATE Version control register (R/W)

3 Reset and Clock

3.1 Reset

3.1.1 Overview

ESP32-S3 provides four reset levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset.

All reset levels mentioned above (except Chip Reset) maintain the data stored in internal memory. Figure 3-1 shows the affected subsystems of the four reset levels.

3.1.2 Architectural Overview

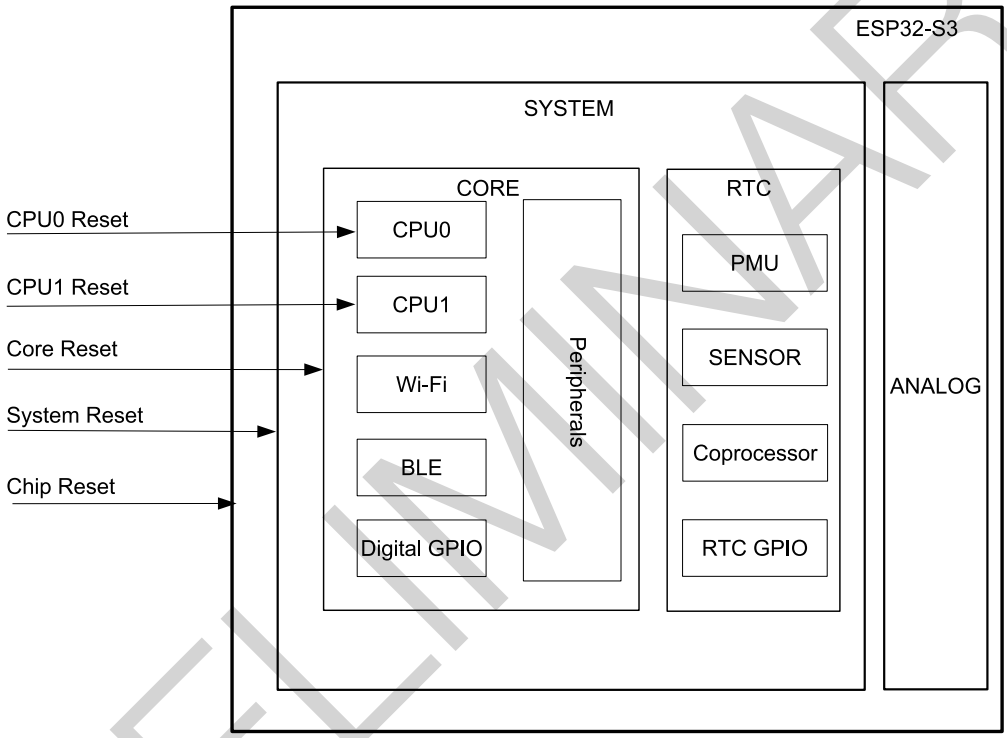


Figure 3-1. Reset Levels

3.1.3 Features

- Support four reset levels:
  - CPU Reset: only resets CPU*x* core. CPU*x* can be CPU0 or CPU1 here. Once such reset is released, programs will be executed from CPU*x* reset vector. Each CPU core has its own reset logic.
  - Core Reset: resets the whole digital system except RTC, including CPU0, CPU1, peripherals, Wi-Fi, Bluetooth® LE (BLE), and digital GPIOs.
  - System Reset: resets the whole digital system, including RTC.
  - Chip Reset: resets the whole chip.
- Support software reset and hardware reset:
  - Software reset is triggered by CPU*x* configuring its corresponding registers.

– Hardware reset is directly triggered by the circuit.

Note:
If CPU Reset is from CPU0, the sensitive registers will be reset, too.

3.1.4 Functional Description

CPU0 and CPU1 will be reset immediately when any of the reset above occurs. After the reset is released, CPU0 and CPU1 can read from the registers RTC_CNTL_RESET_CAUSE_PROCPU and RTC_CNTL_RESET_CAUSE_APPCPU to get the reset source, respectively. The reset sources recorded in the two registers are shared by the two CPUs, except the CPU reset sources, i.e. each CPU has its own CPU reset sources.

Table 3-1 lists the reset sources and the types of reset they trigger.

Table 3-1. Reset Sources

Code	Source	Reset Type	Comments
0x01	Chip reset[1]	Chip Reset	-
0x0F	Brown-out system reset	Chip Reset or System Reset	Triggered by brown-out detector[2]
0x10	RWDT system reset	System Reset	See Chapter 7 *Watchdog Timers*
0x12	Super Watchdog reset	System Reset	See Chapter 7 *Watchdog Timers*
0x13	GLITCH reset	System Reset	See Chapter 18 *Clock Glitch Detection [to be added later]*
0x03	Software system reset	Core Reset	Triggered by configuring RTC_CNTL_SW_SYS_RST
0x05	Deep-sleep reset	Core Reset	See Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*
0x07	MWDT0 core reset	Core Reset	See Chapter 7 *Watchdog Timers*
0x08	MWDT1 core reset	Core Reset	See Chapter 7 *Watchdog Timers*
0x09	RWDT core reset	Core Reset	See Chapter 7 *Watchdog Timers*
0x14	eFuse reset	Core Reset	Triggered by eFuse CRC error
0x0B	MWDT0 CPU*x* reset	CPU Reset	See Chapter 7 *Watchdog Timers*
0x0C	Software CPU*x* reset	CPU Reset	Triggered by configuring RTC_CNTL_SW_PRO(APP)CPU_RST
0x0D	RWDT CPU*x* reset	CPU Reset	See Chapter 7 *Watchdog Timers*
0x11	MWDT1 CPU*x* reset	CPU Reset	See Chapter 7 *Watchdog Timers*

[1] 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).

[2] 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 15 *Low-Power Management (RTC_CNTL) [to be added later]*.

3.2 Clock

3.2.1 Overview

ESP32-S3 clocks are mainly sourced from oscillator (OSC), RC, and PLL circuit, and then processed by the dividers/selectors, which allows most functional modules to select their working clock according to their power consumption and performance requirements. Figure 3-2 shows the system clock structure.

3.2.2 Architectural Overview

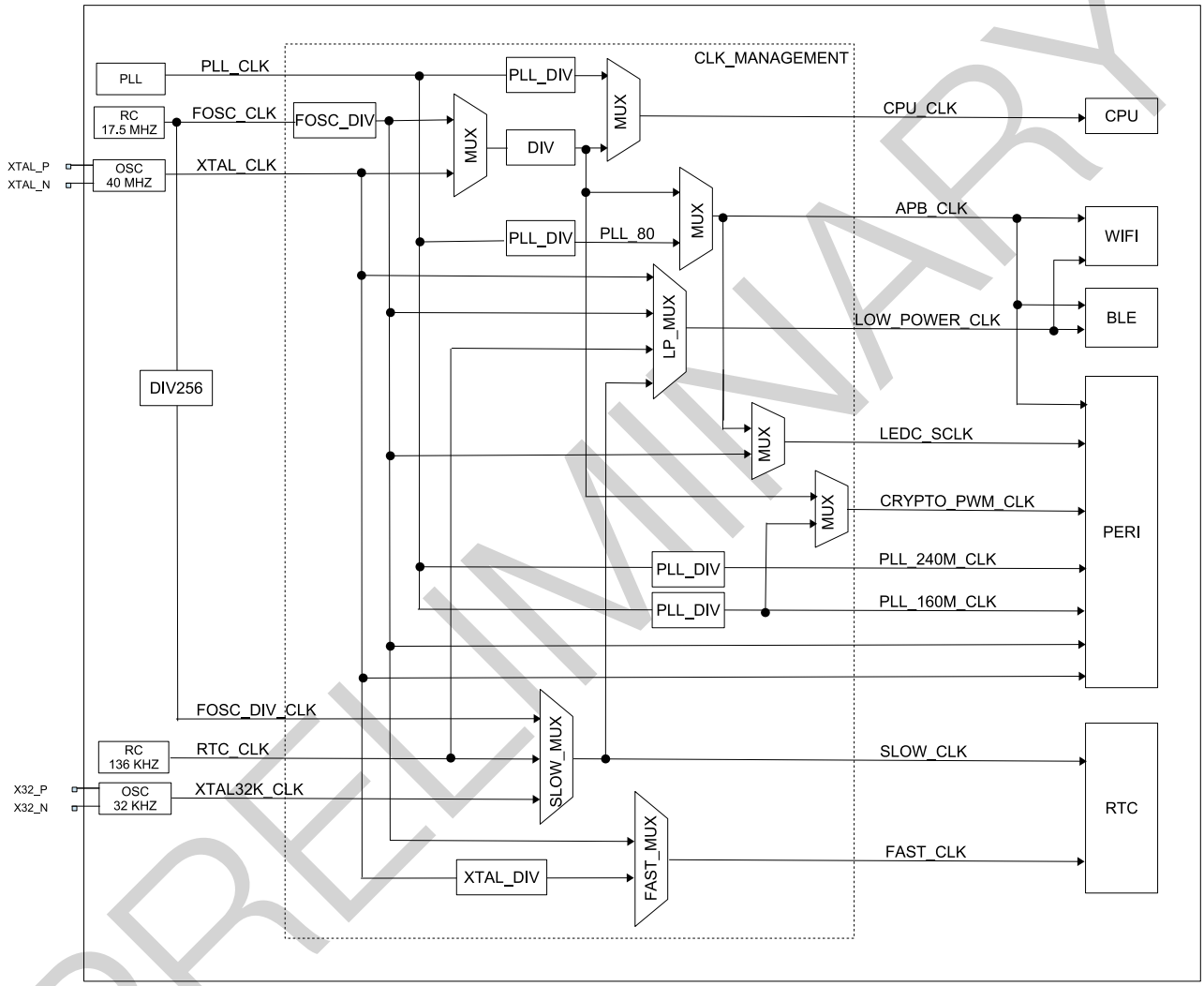


Figure 3-2. Clock Structure

3.2.3 Features

ESP32-S3 clocks can be classified in two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_CLK (320 MHz or 480 MHz): internal PLL clock
  - XTAL_CLK (40 MHz): external crystal clock
- Slow speed clocks for low-power devices, such as RTC module and low-power peripherals
  - XTAL32K_CLK (32 kHz): external crystal clock

- FOSC_CLK (17.5 MHz by default): internal fast RC oscillator clock with adjustable frequency
- FOSC_DIV_CLK: internal fast RC oscillator clock derived from FOSC_CLK divided by 256
- RTC_CLK (136 kHz by default): internal low RC oscillator clock with adjustable frequency

3.2.4 Functional Description

3.2.4.1 CPU Clock

As Figure 3-2 shows, CPU_CLK is the master clock for CPU*x* and it can be as high as 240 MHz when CPU*x* works in high performance mode. Alternatively, CPU*x* can run at lower frequencies, such as at 2 MHz, to lower power consumption.

Users can set PLL_CLK, FOSC_CLK or XTAL_CLK as CPU_CLK clock source by configuring register SYSTEM_SOC_CLK_SEL, see Table 3-2 and Table 3-3. By default, the CPU clock is sourced from XTAL_CLK with a divider of 2, i.e. the CPU clock is 20 MHz.

Table 3-2. CPU Clock Source

SYSTEM_SOC_CLK_SEL Value	CPU Clock Source
0	XTAL_CLK
1	PLL_CLK
2	FOSC_CLK

Table 3-3. CPU Clock Frequency

CPU 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
FOSC_CLK	2	-	-	CPU_CLK = FOSC_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1

* The value of register SYSTEM_SOC_CLK_SEL.
* The value of register SYSTEM_PLL_FREQ_SEL.
* The value of register SYSTEM_CPUPERIOD_SEL.

3.2.4.2 Peripheral Clocks

Peripheral clocks include APB_CLK, CRYPTO_PWM_CLK, PLL_160M_CLK, PLL_240M_CLK, LEDC_CLK, XTAL_CLK, and FOSC_CLK. Table 3-4 shows which clock can be used by each peripheral.

Table 3-4. Peripheral Clocks

Peripheral	XTAL_CLK	APB_CLK	PLL_160M_CLK	PLL_240M_CLK	FOSC_CLK	CRYPTO_PWM_CLK	LEDC_CLK
TIMG	Y	Y					
I2S	Y		Y	Y			
UHCI		Y					
UART	Y	Y			Y		
RMT	Y	Y			Y		
PWM						Y	
I2C	Y				Y		
SPI	Y	Y					
PCNT		Y					
eFuse Controller		Y					
SARADC		Y		Y			
USB		Y					
CRYPTO						Y	
TWAI Controller		Y					
SDIO HOST	Y		Y				
LEDC	Y	Y			Y		Y
LCD_CAM	Y		Y	Y			
SYS_TIMER	Y	Y					

APB_CLK

APB_CLK frequency is determined by the clock source of CPU_CLK as shown in Table 3-5.

Table 3-5. APB_CLK Fequency

CPU_CLK Source	APB_CLK Frequency
PLL_CLK	80 MHz
XTAL_CLK	CPU_CLK
FOSC_CLK	CPU_CLK

CRYPTO_PWM_CLK

The frequency of CRYPTO_PWM_CLK is determined by the CPU_CLK source, as shown in Table 3-6.

Table 3-6. CRYPTO_PWM_CLK Frequency

CPU_CLK Source	CRYPTO_PWM_CLK Frequency
PLL_CLK	160 MHz
XTAL_CLK	CPU_CLK
FOSC_CLK	CPU_CLK

PLL_160M_CLK

PLL_160M_CLK is divided from PLL_CLK according to current PLL frequency.

PLL_240M_CLK

PLL_240M_CLK is divided from PLL_CLK according to current PLL frequency.

LEDC_CLK

LEDC module uses FOSC_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 LEDC can work normally via FOSC_CLK.

3.2.4.3 Wi-Fi and Bluetooth LE Clock

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

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

3.2.4.4 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 is derived from RTC_CLK, XTAL32K_CLK or FOSC_DIV_CLK and 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 FOSC_CLK.

4 Chip Boot Control

4.1 Overview

ESP32-S3 has four strapping pins:

- GPIO0
- GPIO3
- GPIO45
- GPIO46

These strapping pins are used to control the following functions during chip power-on or hardware reset:

- control chip boot mode
- enable or disable ROM code printing to UART
- control the voltage of VDD_SPI
- control the source of JTAG signals

During system reset triggered by power-on, brown-out or by analog super watchdog (see Chapter 3 *Reset and Clock*), hardware captures samples and stores the voltage level of strapping pins as strapping bit of "0" or "1" in latches, and holds these bits until the chip is powered down or shut down. Software can read the latch status (strapping value) from the register GPIO_STRAPPING.

By default, GPIO0, GPIO45, and GPIO46 are connected to the chip's internal pull-up/pull-down resistors. If these pins are not connected or connected to an external high-impedance circuit, the internal weak pull-up/pull-down determines the default input level of these strapping pins (see Table 4-1).

Table 4-1. Default Configuration of Strapping Pins

Strapping Pin	Default Configuration
GPIO0	Pull-up
GPIO3	N/A
GPIO45	Pull-down
GPIO46	Pull-down

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP32-S3. After the reset is released, the strapping pins work as normal-function pins.

4.2 Boot Mode Control

GPIO0 and GPIO46 control the boot mode after the reset is released.

Table 4-2. Boot Mode Control

Boot Mode	GPIO0	GPIO46
SPI Boot	1	x
Download Boot	0	0

Table 4-2 shows the strapping pin values of GPIO0 and GPIO46, and the associated boot modes. "x" means that this value is ignored. The ESP32-S3 chip only supports the two boot modes listed above. 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. SPI Boot mode can be further classified as follows:

- Normal Flash Boot: supports Security Boot and programs run in RAM.
- Direct Boot: does not support Security Boot and programs run directly in flash. To enable this mode, make sure that the first two words of the bin file downloading to flash (address: 0x42000000) are 0xaebd041d.

In Download Boot mode, users can download code to flash using UART0 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 behaviors:

- EFUSE_DIS_FORCE_DOWNLOAD

  If this eFuse is 0 (default), software can force 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, RTC_CNTL_FORCE_DOWNLOAD_BOOT 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. Ignore this eFuse if Download Boot mode is disabled.

USB Serial/JTAG Controller can also force the chip into Download Boot mode from SPI Boot mode, as well as force the chip into SPI Boot mode from Download Boot mode. For detailed information, please refer to Chapter 20 *USB Serial/JTAG Controller (USB_SERIAL_JTAG) [to be added later]*.

4.3 ROM Code Printing Control

During the early boot process,

- if EFUSE_DIS_USB_DEVICE and EFUSE_DIS_USB are cleared, ROM code is always printed to USB Serial/JTAG controller.
- Otherwise, GPIO46 controls ROM code printing, together with EFUSE_UART_PRINT_CONTROL. See Table 4-3.

Table 4-3. ROM Code Printing Control

eFuse[1]	GPIO46	ROM Code Printing
0	x	ROM code is always printed to UART during boot. The value of GPIO46 is ignored.
1	0	Print is enabled during boot.
	1	Print is disabled during boot.
2	0	Print is disabled during boot.
	1	Print is enabled during boot.
3	x	Print is always disabled during boot. The value of GPIO46 is ignored.

[1] eFuse: EFUSE_UART_PRINT_CONTROL

If ROM code is printed to UART, U0TXD is used as the default pin. To print the ROM code to pin U1TXD, configure EFUSE_UART_PRINT_CHANNEL:

- 0: print to pin U0TXD
- 1: print to pin U1TXD

4.4 VDD_SPI Voltage Control

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

- GPIO45 = 0, VDD_SPI pin is powered directly from VDD3P3_RTC via resistor R_{SPI}. Typically this voltage is 3.3 V. For more information, see Figure 4: ESP32-S3 Power Scheme in ESP32-S3 Datasheet.
- GPIO45 = 1, VDD_SPI pin is powered from internal 1.8 V LDO.

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

- EFUSE_VDD_SPI_TIEH = 0, VDD_SPI connects to 1.8 V LDO.
- EFUSE_VDD_SPI_TIEH = 1, VDD_SPI connects to VDD3P3_RTC.

4.5 JTAG Signal Source Control

GPIO3 controls the source of JTAG signals during the early boot process. This GPIO is used together with EFUSE_DIS_PAD_JTAG, EFUSE_DIS_USB_JTAG, and EFUSE_STRAP_JTAG_SEL, see Table 4-4.

Table 4-4. JTAG Signal Source Control

eFuse 1[a]	eFuse 2[b]	eFuse 3[c]	GPIO3	Signal Source
0	0	0	x	JTAG signals come from USB Serial/JTAG Controller. The value of GPIO3 is ignored.
		1	0	JTAG signals come from corresponding pins[d]
			1	JTAG signals come from USB Serial/JTAG Controller.
0	1	x	x	JTAG signals come from corresponding pins[d] The values of EFUSE_STRAP_JTAG_SEL and GPIO3 are ignored.
1	0	x	x	JTAG signals come from USB Serial/JTAG Controller. The values of EFUSE_STRAP_JTAG_SEL and GPIO3 are ignored.
1	1	x	x	JTAG is disabled. The values of EFUSE_STRAP_JTAG_SEL and GPIO3 are ignored.

[a] eFuse 1: EFUSE_DIS_PAD_JTAG

[b] eFuse 2: EFUSE_DIS_USB_JTAG

[c] eFuse 3: EFUSE_STRAP_JTAG_SEL

[d] JTAG pins: MTDI, MTCK, MTMS, and MTDO.

5 Interrupt Matrix (INTERRUPT)

5.1 Overview

The interrupt matrix embedded in ESP32-S3 independently allocates peripheral interrupt sources to the two CPUs' peripheral interrupts, to timely inform CPU0 or CPU1 to process the interrupts once the interrupt signals are generated.

Peripheral interrupt sources must be routed to CPU0/CPU1 peripheral interrupts via this interrupt matrix due to the following considerations:

- ESP32-S3 has 99 peripheral interrupt sources. To map them to 32 CPU0 interrupts or 32 CPU1 interrupts, this matrix is needed.
- Through this matrix, one peripheral interrupt source can be mapped to multiple CPU0 interrupts or CPU1 interrupts according to application requirements.

5.2 Features

- Accept 99 peripheral interrupt sources as input
- Generate 26 peripheral interrupts to CPU0 and 26 peripheral interrupts to CPU1 as output. Note that the remaining six CPU0 interrupts and six CPU1 interrupts are internal interrupts.
- Support to disable CPU non-maskable interrupt (NMI) sources
- Support to query current interrupt status of peripheral interrupt sources

Figure 5-1 shows the structure of the interrupt matrix.

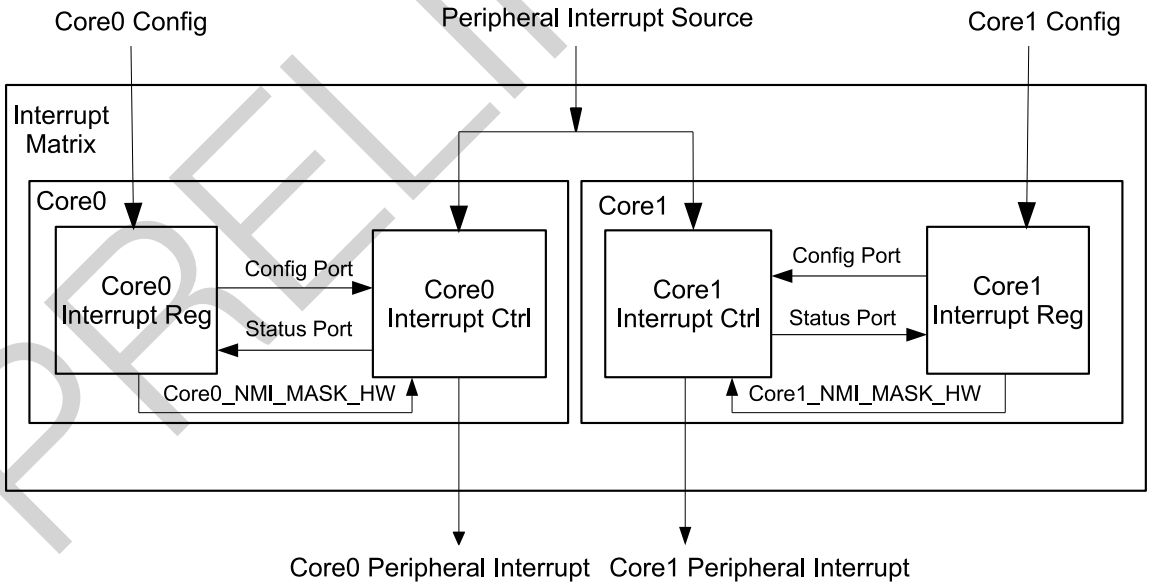


Figure 5-1. Interrupt Matrix Structure

All the interrupts generated by the peripheral interrupt sources can be handled by CPU0 or CPU1. Users can configure CPU0 interrupt registers ("Core0 Interrupt Reg" module in Figure 5-1) to allocate peripheral interrupt sources to CPU0, or configure CPU1 interrupt registers ("Core1 Interrupt Reg" module in Figure 5-1) to allocate

peripheral interrupt sources to CPU1. Peripheral interrupt sources can be allocated both to CPU0 and CPU1 simultaneously, if so, CPU0 and CPU1 will accept the interrupts.

5.3 Functional Description

5.3.1 Peripheral Interrupt Sources

ESP32-S3 has 99 peripheral interrupt sources in total. For the peripheral interrupt sources and their configuration/status registers, please refer to Table 5-1.

- Column “No.”: the peripheral interrupt source number, can be 0 ~ 98
- Column “Source”: all peripheral interrupt sources available
- Column “Configuration Register”: the registers used for routing the peripheral interrupt sources to CPU0/CPU1 peripheral interrupts
- Column “Status Register”: the registers used for indicating the interrupt status of peripheral interrupt sources
  - Column “Status Register - Bit”: the bit position in status registers
  - Column “Status Register - Name”: the name of status registers

The register in column “Configuration Register” and the bit in column “Bit” correspond to the peripheral interrupt source in column “Source”. For example, the configuration register for interrupt source MAC_INTR is INTERRUPT_CORE*x*_MAC_INTR_MAP_REG, and its status bit in INTERRUPT_CORE*x*_INTR_STATUS_0_REG is bit0.

Note that CORE*x* in the table can be CORE0 (CPU0) or CORE1 (CPU1).

Table 5-1. CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources

No.	Source	Configuration Register	Status Register	
			Bit	Name
0	MAC_INTR	INTERRUPT_COREx_MAC_INTR_MAP_REG	0	INTERRUPT_COREx_INTR_STATUS_0_REG
1	MAC_NMI	INTERRUPT_COREx_MAC_NMI_MAP_REG	1	
2	PWR_INTR	INTERRUPT_COREx_PWR_INTR_MAP_REG	2	
3	BB_INT	INTERRUPT_COREx_BB_INT_MAP_REG	3	
4	BT_MAC_INT	INTERRUPT_COREx_BT_MAC_INT_MAP_REG	4	
5	BT_BB_INT	INTERRUPT_COREx_BT_BB_INT_MAP_REG	5	
6	BT_BB_NMI	INTERRUPT_COREx_BT_BB_NMI_MAP_REG	6	
7	RWBT_IRQ	INTERRUPT_COREx_RWBT_IRQ_MAP_REG	7	
8	RWBLE_IRQ	INTERRUPT_COREx_RWBLE_IRQ_MAP_REG	8	
9	RWBT_NMI	INTERRUPT_COREx_RWBT_NMI_MAP_REG	9	
10	RWBLE_NMI	INTERRUPT_COREx_RWBLE_NMI_MAP_REG	10	
11	I2C_MST_INT	INTERRUPT_COREx_I2C_MST_INT_MAP_REG	11	
12	reserved	reserved	12	
13	reserved	reserved	13	
14	UHCI0_INTR	INTERRUPT_COREx_UHCI0_INTR_MAP_REG	14	
15	reserved	reserved	15	
16	GPIO_INTERRUPT_PRO	INTERRUPT_COREx_GPIO_INTERRUPT_PRO_MAP_REG	16	
17	GPIO_INTERRUPT_PRO_NMI	INTERRUPT_COREx_GPIO_INTERRUPT_PRO_NMI_MAP_REG	17	
18	reserved	reserved	18	
19	reserved	reserved	19	
20	SPI_INTR_1	INTERRUPT_COREx_SPI_INTR_1_MAP_REG	20	
21	SPI_INTR_2	INTERRUPT_COREx_SPI_INTR_2_MAP_REG	21	
22	SPI_INTR_3	INTERRUPT_COREx_SPI_INTR_3_MAP_REG	22	
23	reserved	reserved	23	
24	LCD_CAM_INT	INTERRUPT_COREx_LCD_CAM_INT_MAP_REG	24	
25	I2S0_INT	INTERRUPT_COREx_I2S0_INT_MAP_REG	25	
26	I2S1_INT	INTERRUPT_COREx_I2S1_INT_MAP_REG	26	
27	UART_INTR	INTERRUPT_COREx_UART_INTR_MAP_REG	27	
28	UART1_INTR	INTERRUPT_COREx_UART1_INTR_MAP_REG	28	
29	UART2_INTR	INTERRUPT_COREx_UART2_INTR_MAP_REG	29	
30	SDIO_HOST_INTERRUPT	INTERRUPT_COREx_SDIO_HOST_INTERRUPT_MAP_REG	30	
31	PWM0_INTR	INTERRUPT_COREx_PWM0_INTR_MAP_REG	31	
32	PWM1_INTR	INTERRUPT_COREx_PWM1_INTR_MAP_REG	0	INTERRUPT_COREx_INTR_STATUS_1_REG
33	reserved	reserved	1	
34	reserved	reserved	2	

No.	Source	Configuration Register	Status Register	
			Bit	Name
35	LEDC_INT	INTERRUPT_COREx_LEDC_INT_MAP_REG	3	
36	EFUSE_INT	INTERRUPT_COREx_EFUSE_INT_MAP_REG	4	
37	CAN_INT	INTERRUPT_COREx_CAN_INT_MAP_REG	5	
38	USB_INTR	INTERRUPT_COREx_USB_INTR_MAP_REG	6	
39	RTC_CORE_INTR	INTERRUPT_COREx_RTC_CORE_INTR_MAP_REG	7	
40	RMT_INTR	INTERRUPT_COREx_RMT_INTR_MAP_REG	8	
41	PCNT_INTR	INTERRUPT_COREx_PCNT_INTR_MAP_REG	9	
42	I2C_EXT0_INTR	INTERRUPT_COREx_I2C_EXT0_INTR_MAP_REG	10	
43	I2C_EXT1_INTR	INTERRUPT_COREx_I2C_EXT1_INTR_MAP_REG	11	
44	reserved	reserved	12	
45	reserved	reserved	13	
46	reserved	reserved	14	
47	reserved	reserved	15	
48	reserved	reserved	16	
49	reserved	reserved	17	
50	TG_T0_INT	INTERRUPT_COREx_TG_T0_INT_MAP_REG	18	
51	TG_T1_INT	INTERRUPT_COREx_TG_T1_INT_MAP_REG	19	INTERRUPT_COREx_INTR_STATUS_1_REG
52	TG_WDT_INT	INTERRUPT_COREx_TG_WDT_INT_MAP_REG	20	
53	TG1_T0_INT	INTERRUPT_COREx_TG1_T0_INT_MAP_REG	21	
54	TG1_T1_INT	INTERRUPT_COREx_TG1_T1_INT_MAP_REG	22	
55	TG1_WDT_INT	INTERRUPT_COREx_TG1_WDT_INT_MAP_REG	23	
56	CACHE_IA_INT	INTERRUPT_COREx_CACHE_IA_INT_MAP_REG	24	
57	SYSTIMER_TARGET0_INT	INTERRUPT_COREx_SYSTIMER_TARGET0_INT_MAP_REG	25	
58	SYSTIMER_TARGET1_INT	INTERRUPT_COREx_SYSTIMER_TARGET1_INT_MAP_REG	26	
59	SYSTIMER_TARGET2_INT	INTERRUPT_COREx_SYSTIMER_TARGET2_INT_MAP_REG	27	
60	SPI_MEM_REJECT_INTR	INTERRUPT_COREx_SPI_MEM_REJECT_INTR_MAP_REG	28	
61	DCACHE_PRELOAD_INT	INTERRUPT_COREx_DCACHE_PRELOAD_INT_MAP_REG	29	
62	ICACHE_PRELOAD_INT	INTERRUPT_COREx_ICACHE_PRELOAD_INT_MAP_REG	30	
63	DCACHE_SYNC_INT	INTERRUPT_COREx_DCACHE_SYNC_INT_MAP_REG	31	
64	ICACHE_SYNC_INT	INTERRUPT_COREx_ICACHE_SYNC_INT_MAP_REG	0	
65	APB_ADC_INT	INTERRUPT_COREX_APB_ADC_INT_MAP_REG	1	
66	DMA_IN_CH0_INT	INTERRUPT_COREX_DMA_IN_CH0_INT_MAP_REG	2	
67	DMA_IN_CH1_INT	INTERRUPT_COREx_DMA_IN_CH1_INT_MAP_REG	3	
68	DMA_IN_CH2_INT	INTERRUPT_COREx_DMA_IN_CH2_INT_MAP_REG	4	INTERRUPT_COREx_INTR_STATUS_2_REG
69	DMA_IN_CH3_INT	INTERRUPT_COREx_DMA_IN_CH3_INT_MAP_REG	5	
70	DMA_IN_CH4_INT	INTERRUPT_COREx_DMA_IN_CH4_INT_MAP_REG	6	
71	DMA_OUT_CH0_INT	INTERRUPT_COREX_DMA_OUT_CH0_INT_MAP_REG	7	

No.	Source	Configuration Register	Status Register	
			Bit	Name
72	DMA_OUT_CH1_INT	INTERRUPT_COREx_DMA_OUT_CH1_INT_MAP_REG	8	
73	DMA_OUT_CH2_INT	INTERRUPT_COREx_DMA_OUT_CH2_INT_MAP_REG	9	
74	DMA_OUT_CH3_INT	INTERRUPT_COREx_DMA_OUT_CH3_INT_MAP_REG	10	
75	DMA_OUT_CH4_INT	INTERRUPT_COREx_DMA_OUT_CH4_INT_MAP_REG	11	
76	RSA_INTR	INTERRUPT_COREx_RSA_INTR_MAP_REG	12	
77	AES_INTR	INTERRUPT_COREx_AES_INTR_MAP_REG	13	
78	SHA_INTR	INTERRUPT_COREx_SHA_INTR_MAP_REG	14	
79	CPU_INTR_FROM_CPU_0	INTERRUPT_COREx_CPU_INTR_FROM_CPU_0_MAP_REG	15	
80	CPU_INTR_FROM_CPU_1	INTERRUPT_COREx_CPU_INTR_FROM_CPU_1_MAP_REG	16	
81	CPU_INTR_FROM_CPU_2	INTERRUPT_COREx_CPU_INTR_FROM_CPU_2_MAP_REG	17	
82	CPU_INTR_FROM_CPU_3	INTERRUPT_COREx_CPU_INTR_FROM_CPU_3_MAP_REG	18	
83	ASSIST_DEBUG_INTR	INTERRUPT_COREx_ASSIST_DEBUG_INTR_MAP_REG	19	INTERRUPT_COREx_INTR_STATUS_2_REG
84	DMA_APB_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_DMA_APB_PMS_MONITOR_VIOLATE_INTR_MAP_REG	20	
85	CORE_0_IRAM0_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_0_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	21	
86	CORE_0_DRAM0_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_0_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	22	
87	CORE_0_PIF_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_0_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	23	
88	CORE_0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR	INTERRUPT_COREx_CORE_0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	24	
89	CORE_1_IRAM0_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_1_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	25	
90	CORE_1_DRAM0_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_1_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	26	
91	CORE_1_PIF_PMS_MONITOR_VIOLATE_INTR	INTERRUPT_COREx_CORE_1_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	27	
92	CORE_1_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR	INTERRUPT_COREx_CORE_1_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	28	
93	BACKUP_PMS_VIOLATE_INT	INTERRUPT_COREx_BACKUP_PMS_VIOLATE_INTR_MAP_REG	29	
94	CACHE_CORE0_ACS_INT	INTERRUPT_COREx_CACHE_CORE0_ACS_INT_MAP_REG	30	
95	CACHE_CORE1_ACS_INT	INTERRUPT_COREx_CACHE_CORE1_ACS_INT_MAP_REG	31	
96	USB_DEVICE_INT	INTERRUPT_COREx_USB_DEVICE_INT_MAP_REG	0	
97	PERI_BACKUP_INT	INTERRUPT_COREx_PERI_BACKUP_INT_MAP_REG	1	INTERRUPT_COREx_INTR_STATUS_3_REG
98	DMA_EXTMEM_REJECT_INT	INTERRUPT_COREx_DMA_EXTMEM_REJECT_INT_MAP_REG	2	

5.3.2 CPU Interrupts

Each CPU has 32 interrupts, numbered from 0 ~ 31, including 26 peripheral interrupts and six internal interrupts.

- Peripheral interrupts: triggered by peripheral interrupt sources, include the following types:
  - Level-triggered interrupts: triggered by a high level signal. The interrupt sources should hold the level till the CPU*x* handles the interrupts.
  - Edge-triggered interrupts: triggered on a rising edge. CPU*x* responds to this kind of interrupts immediately.
  - NMI interrupt: once triggered, the NMI interrupt can not be masked by software using the CPU*x* internal registers. World Controller provides a way to mask this kind of interrupt. For more information, see Chapter 19 *World Controller (WCTL) [to be added later]*.
- Internal interrupts: generated inside CPU*x*, include the following types:
  - 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.

Level-triggered and edge-triggered both describe the ways of CPU*x* to accept interrupt signals. For level-triggered interrupts, the level of interrupt signal should be kept till the CPU handles the interrupt, otherwise the interrupt may be lost. For edge-triggered interrupts, when a rising edge is detected, this edge will be recorded by CPU*x*, which then allows the interrupt signal to be released.

Interrupt matrix routes the peripheral interrupt sources to any of the CPU*x* peripheral interrupts. By such way, CPU*x* can receive the interrupt signals from peripheral interrupt sources. Table 5-2 lists all the interrupts and their types as well as priorities.

ESP32-S3 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*x* must handle such interrupt. Nested interrupts are also supported, i.e. low-level interrupts can be stopped by high-level interrupts.

Table 5-2. CPU Interrupts

No.	Category	Type	Priority
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	Level-triggered	1

No.	Category	Type	Priority
11	Internal	Profiling	3
12	Peripheral	Level-triggered	1
13	Peripheral	Level-triggered	1
14	Peripheral	NMI	NMI
15	Internal	Timer.1	3
16	Internal	Timer.2	5
17	Peripheral	Level-triggered	1
18	Peripheral	Level-triggered	1
19	Peripheral	Level-triggered	2
20	Peripheral	Level-triggered	2
21	Peripheral	Level-triggered	2
22	Peripheral	Level-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	Level-triggered	4
29	Internal	Software	3
30	Peripheral	Level-triggered	4
31	Peripheral	Level-triggered	5

5.3.3 Allocate Peripheral Interrupt Source to CPU*x* Interrupt

In this section, the following terms are used to describe the operation of the interrupt matrix.

- Source_*Y*: stands for a peripheral interrupt source, wherein, *Y* means the number of this interrupt source in Table 5-1.
- INTERRUPT_CORE*x*_SOURCE_*Y*_MAP_REG: stands for a configuration register for the peripheral interrupt source (Source_*Y*) of CPU*x*.
- Interrupt_P: stands for the CPU*x* 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 5-2.
- Interrupt_I: stands for the CPU*x* internal interrupt numbered as Num_I. The value of Num_I can be 6, 7, 11, 15, 16, and 29. See Table 5-2.

5.3.3.1 Allocate one peripheral interrupt source (Source_*Y*) to CPU*x*

Setting the corresponding configuration register INTERRUPT_CORE*x*_SOURCE_*Y*_MAP_REG of Source_*Y* to Num

_P allocates 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*x* interrupt can be shared by multiple peripherals.

5.3.3.2 Allocate multiple peripheral interrupt sources (Source_Y*n*) to CPU*x*

Setting the corresponding configuration register INTERRUPT_CORE*x*_SOURCE_Y*n*_MAP_REG of each interrupt source to the same Num_P allocates multiple sources to the same Interrupt_P. Any of these sources can trigger CPU*x* Interrupt_P. When an interrupt signal is generated, CPU*x* checks the interrupt status registers to figure out which peripheral the signal comes from.

5.3.3.3 Disable CPU*x* peripheral interrupt source (Source_*Y*)

Setting the corresponding configuration register INTERRUPT_CORE*x*_SOURCE_*Y*_MAP_REG of the source to any Num_I disables this interrupt Source_*Y*. The choice of Num_I (6, 7, 11, 15, 16, 29) does not matter, as none of peripheral interrupt sources allocated to Num_I is connected to the CPU*x*. Therefore this functionality can be used to disable peripheral interrupt sources.

5.3.4 Disable CPU*x* NMI Interrupt

All CPU*x* interrupts, except for NMI interrupt (No.14 in Table 5-2), can be masked and enabled by software using CPU special register (INTENABLE). NMI interrupt can not be masked by the way above, but ESP32-S3 provides two ways to mask NMI interrupt:

- Disconnect peripheral interrupt sources from NMI interrupt, i.e. the sources routed to NMI interrupt before are now routed to other interrupts. By such way, the previous NMI interrupt is maskable.
- Connect peripheral interrupt sources with NMI interrupt, but use World Controller module to mask NMI interrupt. For more information, see Chapter Chapter 19 *World Controller (WCTL) [to be added later]*.

5.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

Users can query current interrupt status of a CPU*x* peripheral interrupt source by reading the bit value in INTERRUPT_CORE*x*_INTR_STATUS_*n*_REG (read only). For the mapping between INTERRUPT_CORE*x*_INTR_STATUS_*n*_REG and peripheral interrupt sources, please refer to Table 5-1.

5.4 Register Summary

The addresses in this section are relative to the Interrupt Matrix base address provided in Table 1-4 in Chapter 1 *System and Memory*.

5.4.1 CPU0 Interrupt Register Summary

Name	Description	Address	Access
Configuration Registers			
INTERRUPT_CORE0_MAC_INTR_MAP_REG	MAC interrupt configuration register	0x0000	R/W
INTERRUPT_CORE0_MAC_NMI_MAP_REG	MAC_NMI interrupt configuration register	0x0004	R/W
INTERRUPT_CORE0_PWR_INTR_MAP_REG	PWR interrupt configuration register	0x0008	R/W
INTERRUPT_CORE0_BB_INT_MAP_REG	BB interrupt configuration register	0x000C	R/W
INTERRUPT_CORE0_BT_MAC_INT_MAP_REG	BB_MAC interrupt configuration register	0x0010	R/W
INTERRUPT_CORE0_BT_BB_INT_MAP_REG	BT_BB interrupt configuration register	0x0014	R/W
INTERRUPT_CORE0_BT_BB_NMI_MAP_REG	BT_BB_NMI interrupt configuration register	0x0018	R/W
INTERRUPT_CORE0_RWBT_IRQ_MAP_REG	RWBT_IRQ interrupt configuration register	0x001C	R/W
INTERRUPT_CORE0_RWBLE_IRQ_MAP_REG	RWBLE_IRQ interrupt configuration register	0x0020	R/W
INTERRUPT_CORE0_RWBT_NMI_MAP_REG	RWBT_NMI interrupt configuration register	0x0024	R/W
INTERRUPT_CORE0_RWBLE_NMI_MAP_REG	RWBLE_NMI interrupt configuration register	0x0028	R/W
INTERRUPT_CORE0_I2C_MST_INT_MAP_REG	I2C_MST interrupt configuration register	0x002C	R/W
INTERRUPT_CORE0_UHCI0_INTR_MAP_REG	UHCI0 interrupt configuration register	0x0038	R/W
INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO interrupt configuration register	0x0040	R/W
INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_NMI_MAP_REG	GPIO_INTERRUPT_PRO_NMI interrupt configuration register	0x0044	R/W
INTERRUPT_CORE0_SPI_INTR_1_MAP_REG	SPI_INTR_1 interrupt configuration register	0x0050	R/W
INTERRUPT_CORE0_SPI_INTR_2_MAP_REG	SPI_INTR_2 interrupt configuration register	0x0054	R/W
INTERRUPT_CORE0_SPI_INTR_3_MAP_REG	SPI_INTR_3 interrupt configuration register	0x0058	R/W
INTERRUPT_CORE0_LCD_CAM_INT_MAP_REG	LCD_CAM interrupt configuration register	0x0060	R/W
INTERRUPT_CORE0_I2S0_INT_MAP_REG	I2S0 interrupt configuration register	0x0064	R/W
INTERRUPT_CORE0_I2S1_INT_MAP_REG	I2S1 interrupt configuration register	0x0068	R/W
INTERRUPT_CORE0_UART_INTR_MAP_REG	UART interrupt configuration register	0x006C	R/W
INTERRUPT_CORE0_UART1_INTR_MAP_REG	UART1 interrupt configuration register	0x0070	R/W
INTERRUPT_CORE0_UART2_INTR_MAP_REG	UART2 interrupt configuration register	0x0074	R/W
INTERRUPT_CORE0_SDIO_HOST_INTERRUPT_MAP_REG	SDIO_HOST interrupt configuration register	0x0078	R/W
INTERRUPT_CORE0_PWM0_INTR_MAP_REG	PWM0 interrupt configuration register	0x007C	R/W

Name	Description	Address	Access
INTERRUPT_CORE0_PWM1_INTR_MAP_REG	PWM1 interrupt configuration register	0x0080	R/W
INTERRUPT_CORE0_LEDC_INT_MAP_REG	LEDC interrupt configuration register	0x008C	R/W
INTERRUPT_CORE0_EFUSE_INT_MAP_REG	EFUSE interrupt configuration register	0x0090	R/W
INTERRUPT_CORE0_CAN_INT_MAP_REG	CAN interrupt configuration register	0x0094	R/W
INTERRUPT_CORE0_USB_INTR_MAP_REG	USB interrupt configuration register	0x0098	R/W
INTERRUPT_CORE0_RTC_CORE_INTR_MAP_REG	RTC_CORE interrupt configuration register	0x009C	R/W
INTERRUPT_CORE0_RMT_INTR_MAP_REG	RMT interrupt configuration register	0x00A0	R/W
INTERRUPT_CORE0_PCNT_INTR_MAP_REG	PCNT interrupt configuration register	0x00A4	R/W
INTERRUPT_CORE0_I2C_EXT0_INTR_MAP_REG	I2C_EXT0 interrupt configuration register	0x00A8	R/W
INTERRUPT_CORE0_I2C_EXT1_INTR_MAP_REG	I2C_EXT1 interrupt configuration register	0x00AC	R/W
INTERRUPT_CORE0_TG_T0_INT_MAP_REG	TG_T0 interrupt configuration register	0x00C8	R/W
INTERRUPT_CORE0_TG_T1_INT_MAP_REG	TG_T1 interrupt configuration register	0x00CC	R/W
INTERRUPT_CORE0_TG_WDT_INT_MAP_REG	TG_WDT interrupt configuration register	0x00D0	R/W
INTERRUPT_CORE0_TG1_T0_INT_MAP_REG	TG1_T0 interrupt configuration register	0x00D4	R/W
INTERRUPT_CORE0_TG1_T1_INT_MAP_REG	TG1_T1 interrupt configuration register	0x00D8	R/W
INTERRUPT_CORE0_TG1_WDT_INT_MAP_REG	TG1_WDT interrupt configuration register	0x00DC	R/W
INTERRUPT_CORE0_CACHE_IA_INT_MAP_REG	CACHE_IA interrupt configuration register	0x00E0	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0 interrupt configuration register	0x00E4	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1 interrupt configuration register	0x00E8	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2 interrupt configuration register	0x00EC	R/W
INTERRUPT_CORE0_SPI_MEM_REJECT_INTR_MAP_REG	SPI_MEM_REJECT interrupt configuration register	0x00F0	R/W
INTERRUPT_CORE0_DCACHE_PRELOAD_INT_MAP_REG	DCACHE_PRELAOD interrupt configuration register	0x00F4	R/W
INTERRUPT_CORE0_ICACHE_PRELOAD_INT_MAP_REG	ICACHE_PRELOAD interrupt configuration register	0x00F8	R/W
INTERRUPT_CORE0_DCACHE_SYNC_INT_MAP_REG	DCACHE_SYNC interrupt configuration register	0x00FC	R/W
INTERRUPT_CORE0_ICACHE_SYNC_INT_MAP_REG	ICACHE_SYNC interrupt configuration register	0x0100	R/W
INTERRUPT_CORE0_APB_ADC_INT_MAP_REG	APB_ADC interrupt configuration register	0x0104	R/W
INTERRUPT_CORE0_DMA_IN_CH0_INT_MAP_REG	DMA_IN_CH0 interrupt configuration register	0x0108	R/W
INTERRUPT_CORE0_DMA_IN_CH1_INT_MAP_REG	DMA_IN_CH1 interrupt configuration register	0x010C	R/W
INTERRUPT_CORE0_DMA_IN_CH2_INT_MAP_REG	DMA_IN_CH2 interrupt configuration register	0x0110	R/W

Name	Description	Address	Access
INTERRUPT_CORE0_DMA_IN_CH3_INT_MAP_REG	DMA_IN_CH3 interrupt configuration register	0x0114	R/W
INTERRUPT_CORE0_DMA_IN_CH4_INT_MAP_REG	DMA_IN_CH4 interrupt configuration register	0x0118	R/W
INTERRUPT_CORE0_DMA_OUT_CH0_INT_MAP_REG	DMA_OUT_CH0 interrupt configuration register	0x011C	R/W
INTERRUPT_CORE0_DMA_OUT_CH1_INT_MAP_REG	DMA_OUT_CH1 interrupt configuration register	0x0120	R/W
INTERRUPT_CORE0_DMA_OUT_CH2_INT_MAP_REG	DMA_OUT_CH2 interrupt configuration register	0x0124	R/W
INTERRUPT_CORE0_DMA_OUT_CH3_INT_MAP_REG	DMA_OUT_CH3 interrupt configuration register	0x0128	R/W
INTERRUPT_CORE0_DMA_OUT_CH4_INT_MAP_REG	DMA_OUT_CH4 interrupt configuration register	0x012C	R/W
INTERRUPT_CORE0_RSA_INT_MAP_REG	RSA interrupt configuration register	0x0130	R/W
INTERRUPT_CORE0_AES_INT_MAP_REG	AES interrupt configuration register	0x0134	R/W
INTERRUPT_CORE0_SHA_INT_MAP_REG	SHA interrupt configuration register	0x0138	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 interrupt configuration register	0x013C	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 interrupt configuration register	0x0140	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 interrupt configuration register	0x0144	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 interrupt configuration register	0x0148	R/W
INTERRUPT_CORE0_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG interrupt configuration register	0x014C	R/W
INTERRUPT_CORE0_DMA_APBPERI_PMS_MONITOR_VIOLATE_INTR_MAP_REG	dma_pms_monitor_violatile interrupt configuration register	0x0150	R/W
INTERRUPT_CORE0_CORE_0_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_IRam0_pms_monitor_violatile interrupt configuration register	0x0154	R/W
INTERRUPT_CORE0_CORE_0_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_DRam0_pms_monitor_violatile interrupt configuration register	0x0158	R/W
INTERRUPT_CORE0_CORE_0_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_PIF_pms_monitor_violatile interrupt configuration register	0x015C	R/W
INTERRUPT_CORE0_CORE_0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	core0_PIF_pms_monitor_violatile_size interrupt configuration register	0x0160	R/W
INTERRUPT_CORE0_CORE_1_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_IRam0_pms_monitor_violatile interrupt configuration register	0x0164	R/W
INTERRUPT_CORE0_CORE_1_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_DRam0_pms_monitor_violatile interrupt configuration register	0x0168	R/W

Name	Description	Address	Access
INTERRUPT_CORE0_CORE_1_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_PIF_pms_monitor_violatile interrupt configuration register	0x016C	R/W
INTERRUPT_CORE0_CORE_1_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	core1_PIF_pms_monitor_violatile_size interrupt configuration register	0x0170	R/W
INTERRUPT_CORE0_BACKUP_PMS_VIOLATE_INTR_MAP_REG	BACKUP_PMS_MONITOR_VIOLATILE interrupt configuration register	0x0174	R/W
INTERRUPT_CORE0_CACHE_CORE0_ACS_INT_MAP_REG	CACHE_CORE0_ACS interrupt configuration register	0x0178	R/W
INTERRUPT_CORE0_CACHE_CORE1_ACS_INT_MAP_REG	CACHE_CORE1_ACS interrupt configuration register	0x017C	R/W
INTERRUPT_CORE0_USB_DEVICE_INT_MAP_REG	USB_DEVICE interrupt configuration register	0x0180	R/W
INTERRUPT_CORE0_PERI_BACKUP_INT_MAP_REG	PERI_BACKUP interrupt configuration register	0x0184	R/W
INTERRUPT_CORE0_DMA_EXTMEM_REJECT_INT_MAP_REG	DMA_EXTMEM_REJECT interrupt configuration register	0x0188	R/W
Status Registers			
INTERRUPT_CORE0_INTR_STATUS_0_REG	Interrupt status register	0x018C	RO
INTERRUPT_CORE0_INTR_STATUS_1_REG	Interrupt status register	0x0190	RO
INTERRUPT_CORE0_INTR_STATUS_2_REG	Interrupt status register	0x0194	RO
INTERRUPT_CORE0_INTR_STATUS_3_REG	Interrupt status register	0x0198	RO
Clock Register			
INTERRUPT_CORE0_CLOCK_GATE_REG	Clock gate register	0x019C	R/W
Version Register			
INTERRUPT_CORE0_DATE_REG	Version control register	0x07FC	R/W

5.4.2 CPU1 Interrupt Register Summary

Name	Description	Address	Access
Configuration Registers			
INTERRUPT_CORE1_MAC_INTR_MAP_REG	MAC interrupt configuration register	0x0800	R/W
INTERRUPT_CORE1_MAC_NMI_MAP_REG	MAC_NMI interrupt configuration register	0x0804	R/W
INTERRUPT_CORE1_PWR_INTR_MAP_REG	PWR interrupt configuration register	0x0808	R/W
INTERRUPT_CORE1_BB_INT_MAP_REG	BB interrupt configuration register	0x080C	R/W

Name	Description	Address	Access
INTERRUPT_CORE1_BT_MAC_INT_MAP_REG	BB_MAC interrupt configuration register	0x0810	R/W
INTERRUPT_CORE1_BT_BB_INT_MAP_REG	BT_BB interrupt configuration register	0x0814	R/W
INTERRUPT_CORE1_BT_BB_NMI_MAP_REG	BT_BB_NMI interrupt configuration register	0x0818	R/W
INTERRUPT_CORE1_RWBT_IRQ_MAP_REG	RWBT_IRQ interrupt configuration register	0x081C	R/W
INTERRUPT_CORE1_RWBLE_IRQ_MAP_REG	RWBLE_IRQ interrupt configuration register	0x0820	R/W
INTERRUPT_CORE1_RWBT_NMI_MAP_REG	RWBT_NMI interrupt configuration register	0x0824	R/W
INTERRUPT_CORE1_RWBLE_NMI_MAP_REG	RWBLE_NMI interrupt configuration register	0x0828	R/W
INTERRUPT_CORE1_I2C_MST_INT_MAP_REG	I2C_MST interrupt configuration register	0x082C	R/W
INTERRUPT_CORE1_UHCI0_INTR_MAP_REG	UHCI0 interrupt configuration register	0x0838	R/W
INTERRUPT_CORE1_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO interrupt configuration register	0x0840	R/W
INTERRUPT_CORE1_GPIO_INTERRUPT_PRO_NMI_MAP_REG	GPIO_INTERRUPT_PRO_NMI Interrupt configuration register	0x0844	R/W
INTERRUPT_CORE1_SPI_INTR_1_MAP_REG	SPI_INTR_1 interrupt configuration register	0x0850	R/W
INTERRUPT_CORE1_SPI_INTR_2_MAP_REG	SPI_INTR_2 interrupt configuration register	0x0854	R/W
INTERRUPT_CORE1_SPI_INTR_3_MAP_REG	SPI_INTR_3 interrupt configuration register	0x0858	R/W
INTERRUPT_CORE1_LCD_CAM_INT_MAP_REG	LCD_CAM interrupt configuration register	0x0860	R/W
INTERRUPT_CORE1_I2S0_INT_MAP_REG	I2S0 interrupt configuration register	0x0864	R/W
INTERRUPT_CORE1_I2S1_INT_MAP_REG	I2S1 interrupt configuration register	0x0868	R/W
INTERRUPT_CORE1_UART_INTR_MAP_REG	UART interrupt configuration register	0x086C	R/W
INTERRUPT_CORE1_UART1_INTR_MAP_REG	UART1 interrupt configuration register	0x0870	R/W
INTERRUPT_CORE1_UART2_INTR_MAP_REG	UART2 interrupt configuration register	0x0874	R/W
INTERRUPT_CORE1_SDIO_HOST_INTERRUPT_MAP_REG	SDIO_HOST interrupt configuration register	0x0878	R/W
INTERRUPT_CORE1_PWM0_INTR_MAP_REG	PWM0 interrupt configuration register	0x087C	R/W
INTERRUPT_CORE1_PWM1_INTR_MAP_REG	PWM1 interrupt configuration register	0x0880	R/W
INTERRUPT_CORE1_LEDC_INT_MAP_REG	LEDC interrupt configuration register	0x088C	R/W
INTERRUPT_CORE1_EFUSE_INT_MAP_REG	EFUSE interrupt configuration register	0x0890	R/W
INTERRUPT_CORE1_CAN_INT_MAP_REG	CAN interrupt configuration register	0x0894	R/W
INTERRUPT_CORE1_USB_INTR_MAP_REG	USB interrupt configuration register	0x0898	R/W
INTERRUPT_CORE1_RTC_CORE_INTR_MAP_REG	RTC_CORE interrupt configuration register	0x089C	R/W
INTERRUPT_CORE1_RMT_INTR_MAP_REG	RMT interrupt configuration register	0x08A0	R/W

Name	Description	Address	Access
INTERRUPT_CORE1_PCNT_INTR_MAP_REG	PCNT interrupt configuration register	0x08A4	R/W
INTERRUPT_CORE1_I2C_EXT0_INTR_MAP_REG	I2C_EXT0 interrupt configuration register	0x08A8	R/W
INTERRUPT_CORE1_I2C_EXT1_INTR_MAP_REG	I2C_EXT1 interrupt configuration register	0x08AC	R/W
INTERRUPT_CORE1_TG_T1_INT_MAP_REG	TG_T1 interrupt configuration register	0x08CC	R/W
INTERRUPT_CORE1_TG_WDT_INT_MAP_REG	TG_WDT interrupt configuration register	0x08D0	R/W
INTERRUPT_CORE1_TG1_T0_INT_MAP_REG	TG1_T0 interrupt configuration register	0x08D4	R/W
INTERRUPT_CORE1_TG1_T1_INT_MAP_REG	TG1_T1 interrupt configuration register	0x08D8	R/W
INTERRUPT_CORE1_TG1_WDT_INT_MAP_REG	TG1_WDT interrupt configuration register	0x08DC	R/W
INTERRUPT_CORE1_CACHE_IA_INT_MAP_REG	CACHE_IA interrupt configuration register	0x08E0	R/W
INTERRUPT_CORE1_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0 interrupt configuration register	0x08E4	R/W
INTERRUPT_CORE1_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1 interrupt configuration register	0x08E8	R/W
INTERRUPT_CORE1_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2 interrupt configuration register	0x08EC	R/W
INTERRUPT_CORE1_SPI_MEM_REJECT_INTR_MAP_REG	SPI_MEM_REJECT interrupt configuration register	0x08F0	R/W
INTERRUPT_CORE1_DCACHE_PRELOAD_INT_MAP_REG	DCACHE_PRELAOD interrupt configuration register	0x08F4	R/W
INTERRUPT_CORE1_ICACHE_PRELOAD_INT_MAP_REG	ICACHE_PRELOAD interrupt configuration register	0x08F8	R/W
INTERRUPT_CORE1_DCACHE_SYNC_INT_MAP_REG	DCACHE_SYNC interrupt configuration register	0x08FC	R/W
INTERRUPT_CORE1_ICACHE_SYNC_INT_MAP_REG	ICACHE_SYNC interrupt configuration register	0x0900	R/W
INTERRUPT_CORE1_APB_ADC_INT_MAP_REG	APB_ADC interrupt configuration register	0x0904	R/W
INTERRUPT_CORE1_DMA_IN_CH0_INT_MAP_REG	DMA_IN_CH0 interrupt configuration register	0x0908	R/W
INTERRUPT_CORE1_DMA_IN_CH1_INT_MAP_REG	DMA_IN_CH1 interrupt configuration register	0x090C	R/W
INTERRUPT_CORE1_DMA_IN_CH2_INT_MAP_REG	DMA_IN_CH2 interrupt configuration register	0x0910	R/W
INTERRUPT_CORE1_DMA_IN_CH3_INT_MAP_REG	DMA_IN_CH3 interrupt configuration register	0x0914	R/W
INTERRUPT_CORE1_DMA_IN_CH4_INT_MAP_REG	DMA_IN_CH4 interrupt configuration register	0x0918	R/W
INTERRUPT_CORE1_DMA_OUT_CH0_INT_MAP_REG	DMA_OUT_CH0 interrupt configuration register	0x091C	R/W
INTERRUPT_CORE1_DMA_OUT_CH1_INT_MAP_REG	DMA_OUT_CH1 interrupt configuration register	0x0920	R/W
INTERRUPT_CORE1_DMA_OUT_CH2_INT_MAP_REG	DMA_OUT_CH2 interrupt configuration register	0x0924	R/W
INTERRUPT_CORE1_DMA_OUT_CH3_INT_MAP_REG	DMA_OUT_CH3 interrupt configuration register	0x0928	R/W
INTERRUPT_CORE1_DMA_OUT_CH4_INT_MAP_REG	DMA_OUT_CH4 interrupt configuration register	0x092C	R/W
INTERRUPT_CORE1_RSA_INT_MAP_REG	RSA interrupt configuration register	0x0930	R/W

Name	Description	Address	Access
INTERRUPT_CORE1_AES_INT_MAP_REG	AES interrupt configuration register	0x0934	R/W
INTERRUPT_CORE1_SHA_INT_MAP_REG	SHA interrupt configuration register	0x0938	R/W
INTERRUPT_CORE1_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 interrupt configuration register	0x093C	R/W
INTERRUPT_CORE1_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 interrupt configuration register	0x0940	R/W
INTERRUPT_CORE1_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 interrupt configuration register	0x0944	R/W
INTERRUPT_CORE1_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 interrupt configuration register	0x0948	R/W
INTERRUPT_CORE1_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG interrupt configuration register	0x094C	R/W
INTERRUPT_CORE1_DMA_APBPERI_PMS_MONITOR_VIOLATE_INTR_MAP_REG	dma_pms_monitor_violatile interrupt configuration register	0x0950	R/W
INTERRUPT_CORE1_CORE_0_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_IRam0_pms_monitor_violatile interrupt configuration register	0x0954	R/W
INTERRUPT_CORE1_CORE_0_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_DRam0_pms_monitor_violatile interrupt configuration register	0x0958	R/W
INTERRUPT_CORE1_CORE_0_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core0_PIF_pms_monitor_violatile interrupt configuration register	0x095C	R/W
INTERRUPT_CORE1_CORE_0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	core0_PIF_pms_monitor_violatile_size interrupt configuration register	0x0960	R/W
INTERRUPT_CORE1_CORE_1_IRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_IRam0_pms_monitor_violatile interrupt configuration register	0x0964	R/W
INTERRUPT_CORE1_CORE_1_DRAM0_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_DRam0_pms_monitor_violatile interrupt configuration register	0x0968	R/W
INTERRUPT_CORE1_CORE_1_PIF_PMS_MONITOR_VIOLATE_INTR_MAP_REG	core1_PIF_pms_monitor_violatile interrupt configuration register	0x096C	R/W
INTERRUPT_CORE1_CORE_1_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	core1_PIF_pms_monitor_violatile_size interrupt configuration register	0x0970	R/W
INTERRUPT_CORE1_BACKUP_PMS_VIOLATE_INTR_MAP_REG	BACKUP_PMS_MONITOR_VIOLATILE interrupt configuration register	0x0974	R/W
INTERRUPT_CORE1_CACHE_CORE0_ACS_INT_MAP_REG	CACHE_CORE0_ACS interrupt configuration register REG	0x0978	R/W
INTERRUPT_CORE1_CACHE_CORE1_ACS_INT_MAP_REG	CACHE_CORE1_ACS interrupt configuration register REG	0x097C	R/W

Name	Description	Address	Access
INTERRUPT_CORE1_USB_DEVICE_INT_MAP_REG	USB_DEVICE interrupt configuration register	0x0980	R/W
INTERRUPT_CORE1_PERI_BACKUP_INT_MAP_REG	PERI_BACKUP interrupt configuration register	0x0984	R/W
INTERRUPT_CORE1_DMA_EXTMEM_REJECT_INT_MAP_REG	DMA_EXTMEM_REJECT interrupt configuration register	0x0988	R/W
Status Registers			
INTERRUPT_CORE1_INTR_STATUS_0_REG	Interrupt status register	0x098C	RO
INTERRUPT_CORE1_INTR_STATUS_1_REG	Interrupt status register	0x0990	RO
INTERRUPT_CORE1_INTR_STATUS_2_REG	Interrupt status register	0x0994	RO
INTERRUPT_CORE1_INTR_STATUS_3_REG	Interrupt status register	0x0998	RO
Clock Register			
INTERRUPT_CORE1_CLOCK_GATE_REG	Clock gate register	0x099C	R/W
Version Register			
INTERRUPT_CORE1_DATE_REG	Version control register	0x0FFC	R/W

5.5 Registers

5.5.1 CPU0 Interrupt Registers

Register 5.1. INTERRUPT_CORE0_*MAC_INTR*_MAP_REG (0x0000)

Register 5.2. INTERRUPT_CORE0_*MAC_NMI*_MAP_REG (0x0004)

Register 5.3. INTERRUPT_CORE0_*PWR_INTR*_MAP_REG (0x0008)

Register 5.4. INTERRUPT_CORE0_*BB_INT*_MAP_REG (0x000C)

Register 5.5. INTERRUPT_CORE0_*BT_MAC_INT*_MAP_REG (0x0010)

Register 5.6. INTERRUPT_CORE0_*BT_BB_INT*_MAP_REG (0x0014)

Register 5.7. INTERRUPT_CORE0_*BT_BB_NMI*_MAP_REG (0x0018)

Register 5.8. INTERRUPT_CORE0_*RWBT_IRQ*_MAP_REG (0x001C)

Register 5.9. INTERRUPT_CORE0_*RWBLE_IRQ*_MAP_REG (0x0020)

Register 5.10. INTERRUPT_CORE0_*RWBT_NMI*_MAP_REG (0x0024)

Register 5.11. INTERRUPT_CORE0_*RWBLE_NMI*_MAP_REG (0x0028)

Register 5.12. INTERRUPT_CORE0_*I2C_MST_INT*_MAP_REG (0x002C)

Register 5.13. INTERRUPT_CORE0_*UHCI0_INTR*_MAP_REG (0x0038)

Register 5.14. INTERRUPT_CORE0_*GPIO_INTERRUPT_PRO*_MAP_REG (0x0040)

Register 5.15. INTERRUPT_CORE0_*GPIO_INTERRUPT_PRO_NMI*_MAP_REG (0x0044)

Register 5.16. INTERRUPT_CORE0_*SPI_INTR_1*_MAP_REG (0x0050)

Register 5.17. INTERRUPT_CORE0_*SPI_INTR_2*_MAP_REG (0x0054)

Register 5.18. INTERRUPT_CORE0_*SPI_INTR_3*_MAP_REG (0x0058)

Register 5.19. INTERRUPT_CORE0_*LCD_CAM_INT*_MAP_REG (0x0060)

Register 5.20. INTERRUPT_CORE0_*I2S0_INT*_MAP_REG (0x0064)

Register 5.21. INTERRUPT_CORE0_*I2S1_INT*_MAP_REG (0x0068)

Register 5.22. INTERRUPT_CORE0_*UART_INTR*_MAP_REG (0x006C)

Register 5.23. INTERRUPT_CORE0_*UART1_INTR*_MAP_REG (0x0070)

Register 5.24. INTERRUPT_CORE0_*UART2_INTR*_MAP_REG (0x0074)

Register 5.25. INTERRUPT_CORE0_*SDIO_HOST_INTERRUPT*_MAP_REG (0x0078)

Register 5.26. INTERRUPT_CORE0_*PWM0_INTR*_MAP_REG (0x007C)

Register 5.27. INTERRUPT_CORE0_*PWM1_INTR*_MAP_REG (0x0080)

Register 5.28. INTERRUPT_CORE0_*LEDC_INT*_MAP_REG (0x008C)

Register 5.29. INTERRUPT_CORE0_*EFUSE_INT*_MAP_REG (0x0090)

Register 5.30. INTERRUPT_CORE0_*CAN_INT*_MAP_REG (0x0094)

Register 5.31. INTERRUPT_CORE0_*USB_INTR*_MAP_REG (0x0098)

Register 5.32. INTERRUPT_CORE0_*RTC_CORE_INTR*_MAP_REG (0x009C)

Register 5.68. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_2*_MAP_REG (0x0144)

Register 5.69. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_3*_MAP_REG (0x0148)

Register 5.70. INTERRUPT_CORE0_*ASSIST_DEBUG_INTR*_MAP_REG (0x014C)

Register 5.71. INTERRUPT_CORE0_*DMA_APBPERI_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x0150)

Register 5.72. INTERRUPT_CORE0_*CORE_0_IRAM0_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x0154)

Register 5.73. INTERRUPT_CORE0_*CORE_0_DRAM0_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x0158)

Register 5.74. INTERRUPT_CORE0_*CORE_0_PIF_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x015C)

Register 5.75. INTERRUPT_CORE0_*CORE_0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR*_MAP_REG (0x0160)

Register 5.76. INTERRUPT_CORE0_*CORE_1_IRAM0_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x0164)

Register 5.77. INTERRUPT_CORE0_*CORE_1_DRAM0_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x0168)

Register 5.78. INTERRUPT_CORE0_*CORE_1_PIF_PMS_MONITOR_VIOLATE_INTR*_MAP_REG (0x016C)

Register 5.79. INTERRUPT_CORE0_*CORE_1_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR*_MAP_REG (0x0170)

Register 5.80. INTERRUPT_CORE0_*BACKUP_PMS_VIOLATE_INTR*_MAP_REG (0x0174)

Register 5.81. INTERRUPT_CORE0_*CACHE_CORE0_ACS_INT*_MAP_REG (0x0178)

Register 5.82. INTERRUPT_CORE0_*CACHE_CORE1_ACS_INT*_MAP_REG (0x017C)

Register 5.83. INTERRUPT_CORE0_*USB_DEVICE_INT*_MAP_REG (0x0180)

Register 5.84. INTERRUPT_CORE0_*PERI_BACKUP_INT*_MAP_REG (0x0184)

Register 5.85. INTERRUPT_CORE0_*DMA_EXTMEM_REJECT_INT*_MAP_REG (0x0188)

(reserved) INTERRUPT_CORE0_SOURCE_*Y*_MAP

31	5	4	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		16		Reset

INTERRUPT_CORE0_SOURCE_*Y*_MAP Map interrupt signal of Source_*Y* to one of CPU0 external interrupt, can be configured as 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28, 30 ~ 31. The remaining values are invalid. For Source_*Y*, see Table 5-1. (R/W)

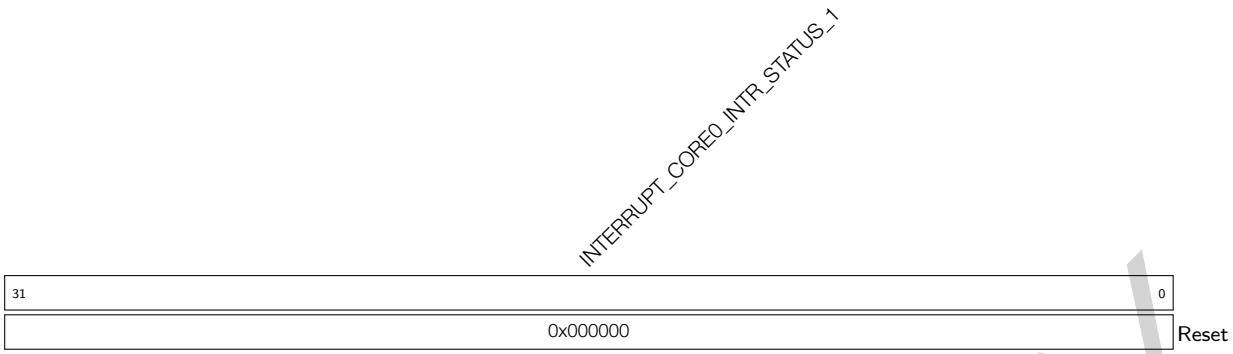
Register 5.86. INTERRUPT_CORE0_INTR_STATUS_0_REG (0x018C)

INTERRUPT_CORE0_INTR_STATUS_0

31	0	
0x000000		Reset

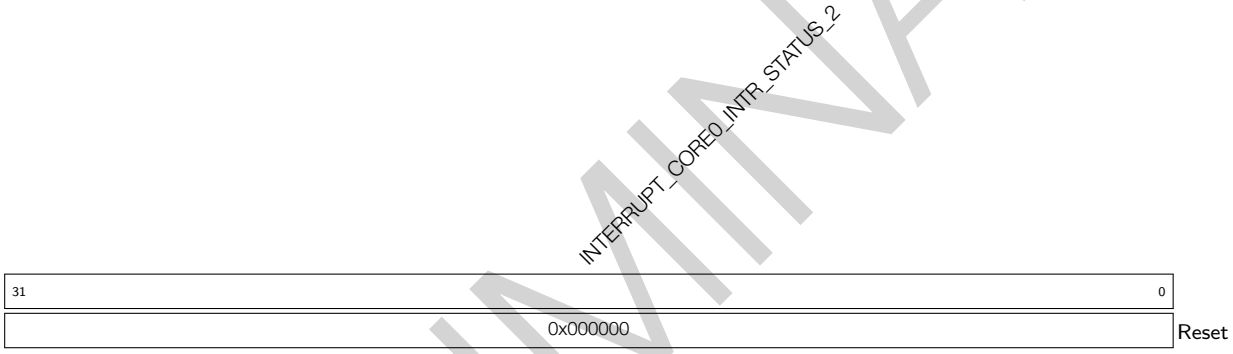
INTERRUPT_CORE0_INTR_STATUS_0 This register stores the status of the first 32 interrupt sources. (RO)

Register 5.87. INTERRUPT_CORE0_INTR_STATUS_1_REG (0x0190)



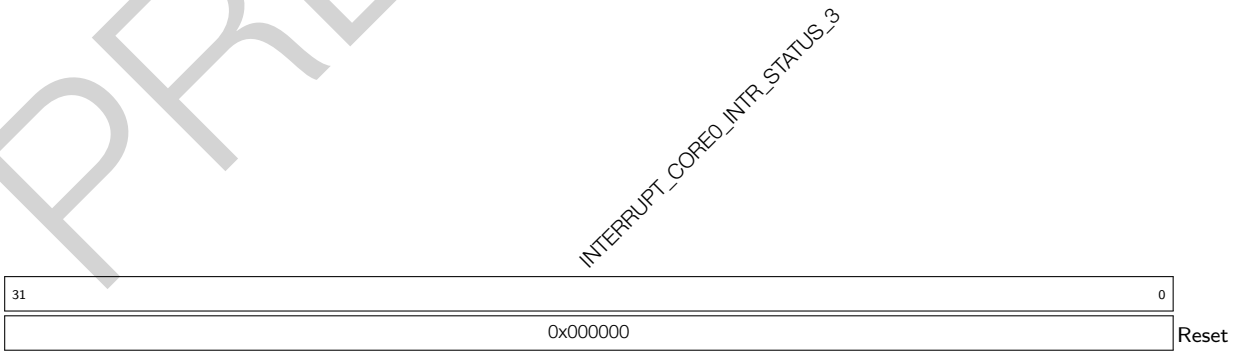
INTERRUPT_CORE0_INTR_STATUS_1 This register stores the status of the second 32 interrupt sources. (RO)

Register 5.88. INTERRUPT_CORE0_INTR_STATUS_2_REG (0x0194)



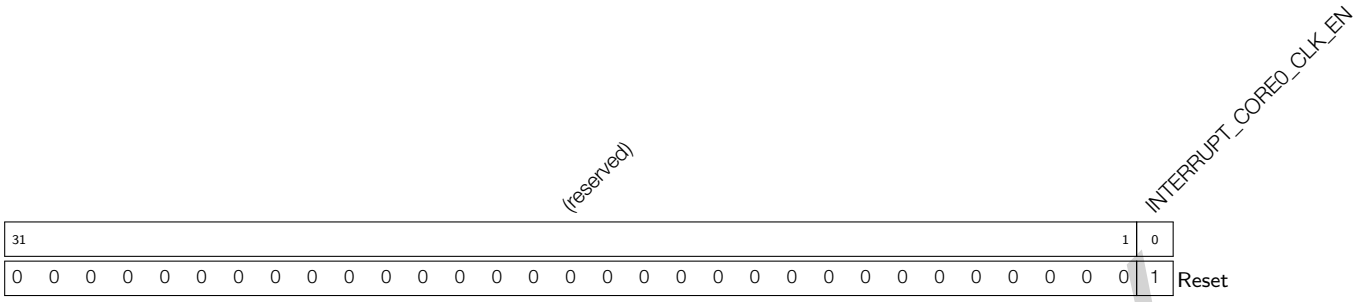
INTERRUPT_CORE0_INTR_STATUS_2 This register stores the status of the third 32 interrupt sources. (RO)

Register 5.89. INTERRUPT_CORE0_INTR_STATUS_3_REG (0x0198)



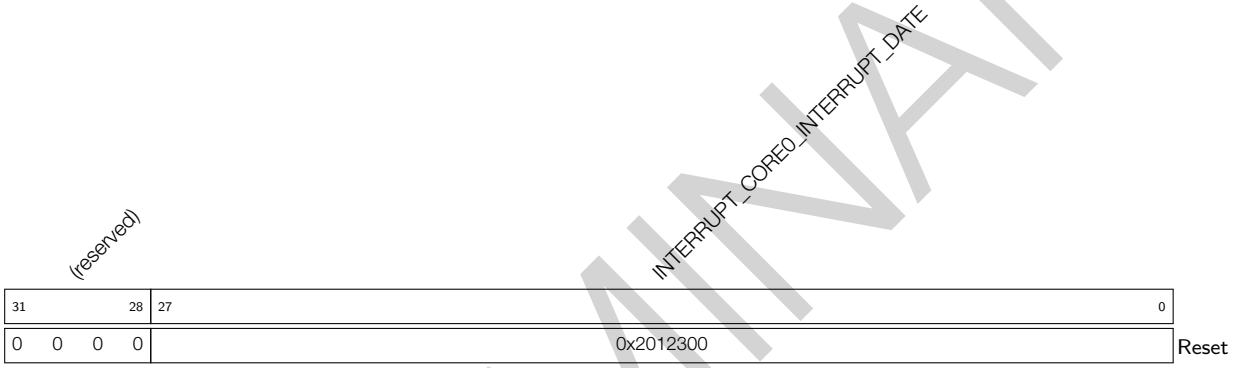
INTERRUPT_CORE0_INTR_STATUS_3 This register stores the status of the last 3 interrupt sources. (RO)

Register 5.90. INTERRUPT_CORE0_CLOCK_GATE_REG (0x019C)



INTERRUPT_CORE0_CLK_EN This register is used to control clock-gating of interrupt matrix. (R/W)

Register 5.91. INTERRUPT_CORE0_DATE_REG (0x07FC)



INTERRUPT_CORE0_INTERRUPT_DATE Version control register (R/W)

5.5.2 CPU1 Interrupt Registers

Register 5.92. INTERRUPT_CORE1_*MAC_INTR*_MAP_REG (0x0800)

Register 5.93. INTERRUPT_CORE1_*MAC_NMI*_MAP_REG (0x0804)

Register 5.94. INTERRUPT_CORE1_*PWR_INTR*_MAP_REG (0x0808)

Register 5.95. INTERRUPT_CORE1_*BB_INT*_MAP_REG (0x080C)

Register 5.96. INTERRUPT_CORE1_*BT_MAC_INT*_MAP_REG (0x0810)

Register 5.97. INTERRUPT_CORE1_*BT_BB_INT*_MAP_REG (0x0814)

Register 5.98. INTERRUPT_CORE1_*BT_BB_NMI*_MAP_REG (0x0818)

Register 5.99. INTERRUPT_CORE1_*RWBT_IRQ*_MAP_REG (0x081C)

Register 5.100. INTERRUPT_CORE1_*RWBLE_IRQ*_MAP_REG (0x0820)

Register 5.101. INTERRUPT_CORE1_*RWBT_NMI*_MAP_REG (0x0824)

Register 5.102. INTERRUPT_CORE1_*RWBLE_NMI*_MAP_REG (0x0828)

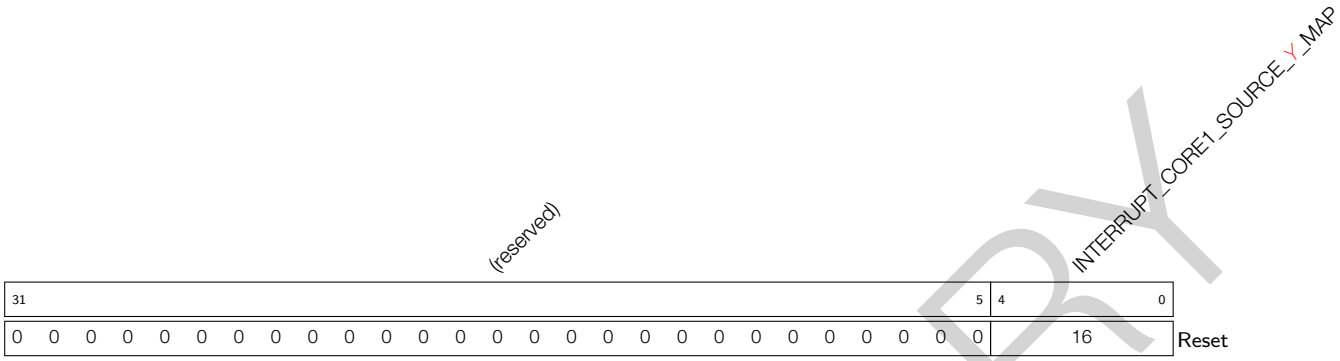
Register 5.103. INTERRUPT_CORE1_*I2C_MST_INT*_MAP_REG (0x082C)

Register 5.173. INTERRUPT_CORE1_*CACHE_CORE1_ACS_INT*_MAP_REG (0x097C)

Register 5.174. INTERRUPT_CORE1_*USB_DEVICE_INT*_MAP_REG (0x0980)

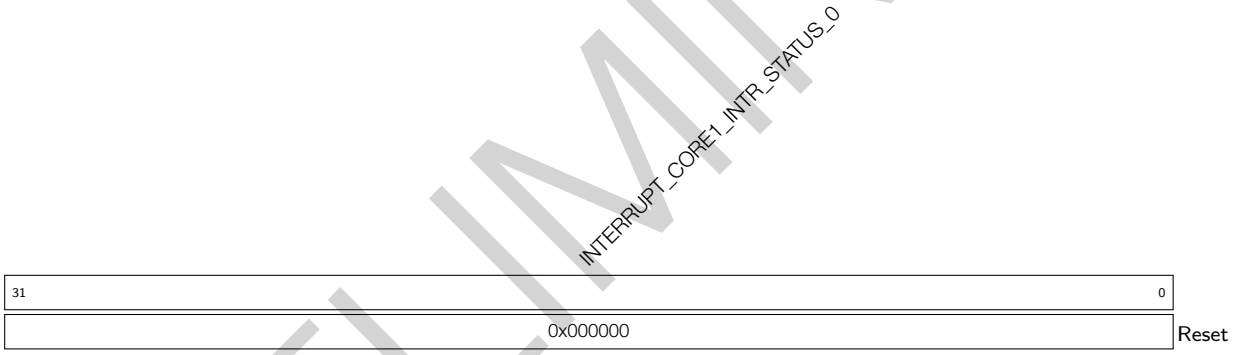
Register 5.175. INTERRUPT_CORE1_*PERI_BACKUP_INT*_MAP_REG (0x0984)

Register 5.176. INTERRUPT_CORE1_DMA_EXTMEM_REJECT_INT_MAP_REG (0x0988)



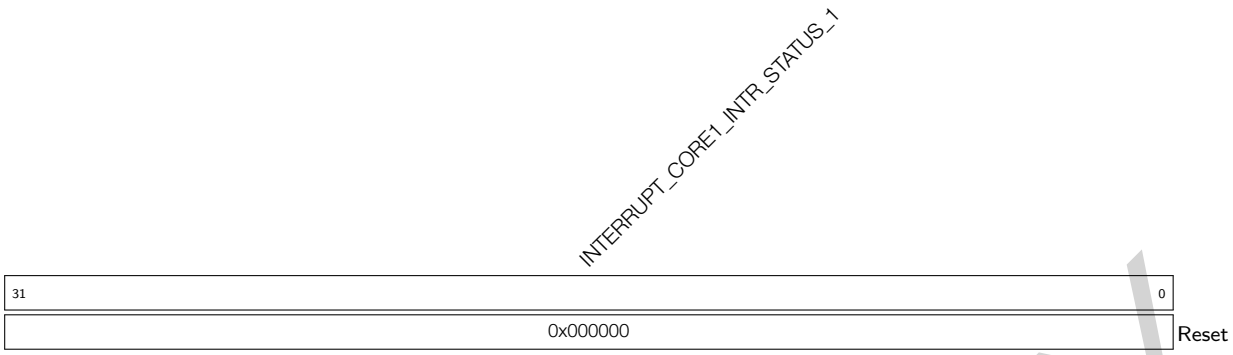
INTERRUPT_CORE1_SOURCE_*Y*_MAP Map interrupt signal of Source_*Y* to one of CPU1 external interrupt, can be configured as 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28, 30 ~ 31. The remaining values are invalid. For Source_*Y*, see Table 5-1. (R/W)

Register 5.177. INTERRUPT_CORE1_INTR_STATUS_0_REG (0x098C)



INTERRUPT_CORE1_INTR_STATUS_0 This register stores the status of the first 32 interrupt sources. (RO)

Register 5.178. INTERRUPT_CORE1_INTR_STATUS_1_REG (0x0990)

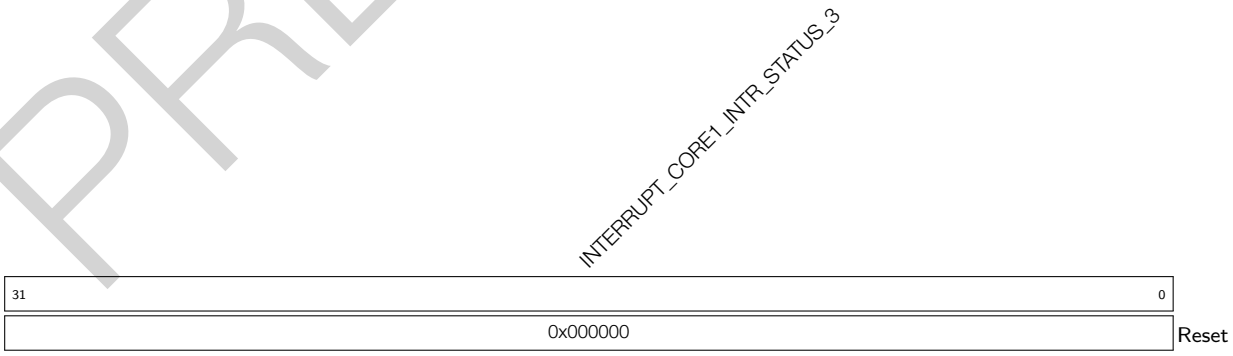


INTERRUPT_CORE1_INTR_STATUS_1 This register stores the status of the second 32 interrupt sources. (RO)

Register 5.179. INTERRUPT_CORE1_INTR_STATUS_2_REG (0x0994)

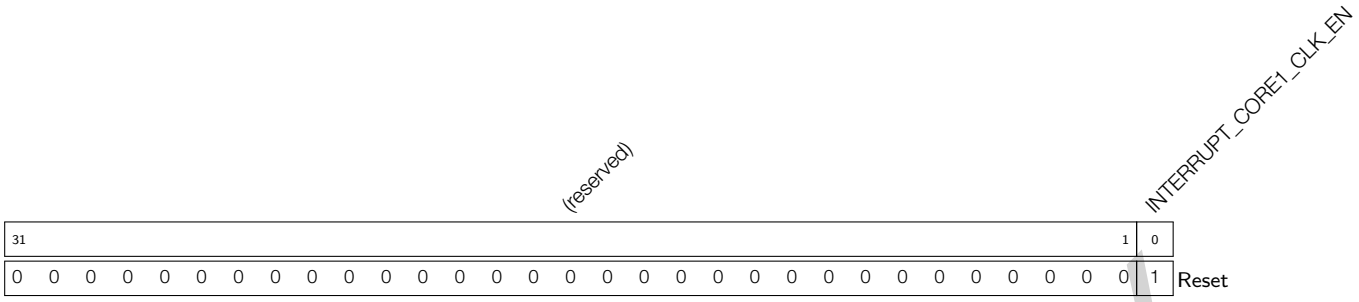
INTERRUPT_CORE1_INTR_STATUS_2 This register stores the status of the third 32 interrupt sources. (RO)

Register 5.180. INTERRUPT_CORE1_INTR_STATUS_3_REG (0x0998)



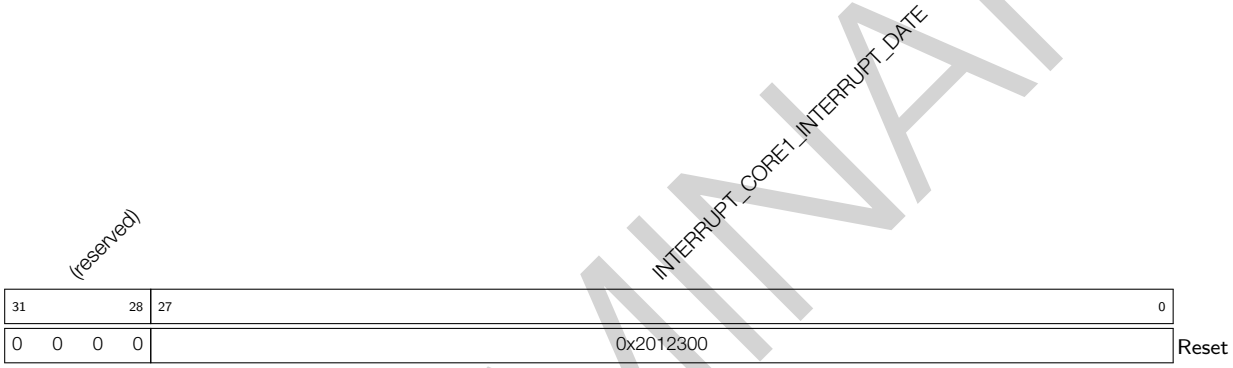
INTERRUPT_CORE1_INTR_STATUS_3 This register stores the status of the last 3 interrupt sources. (RO)

Register 5.181. INTERRUPT_CORE1_CLOCK_GATE_REG (0x099C)



INTERRUPT_CORE1_CLK_EN This register is used to control clock-gating of interrupt matrix. (R/W)

Register 5.182. INTERRUPT_CORE1_DATE_REG (0x0FFC)



INTERRUPT_CORE1_INTERRUPT_DATE Version control register. (R/W)

6 Timer Group (TIMG)

6.1 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 6-1, the ESP32-S3 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 54-bit auto-reload-capable up-down counters.

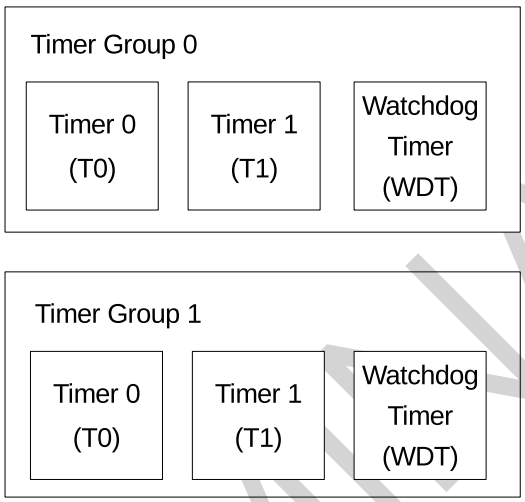


Figure 6-1. Timer Units within Groups

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 7 *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 2 to 65536
- A 54-bit time-base counter programmable to 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 interrupt generation

6.2 Functional Description

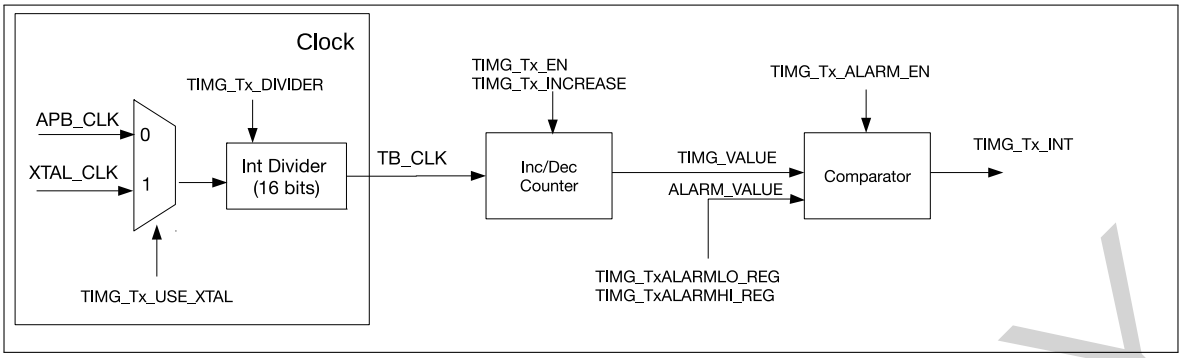


Figure 6-2. Timer Group Architecture

Figure6-2 is a diagram of timer T*x* in a timer group. T*x* contains a clock selector, a 16-bit integer divider as a prescaler, a timer-based counter and a comparator for alarm generation.

6.2.1 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. When the TIMG_T*x*_DIVIDER field is configured as 2 ~ 65536, the divisor of the prescaler would be 2 ~ 65536. Note that programming value 0 to TIMG_T*x*_DIVIDER will result in the divisor being 65536. When the prescaler is set to 1, the actual divisor is 2, so the timer counter value represents the half of real time.

Before you modify the 16-bit prescaler, the timer must be disabled (i.e. TIMG_T*x*_EN should be cleared). Otherwise, the result can be unpredictable.

6.2.2 54-bit Time-base Counter

The 54-bit time-base counters are based on TB_CLK and can be configured to increment or decrement via the TIMG_T*x*_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG_T*x*_EN field, respectively. When enabled, the time-base counter increments or decrements on each cycle of TB_CLK. When disabled, the time-base counter is essentially frozen. Note that the TIMG_T*x*_INCREASE field can be changed while TIMG_T*x*_EN is set and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_T*x*UPDATE_REG, the current value of the 54-bit timer is instantly latched into the TIMG_T*x*LO_REG and TIMG_T*x*HI_REG registers containing the lower 32-bits and higher 22-bits, respectively. 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.

6.2.3 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 6.2.4). The 54-bit alarm value is configured using TIMG_T*x*ALARMLO_REG and TIMG_T*x*ALARMHI_REG, which represent the lower 32-bits and higher 22-bits of the alarm value, respectively. However, the configured alarm

value is ineffective until the alarm is enabled by setting the TIMG_T*x*_ALARM_EN field. To avoid alarm being enabled 'too late' (i.e. the timer value has already passed the alarm value when the alarm is enabled), the hardware will trigger the alarm immediately if the current timer value is higher than the alarm value (within a defined range) when the up-down counter increments, or lower than the alarm value (within a defined range) of when the up-down counter decrements. Table 6-1 and Table 6-2 show the relationship between the current value of the timer, the alarm value, and when an alarm is triggered.The current time value and the alarm value are defined as follows:

- TIMG_VALUE = {TIMG_T*x*HI_REG, TIMG_T*x*LO_REG}
- ALARM_VALUE = {TIMG_T*x*ALARMHI_REG, TIMG_T*x*ALARMLO_REG}

Table 6-1. Alarm Generation When Up-Down Counter Increments

Scenario	Range	Alarm
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	Triggered
2	$0 <$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE up to ALARM_VALUE
3	$0 \leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	Triggered
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting up from 0 after reaching the timer's maximum value and counts TIMG_VALUE up to ALARM_VALUE

Table 6-2. Alarm Generation When Up-Down Counter Decrements

Scenario	Range	Alarm
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	Triggered
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	Triggered when the up-down counter counts TIMG_VALUE down to ALARM_VALUE
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	Triggered
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	Triggered when the up-down counter restarts counting down from the timer's maximum value after reaching the minimum value and counts TIMG_VALUE down to ALARM_VALUE

When an alarm occurs, the TIMG_T*x*_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_T*x*_ALARM_EN is set next time.

6.2.4 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 fields that correspond to the lower 32-bits and higher 22-bits of the timer's new value, respectively. However, writing a reload value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI will not cause the timer's current value to change. Instead, the reload value is ignored by the timer until a reload event occurs. A reload event can be triggered either by a software instant reload or an auto-reload at alarm.

A software instant reload is triggered by the CPU writing any value to TIMG_TxLOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG_Tx_EN is set, the timer will continue incrementing or decrementing from the new value. If TIMG_Tx_EN is cleared, the timer will remain frozen at the new value until counting is re-enabled.

An auto-reload at alarm will cause a timer reload when an alarm occurs, thus allowing the timer to continue incrementing or decrementing from the reload value. This is generally useful for resetting the timer's value when using periodic alarms. To enable auto-reload at alarm, the TIMG_Tx_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment or decrement past the alarm value after an alarm.

6.2.5 SLOW_CLK Frequency Calculation

Via XTAL_CLK, a timer could calculate the frequency of clock sources for SLOW_CLK (i.e. RTC_CLK, RTC20M_D256_CLK, and XTAL32K_CLK) as follows:

1. Start periodic or one-shot frequency calculation;
2. Once receiving the signal to start calculation, the counter of XTAL_CLK and the counter of SLOW_CLK begin to work at the same time. When the counter of SLOW_CLK counts to C0, the two counters stop counting simultaneously;
3. Assume the value of XTAL_CLK's counter is C1, and the frequency of SLOW_CLK would be calculated as: $f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

6.2.6 Interrupts

Each timer has its own interrupt line that can be routed to the CPU, and thus each timer group has a total of three interrupt lines. Timers generate level interrupts that must be explicitly cleared by the CPU on each triggering.

Interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupts will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. To enable a timer's interrupt, the TIMG_Tx_INT_ENA bit should be set.

The interrupts of each timer group are governed by a set of registers. Each timer within the group has a corresponding bit in each of these registers:

- TIMG_Tx_INT_RAW : An alarm event sets it to 1. The bit will remain set until the timer's corresponding bit in TIMG_Tx_INT_CLR is written.
- TIMG_WDT_INT_RAW : A stage time out will set the timer's bit to 1. The bit will remain set until the timer's corresponding bit in TIMG_WDT_INT_CLR is written.
- TIMG_Tx_INT_ST : Reflects the status of each timer's interrupt and is generated by masking the bits of TIMG_Tx_INT_RAW with TIMG_Tx_INT_ENA.
- TIMG_WDT_INT_ST : Reflects the status of each watchdog timer's interrupt and is generated by masking the bits of TIMG_WDT_INT_RAW with TIMG_WDT_INT_ENA.
- TIMG_Tx_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.
- TIMG_WDT_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The watchdog timer's corresponding bit in TIMG_WDT_INT_RAW and TIMG_WDT_INT_ST will be cleared as a result. Note that a watchdog timer's interrupt must be cleared before the next interrupt occurs.

6.3 Configuration and Usage

6.3.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting or clearing TIMG_T*x*_USE_XTAL field.
   - Configure the 16-bit prescaler by setting TIMG_T*x*_DIVIDER.
   - Configure the timer direction by setting or clearing TIMG_T*x*_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI, then reloading it into the timer by writing any value to TIMG_T*x*LOAD_REG.
2. Start the timer by setting TIMG_T*x*_EN.
3. Get the timer's current value.
   - Write any value to TIMG_T*x*UPDATE_REG to latch the timer's current value.
   - Read the latched timer value from TIMG_T*x*LO_REG and TIMG_T*x*HI_REG.

6.3.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 6.3.1.
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*_INT_ENA.
3. Disable auto reload by clearing TIMG_T*x*_AUTORELOAD.
4. Start the alarm by setting TIMG_T*x*_ALARM_EN.
5. Handle the alarm interrupt.
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

6.3.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 in Section 6.3.1.
2. Configure the alarm following step 2 in Section 6.3.2.
3. Enable auto reload by setting TIMG_T*x*_AUTORELOAD and configure the reload value via TIMG_T*x*_LOAD_LO and TIMG_T*x*_LOAD_HI.

4. Start the alarm by setting TIMG_T*x*_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e. different alarm interval per iteration), then TIMG_T*x*ALARMLO_REG, TIMG_T*x*ALARMHI_REG, TIMG_T*x*_LOAD_LO, and TIMG_T*x*_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_T*x*_ALARM_EN.
6. Stop the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T*x*_INT_CLR.
   - Disable the timer by clearing TIMG_T*x*_EN.

6.3.4 SLOW_CLK Frequency Calculation

1. One-shot frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of SLOW_CLK) via TIMG_RTC_CALI_CLK_SEL, and configure the time of calculation via TIMG_RTC_CALI_MAX.
   - Select one-shot frequency calculation by clearing TIMG_RTC_CALI_START_CYCLING, and enable the two counters via TIMG_RTC_CALI_START.
   - Once TIMG_RTC_CALI_RDY becomes 1, read TIMG_RTC_CALI_VALUE to get the value of XTAL_CLK's counter, and calculate the frequency of SLOW_CLK.
2. Periodic frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of SLOW_CLK) via TIMG_RTC_CALI_CLK_SEL, and configure the time of calculation via TIMG_RTC_CALI_MAX.
   - Select periodic frequency calculation by enabling TIMG_RTC_CALI_START_CYCLING.
   - When TIMG_RTC_CALI_CYCLING_DATA_VLD is 1, TIMG_RTC_CALI_VALUE is valid.
3. Timeout
   If the counter of SLOW_CLK cannot finish counting in TIMG_RTC_CALI_TIMEOUT_RST_CNT cycles, TIMG_RTC_CALI_TIMEOUT will be set to indicate a timeout.

6.4 Register Summary

The addresses in this section are relative to Timer Group base address provided in Table 1-4 in Chapter 1 *System and Memory*.

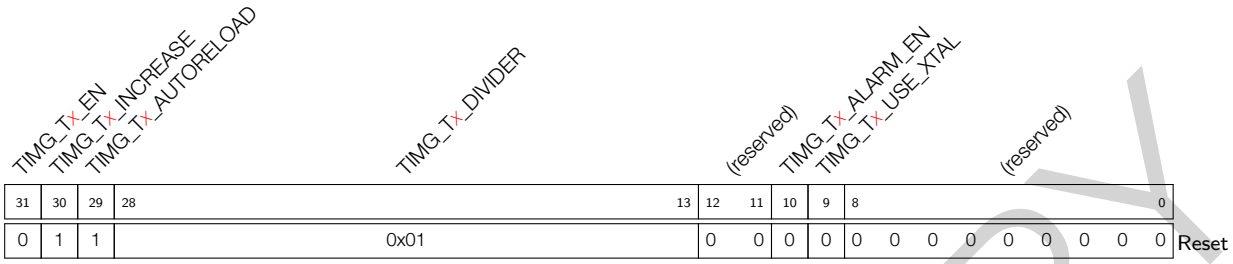
Name	Description	Address	Access
Timer 0 configuration and control registers			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	varies
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 22 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMG_T0LO_REG or TIMG_T0HI_REG	0x000C	R/W/SC
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x0014	R/W
TIMG_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x0018	R/W
TIMG_T0LOADHI_REG	Timer 0 reload value, high 22 bits	0x001C	R/W
TIMG_T0LOAD_REG	Write to reload timer from TIMG_T0LOADLO_REG or TIMG_T0LOADHI_REG	0x0020	WT
Timer 1 configuration and control registers			
TIMG_T1CONFIG_REG	Timer 1 configuration register	0x0024	varies
TIMG_T1LO_REG	Timer 1 current value, low 32 bits	0x0028	RO
TIMG_T1HI_REG	Timer 1 current value, high 22 bits	0x002C	RO
TIMG_T1UPDATE_REG	Write to copy current timer value to TIMG_T1LO_REG or TIMG_T1HI_REG	0x0030	R/W/SC
TIMG_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x0034	R/W
TIMG_T1ALARMHI_REG	Timer 1 alarm value, high bits	0x0038	R/W
TIMG_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x003C	R/W
TIMG_T1LOADHI_REG	Timer 1 reload value, high 22 bits	0x0040	R/W
TIMG_T1LOAD_REG	Write to reload timer from TIMG_T1LOADLO_REG or TIMG_T1LOADHI_REG	0x0044	WT
Configuration and control registers 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	WT
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
Configuration and control registers for RTC frequency calculation			
TIMG_RTCCALICFG_REG	RTC frequency calculation configuration register 0	0x0068	varies

Name	Description	Address	Access
TIMG_RTCCALICFG1_REG	RTC frequency calculation configuration register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC frequency calculation calibration register 2	0x0080	varies
Interrupt registers			
TIMG_INT_ENA_TIMERS_REG	Interrupt enable bits	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	Raw interrupt status	0x0074	R/WTC/SS
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x007C	WT
Version register			
TIMG_NTIMERS_DATE_REG	Timer version control register	0x00F8	R/W
Timer group configuration registers			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

6.5 Registers

The addresses in this section are relative to Timer Group base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 6.1. TIMG_TxCONFIG_REG (x: 0-1) (0x0000+0x24*x)



TIMG_Tx_USE_XTAL 0: Use APB_CLK as the source clock of timer group; 1: Use XTAL_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/SC)

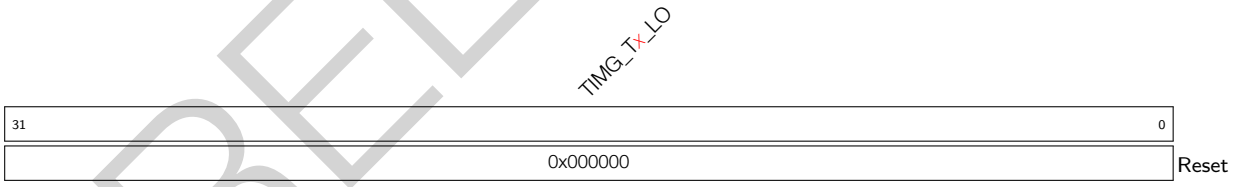
TIMG_Tx_DIVIDER Timer *x* clock (Tx_clk) prescaler value. (R/W)

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)

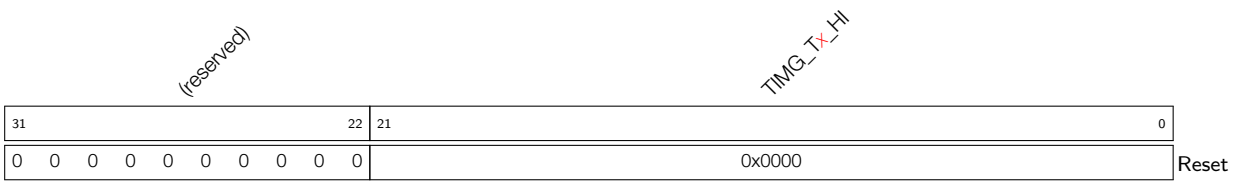
TIMG_Tx_EN When set, the timer *x* time-base counter is enabled. (R/W)

Register 6.2. TIMG_TxLO_REG (x: 0-1) (0x0004+0x24*x)



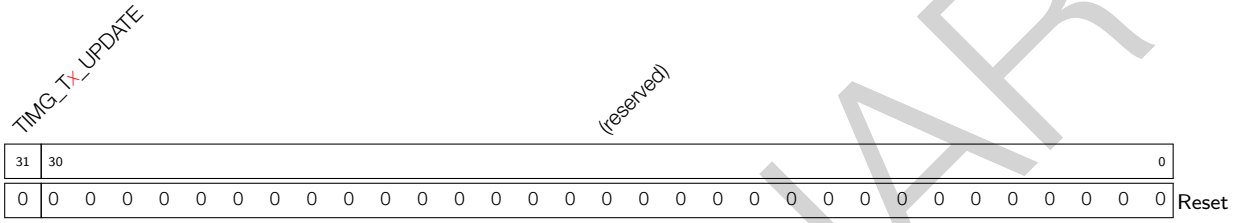
TIMG_Tx_LO After writing to TIMG_TxUPDATE_REG, the low 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 6.3. TIMG_TxHI_REG (x: 0-1) (0x0008+0x24*x)



TIMG_Tx_HI After writing to TIMG_TxUPDATE_REG, the high 22 bits of the time-base counter of timer *x* can be read here. (RO)

Register 6.4. TIMG_TxUPDATE_REG (x: 0-1) (0x000C+0x24*x)

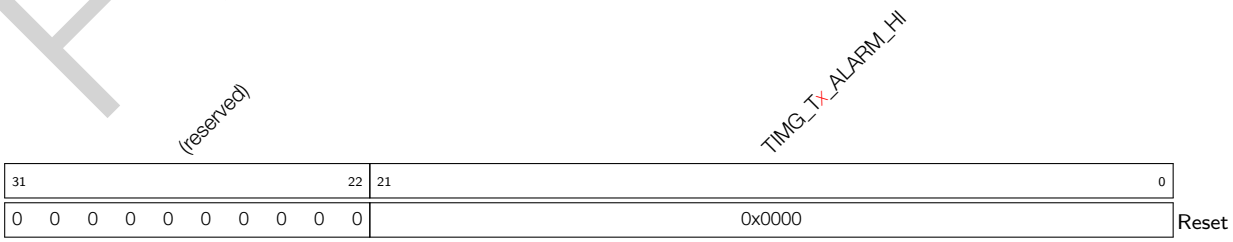


TIMG_Tx_UPDATE After writing 0 or 1 to TIMG_TxUPDATE_REG, the counter value is latched. (R/W/SC)

Register 6.5. TIMG_TxALARMLO_REG (x: 0-1) (0x0010+0x24*x)

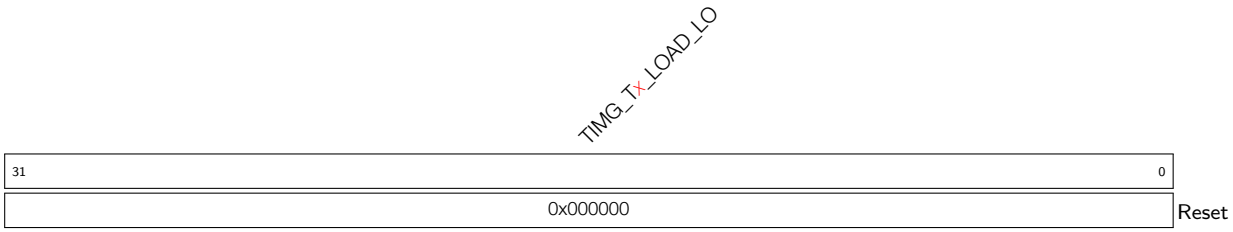
TIMG_Tx_ALARM_LO Timer *x* alarm trigger time-base counter value, low 32 bits. (R/W)

Register 6.6. TIMG_TxALARMHI_REG (x: 0-1) (0x0014+0x24*x)



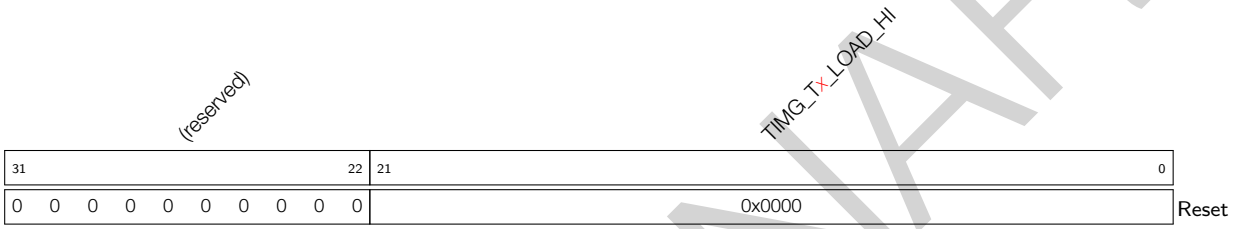
TIMG_Tx_ALARM_HI Timer *x* alarm trigger time-base counter value, high 22 bits. (R/W)

Register 6.7. TIMG_TxLOADLO_REG (x: 0-1) (0x0018+0x24*x)



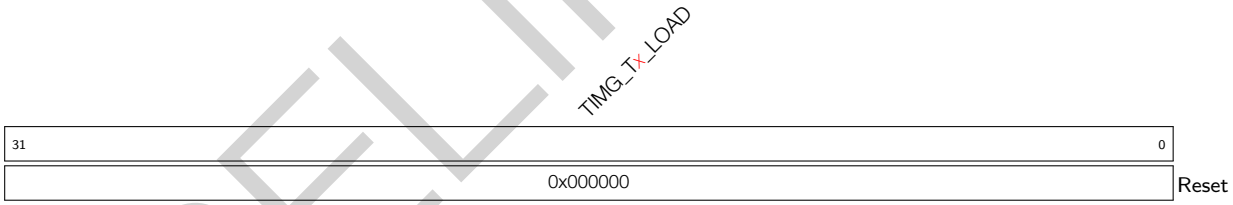
TIMG_Tx_LOAD_LO Low 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 6.8. TIMG_TxLOADHI_REG (x: 0-1) (0x001C+0x24*x)



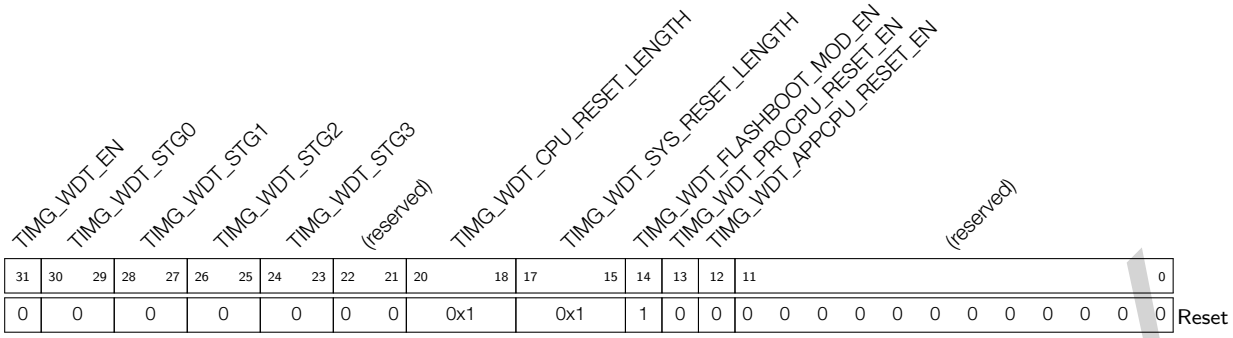
TIMG_Tx_LOAD_HI High 22 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 6.9. TIMG_TxLOAD_REG (x: 0-1) (0x0020+0x24*x)



TIMG_Tx_LOAD Write any value to trigger a timer *x* time-base counter reload. (WT)

Register 6.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN Reserved. (R/W)

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 μs; 7: 3.2 μs. (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 μs; 7: 3.2 μs. (R/W)

TIMG_WDT_STG3 Stage 3 configuration. 0: off; 1: interrupt; 2: reset CPU; 3: reset system. (R/W)

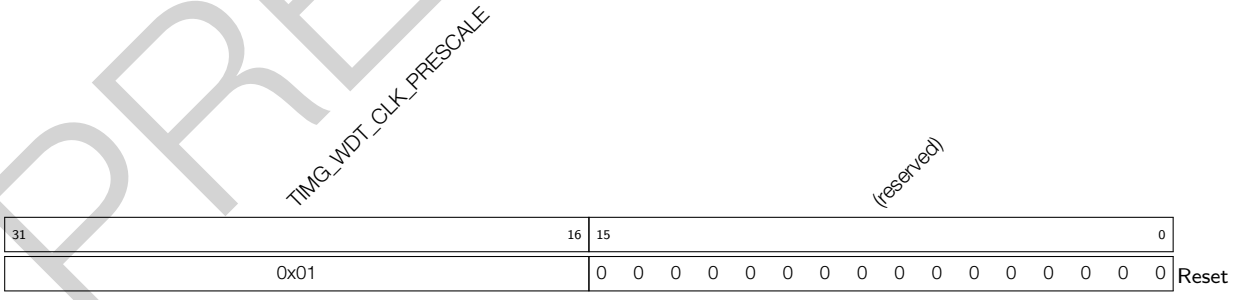
TIMG_WDT_STG2 Stage 2 configuration. 0: off; 1: interrupt; 2: reset CPU; 3: reset system. (R/W)

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)

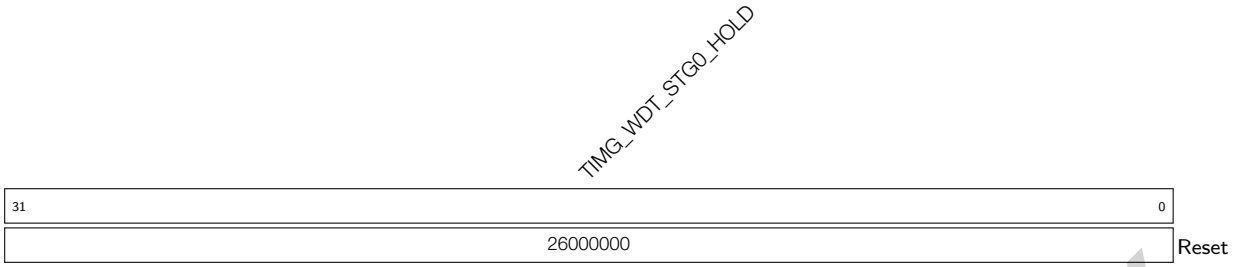
TIMG_WDT_EN When set, MWDT is enabled. (R/W)

Register 6.11. TIMG_WDTCONFIG1_REG (0x004C)



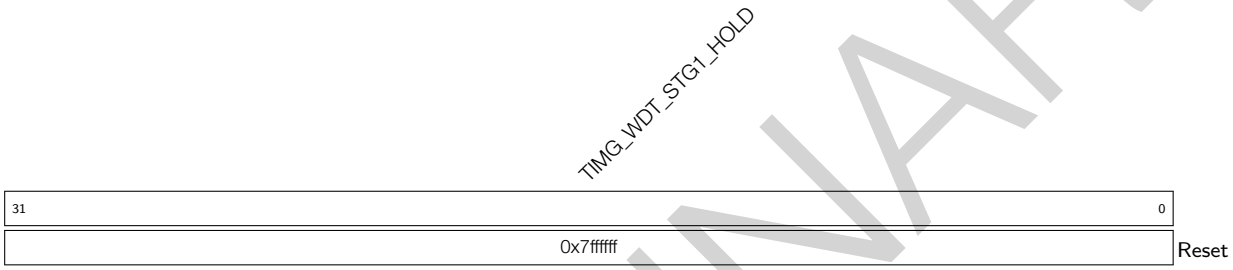
TIMG_WDT_CLK_PRESCALE MWDT clock prescaler value. MWDT clock period = 12.5 ns * TIMG_WDT_CLK_PRESCALE. (R/W)

Register 6.12. TIMG_WDTCONFIG2_REG (0x0050)



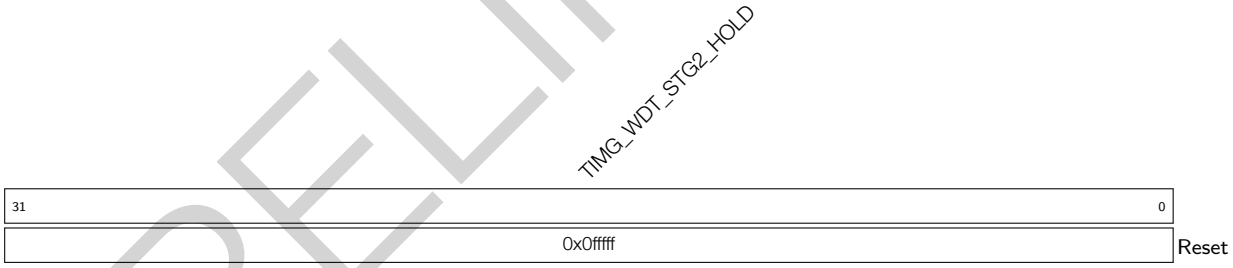
TIMG_WDT_STG0_HOLD Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 6.13. TIMG_WDTCONFIG3_REG (0x0054)



TIMG_WDT_STG1_HOLD Stage 1 timeout value, in MWDT clock cycles. (R/W)

Register 6.14. TIMG_WDTCONFIG4_REG (0x0058)

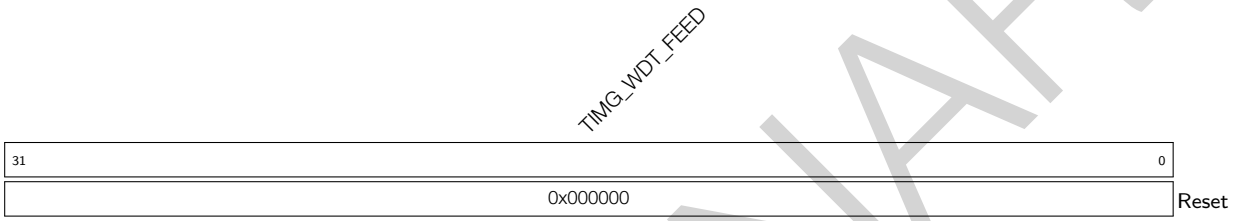


TIMG_WDT_STG2_HOLD Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 6.15. TIMG_WDTCONFIG5_REG (0x005C)

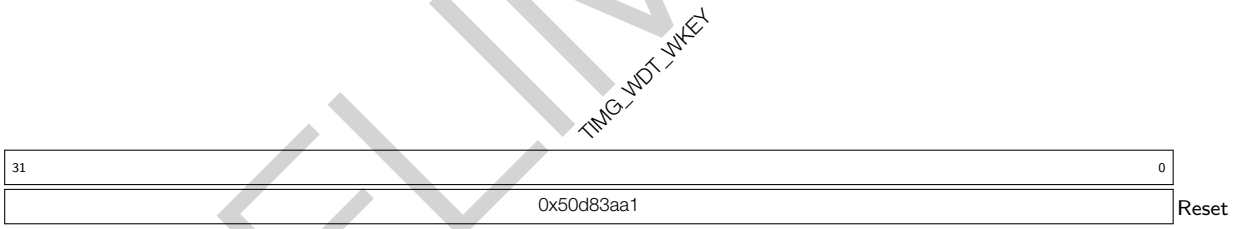
TIMG_WDT_STG3_HOLD Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 6.16. TIMG_WDTFEED_REG (0x0060)



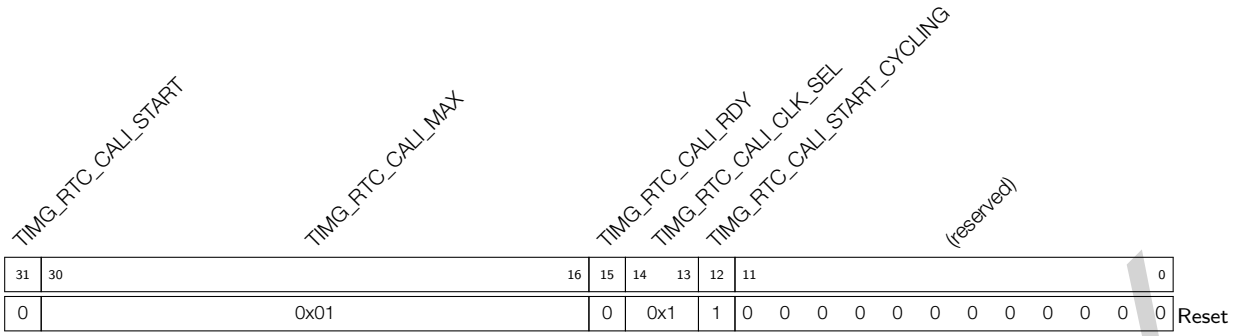
TIMG_WDT_FEED Write any value to feed the MWDT. (WT)

Register 6.17. 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 6.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING Enables periodic frequency calculation. (R/W)

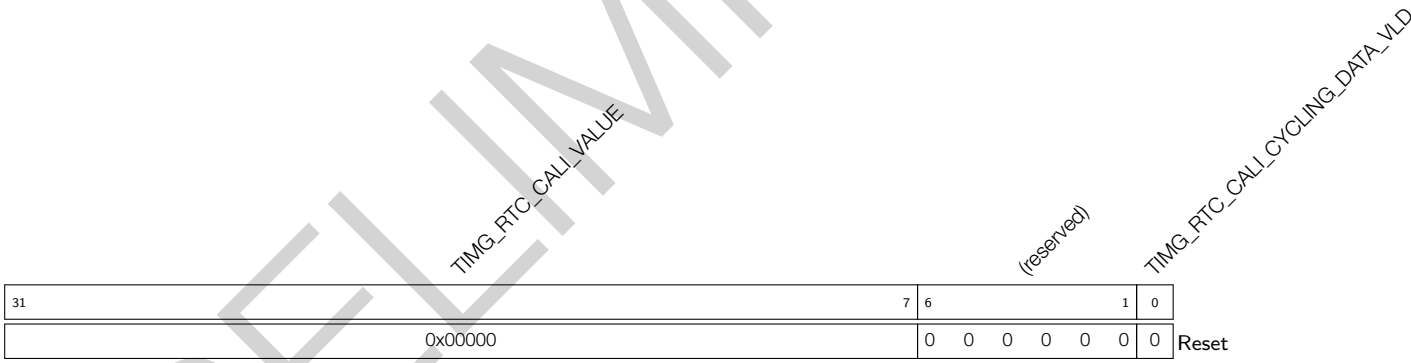
TIMG_RTC_CALI_CLK_SEL Used to select the clock to be calibrated. 0: RTC_CLK; 1: RTC20M_D256_CLK; 2: XTAL32K_CLK. (R/W)

TIMG_RTC_CALI_RDY Marks the completion of frequency calculation. (RO)

TIMG_RTC_CALI_MAX Configures the time of frequency calculation. (R/W)

TIMG_RTC_CALI_START Enables one-shot frequency calculation. (R/W)

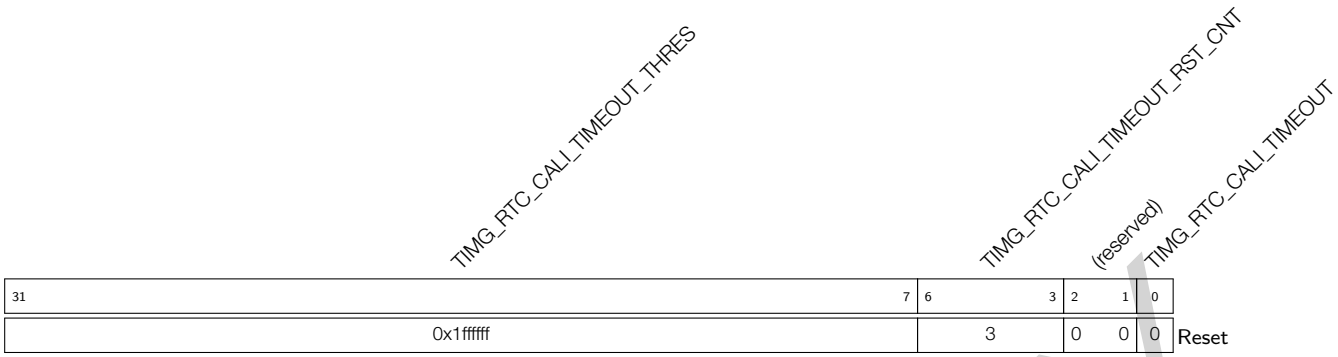
Register 6.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD Marks the completion of periodic frequency calculation. (RO)

TIMG_RTC_CALI_VALUE Frequency calculation result. (RO)

Register 6.20. TIMG_RTCCALICFG2_REG (0x0080)

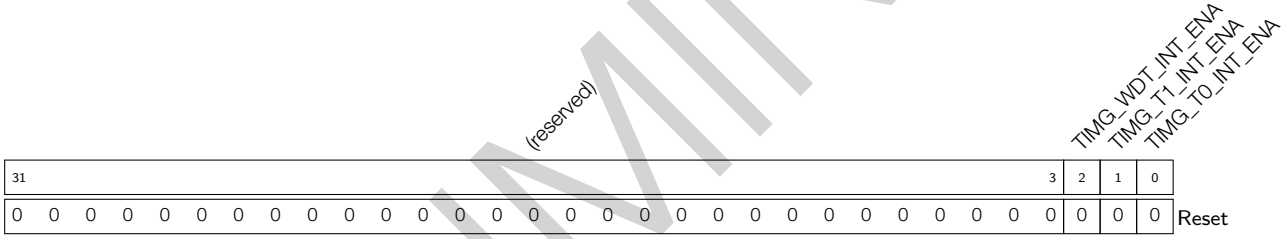


TIMG_RTC_CALI_TIMEOUT Indicates frequency calculation timeout. (RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT Cycles to reset frequency calculation timeout. (R/W)

TIMG_RTC_CALI_TIMEOUT_THRES Threshold value for the frequency calculation timer. If the timer's value exceeds this threshold, a timeout is triggered. (R/W)

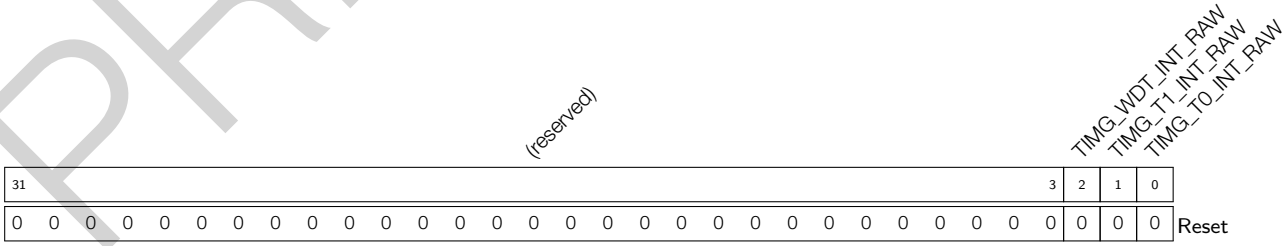
Register 6.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



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)

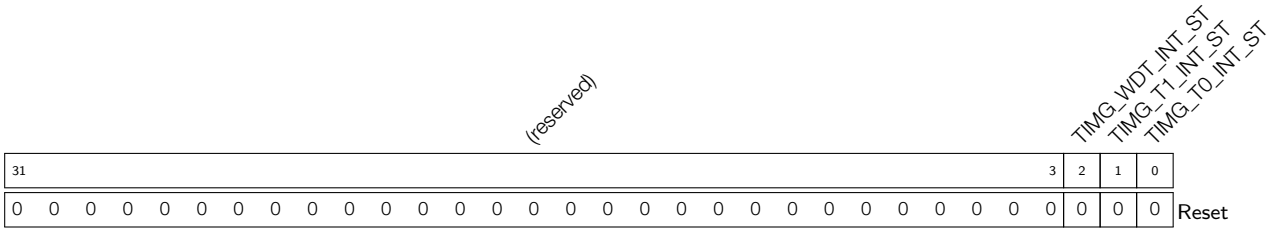
Register 6.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T*x*_INT_RAW The raw interrupt status bit for the TIMG_T*x*_INT interrupt. (R/WTC/SS)

TIMG_WDT_INT_RAW The raw interrupt status bit for the TIMG_WDT_INT interrupt. (R/WTC/SS)

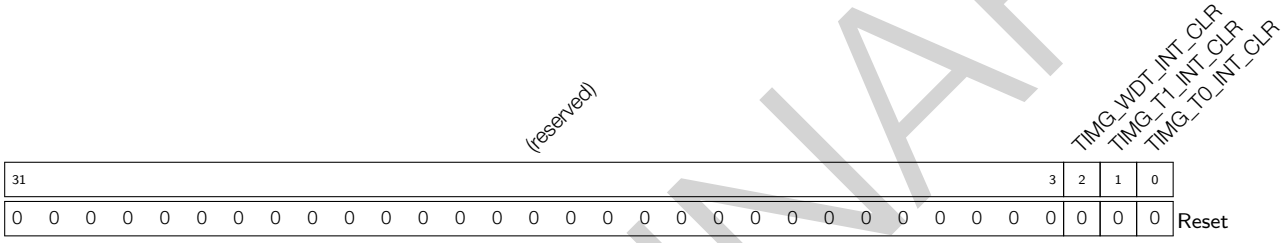
Register 6.23. TIMG_INT_ST_TIMERS_REG (0x0078)



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)

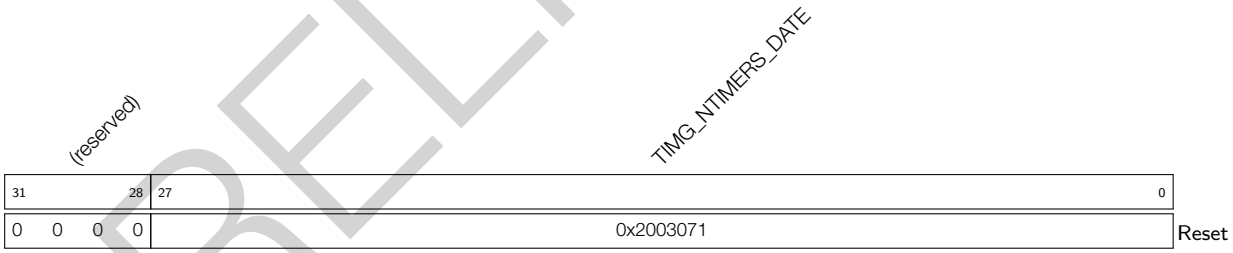
Register 6.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T*x*_INT_CLR Set this bit to clear the TIMG_T*x*_INT interrupt. (WT)

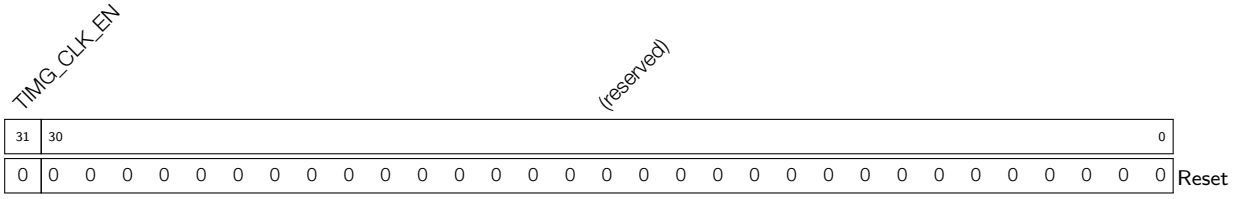
TIMG_WDT_INT_CLR Set this bit to clear the TIMG_WDT_INT interrupt. (WT)

Register 6.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMERS_DATE Timer version control register. (R/W)

Register 6.26. TIMG_REGCLK_REG (0x00FC)



TIMG_CLK_EN Register clock gate signal. 0: The clock used by software to read and write registers is on only when there is software operation. 1: The clock used by software to read and write registers is always on. (R/W)

7 Watchdog Timers

7.1 Overview

Watchdog timers are hardware timers used to detect and recover from malfunctions. They must be periodically fed (reset) to prevent a timeout. A system/software that is behaving unexpectedly (e.g. is stuck in a software loop or in overdue events) will fail to feed the watchdog thus trigger a watchdog timeout. Therefore, watchdog timers are useful for detecting and handling erroneous system/software behavior.

As shown in Figure 7-1, ESP32-S3 contains three digital watchdog timers: one in each of the two timer groups in Chapter 6 *Timer Group (TIMG)*(called Main System Watchdog Timers, or MWDT) and one in the RTC Module (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages and each stage can be programmed to take one action upon expiry, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, CPU reset, and core reset, while RWDT supports four timeout actions: interrupt, CPU reset, core reset, and system reset (see details in Section 7.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

During the flash boot process, RWDT and the first MWDT in timergroup 0 are enabled automatically in order to detect and recover from booting errors.

ESP32-S3 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

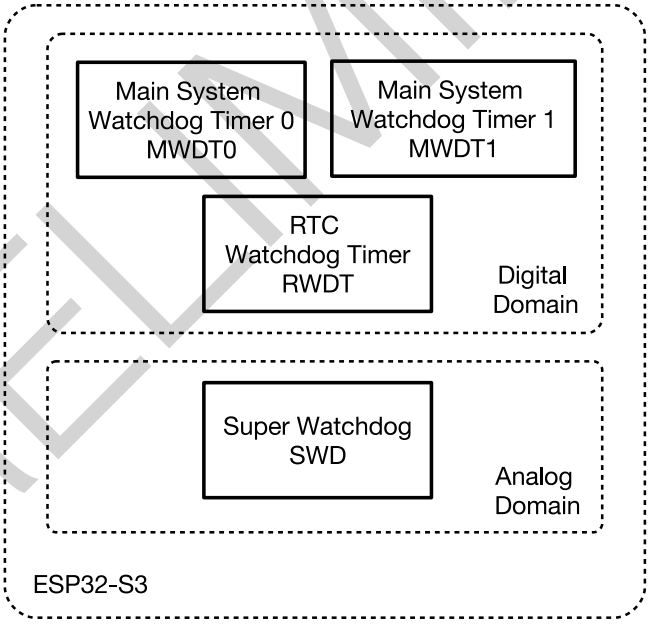


Figure 7-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer's, their register descriptions are provided in Chapter 6 *Timer Group (TIMG)* and Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*.

7.2 Digital Watchdog Timers

7.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a programmable timeout value. Each stage can be configured and enabled/disabled separately
- Three timeout actions (interrupt, CPU reset, or core reset) for MWDT and four timeout actions (interrupt, CPU reset, core reset, or system reset) for RWDT 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.

7.2.2 Functional Description

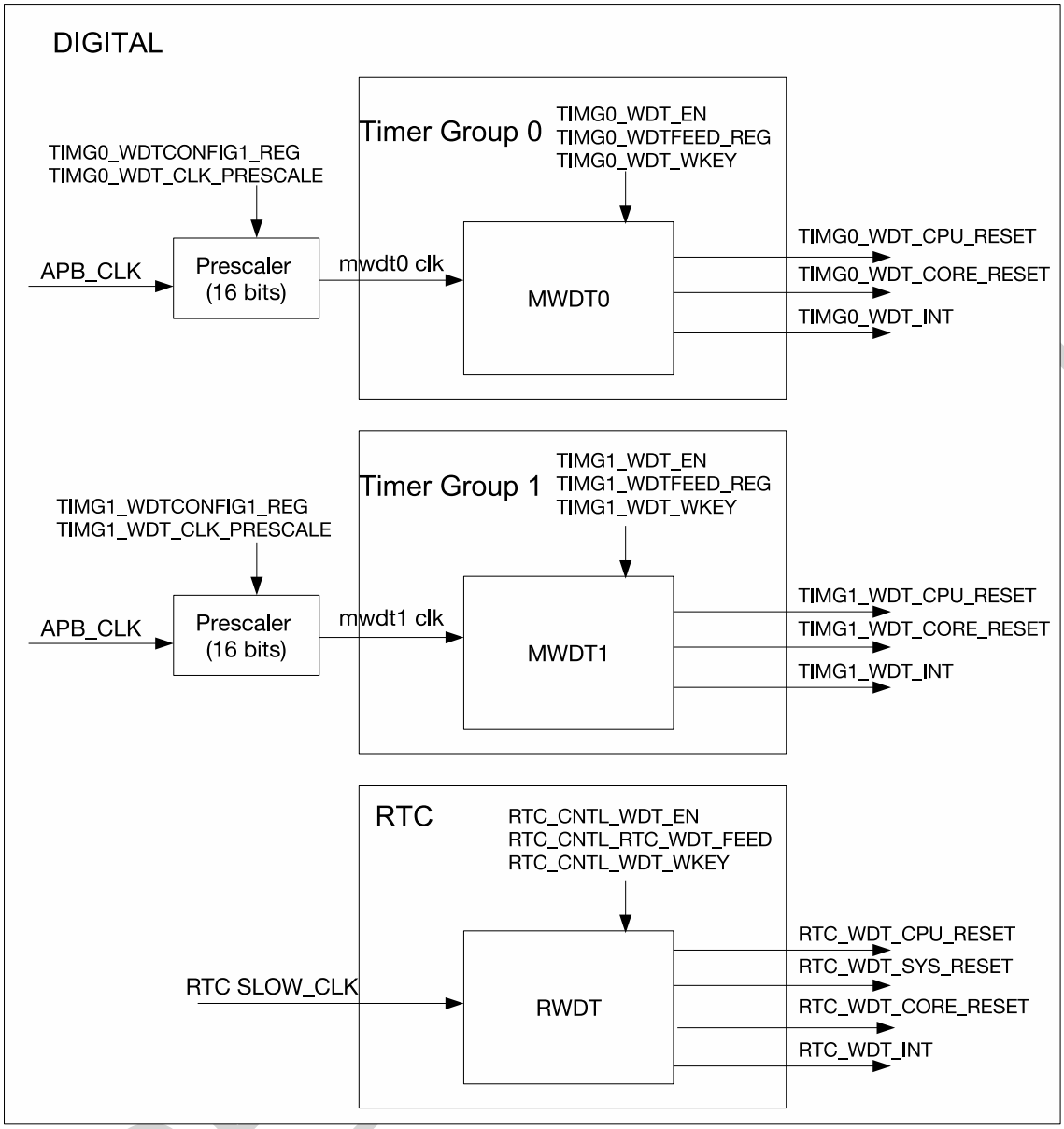


Figure 7-2. Watchdog Timers in ESP32-S3

Figure 7-2 shows the three watchdog timers in ESP32-S3 digital systems.

7.2.2.1 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. In contrast, the clock source of RWDT is derived directly from an RTC slow clock (the RTC slow clock source shown in Chapter 3 *Reset and Clock*). The 16-bit prescaler for MWDTs is configured via the TIMG_WDT_CLK_PRESCALE 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.

7.2.2.2 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 using RTC_CNTL_WDT_STG*j*_HOLD field (where *j* ranges from 0 to 3).

Please note that the timeout value of stage 0 for RWDT (T_{hold0}) is determined by the combination of the EFUSE_WDT_DELAY_SEL field of eFuse register EFUSE_RD_REPEAT_DATA1_REG and RTC_CNTL_WDT_STG0_HOLD. The relationship is as follows:

$$T_{hold0} = RTC\_CNTL\_WDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

where $<<$ is a left-shift operator.

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.
- CPU reset – Reset a CPU core
  When the stage expires, the CPU core will be reset.
- Core reset – Reset the main system
  When the stage expires, the main system (which includes MWDTs, CPU, and all peripherals) will be reset. The power management unit and RTC peripheral will not be reset.
- System reset – Reset the main system, power management unit and RTC peripheral
  When the stage expires the main system, power management unit and RTC peripheral (see details in Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*) will all 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_CNTL_WDTCONFIG0_REG.

7.2.2.3 Write Protection

Watchdog timers are critical to detecting and handling erroneous system/software behavior, thus should not be disabled easily (e.g. due to a misplaced register write). Therefore, 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 field for each timer (TIMG_WDT_WKEY for MWDT, RTC_CNTL_WDT_WKEY for RWDT). The value 0x50D83AA1 must be written to the watchdog timer's write-key field before any other register of the same watchdog timer can be changed. Any attempts to write to a watchdog timer's registers (other than the write-key field itself) whilst the write-key field's value is not 0x50D83AA1 will be ignored. The recommended procedure for accessing a watchdog timer is as follows:

1. Disable the write protection by writing the value 0x50D83AA1 to the timer's write-key field.
2. Make the required modification of the watchdog such as feeding or changing its configuration.
3. Re-enable write protection by writing any value other than 0x50D83AA1 to the timer's write-key field.

7.2.2.4 Flash Boot Protection

During flash booting process, MWDT in timer group 0 (see Figure 6-1 *Timer Units within Groups*), 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.

7.3 Super Watchdog

Super watchdog (SWD) is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required. SWD contains a watchdog circuit that needs to be fed for at least once during its timeout period, which is slightly less than one second. About 100 ms before watchdog timeout, it will also send out a WD_INTR signal as a request to remind the system to feed the watchdog.

If the system doesn't respond to SWD feed request and watchdog finally times out, SWD will generate a system level signal SWD_RSTB to reset whole digital circuits on the chip.

7.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD timeout period is close to expiring
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

7.3.2 Super Watchdog Controller

7.3.2.1 Structure

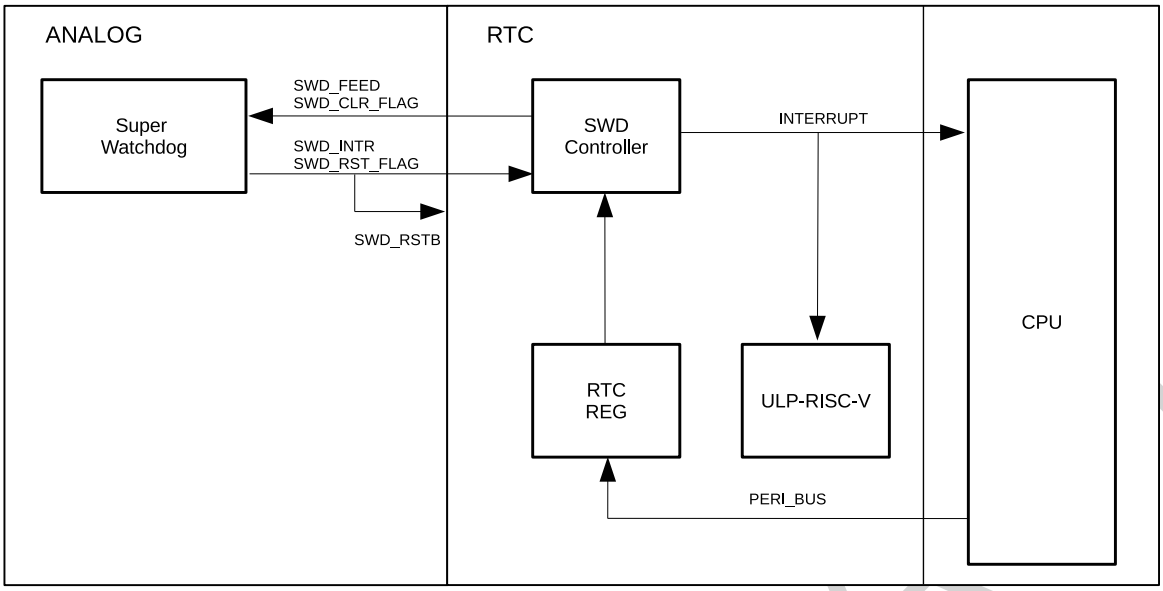


Figure 7-3. Super Watchdog Controller Structure

7.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU or ULP-RISC-V.
- Main CPU can decide whether to feed SWD directly by setting RTC_CNTL_SWD_FEED, or send an interrupt to ULP-RISC-V and ask ULP-RISC-V to feed SWD by setting RTC_CNTL_SWD_FEED.
- When trying to feed SWD, CPU or ULP-RISC-V needs to disable SWD controller's write protection by writing 0x8F1D312A to RTC_CNTL_SWD_WKEY. This prevents SWD from being fed by mistake when the system is operating in sub-optimal state.
- If setting RTC_CNTL_SWD_AUTO_FEED_EN to 1, SWD controller can also feed SWD itself without any interaction with CPU or ULP-RISC-V.

After reset:

- Check RTC_CNTL_RESET_CAUSE_PROCPU[5:0] for the cause of CPU reset.
  If RTC_CNTL_RESET_CAUSE_PROCPU[5:0] == 0x12, it indicates that the cause is SWD reset.
- Set RTC_CNTL_SWD_RST_FLAG_CLR to clear the SWD reset flag.

7.4 Interrupts

For watchdog timer interrupts, please refer to Section 6.2.6 *Interrupts* in Chapter 6 *Timer Group (TIMG)*.

7.5 Registers

MWDT registers are part of the timer submodule and are described in Section 6.4 *Register Summary* in Chapter 6 *Timer Group (TIMG)*. RWDT and SWD registers are part of the RTC submodule and are described in Section 21 *Register Summary* in Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*.

8 XTAL32K Watchdog Timers (XTWDT)

8.1 Overview

The XTAL32K watchdog timer on ESP32-S3 is used to monitor the status of external crystal XTAL32K_CLK. This watchdog timer 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 (for clock description, see Chapter 3 *Reset and Clock*) and stops oscillating, the XTAL32K watchdog timer first switches to BACKUP32K_CLK derived from RTC_CLK and generates an interrupt (if the chip is in Light-sleep or Deep-sleep mode, the CPU will be woken up), and then switches back to XTAL32K_CLK after it is restarted by software.

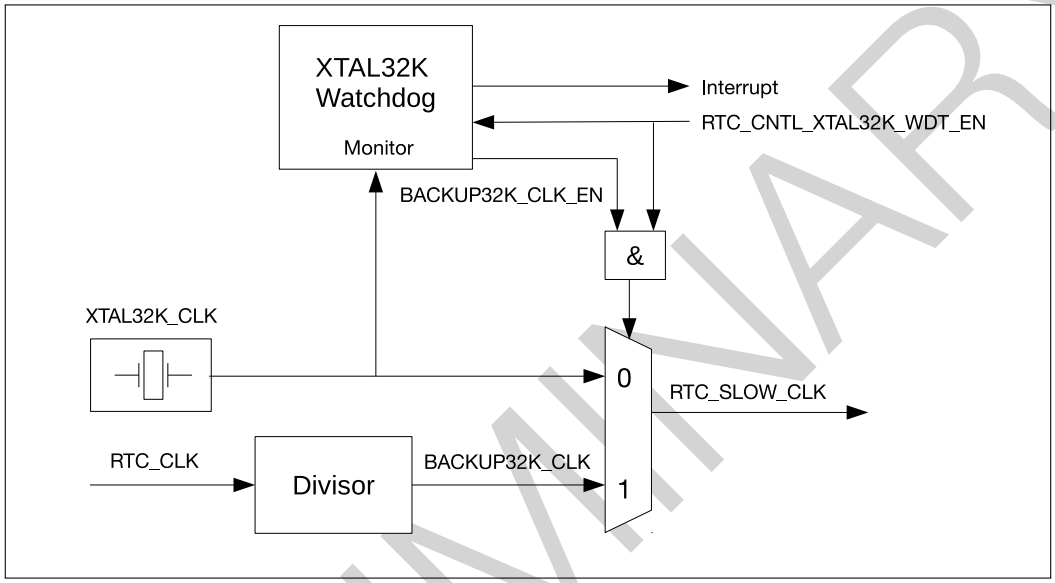


Figure 8-1. XTAL32K Watchdog Timer

8.2 Features

8.2.1 Interrupt and Wake-Up

When the XTAL32K watchdog timer detects the oscillation failure of XTAL32K_CLK, an oscillation failure interrupt RTC_XTAL32K_DEAD_INT (for interrupt description, please refer to Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*) is generated. At this point, the CPU will be woken up if in Light-sleep mode or Deep-sleep mode.

8.2.2 BACKUP32K_CLK

Once the XTAL32K watchdog timer detects the oscillation failure of XTAL32K_CLK, it replaces XTAL32K_CLK with BACKUP32K_CLK (with a frequency of 32 kHz or so) derived from RTC_CLK as RTC's SLOW_CLK, so as to ensure proper functioning of the system.

8.3 Functional Description

8.3.1 Workflow

1. The XTAL32K watchdog timer starts counting when RTC_CNTL_XTAL32K_WDT_EN is enabled. The counter based on RTC_CLK 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. If RTC_CNTL_XTAL32K_AUTO_BACKUP is set and step 1 is finished, the XTAL32K watchdog timer will automatically enable 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 8.3.2.
3. To restore the XTAL32K watchdog timer, software restarts XTAL32K_CLK by turning its XPD (meaning no power-down) signal off and on again via RTC_CNTL_XPD_XTAL_32K bit. Then, the XTAL32K watchdog timer switches back to XTAL32K_CLK as the clock source of RTC SLOW_CLK by clearing RTC_CNTL_XTAL32K_WDT_EN (BACKUP32K_CLK_EN is also automatically cleared). If the chip is in Light-sleep or Deep-sleep mode mode, the XTAL32K watchdog timer will wake up the CPU to finish the above steps.

8.3.2 BACKUP32K_CLK Working Principle

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 (see details in Chapter 15 *Low-Power Management (RTC_CNTL) [to be added later]*) via RTC_CNTL_XTAL32K_CLK_FACTOR_REG register. Each byte in this register corresponds to a divisor component (x_0 ~x_7). BACKUP32K_CLK is divided by a fraction where the denominator is always 4, as calculated below.

$$f\_back\_clk/4 = f\_rtc\_clk/S$$
$$S = x_0 + x_1 + ... + x_7$$

f_back_clk is the desired frequency of BACKUP32K_CLK; f_rtc_clk is the actual frequency of RTC_CLK; x_0 ~x_7 correspond to the pulse width in high and low state of four BACKUP32K_CLK clock signals (unit: RTC_CLK clock cycle).

8.3.3 Configuring the Divisor Component of BACKUP32K_CLK

Based on principles described in Section 8.3.2, you can configure the divisor component as follows:

- Calculate the sum of divisor components S according to the frequency of RTC_CLK and the desired frequency of BACKUP32K_CLK;
- Calculate the integer part of divisor $N = f\_rtc\_clk/f\_back\_clk$;
- Calculate the integer part of divisor component $M = N/2$. The integer part of divisor N are separated into two parts because a divisor component corresponds to a pulse width in high or low state;
- Calculate the number of divisor components that equal M (x_n = M) and the number of divisor components that equal M + 1 (x_n = M + 1) according to the value of M and S. (M + 1) is the fractional part of divisor component.

For example, if the frequency of RTC_CLK is 163 kHz, then $f\_rtc\_clk = 163000$, $f\_back\_clk = 32768$, $S = 20$, $M = 2$, 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.

9 SHA Accelerator (SHA)

9.1 Introduction

ESP32-S3 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-S3 has two working modes, which are Typical SHA and DMA-SHA.

9.2 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
- interleaved function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

9.3 Working Modes

The SHA accelerator integrated in ESP32-S3 has two working modes.

- Typical SHA Working Mode: all the data is written and read via CPU directly.
- DMA-SHA Working Mode: all the data is read via DMA. That is, users can configure the DMA controller to read all the data needed for hash operation, thus releasing CPU for completing other tasks.

Users can start the SHA accelerator with different working modes by configuring registers SHA_START_REG and SHA_DMA_START_REG. For details, please see Table 9-1.

Table 9-1. SHA Accelerator Working Mode

Working Mode	Configuration Method
Typical SHA	Set SHA_START_REG to 1
DMA-SHA	Set SHA_DMA_START_REG to 1

Users can choose hash algorithms by configuring the SHA_MODE_REG register. For details, please see Table 9-2.

Table 9-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/t	7

Notice:

ESP32-S3's Digital Signature (DS) and HMAC Accelerator (HMAC) [to be added later] modules also call the SHA accelerator. Therefore, users cannot access the SHA accelerator when these modules are working.

9.4 Function Description

SHA accelerator can generate the message digest via two steps: Preprocessing and Hash operation.

9.4.1 Preprocessing

Preprocessing consists of three steps: padding the message, parsing the message into message blocks and setting the initial hash value.

9.4.1.1 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 task.

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 of value equal to the number m expressed using a binary representation.

- **SHA-384**, **SHA-512**, **SHA-512/224**, **SHA-512/256** and **SHA-512/t**

  1. First, append the bit "1" to the end of the message;
  2. Second, append k zero bits, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 896 \ mod \ 1024$;
  3. Last, append the 128-bit block of value equal to the number m expressed using a binary representation.

For more details, please refer to Section "5.1 Padding the Message" in FIPS PUB 180-4 Spec.

9.4.1.2 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 $M_{15}^{(i)}$.

During the task, 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.

9.4.1.3 Initial Hash Value

Before hash task begins for each of the secure hash algorithms, the initial Hash value $H^{(0)}$ must be set based on different algorithms, among which the SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 algorithms use the initial Hash values (constant C) stored in the hardware.

However, SHA-512/t requires a distinct initial hash value for each operation for a given value of t. Simply put, SHA-512/t is the generic name for a t-bit hash function based on SHA-512 whose output is truncated to t bits. t is any positive integer without a leading zero such that t<512, and t is not 384. The initial hash value for SHA-512/t for a given value of t can be calculated by performing SHA-512 from hexadecimal representation of the string "SHA-512/t". It's not hard to observe that when determining the initial hash values for SHA-512/t algorithms with different t, the only difference lies in the value of t.

Therefore, we have specially developed the following simplified method to calculate the initial hash value for SHA-512/t:

1. **Generate t_string and t_length**: t_string is a 32-bit data that stores the input message of t. t_length is a 7-bit data that stores the length of the input message. The t_string and t_length are generated in methods described below, depending on the value of t:

   - If $1 <= t <= 9$, then $t\_length = 7'h48$ and t_string is padded in the following format:

$8'h30 + 8'ht_0$	$1'b1$	$23'b0$

     where $t_0 = t$.

     For example, if t = 8, then t_0 = 8 and t_string = $32'h38800000$.

   - If $10 <= t <= 99$, then $t\_length = 7'h50$ and t_string is padded in the following format:

$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$15'b0$

     where, $t_0 = t\%10$ and $t_1 = t/10$.

     For example, if t = 56, then t_0 = 6, t_1 = 5, and t_string = $32'h35368000$.

   - If $100 <= t < 512$, then $t\_length = 7'h58$ and t_string is padded in the following format:

$8'h30 + 8'ht_2$	$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$7'b0$

     where, $t_0 = t\%10$, $t_1 = (t/10)\%10$, and $t_2 = t/100$.

     For example, if t = 231, then t_0 = 1, t_1 = 3, t_2 = 2, and t_string = $32'h32333180$.

2. **Initialize relevant registers**: Initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length in the previous step.

3. **Obtain initial hash value**: Set the SHA_MODE_REG register to 7. Set the SHA_START_REG register to 1 to start the SHA accelerator. Then poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.

Please note that the initial value for SHA-512/t can be also calculated according to the Section "5.3.6 SHA-512/t" in FIPS PUB 180-4 Spec, 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".

9.4.2 Hash task Process

After the preprocessing, the ESP32-S3 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-S3 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.

9.4.2.1 Typical SHA Mode Process

Usually, the SHA accelerator will process all blocks of a message and produce a message digest before starting the next message digest.

However, ESP32-S3 SHA working in Typical SHA mode also supports optional "interleaved" message digest calculation. Users can insert new calculation (both Typical SHA and DMA-SHA) each 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 calculation completes, users can put the message digest stored back to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused calculation.

Typical SHA Process (except for SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 9-2.
2. Process the current message block [1].
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator.
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;
   - If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.
4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Typical SHA Process (SHA-512/*t*)

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register to 7 for SHA-512/*t*.
2. Calculate the initial hash value.
   (a) Calculate t_stiring and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 9.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
3. Process the current message block[1].
   - Write the message block in registers SHA_M_*n*_REG.
4. Start the SHA accelerator

- Set the SHA_CONTINUE_REG register to 1. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.

5. Check the progress of the calculation.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status[3].
6. Decide if you have more message blocks to process:
   - If yes, please go back to Step 3.
   - Otherwise, please continue.
7. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Note:

1. In this step, the software can also write the next message block (to be processed) in registers SHA_M_*n*_REG, if any, while the hardware starts SHA calculation, to save time.
2. You are resuming the SHA accelerator with the previously paused calculation.
3. Here you can decide if you want to insert other calculations. If yes, please go to the process for interleaved calculations for details.

As mentioned above, ESP32-S3 SHA accelerator supports **"interleaving" calculation under the Typical SHA working mode**.

The process to implement interleaved calculation is described below.

1. Prepare to hand the SHA accelerator over for an interleaved calculation by saving the following data of the previous calculation.
   - The selected hash algorithm stored in the SHA_MODE_REG register.
   - The message digest stored in registers SHA_H_*n*_REG.
2. Perform the interleaved calculation. For the detailed process of the interleaved calculation, please refer to Typical SHA process or DMA-SHA process, depending on the working mode of your interleaved calculation.
3. Prepare to hand the SHA accelerator back to the previously paused calculation by restoring the following data of the previous calculation.
   - Write the previously stored hash algorithm back to register SHA_MODE_REG
   - Write the previously stored message digest back to registers SHA_H_*n*_REG
4. Write the next message block from the previous paused calculation in registers SHA_M_*n*_REG, and set the SHA_CONTINUE_REG register to 1 to restart the SHA accelerator with the previously paused calculation.

9.4.2.2 DMA-SHA Mode Process

ESP32-S3 SHA accelerator does not support "interleaving" message digest calculation when using the DMA, which means you cannot insert new calculation before the whole DMA-SHA process completes. In this case,

users who need inserted calculation are recommended to divide your message blocks and perform several DMA-SHA calculation, 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 DMA.

Therefore, users are required to configure the DMA controller following the description in Chapter 9 *GDMA Controller (DMA) [to be added later]*.

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 9-2.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
4. Start the DMA-SHA calculation.
   - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
6. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

DMA-SHA process for SHA-512/t

1. Select a hash algorithm.
   - Select SHA-512/t algorithm by configuring the SHA_MODE_REG register to 7.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Calculate the initial hash value.
   (a) Calculate t_string and t_length and initialize SHA_T_STRING_REG and SHA_T_LENGTH_REG with the generated t_string and t_length. For details, please refer to Section 9.4.1.3.
   (b) Set the SHA_START_REG register to 1 to start the SHA accelerator.
   (c) Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the calculation of initial hash value is completed.
4. Configure the number of message blocks.

- Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.

5. Start the DMA-SHA calculation.
   - Write 1 to register SHA_DMA_CONTINUE_REG to start the accelerator.
6. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
7. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

9.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG(*n*: 0~15). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 9-6 below:

Table 9-6. The Storage and Length of Message digest from Different Algorithms

Hash Algorithm	Length of Message Digest (in bits)	Storage[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t*[2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG

[1] The message digest 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 9.4.1.2.

[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.

9.4.4 Interrupt

SHA accelerator supports interrupt on the completion of message digest calculation when working in the DMA-SHA mode. To enable this function, write 1 to register SHA_INT_ENA_REG. Note that the interrupt should be cleared by software after use via setting the SHA_INT_CLEAR_REG register to 1.

9.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

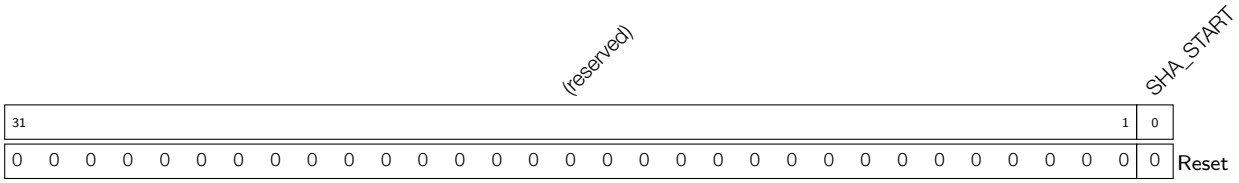
Name	Description	Address	Access
Control/Status registers			
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_BUSY_REG	Indicates if SHA Accelerator is busy or not	0x0018	RO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG	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
SHA_H_10_REG	Hash value	0x0068	R/W

Name	Description	Address	Access
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

9.6 Registers

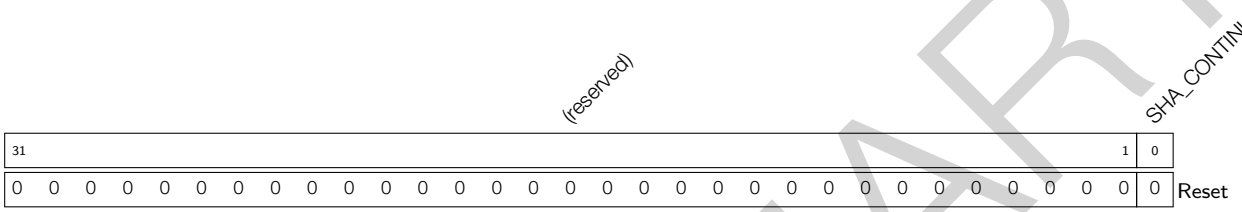
The addresses in this section are relative to the SHA accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 9.1. SHA_START_REG (0x0010)



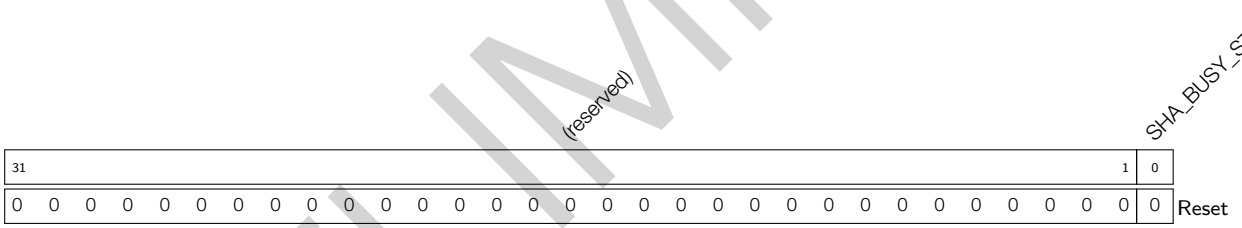
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 9.2. SHA_CONTINUE_REG (0x0014)



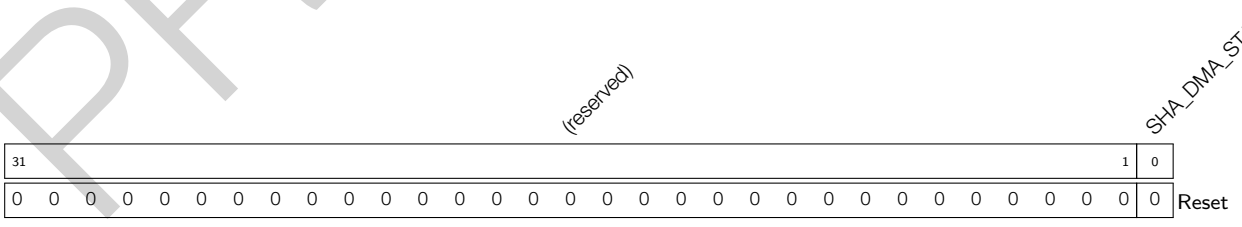
SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 9.3. SHA_BUSY_REG (0x0018)



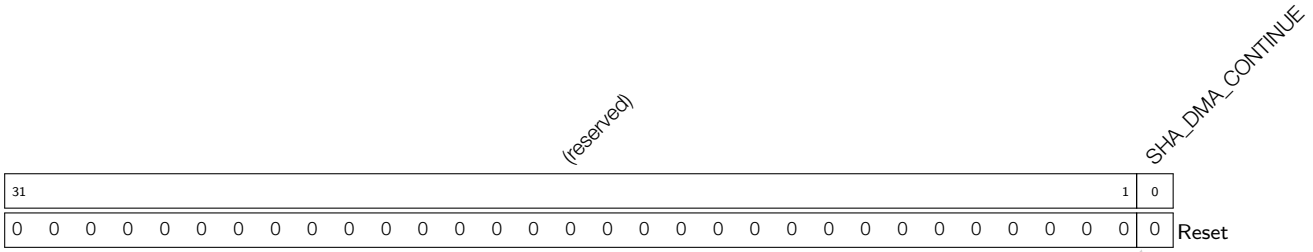
SHA_BUSY_STATE Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

Register 9.4. SHA_DMA_START_REG (0x001C)



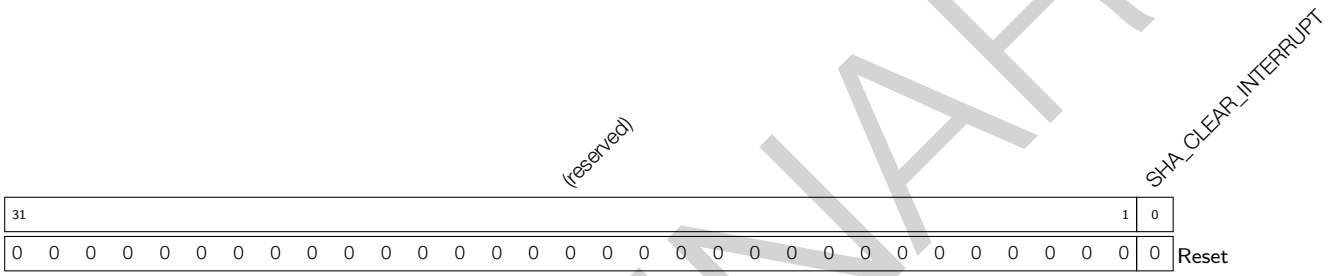
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 9.5. SHA_DMA_CONTINUE_REG (0x0020)



SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 9.6. SHA_INT_CLEAR_REG (0x0024)

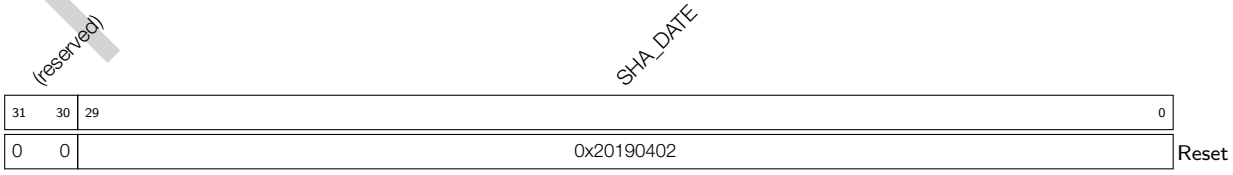


SHA_CLEAR_INTERRUPT Clears DMA-SHA interrupt. (WO)

Register 9.7. SHA_INT_ENA_REG (0x0028)

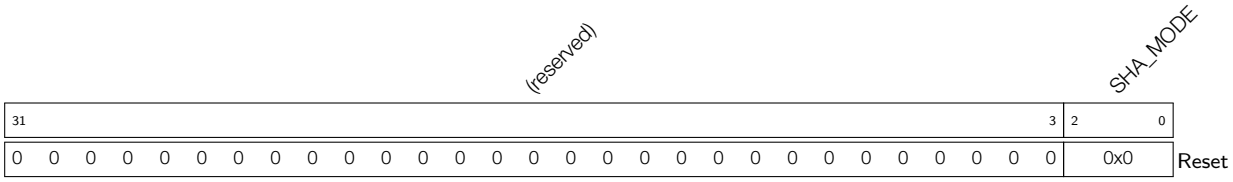
SHA_INTERRUPT_ENA Enables DMA-SHA interrupt. (R/W)

Register 9.8. SHA_DATE_REG (0x002C)



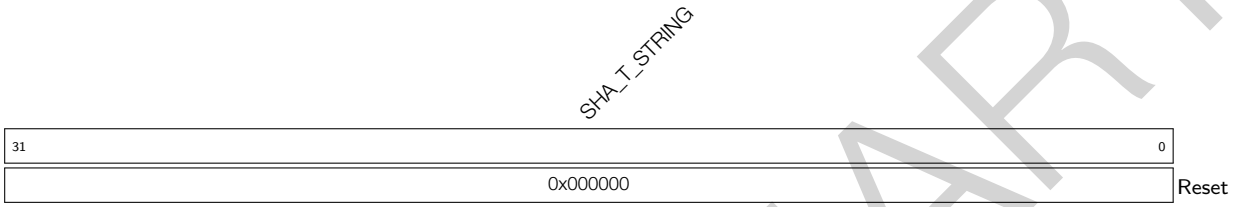
SHA_DATE Version control register. (R/W)

Register 9.9. SHA_MODE_REG (0x0000)



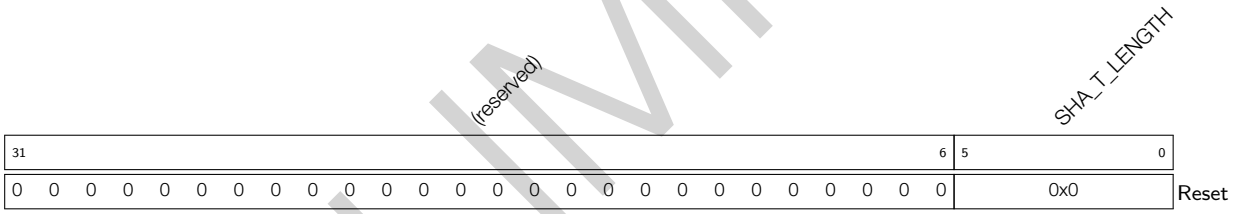
SHA_MODE Defines the SHA algorithm. For details, please see Table 9-2. (R/W)

Register 9.10. SHA_T_STRING_REG (0x0004)



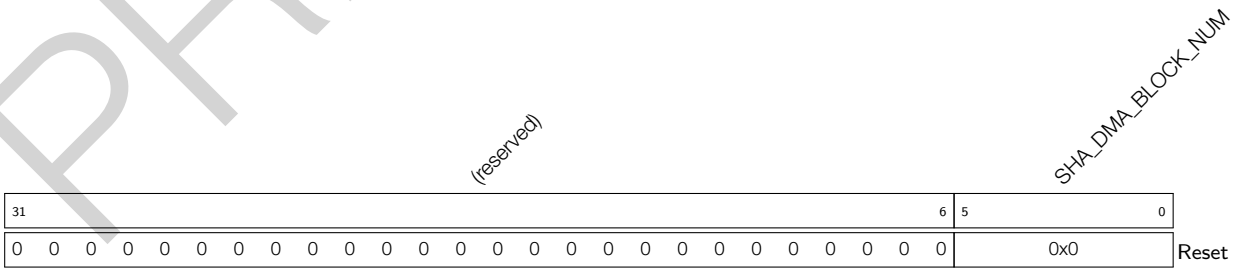
SHA_T_STRING Defines t_string for calculating the initial Hash value for SHA-512/t. (R/W)

Register 9.11. SHA_T_LENGTH_REG (0x0008)



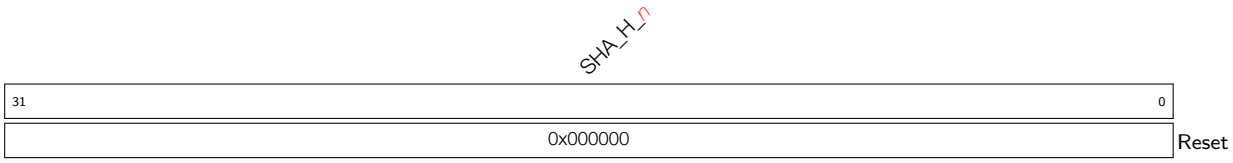
SHA_T_LENGTH Defines t_length for calculating the initial Hash value for SHA-512/t. (R/W)

Register 9.12. SHA_DMA_BLOCK_NUM_REG (0x000C)



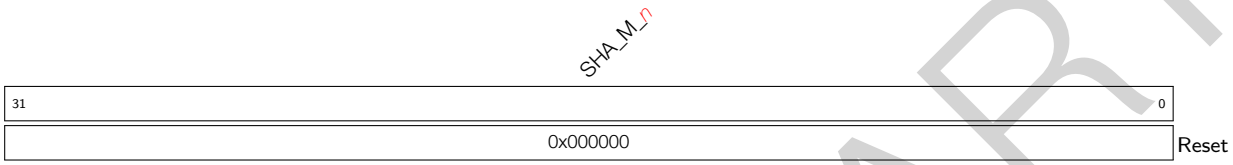
SHA_DMA_BLOCK_NUM Defines the DMA-SHA block number. (R/W)

Register 9.13. 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 9.14. SHA_M_*n*_REG (*n*: 0-31) (0x0080+4n*)**



SHA_M_*n* Stores the *n*th 32-bit piece of the message. (R/W)

10 AES Accelerator (AES)

10.1 Introduction

ESP32-S3 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-S3 has two working modes, which are Typical AES and DMA-AES.

10.2 Features

The following functionality is supported:

- Typical AES working mode
  - AES-128/AES-256 encryption and decryption
- DMA-AES working mode
  - AES-128/AES-256 encryption and decryption
  - Block cipher mode
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - Interrupt on completion of computation

10.3 AES Working Modes

The AES Accelerator integrated in ESP32-S3 has two working modes, which are Typical AES and DMA-AES.

- Typical AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197.

  In this working mode, the plaintext and ciphertext is written and read via CPU directly.
- DMA-AES Working Mode:
  - Supports encryption and decryption using cryptographic keys of 128 and 256 bits, specified in NIST FIPS 197;
  - Supports block cipher modes ECB/CBC/OFB/CTR/CFB8/CFB128 under NIST SP 800-38A.

  In this working mode, the plaintext and ciphertext is written and read via DMA. An interrupt will be generated when operation completes.

Users can choose the working mode for AES accelerator by configuring the AES_DMA_ENABLE_REG register according to Table 10-1 below.

Table 10-1. AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode
0	Typical AES
1	DMA-AES

Users can choose the length of cryptographic keys and encryption / decryption by configuring the AES_MODE_REG register according to Table 10-2 below.

Table 10-2. Key Length and Encryption / Decryption

AES_MODE_REG[2:0]	Key Length and Encryption / Decryption
0	AES-128 encryption
1	reserved
2	AES-256 encryption
3	reserved
4	AES-128 decryption
5	reserved
6	AES-256 decryption
7	reserved

For detailed introduction on these two working modes, please refer to Section 10.4 and Section 10.5 below.

Notice: ESP32-S3's Digital Signature (DS) module will call the AES accelerator. Therefore, users cannot access the AES accelerator when Digital Signature (DS) module is working.

10.4 Typical AES Working Mode

In the Typical AES working mode, users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 10-3 below.

Table 10-3. Working Status under Typical AES Working Mode

AES_STATE_REG	Status	Description
0	IDLE	The AES accelerator is idle or completed operation.
1	WORK	The AES accelerator is in the middle of an operation.

10.4.1 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-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-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-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES Accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

10.4.2 Endianness

Text Endianness

In Typical AES working mode, the AES Accelerator uses cryptographic keys to encrypt and decrypt data in blocks of 128 bits. When filling data into AES_TEXT_IN_*m*_REG register or reading result from AES_TEXT_OUT_*m*_REG registers, users should follow the text endianness type specified in Table 10-4.

Table 10-4. Text Endianness Type for Typical AES

Plaintext/Ciphertext					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] The definition of "State (including c and r)" is described in Section 3.4 The State in NIST FIPS 197.

[2] Where *x* = IN or OUT.

Key Endianness

In Typical AES working mode, When filling key into AES_KEY_*m*_REG registers, users should follow the key endianness type specified in Table 10-5 and Table 10-6.

Table 10-5. Key Endianness Type for AES-128 Encryption and Decryption

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Column "Bit" specifies the bytes of each word stored in w[0] ~ w[3].

[2] w[0] ~ w[3] are "the first Nk words of the expanded key" as specified in Section 5.2 Key Expansion in NIST FIPS 197.

Table 10-6. Key Endianness Type for AES-256 Encryption and Decryption

Bit[1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

[1] Column “Bit” specifies the bytes of each word stored in w[0] ~ w[7].

[2] w[0] ~ w[7] are “the first Nk words of the expanded key” as specified in Chapter 5.2 Key Expansion in NIST FIPS 197.

10.4.3 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.
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 is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

1. Write 0 to the AES_DMA_ENABLE_REG register before starting the first operation.
2. Initialize registers AES_MODE_REG and AES_KEY_*n*_REG before starting the first operation.
3. Update the content of AES_TEXT_IN_*m*_REG.
4. Start operation by writing 1 to the AES_TRIGGER_REG register.
5. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation completes.
6. Read results from the AES_TEXT_OUT_*m*_REG register, and return to Step 3 to continue the next operation.

10.5 DMA-AES Working Mode

In the DMA-AES working mode, the AES accelerator supports six block cipher modes including ECB/CBC/OFB/CTR/CFB8/CFB128. Users can choose the block cipher mode by configuring the AES_BLOCK_MODE_REG register according to Table 10-7 below.

Table 10-7. Block Cipher Mode

AES_BLOCK_MODE_REG[2:0]	Block Cipher Mode
0	ECB (Electronic Codebook)
1	CBC (Cipher Block Chaining)
2	OFB (Output Feedback)
3	CTR (Counter)
4	CFB8 (8-bit Cipher Feedback)
5	CFB128 (128-bit Cipher Feedback)
6	reserved
7	reserved

Users can check the working status of the AES accelerator by inquiring the AES_STATE_REG register and comparing the return value against the Table 10-8 below.

Table 10-8. Working Status under DMA-AES Working mode

AES_STATE_REG[1:0]	Status	Description
0	IDLE	The AES accelerator is idle.
1	WORK	The AES accelerator is in the middle of an operation.
2	DONE	The AES accelerator completed operations.

When working in the DMA-AES working mode, the AES accelerator supports interrupt on the completion of computation. To enable this function, write 1 to the AES_INT_ENA_REG register. By default, the interrupt function is disabled. Also, note that the interrupt should be cleared by software after use.

10.5.1 Key, Plaintext, and Ciphertext

Block Operation

During the block operations, the AES Accelerator reads source data from DMA, and write result data to DMA after the computation.

- For encryption, DMA reads plaintext from memory, then passes it to AES as source data. After computation, AES passes ciphertext as result data back to DMA to write into memory.
- For decryption, DMA reads ciphertext from memory, then passes it to AES as source data. After computation, AES passes plaintext as result data back to DMA to write into memory.

During block operations, the lengths of the source data and result data are the same. The total computation time is reduced because the DMA data operation and AES computation can happen concurrently.

The length of source data for AES Accelerator under DMA-AES working mode must be 128 bits or the integral multiples of 128 bits. Otherwise, trailing zeros will be added to the original source data, so the length of source data equals to the nearest integral multiples of 128 bits. Please see details in Table 10-9 below.

Table 10-9. TEXT-PADDING

Function : TEXT-PADDING( )
Input : X, bit string. **Output** : Y = **TEXT-PADDING**(X), whose length is the nearest integral multiples of 128 bits.
Steps Let us assume that X is a data-stream that can be split into n parts as following: $X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$ Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is t (0<=t<=127). If $t = 0$, then **TEXT-PADDING**(X) = X; If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n \| \| 0^{128-t}$, then **TEXT-PADDING**($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

10.5.2 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 10-10 below.

Table 10-10. 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-S3. 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. Details can be found in Chapter 9 *GDMA Controller (DMA) [to be added later]*.

10.5.3 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.

10.5.4 Block Number

Register AES_BLOCK_NUM_REG stores the Block Number of plaintext P or ciphertext C. The length of this register equals to length(**TEXT-PADDING**(P))/128 or length(**TEXT-PADDING**(C))/128. The AES Accelerator only uses this register when working in the DMA-AES mode.

10.5.5 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 ··· Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 10-10, i.e. the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

For more details on IV and ICB, please refer to NIST SP 800-38A.

10.5.6 Block Operation Process

1. Select one of DMA channels to connect with AES, configure the DMA chained list, and then start DMA. For details, please refer to Chapter 9 *GDMA Controller (DMA) [to be added later]*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 10-7.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 10.5.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES Accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (This is always needed except for ECB block operation).
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.
5. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 9 *GDMA Controller (DMA) [to be added later]*.
6. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occurred during the computation.
7. Release the AES Accelerator by writing 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 4 is completed.

10.6 Memory Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Size (byte)	Starting Address	Ending Address	Access
AES_IV_MEM	Memory IV	16 bytes	0x0050	0x005F	R/W

10.7 Register Summary

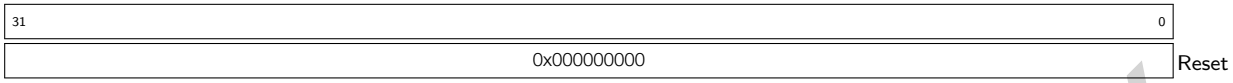
The addresses in this section are relative to the AES accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

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
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	Defines key length and encryption / decryption	0x0040	R/W
AES_DMA_ENABLE_REG	Selects the working mode of the AES accelerator	0x0090	R/W
AES_BLOCK_MODE_REG	Defines the block cipher mode	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
Controlling / Status Registers			
AES_TRIGGER_REG	Operation start controlling register	0x0048	WO
AES_STATE_REG	Operation status register	0x004C	RO
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

10.8 Registers

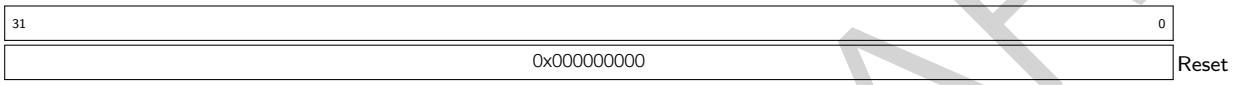
The addresses in this section are relative to the AES accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 10.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



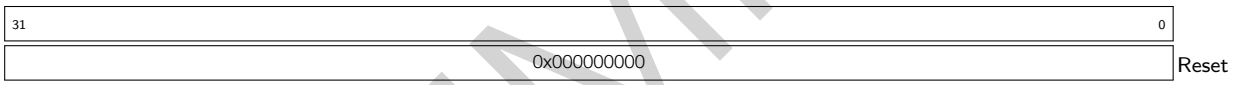
AES_KEY_*n*_REG (*n*: 0-7) Stores AES keys. (R/W)

Register 10.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



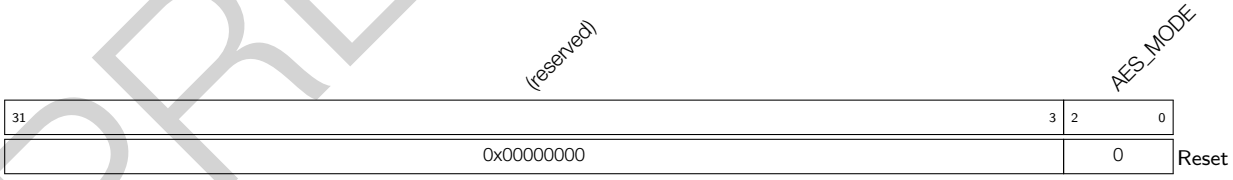
AES_TEXT_IN_*m*_REG (*m*: 0-3) Stores the source data when the AES Accelerator operates in the Typical AES working mode. (R/W)

Register 10.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**



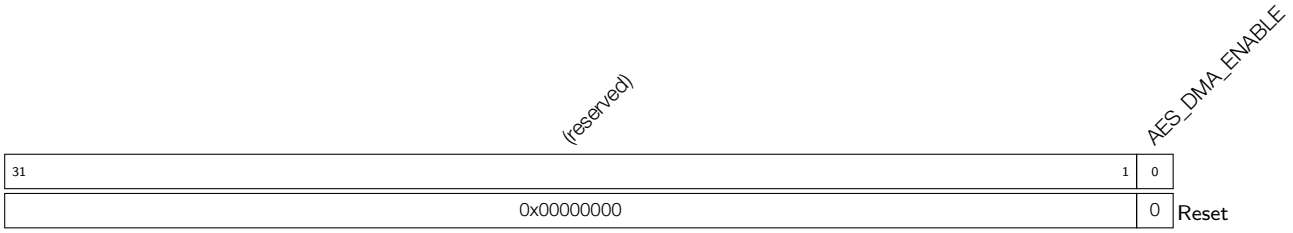
AES_TEXT_OUT_*m*_REG (*m*: 0-3) Stores the result data when the AES Accelerator operates in the Typical AES working mode. (RO)

Register 10.4. AES_MODE_REG (0x0040)



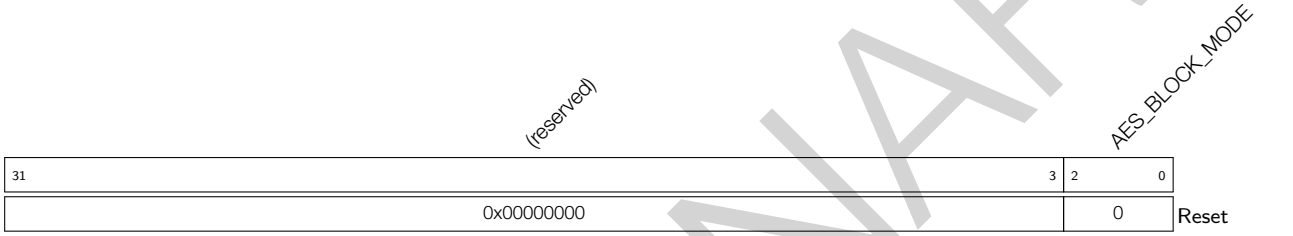
AES_MODE Defines the key length and encryption / decryption of the AES Accelerator. For details, see Table 10-2. (R/W)

Register 10.5. AES_DMA_ENABLE_REG (0x0090)



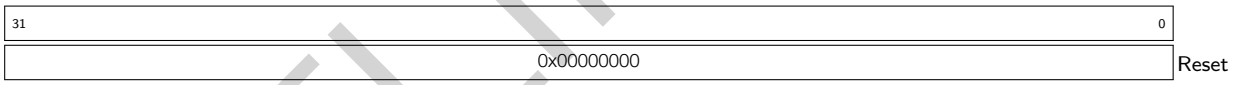
AES_DMA_ENABLE Defines the working mode of the AES Accelerator. 0: Typical AES, 1: DMA-AES. For details, see Table 10-1. (R/W)

Register 10.6. AES_BLOCK_MODE_REG (0x0094)



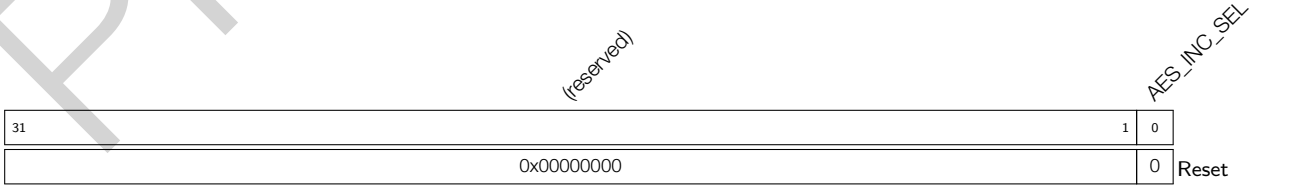
AES_BLOCK_MODE Defines the block cipher mode of the AES Accelerator operating under the DMA-AES working mode. For details, see Table 10-7. (R/W)

Register 10.7. AES_BLOCK_NUM_REG (0x0098)



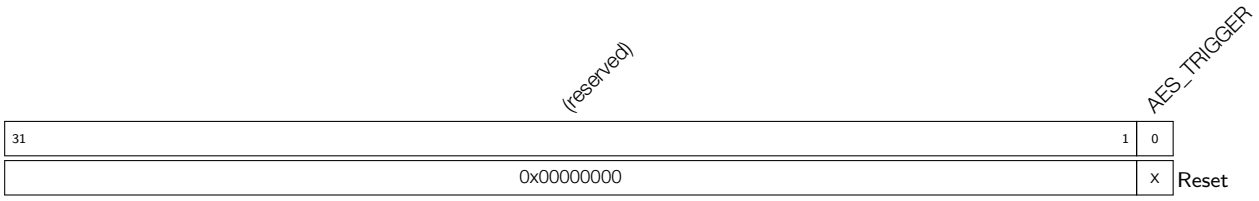
AES_BLOCK_NUM Stores the Block Number of plaintext or ciphertext when the AES Accelerator operates under the DMA-AES working mode. For details, see Section 10.5.4. (R/W)

Register 10.8. AES_INC_SEL_REG (0x009C)



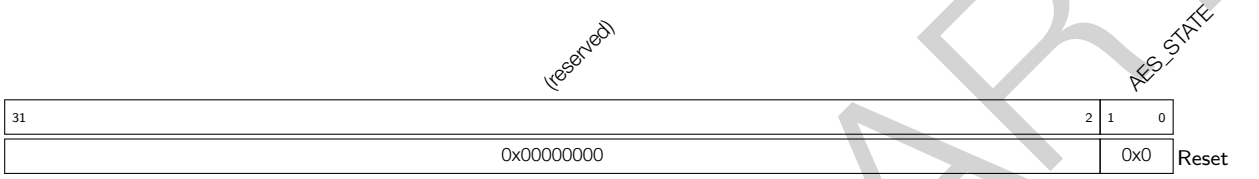
AES_INC_SEL Defines the Standard Incrementing Function for CTR block operation. Set this bit to 0 or 1 to choose INC_{32} or INC_{128}. (R/W)

Register 10.9. AES_TRIGGER_REG (0x0048)



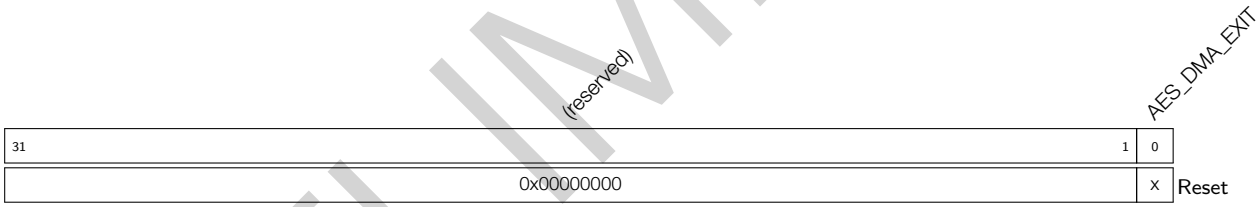
AES_TRIGGER Set this bit to 1 to start AES operation. (WO)

Register 10.10. AES_STATE_REG (0x004C)



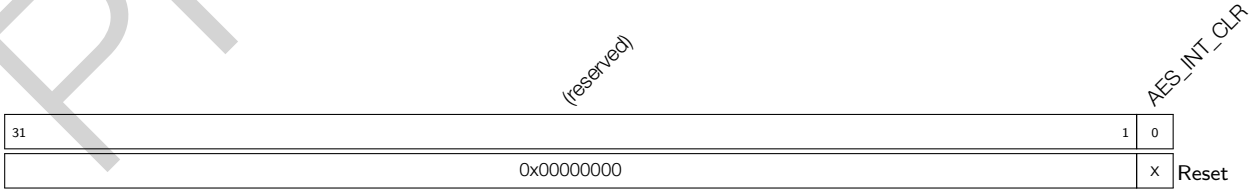
AES_STATE Stores the working status of the AES Accelerator. For details, see Table 10-3 for Typical AES working mode and Table 10-8 for DMA AES working mode. (RO)

Register 10.11. AES_DMA_EXIT_REG (0x00B8)



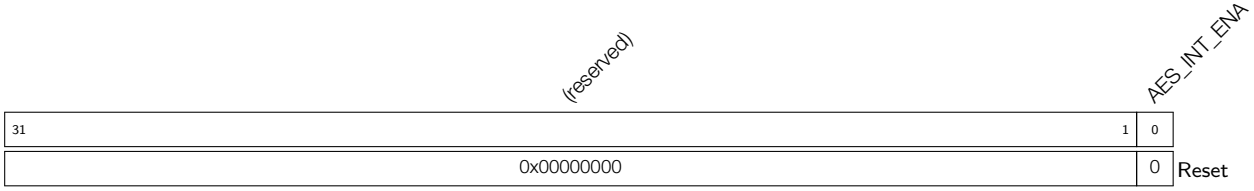
AES_DMA_EXIT Set this bit to 1 to exit AES operation. This register is only effective for DMA-AES operation. (WO)

Register 10.12. AES_INT_CLR_REG (0x00AC)



AES_INT_CLR Set this bit to 1 to clear AES interrupt. (WO)

Register 10.13. AES_INT_ENA_REG (0x00B0)



AES_INT_ENA Set this bit to 1 to enable AES interrupt and 0 to disable interrupt. (R/W)

11 RSA Accelerator (RSA)

11.1 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.

11.2 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

11.3 Functional Description

The RSA Accelerator is activated by setting the SYSTEM_CRYPTO_RSA_CLK_EN bit in the SYSTEM_PERIP_CLK_EN1_REG register and clearing the SYSTEM_RSA_MEM_PD bit in the SYSTEM_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-S3's Digital Signature (DS) module also calls the RSA accelerator. Therefore, users cannot access the RSA accelerator when Digital Signature (DS) is working.

11.3.1 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 $\overline{r} = R^2 \bmod M$.

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 11.3.4.
3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length 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 $i \in \{0, 1, \ldots, n-1\}$ 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.

11.3.2 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This computation is based on Montgomery multiplication. Therefore, similar to the large number modular exponentiation, two additional arguments are needed – $\overline{r}$ and M', which need to be calculated in advance by software.

The RSA Accelerator supports large-number modular multiplication with operands of 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 $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length 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 $i \in \{0, 1, \ldots, n-1\}$ 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.

11.3.3 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 $x \in \{1, 2, 3, \ldots, 64\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_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 $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. The capacity of each memory block is 64 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 $i \in \{0, 1, \ldots, n-1\}$ to the address of the i words of the RSA_X_MEM memory block. Note that Y_i for $i \in \{0, 1, \ldots, n-1\}$ will not be written to the address of the i words of the RSA_Z_MEM register, but the address of the $n+i$ words, i.e. the base address of the RSA_Z_MEM memory plus the address offset $4 \times (n+i)$.

   Users need to write data to each memory block only according to the length of the number; data beyond this length 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 $i \in \{0, 1, \ldots, \hat{n}-1\}$ 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 Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

11.3.4 Options for Acceleration

The ESP32-S3 RSA accelerator also provides SEARCH and CONSTANT_TIME options that can be configured to accelerate the large-number modular exponentiation. By default, both options are configured for no acceleration. Users can choose to use one or two of these options to accelerate the computation.

To be more specific, when neither of these two options are configured for acceleration, the time required to calculate $Z = X^Y \bmod M$ is solely determined by the lengths of operands. When either or both of these two options are configured for acceleration, the time required is also correlated with the 0/1 distribution of Y.

To better illustrate how these two options work, first assume Y is represented in binaries as

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

where,

- N is the length of Y,
- $\widetilde{Y}_t$ is 1,
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ are all equal to 0,
- and $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ are either 0 or 1 but exactly m bits should be equal to 0 and t-m bits 1, i.e. the Hamming weight of $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ is $t - m$.

When either of these two options is configured for acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. The maximum value of α is N-1, which leads to the same result when this option is not used for acceleration. The best acceleration performance can be achieved by setting α to t, in which case, all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 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 (Configuring RSA_CONSTANT_TIME_REG to 0 for acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better the acceleration performance is.

We provide an example to demonstrate the performance of the RSA Accelerator under different combinations of SEARCH and CONSTANT_TIME configuration. Here we perform $Z = X^Y \bmod M$ with N = 3072 and Y = 65537. Table 11-1 below demonstrates the time costs under different combinations of SEARCH and CONSTANT_TIME configuration. Here, we should also mention that, α is set to 16 when the SEARCH option is enabled.

Table 11-1. Acceleration Performance

SEARCH Option	CONSTANT_TIME Option	Time Cost
No acceleration	No acceleration	376.405 ms
Accelerated	No acceleration	2.260 ms
No acceleration	Acceleration	1.203 ms
Acceleration	Acceleration	1.165 ms

It's obvious that:

- The time cost is the biggest when none of these two options is configured for acceleration.
- The time cost is the smallest when both of these two options are configured for acceleration.
- The time cost can be dramatically reduced when either or both option(s) are configured for acceleration.

11.4 Memory Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Table 11-2. RSA Accelerator Memory Blocks

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

11.5 Register Summary

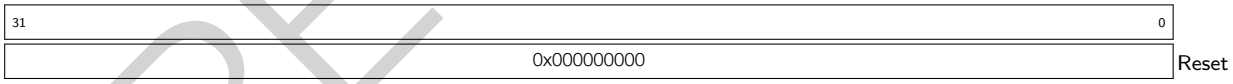
The addresses in this section are relative to the RSA accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

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
Interrupt 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

11.6 Registers

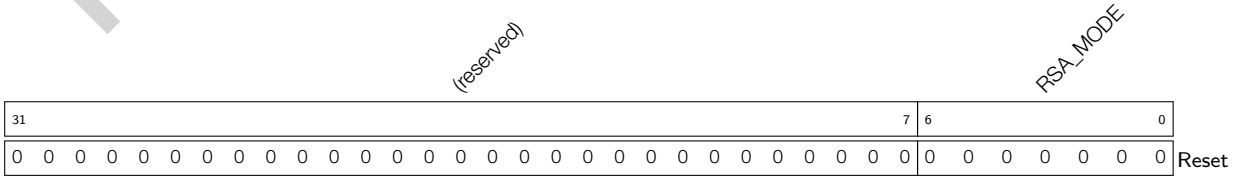
The addresses in this section are relative to the RSA accelerator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 11.1. RSA_M_PRIME_REG (0x0800)



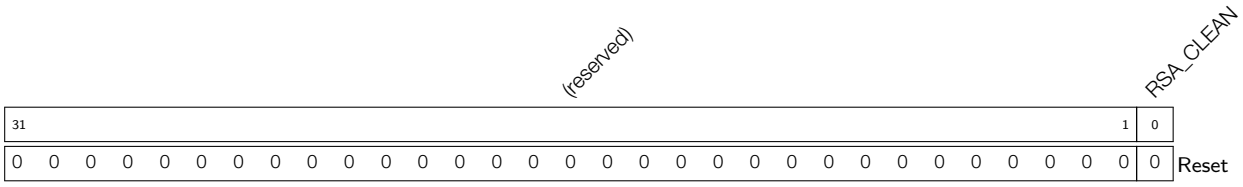
RSA_M_PRIME_REG Stores M'.(R/W)

Register 11.2. RSA_MODE_REG (0x0804)



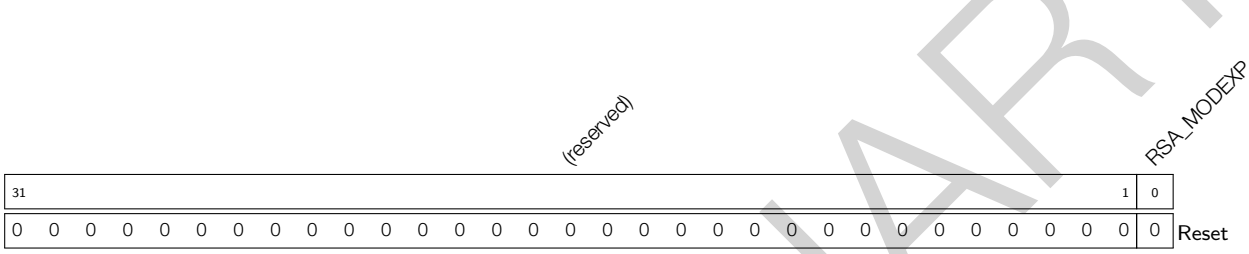
RSA_MODE Stores the mode of modular exponentiation. (R/W)

Register 11.3. RSA_CLEAN_REG (0x0808)



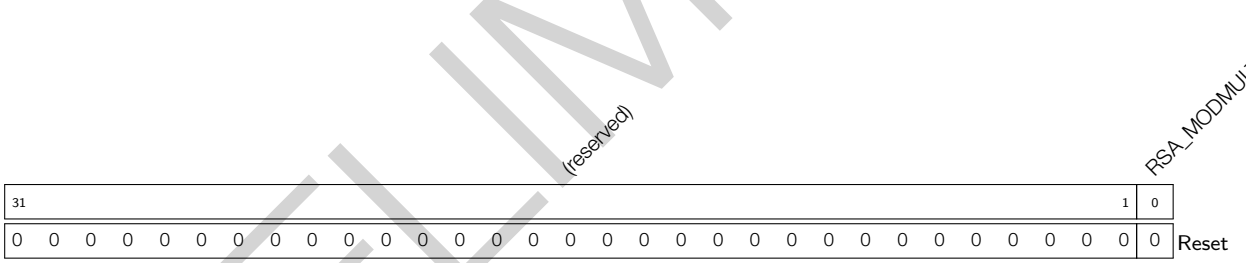
RSA_CLEAN The content of this bit is 1 when memories complete initialization. (RO)

Register 11.4. RSA_MODEXP_START_REG (0x080C)



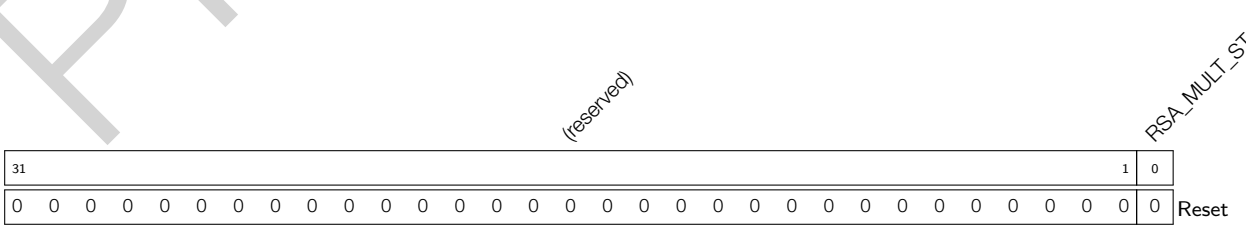
RSA_MODEXP_START Set this bit to 1 to start the modular exponentiation. (WO)

Register 11.5. RSA_MODMULT_START_REG (0x0810)



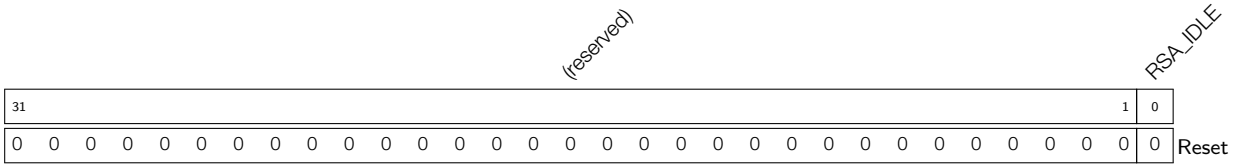
RSA_MODMULT_START Set this bit to 1 to start the modular multiplication. (WO)

Register 11.6. RSA_MULT_START_REG (0x0814)



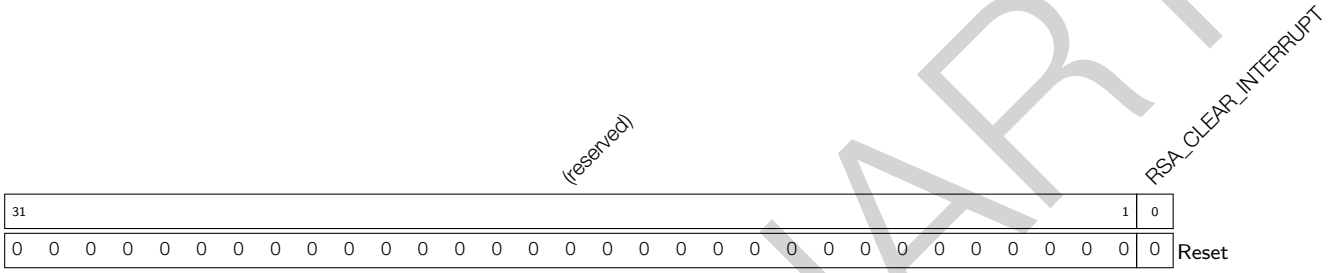
RSA_MULT_START Set this bit to 1 to start the multiplication. (WO)

Register 11.7. RSA_IDLE_REG (0x0818)



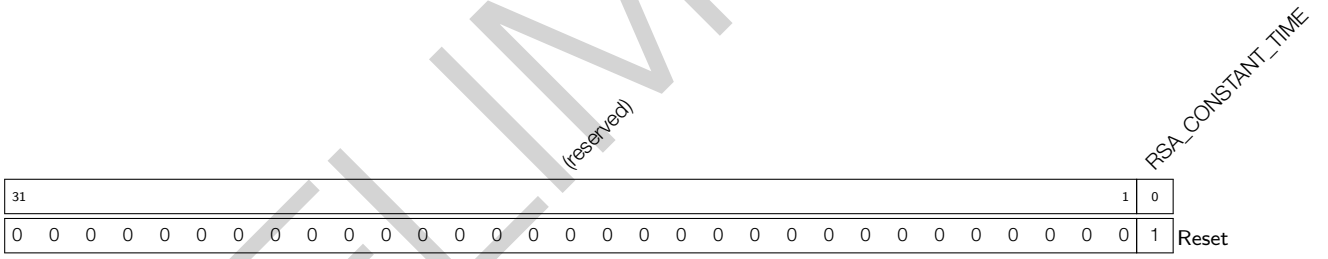
RSA_IDLE The content of this bit is 1 when the RSA accelerator is idle. (RO)

Register 11.8. RSA_CLEAR_INTERRUPT_REG (0x081C)



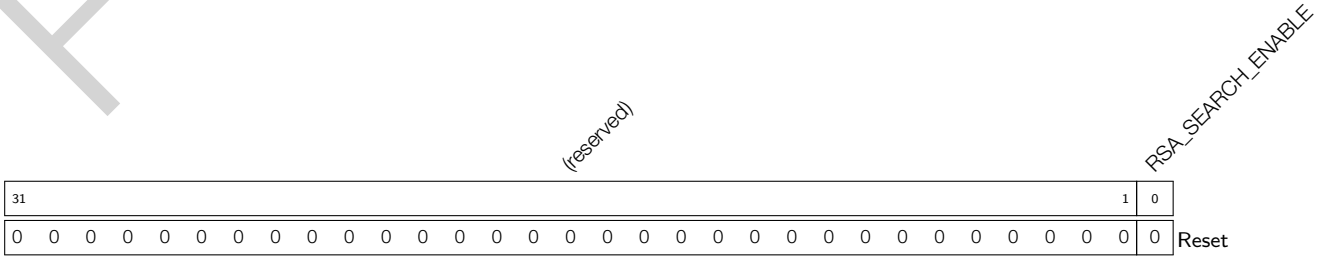
RSA_CLEAR_INTERRUPT Set this bit to 1 to clear the RSA interrupts. (WO)

Register 11.9. RSA_CONSTANT_TIME_REG (0x0820)



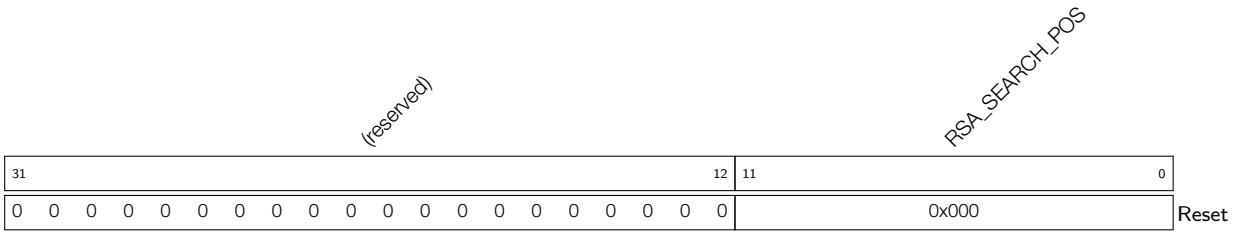
RSA_CONSTANT_TIME_REG Controls the constant_time option. 0: acceleration. 1: no acceleration (by default). (R/W)

Register 11.10. RSA_SEARCH_ENABLE_REG (0x0824)



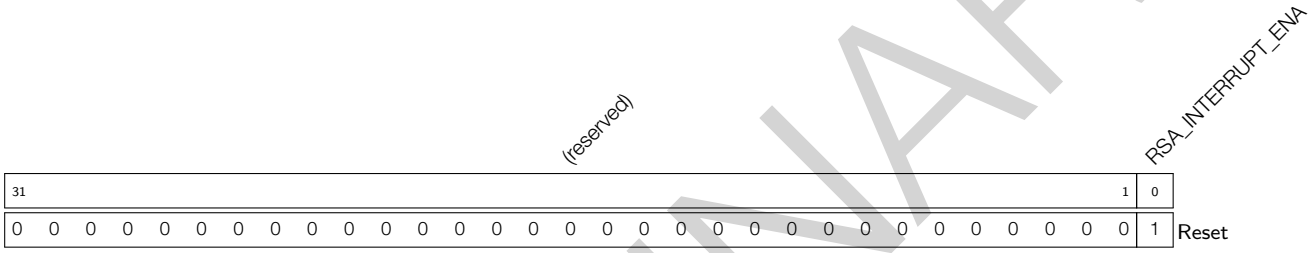
RSA_SEARCH_ENABLE Controls the search option. 0: no acceleration (by default). 1: acceleration. (R/W)

Register 11.11. RSA_SEARCH_POS_REG (0x0828)



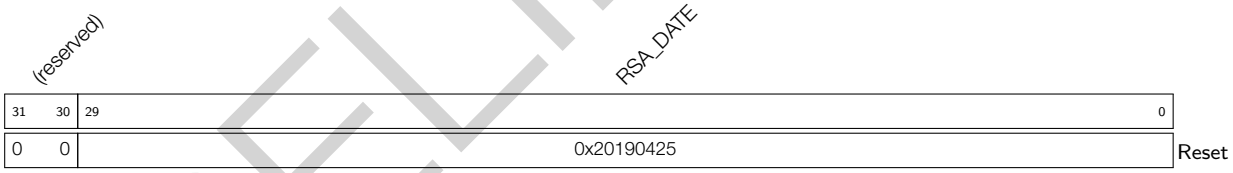
RSA_SEARCH_POS Is used to configure the starting address when the acceleration option of search is used. (R/W)

Register 11.12. 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 11.13. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

12 Digital Signature (DS)

12.1 Overview

A Digital Signature is used to verify the authenticity and integrity of a message using a cryptographic algorithm. This can be used to validate a device's identity to a server, or to check the integrity of a message.

The ESP32-S3 includes a Digital Signature (DS) module providing hardware acceleration of messages' signatures based on RSA. It uses pre-encrypted parameters to calculate a signature. The parameters are encrypted using HMAC as a key-derivation function. In turn, the HMAC uses eFuses as an input key. The whole process happens in hardware so that neither the decryption key for the RSA parameters nor the input key for the HMAC key derivation function can be seen by the software while calculating the signature.

12.2 Features

- RSA Digital Signatures with key length up to 4096 bits
- Encrypted private key data, only decryptable by DS peripheral
- SHA-256 digest to protect private key data against tampering by an attacker

12.3 Functional Description

12.3.1 Overview

The DS peripheral calculates RSA signature as $Z = X^Y \bmod M$ 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 other memory as ciphertext. They are decrypted using a key (DS_KEY) which can only be read by the DS peripheral via the HMAC peripheral. The required inputs ($HMAC\_KEY$) to generate the key are only stored in eFuse and can only be accessed by the HMAC peripheral. The DS peripheral hardware can decrypt the private key, and the private key in plaintext is never accessed by the software. For more detailed information about eFuse and HMAC peripherals, please refer to Chapter 5 *eFuse Controller (eFuse) [to be added later]* and 16 *HMAC Accelerator (HMAC) [to be added later]* peripheral.

The input message X will be sent directly to the DS peripheral by the software, each time a signature is needed. After the RSA signature operation, the signature Z is read back by the software.

For better understanding, we define some symbols and functions here, which are only applicable to this chapter:

- 1^s A bit string consist of s bits that stores "1".
- $[x]_s$ A bit string of s bits, in which s should be the integral multiple of 8 bits. If x is a number ($x < 2^s$), it is represented in little endian byte order in the bit string. x may be a variable value such as $[Y]_{4096}$ or as a hexadecimal constant such as $[0x0C]_8$. If necessary, the value $[x]_t$ can be right-padded with $(s - t)$ number of 0 to reach s bits in length, and finally get $[x]_s$. For example, $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$.
- $||$ A bit string concatenation operator for joining multiple bit strings into a longer bit string.

12.3.2 Private Key Operands

Private key operands Y (private key exponent) and M (key modulus) are generated by the user. They have a particular RSA key length (up to 4096 bits). Two additional private key operands are needed: $\overline{r}$ and M'. These two operands are derived from Y and M.

Operands Y, M, $\overline{r}$ and M' are encrypted by the user along with an authentication digest and stored as a single ciphertext C. C is inputted to the DS peripheral in this encrypted format, decrypted by the hardware, and then used for RSA signature calculation. Detailed description of how to generate C is provided in Section 12.3.3.

The DS peripheral supports RSA signature calculation $Z = X^Y \bmod M$, in which the length of operands should be $N = 32 \times x$ where $x \in \{1, 2, 3, \ldots, 128\}$. The bit lengths of arguments Z, X, Y, M and $\overline{r}$ should be an arbitrary value in N, and all of them in a calculation must be of the same length, while the bit length of M' should always be 32. For more detailed information about RSA calculation, please refer to Section 11.3.1 *Large Number Modular Exponentiation* in Chapter 11 *RSA Accelerator (RSA)*.

12.3.3 Software Prerequisites

The left side of Figure 12-1 lists preparations required by the software before the hardware starts RSA signature calculation, while the right side lists the hardware workflow during the entire calculation procedure.

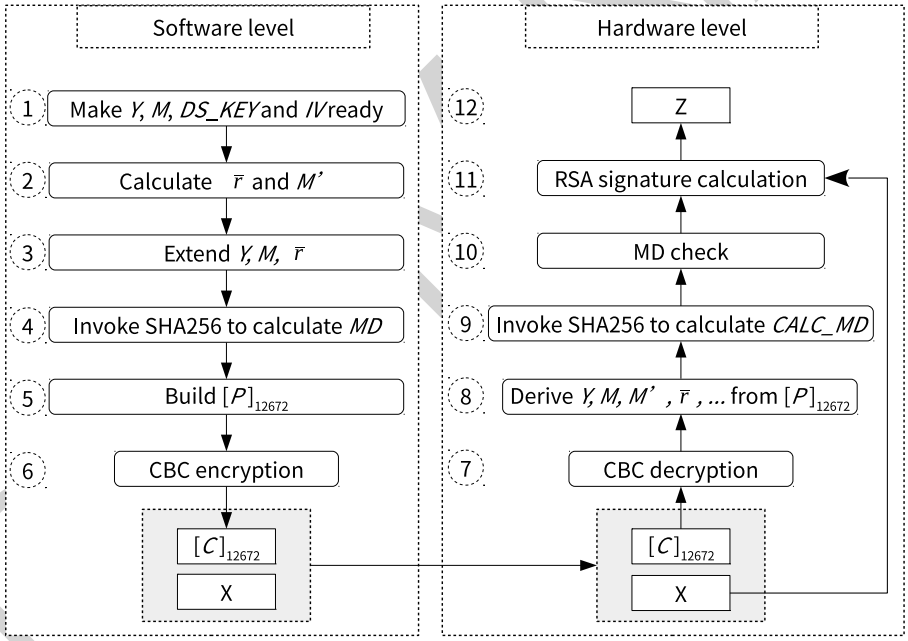


Figure 12-1. Software Preparations and Hardware Working Process

Note:

1. The software preparation (left side in the Figure 12-1) is a one-time operation before any signature is calculated, while the hardware calculation (right side in the Figure 1-1) repeats for every signature calculation.

Users need to follow the steps shown in the left part of Figure 12-1 to calculate C. Detailed instructions are as follows:

- **Step 1**: Prepare operands Y and M whose lengths should meet the requirements in Section 12.3.2.

Define $[L]_{32} = \frac{N}{32}$ (i.e., for RSA 4096, $[L]_{32}$ == [0x80]$_{32}$). Prepare $[HMAC\_KEY]_{256}$ and calculate $[DS\_KEY]_{256}$ based on DS_KEY = HMAC-SHA256 ($[HMAC\_KEY]_{256}$, 1^{256}). Generate a random $[IV]_{128}$ which should meet the requirements of the AES-CBC block encryption algorithm. For more information on AES, please refer to Chapter 10 *AES Accelerator (AES)*.

- **Step 2**: Calculate $\overline{r}$ and M' based on M.
- **Step 3**: Extend Y, M and $\overline{r}$, in order to get $[Y]_{4096}$, $[M]_{4096}$ and $[\overline{r}]_{4096}$, respectively. This step is only required for Y, M and $\overline{r}$ whose length are less than 4096 bits, since their largest length are 4096 bits.
- **Step 4** Calculate MD authentication code using the SHA-256:
  $[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **Step 5**: Build $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$), where $[\beta]_{64}$ is a PKCS#7 padding value, i.e., a 64-bit string [0x0808080808080808]$_{64}$ composed of 8 bytes (value = 0x80). The purpose of $[\beta]_{64}$ is to make the bit length of P a multiple of 128.
- **Step 6**: Calculate C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}$, $[DS\_KEY]_{256}$, $[IV]_{128}$), where C is the ciphertext with length of 12672 bits.

12.3.4 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 can be divided into the following three stages:

1. **Decryption: Step 7 and 8 in Figure 12-1**

   The decryption process is the inverse of Step 6 in figure 12-1. 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. $[DS\_KEY]_{256}$, as well as $[HMAC\_KEY]_{256}$ are not readable by the software.

   With P, the DS peripheral can derive $[Y]_{4096}$, $[M]_{4096}$, $[\overline{r}]_{4096}$, $[M']_{32}$, $[L]_{32}$, MD authentication code, and the padding value $[\beta]_{64}$. This process is the inverse of Step 5.

2. **Check: Step 9 and 10 in Figure 12-1**

   The DS peripheral will perform two checks: MD check and padding check. Padding check is not shown in Figure 12-1, as it happens at the same time with MD check.

   - MD check: The DS peripheral calls SHA-256 to calculate the MD authentication code $[CALC\_MD]_{256}$ from $[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$). Then, $[CALC\_MD]_{256}$ is compared against the pre-calculated MD authentication code $[MD]_{256}$ from step 4. 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.

   The DS peripheral will only perform subsequent operations if MD check passes. If padding check fails, an error bit is set in the query register, but it does not affect the subsequent operations, i.e., it is up to the user to proceed or not.

3. **Calculation: Step 11 and 12 in Figure 12-1**

   The DS peripheral treats X (input by users) and Y, M, $\overline{r}$ (compiled) 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.

12.3.5 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 12.3.4.

We assume that the software has called the HMAC peripheral and HMAC on the hardware has calculated DS_KEY based on $HMAC\_KEY$.

1. **Prerequisites**: Prepare operands C, X, IV according to Section 12.3.3.
2. **Activate the DS peripheral**: Write 1 to DS_SET_START_REG.
3. **Check if DS_KEY is ready**: Poll DS_QUERY_BUSY_REG until the software reads 0.

   If the software does not read 0 in DS_QUERY_BUSY_REG after approximately 1 ms, it indicates a problem with HMAC initialization. In such a case, the software can read register DS_QUERY_KEY_WRONG_REG to get more information:

   - If the software reads 0 in DS_QUERY_KEY_WRONG_REG, it indicates that the HMAC peripheral has not been activated.
   - If the software reads any value from 1 to 15 in DS_QUERY_KEY_WRONG_REG, 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 has been interrupted due to a software concurrency problem.
4. **Configure register**: Write IV block to register DS_IV_*m*_REG (*m*: 0-3). For more information on the IV block, please refer to Chapter 10 *AES Accelerator (AES)*.
5. **Write X to memory block DS_X_MEM**: Write X_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, 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 12.3.2) are ignored.
6. **Write C to memory block DS_C_MEM**: Write C_i ($i \in \{0, 1, \ldots, 395\}$) to memory block DS_C_MEM whose capacity is 396 words. Each word can store one base-b digit.
7. **Start DS operation**: Write 1 to register DS_SET_ME_REG.
8. **Wait for the operation to be completed**: Poll register DS_QUERY_BUSY_REG until the software reads 0.
9. **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 will resume directly from Step 11.
- 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. But please note that the data encapsulation format does not complie with the aforementioned PKCS#7 format, which may not be what you want.
- If the value is 3, it indicates that both padding check and MD check fail. In this case, some fatal errors may occurred and the signed result Z is invalid. The operation will resume directly from Step 11.

10. **Read the signed result**: Read the signed result Z_i ($i \in \{0, 1, \ldots, n-1\}$), where $n = \frac{N}{32}$, from memory block DS_Z_MEM. The memory block stores Z in little-endian byte order.
11. **Exit the operation**: Write 1 to DS_SET_FINISH_REG, then poll DS_QUERY_BUSY_REG until the software reads 0.

After the operation, all the input/output registers and memory blocks are cleared.

12.4 Memory Summary

The addresses in this section are relative to the [Digital Signature] base address provided in Table 1-4 in Chapter 1 *System and Memory*.

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

12.5 Register Summary

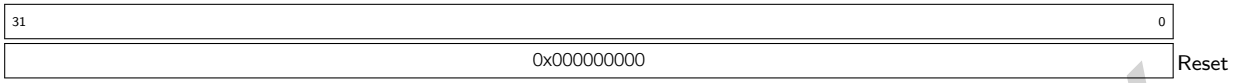
The addresses in this section are relative to the Digital Signature base address provided in Table 1-4 in Chapter 1 *System and Memory*.

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 peripheral	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 register	0x0820	W/R

12.6 Registers

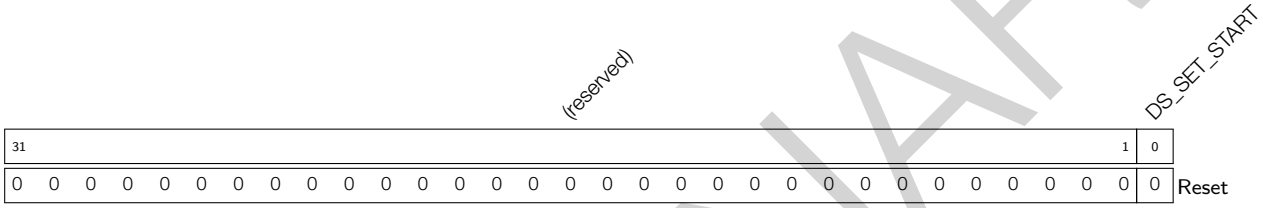
The addresses in this section are relative to the Digital Signature base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 12.1. DS_IV_*m*_REG (*m*: 0-3) (0x0630+4*m)



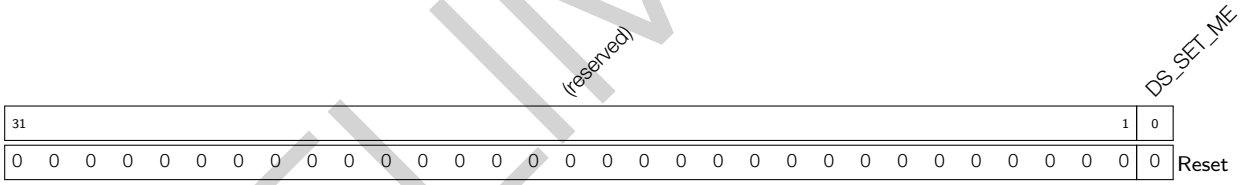
DS_IV_*m*_REG (*m*: 0-3) IV block data. (WO)

Register 12.2. DS_SET_START_REG (0x0E00)



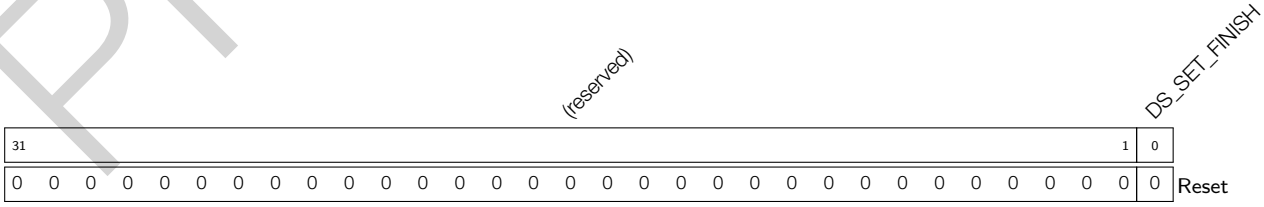
DS_SET_START Write 1 to this register to activate the DS peripheral. (WO)

Register 12.3. DS_SET_ME_REG (0x0E04)



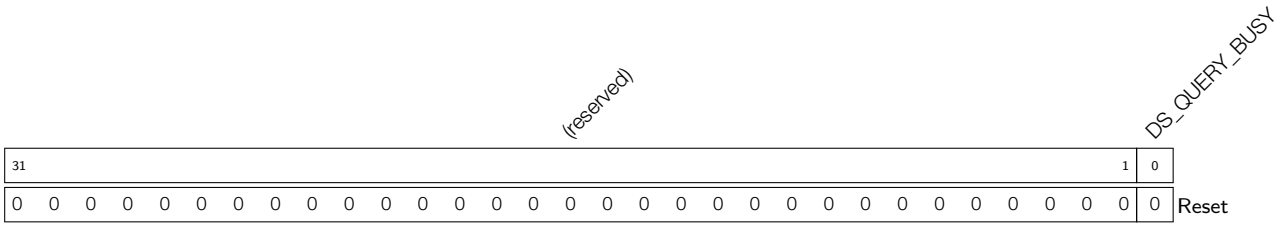
DS_SET_ME Write 1 to this register to start DS operation. (WO)

Register 12.4. DS_SET_FINISH_REG (0x0E08)



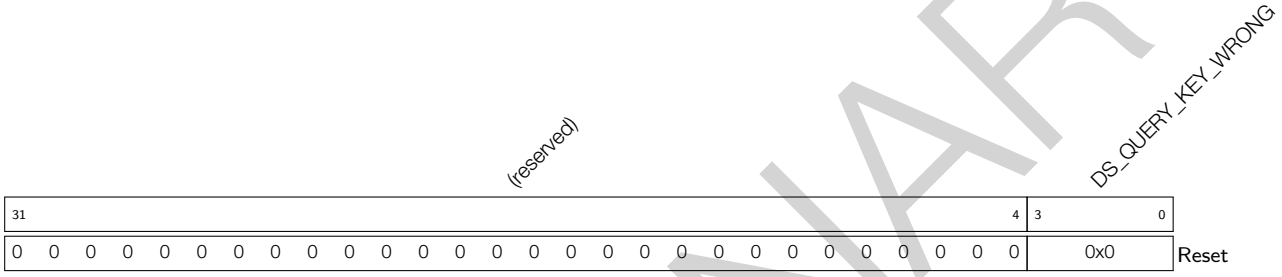
DS_SET_FINISH Write 1 to this register to end DS operation. (WO)

Register 12.5. DS_QUERY_BUSY_REG (0x0E0C)



DS_QUERY_BUSY 1: The DS peripheral is busy; 0: The DS peripheral is idle. (RO)

Register 12.6. DS_QUERY_KEY_WRONG_REG (0x0E10)



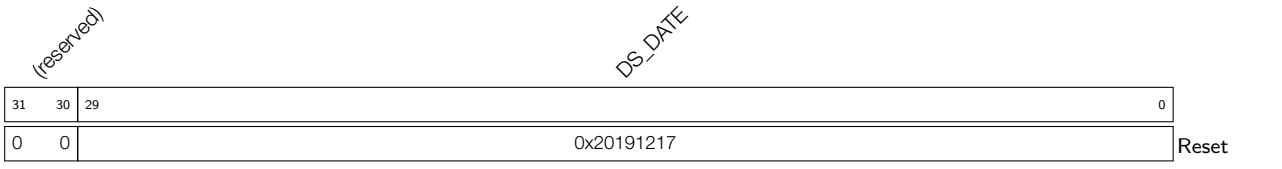
DS_QUERY_KEY_WRONG 1-15: HMAC was activated, but the DS peripheral did not successfully receive the DS_KEY from the HMAC peripheral. (The biggest value is 15); 0: HMAC is not activated. (RO)

Register 12.7. 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 12.8. DS_DATE_REG (0x0E20)



DS_DATE Version control register. (R/W)

13 External Memory Encryption and Decryption (XTS_AES)

13.1 Overview

The ESP32-S3 integrates an External Memory Encryption and Decryption module that complies with the XTS_AES standard algorithm specified in IEEE Std 1619-2007, providing security for users' application code and data stored in the external memory (flash and RAM). Users can store proprietary firmware and sensitive data (e.g., credentials for gaining access to a private network) to the external flash, or store general data to the external RAM.

13.2 Features

- General XTS_AES algorithm, compliant with IEEE Std 1619-2007
- Software-based manual encryption
- High-speed auto encryption, without software's participation
- High-speed auto decryption, without software's participation
- Encryption and decryption functions jointly determined by registers configuration, eFuse parameters, and boot mode

13.3 Module Structure

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 13-1.

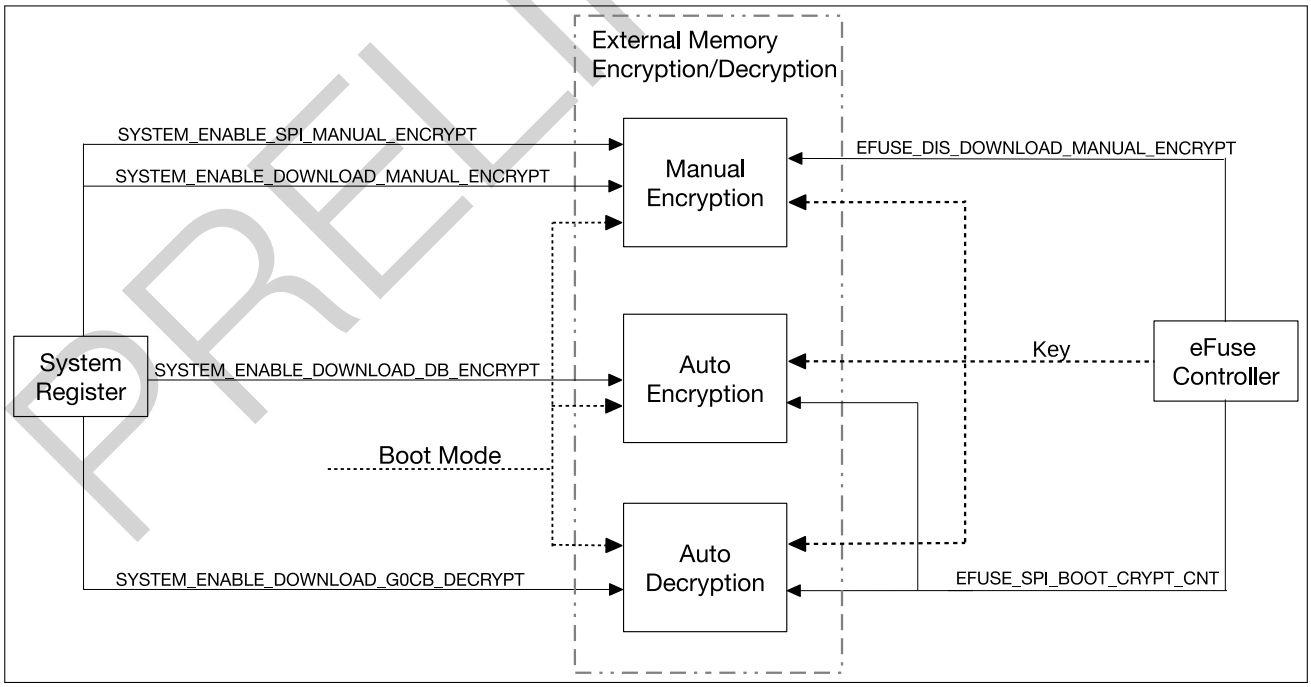


Figure 13-1. External Memory Encryption and Decryption Operation Settings

The Manual Encryption block can encrypt instructions/data which will then be written to the external flash as ciphertext via SPI1.

When the CPU writes data to the external RAM through cache, the Auto Encryption block will automatically encrypt the data first, then the data will be written to the external RAM as ciphertext.

When the CPU reads from the external flash or external RAM through cache, the Auto Decryption block will automatically decrypt the ciphertext to retrieve instructions and data.

In the System Registers (SYSREG) peripheral, the following four bits in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG are relevant to the external memory encryption and decryption:

- SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

The XTS_AES module also fetches two parameters from the peripheral 5 *eFuse Controller (eFuse) [to be added later]*, which are: EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT and EFUSE_SPI_BOOT_CRYPT_CNT.

13.4 Functional Description

13.4.1 XTS Algorithm

The manual encryption and auto encryption/decryption all use the same algorithm, i.e., XTS algorithm. During implementation, the XTS algorithm is characterized by a "data unit" of 1024 bits, which is defined in the Section *XTS-AES encryption procedure* of *XTS-AES Tweakable Block Cipher* Standard. For more information about XTS-AES algorithm, please refer to IEEE Std 1619-2007.

13.4.2 Key

The Manual Encryption block, Auto Encryption block and Auto Decryption block share the same Key when implementing XTS algorithm. The Key is provided by the eFuse hardware and cannot be accessed by users.

The Key can be either 256-bit or 512-bit long. The value and length of the Key are determined by eFuse parameters. For easier description, now define:

- Block$_A$: the BLOCK in BLOCK4 ~ BLOCK9 whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1. If Block$_A$ is true, then the 256-bit Key_A is stored in it.
- Block$_B$: the BLOCK in BLOCK4 ~ BLOCK9 whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2. If Block$_B$ is true, then the 256-bit Key_B is stored in it.
- Block$_C$: the BLOCK in BLOCK4 ~ BLOCK9 whose key purpose is EFUSE_KEY_PURPOSE_XTS_AES_128_KEY. If Block$_C$ is true, then the 256-bit Key_C is stored in it.

There are five possibilities of how the Key is generated depending on whether Block$_A$, Block$_B$ and Block$_C$ exists or not, as shown in Table 13-1. In each case, the Key can be uniquely determined by Block$_A$, Block$_B$ or Block$_C$.

Table 13-1. Key generated based on Key_A, Key_B and Key_C

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
No	No	No	0^{256}	256

Notes:

"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 bit string to another.

For more information of key purposes, please refer to Table *Key* in Chapter 5 *eFuse Controller (eFuse) [to be added later]*.

13.4.3 Target Memory Space

The target memory space refers to a continuous address space in the external memory where the first encrypted ciphertext is stored. The target memory space can be uniquely determined by three relevant parameters: type, size and base address, whose definitions are listed below.

- Type: the $type$ of the target memory space, either external flash or external RAM. Value 0 indicates external flash, while 1 indicates external RAM.
- Size: the $size$ of the target memory space, indicating the number bytes encrypted in one encryption operation, which supports 16, 32 or 64 bytes.
- Base address: the $base\_addr$ of the target memory space. It is a 30-bit physical address, with range of 0x0000_0000 ~ 0x3FFF_FFFF. It should be aligned to $size$, i.e., $base\_addr\%size == 0$.

For example, if there are 16 bytes of instruction data need to be encrypted and written to address 0x130 ~ 0x13F in the external flash, then the target space is 0x130 ~ 0x13F, type is 0 (external flash), size is 16 (bytes), and base address is 0x130.

The encryption of any length (must be multiples of 16 bytes) of plaintext instruction/data can be completed separately in multiple operations, and each operation has individual target memory space and the relevant parameters.

For Auto Encryption/Decryption blocks, these parameters are automatically defined by hardware. For Manual Encryption block, these parameters should be configured manually by users.

Note:

The "tweak" defined in Chapter *5.1 Data units and tweaks* of IEEE Std 1619-2007 is a 128-bit non-negative integer ($tweak$), which can be generated according to $tweak = type * 2^{30}$ + ($base\_addr$ & 0x3FFFFF80). The lowest 7 bits and the highest 97 bits in $tweak$ are always zero.

13.4.4 Data Padding

For Auto Encryption/Decryption blocks, data padding is automatically completed by hardware. For Manual Encryption block, data padding should be completed manually by users. The Manual Encryption block has a

registers block which consists of 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 instructions/data.

Actually, the Manual Encryption block does not care where the plaintext comes from, but only where the ciphertext will be stored. Because of the strict correspondence between plaintext and ciphertext, in order to better describe how the plaintext is stored in the register block, we assume that the plaintext is stored in the target memory space in the first place and replaced by ciphertext after encryption. Therefore, the following description no longer has the concept of "plaintext", but uses "target memory space" instead. Please note that the plaintext can come from everywhere in actual use, but users should understand how the plaintext is stored in the register block.

How mapping works between target memory space and registers:

Assume a word in the target memory space 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 13-2.

Table 13-2. 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

13.4.5 Manual Encryption Block

The Manual Encryption block is a peripheral module. It is equipped with registers and can be accessed by the CPU directly. Registers embedded in this block, the System Registers (SYSREG) peripheral, eFuse parameters, and boot mode jointly configure and use this module. Please note that the Manual Encryption block can only encrypt for storage in the external flash.

The Manual Encryption block is operational only under certain conditions. 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 can be enabled. 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 can be enabled. Otherwise, it is not operational.

Note:

- Even though the CPU can skip cache and get the encrypted instruction/data directly by reading the external memory, software can by no means access Key.

13.4.6 Auto Encryption Block

The Auto Encryption block is not a conventional peripheral, so it does not have any registers and cannot be accessed by the CPU directly. The System Registers (SYSREG) peripheral, eFuse parameters, and boot mode jointly configure and use this block.

The Auto Encryption block is operational only under certain conditions. The operating conditions are:

- In SPI Boot mode

  If the first bit or the third bit in parameter SPI_BOOT_CRYPT_CNT (3 bits) is set to 1, then the Auto Encryption block can be enabled. 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 can be enabled. Otherwise, it is not operational.

Note:

- When the Auto Encryption block is enabled, it will automatically encrypt data if the CPU writes data to the external RAM, and then the encrypted ciphertext will be written to the external RAM. The entire encryption process does not need software participation and is transparent to the cache. Software can by no means obtain the encryption Key during the process.
- When the Auto Encryption block is disabled, it will ignore the CPU's access request to cache and do not process the data. Therefore, the data will be written to the external RAM as plaintext directly.

13.4.7 Auto Decryption Block

The Auto Decryption block is not a conventional peripheral, so it does not have any registers and cannot be accessed by the CPU directly. The System Registers (SYSREG) peripheral, eFuse parameters, and boot mode jointly configure and use this block.

The Auto Decryption block is operational only under certain conditions. The operating conditions are:

- In SPI Boot mode

  If the first bit or the third bit in parameter SPI_BOOT_CRYPT_CNT (3 bits) is set to 1, then the Auto Decryption block can be enabled. 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

can be enabled. Otherwise, it is not operational.

Note:

- When the Auto Decryption block is enabled, it will automatically decrypt the ciphertext if the CPU reads instructions/data from the external memory via cache to retrieve the instructions/data. The entire decryption process does not need software participation and is transparent to the cache. Software can by no means obtain the decryption Key during the process.
- When the Auto Decryption block is disabled, it does not have any effect on the contents stored in the external memory, no matter they are encrypted or not. Therefore, what the CPU reads via cache is the original information stored in the external memory.

13.5 Software Process

When the Manual Encryption block operates, software needs to be involved in the process. The steps are as follows:

1. Configure XTS_AES:
   - Set register XTS_AES_DESTINATION_REG to $type$ = 0.
   - Set register XTS_AES_PHYSICAL_ADDRESS_REG to $base\_addr$.
   - Set register XTS_AES_LINESIZE_REG to $\frac{size}{32}$.

   For definitions of $type$, $base\_addr$ and $size$, please refer to Section 13.4.3.
2. Pad plaintext data to the registers block XTS_AES_PLAIN_*n*_REG (*n*: 0-15). For detailed information, please refer to Section 13.4.4.
   Please pad data to registers according to your actual needs, and the unused ones could be set to arbitrary values.
3. Wait for Manual Encrypt block to be idle. Poll register XTS_AES_STATE_REG until the software reads 0.
4. Trigger manual encryption by writing 1 to register XTS_AES_TRIGGER_REG.
5. Wait for the encryption process. Poll register XTS_AES_STATE_REG until the software reads 2.
   Step 1 to 5 are the steps of encrypting plaintext instructions with the Manual Encryption block using the Key.
6. Grant the ciphertext access to SPI1. Write 1 to register XTS_AES_RELEASE_REG to grant SPI1 the access to the encrypted ciphertext. After this, the value of register XTS_AES_STATE_REG will become 3.
7. Call SPI1 to write the ciphertext in the external flash (see Chapter 12 *SPI Controller (SPI) [to be added later]*).
8. Destroy the ciphertext. Write 1 to register XTS_AES_DESTROY_REG. After this, the value of register XTS_AES_STATE_REG will become 0.

Repeat above steps to meet plaintext instructions/data encryption demands.

13.6 Register Summary

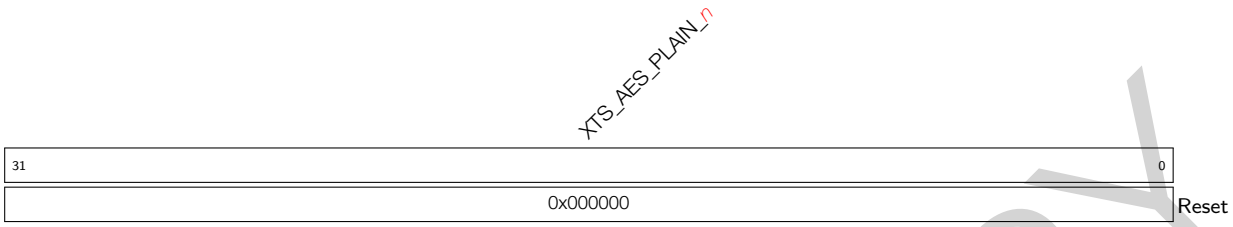
The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Address	Access
Plaintext Register Heap			
XTS_AES_PLAIN_0_REG	Plaintext register 0	0x0000	R/W
XTS_AES_PLAIN_1_REG	Plaintext register 1	0x0004	R/W
XTS_AES_PLAIN_2_REG	Plaintext register 2	0x0008	R/W
XTS_AES_PLAIN_3_REG	Plaintext register 3	0x000C	R/W
XTS_AES_PLAIN_4_REG	Plaintext register 4	0x0010	R/W
XTS_AES_PLAIN_5_REG	Plaintext register 5	0x0014	R/W
XTS_AES_PLAIN_6_REG	Plaintext register 6	0x0018	R/W
XTS_AES_PLAIN_7_REG	Plaintext register 7	0x001C	R/W
XTS_AES_PLAIN_8_REG	Plaintext register 8	0x0020	R/W
XTS_AES_PLAIN_9_REG	Plaintext register 9	0x0024	R/W
XTS_AES_PLAIN_10_REG	Plaintext register 10	0x0028	R/W
XTS_AES_PLAIN_11_REG	Plaintext register 11	0x002C	R/W
XTS_AES_PLAIN_12_REG	Plaintext register 12	0x0030	R/W
XTS_AES_PLAIN_13_REG	Plaintext register 13	0x0034	R/W
XTS_AES_PLAIN_14_REG	Plaintext register 14	0x0038	R/W
XTS_AES_PLAIN_15_REG	Plaintext register 15	0x003C	R/W
Configuration Registers			
XTS_AES_LINESIZE_REG	Configures the size of target memory space	0x0040	R/W
XTS_AES_DESTINATION_REG	Configures the type of the external memory	0x0044	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	Physical address	0x0048	R/W
Contro/Status Registers			
XTS_AES_TRIGGER_REG	Activates AES algorithm	0x004C	WO
XTS_AES_RELEASE_REG	Release control	0x0050	WO
XTS_AES_DESTROY_REG	Destroys control	0x0054	WO
XTS_AES_STATE_REG	Status register	0x0058	RO
Version Register			
XTS_AES_DATE_REG	Version control register	0x005C	RO

13.7 Registers

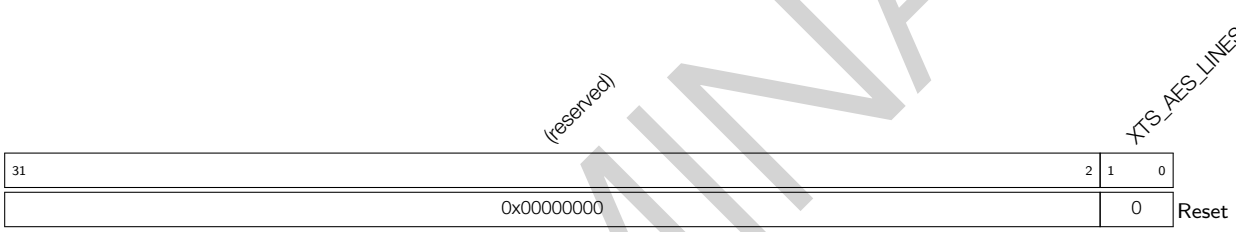
The addresses in this section are relative to the External Memory Encryption and Decryption base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 13.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0000+4n*)**



XTS_AES_PLAIN_*n* Stores *n*th 32-bit piece of plain text. (R/W)

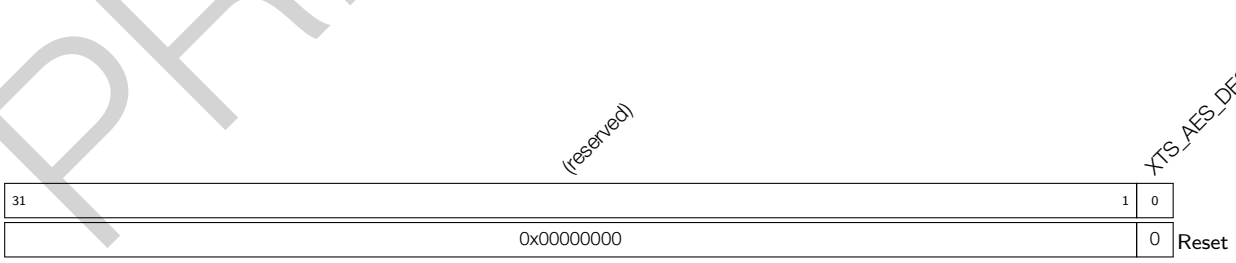
Register 13.2. XTS_AES_LINESIZE_REG (0x0040)



XTS_AES_LINESIZE Configures the data size of one encryption.

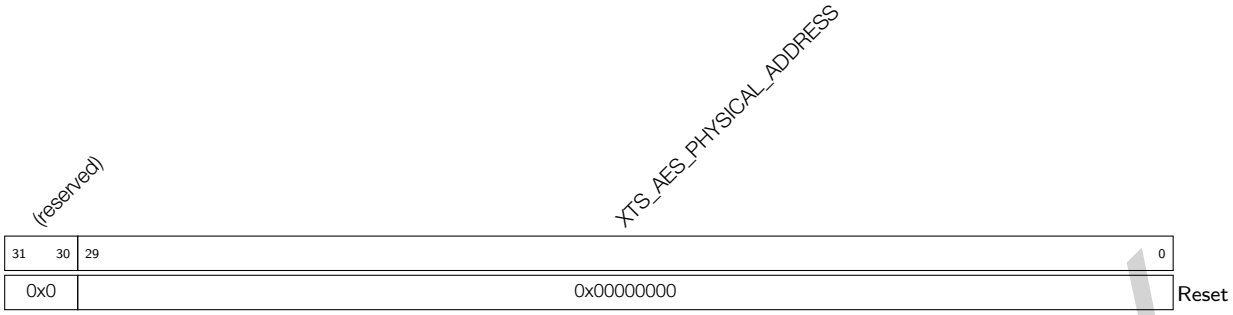
- 0: 16 bytes;
- 1: 32 bytes;
- 2: 64 bytes. (R/W)

Register 13.3. XTS_AES_DESTINATION_REG (0x0044)



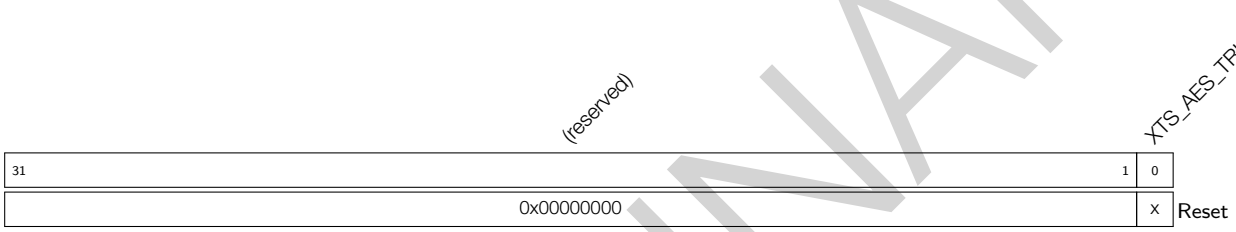
XTS_AES_DESTINATION Configures the type of the external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Errors may occur if users write 1. 0: flash; 1: external RAM. (R/W)

Register 13.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0048)



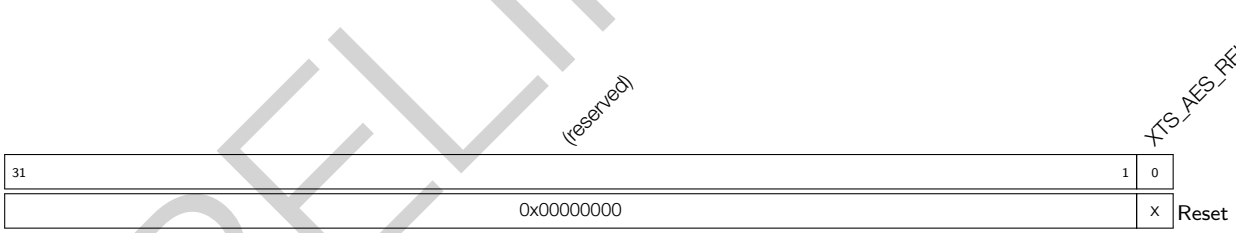
XTS_AES_PHYSICAL_ADDRESS Physical address. (R/W)

Register 13.5. XTS_AES_TRIGGER_REG (0x004C)



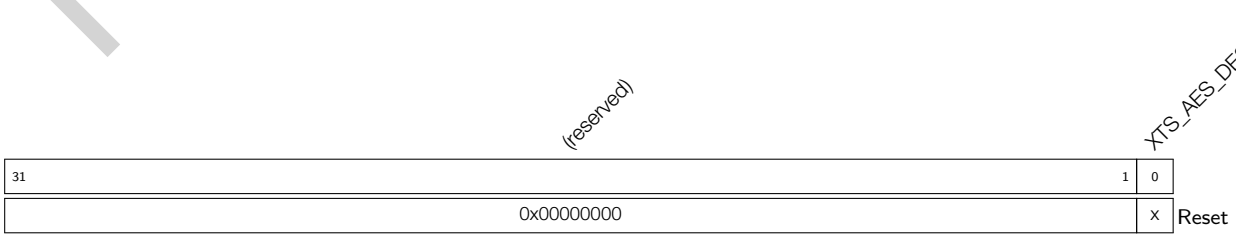
XTS_AES_TRIGGER Write 1 to enable manual encryption. (WO)

Register 13.6. XTS_AES_RELEASE_REG (0x0050)



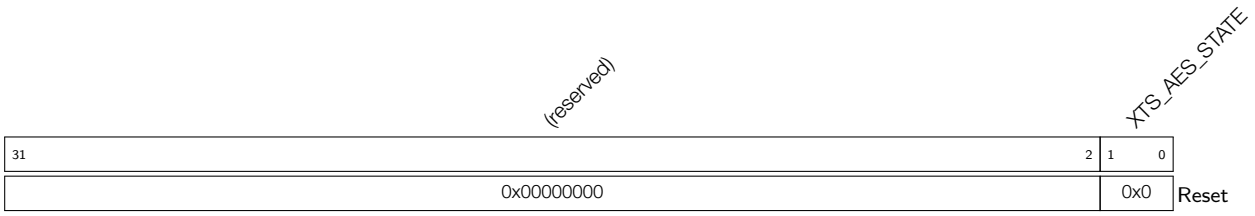
XTS_AES_RELEASE Write 1 to grant SPI1 access to encrypted result. (WO)

Register 13.7. XTS_AES_DESTROY_REG (0x0054)



XTS_AES_DESTROY Write 1 to destroy encrypted result. (WO)

Register 13.8. XTS_AES_STATE_REG (0x0058)



XTS_AES_STATE Indicates the status of the Manual Encryption block. (RO)

- 0x0 (XTS_AES_IDLE): idle;
- 0x1 (XTS_AES_BUSY): busy with encryption;
- 0x2 (XTS_AES_DONE): encryption is completed, but the encrypted result is not accessible to SPI;
- 0x3 (XTS_AES_RELEASE): encrypted result is accessible to SPI.

Register 13.9. XTS_AES_DATE_REG (0x005C)

XTS_AES_DATE Version control register. (R/W)

14 Random Number Generator (RNG)

14.1 Introduction

The ESP32-S3 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

14.2 Features

The random number generator in ESP32-S3 generates true random numbers, which means random number generated from a physical process, rather than by means of an algorithm. No number generated within the specified range is more or less likely to appear than any other number.

14.3 Functional Description

Every 32-bit value that the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the thermal noise in the system and the asynchronous clock mismatch.

Thermal noise comes from the high-speed ADC or SAR ADC or both. Whenever the high-speed ADC or SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.

When the RTC20M_CLK clock is enabled for the digital core, the random number generator will also sample RTC20M_CLK (20 MHz) as a random bit seed. RTC20M_CLK is an asynchronous clock source and it increases the RNG entropy by introducing circuit metastability. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

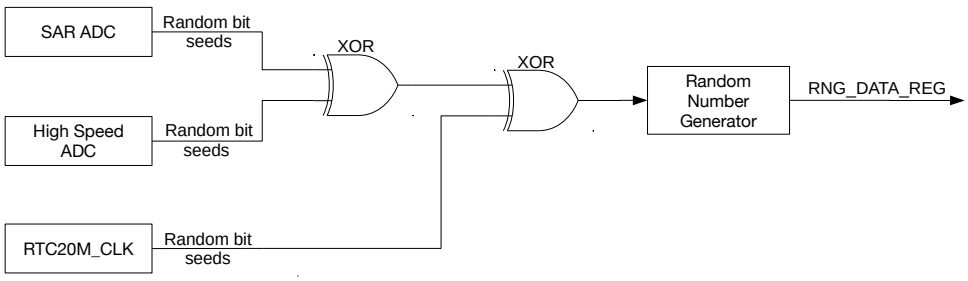


Figure 14-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle of RTC20M_CLK (20 MHz), which is generated from an internal RC oscillator (see Chapter 3 *Reset and Clock* for details). Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 500 kHz to obtain the maximum entropy.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the

high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

14.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC, high-speed ADC, or RTC20M_CLK is enabled. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 13 *On-Chip Sensors and Analog Signal Processing [to be added later]*.
- High-speed ADC is enabled automatically when the Wi-Fi or Bluetooth modules is enabled.
- RTC20M_CLK is enabled by setting the RTC_CNTL_DIG_CLK20M_EN bit in the RTC_CNTL_CLK_CONF_REG register.

Note:
Note that, when the Wi-Fi module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.

When using the random number generator, read the RNG_DATA_REG register multiple times until sufficient random numbers have been generated. Ensure the rate at which the register is read does not exceed the frequencies described in section 14.3 above.

14.5 Register Summary

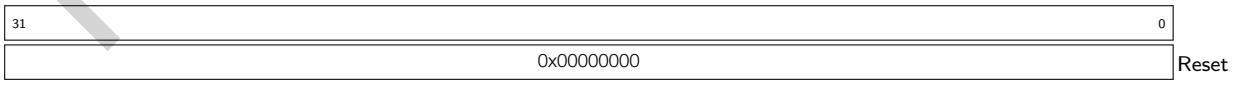
The address in the following table is relative to the random number generator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x0110	RO

14.6 Register

The address in this section is relative to the random number generator base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 14.1. RNG_DATA_REG (0x0110)



RNG_DATA Random number source. (RO)

15 Two-wire Automotive Interface® (TWAI)

15.1 Overview

The Two-wire Automotive Interface (TWAI)® is a multi-master, multi-cast communication protocol with error detection and signaling and inbuilt message priorities and arbitration.The TWAI protocol is suited for automotive and industrial applications (see Section 15.3 for more details).

ESP32-S3 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.

15.2 Features

ESP32-S3 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 (no acknowledgment required during data transmission)
- 64-byte Receive FIFO
- Special transmissions
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self Reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports single and dual filter modes)
- Error detection and handling
  - Error Counters
  - Configurable Error Warning Limit
  - Error Code Capture
  - Arbitration Lost Capture

15.3 Functional Protocol

15.3.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows nodes to exchange messages in a latency bounded manner. A TWAI bus has the 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 implemented in this channel, so extra channels (e.g., clock or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a dominant or recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting data in a dominant state will always override another node transmitting data in a recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential pair or a single wire).

Bit Stuffing: Certain fields of TWAI messages are bit-stuffed. A transmitter that transmits five consecutive bits of the same value should automatically insert a complementary bit. Likewise, a receiver that receives five consecutive bits should treat the next bit as a stuffed bit. Bit stuffing is applied to the following fields: SOF, arbitration field, control field, data field, and CRC sequence (see Section 15.3.2 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 15.3.3 for more details).

Multi-master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before 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 transmitters if they have yet to lose arbitration.
- All nodes that are not transmitters are receivers.

15.3.2 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

15.3.2.1 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 for nodes to request a data frame with the same identifier from another node, thus they do not contain any data bytes. However, data frames and remote frames share many common fields. Figure 15-1 illustrates the fields and sub-fields of different frames and formats.

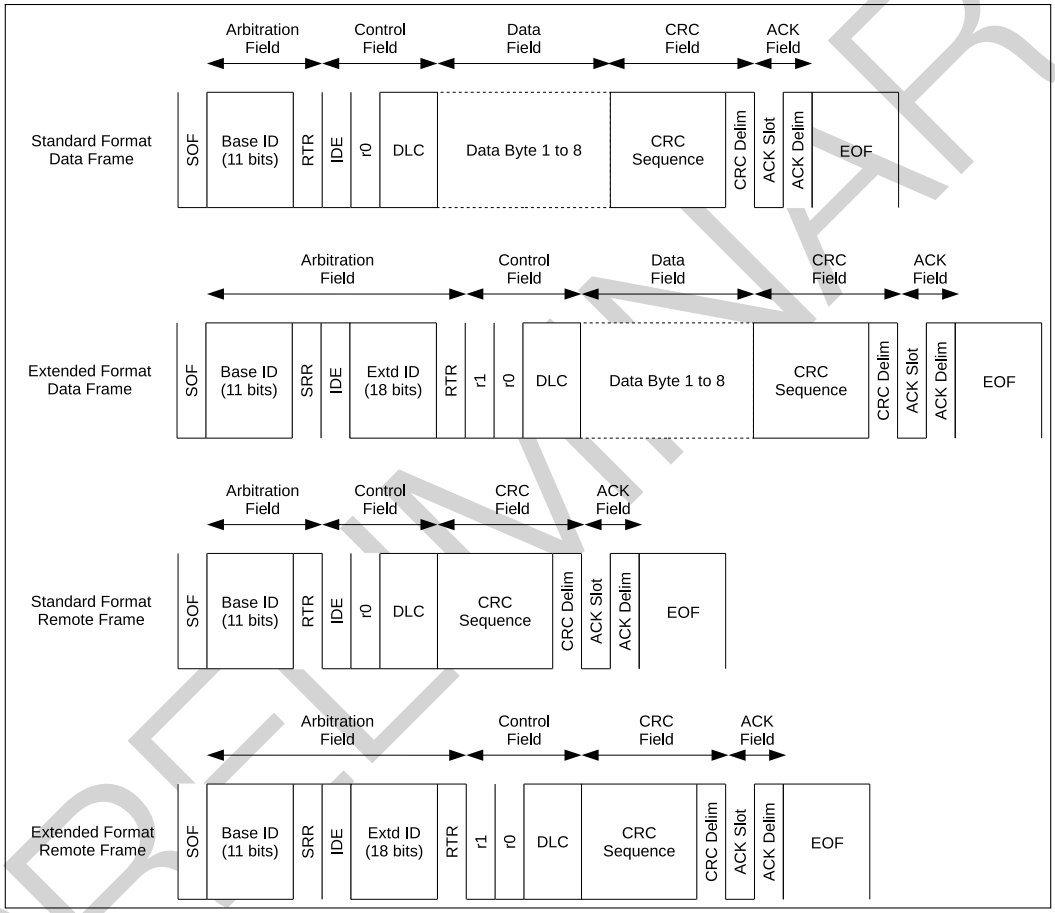


Figure 15-1. Bit Fields in Data Frames and Remote Frames

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 while 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 switch to be a receiver.

The arbitration field primarily consists of the frame identifier that is transmitted from the most significant bit first. Given that a dominant bit represents a logical 0, and a recessive bit represents a logical 1:

- A frame with the smallest ID value will always win arbitration.

- Given the same ID and format, data frames will always prevail over remote 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 from the most significant bit first.

Data Field

The data field contains the actual payload data bytes of a data frame. Remote frames do not contain a data field.

CRC Field

The CRC field primarily consists of a CRC sequence. The CRC sequence is a 15-bit cyclic redundancy code calculated form the de-stuffed contents (everything from the SOF to the end of the data field) of a data or remote frame.

ACK Field

The ACK field primarily consists of an ACK Slot and an ACK Delim. The ACK field is mainly intended for the receiver to indicate to a transmitter that it has received an effective message.

Table 15-1. Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit identifier for SFF, or the first 11-bits of the 29-bit identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a data frame (dominant) or a remote frame (recessive). This means that a remote frame will always lose arbitration to a data frame given they have the same ID.
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
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 bit (reserved bit 1) is always dominant.
r0	The r0 bit (reserved bit 0) is always dominant.
DLC	The DLC (Data Length Code) is 4-bit long and should contain any value from 0 to 8. Data frames use the DLC to indicate the number of data bytes in the data frame. Remote frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of data frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted from the most significant bit first.

Data/Remote Frames	Description
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.
CRC Delim	The CRC Delim (CRC Delimiter) is a single recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) is intended for receiver nodes to indicate that the data or remote frame was received without an issue. The transmitter node will send a recessive bit in the ACK Slot and receiver nodes should override the ACK Slot with a dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimiter) is a single recessive bit.
EOF	The EOF (End of Frame) marks the end of a data or remote frame, and consists of seven recessive bits.

15.3.2.2 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 from 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 15.3.3 for more details). The following Figure 15-2 shows different fields of an error frame:

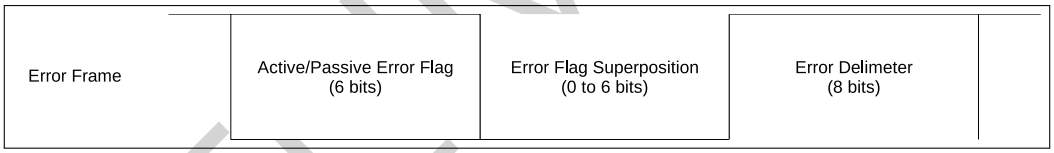


Figure 15-2. Fields of an Error Frame

Table 15-2. Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of 6 dominant bits and the Passive Error Flag consisting of 6 recessive bits (unless overridden by dominant bits of other nodes). Active Error Flags are sent by error active nodes, whilst Passive Error Flags are sent by error passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first recessive bit is detected (i.e., the first it of the Delimiter).
Error Delimeter	The Delimiter field marks the end of the error/overload frame, and consists of 8 recessive bits.

Overload Frames

An overload frame has the same bit fields as an error frame containing an Active Error Flag. The key difference is in the conditions that can trigger the transmission of an overload frame. Figure 15-3 below shows the bit fields of an overload frame.

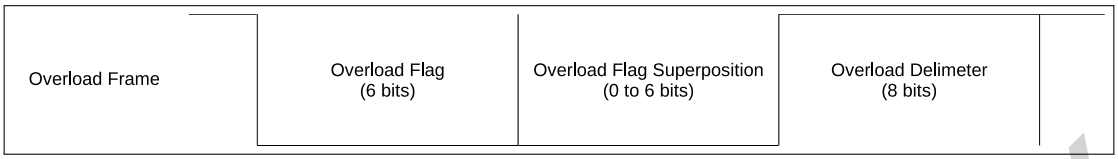


Figure 15-3. Fields of an Overload Frame

Table 15-3. Overload Frame

Overload Flag	Description
Overload Flag	Consists of 6 dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimiter	Consists of 8 recessive bits. Same as an Error Delimiter.

Overload frames will be transmitted under the following conditions:

1. A receiver requires a delay of the next data or remote frame.
2. A dominant bit is detected at the first and second bit of intermission.
3. A dominant bit is detected at the eighth (last) bit of an Error Delimiter. Note that in this case, TEC and REC will not be incremented (see Section 15.3.3 for more details).

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.

15.3.2.3 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 15-4 shows the fields within an Interframe Space:

Table 15-4. Interframe Space

Interframe Space	Description
Intermission	The Intermission consists of 3 recessive bits.

Interframe Space	Description
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.

15.3.3 TWAI Errors

15.3.3.1 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 rules).

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.

Format Error

A Format 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.

ACK Error

An ACK Error occurs when a transmitter does not detect a dominant bit at the ACK Slot.

15.3.3.2 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.

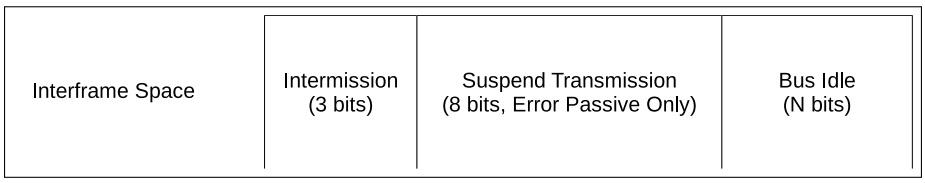


Figure 15-4. The Fields within an Interframe Space

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit anything).

15.3.3.3 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 is increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a receiver detects a dominant bit as the first bit after sending an Error Flag, the REC is increased by 8.
3. When a transmitter sends an Error Flag, the TEC is increased by 8. However, the following scenarios are exempt from 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 TEC 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. A node can tolerate up to 7 consecutive dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit (when sending an Active Error Flag or Overload Flag), or the 8th consecutive dominant bit following a Passive Error Flag, a transmitter will increase its TEC by 8 and a receiver will increase its REC by 8. Every additional 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 invalid 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.

15.3.4 TWAI Bit Timing

15.3.4.1 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 the 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 15-5 illustrates the segments within a single Nominal Bit Time.

TWAI controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed. If 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.

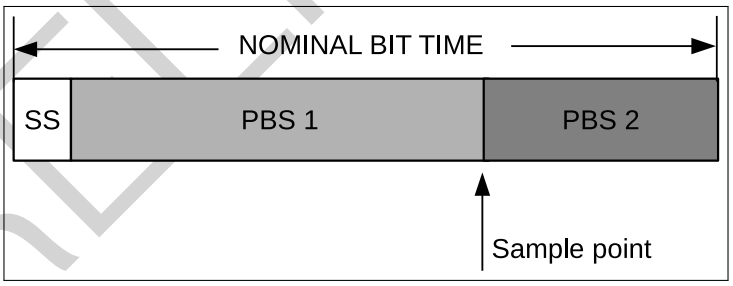


Figure 15-5. Layout of a Bit

Table 15-5. Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.

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.

15.3.4.2 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 when the bus is idle (i.e., the first SOF bit after Bus Idle). All nodes will restart their internal bit timings so that the recessive to dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on recessive to dominant edges 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 programmable.

- When the magnitude of the Phase Error (**e**) is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by the **e** number of Time Quanta. This has 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.

15.4 Architectural Overview

The major functional blocks of the TWAI controller are shown in Figure 15-6.

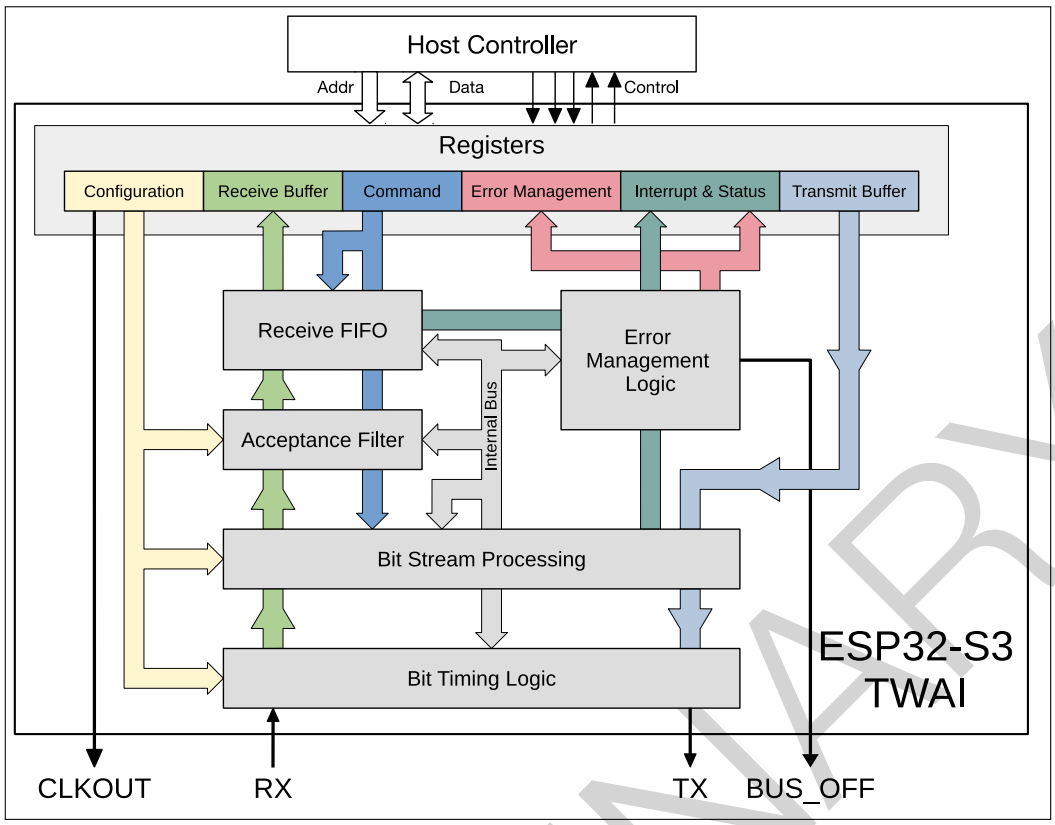


Figure 15-6. TWAI Overview Diagram

15.4.1 Registers Block

The ESP32-S3 CPU accesses peripherals using 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration items for the TWAI controller such as bit rates, operation mode, Acceptance Filter etc. Configuration registers can only be modified whilst the TWAI controller is in Reset Mode (See Section 15.5.1).

Command Registers

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation Mode (see section 15.5.1).

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 of Receive FIFO, whose first message will be mapped into the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset Mode, all reads and writes to the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation Mode:
  - All reads to the address range maps to the Receive Buffer registers.
  - All writes to the address range maps to the Transmit Buffer registers.

15.4.2 Bit Stream Processor

The Bit Stream Processing (BSP) module frames 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).

15.4.3 Error Management Logic

The Error Management Logic (EML) module updates 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.

15.4.4 Bit Timing Logic

The Bit Timing Logic (BTL) module transmits and receives messages at the configured bit rate. The BTL module also handles synchronization of out of phase bits so 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.

15.4.5 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 two separate filters (dual filter mode).

15.4.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (inside the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13-bytes). When the Receive FIFO is full (or does not have enough space to store the 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.

15.5 Functional Description

15.5.1 Modes

The ESP32-S3 TWAI controller has two working modes: Reset Mode and Operation Mode. Reset Mode and Operation Mode are entered by setting or clearing the TWAI_RESET_MODE bit.

15.5.1.1 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 signals). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will not be able to receive any messages either.

15.5.1.2 Operation Mode

In operation mode, the TWAI controller connects to the bus and write-protect all configuration registers to ensure consistency during operation. When in Operation Mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operation sub-mode the TWAI controller was configured with. The TWAI controller supports the following operation sub-modes:

- **Normal Mode:** The TWAI controller can transmit and receive messages including error signaling (such as error and overload Frames).
- **Self-test Mode:** Self-test mode is similar to normal Mode, but the TWAI controller will consider the transmission of a data or RTR frame successful and do not generate ACK error 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 monitoring.

Note that when exiting Reset Mode (i.e., entering Operation Mode), the TWAI controller must wait for 11 consecutive recessive bits to occur before being able to fully connect the TWAI bus (i.e., be able to transmit or receive).

15.5.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset Mode. The bit rate is configured using TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 15-6 illustrates the bit fields of TWAI_BUS_TIMING_0_REG.

Table 15-6. Bit Information of TWAI_BUS_TIMING_0_REG (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	Reserved	BRP.12	.....	BRP.1	BRP.0

Notes:

- BRP: The TWAI Time Quanta clock is derived from 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 cycle and t_{CLK} is APB clock cycle:
  $t_{Tq} = 2 \times t_{CLK} \times (2^{12} \times BRP.12 + 2^{11} \times BRP.11 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$
- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1)

The following Table 15-7 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 15-7. Bit Information of TWAI_BUS_TIMING_1_REG (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined 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 to filter spikes on the bus line.

15.5.3 Interrupt Management

The ESP32-S3 TWAI controller provides eight interrupts, each represented by a single bit in the TWAI_INT_RAW_REG. For a particular interrupt to be triggered, the corresponding enable bit in TWAI_INT ENA_REG must be set.

The TWAI controller provides the following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt
- Arbitration Lost Interrupt
- Bus Error Interrupt
- Bus Status 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, except for the Receive Interrupt which can only be cleared when all the messages are released by setting the TWAI_RELEASE_BUF bit.

15.5.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to be read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_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.

15.5.3.2 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 resent.)
- 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.

15.5.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST and TWAI_BUS_OFF_ST bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_ERR_ST and TWAI_BUS_OFF_ST at the moment when the EWI is triggered.

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
  - If the TWAI controller was previously in the Bus Off Recovery state, it indicates that Bus Recovery has completed successfully.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).
- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

15.5.3.4 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 by the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI could be triggered again when all received messages (valid or overrun) have been cleared.

15.5.3.5 Error Passive Interrupt (TXI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller switches from Error Active to Error Passive, or vice versa.

15.5.3.6 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 reading this register through the CPU).

15.5.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever TWAI controller detects an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).

15.5.3.8 Bus Status Interrupt (BSI)

The Bus Status Interrupt (BSI) is triggered whenever TWAI controller is switching between receive/transmit status and idle status. When a BSI occurs, the current status of TWAI controller can be measured by reading TWAI_RX_ST and TWAI_TX_ST in TWAI_STATUS_REG register.

15.5.4 Transmit and Receive Buffers

15.5.4.1 Overview of Buffers

Table 15-8. 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

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
TWAI Address	Content	TWAI Address	Content
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 15-8 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 access the Transmit Buffer registers, and CPU read operations 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 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 to clear the Receive Buffer registers. If there are still messages in the Receive FIFO, the Receive Buffer registers will map the first message again.

15.5.4.2 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 15-9.

Table 15-9. TX/RX Frame Information (SFF/EFF) TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	DLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

1. FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when FF bit is 1, and SFF when FF bit is 0.
2. RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a data frame or a remote frame. The message is a remote frame when the RTR bit is 1, and a data frame when the RTR bit is 0.
3. X: Don't care, can be any value.
4. DLC: The Data Length Code (DLC) field specifies the number of data bytes for a data frame, or the number of data bytes to request in a remote frame. TWAI data frames are limited to a maximum payload of 8 data bytes, and thus the DLC should range anywhere from 0 to 8.

15.5.4.3 Frame Identifier

The Frame Identifier fields is two-byte (11-bit) long if the message is SFF, and four-byte (29-bit) long if the message is EFF.

The Frame Identifier fields for an SFF (11-bit) message is shown in Table 15-10-15-11.

Table 15-10. TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

Table 15-11. TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self reception functionality (or together with self-test functionality).

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 15-12-15-15.

Table 15-12. 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 15-13. 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 15-14. 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 15-15. TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self reception functionality (or together with self-test functionality).

2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self reception functionality (or together with self-test functionality).

15.5.4.4 Frame Data

The Frame Data field contains the payloads of transmitted or received data frame, and can range from 0 to eight bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than eight, the number of valid bytes would still be limited to eight. Remote frames do not have data payloads, thus their Frame Data fields will be unused.

For example, when transmitting a data frame with five bytes, the CPU should write five to the DLC field, and then write data to the corresponding register of the first to the fifth data field. Likewise, when receiving a data frame with a DLC of five data bytes, only the first to the fifth data byte will contain valid payload data for the CPU to read.

15.5.5 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 three to 13 bytes of space in the Receive FIFO, and their endianness is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO.

When the TWAI controller receives 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:

- The free space left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- When data in the Receive FIFO overruns for the first time, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The 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 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.

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) since the TWAI Controller only need to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset Mode, since they share the same address spaces as the Transmit Buffer and Receive Buffer registers.

The configuration registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Acceptance Code value specifies a bit pattern which each filtered bit of the message must match in order for the message to be accepted. The Acceptance Mask Value is able to mask out certain bits of the Code value (i.e., set as “Don’t Care” bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 15-7.

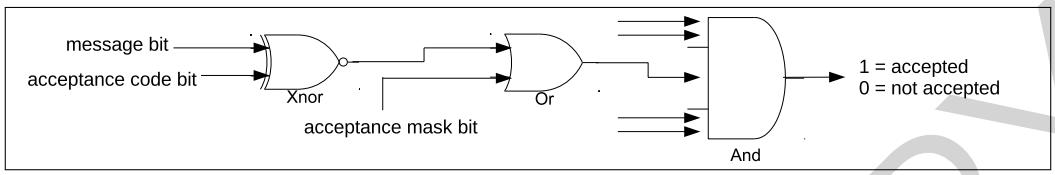


Figure 15-7. Acceptance Filter

The TWAI controller Acceptance Filter allows the 32-bit Acceptance Code and Mask Values to either define a single filter (i.e., Single Filter Mode), or two filters (i.e., Dual Filter Mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on whether Single Filter Mode is enabled, and the received message format (i.e., SFF or EFF).

15.5.5.1 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 15-8 illustrates how the 32-bit code and mask values will be interpreted under Single Filter Mode.

15.5.5.2 Dual FIlter Mode

Dual Filter Mode is enabled by clearing 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 filter 2. Under Dual Filter Mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of a data or remote frame:

- SFF
  - The entire 11-bit ID

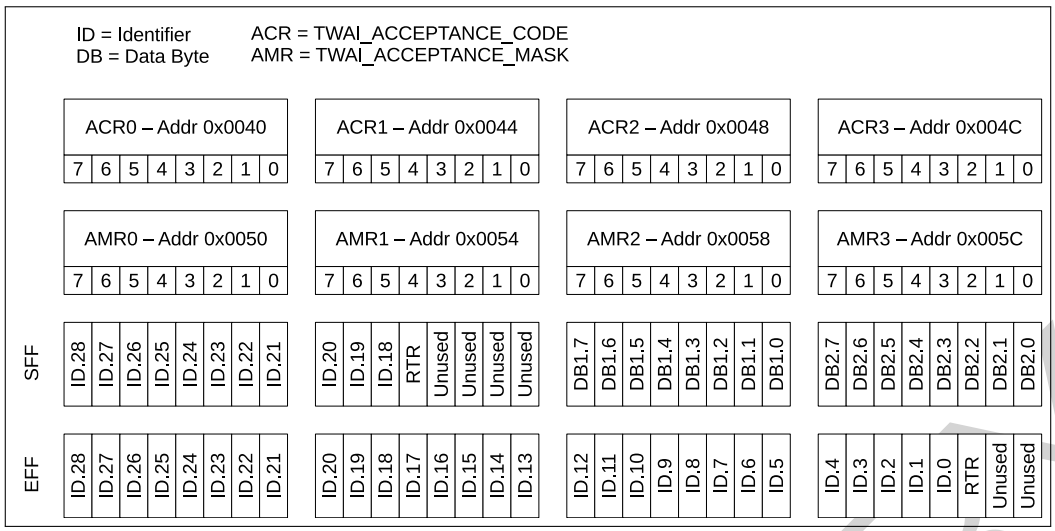


Figure 15-8. Single Filter Mode

- RTR bit
- Data byte 1 (for filter 1 only)

- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 15-9 illustrates how the 32-bit code and mask values will be interpreted in Dual Filter Mode.

15.5.6 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 determines the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in the TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and they can be read by the CPU anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn the user of the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_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 15.5.3). The following figure 15-10 shows the relation between the error states, values and bits, and error state related interrupts.

15.5.6.1 Error Warning Limit

The Error Warning Limit (EWL) feature is a configurable threshold value for the TEC and REC, which will trigger an interrupt when exceeded. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in 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

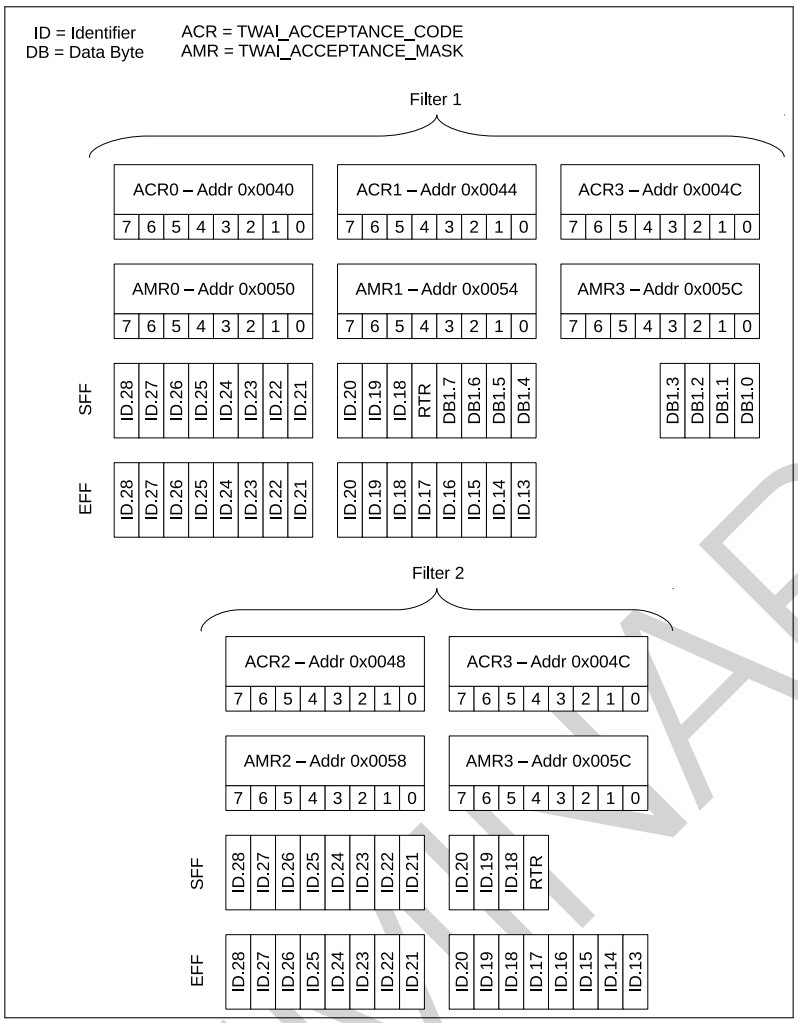


Figure 15-9. Dual Filter Mode

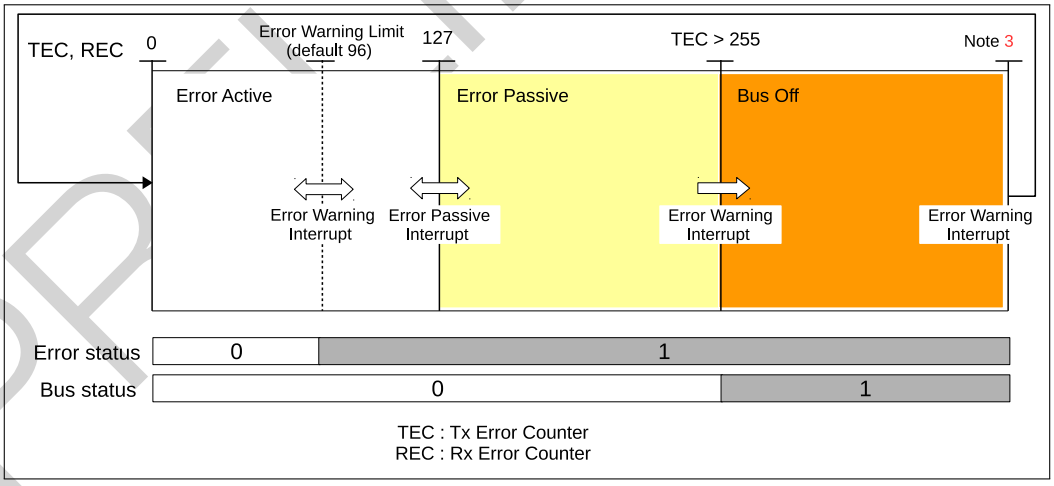


Figure 15-10. Error State Transition

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.

15.5.6.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

15.5.6.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_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 when the TWAI controller observes 11 consecutive recessive bits. When Bus-Off Recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

15.5.7 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in 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 15-16 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 15-16. Bit Information of TWAI_ERR_CODE_CAP_REG (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error: 00 for bit error, 01 for format error, 10 for stuff error, 11 for other types of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error occurred: 0 for 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 15-17 shows how to interpret the SEG.0 to SEG.4 bits.

Table 15-17. Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	ACK slot
1	1	0	1	1	ACK delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit SRTR: under Standard Frame Format.
- Bit IDE: Identifier Extension Bit, 0 for Standard Frame Format.

15.5.8 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in 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 15-18 illustrates bits and fields of the TWAI_ERR_CODE_CAP_REG whilst Figure 15-11 illustrates the bit positions of a TWAI message.

Table 15-18. Bit Information of TWAI_ARB LOST CAP_REG (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

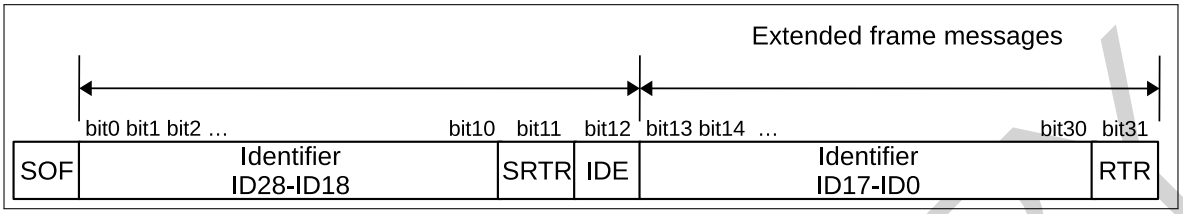


Figure 15-11. Positions of Arbitration Lost Bits

15.6 Register Summary

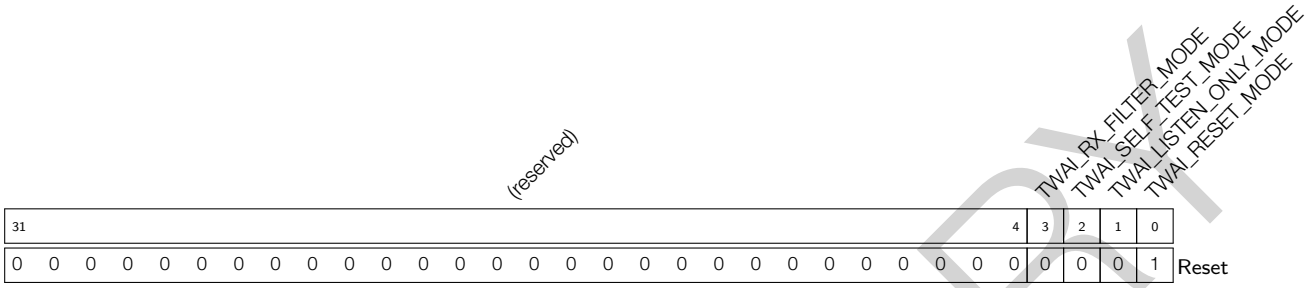
'|' here means separate line. The left describes the access in Operation Mode. The right belongs to Reset Mode. The addresses in this section are relative to the [Two-wire Automotive Interface] base address provided in Table 1-4 in Chapter 1 *System and Memory*.

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
Contro Registers			
TWAI_CMD_REG	Command Register	0x0004	WO
Status Register			
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

15.7 Registers

'I' here means separate line. The left describes the access in Operation Mode. The right belongs to Reset Mode with red color. The addresses in this section are relative to the Two-wire Automotive Interface base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 15.1. TWAI_MODE_REG (0x0000)



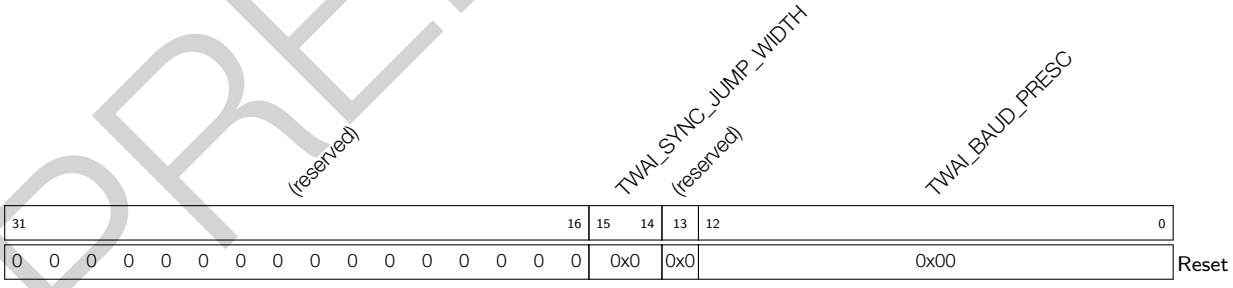
TWAI_RESET_MODE This bit is used to configure the operation mode of the TWAI Controller. 1: Reset mode; 0: Operation mode (R/W)

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)

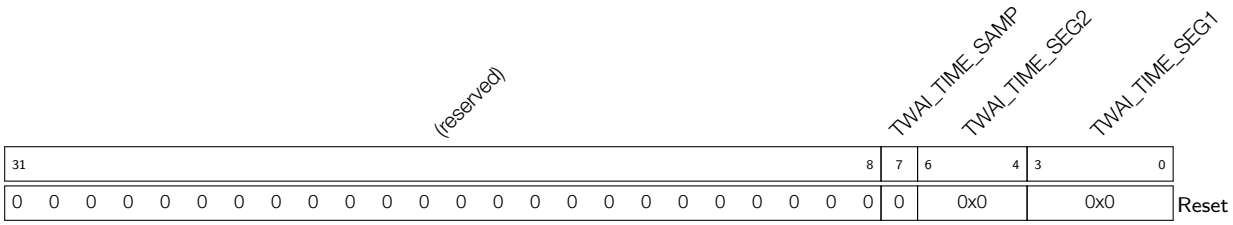
Register 15.2. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Baud Rate Prescaler value, determines the frequency dividing ratio. (RO | R/W)

TWAI_SYNC_JUMP_WIDTH Synchronization Jump Width (SJW), 1 ~ 14 Tq wide. (RO | R/W)

Register 15.3. TWAI_BUS_TIMING_1_REG (0x001C)

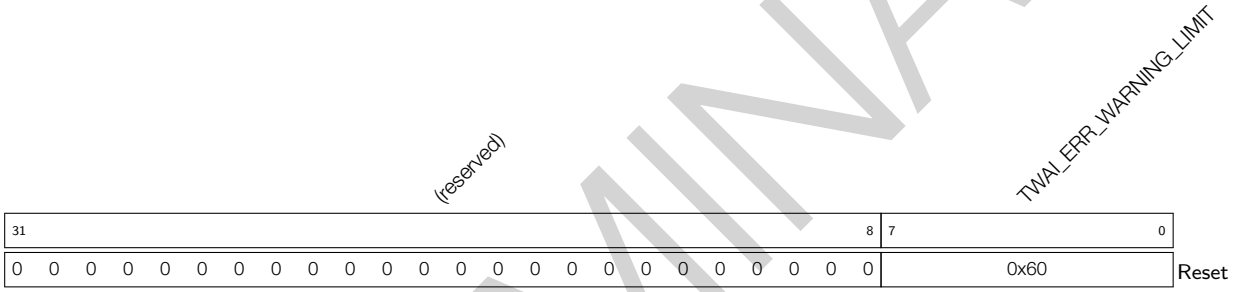


TWAI_TIME_SEG1 The width of PBS1. (RO | R/W)

TWAI_TIME_SEG2 The width of PBS2. (RO | R/W)

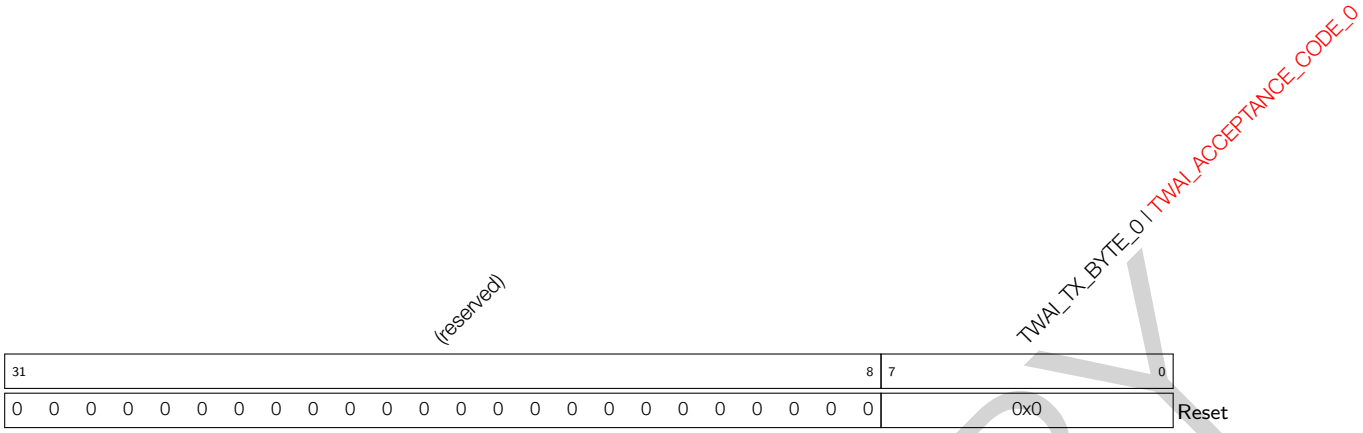
TWAI_TIME_SAMP The number of sample points. 0: the bus is sampled once; 1: the bus is sampled three times (RO | R/W)

Register 15.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Error warning threshold. In the case when any of an 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)

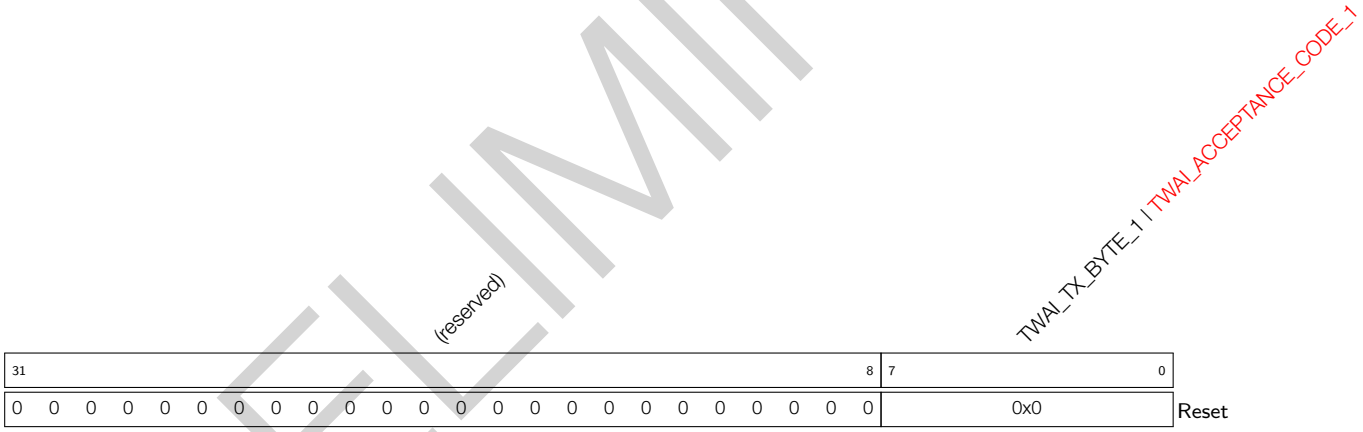
Register 15.5. TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 Stored the 0th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_0 Stored the 0th byte of the filter code in reset mode. (R/W)

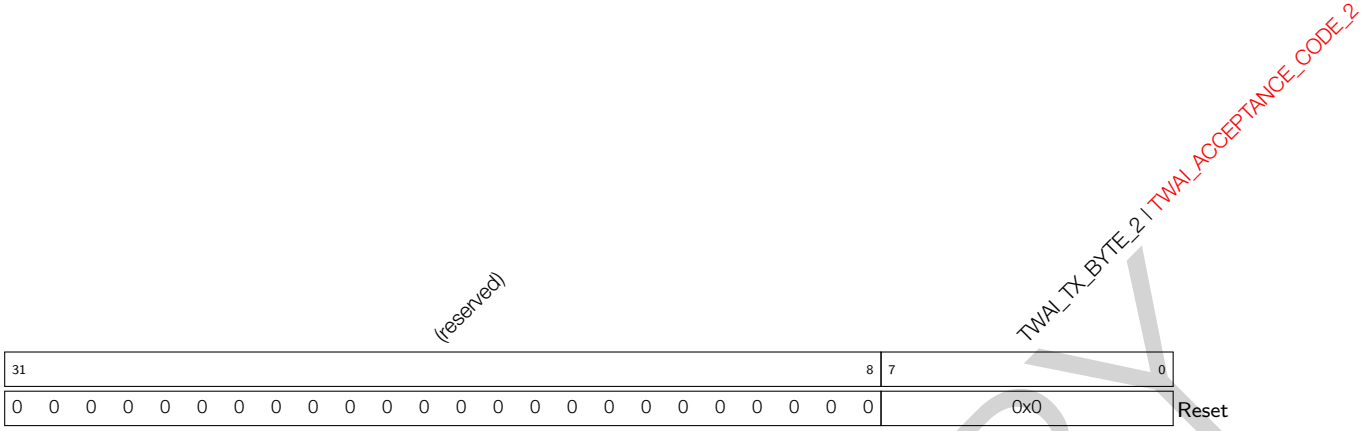
Register 15.6. TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 Stored the 1st byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_1 Stored the 1st byte of the filter code in reset mode. (R/W)

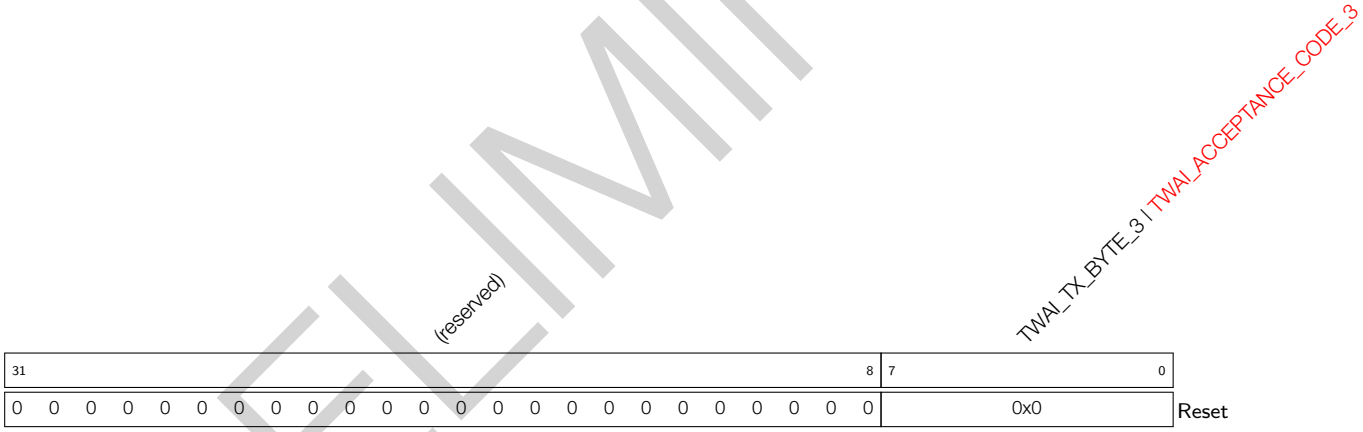
Register 15.7. TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 Stored the 2nd byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_2 Stored the 2nd byte of the filter code in reset mode. (R/W)

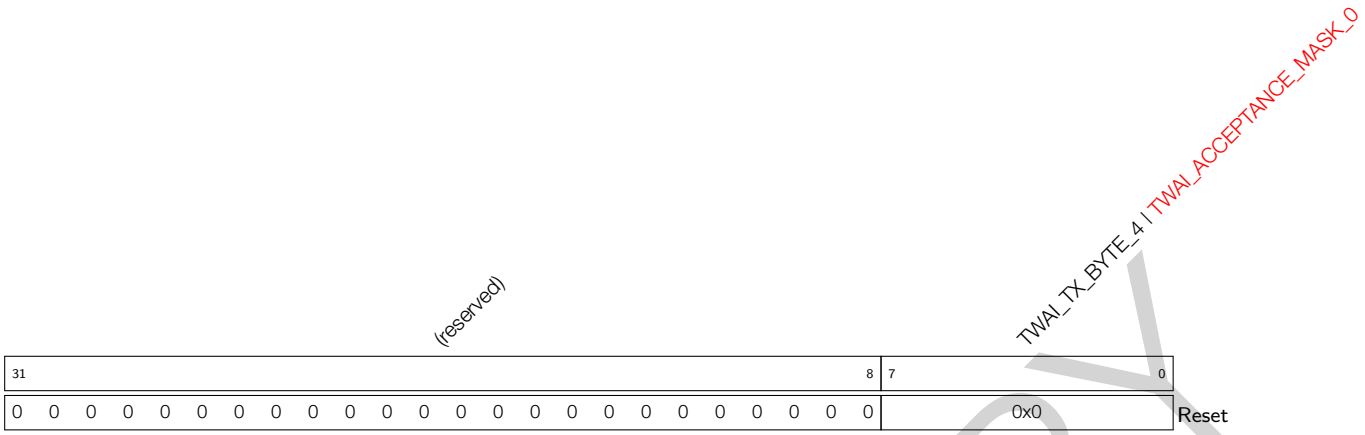
Register 15.8. TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 Stored the 3rd byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_3 Stored the 3rd byte of the filter code in reset mode. (R/W)

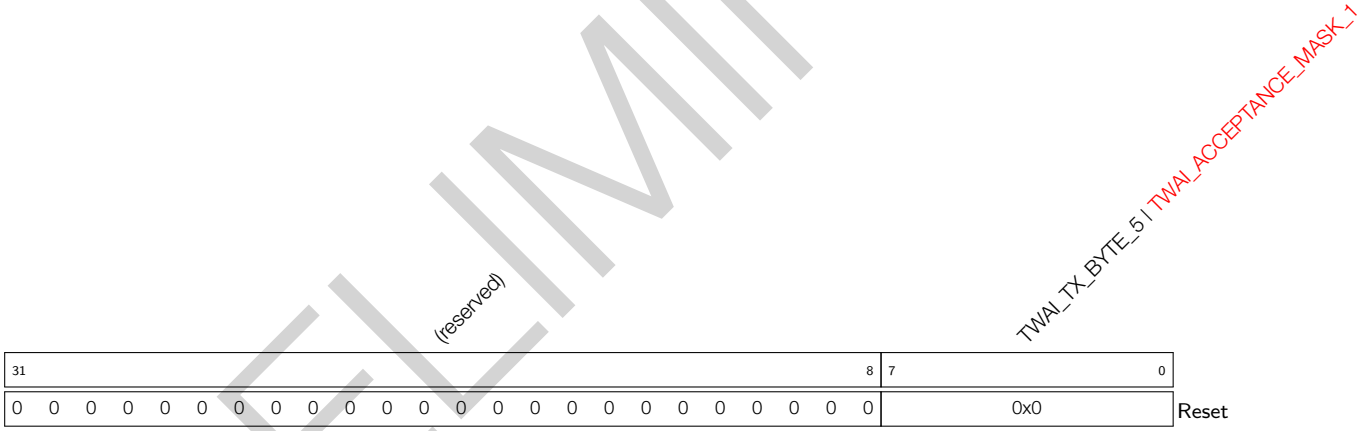
Register 15.9. TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 Stored the 4th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_0 Stored the 0th byte of the filter code in reset mode. (R/W)

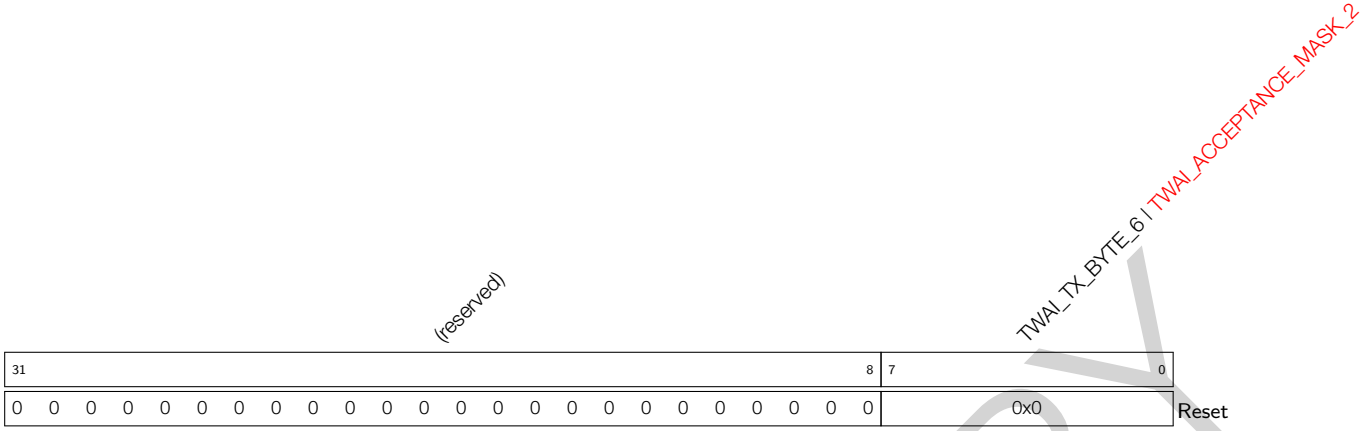
Register 15.10. TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 Stored the 5th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_1 Stored the 1st byte of the filter code in reset mode. (R/W)

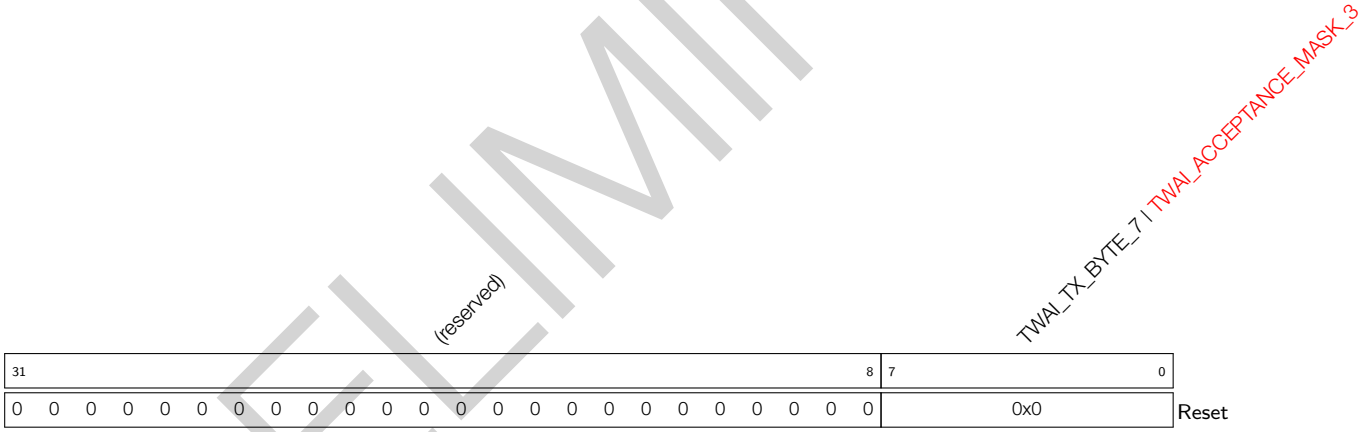
Register 15.11. TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 Stored the 6th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_2 Stored the 2nd byte of the filter code in reset mode. (R/W)

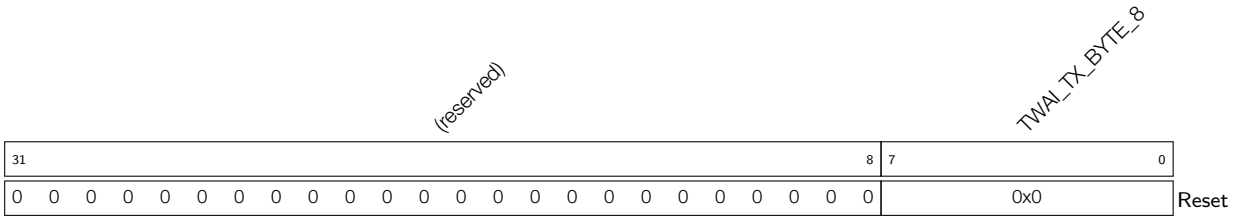
Register 15.12. TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 Stored the 7th byte information of the data to be transmitted in operation mode. (WO)

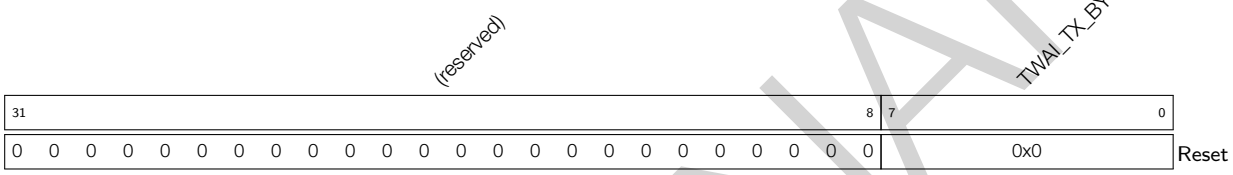
TWAI_ACCEPTANCE_MASK_3 Stored the 3rd byte of the filter code in reset mode. (R/W)

Register 15.13. TWAI_DATA_8_REG (0x0060)



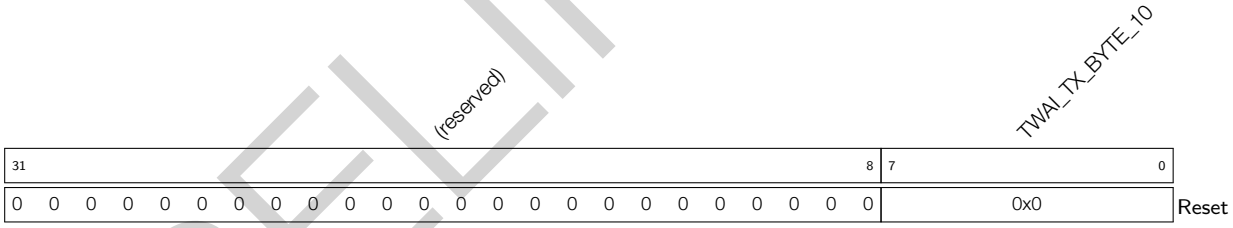
TWAI_TX_BYTE_8 Stored the 8th byte information of the data to be transmitted in operation mode. (WO)

Register 15.14. TWAI_DATA_9_REG (0x0064)



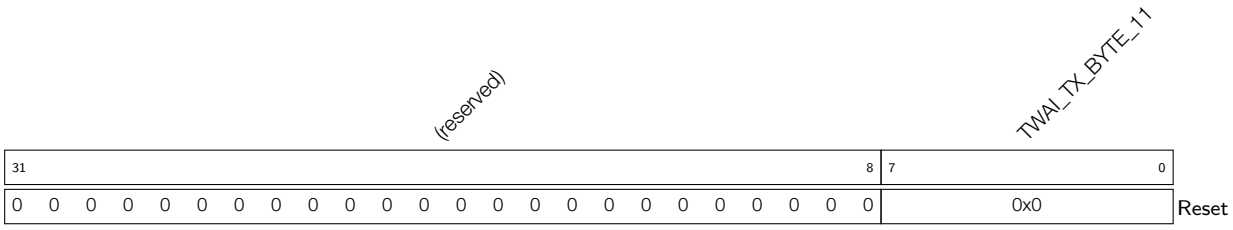
TWAI_TX_BYTE_9 Stored the 9th byte information of the data to be transmitted in operation mode. (WO)

Register 15.15. TWAI_DATA_10_REG (0x0068)



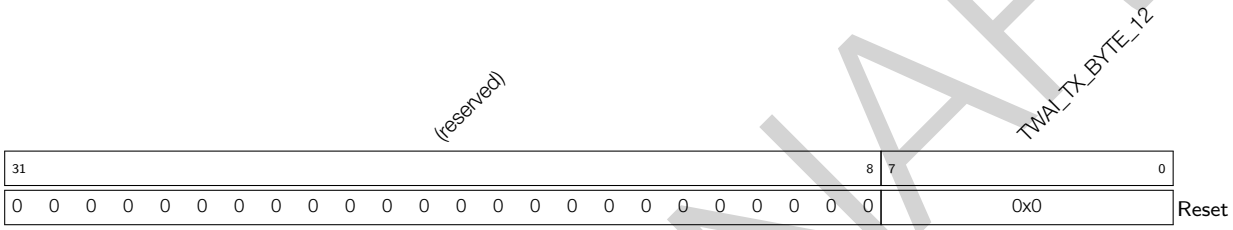
TWAI_TX_BYTE_10 Stored the 10th byte information of the data to be transmitted in operation mode. (WO)

Register 15.16. TWAI_DATA_11_REG (0x006C)



TWAI_TX_BYTE_11 Stored the 11th byte information of the data to be transmitted in operation mode. (WO)

Register 15.17. TWAI_DATA_12_REG (0x0070)



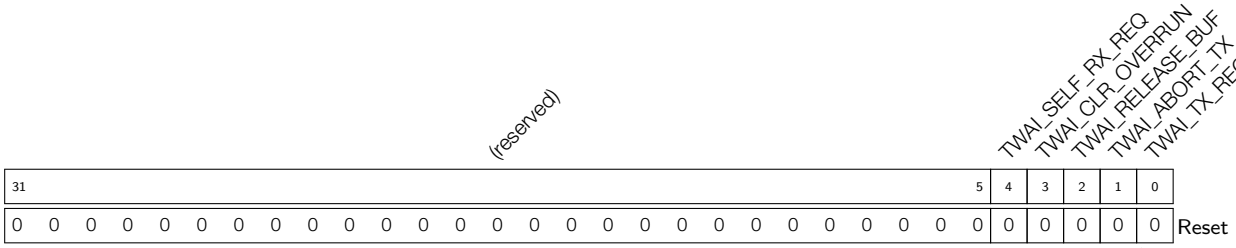
TWAI_TX_BYTE_12 Stored the 12th byte information of the data to be transmitted in operation mode. (WO)

Register 15.18. TWAI_CLOCK_DIVIDER_REG (0x007C)

TWAI_CD These bits are used to configure the divisor of the external CLKOUT pin. (R/W)

TWAI_CLOCK_OFF This bit can be configured in reset mode. 1: Disable the external CLKOUT pin; 0: Enable the external CLKOUT pin (RO | R/W)

Register 15.19. TWAI_CMD_REG (0x0004)



TWAI_TX_REQ Set the bit to 1 to drive nodes to start transmission. (WO)

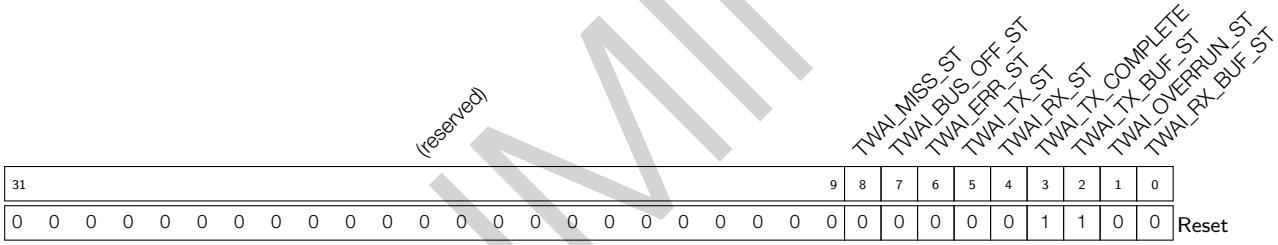
TWAI_ABORT_TX Set the bit to 1 to cancel a pending transmission request. (WO)

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 15.20. 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)

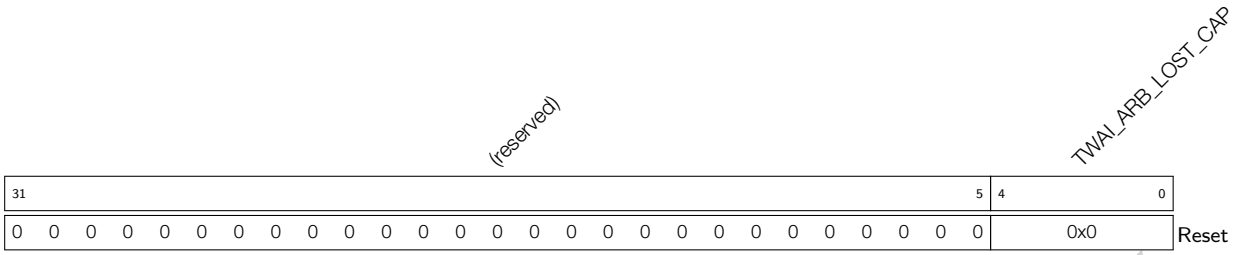
TWAI_TX_ST 1: The TWAI Controller is transmitting a message to the bus. (RO)

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)

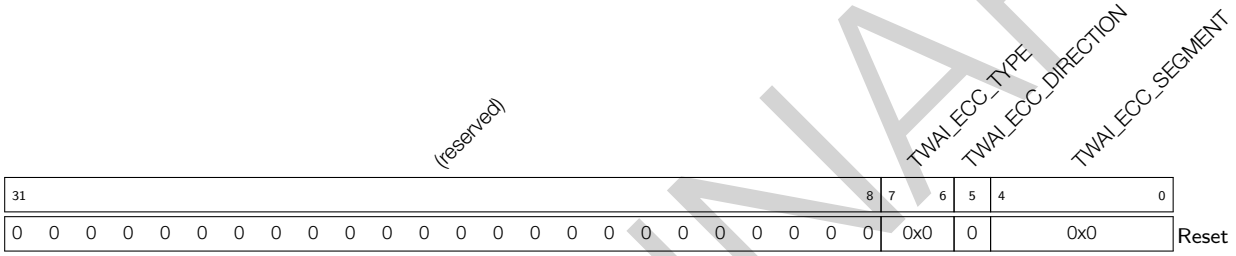
TWAI_MISS_ST This bit reflects whether the data packet in the RX FIFO is complete. 1: The current packet is missing; 0: The current packet is complete (RO)

Register 15.21. TWAI_ARB LOST CAP_REG (0x002C)



TWAI_ARB_LOST_CAP This register contains information about the bit position of lost arbitration. (RO)

Register 15.22. TWAI_ERR_CODE_CAP_REG (0x0030)

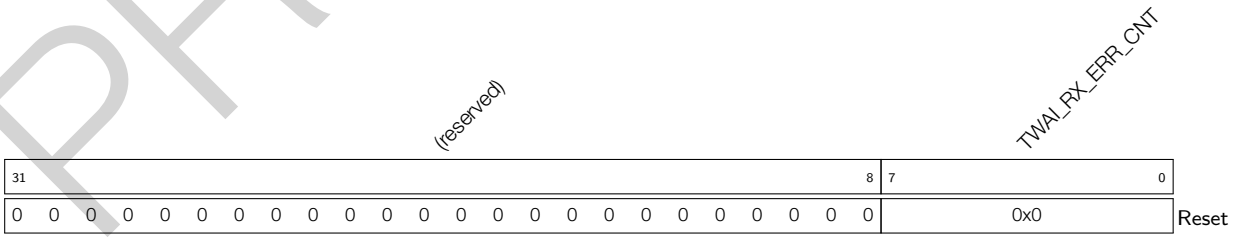


TWAI_ECC_SEGMENT This register contains information about the location of errors, see Table 15-16 for details. (RO)

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)

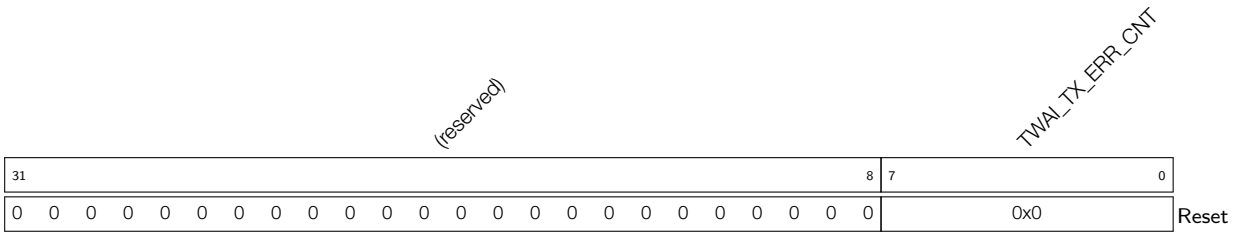
TWAI_ECC_TYPE This register contains information about error types: 00: bit error; 01: form error; 10: stuff error; 11: other type of error (RO)

Register 15.23. TWAI_RX_ERR_CNT_REG (0x0038)



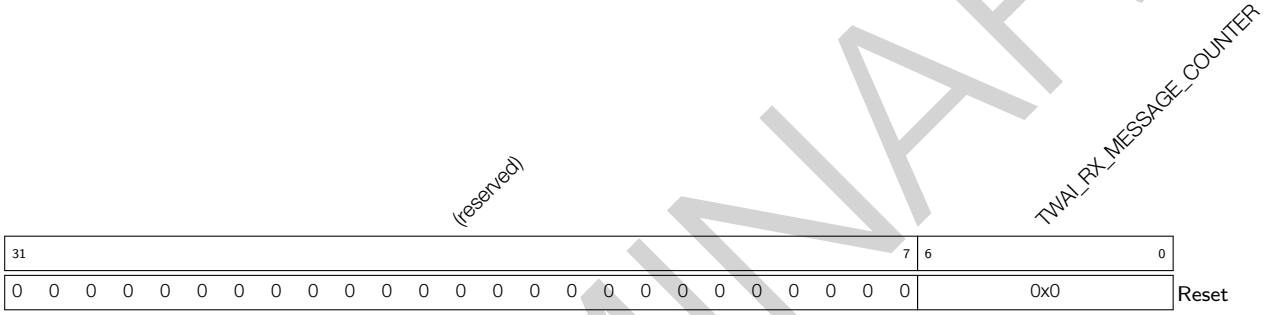
TWAI_RX_ERR_CNT The RX error counter register, reflects value changes in reception status. (RO | R/W)

Register 15.24. TWAI_TX_ERR_CNT_REG (0x003C)



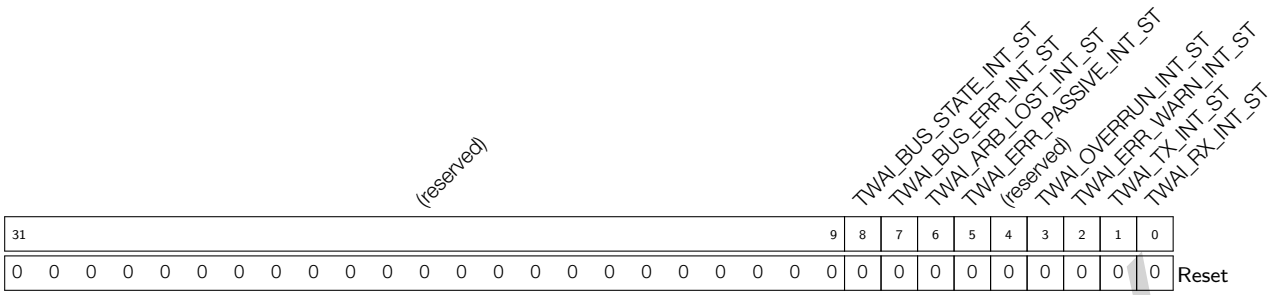
TWAI_TX_ERR_CNT The TX error counter register, reflects value changes in transmission status. (RO | R/W)

Register 15.25. TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER This register reflects the number of messages available within the RX FIFO. (RO)

Register 15.26. 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 transmission is finished and a new transmission is able to start. (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)

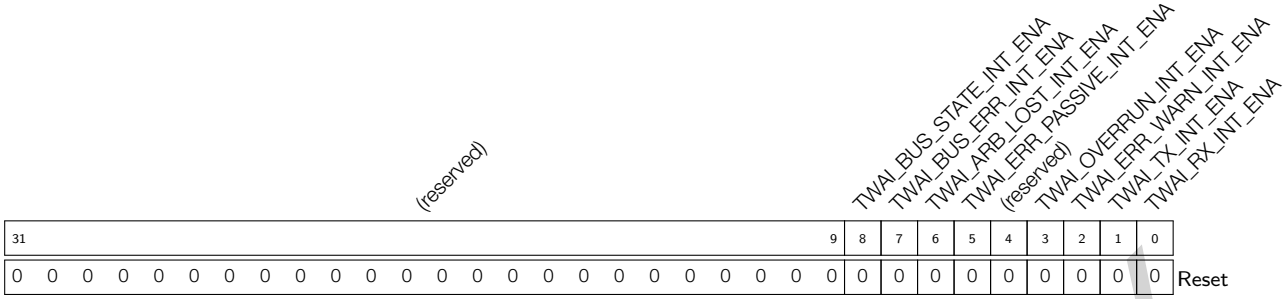
TWAI_ERR_PASSIVE_INT_ST Error passive interrupt. If this bit is set to 1, it indicates the TWAI Controller is switched between error active status and error passive status due to the change of error counters. (RO)

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)

TWAI_BUS_STATE_INT_ST Bus state interrupt. If this bit is set to 1, it indicates the status of TWAI controller has changed. (RO)

Register 15.27. 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 bus error interrupt. (R/W)

TWAI_BUS_STATE_INT_ENA Set this bit to 1 to enable bus state interrupt. (R/W)

16 USB On-The-Go (USB)

16.1 Overview

The ESP32-S3 features a USB On-The-Go peripheral (henceforth referred to as OTG_FS) along with an integrated transceiver. The OTG_FS can operate as either a USB Host or Device and supports 12 Mbit/s full-speed (FS) and 1.5 Mbit/s low-speed (LS) data rates of the USB1.1 specification. The Host Negotiation Protocol (HNP) and the Session Request Protocol (SRP) are also supported.

16.2 Features

16.2.1 General Features

- FS and LS data rates
- HNP and SRP as A-device or B-device
- Dynamic FIFO (DFIFO) sizing
- Multiple modes of memory access
  - Scatter/Gather DMA mode
  - Buffer DMA mode
  - Slave mode
- Can choose integrated transceiver or external transceiver
- Utilizing integrated transceiver with USB Serial/JTAG by time-division multiplexing when only integrated transceiver is used
- Support USB OTG using one of the transceivers while USB Serial/JTAG using the other one when both integrated transceiver or external transceiver are used
- Can be used as a light sleep wake-up source

16.2.2 Device Mode Features

- Endpoint number 0 always present (bi-directional, consisting of EP0 IN and EP0 OUT)
- Six additional endpoints (endpoint numbers 1 to 6), configurable as IN or OUT
- Maximum of five IN endpoints concurrently active at any time (including EP0 IN)
- All OUT endpoints share a single RX FIFO
- Each IN endpoint has a dedicated TX FIFO

16.2.3 Host Mode Features

- Eight channels (pipes)
  - A control pipe consists of two channels (IN and OUT), as IN and OUT transactions must be handled separately. Only Control transfer type is supported.

- Each of the other seven channels is dynamically configurable to be IN or OUT, and supports Bulk, Isochronous, and Interrupt transfer types.

- All channels share an RX FIFO, non-periodic TX FIFO, and periodic TX FIFO. The size of each FIFO is configurable.

16.3 Functional Description

16.3.1 Controller Core and Interfaces

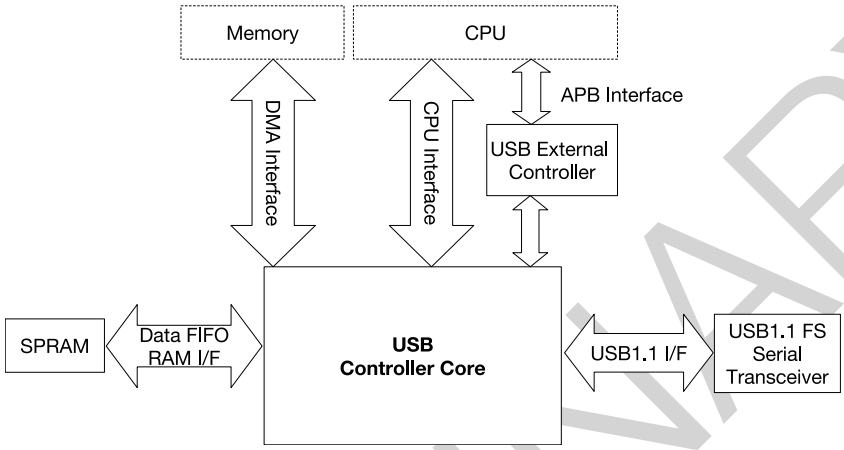


Figure 16-1. OTG_FS System Architecture

The core part of the OTG_FS peripheral is the USB Controller Core. The controller core has the following interfaces (see Figure 16-1):

- **CPU Interface**
  Provides the CPU with read/write access to the controller core's various registers and FIFOs. This interface is internally implemented as an AHB Slave Interface. The way to access the FIFOs through the CPU interface is called Slave mode.

- **APB Interface**
  Allows the CPU to control the USB controller core via the USB external controller.

- **DMA Interface**
  Provides the controller core's internal DMA with read/write access to system memory (e.g., fetching and writing data payloads when operating in DMA mode). This interface is internally implemented as an AHB Master interface.

- **USB1.1 Interface**
  This interface is used to connect the controller core to a USB1.1 FS serial transceiver. Aside from USB OTG, ESP32-S3 also includes a USB Serial/JTAG controller (see Chapter 20 *USB Serial/JTAG Controller (USB_SERIAL_JTAG) [to be added later]*). These two USB controllers can utilize the integrated internal transceiver by time-division multiplexing or one USB controller connects to internal transceiver and the other one connects to an external transceiver.

  When only internal transceiver is used, it is shared by USB OTG and USB Serial/JTAG. In default, internal transceiver is connected to USB Serial/JTAG. When RTC_CNTL_SW _HW_USB_PHY_SEL_CFG is 0, the connection of internal transceiver is controlled by efuse bit EFUSE_USB_PHY_SEL. When EFUSE_USB_PHY_SEL is 0, internal transceiver is connected with USB Serial/JTAG. Otherwise, it is

connected to USB OTG. When RTC_CNTL_SW _HW_USB_PHY_SEL_CFG is 1, the connection switching is controlled by RTC_CNTL_SW _USB_PHY_SEL_CFG(it has the same meaning with EFUSE_USB_PHY_SEL).

When both internal transceiver and external transceiver are used, one USB controller select one of transceivers, the other would select the other transceiver. The specific connection mapping please refer to Chapter 20 *USB Serial/JTAG Controller (USB_SERIAL_JTAG) [to be added later]*.

- **USB External Controller**
  The USB External Controller is primarily used to control the routing of the USB1.1 FS serial interface to either the internal or external transceiver. The External Controller can also enable a power saving mode by gating the controller core's clock (AHB clock) or powering down the connected SPRAM. Note that this power saving mode is different for the power savings via SRP.
- **Data FIFO RAM Interface**
  The multiple FIFOs used by the controller core are not actually located within the controller core itself, but on the SPRAM (Single-Port RAM). FIFOs are dynamically sized, thus are allocated at run-time in the SPRAM. When the CPU, DMA, or the controller core attempts to read/write to FIFOs, those accesses are routed through the data FIFO RAM interface.

16.3.2 Memory Layout

The following diagram illustrates the memory layout of the OTG_FS registers which are used to configure and control the USB Controller Core. Note that USB External Controller uses a separate set of registers (called wrap registers).

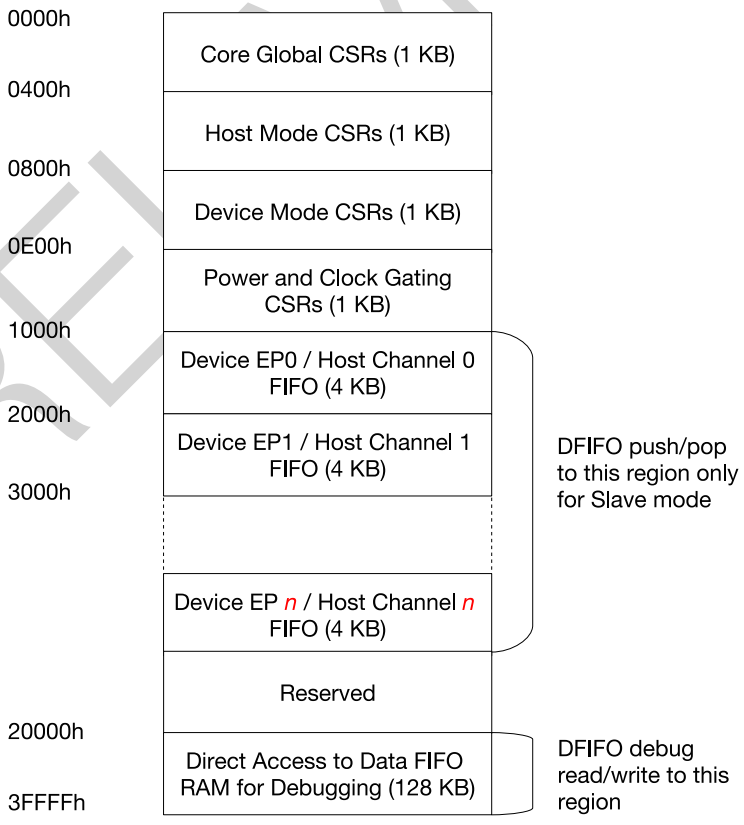


Figure 16-2. OTG_FS Register Layout

16.3.2.1 Control & Status Registers

- **Global CSRs**
  These registers are responsible for the configuration/control/status of the global features of OTG_FS (i.e., features which are common to both Host and Device modes). These features include OTG control (HNP, SRP, and A/B-device detection), USB configuration (selecting Host or Device mode and PHY selection), and system-level interrupts. Software can access these registers whilst in Host or Device modes.
- **Host Mode CSRs**
  These registers are responsible for the configuration/control/status when operating in Host mode, thus should only be accessed when operating in Host mode. Each channel will have its own set of registers within the Host mode CSRs.
- **Device Mode CSRs**
  These registers are responsible for the configuration/control/status when operating in Device mode, thus should only be accessed when operating in Device mode. Each Endpoint will have its own set of registers within the Device mode CSRs.
- **Power and Clock Gating**
  A single register used to control power-down and gate various clocks.

16.3.2.2 FIFO Access

The OTG_FS makes use of multiple FIFOs to buffer transmitted or received data payloads. The number and type of FIFOs are dependent on Host or Device mode, and the number of channels or endpoints used (see Section 16.3.3). There are two ways to access the FIFOs: DMA mode and Slave mode. When using Slave mode, the CPU will need to access to these FIFOs by reading and writing to either the DFIFO push/pop regions or the DFIFO read/write debug region. FIFO access is governed by the following rules:

- Read access to any address in any one of the 4 KB push/pop regions will result in a pop from the shared RX FIFO.
- Write access to a particular 4 KB push/pop region will result in a push to the corresponding endpoint or channel's TX FIFO given that the endpoint is an IN endpoint, or the channel is an OUT channel.
  - In Device mode, data is pushed to the corresponding IN endpoint's dedicated TX FIFO.
  - In Host mode, data is pushed to the non-periodic TX FIFO or the periodic TX FIFO depending on whether the channel is a non-periodic channel, or a periodic channel.
- Access to the 128 KB read/write region will result in direct read/write instead of a push/pop. This is generally used for debugging purposes only.

Note that pushing and popping data to and from the FIFOs by the CPU is only required when operating in Slave mode. When operating in DMA mode, the internal DMA will handle all pushing/popping of data to and from the TX and RX FIFOs.

16.3.3 FIFO and Queue Organization

The FIFOs in OTG_FS are primarily used to hold data packet payloads (the data field of USB Data packets). TX FIFOs are used to store data payloads that will be transmitted by OUT transactions in Host mode or IN transactions in Device mode. RX FIFOs are used to store received data payloads of IN transactions in Host mode

or OUT transactions in Device mode. In addition to storing data payloads, RX FIFOs also store a **status entry** for each data payload. Each status entry contains information about a data payload such as channel number, byte count, and validity status. When operating in slave mode, status entries are also used to indicate various channel events.

The portion of SPRAM that can be used for FIFO allocation has a depth of 256 and a width of 35 bits (32 data bits plus 3 control bits). The multiple FIFOs used by each channel (in Host mode) or endpoint (in Device mode) are allocated into the SPRAM and can be dynamically sized.

16.3.3.1 Host Mode FIFOs and Queues

The following FIFOs are used when operating in Host mode (see Figure 16-3):

- **Non-periodic TX FIFO**: Stores data payloads of bulk and control OUT transactions for all channels.
- **Periodic TX FIFO**: Stores data payloads of interrupt or isochronous OUT transactions for all channels.
- **RX FIFO**: Stores data payloads of all IN transactions, and status entries that are used to indicate size of data payloads and transaction/channel events such as transfer complete or channel halted.

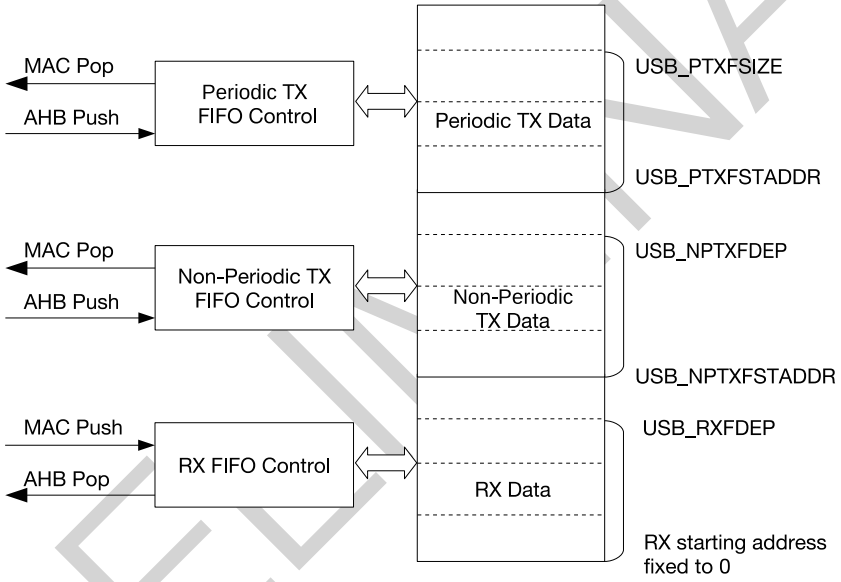


Figure 16-3. Host Mode FIFOs

In addition to FIFOs, Host mode also contains two request queues used to queue up the various transaction request from the multiple channels. Each entry in a request queue holds the IN/OUT channel number along with other information to perform the transaction (such as transaction type). Request queues are also used to queue other types of requests such as a channel halt request.

Unlike FIFOs, request queues are fixed in size and cannot be accessed directly by software. Rather, once a channel is enabled, requests will be automatically written to the request queue by the Host core. The order in which the requests are written into the queue determines the sequence of transactions on the USB.

Host mode contains the following request queues:

- **Non-periodic request queue**: Request queue for all non-periodic channels (bulk and control). The queue has a depth of four entries.

- **Periodic request queue**: Request queue for all periodic channels (interrupt and isochronous). The queue has a depth of eight entries.

When scheduling transactions, hardware will execute all requests on the periodic request queue first before executing requests on the non-periodic request queue.

16.3.3.2 Device Mode FIFOs

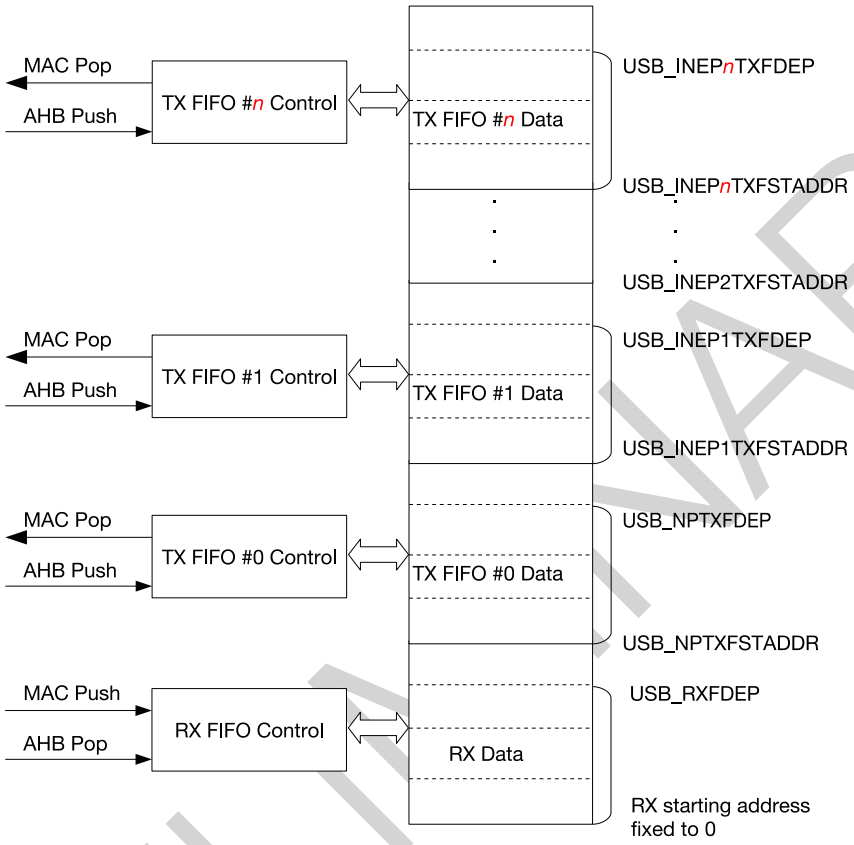


Figure 16-4. Device Mode FIFOs

The following FIFOs are used when operating in Device mode (See Figure 16-4):

- **RX FIFO**: Stores data payloads received in Data packet, and status entries (used to indicate size of those data payloads).
- **Dedicated TX FIFO**: Each active IN endpoint will have a dedicated TX FIFO used to store all IN data payloads of that endpoint, regardless of the transaction type (both periodic and non-periodic IN transactions).

Due to the dedicated FIFOs, Device mode does not use any request queues. Instead, the order of IN transactions are determined by the Host.

16.3.4 Interrupt Hierarchy

OTG_FS provides a single interrupt line which can be routed via the interrupt matrix to one of the CPUs. The interrupt signal can be unmasked by setting USB_GLBLINTRMSK. The OTG_FS interrupt is an OR of all bits in the USB_GINTSTS_REG register, and the bits in USB_GINTSTS_REG can be unmasked by setting the corresponding bits in the USB_GINTMSK_REG register. USB_GINTSTS_REG contains system level interrupts,

and also specific bits for Host or Device mode interrupts, and OTG specific interrupts. OTG_FS interrupt sources are organized as Figure 16-5 shows.

The following bits of the USB_GINTSTS_REG register indicate an interrupt source lower in the hierarchy:

- **USB_PRTINT** indicates that the Host port has a pending interrupt. The USB_HPRT_REG register indicates the interrupt source.
- **USB_HCHINT** indicates that one or more Host channels have a pending interrupt. Read the USB_HAINT_REG register to determine which channel(s) have a pending interrupt, then read the pending channel's USB_HCINT*n*_REG register to determine the interrupt source.
- **USB_OEPINT** indicates that one or more OUT endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which OUT endpoint(s) have a pending interrupt, then read the USB_DOEPINT*n*_REG register to determine the interrupt source.
- **USB_IEPINT** indicates that one or more IN endpoints have a pending interrupt. Read the USB_DAINT_REG register to determine which IN endpoint(s) are pending, then read the pending IN endpoint's USB_DIEPINT*n*_REG register to determine the interrupt source.
- **USB_OTGINT** indicates an On-The-Go event has triggered an interrupt. Read the USB_GOTGINT_REG register to determine which OTG event(s) triggered the interrupt.

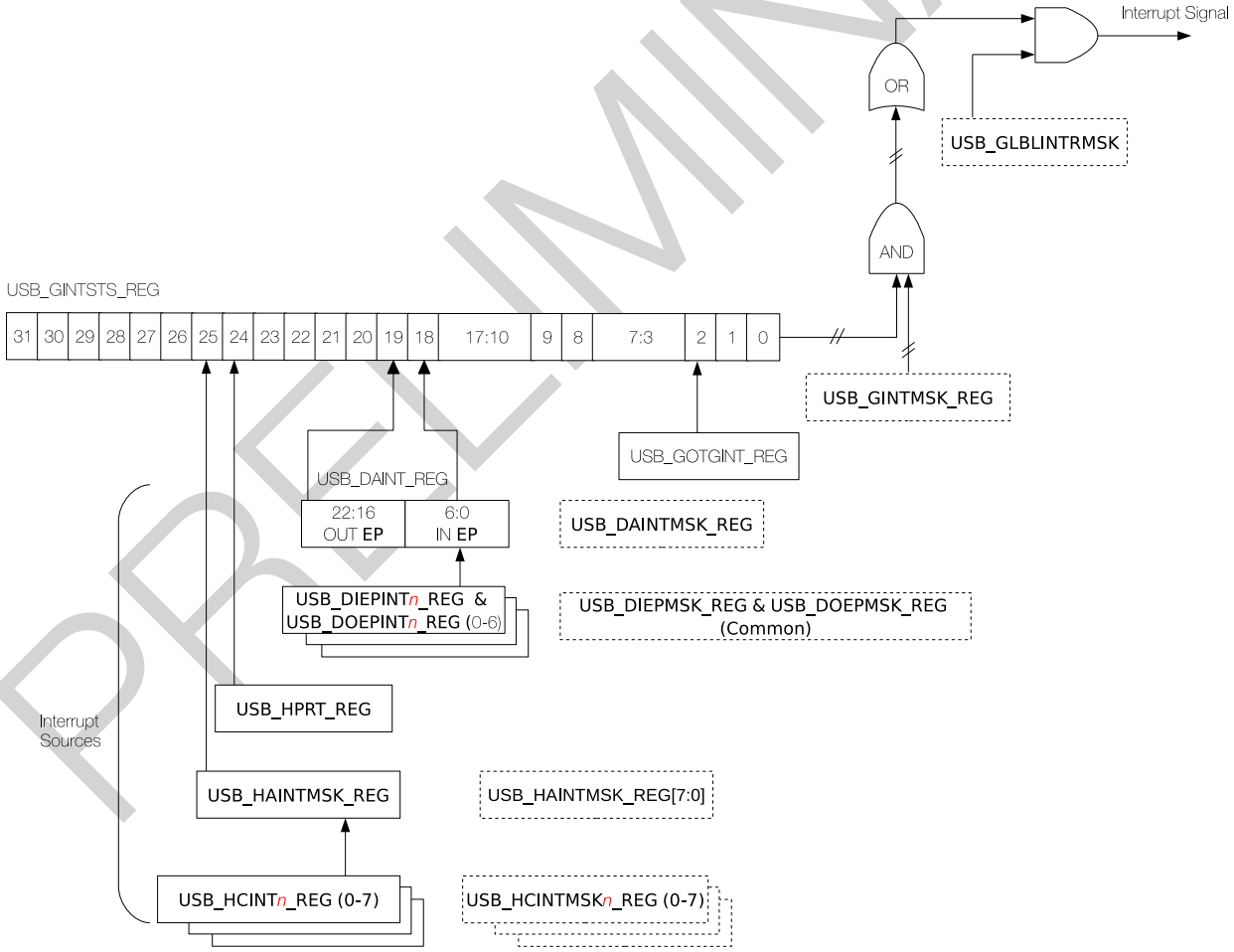


Figure 16-5. OTG_FS Interrupt Hierarchy

16.3.5 DMA Modes and Slave Mode

USB On-The-Go supports three ways to access memory: Scatter/Gather DMA mode, Buffer DMA mode, and Slave mode.

16.3.5.1 Slave Mode

When operating in Slave mode, all data payloads must be pushed/popped to and from the FIFOs by the CPU.

- When transmitting a packet using IN endpoints or OUT channels, the data payload must be pushed into the corresponding endpoint or channel's TX FIFO.
- When receiving a packet, the packet's status entry must first be popped off the RX FIFO by reading USB_GRXSTSP_REG. The status entry should be used to determine the length of the packet's payload (in bytes). The corresponding number of bytes must then be manually popped off the RX FIFO by reading from the RX FIFO's memory region.

16.3.5.2 Buffer DMA Mode

Buffer mode is similar to Slave mode but utilizes the internal DMA to push and pop data payloads to the FIFOs.

- When transmitting a packet using IN endpoints or OUT channels, the data payload's address in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DOEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will push the data payload from memory into the TX FIFO of the channel or endpoint.
- When receiving a packet using OUT endpoints or IN channels, the address of an empty buffer in memory should be written to the USB_HCDMA*n*_REG (in Host mode) or USB_DOEPDMA*n*_REG (in Device mode) registers. When the endpoint or channel is enabled, the internal DMA will pop the data payload from RX FIFO into the buffer.

16.3.5.3 Scatter/Gather DMA Mode

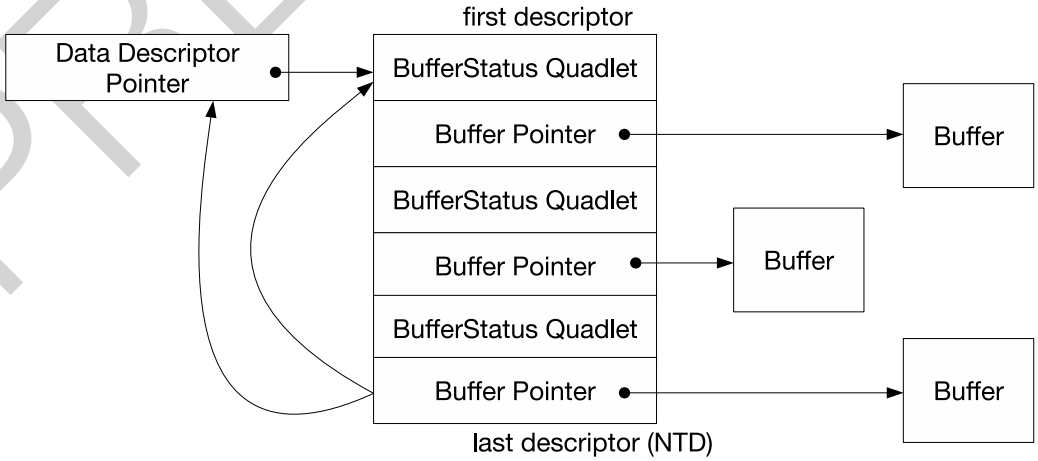


Figure 16-6. Scatter/Gather DMA Descriptor List

When operating in Scatter/Gather DMA mode, buffers containing data payloads can be scattered throughout memory. Each endpoint or channel will have a contiguous DMA descriptor list, where each descriptor contains a 32-bit pointer to the data payload or buffer and a 32-bit buffer descriptor (BufferStatus Quadlet). The data payloads and buffers can correspond to a single transaction (i.e., < 1 MPS bytes) or an entire transfer (> 1 MPS bytes). (MPS: maximum packet size) The list is implemented as a ring buffer meaning that the DMA will return to the first entry when it encounters the last entry on the list.

- When transmitting a transfer/transaction using IN endpoints or OUT channels, the DMA will gather the data payloads from the multiple buffers and push them into a TX FIFO.
- When receiving a transfer/transaction using OUT endpoints or IN channels, the DMA will pop the received data payloads from the RX FIFO and scatter them to the multiple buffers pointed to by the DMA list entries.

16.3.6 Transaction and Transfer Level Operation

When operating in either Host or Device mode, communication can operate either at the transaction level or the transfer level.

16.3.6.1 Transaction and Transfer Level in DMA Mode

When operating at the transfer level in DMA Host mode, software is interrupted only when a channel has been halted. Channels are halted when their programmed transfer size has completed successfully, has received a STALL, or if there are excessive transaction errors (i.e., 3 consecutive transaction errors). When operating in DMA Device mode, all errors are handled by the controller core itself.

When operating at the transaction level in DMA mode, the transfer size is set to the size of one data packet (either a maximum packet size or a short packet size).

16.3.6.2 Transaction and Transfer Level in Slave Mode

When operating at the transaction level in Slave Mode, transfers are handled one transaction at a time. Each data payload should correspond to a single data packet, and software must determine whether a retry of the transaction is necessary based on the handshake response received on the USB (e.g., ACK or NAK).

The following table describes transaction level operation in Slave mode for both IN and OUT transactions.

Table 16-1. IN and OUT Transactions in Slave Mode

Host Mode	Device Mode
OUT Transactions	
1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_HCTSIZ*n*_REG register, enables the channel, then copies the packet's data payload into the TX FIFO. 2. When the last DWORD of the data payload has been pushed, the controller core will automatically write a request into the appropriate request queue. 3. If the transaction was successful, the USB_XFERCOMPL interrupt will be generated. If the transaction was unsuccessful, an error interrupt (e.g. USB_H_NACK*n*) will occur.	1. Software specifies the expected size of the data packet (1 MPS) and the number of data packets (1 data packet) in the USB_DIEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to transmit a packet to it. 2. The received packet will be pushed into the RX FIFO along with a packet status entry. 3. If the transaction was unsuccessful (e.g., due to a full RX FIFO), the endpoint will automatically NAK the incoming packet.
IN Transactions	
1. Software specifies the expected size of the data packet and the number of packets (1 data packet) in the USB_HCTSIZ*n*_REG register, then enables the channel. 2. The controller core will automatically write a request into the appropriate request queue. 3. If the transaction was successful, the received data along with a status entry should be written to the RX FIFO. If the transaction was unsuccessful, an error interrupt (e.g., USB_H_NACK*n*) will occur.	1. Software specifies the size of the data packet and the number of data packets (1 data packet) in the USB_DIEPTSIZ*n*_REG register. Once the endpoint is enabled, it will wait for the host to read the packet. 2. When the packet has been transmitted, the USB_XFERCOMPL interrupt will be generated.

When operating at the transfer level in Slave mode, one or more transaction-level operations can be pipelined thus being analogous to transfer level operation in DMA mode. Within pipelined transactions, multiple packets of the same transfer can be read/written from the FIFOs in single instance, thus preventing the need for interrupting the software on a per-packet basis.

Operating on a transfer level in Slave mode is similar to operating on the transaction-level, except the transfer size and packet count for each transfer in the USB_HCTSIZ*n*_REG or USB_DIEPTSIZ*n*_REG register will need to be set to reflect the entire transfer. After the channel or endpoint is enabled, multiple data packets worth of payloads should be written to or read from the TX or RX FIFOs respectively (given that there is enough space or enough data).

16.4 OTG

USB OTG allows OTG devices to act in the USB Host role or the USB Device role. Thus, OTG devices will typically have a Mini-AB or Micro-AB receptacle so that it can receive an A-plug or B-plug. OTG devices will become either an A-device or a B-device depending on whether an A-plug or a B-plug is connected.

- A-device defaults to the Host role (A-Host) whilst B-device defaults to the Device role (B-Peripheral).
- A-device and B-device may exchange roles by using the Host Negotiation Protocol (HNP), thus becoming A-peripheral and B-Host.
- A-device can turn off Vbus to save power. B-device can then wake up the A-device by requesting it to turn on Vbus and start a new session. This mechanism is called session request protocol (SRP).
- A-device always powers Vbus even if it is an A-peripheral.

OTG devices are able to determine whether they are connected to an A plug or a B plug using the ID pin of the plugs. The ID pin in A-plugs are pulled to ground whilst B-plugs have the ID pin left floating.

16.4.1 OTG Interface

The OTG_FS supports both the Session Request Protocol (SRP) and Host Negotiation Protocol (HNP) of the OTG Revision 1.3 specification. The OTG_FS controller core interfaces with the transceiver (internal or external) using the UTMI+ OTG interface. The UTMI+ OTG interface allows the controller core to manipulate the transceiver for OTG purposes (e.g., enabling/disabling pull-ups and pull-downs in HNP), and also allows the transceiver to indicate OTG related events. If an external transceiver is used instead, the UTMI+ OTG interface signals will be routed to the ESP32-S3's GPIOs instead through GPIO Matrix, please refer to Chapter 2 *IO MUX and GPIO Matrix (GPIO, IO MUX)*. The UTMI+ OTG interface signals are described in Table 16-2.

Table 16-2. UTMI OTG Interface

Signal Name	I/O	Description
usb_otg_iddig_in	I	Mini A/B Plug Indicator. Indicates whether the connected plug is mini-A or mini-B. Valid only when usb_otg_idpullup is sampled asserted. 1'b0: Mini-A connected 1'b1: Mini-B connected
usb_otg_avalid_in	I	A-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for an A-peripheral session. The comparator thresholds are: 1'b0: Vbus <0.8 V 1'b1: Vbus = 0.2 V to 2.0 V
usb_otg_bvalid_in	I	B-Peripheral Session Valid. Indicates if the voltage Vbus is at a valid level for a B-peripheral session. The comparator thresholds are: 1'b0: Vbus <0.8 V 1'b1: Vbus = 0.8 V to 4 V
usb_otg_vbusvalid_in	I	Vbus Valid. Indicates if the voltage Vbus is valid for A/B-device/peripheral operation. The comparator thresholds are: 1'b0: Vbus <4.4 V 1'b1: Vbus >4.75 V

Signal Name	I/O	Description
usb_srp_sessend_in	I	B-device Session End. Indicates if the voltage Vbus is below the B-device Session End threshold. The comparator thresholds are: 1'b0: Vbus >0.8 V 1'b1: Vbus <0.2 V
usb_otg_idpullup	O	Analog ID input Sample Enable. Enables sampling the analog ID line. 1'b0: ID pin sampling disabled 1'b1: ID pin sampling enabled
usb_otg_dppulldown	O	D+ Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D+ line.
usb_otg_dmpulldown	O	D- Pull-down Resistor Enable. Enables the 15 kΩ pull-down resistor on the D- line.
usb_otg_drvvbus	O	Drive Vbus. Enables driving Vbus to 5 V. 1'b0: Do not drive Vbus 1'b1: Drive Vbus
usb_srp_chrgvbus	O	Vbus Input Charge Enable. Directs the PHY to charge Vbus. 1'b0: Do not charge Vbus through a resistor 1'b1: Charge Vbus through a resistor (must be active for at least 30 ms)
usb_srp_dischrgvbus	O	Vbus Input Discharge Enable. Directs the PHY to discharge Vbus. 1'b0: Do not discharge Vbus through a resistor. 1'b1: Discharge Vbus through a resistor (must be active for at least 50 ms).

16.4.2 ID Pin Detection

Bit USB_CONIDSTS in register USB_GOTGCTL_REG indicates whether the OTG controller is an A-device (1'b0) or a B-device (1'b1). The USB_CONIDSTSCHNG interrupt will trigger whenever there is a change to USB_CONIDSTS (i.e., when a plug is connected or disconnected).

16.4.3 Session Request Protocol (SRP)

16.4.3.1 A-Device SRP

Figure 16-7 illustrates the flow of SRP when the OTG_FS is acting as an A-device (i.e., default host and the device that powers Vbus).

1. To save power, the application suspends and turns off port power when the bus is idle by writing to the Port Suspend (USB_PRTSUSP to 1'b0) and Port Power (USB_PRTPWR to 1'b0) bits in the Host Port Control and Status register.
2. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal.
3. The A-device must detect SE0 for at least 2 ms to start SRP when Vbus power is off.
4. To initiate SRP, the B-device turns on its data line pull-up resistor for 5 to 10 ms. The OTG_FS core detects data-line pulsing.
5. The device drives Vbus above the A-device session valid (2.0 V minimum) for Vbus pulsing. The OTF_FS core interrupts the application on detecting SRP. The Session Request Detected bit (USB_SESSREQINT) is set in Global Interrupt Status register.

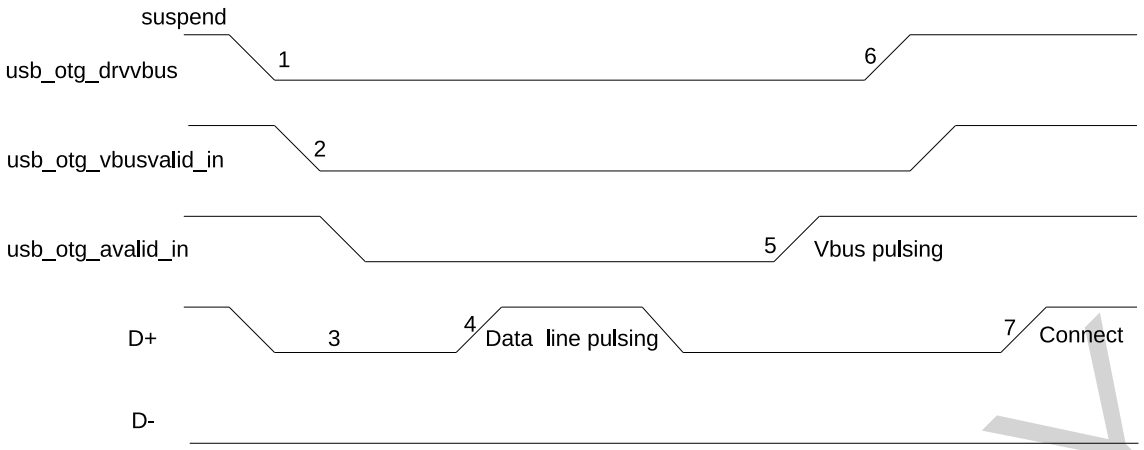


Figure 16-7. A-Device SRP

6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by asserting usb_otg_vbusvalid_in signal.
7. When the USB is powered, the B-device connects, completing the SRP process.

16.4.3.2 B-Device SRP

Figure 16-8 illustrates the flow of SRP when the OTG_FS is acting as a B-device (i.e., does not power Vbus).

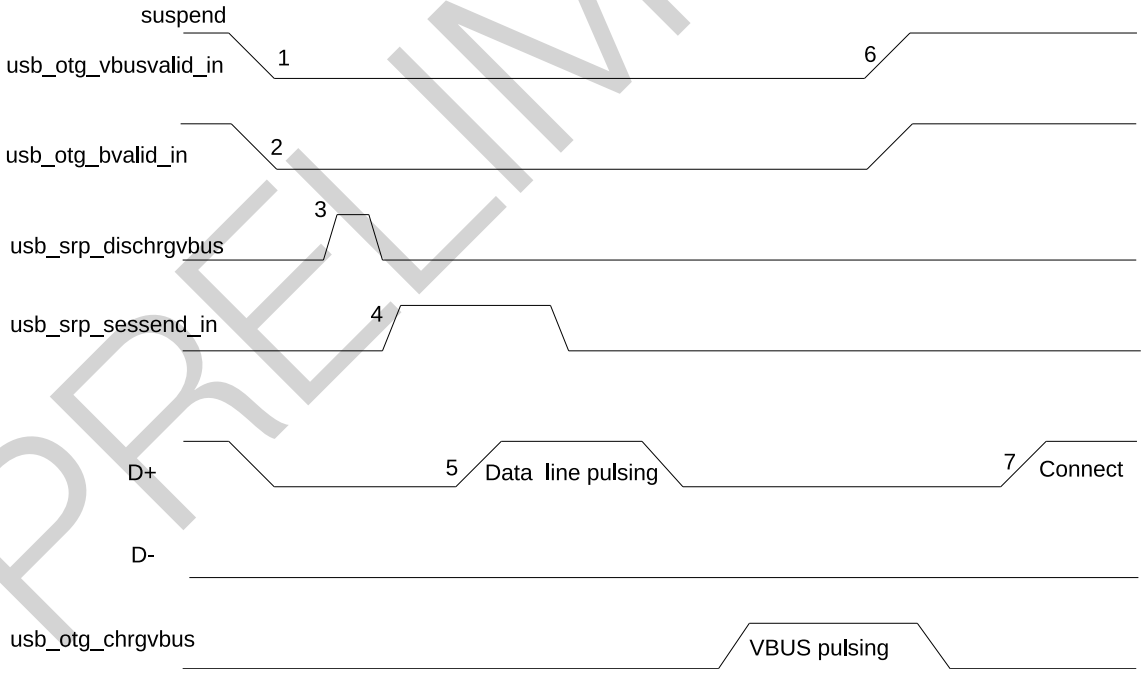


Figure 16-8. B-Device SRP

1. To save power, the host (A-device) suspends and turns off port power when the bus is idle. PHY indicates port power off by deasserting the usb_otg_vbusvalid_in signal. The OTG_FS core sets the Early Suspend bit in the Core Interrupt register (USB_ERLYSUSP interrupt) after detecting 3 ms of bus idleness. Following this, the OTF_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register. The PHY

indicates the end of the B-device session by deasserting the usb_otg_bvalid_in signal.

2. The OTG_FS core asserts the usb_otg_dischrgvbus signal to indicate to the PHY to speed up Vbus discharge.
3. The PHY indicates the session's end by asserting the usb_otg_sessend_in signal. This is the initial condition for SRP. The OTG_FS core requires 2 ms of SE0 before initiating SRP. For a USB 1.1 full-speed serial transceiver, the application must wait until Vbus discharges to 0.2 V after USB_BSESVLD is deasserted.
4. The application waits for 1.5 seconds (TB_SE0_SRP time) before initiating SRP by writing the Session Request bit (USB_SESREQ) in the OTG Control and Status register. The OTG_FS core performs data-line pulsing followed by Vbus pulsing.
5. The host (A-device) detects SRP from either the data-line or Vbus pulsing, and turns on Vbus. The PHY indicates Vbus power-on by asserting usb_otg_vbusvalid_in.
6. The OTG_FS core performs Vbus pulsing by asserting usb_srp_chrgvbus. The host (A-device) starts a new session by turning on Vbus, indicating SRP success. The OTG_FS core interrupts the application by setting the Session Request Success Status Change bit (USB_SESREQSC) in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
7. When the USB is powered, the OTG_FS core connects, completing the SRP process.

16.4.4 Host Negotiation Protocol (HNP)

16.4.4.1 A-Device HNP

Figure 16-9 illustrates the flow of HNP when the OTG_FS is acting as an A-device.

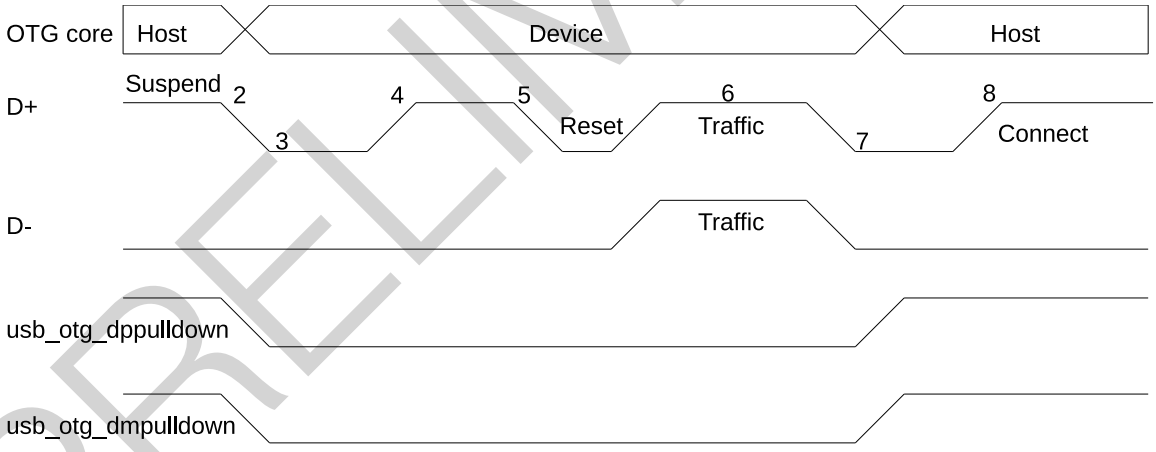


Figure 16-9. A-Device HNP

1. The OTG_FS core sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set Host Set HNP Enable bit (USB_HSTSETHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core that the B-device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be

suspended. The OTG_FS core sets the Host Negotiation Detected interrupt (USB_HSTNEGDET) in the OTG Interrupt Status register, indicating the start of HNP. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate a device role. The PHY enables the D+ pull-up resistor, thus indicates a connection for the B-device. The application must read the Current Mode bit (USB_CURMOD_INT) in the OTG Control and Status register to determine Device mode operation.

4. The B-device detects the connection, issues a USB reset, and enumerates the OTG_FS core for data traffic.
5. The B-device continues the host role, initiating traffic, and suspends the bus when done. The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after detecting 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register.
6. In Negotiated mode, the OTG_FS core detects the suspend, disconnects, and switches back to the host role. The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.
7. The OTG_FS core sets the Connector ID Status Change interrupt (USB_CONIDSTS) in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the OTG_FS core's operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current Mode bit in the OTG Control and Status register to determine Host mode operation.
8. The B-device connects, completing the HNP process.

16.4.4.2 B-Device HNP

Figure 16-10 illustrates the flow of HNP when the OTG_FS is acting as an B-device.

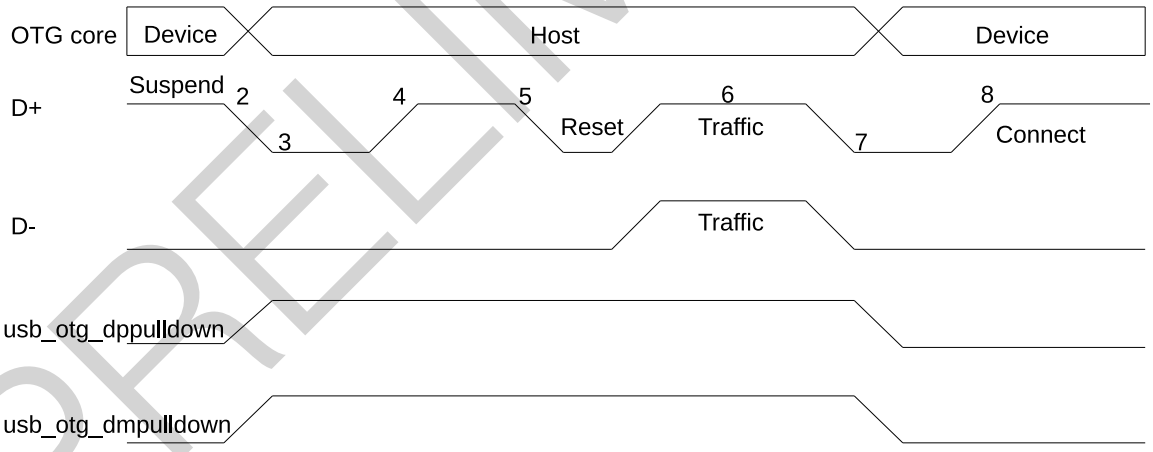


Figure 16-10. B-Device HNP

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The OTG_FS core's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate HNP support. The application sets the HNP Request bit (USB_DEVHNPEN) in the OTG Control and Status register to indicate to the OTG_FS core to initiate HNP.
2. When A-device has finished using the bus, it suspends the bus.

(a) The OTG_FS core sets the Early Suspend bit (USB_ERLYSUSP) in the Core Interrupt register after 3 ms of bus idleness. Following this, the OTG_FS core sets the USB Suspend bit (USB_USBSUSP) in the Core Interrupt register. The OTG_FS core disconnects and the A-device detects SE0 on the bus, indicating HNP.

(b) The OTG_FS core asserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate its assumption of the host role.

(c) The A-device responds by activating its D+ pull-up resistor within 3 ms of detecting SE0. The OTG_FS core detects this as a connect.

(d) The OTG_FS core sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register (USB_CONIDSTS), indicating the HNP status. The application must read the Host Negotiation Success bit (USB_HSTNEGSCS) in the OTG Control and Status register to determine host negotiation success. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine Host mode operation.

3. Program the USB_PRTPWR bit to 1'b1. This drives Vbus on the USB.
4. Wait for the USB_PRTCONNDET interrupt. This indicates that a device is connected to the port.
5. The application sets the reset bit (USB_PRTRST) and the OTG_FS core issues a USB reset and enumerates the A-device for data traffic.
6. Wait for the USB_PRTENCHNG interrupt.
7. The OTG_FS core continues the host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit (USB_PRTSUSP) in the Host Port Control and Status register.
8. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The OTG_FS core deasserts the usb_otg_dppulldown and usb_otg_dmpulldown signals to indicate the assumption of the device role.
9. The application must read the Current Mode bit (USB_CURMOD_INT) in the Core Interrupt register to determine the Host mode operation.
10. The OTG_FS core connects, completing the HNP process.

17 SD/MMC Host Controller (SDHOST)

17.1 Overview

The ESP32-S3 memory card interface controller provides a hardware interface between the Advanced Peripheral Bus (APB) and an external memory device. The memory card interface allows the ESP32-S3 to be connected to SDIO memory cards, MMC cards and devices with a CE-ATA interface. It supports two external cards (Card0 and Card1). And all SD/MMC module interface signal only connect to GPIO pad by GPIO matrix.

17.2 Features

This module supports the following features:

- Two external cards
- SD Memory Card standard: V3.0 and V3.01
- MMC: V4.41, V4.5, and V4.51
- CE-ATA: V1.1
- 1-bit, 4-bit, and 8-bit modes

The SD/MMC controller topology is shown in Figure 17-1. The controller supports two peripherals which cannot be functional at the same time.

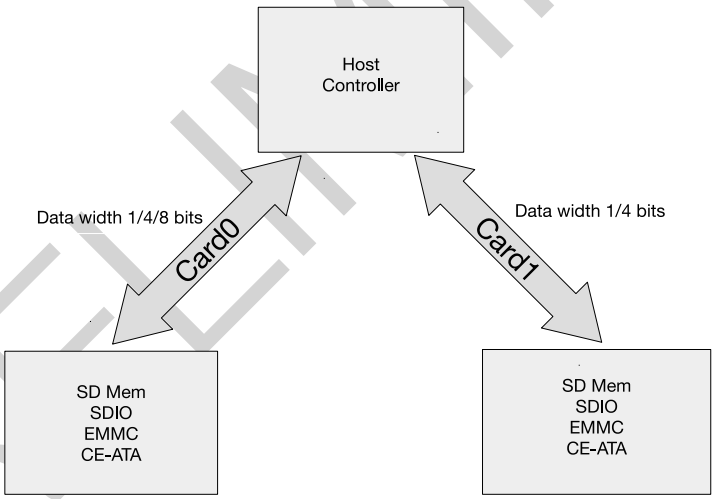


Figure 17-1. SD/MMC Controller Topology

17.3 SD/MMC External Interface Signals

The primary external interface signals, which enable the SD/MMC controller to communicate with an external device, are clock (sdhost_cclk_out_1.eg:card1), command (sdhost_ccmd_out_1) and data signals (sdhost_cdata_in_1[7:0]/sdhost_cdata_out_1[7:0]). Additional signals include the card interrupt, card detect, and write-protect signals. The direction of each signal is shown in Figure 17-2. The direction and description of each pin are listed in Table 17-1.

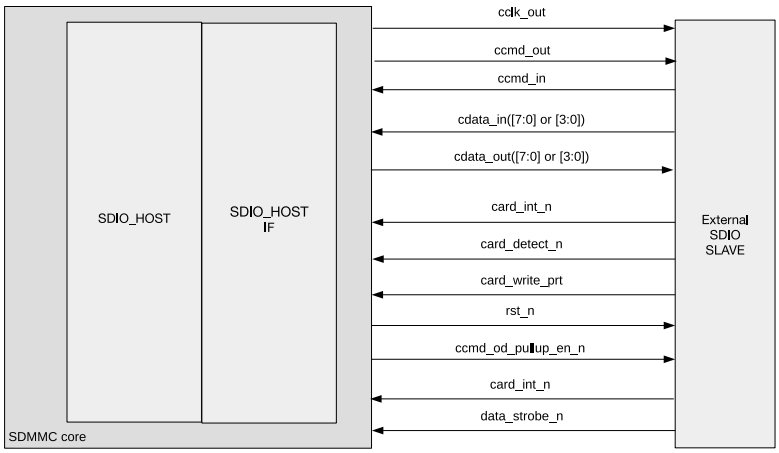


Figure 17-2. SD/MMC Controller External Interface Signals

Table 17-1. SD/MMC Signal Description

Pin	Direction	Description
sdhost_cclk_out	Output	Clock signals for slave device
sdhost_ccmd	Duplex	Duplex command/response lines
sdhost_cdata	Duplex	Duplex data read/write lines
sdhost_card_detect_n	Input	Card detection input line
sdhost_card_write_prt	Input	Card write protection status input
sdhost_rst_n	Output	Hardware reset for MMC4.4 cards
sdhost_ccmd_od_pullup_en_n	output	Card Cmd Open-Drain Pullup
sdhost_card_int_n	Input	Interrupt pin for eSDIO devices
sdhost_data_strobe_n	Input	Card HS400 Data Strobe

17.4 Functional Description

17.4.1 SD/MMC Host Controller Architecture

The SD/MMC host controller consists of two main functional blocks, as shown in Figure 17-3:

- Bus Interface Unit (BIU): It provides APB interfaces for registers, data access method for RMA, and data read and write operation by DMA.
- Card Interface Unit (CIU): It handles external memory card interface protocols. It also provides clock control.

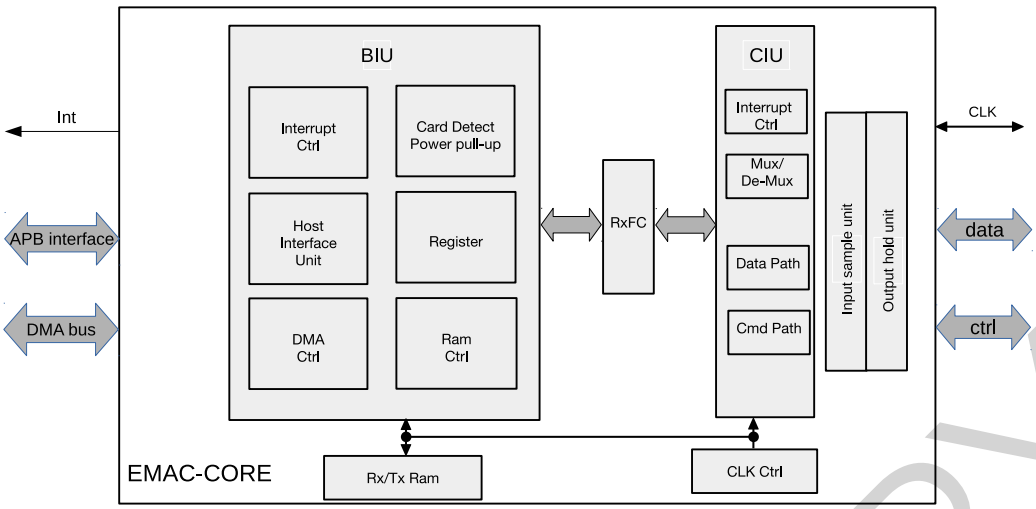


Figure 17-3. SDIO Host Block Diagram

17.4.1.1 Bus Interface Unit (BIU)

The BIU provides the access to registers and RAM data through the Host Interface Unit (HIU). Additionally, it provides a method to access to memory data through a DMA interface. Figure 17-3 illustrates the internal components of the BIU. Figure 17-9 illustrates the clcok slection. The BIU provides the following functions:

- Host interface
- DMA interface
- Interrupt control
- Register access
- FIFO access
- Power/pull-up control and card detection

17.4.1.2 Card Interface Unit (CIU)

The CIU module implements the card-specific protocols. Within the CIU, the command path control unit and data path control unit are used to interface with the command and data ports, respectively, of the SD/MMC/CE-ATA cards. The CIU also provides clock control. Figure 17-3 illustrates the internal structure of the CIU, which consists of the following primary functional blocks:

- Command path
- Data path
- SDIO interrupt control
- Clock control
- Mux/De-Mux unit

17.4.2 Command Path

The command path performs the following functions:

- Configures clock parameters
- Configures card command parameters
- Sends commands to card bus (sdhost_ccmd_out line)
- Receives responses from card bus (sdhost_ccmd_in line)
- Sends responses to BIU
- Drives the P-bit on the command line

The command path State Machine is shown in Figure 17-4.

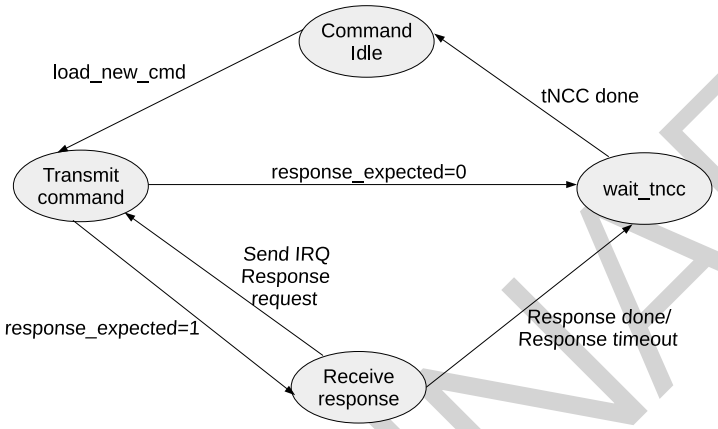


Figure 17-4. Command Path State Machine

17.4.3 Data Path

The data path block pops RAM data and transmits them on sdhost_cdata_out during a write-data transfer, or it receives data on sdhost_cdata_in and pushes them into RAM during a read-data transfer. The data path loads new data parameters, i.e., expected data, read/write data transfer, stream/block transfer, block size, byte count, card type, timeout registers, etc., whenever a data transfer command is not in progress.

If the SDHOST_DATA_EXPECTED bit is set in SDHOST_CMD_REG register, the new command is a data-transfer command and the data path starts one of the following operations:

- Transmitting data if the SDHOST_READ_WRITE bit is 1
- Receiving data if the SDHOST_READ_WRITE bit is 0

17.4.3.1 Data Transmit Operation

The module starts data transmission two clock cycles after a response for the data-write command is received. This occurs even if the command path detects a response error or a cyclic redundancy check (CRC) error in a response. If no response is received from the card until the response timeout, no data are transmitted. Depending on the value of the SDHOST_TRANSFER_MODE bit in SDHOST_CMD_REG register, the data-transmit state machine adds data to the card's data bus in a stream or in block(s). The data transmit state machine is shown in Figure 17-5.

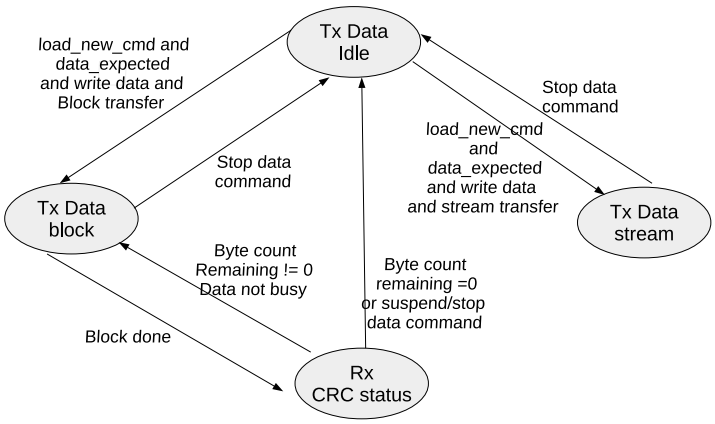


Figure 17-5. Data Transmit State Machine

17.4.3.2 Data Receive Operation

The module receives data two clock cycles after the end bit of a data-read command, even if the command path detects a response error or a CRC error. If no response is received from the card and a response timeout occurs, the BIU does not receive a signal about the completion of the data transfer. If the command sent by the CIU is an illegal operation for the card, it would prevent the card from starting a read-data transfer, and the BIU will not receive a signal about the completion of the data transfer.

If no data is received by the data timeout, the data path signals a data timeout to the BIU, which marks an end to the data transfer. Based on the value of the SDHOST_TRANSFER_MODE bit in SDHOST_CMD_REG register, the data-receive state machine gets data from the card's data bus in a stream or block(s). The data receive state machine is shown in Figure 17-6.

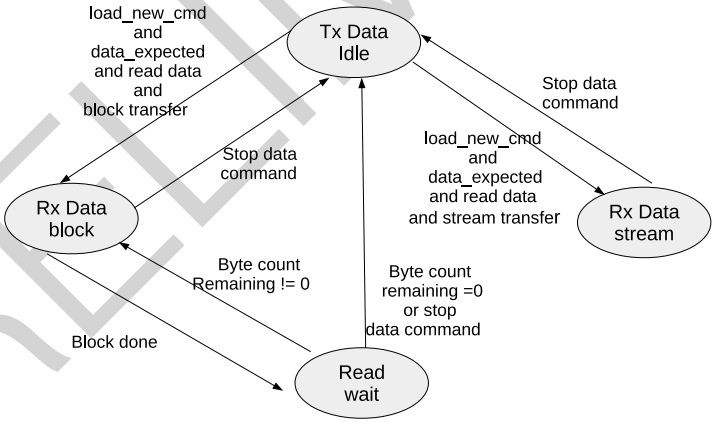


Figure 17-6. Data Receive State Machine

17.5 Software Restrictions for Proper CIU Operation

- Only one card at a time can be selected to execute a command or data transfer. For example, when data are being transferred to or from a card, a new command must not be issued to another card. A new command, however, can be issued to the same card, allowing it to read the device status or stop the transfer.
- Only one command at a time can be issued for data transfers.

- During an open-ended card-write operation, if the card clock is stopped due to RAM being empty, the software must fill RAM with data first, and then start the card clock. Only then can it issue a stop/abort command to the card.
- During an SDIO/Combo card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset RAM, setting SDHOST_FIFO_RESET bits and then issue the resume command as if it were a new data-transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52_reset), while a card data transfer is in progress, the software must set the SDHOST_STOP_ABORT_CMD bit in SDHOST_CMD_REG register, so that the CIU can stop the data transfer after issuing the card reset command.
- When the data's end bit error is set in the SDHOST_RINTSTS_REG register, the CIU does not guarantee SDIO interrupts. In such a case, the software ignores SDIO interrupts and issues a stop/abort command to the card, so that the card stops sending read-data.
- If the card clock is stopped due to RAM being full during a card read, the software will read at least two RAM locations to restart the card clock.
- Only one CE-ATA device at a time can be selected for a command or data transfer. For example, when data are transferred from a CE-ATA device, a new command should not be sent to another CE-ATA device.
- If a CE-ATA device's interrupts are enabled (nIEN=0), a new SDHOST_RW_BLK command should not be sent to the same device if the execution of a SDHODT_RW_BLK command is already in progress. Only the CCSD can be sent while waiting for the CCS.
- If, however, a CE-ATA device's interrupts are disabled (nIEN=1), a new command can be issued to the same device, allowing it to read status information.
- Open-ended transfers are not supported in CE-ATA devices.
- The sdhost_send_auto_stop signal is not supported (software should not set the sdhost_send_auto_stop bit) in CE-ATA transfers.

After configuring the command start bit to 1, the values of the following registers cannot be changed before a command has been issued:

- CMD - command
- CMDARG - command argument
- BYTCNT - byte count
- BLKSIZ - block size
- CLKDIV - clock divider
- CKLENA - clock enable
- CLKSRC - clock source
- TMOUT - timeout
- CTYPE - card type

17.6 RAM for Receiving and Sending Data

The submodule RAM is a buffer area for sending and receiving data. It can be divided into two units: the one is for sending data, and the other is for receiving data. The process of sending and receiving data can also be achieved by the CPU and DMA for reading and writing. The latter method is described in detail in Section 17.8.

17.6.1 TX RAM Module

There are two ways to enable a write operation: DMA and CPU read/write.

If SDIO-sending is enabled, data can be written to the TX RAM module by APB interface. Data will be written to register SDHOST_BUFFIFO_REG from the CPU, directly, by an APB interface.

Another way of data transmission is by DMA.

17.6.2 RX RAM Module

There are two ways to enable a read operation: DMA and CPU read/write.

When the data path receives data, the data will be written to the RX RAM. Then, these data can be read with the APB method at the reading end. Register SDHOST_BUFFIFO_REG can be read by the APB directly.

Another way of receiving data is by DMA.

17.7 DMA Descriptor Chain

Each linked list module consists of two parts: the linked list itself and a data buffer. In other words, each module points to a unique data buffer and the linked list that follows the module. Figure 17-7 shows the descriptor chain.

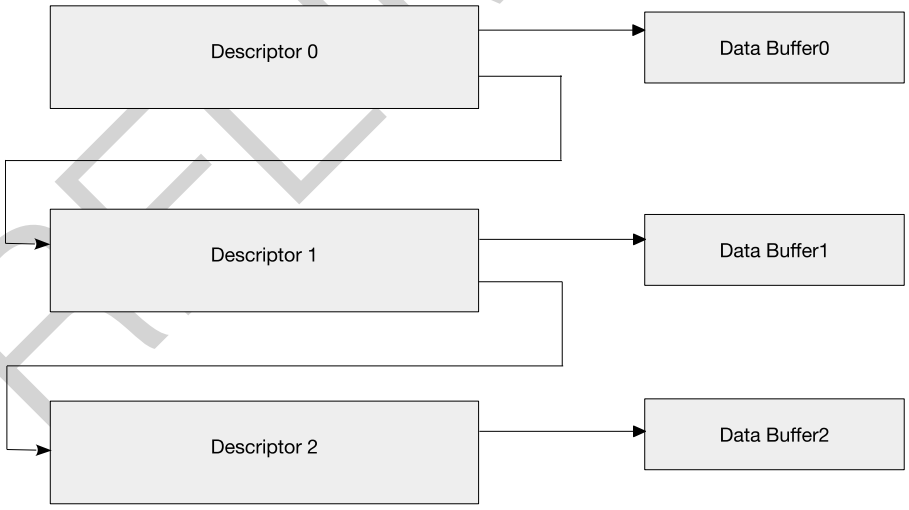


Figure 17-7. Descriptor Chain

17.8 The Structure of DMA descriptor chain

Each linked list consists of four words. As is shown below, Figure 17-8 demonstrates the linked list's structure, and Table 17-2, Table 17-3, Table 17-4, Table 17-5 provide the descriptions of linked lists.

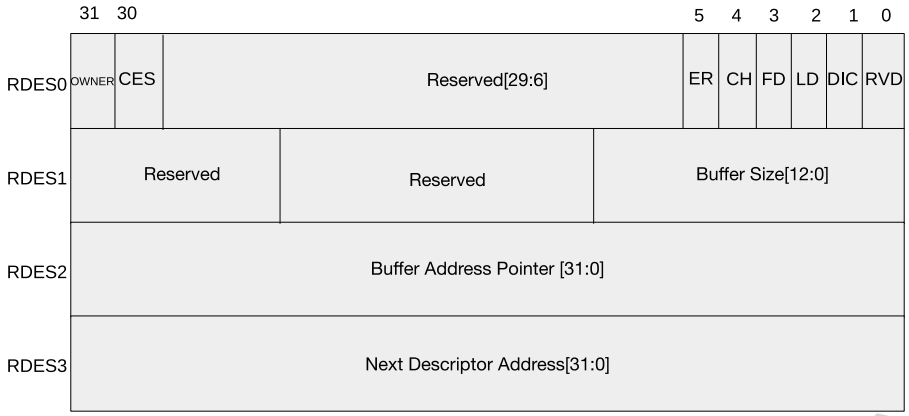


Figure 17-8. The Structure of a Linked List

The DES0 element contains control and status information.

Table 17-2. Word DES0 of SD/MMC GDMA Linked List

Bits	Name	Description
31	OWNER	When set, this bit indicates that the descriptor is owned by the DMA Controller. When reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit when it completes the data transfer.
30	CES (Card Error Summary)	These error bits indicate the status of the transition to or from the card. The following bits are also present in SDHOST_RINTSTS_REG, which indicates their digital logic OR gate. • EBE: End Bit Error • RTO: Response Time out • RCRC: Response CRC • SBE: Start Bit Error • DRTO: Data Read Timeout • DCRC: Data CRC for Receive • RE: Response Error
29:6	Reserved	Reserved
5	ER (End of Ring)	When set, this bit indicates that the descriptor list has reached its final descriptor. The DMA Controller then returns to the base address of the list, creating a Descriptor Chain.
4	CH (Second Address Chained)	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address. When this bit is set, BS2 (DES1[25:13]) should be all zeros.

Bits	Name	Description
3	FD (First Descriptor)	When set, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, the Next Descriptor contains the beginning of the data.
2	LD (Last Descriptor)	This bit is associated with the last block of a DMA transfer. When set, the bit indicates that the buffers pointed by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count should be 0.
1	DIC (Disable Interrupt on Completion)	When set, this bit will prevent the setting of the TI/RI bit of the DMA Status Register (IDSTS) for the data that ends in the buffer pointed by this descriptor.
0	Reserved	Reserved

The DES1 element contains the buffer size.

Table 17-3. Word DES1 of SD/MMC GDMA Linked List

Bits	Name	Description
31:26	Reserved	Reserved
25:13	Reserved	Reserved
12:0	BS (Buffer Size)	Indicates the size of the data buffer (in Byte), which must be a multiple of four. In the case where the buffer size is not a multiple of four, the resulting behavior is undefined. This field should not be zero.

The DES2 element contains the address pointer to the data buffer.

Table 17-4. Word DES2 of SD/MMC GDMA Linked List

Bits	Name	Description
31:0	Buffer Address Pointer	These bits indicate the physical address of the data buffer. And the buffer address must be word-aligned.

The DES3 element contains the address pointer to the next descriptor if the present descriptor is not the last one in a chained descriptor structure.

Table 17-5. Word DES3 of SD/MMC GDMA Linked List

Bits	Name	Description
31:0	Next Descriptor Address	If CH (DES0[4]) is set, this bit contains the address pointer to the next descriptor.

Bits	Name	Description
		If this is not the last descriptor in a chained descriptor structure, the address pointer to the next descriptor should be: DES3[1:0] = 0.

17.9 Initialization

17.9.1 DMA Initialization

The DMA Controller initialization should proceed as follows:

1. Write to the DMA Bus Mode Register (SDHOST_BMOD_REG) will set the Host bus's access parameters.
2. Write to the DMA Interrupt Enable Register (SDHOST_IDINTEN_REG) will mask any unnecessary interrupt causes.
3. The software driver creates either the inlink or the outlink descriptors. Then, it writes to the DMA Descriptor List Base Address Register (SDHOST_DBADDR_REG), providing the DMA Controller with the starting address of the list.
4. The DMA Controller engine attempts to acquire descriptors from descriptor lists.

17.9.2 DMA Transmission Initialization

The DMA transmission occurs as follows:

1. The Host sets up the elements (DES0-DES3) for transmission, and sets the OWNER bit (DES0[31]). The Host also prepares the data buffer.
2. The Host programs the write-data command in the CMD register in BIU.
3. The Host also programs the required transmit threshold (SDHOST_TX_WMARK field in SDHOST_FIFOTH_REG register).
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA Controller enters a suspend-state and asserts the Descriptor Unable interrupt in the SDHOST_IDSTS_REG register. In such a case, the host needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. It then waits for the Command Done (CD) bit in DHOST_RINTSTS_REG register and no errors from BIU, which indicates that a transfer has completed.
6. Subsequently, the DMA Controller engine waits for a DMA interface request from BIU. This request will be generated, based on the programmed transmit-threshold value. For the last bytes of data which cannot be accessed using a burst, single transfers are performed on the AHB Master Interface.
7. The DMA Controller fetches the transmit data from the data buffer in the Host memory and transfers them to RAM for transmission to card.
8. When data span across multiple descriptors, the DMA Controller fetches the next descriptor and extends its operation using the following descriptor. The last descriptor bit indicates whether the data span multiple descriptors or not.

9. When data transmission is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting the SDHOST_IDSTS_TI, if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by performing a write transaction to DES0.

17.9.3 DMA Reception Initialization

The DMA reception occurs as follows:

1. The Host sets up the element (DES0-DES3) for reception, and sets the OWNER bit (DES0[31]).
2. The Host programs the read-data command in the CMD register in BIU.
3. Then, the Host programs the required level of the receive-threshold (SDHOST_RX_WMARK field in SDHOST_FIFOTH_REG register).
4. The DMA Controller engine fetches the descriptor and checks the OWNER bit. If the OWNER bit is not set, it means that the host owns the descriptor. In this case, the DMA enters a suspend-state and asserts the Descriptor Unable interrupt in the SDHOST_IDSTS_REG register. In such a case, the host needs to release the DMA Controller by writing any value to SDHOST_PLDMND_REG.
5. It then waits for the Command Done (CD) bit and no errors from BIU, which indicates that a reception can be done.
6. The DMA Controller engine then waits for a DMA interface request from BIU. This request will be generated, based on the programmed receive-threshold value. For the last bytes of the data which cannot be accessed using a burst, single transfers are performed on the AHB.
7. The DMA Controller fetches the data from RAM and transfers them to the Host memory.
8. When data span across multiple descriptors, the DMA Controller will fetch the next descriptor and extend its operation using the following descriptor. The last descriptor bit indicates whether the data span multiple descriptors or not.
9. When data reception is complete, the status information is updated in the SDHOST_IDSTS_REG register by setting SDHOST_IDSTS_RI, if it has already been enabled. Also, the OWNER bit is cleared by the DMA Controller by performing a write-transaction to DES0.

17.10 Clock Phase Selection

If the setup time requirements for the input or output data signal are not met, users can specify the clock phase, as shown in the figure 17-9.

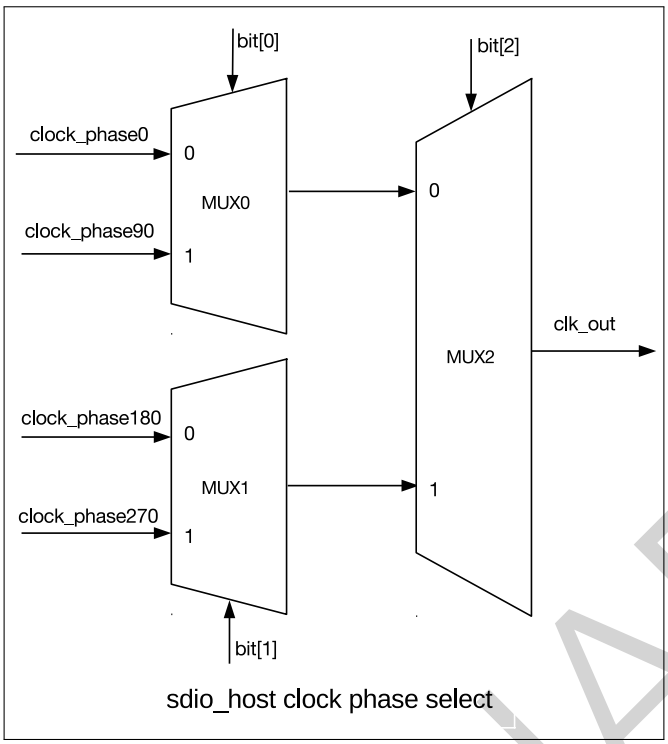


Figure 17-9. Clock Phase Selection

This issue can be fixed by configuring register SDHOST_CLK_DIV_EDGE_REG. For example, set CCLKIN_EDGE_DRV_SEL bit to 0 to drive the output data in phase0, and set the CCLKIN_EDGE_SAM_SEL bit to 1 to select phase90 to sample the data from SDIO slave, if there are still timing issue, please set bit 4 or 6 to use phase180 or phase 270 to sample the data from SDIO slave.

Please find detailed information on the clock phase selection register SDHOST_CLK_DIV_EDGE_REG in Section Registers.

Table 17-6. SDHOST Clk Phase Selection

Clock phase	phase_select value
0	0
90	1
180	4
270	6

17.11 Interrupt

Interrupts can be generated as a result of various events. The SDHOST_IDSTS_REG register contains all the bits that might cause an interrupt. The SDHOST_IDINTEN_REG register contains an enable bit for each of the events that can cause an interrupt.

There are two groups of summary interrupts, ”Normal” ones (bit8 SDHOST_IDSTS_NIS) and ”Abnormal” ones (bit9 SDHOST_IDSTS_AIS), as outlined in the SDHOST_IDSTS_REG register. Interrupts are cleared by writing 1 to the position of the corresponding bit. When all the enabled interrupts within a group are cleared, the corresponding summary bit is also cleared. When both summary bits are cleared, the interrupt signal connected to CPU is de-asserted (stops signalling).

Interrupts are not queued up, and if a new interrupt-event occurs before the driver has responded to it, no additional interrupts are generated. For example, the SDHOST_IDSTS_RI indicates that one or more data were transferred to the Host buffer.

An interrupt is generated only once for concurrent events. The driver must scan the SDHOST_IDSTS_REG register for the interrupt cause.

17.12 Register Summary

The addresses in this section are relative to SD/MMC Host Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

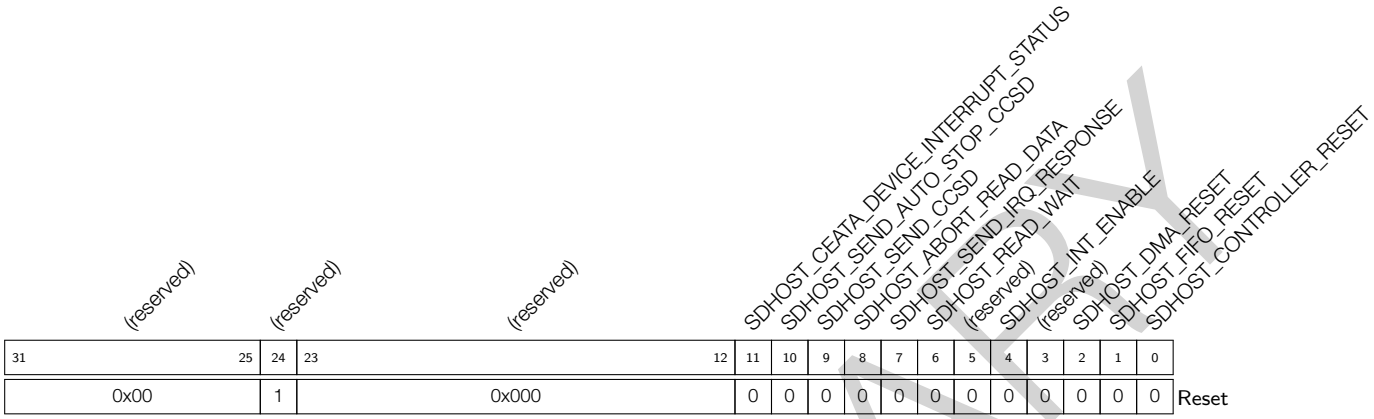
Name	Description	Address	Access
SDHOST_CTRL_REG	Control register	0x0000	R/W
SDHOST_CLKDIV_REG	Clock divider configuration register	0x0008	R/W
SDHOST_CLKSRC_REG	Clock source selection register	0x000C	R/W
SDHOST_CLKENA_REG	Clock enable register	0x0010	R/W
SDHOST_TMOUT_REG	Data and response timeout configuration register	0x0014	R/W
SDHOST_CTYPE_REG	Card bus width configuration register	0x0018	R/W
SDHOST_BLKSIZ_REG	Card data block size configuration register	0x001C	R/W
SDHOST_BYTCNT_REG	Data transfer length configuration register	0x0020	R/W
SDHOST_INTMASK_REG	SDIO interrupt mask register	0x0024	R/W
SDHOST_CMDARG_REG	Command argument data register	0x0028	R/W
SDHOST_CMD_REG	Command and boot configuration register	0x002C	R/W
SDHOST_RESP0_REG	Response data register	0x0030	RO
SDHOST_RESP1_REG	Long response data register	0x0034	RO
SDHOST_RESP2_REG	Long response data register	0x0038	RO
SDHOST_RESP3_REG	Long response data register	0x003C	RO
SDHOST_MINTSTS_REG	Masked interrupt status register	0x0040	RO
SDHOST_RINTSTS_REG	Raw interrupt status register	0x0044	R/W
SDHOST_STATUS_REG	SD/MMC status register	0x0048	RO
SDHOST_FIFOTH_REG	FIFO configuration register	0x004C	R/W
SDHOST_CDETECT_REG	Card detect register	0x0050	RO
SDHOST_WRTPRT_REG	Card write protection (WP) status register	0x0054	RO
SDHOST_TCBCNT_REG	Transferred byte count register	0x005C	RO
SDHOST_TBBCNT_REG	Transferred byte count register	0x0060	RO
SDHOST_DEBNCE_REG	Debounce filter time configuration register	0x0064	R/W
SDHOST_USRID_REG	User ID (scratchpad) register	0x0068	R/W
SDHOST_VERID_REG	Version ID (scratchpad) register	0x006C	RO
SDHOST_HCON_REG	Hardware feature register	0x0070	RO
SDHOST_UHS_REG	UHS-1 register	0x0074	R/W
SDHOST_RST_N_REG	Card reset register	0x0078	R/W
SDHOST_BMOD_REG	Burst mode transfer configuration register	0x0080	R/W
SDHOST_PLDMND_REG	Poll demand configuration register	0x0084	WO
SDHOST_DBADDR_REG	Descriptor base address register	0x0088	R/W
SDHOST_IDSTS_REG	IDMAC status register	0x008C	R/W
SDHOST_IDINTEN_REG	IDMAC interrupt enable register	0x0090	R/W
SDHOST_DSCADDR_REG	Host descriptor address pointer	0x0094	RO
SDHOST_BUFADDR_REG	Host buffer address pointer register	0x0098	RO
SDHOST_CARDTHRCTL_REG	Card Threshold Control register	0x0100	R/W
SDHOST_EMMCDDR_REG	eMMC DDR register	0x010C	R/W
SDHOST_ENSHIFT_REG	Enable Phase Shift register	0x0110	R/W

Name	Description	Address	Access
SDHOST_BUFFIFO_REG	CPU write and read transmit data by FIFO	0x0200	R/W
SDHOST_CLK_DIV_EDGE_REG	Clock phase selection register	0x0800	R/W

17.13 Registers

The addresses in this section are relative to SD/MMC Host Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 17.1. SDHOST_CTRL_REG (0x0000)



SDHOST_CEATA_DEVICE_INTERRUPT_STATUS Software should appropriately write to this bit after the power-on reset or any other reset to the CE-ATA device. After reset, the CE-ATA device's interrupt is usually disabled (nIEN = 1). If the host enables the CE-ATA device's interrupt, then software should set this bit. (R/W)

SDHOST_SEND_AUTO_STOP_CCSD Always set SDHOST_SEND_AUTO_STOP_CCSD and SDHOST_SEND_CCSD bits together; SDHOST_SEND_AUTO_STOP_CCSD should not be set independently of send_ccsd. When set, SD/MMC automatically sends an internally-generated STOP command (CMD12) to the CE-ATA device. After sending this internally-generated STOP command, the Auto Command Done (ACD) bit in SDHOST_RINTSTS_REG is set and an interrupt is generated for the host, in case the ACD interrupt is not masked. After sending the Command Completion Signal Disable (CCSD), SD/MMC automatically clears the SDHOST_SEND_AUTO_STOP_CCSD bit. (R/W)

SDHOST_SEND_CCSD When set, SD/MMC sends CCSD to the CE-ATA device. Software sets this bit only if the current command is expecting CCS (that is, RW_BLK), and if interrupts are enabled for the CE-ATA device. Once the CCSD pattern is sent to the device, SD/MMC automatically clears the SDHOST_SEND_CCSD bit. It also sets the Command Done (CD) bit in the SDHOST_RINTSTS_REG register, and generates an interrupt for the host, in case the Command Done interrupt is not masked.

NOTE: Once the SDHOST_SEND_CCSD bit is set, it takes two card clock cycles to drive the CCSD on the CMD line. Due to this, within the boundary conditions the CCSD may be sent to the CE-ATA device, even if the device has signalled CCS. (R/W)

Continued on the next page...

Register 17.1. SDHOST_CTRL_REG (0x0000)

Continued from the previous page...

SDHOST_ABORT_READ_DATA After a suspend-command is issued during a read-operation, software polls the card to find when the suspend-event occurred. Once the suspend-event has occurred, software sets the bit which will reset the data state machine that is waiting for the next block of data. This bit is automatically cleared once the data state machine is reset to idle. (R/W)

SDHOST_SEND_IRQ_RESPONSE Bit automatically clears once response is sent. To wait for MMC card interrupts, host issues CMD40 and waits for interrupt response from MMC card(s). In the meantime, if host wants SD/MMC to exit waiting for interrupt state, it can set this bit, at which time SD/MMC command state-machine sends CMD40 response on bus and returns to idle state. (R/W)

SDHOST_READ_WAIT For sending read-wait to SDIO cards. (R/W)

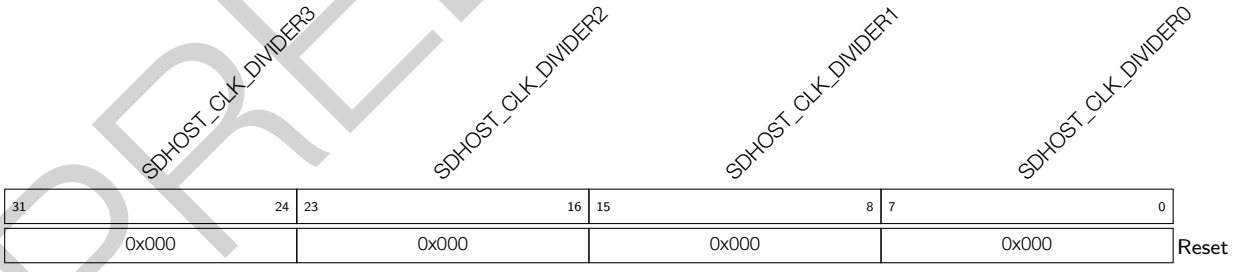
SDHOST_INT_ENABLE Global interrupt enable/disable bit. 0: Disable; 1: Enable. (R/W)

SDHOST_DMA_RESET To reset DMA interface, firmware should set bit to 1. This bit is auto-cleared after two AHB clocks. (R/W)

SDHOST_FIFO_RESET To reset FIFO, firmware should set bit to 1. This bit is auto-cleared after completion of reset operation.
Note: FIFO pointers will be out of reset after 2 cycles of system clocks in addition to synchronization delay (2 cycles of card clock), after the fifo_reset is cleared. (R/W)

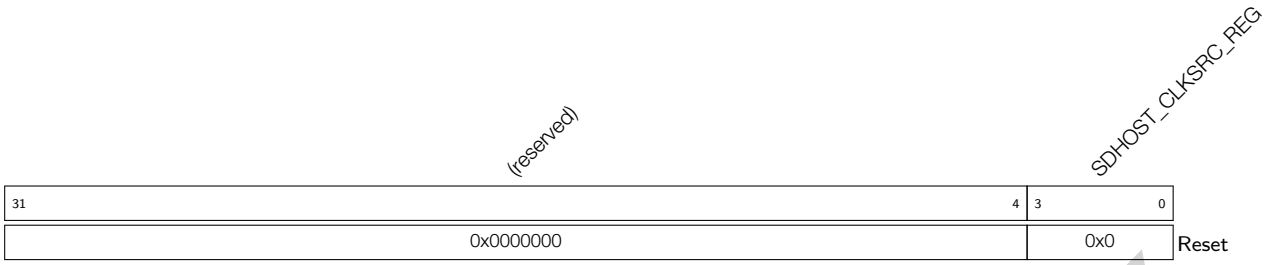
SDHOST_CONTROLLER_RESET To reset controller, firmware should set this bit. This bit is auto-cleared after two AHB and two sdhost_cclk_in clock cycles. (R/W)

Register 17.2. SDHOST_CLKDIV_REG (0x0008)



SDHOST_CLK_DIVIDER*m* Clock divider (*m*) value. Clock divisor is 2**n*, where *n* = 0 bypasses the divider (divisor of 1). For example, a value of 1 means divided by 2*1 = 2, a value of 0xFF means divided by 2*255 = 510, and so on. The range of *m* is 0 ~ 3. (R/W)

Register 17.3. SDHOST_CLKSRC_REG (0x000C)



SDHOST_CLKSRC_REG Clock divider source for two SD cards is supported. Each card has two bits assigned to it. For example, bit[1:0] are assigned for card 0, bit[3:2] are assigned for card 1. Card 0 maps and internally routes clock divider[0:3] outputs to cclk_out[1:0] pins, depending on bit value. (R/W)

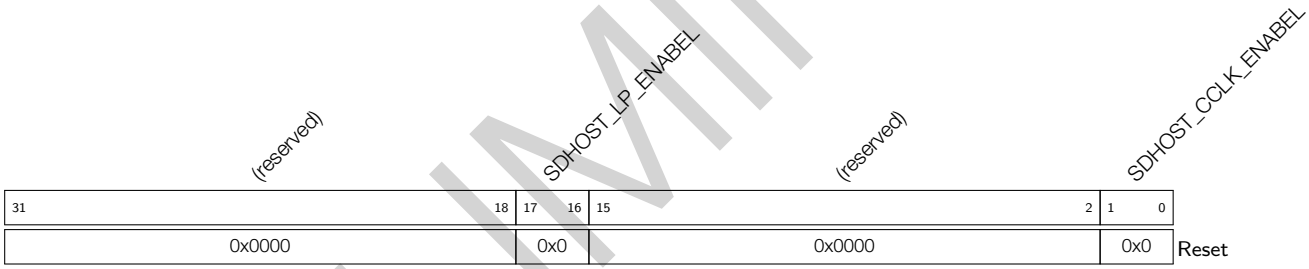
00 : Clock divider 0;

01 : Clock divider 1;

10 : Clock divider 2;

11 : Clock divider 3.

Register 17.4. SDHOST_CLKENA_REG (0x0010)



SDHOST_LP_ENABLE Disable clock when the card is in IDLE state. One bit per card. (R/W)

0: clock disabled;

1: clock enabled.

SDHOST_CCLK_ENABLE Clock-enable control for two SD card clocks and one MMC card clock is supported. One bit per card. (R/W)

0: Clock disabled;

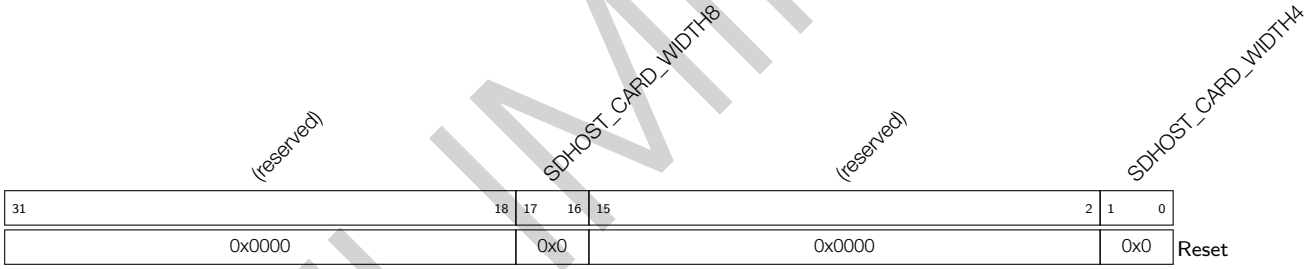
1: Clock enabled.

Register 17.5. SDHOST_TMOUT_REG (0x0014)

SDHOST_DATA_TIMEOUT Value for card data read timeout. This value is also used for data starvation by host timeout. The timeout counter is started only after the card clock is stopped. This value is specified in number of card output clocks, i.e. sdhost_cclk_out of the selected card. (R/W)
NOTE: The software timer should be used if the timeout value is in the order of 100 ms. In this case, read data timeout interrupt needs to be disabled.

SDHOST_RESPONSE_TIMEOUT Response timeout value. Value is specified in terms of number of card output clocks, i.e., sdhost_cclk_out. (R/W)

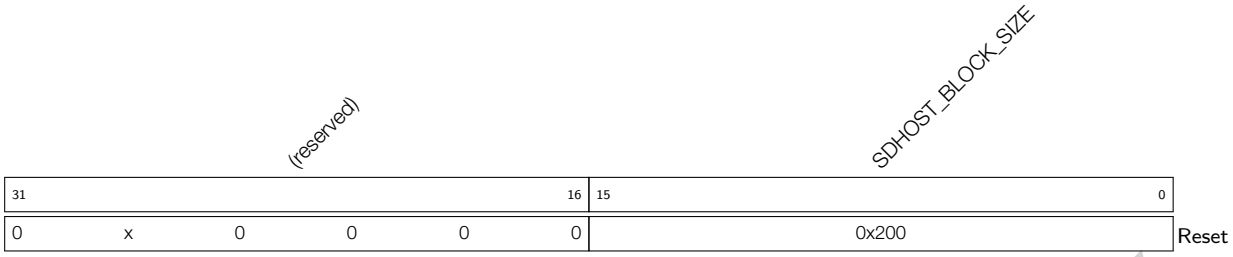
Register 17.6. SDHOST_CTYPE_REG (0x0018)



SDHOST_CARD_WIDTH8 One bit per card indicates if card is in 8-bit mode. (R/W)
0: Non 8-bit mode;
1: 8-bit mode.
Bit[17:16] correspond to card[1:0] respectively.

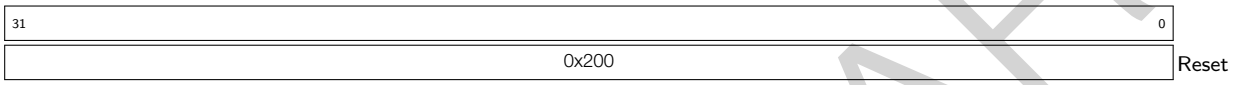
SDHOST_CARD_WIDTH4 One bit per card indicates if card is 1-bit or 4-bit mode. (R/W)
0: 1-bit mode;
1: 4-bit mode.
Bit[1:0] correspond to card[1:0] respectively.

Register 17.7. SDHOST_BLKSIZ_REG (0x001C)



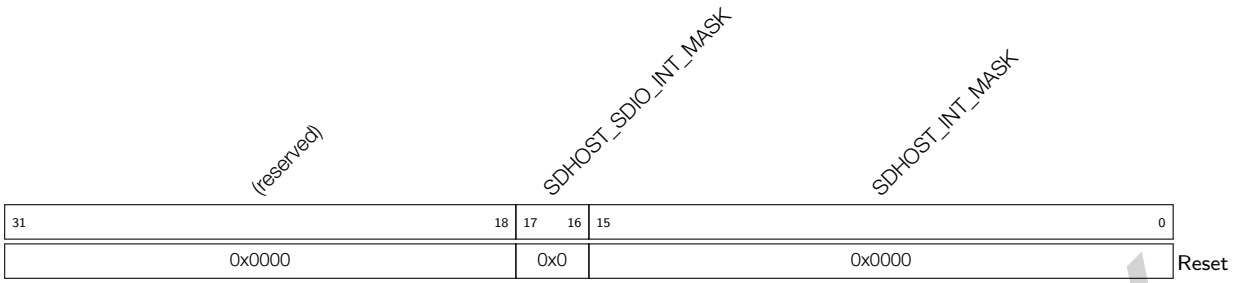
SDHOST_BLOCK_SIZE Block size. (R/W)

Register 17.8. SDHOST_BYTCNT_REG (0x0020)



SDHOST_BYTCNT_REG Number of bytes to be transferred, should be an integral multiple of Block Size for block transfers. For data transfers of undefined byte lengths, byte count should be set to 0. When byte count is set to 0, it is the responsibility of host to explicitly send stop/abort command to terminate data transfer. (R/W)

Register 17.9. SDHOST_INTMASK_REG (0x0024)

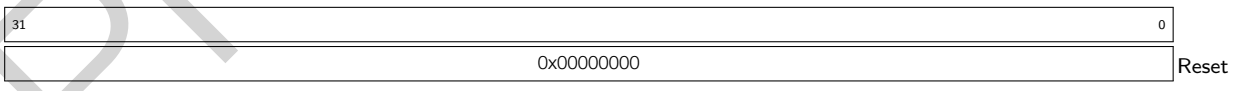


SDHOST_SDIO_INT_MASK SDIO interrupt mask, one bit for each card. Bit[17:16] correspond to card[15:0] respectively. When masked, SDIO interrupt detection for that card is disabled. 0 masks an interrupt, and 1 enables an interrupt. (R/W)

SDHOST_INT_MASK These bits used to mask unwanted interrupts. A value of 0 masks interrupt, and a value of 1 enables the interrupt. (R/W)

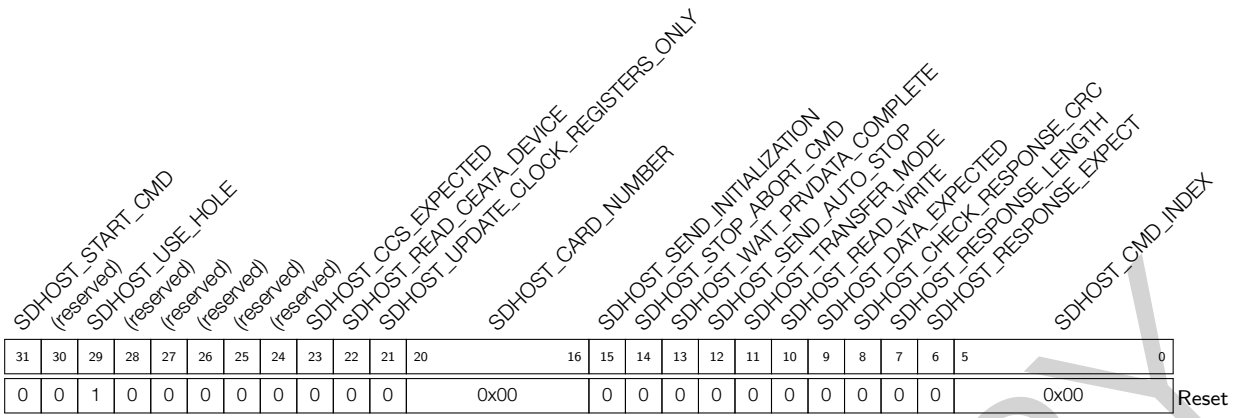
Bit 15 (EBE): End-bit error/no CRC error;
Bit 14 (ACD): Auto command done;
Bit 13 (SBE/BCI): Rx Start Bit Error;
Bit 12 (HLE): Hardware locked write error;
Bit 11 (FRUN): FIFO underrun/overrun error;
Bit 10 (HTO): Data starvation-by-host timeout;
Bit 9 (DRTO): Data read timeout;
Bit 8 (RTO): Response timeout;
Bit 7 (DCRC): Data CRC error;
Bit 6 (RCRC): Response CRC error;
Bit 5 (RXDR): Receive FIFO data request;
Bit 4 (TXDR): Transmit FIFO data request;
Bit 3 (DTO): Data transfer over;
Bit 2 (CD): Command done;
Bit 1 (RE): Response error;
Bit 0 (CD): Card detect.

Register 17.10. SDHOST_CMDARG_REG (0x0028)



SDHOST_CMDARG_REG Value indicates command argument to be passed to the card. (R/W)

Register 17.11. SDHOST_CMD_REG (0x002C)



SDHOST_START_CMD Start command. Once command is served by the CIU, this bit is automatically cleared. When this bit is set, host should not attempt to write to any command registers. If a write is attempted, hardware lock error is set in raw interrupt register. Once command is sent and a response is received from SD_MMC_CEATA cards, Command Done bit is set in the raw interrupt Register. (R/W)

SDHOST_USE_HOLE Use Hold Register. (R/W)

0: CMD and DATA sent to card bypassing HOLD Register;

1: CMD and DATA sent to card through the HOLD Register.

SDHOST_CCS_EXPECTED Expected Command Completion Signal (CCS) configuration. (R/W)

0: Interrupts are not enabled in CE-ATA device (nIEN = 1 in ATA control register), or command does not expect CCS from device;

1: Interrupts are enabled in CE-ATA device (nIEN = 0), and RW_BLK command expects command completion signal from CE-ATA device.

If the command expects Command Completion Signal (CCS) from the CE-ATA device, the software should set this control bit. SD/MMC sets Data Transfer Over (DTO) bit in RINTSTS register and generates interrupt to host if Data Transfer Over interrupt is not masked.

SDHOST_READ_CEATA_DEVICE Read access flag. (R/W)

0: Host is not performing read access (RW_REG or RW_BLK)towards CE-ATA device;

1: Host is performing read access (RW_REG or RW_BLK) towards CE-ATA device.

Software should set this bit to indicate that CE-ATA device is being accessed for read transfer. This bit is used to disable read data timeout indication while performing CE-ATA read transfers. Maximum value of I/O transmission delay can be no less than 10 seconds. SD/MMC should not indicate read data timeout while waiting for data from CE-ATA device.

Continued on the next page...

Register 17.11. SDHOST_CMD_REG (0x002C)

Continued from the previous page...

SDHOST_UPDATE_CLOCK_REGISTERS_ONLY 0: Normal command sequence; 1: Do not send commands, just update clock register value into card clock domain. (R/W)
Following register values are transferred into card clock domain: CLKDIV, CLRSRC, and CLKENA. Changes card clocks (change frequency, truncate off or on, and set low-frequency mode). This is provided in order to change clock frequency or stop clock without having to send command to cards.
During normal command sequence, when sdhost_update_clock_registers_only = 0, following control registers are transferred from BIU to CIU: CMD, CMDARG, TMOUT, CTYPE, BLKSIZ, and BYTCNT. CIU uses new register values for new command sequence to card(s). When bit is set, there are no Command Done interrupts because no command is sent to SD_MMC_CEATA cards.

SDHOST_CARD_NUMBER Card number in use. Represents physical slot number of card being accessed. In SD-only mode, up to two cards are supported. (R/W)

SDHOST_SEND_INITIALIZATION 0: Do not send initialization sequence (80 clocks of 1) before sending this command; 1: Send initialization sequence before sending this command. (R/W)
After powered on, 80 clocks must be sent to card for initialization before sending any commands to card. Bit should be set while sending first command to card so that controller will initialize clocks before sending command to card.

SDHOST_STOP_ABORT_CMD 0: Neither stop nor abort command can stop current data transfer. If abort is sent to function-number currently selected or not in data-transfer mode, then bit should be set to 0; 1: Stop or abort command intended to stop current data transfer in progress. (R/W)
When open-ended or predefined data transfer is in progress, and host issues stop or abort command to stop data transfer, bit should be set so that command/data state-machines of CIU can return correctly to idle state.

SDHOST_WAIT_PRVDATA_COMPLETE 0: Send command at once, even if previous data transfer has not completed; 1: Wait for previous data transfer to complete before sending Command. (R/W)
The SDHOST_WAIT_PRVDATA_COMPLETE] = 0 option is typically used to query status of card during data transfer or to stop current data transfer. SDHOST_CARD_NUMBERr should be same as in previous command.

SDHOST_SEND_AUTO_STOP 0: No stop command is sent at the end of data transfer; 1: Send stop command at the end of data transfer. (R/W)

Continued on the next page...

Register 17.11. SDHOST_CMD_REG (0x002C)

Continued from the previous page ...

SDHOST_TRANSFER_MODE 0: Block data transfer command; 1: Stream data transfer command. (R/W)
Don't care if no data expected.

SDHOST_READ_WRITE 0: Read from card; 1: Write to card.
Don't care if no data is expected from card. (R/W)

SDHOST_DATA_EXPECTED 0: No data transfer expected; 1: Data transfer expected. (R/W)

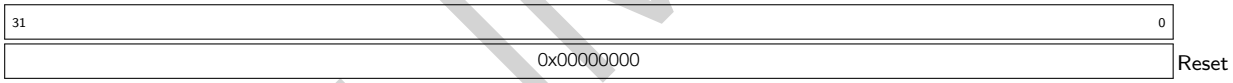
SDHOST_CHECK_RESPONSE_CRC 0: Do not check; 1: Check response CRC.
Some of command responses do not return valid CRC bits. Software should disable CRC checks for those commands in order to disable CRC checking by controller. (R/W)

SDHOST_RESPONSE_LENGTH 0: Short response expected from card; 1: Long response expected from card. (R/W)

SDHOST_RESPONSE_EXPECT 0: No response expected from card; 1: Response expected from card. (R/W)

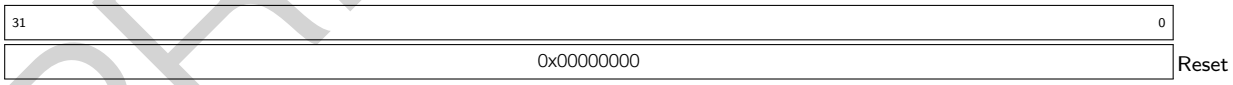
SDHOST_CMD_INDEX Command index. (R/W)

Register 17.12. SDHOST_RESP0_REG (0x0030)



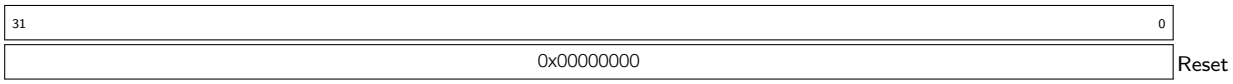
SDHOST_RESP0_REG Bit[31:0] of response. (RO)

Register 17.13. SDHOST_RESP1_REG (0x0034)



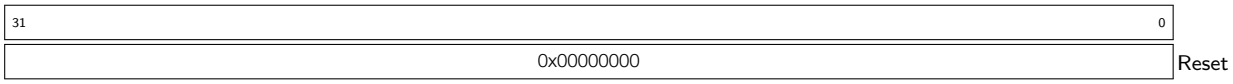
SDHOST_RESP1_REG Bit[63:32] of long response. (RO)

Register 17.14. SDHOST_RESP2_REG (0x0038)



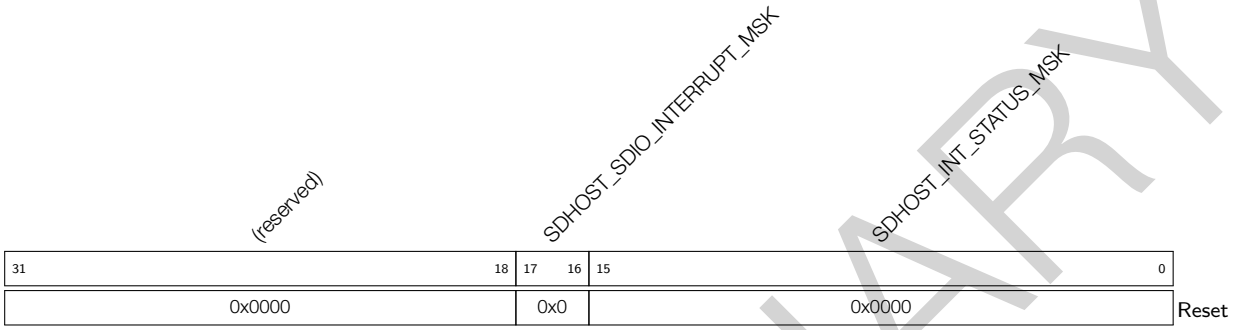
SDHOST_RESP2_REG Bit[95:64] of long response. (RO)

Register 17.15. SDHOST_RESP3_REG (0x003C)



SDHOST_RESP3_REG Bit[127:96] of long response. (RO)

Register 17.16. SDHOST_MINTSTS_REG (0x0040)

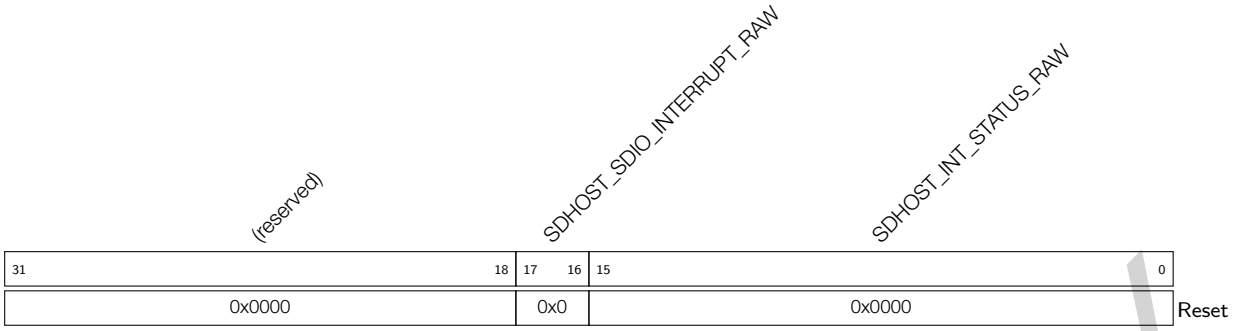


SDHOST_SDIO_INTERRUPT_MSK Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. SDIO interrupt for card is enabled only if corresponding sdhost_sdio_int_mask bit is set in Interrupt mask register (Setting mask bit enables interrupt). (RO)

SDHOST_INT_STATUS_MSK Interrupt enabled only if corresponding bit in interrupt mask register is set. (RO)

Bit 15 (EBE): End-bit error/no CRC error;
Bit 14 (ACD): Auto command done;
Bit 13 (SBE/BCI): RX Start Bit Error;
Bit 12 (HLE): Hardware locked write error;
Bit 11 (FRUN): FIFO underrun/overrun error;
Bit 10 (HTO): Data starvation by host timeout (HTO);
Bit 9 (DTRO): Data read timeout;
Bit 8 (RTO): Response timeout;
Bit 7 (DCRC): Data CRC error;
Bit 6 (RCRC): Response CRC error;
Bit 5 (RXDR): Receive FIFO data request;
Bit 4 (TXDR): Transmit FIFO data request;
Bit 3 (DTO): Data transfer over;
Bit 2 (CD): Command done;
Bit 1 (RE): Response error;
Bit 0 (CD): Card detect.

Register 17.17. SDHOST_RINTSTS_REG (0x0044)



SDHOST_SDIO_INTERRUPT_RAW Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. Setting a bit clears the corresponding interrupt bit and writing 0 has no effect. (R/W)

0: No SDIO interrupt from card;

1: SDIO interrupt from card.

SDHOST_INT_STATUS_RAW Setting a bit clears the corresponding interrupt and writing 0 has no effect. Bits are logged regardless of interrupt mask status. (R/W)

Bit 15 (EBE): End-bit error/no CRC error;

Bit 14 (ACD): Auto command done;

Bit 13 (SBE/BCI): RX Start Bit Error;

Bit 12 (HLE): Hardware locked write error;

Bit 11 (FRUN): FIFO underrun/overrun error;

Bit 10 (HTO): Data starvation by host timeout (HTO);

Bit 9 (DTRO): Data read timeout;

Bit 8 (RTO): Response timeout;

Bit 7 (DCRC): Data CRC error;

Bit 6 (RCRC): Response CRC error;

Bit 5 (RXDR): Receive FIFO data request;

Bit 4 (TXDR): Transmit FIFO data request;

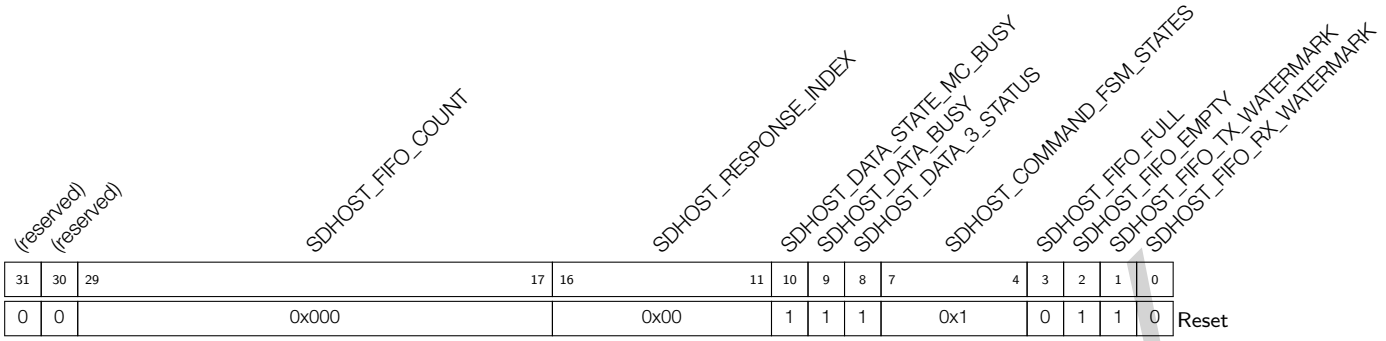
Bit 3 (DTO): Data transfer over;

Bit 2 (CD): Command done;

Bit 1 (RE): Response error;

Bit 0 (CD): Card detect.

Register 17.18. SDHOST_STATUS_REG (0x0048)



SDHOST_FIFO_COUNT FIFO count, number of filled locations in FIFO. (RO)

SDHOST_RESPONSE_INDEX Index of previous response, including any auto-stop sent by core. (RO)

SDHOST_DATA_STATE_MC_BUSY Data transmit or receive state-machine is busy. (RO)

SDHOST_DATA_BUSY Inverted version of raw selected sdhost_card_data[0].

0: Card data not busy;

1: Card data busy. (RO)

SDHOST_DATA_3_STATUS Raw selected sdhost_card_data[3], checks whether card is present.

0: card not present;

1: card present. (RO)

SDHOST_COMMAND_FSM_STATES Command FSM states. (RO)

0: Idle;

1: Send init sequence;

2: Send cmd start bit;

3: Send cmd tx bit;

4: Send cmd index + arg;

5: Send cmd crc7;

6: Send cmd end bit;

7: Receive resp start bit;

8: Receive resp IRQ response;

9: Receive resp tx bit;

10: Receive resp cmd idx;

11: Receive resp data;

12: Receive resp crc7;

13: Receive resp end bit;

14: Cmd path wait NCC;

15: Wait, cmd-to-response turnaround.

Continued on the next page...

Register 17.18. SDHOST_STATUS_REG (0x0048)

Continued from the previous page ...

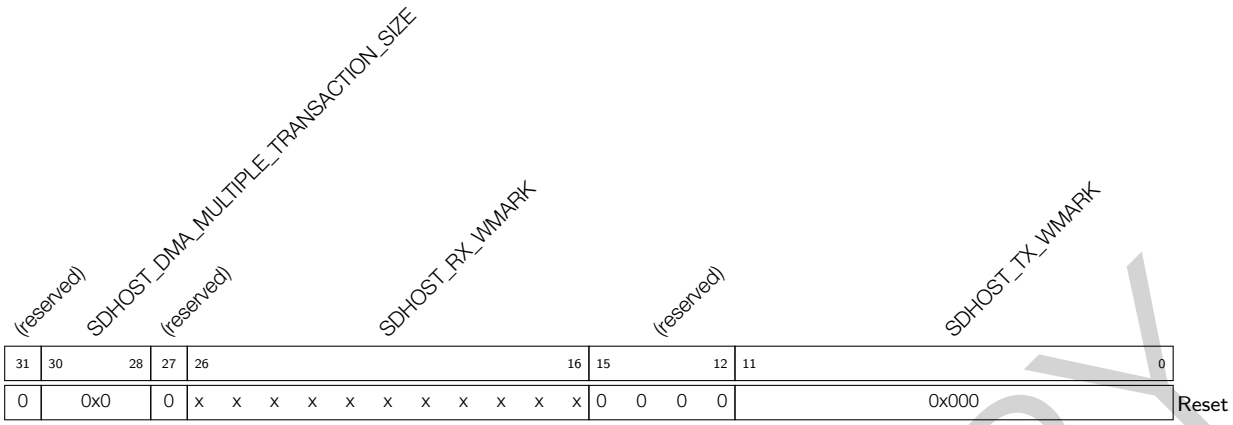
SDHOST_FIFO_FULL FIFO is full status. (RO)

SDHOST_FIFO_EMPTY FIFO is empty status. (RO)

SDHOST_FIFO_TX_WATERMARK FIFO reached Transmit watermark level, not qualified with data transfer. (RO)

SDHOST_FIFO_RX_WATERMARK FIFO reached Receive watermark level, not qualified with data transfer. (RO)

Register 17.19. SDHOST_FIFOTH_REG (0x004C)



SDHOST_DMA_MULTIPLE_TRANSACTION_SIZE Burst size of multiple transaction, should be programmed same as DMA controller multiple-transaction-size SDHOST_SRC/DEST_MSIZE. (R/W)

000: 1-byte transfer;

001: 4-byte transfer;

010: 8-byte transfer;

011: 16-byte transfer;

100: 32-byte transfer;

101: 64-byte transfer;

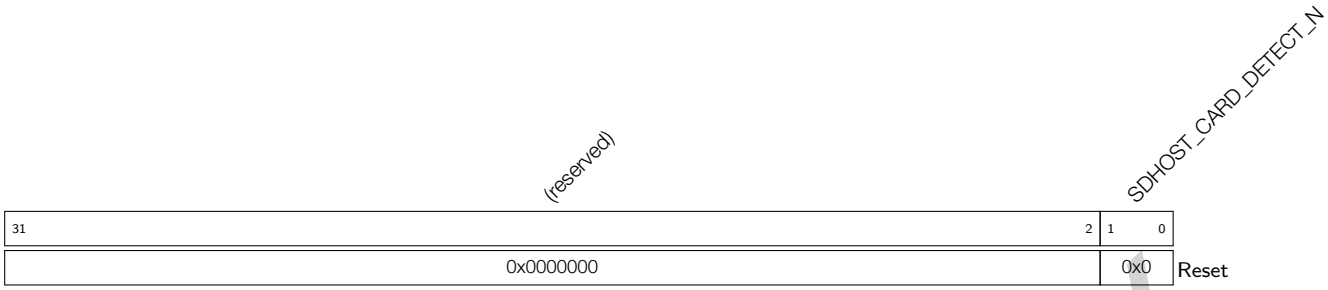
110: 128-byte transfer;

111: 256-byte transfer.

SDHOST_RX_WMARK FIFO threshold watermark level when receiving data to card.When FIFO data count reaches greater than this number , DMA/FIFO request is raised. During end of packet, request is generated regardless of threshold programming in order to complete any remaining data.In non-DMA mode, when receiver FIFO threshold (RXDR) interrupt is enabled, then interrupt is generated instead of DMA request.During end of packet, interrupt is not generated if threshold programming is larger than any remaining data. It is responsibility of host to read remaining bytes on seeing Data Transfer Done interrupt.In DMA mode, at end of packet, even if remaining bytes are less than threshold, DMA request does single transfers to flush out any remaining bytes before Data Transfer Done interrupt is set. (R/W)

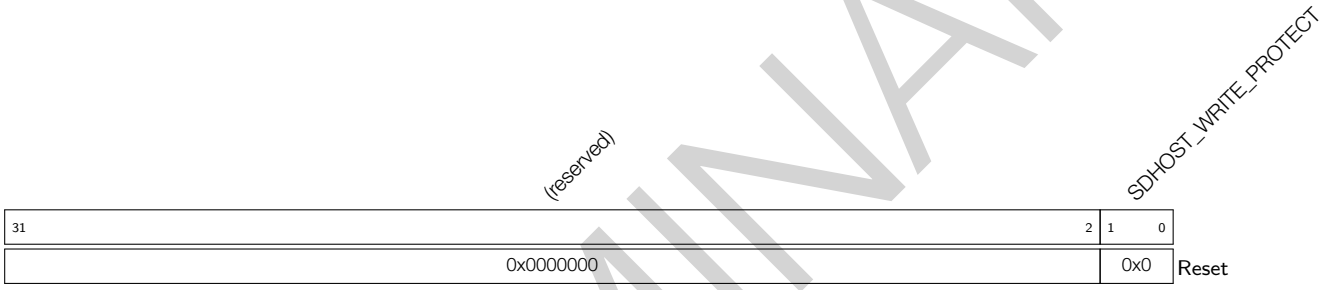
SDHOST_TX_WMARK FIFO threshold watermark level when transmitting data to card. When FIFO data count is less than or equal to this number, DMA/FIFO request is raised. If Interrupt is enabled, then interrupt occurs. During end of packet, request or interrupt is generated, regardless of threshold programming.In non-DMA mode, when transmit FIFO threshold (TXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, on last interrupt, host is responsible for filling FIFO with only required remaining bytes (not before FIFO is full or after CIU completes data transfers, because FIFO may not be empty). In DMA mode, at end of packet, if last transfer is less than burst size, DMA controller does single cycles until required bytes are transferred. (R/W)

Register 17.20. SDHOST_CDETECT_REG (0x0050)



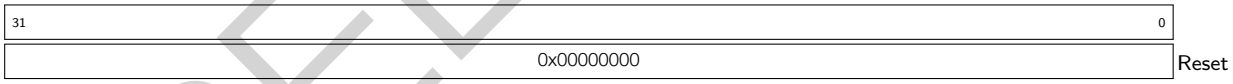
SDHOST_CARD_DETECT_N Value on sdhost_card_detect_n input ports (1 bit per card), read-only bits. 0 represents presence of card. Only NUM_CARDS number of bits are implemented. (RO)

Register 17.21. SDHOST_WRTPRT_REG (0x0054)



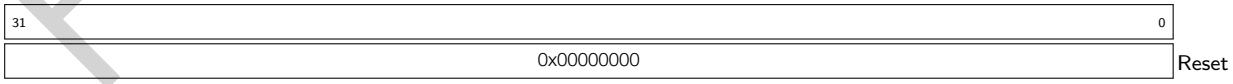
SDHOST_WRITE_PROTECT Value on sdhost_card_write_prt input ports (1 bit per card). 1 represents write protection. Only NUM_CARDS number of bits are implemented. (RO)

Register 17.22. SDHOST_TCBCNT_REG (0x005C)



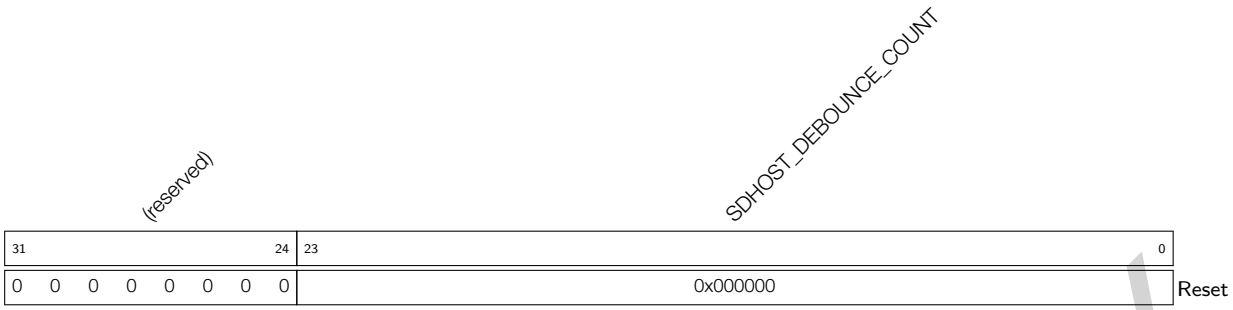
SDHOST_TCBCNT_REG Number of bytes transferred by CIU unit to card. (RO)

Register 17.23. SDHOST_TBBCNT_REG (0x0060)



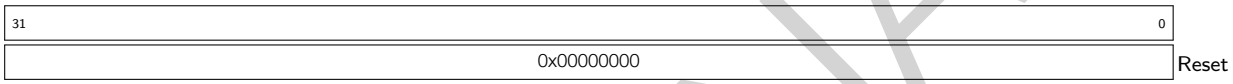
SDHOST_TBBCNT_REG Number of bytes transferred between Host/DMA memory and BIU FIFO. (RO)

Register 17.24. SDHOST_DEBNCE_REG (0x0064)



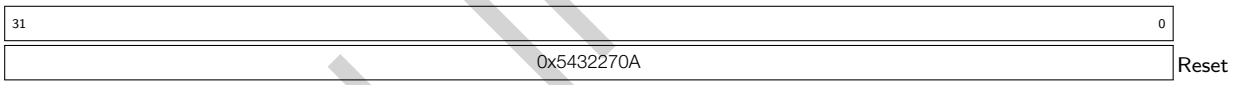
SDHOST_DEBOUNCE_COUNT Number of host clocks (clk) used by debounce filter logic. The typical debounce time is 5 ~ 25 ms to prevent the card instability when the card is inserted or removed. (R/W)

Register 17.25. SDHOST_USRID_REG (0x0068)



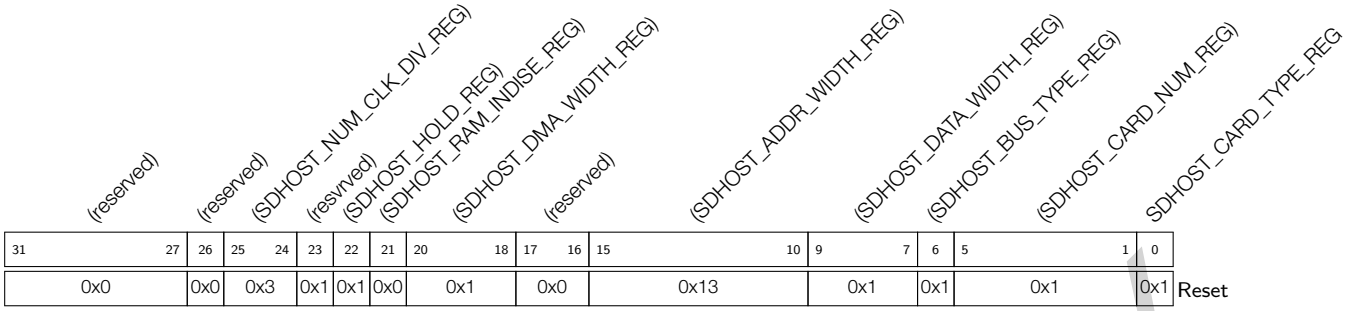
SDHOST_USRID_REG User identification register, value set by user. Can also be used as a scratch-pad register by user. (R/W)

Register 17.26. SDHOST_VERID_REG (0x006C)



SDHOST_VERSIONID_REG Hardware version register. Can also be read by fireware. (RO)

Register 17.27. SDHOST_HCON_REG (0x0070)



SDHOST_NUM_CLK_DIV_REG Have 4 clk divider in design . (RO)

SDHOST_HOLD_REG Have a hold regiser in data path . (RO)

SDHOST_RAM_INDISE_REG Inside RAM in SDMMC module. (RO)

SDHOST_DMA_WIDTH_REG DMA data witdth is 32. (RO)

SDHOST_ADDR_WIDTH_REG Register address width is 32. (RO)

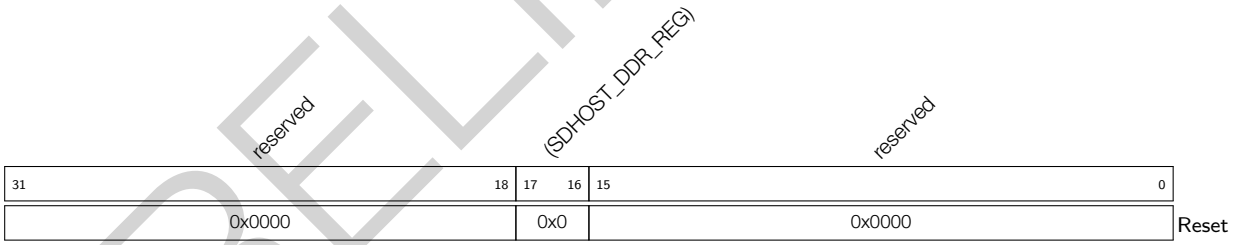
SDHOST_DATA_WIDTH_REG Regisger data widht is 32. (RO)

SDHOST_BUS_TYPE_REG Register config is APB bus. (RO)

SDHOST_CARD_NUM_REG Support card number is 2. (RO)

SDHOST_CARD_TYPE_REG Hardware support SDIO and MMC. (RO)

Register 17.28. SDHOST_UHS_REG (0x0074)

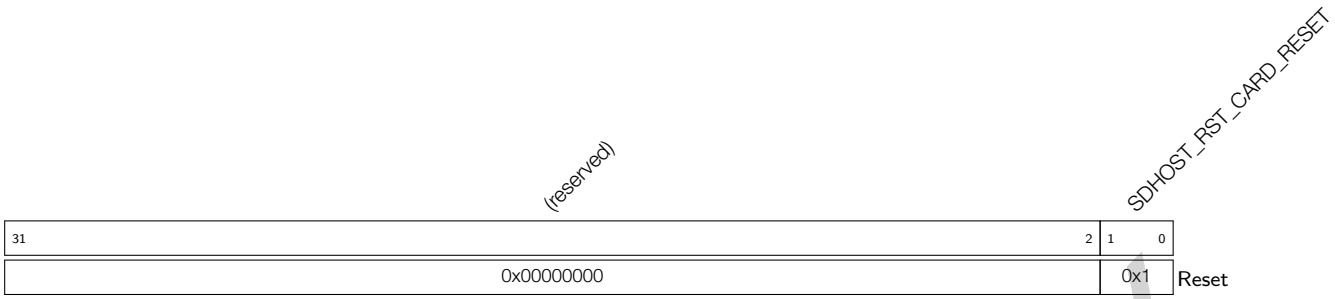


SDHOST_DDR_REG DDR mode selecton,1 bit for each card. (R/W)

0-Non-DDR mdoe.

1-DDR mdoe.

Register 17.29. SDHOST_RST_N_REG (0x0078)



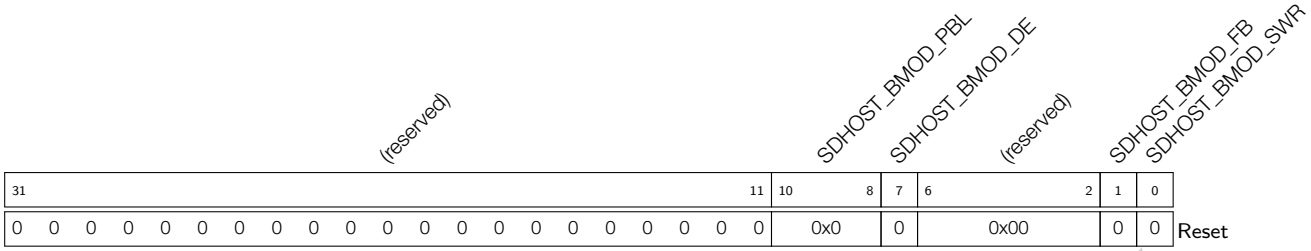
SDHOST_RST_CARD_RESET Hardware reset.

1: Active mode;

0: Reset.

These bits cause the cards to enter pre-idle state, which requires them to be re-initialized. SDHOST_RST_CARD_RESET[0] should be set to 1'b0 to reset card0, SDHOST_RST_CARD_RESET[1] should be set to 1'b0 to reset card1. (R/W)

Register 17.30. SDHOST_BMOD_REG (0x0080)



SDHOST_BMOD_PBL Programmable Burst Length. These bits indicate the maximum number of beats to be performed in one IDMAC Internal DMA Control transaction. The IDMAC will always attempt to burst as specified in PBL each time it starts a burst transfer on the host bus. The permissible values are 1, 4, 8, 16, 32, 64, 128 and 256. This value is the mirror of MSIZE of FIFOTH register. In order to change this value, write the required value to FIFOTH register. This is an encode value as follows: (RO)

000: 1-byte transfer;

001: 4-byte transfer;

010: 8-byte transfer;

011: 16-byte transfer;

100: 32-byte transfer;

101: 64-byte transfer;

110: 128-byte transfer;

111: 256-byte transfer.

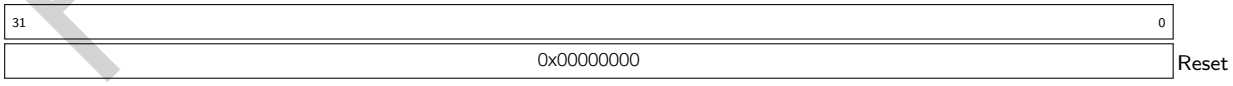
PBL is a read-only value and is applicable only for data access, it does not apply to descriptor access.

SDHOST_BMOD_DE IDMAC Enable. When set, the IDMAC is enabled. (RO)

SDHOST_BMOD_FB Fixed Burst. Controls whether the AHB Master interface performs fixed burst transfers or not. When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB will use SINGLE and INCR burst transfer operations. (R/W)

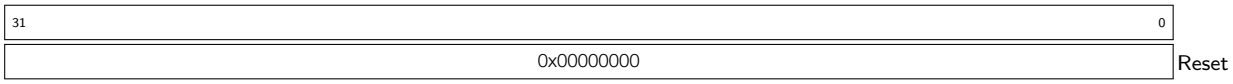
SDHOST_BMOD_SWR Software Reset. When set, the DMA Controller resets all its internal registers. It is automatically cleared after one clock cycle. (R/W)

Register 17.31. SDHOST_PLDMND_REG (0x0080)



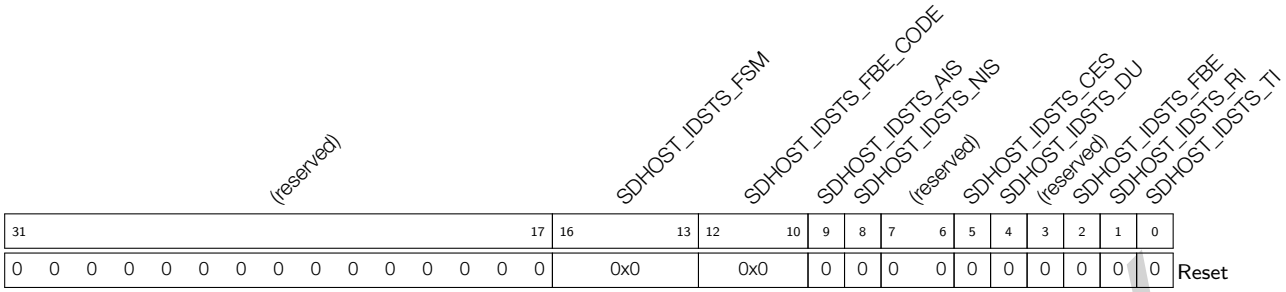
SDHOST_PLDMND_REG Poll Demand. If the OWNER bit of a descriptor is not set, the FSM goes to the Suspend state. The host needs to write any value into this register for the IDMAC FSM to resume normal descriptor fetch operation. This is a write only . (WO)

Register 17.32. SDHOST_DBADDR_REG (0x0088)



SDHOST_DBADDR_REG Start of Descriptor List. Contains the base address of the First Descriptor. The LSB bits [1:0] are ignored and taken as all-zero by the IDMAC internally. Hence these LSB bits may be treated as read-only. (R/W)

Register 17.33. SDHOST_IDSTS_REG (0x008C)



SDHOST_IDSTS_FSM DMAC FSM present state. (RO)

0: DMA_IDLE (idle state);
1: DMA_SUSPEND (suspend state);
2: DESC_RD (descriptor reading state);
3: DESC_CHK (descriptor checking state);
4: DMA_RD_REQ_WAIT (read-data request waiting state);
5: DMA_WR_REQ_WAIT (write-data request waiting state);
6: DMA_RD (data-read state);
7: DMA_WR (data-write state);
8: DESC_CLOSE (descriptor close state).

SDHOST_IDSTS_FBE_CODE Fatal Bus Error Code. Indicates the type of error that caused a Bus Error. Valid only when the Fatal Bus Error bit IDSTS[2] is set. This field does not generate an interrupt. (RO)

001: Host Abort received during transmission;
010: Host Abort received during reception;
Others: Reserved.

SDHOST_IDSTS_AIS Abnormal Interrupt Summary. Logical OR of the following: IDSTS[2] : Fatal Bus Interrupt, IDSTS[4] : DU bit Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes AIS to be set is cleared. Writing 1 clears this bit. (R/W)

SDHOST_IDSTS_NIS Normal Interrupt Summary. Logical OR of the following: IDSTS[0] : Transmit Interrupt, IDSTS[1] : Receive Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes NIS to be set is cleared. Writing 1 clears this bit. (R/W)

Continued on the next page...

Register 17.33. SDHOST_IDSTS_REG (0x008C)

Continued from the previous page...

SDHOST_IDSTS_CES Card Error Summary. Indicates the status of the transaction to/from the card, also present in RINTSTS. Indicates the logical OR of the following bits: (R/W)
EBE : End Bit Error;
RTO : Response Timeout/Boot Ack Timeout;
RCRC : Response CRC;
SBE : Start Bit Error;
DRTO : Data Read Timeout/BDS timeout;
DCRC : Data CRC for Receive;
RE : Response Error.
Writing 1 clears this bit. The abort condition of the IDMAC depends on the setting of this CES bit. If the CES bit is enabled, then the IDMAC aborts on a response error.

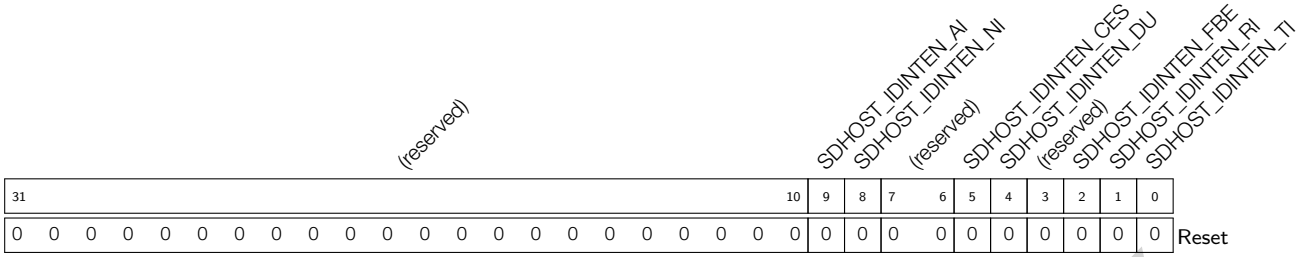
SDHOST_IDSTS_DU Descriptor Unavailable Interrupt. This bit is set when the descriptor is unavailable due to OWNER bit = 0 (DES0[31] = 0). Writing 1 clears this bit. (R/W)

SDHOST_IDSTS_FBE Fatal Bus Error Interrupt. Indicates that a Bus Error occurred (IDSTS[12:10]) . When this bit is set, the DMA disables all its bus accesses. Writing 1 clears this bit. (R/W)

SDHOST_IDSTS_RI Receive Interrupt. Indicates the completion of data reception for a descriptor. Writing 1 clears this bit. (R/W)

SDHOST_IDSTS_TI Transmit Interrupt. Indicates that data transmission is finished for a descriptor. Writing 1 clears this bit. (R/W)

Register 17.34. SDHOST_IDINTEN_REG (0x0090)



SDHOST_IDINTEN_AI Abnormal Interrupt Summary Enable. When set, an abnormal interrupt is enabled. This bit enables the following bits:
IDINTEN[2]: Fatal Bus Error Interrupt; (R/W)
IDINTEN[4]: DU Interrupt.

SDHOST_IDINTEN_NI Normal Interrupt Summary Enable. When set, a normal interrupt is enabled. When reset, a normal interrupt is disabled. This bit enables the following bits: (R/W)
IDINTEN[0]: Transmit Interrupt;
IDINTEN[1]: Receive Interrupt.

SDHOST_IDINTEN_CES Card Error summary Interrupt Enable. When set, it enables the Card Interrupt summary. (R/W)

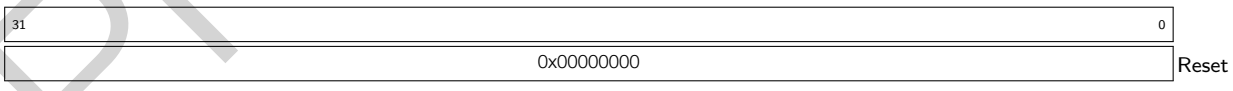
SDHOST_IDINTEN_DU Descriptor Unavailable Interrupt. When set along with Abnormal Interrupt Summary Enable, the DU interrupt is enabled. (R/W)

SDHOST_IDINTEN_FBE Fatal Bus Error Enable. When set with Abnormal Interrupt Summary Enable, the Fatal Bus Error Interrupt is enabled. When reset, Fatal Bus Error Enable Interrupt is disabled. (R/W)

SDHOST_IDINTEN_RI Receive Interrupt Enable. When set with Normal Interrupt Summary Enable, Receive Interrupt is enabled. When reset, Receive Interrupt is disabled. (R/W)

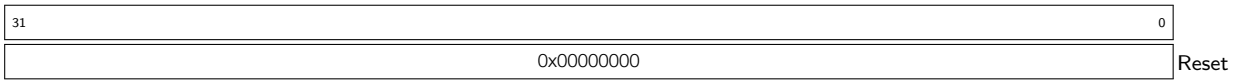
SDHOST_IDINTEN_TI Transmit Interrupt Enable. When set with Normal Interrupt Summary Enable, Transmit Interrupt is enabled. When reset, Transmit Interrupt is disabled. (R/W)

Register 17.35. SDHOST_DSCADDR_REG (0x0094)



SDHOST_DSCADDR_REG Host Descriptor Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the start address of the current descriptor read by the IDMAC. (RO)

Register 17.36. SDHOST_BUFADDR_REG (0x0098)



SDHOST_BUFADDR_REG Host Buffer Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the current Data Buffer Address being accessed by the IDMAC. (RO)

Register 17.37. SDHOST_CARDTHRCTL_REG (0x0100)

SDHOST_CARDTHRESHOLD_REG The inside FIFO size is 512,This register is applicable when SDHOST_CARDERTHREN_REG is set to 1 or SDHOST_CARDRDTHREN_REG set to 1. (R/W)

SDHOST_CARDWRTHREN_REG Applicable when HS400 mode is enabled. (R/W)

1'b0-Card write Threshold disabled.

1'b1-Card write Threshold enabled.

SDHOST_CARDCLRINTEN_REG Busy clear interrupt generation: (R/W)

1'b0-Busy clear interrypt disabled.

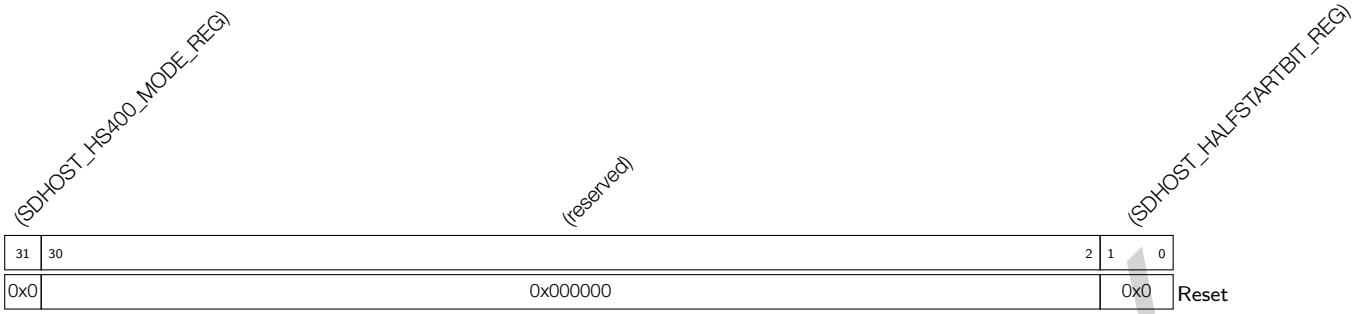
1'b1-Busy clear interrypt enabled.

SDHOST_CARDRDTHREN_REG Card read threshold enable. (R/W)

1'b0-Card read threshold disabled.

1'b1-Card read threshold enabled.

Register 17.38. SDHOST_EMMC_DDR_REG (0x010C)



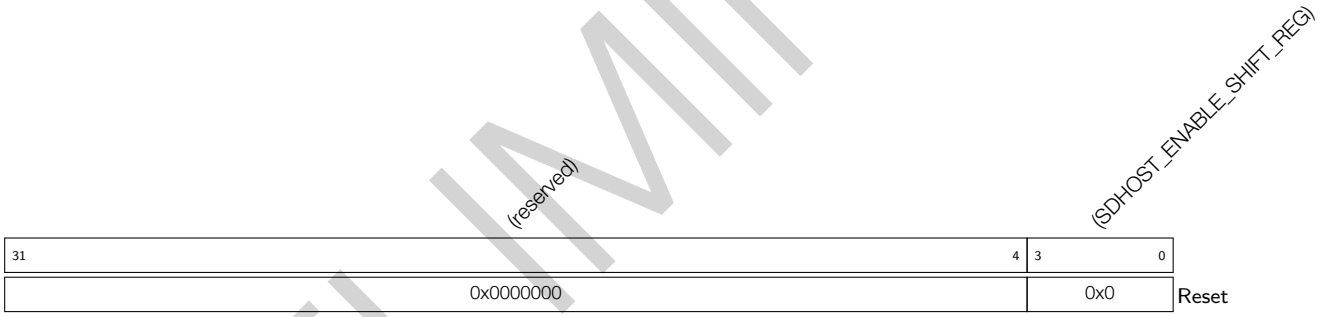
SDHOST_HS400_MODE_REG Set 1 to enable HS400 mode. (R/W)

SDHOST_HALFSTARTBIT_REG Control for start bit detection mechanism duration of start bit.Each bit refers to one slot.Set this bit to 1 for eMMC4.5 and above,set to 0 for SD applications.For eMMC4.5,start bit can be: (R/W)

1'b0-Full cycle.

1'b1-less than one full cycle.

Register 17.39. SDHOST_ENSHIFT_REG (0x0110)



DHOST_ENABLE_SHIFT_REG Control for the amount of phase shift provided on the default enables in the design.Two bits assigned for each card. (R/W)

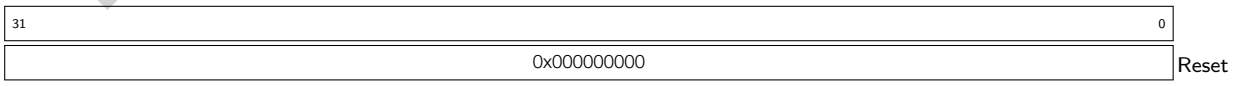
2'b00-Default phase shift.

2'b01-Enables shifted to next immediate positive edge.

2'b10-Enables shifted to next immediate negative edge.

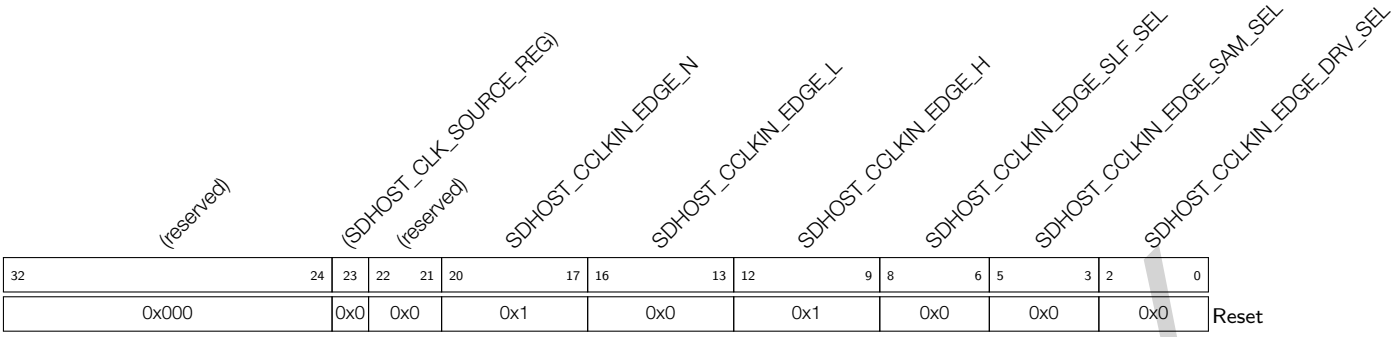
2'b11-Reserved.

Register 17.40. SDHOST_BUFFIFO_REG (0x0200)



SDHOST_BUFFIFO_REG CPU write and read transmit data by FIFO. This register points to the current Data FIFO . (RO)

Register 17.41. SDHOST_CLK_DIV_EDGE_REG (0x0800)



SDHOST_CLK_SOURCE_REG Set to 1 to use 160M PLL clock ,Set to 0 to use 40M XLTAL clock. (R/W)

CCLKIN_EDGE_N This value should be equal to CCLKIN_EDGE_L. (R/W)

CCLKIN_EDGE_L The low level of the divider clock. The value should be larger than CCLKIN_EDGE_H. (R/W)

CCLKIN_EDGE_H The high level of the divider clock. The value should be smaller than CCLKIN_EDGE_L. (R/W)

CCLKIN_EDGE_SLF_SEL It is used to select the clock phase of the internal signal from phase90, phase180, or phase270. (R/W)

CCLKIN_EDGE_SAM_SEL It is used to select the clock phase of the input signal from phase90, phase180, or phase270. (R/W)

CCLKIN_EDGE_DRV_SEL It is used to select the clock phase of the output signal from phase90, phase180, or phase270. (R/W)

Note: SD/MMC use this register to divide the 160M clock(CCLKIN_EDGE_H/CCLKIN_EDGE_L). The output clock connect to sdio slave divider by this register and SDHOST_CLKDIV_REG,there are 4 clock source to seleced by SDHOST_CLKSRC_REG register.

18 LED PWM Controller (LEDC)

18.1 Overview

The LED PWM Controller is a peripheral designed to generate PWM signals for LED control. It has specialized features such as automatic duty cycle fading. However, the LED PWM Controller can also be used to generate PWM signals for other purposes.

18.2 Features

The LED PWM Controller has the following features:

- Eight independent PWM generators (i.e. eight channels)
- Four independent timers that support division by fractions
- Automatic duty cycle fading (i.e. gradual increase/decrease of a PWM's duty cycle without interference from the processors) with interrupt generation on fade completion
- Adjustable phase of PWM signal output
- PWM signal output in low-power mode (Light-sleep mode)
- Maximum PWM resolution: 14 bits

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the eight PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 7).

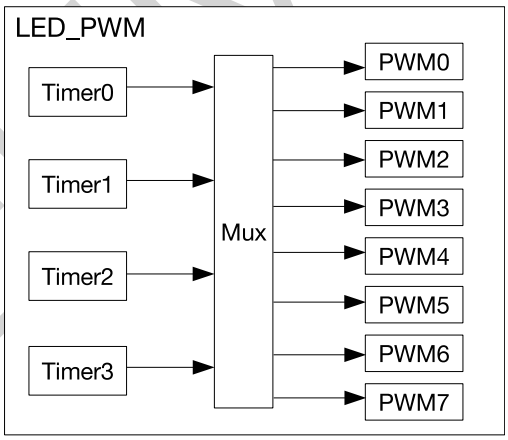


Figure 18-1. LED PWM Architecture

18.3 Functional Description

18.3.1 Architecture

Figure 18-1 shows the architecture of the LED PWM Controller.

The four timers can be independently configured (i.e. clock divider, and counter overflow value) and each internally maintains a timebase counter (i.e. a counter that counts on cycles of a reference clock). Each PWM

generator will select one of the timers and uses the timer's counter value as a reference to generate its PWM signal.

Figure 18-2 illustrates the main functional blocks of the timer and the PWM generator.

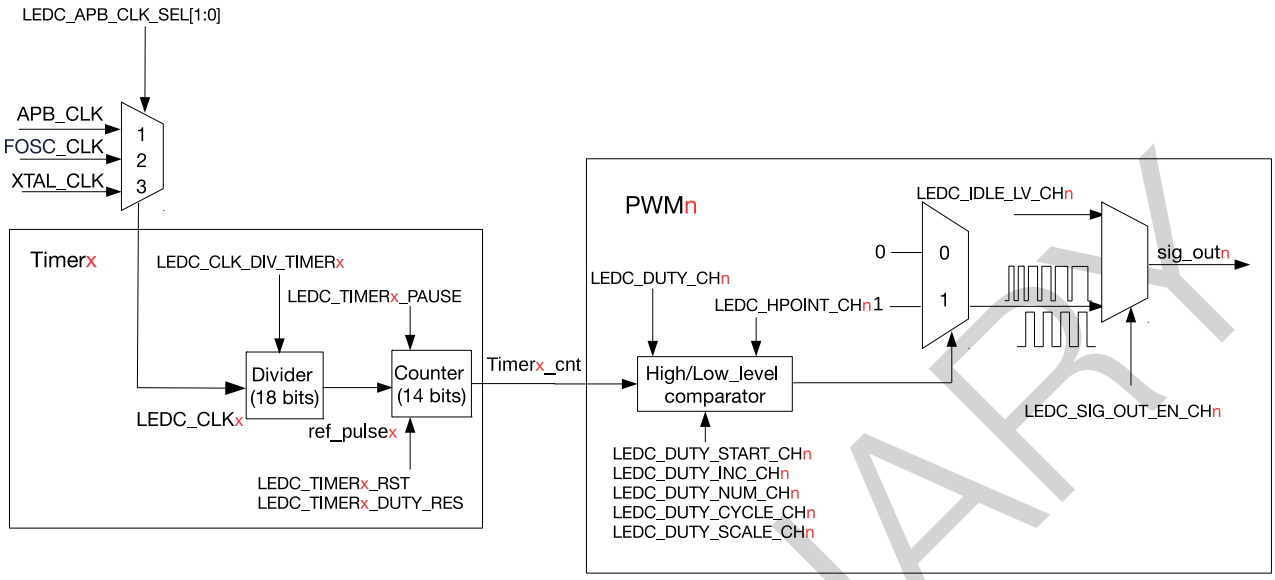


Figure 18-2. LED PWM Generator Diagram

18.3.2 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 18-2, this clock signal used by the timebase counter is named ref_pulse*x*. All timers use the same clock source LEDC_CLK*x*, which is then passed through a clock divider to generate ref_pulse*x* for the counter.

18.3.2.1 Clock Source

Software configuring registers for LED PWM is clocked by APB_CLK. For more information about APB_CLK, see Chapter 3 *Reset and Clock*. To use the LED PWM pheripheral, the APB_CLK signal to the LED PWM has to be enabled. The APB_CLK signal to LED PWM can be enabled by setting the SYSTEM_LEDC_CLK_EN field in the register SYSTEM_PERIP_CLK_EN0_REG and be reset via software by setting the SYSTEM_LEDC_RST field in the register SYSTEM_PERIP_RST_EN0_REG. For more information, please refer to Table 21 in Chapter 11 *System Registers (SYSREG) [to be added later]*.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: APB_CLK, FOSC_CLK and XTAL_CLK (see Chapter 3 *Reset and Clock* for more details about each clock signal). The procedure for selecting a clock source signal for LEDC_CLK*x* is described below:

- APB_CLK: Set LEDC_APB_CLK_SEL[1:0] to 1
- FOSC_CLK: Set LEDC_APB_CLK_SEL[1:0] to 2
- XTAL_CLK: Set LEDC_APB_CLK_SEL[1:0] to 3

The LEDC_CLK*x* signal will then be passed through the clock divider.

18.3.2.2 Clock Divider Configuration

The LEDC_CLKx signal is passed through a clock divider to generate the ref_pulsex signal for the counter. The frequency of ref_pulsex is equal to the frequency of LEDC_CLKx divided by the LEDC_CLK_DIV_TIMERx divider value (see Figure 18-2).

The LEDC_CLK_DIV_TIMERx divider value is a fractional clock divider. Thus, it supports non-integer divider values. LEDC_CLK_DIV_TIMERx is configured via the LEDC_CLK_DIV_TIMERx field according to the following equation.

$$LEDC\_CLK\_DIV\_TIMERx = A + \frac{B}{256}$$

- A corresponds to the most significant 10 bits of LEDC_CLK_DIV_TIMERx (i.e. LEDC_TIMERx_CONF_REG[21:12])
- The fractional part B corresponds to the least significant 8 bits of LEDC_CLK_DIV_TIMERx (i.e. LEDC_TIMERx_CONF_REG[11:4])

When the fractional part B is zero, LEDC_CLK_DIV_TIMERx is equivalent to an integer divider value (i.e. an integer prescaler). In other words, a ref_pulsex clock pulse is generated after every A number of LEDC_CLKx clock pulses.

However, when B is nonzero, LEDC_CLK_DIV_TIMERx becomes a non-integer divider value. The clock divider implements non-integer frequency division by alternating between A and (A+1) LEDC_CLKx clock pulses per ref_pulsex clock pulse. This will result in the average frequency of ref_pulsex clock pulse being the desired frequency (i.e. the non-integer divided frequency). For every 256 ref_pulsex clock pulses:

- A number of B ref_pulsex clock pulses will consist of (A+1) LEDC_CLKx clock pulses
- A number of (256-B) ref_pulsex clock pulses will consist of A LEDC_CLKx clock pulses
- The ref_pulsex clock pulses consisting of (A+1) pulses are evenly distributed amongst those consisting of A pulses

Figure 18-3 illustrates the relation between LEDC_CLKx clock pulses and ref_pulsex clock pulses when dividing by a non-integer LEDC_CLK_DIV_TIMERx.

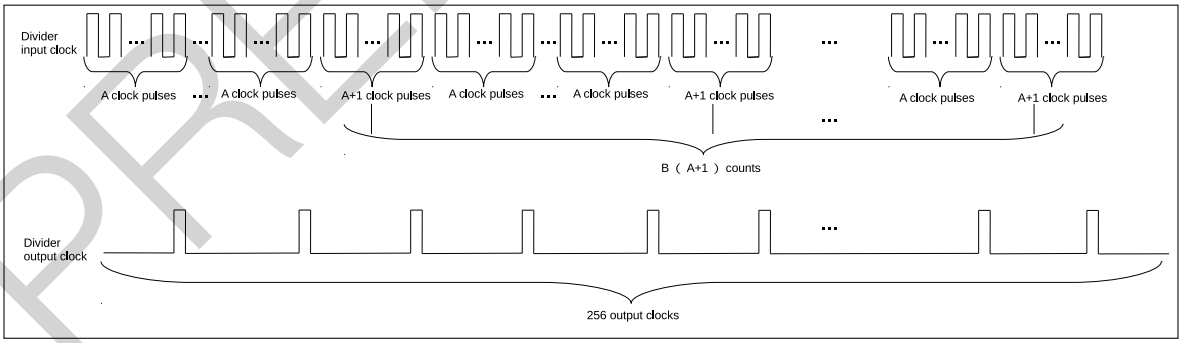


Figure 18-3. Frequency Division When LEDC_CLK_DIV_TIMERx is a Non-Integer Value

To change the timer's clock divider value at runtime, first set the LEDC_CLK_DIV_TIMERx field, and then set the LEDC_TIMERx_PARA_UP field to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_TIMERx_PARA_UP field will be automatically cleared by hardware.

18.3.2.3 14-bit Counter

Each timer contains a 14-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 18-2). The LEDC_TIMER*x*_DUTY_RES field configures the overflow value of this 14-bit counter. Hence, the maximum resolution of the PWM signal is 14 bits. The counter counts up to $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$, overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software.

The counter can trigger LEDC_TIMER*x*_OVF_INT interrupt (generated automatically by hardware without configuration) every time the counter overflows. It can also be configured to trigger LEDC_OVF_CNT_CH*n*_INT interrupt after the counter overflows $LEDC\_OVF\_NUM\_CHn + 1$ times. To configure LEDC_OVF_CNT_CH*n*_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CH*n* as the counter for the PWM generator
2. Enable the counter by setting LEDC_OVF_CNT_EN_CH*n*
3. Set LEDC_OVF_NUM_CH*n* to the number of counter overflows to generate an interrupt, minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CH*n*_INT_ENA
5. Set LEDC_TIMER*x*_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

Referring to Figure 18-2, the frequency of a PWM generator output signal (sig_out*n*) is dependent on the frequency of the timer's clock source (LEDC_CLK*x*), the clock divider value (LEDC_CLK_DIV_TIMER*x*), and the range of the counter (LEDC_TIMER*x*_DUTY_RES):

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV}x \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_TIMER*x*_PARA_UP should also be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP. LEDC_TIMER*x*_PARA_UP field will be automatically cleared by hardware.

18.3.3 PWM Generators

To generate a PWM signal, a PWM generator (PWM*n*) selects a timer (Timer*x*). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CH*n* to use one of four timers to generate the PWM output.

As shown in Figure 18-2, each PWM generator has a comparator and two multiplexers. A PWM generator compares the timer's 14-bit counter value (Timer*x*_cnt) to two trigger values Hpoint*n* and Lpoint*n*. When the timer's counter value is equal to Hpoint*n* or Lpoint*n*, the PWM signal is high or low, respectively, as described below:

- If Timer*x*_cnt == Hpoint*n*, sig_out*n* is 1.
- If Timer*x*_cnt == Lpoint*n*, sig_out*n* is 0.

Figure 18-4 illustrates how Hpoint*n* or Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

For a particular PWM generator (PWM*n*), its Hpoint*n* is sampled from the LEDC_HPOINT_CH*n* field each time the selected timer's counter overflows. Likewise, Lpoint*n* is also sampled on every counter overflow and is calculated from the sum of the LEDC_DUTY_CH*n*[18:4] and LEDC_HPOINT_CH*n* fields. By setting Hpoint*n* and Lpoint*n* via

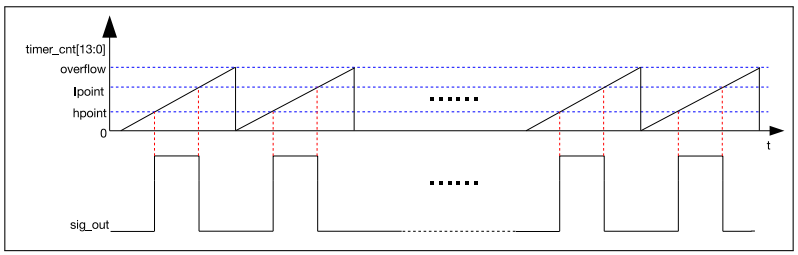


Figure 18-4. LED_PWM Output Signal Diagram

the LEDC_HPOINT_CH*n* and LEDC_DUTY_CH*n*[18:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig_out*n*) is enabled by setting LEDC_SIG_OUT_EN_CH*n*. When LEDC_SIG_OUT_EN_CH*n* is cleared, PWM signal output is disabled, and the output signal (sig_out*n*) will output a constant level as specified by LEDC_IDLE_LV_CH*n*.

The bits LEDC_DUTY_CH*n*[3:0] are used to dither the duty cycles of the PWM output signal (sig_out*n*) by periodically altering the duty cycle of sig_out*n*. When LEDC_DUTY_CH*n*[3:0] is set to a non-zero value, then for every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC_DUTY_CH*n*[3:0]) cycles. For instance, if LEDC_DUTY_CH*n*[18:4] is set to 10 and LEDC_DUTY_CH*n*[3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC_TIMER_SEL_CH*n*, LEDC_HPOINT_CH*n*, LEDC_DUTY_CH*n*[18:4] and LEDC_SIG_OUT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

18.3.4 Duty Cycle Fading

The PWM generators can fade the duty cycle of a PWM output signal (i.e. gradually change the duty cycle from one value to another). If Duty Cycle Fading is enabled, the value of Lpoint*n* will be incremented/decremented after a fixed number of counter overflows occurs. Figure 18-5 illustrates Duty Cycle Fading.

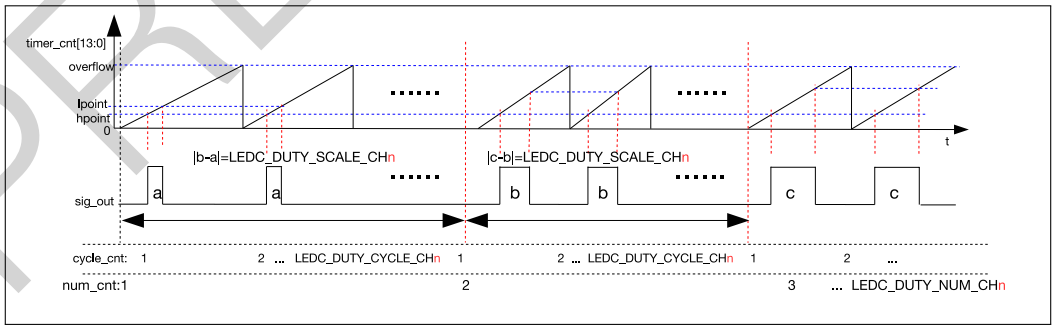


Figure 18-5. Output Signal Diagram of Fading Duty Cycle

Duty Cycle Fading is configured using the following register fields:

- LEDC_DUTY_CH*n* is used to set the initial value of Lpoint*n*
- LEDC_DUTY_START_CH*n* will enable/disable duty cycle fading when set/cleared

- LEDC_DUTY_CYCLE_CH*n* sets the number of counter overflow cycles for every Lpoint*n* increment/decrement. In other words, Lpoint*n* will be incremented/decremented after LEDC_DUTY_CYCLE_CH*n* counter overflows.
- LEDC_DUTY_INC_CH*n* configures whether Lpoint*n* is incremented/decremented if set/cleared
- LEDC_DUTY_SCALE_CH*n* sets the amount that Lpoint*n* is incremented/decremented
- LEDC_DUTY_NUM_CH*n* sets the maximum number of increments/decrements before duty cycle fading stops.

If the fields LEDC_DUTY_CH*n*, LEDC_DUTY_START_CH*n*, LEDC_DUTY_CYCLE_CH*n*, LEDC_DUTY_INC_CH*n*, LEDC_DUTY_SCALE_CH*n*, and LEDC_DUTY_NUM_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. After this field is set, the values for duty cycle fading will take effect at once. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

18.3.5 Interrupts

- LEDC_OVF_CNT_CH*n*_INT: Triggered when the timer counter overflows for (LEDC_OVF_NUM_CH*n* + 1) times and the register LEDC_OVF_CNT_EN_CH*n* is set to 1.
- LEDC_DUTY_CHNG_END_CH*n*_INT: Triggered when a fade on an LED PWM generator has finished.
- LEDC_TIMER*x*_OVF_INT: Triggered when an LED PWM timer has reached its maximum counter value.

18.4 Register Summary

The addresses in this section are relative to LED PWM Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

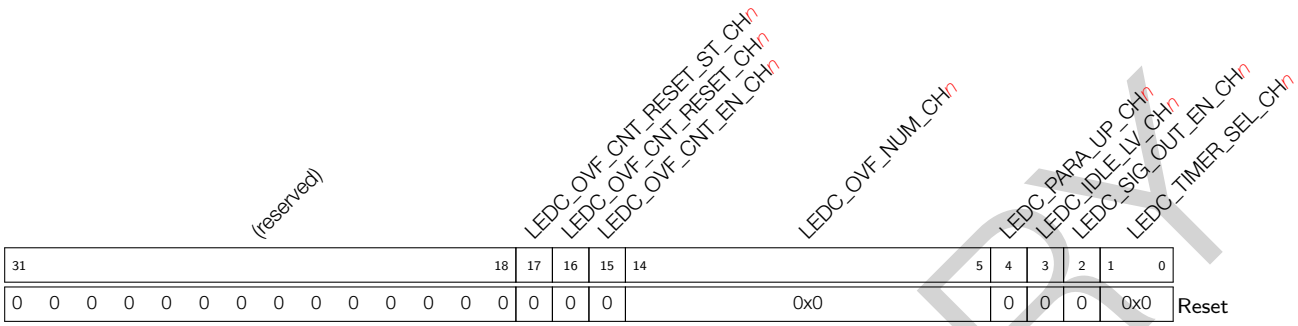
Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	R/W
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_CONF1_REG	Configuration 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

Name	Description	Address	Access
LEDC_CH5_DUTY_R_REG	Current duty cycle for channel 5	0x0074	RO
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

18.5 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 18.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**



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 (when LEDC_SIG_OUT_EN_CH*n* is 0). (R/W)

LEDC_PARA_UP_CH*n* This bit is used to update the listed fields below for channel *n*, and will be automatically cleared by hardware. (WO)

- LEDC_HPOINT_CH*n*
- LEDC_DUTY_START_CH*n*
- LEDC_SIG_OUT_EN_CH*n*
- LEDC_TIMER_SEL_CH*n*
- LEDC_DUTY_NUM_CH*n*
- LEDC_DUTY_CYCLE_CH*n*
- LEDC_DUTY_SCALE_CH*n*
- LEDC_DUTY_INC_CH*n*
- LEDC_OVF_CNT_EN_CH*n*

Continued on the next page...

Register 18.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14n*)**

Continued from the previous page...

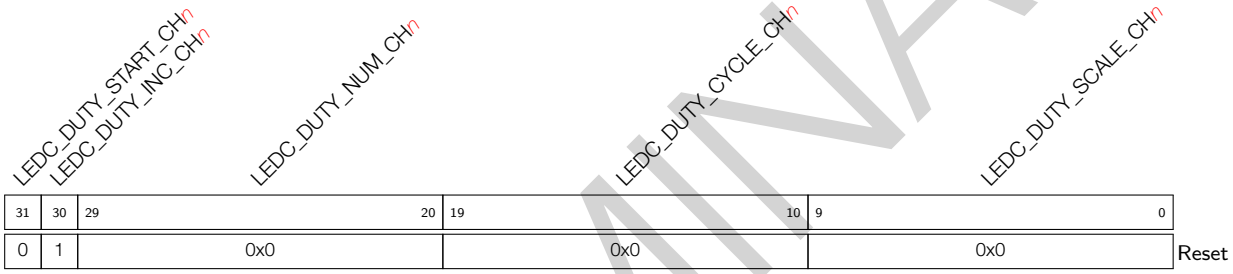
LEDC_OVF_NUM_CH*n* This register is used to configure the maximum times of overflow minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* This bit is used to count the number of times when the timer selected by channel *n* overflows.(R/W)

LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the timer-overflow counter of channel *n*. (WO)

LEDC_OVF_CNT_RESET_ST_CH*n* This is the status bit of LEDC_OVF_CNT_RESET_CH*n*. (RO)

Register 18.2. LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+0x14n*)**



LEDC_DUTY_SCALE_CH*n* This register is used to configure the changing step scale of duty on channel *n*. (R/W)

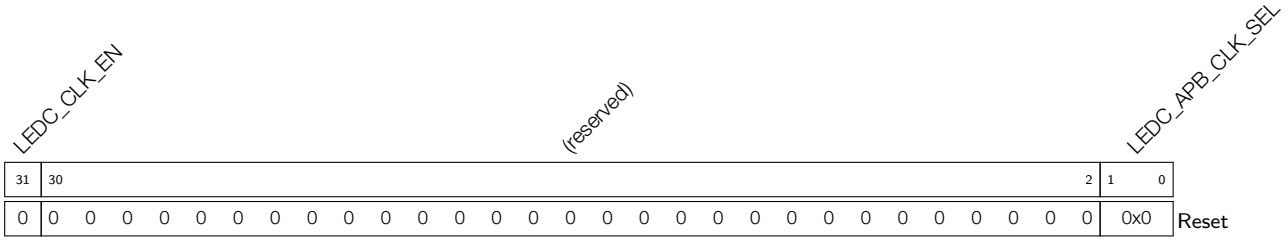
LEDC_DUTY_CYCLE_CH*n* The duty will change every LEDC_DUTY_CYCLE_CH*n* on channel *n*. (R/W)

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 upon the next timer overflow when this bit is set to 1. (R/W)

Register 18.3. LEDC_CONF_REG (0x00D0)



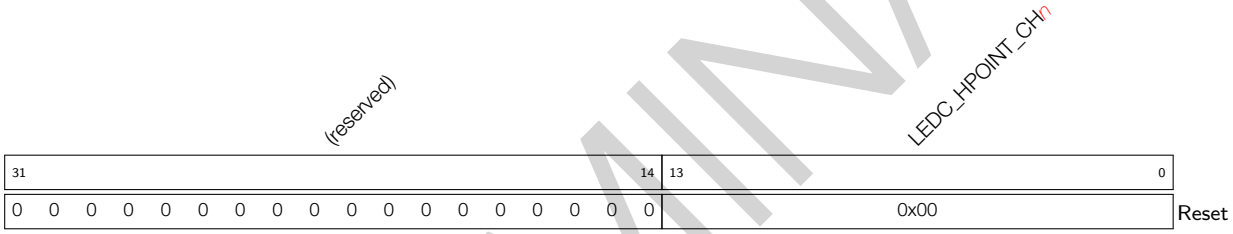
LEDC_APB_CLK_SEL This field is used to select the common clock source for all the 4 timers.

1: APB_CLK; 2: FOSC_CLK; 3: XTAL_CLK. (R/W)

LEDC_CLK_EN This bit is used to control clock.

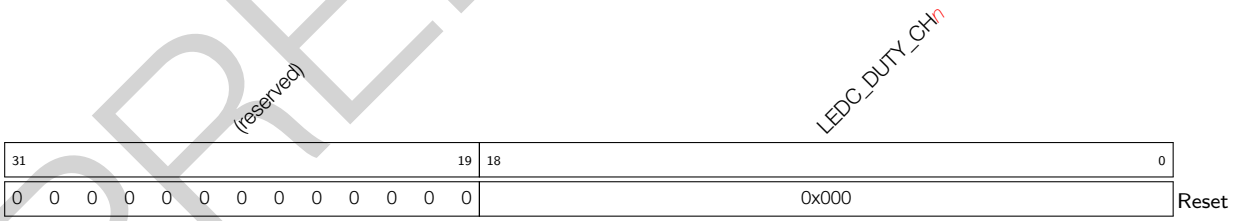
1: Force clock on for register. 0: Support clock only when application writes registers. (R/W)

Register 18.4. LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+0x14n*)**



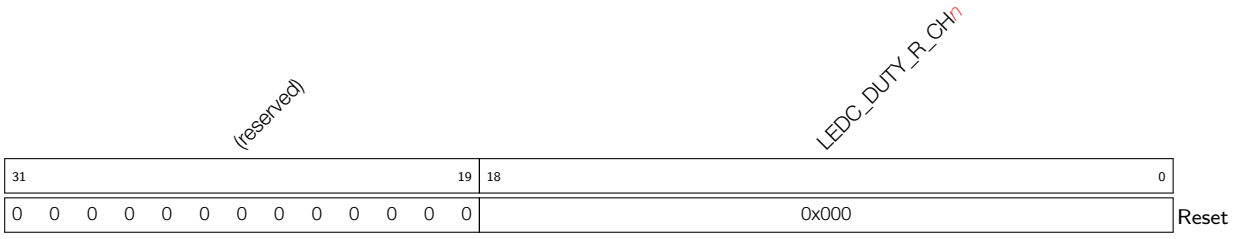
LEDC_HPOINT_CH*n* The output value changes to high when the selected timers has reached the value specified by this register. (R/W)

Register 18.5. LEDC_CH*n*_DUTY_REG (*n*: 0-7) (0x0008+0x14n*)**



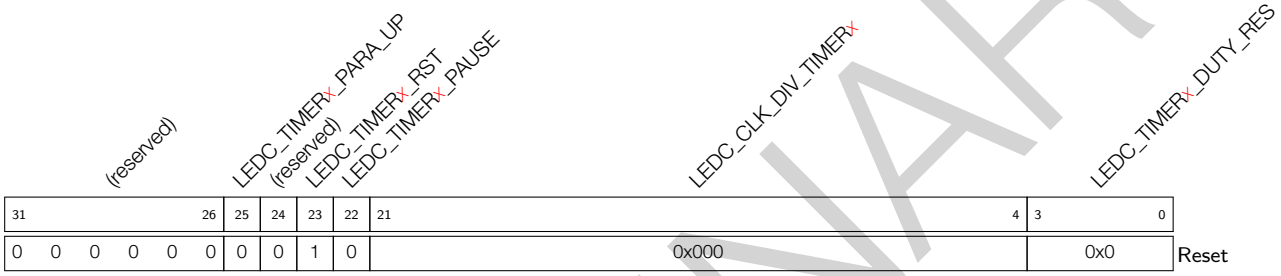
LEDC_DUTY_CH*n* This register is used to change the output duty by controlling the Lpoint. The output value turns to low when the selected timers has reached the Lpoint. (R/W)

Register 18.6. LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+0x14n*)**



LEDC_DUTY_R_CH*n* This register stores the current duty of output signal on channel *n*. (RO)

Register 18.7. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8x*)**



LEDC_TIMER*x*_DUTY_RES This register is used to control the range of the counter in timer *x*. (R/W)

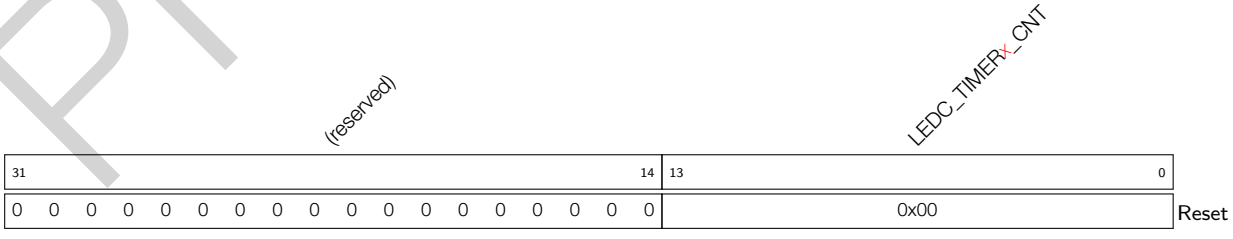
LEDC_CLK_DIV_TIMER*x* This register is used to configure the divisor for the divider in timer *x*. The least significant eight bits represent the fractional part. (R/W)

LEDC_TIMER*x*_PAUSE This bit is used to suspend the counter in timer *x*. (R/W)

LEDC_TIMER*x*_RST This bit is used to reset timer *x*. The counter will show 0 after reset. (R/W)

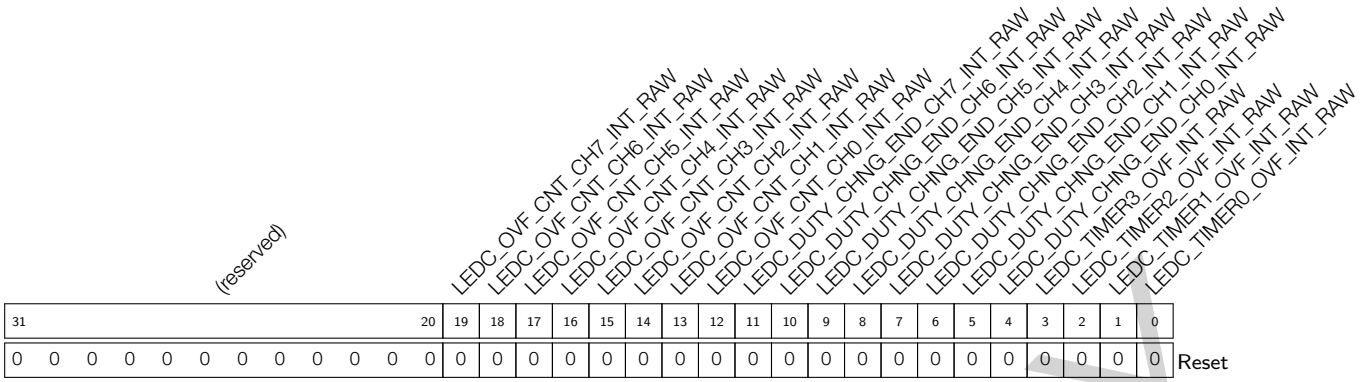
LEDC_TIMER*x*_PARA_UP Set this bit to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES. (WO)

Register 18.8. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8x*)**



LEDC_TIMER*x*_CNT This register stores the current counter value of timer *x*. (RO)

Register 18.9. LEDC_INT_RAW_REG (0x00C0)

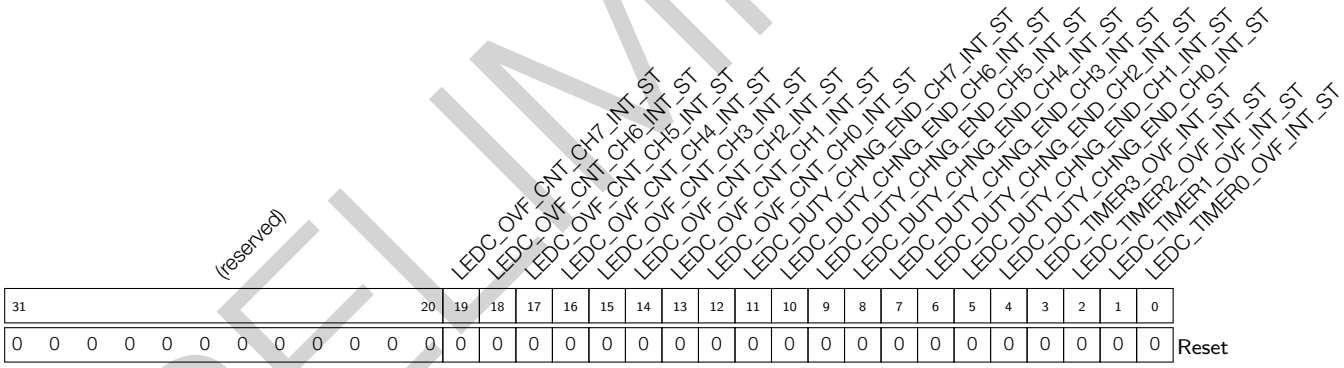


LEDC_TIMER*x*_OVF_INT_RAW Triggered when the timer*x* has reached its maximum counter value. (RO)

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 18.10. LEDC_INT_ST_REG (0x00C4)

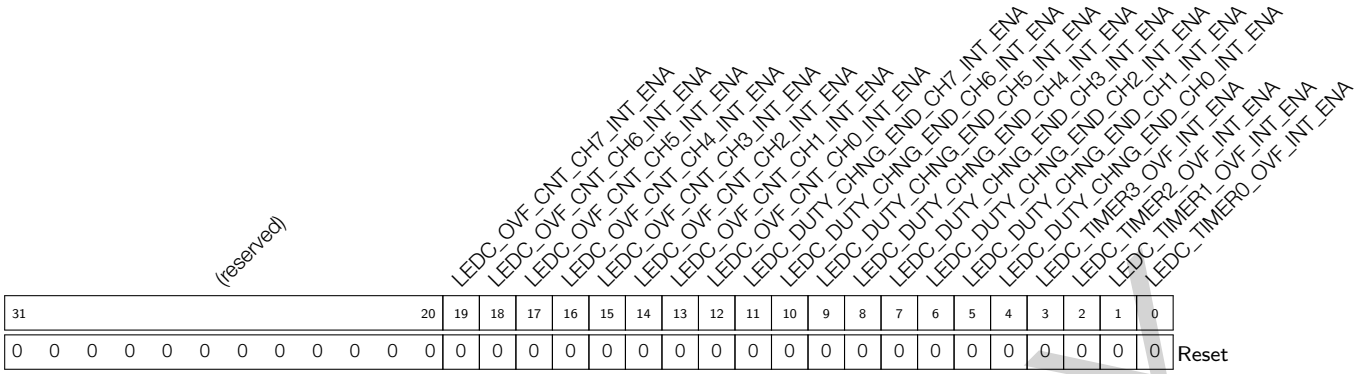


LEDC_TIMER*x*_OVF_INT_ST This is the masked interrupt status bit for the LEDC_TIMER*x*_OVF_INT interrupt when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

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 18.11. LEDC_INT_ENA_REG (0x00C8)

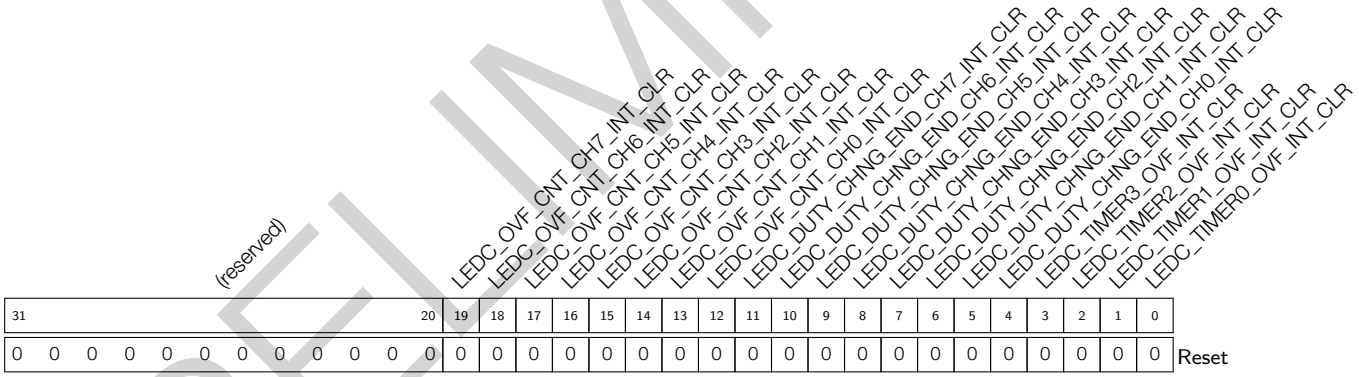


LEDC_TIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_TIMER*x*_OVF_INT interrupt. (R/W)

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 18.12. LEDC_INT_CLR_REG (0x00CC)

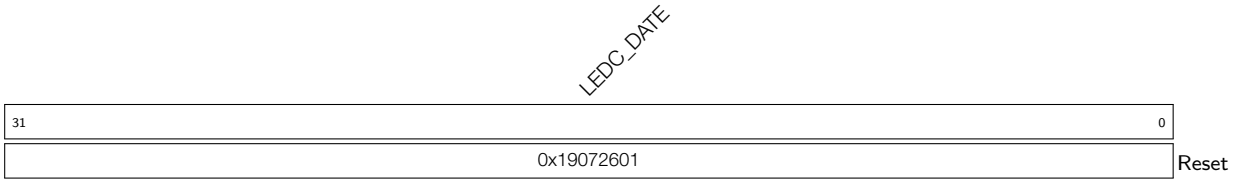


LEDC_TIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_TIMER*x*_OVF_INT interrupt. (WO)

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 18.13. LEDC_DATE_REG (0x00FC)



LEDC_DATE This is the version control register. (R/W)

19 Pulse Count Controller (PCNT)

The pulse count controller (PCNT) is designed to count input pulses. 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. There is only one clock in PCNT, which is APB_CLK. 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 19-1, each channel has two input signals:

1. One input pulse signal (e.g. sig_ch0_u*n*, the input pulse signal for ch0 of unit *n* ch0)
2. One control signal (e.g. ctrl_ch0_u*n*, the control signal for ch0 of unit *n* ch0)

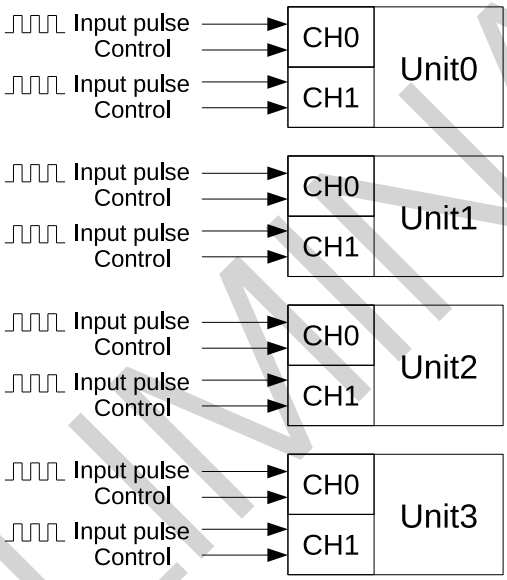


Figure 19-1. PCNT Block Diagram

19.1 Features

A PCNT has the following features:

- Four independent pulse counters (units) that count from 1 to 65535
- Each unit consists of two independent channels sharing one pulse counter
- All channels have input pulse signals (e.g. sig_ch0_u*n*) with their corresponding control signals (e.g. ctrl_ch0_u*n*)
- Independently filter glitches of input pulse signals (sig_ch0_u*n* and sig_ch1_u*n*) and control signals (ctrl_ch0_u*n* and ctrl_ch1_u*n*) on each unit
- Each channel has the following parameters:
  1. Selection between counting on 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

19.2 Functional Description

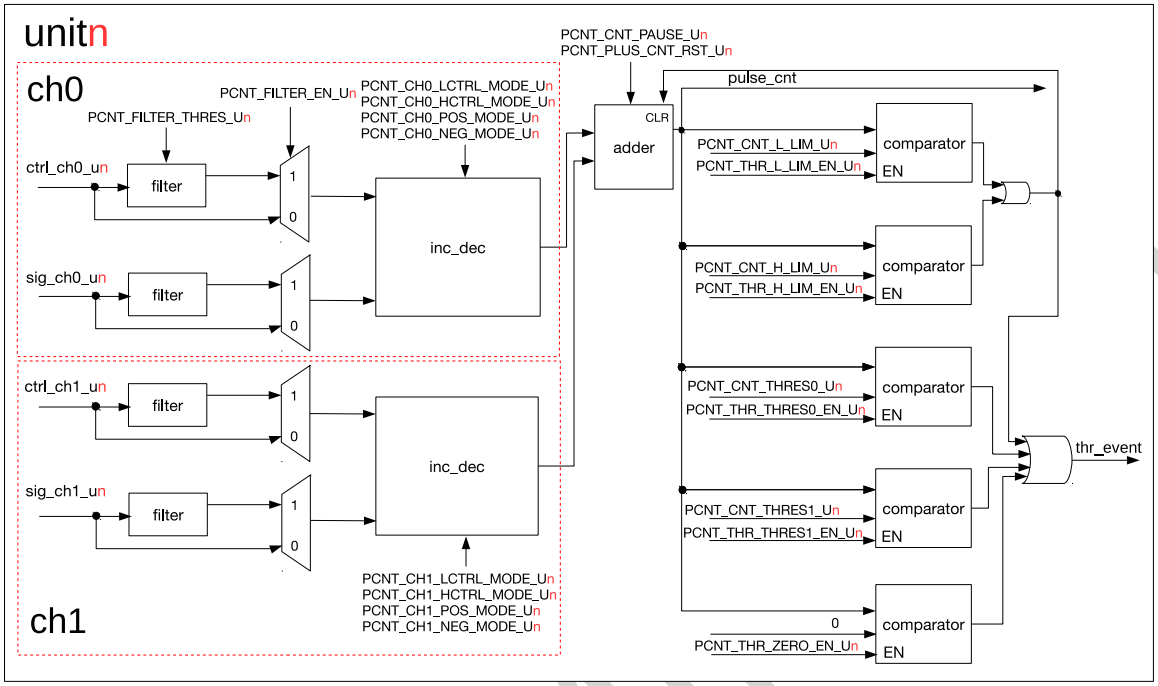


Figure 19-2. PCNT Unit Architecture

Figure 19-2 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* (can be configured by software), the counter value pulse_cnt increases by 1. Upon reaching PCNT_CNT_H_LIM_U*n*, pulse_cnt is cleared. If 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* (can be configured by software), the counter value pulse_cnt decreases by 1. Upon reaching PCNT_CNT_L_LIM_U*n*, pulse_cnt is cleared. If 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 19-1 to Table 19-4 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 19-1. 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 19-2. 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 19-3. 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 19-4. 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 which has a 16-bit wide signed register. 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*, a high limit interrupt is triggered and PCNT_CNT_THR_H_LIM_LAT_U*n* is high.
- Minimum count value: When pulse_cnt reaches PCNT_CNT_L_LIM_U*n*, a low limit interrupt is triggered and PCNT_CNT_THR_L_LIM_LAT_U*n* is high.
- Two threshold values: When pulse_cnt equals either PCNT_CNT_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.

19.3 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.

19.3.1 Channel 0 Incrementing Independently

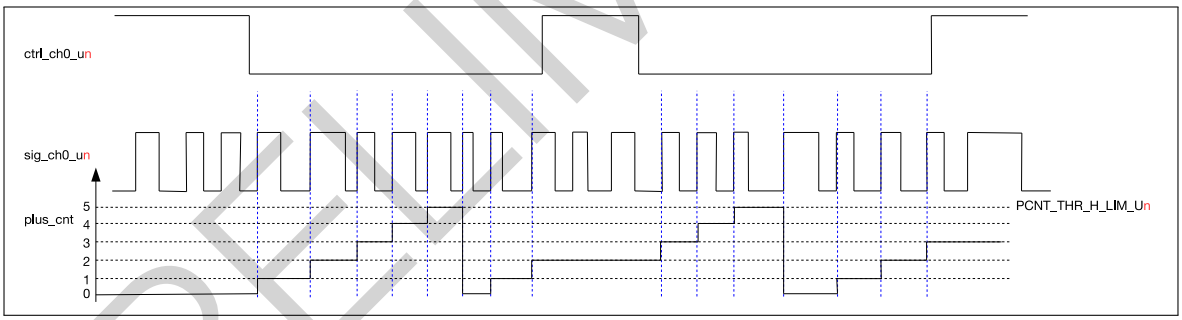


Figure 19-3. Channel 0 Up Counting Diagram

Figure 19-3 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 19.2 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.

19.3.2 Channel 0 Decrementing Independently

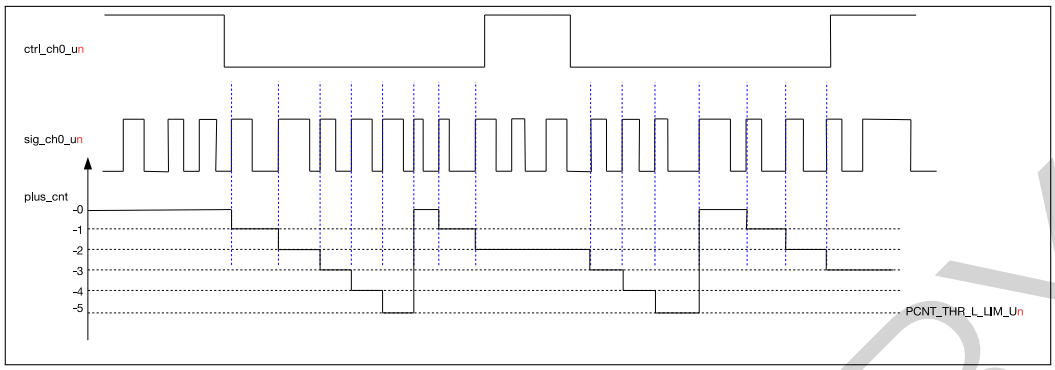


Figure 19-4. Channel 0 Down Counting Diagram

Figure 19-4 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 19-3 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.

19.3.3 Channel 0 and Channel 1 Incrementing Together

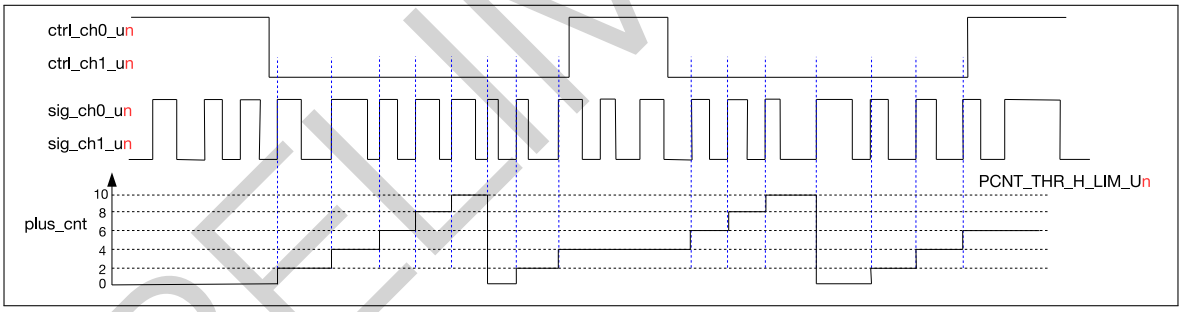


Figure 19-5. Two Channels Up Counting Diagram

Figure 19-5 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 19-5 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.

19.4 Register Summary

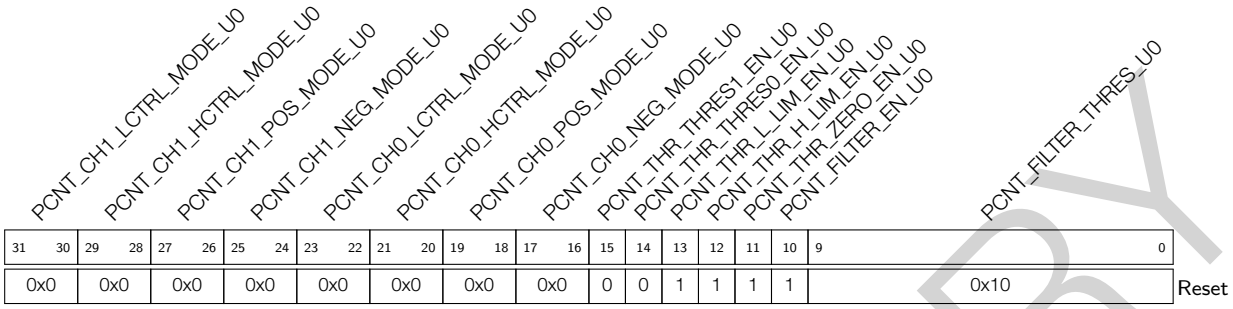
The addresses in this section are relative to Pulse Count Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Name	Description	Address	Access
Configuration Register			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x0000	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x0004	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x0008	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x000C	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x0010	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x0014	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x0018	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x001C	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x0020	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x0024	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x0028	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x002C	R/W
PCNT_CTRL_REG	Control register for all counters	0x0060	R/W
Status Register			
PCNT_U0_CNT_REG	Counter value for unit 0	0x0030	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x0034	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x0038	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x003C	RO
PCNT_U0_STATUS_REG	PNCT UNIT0 status register	0x0050	RO
PCNT_U1_STATUS_REG	PNCT UNIT1 status register	0x0054	RO
PCNT_U2_STATUS_REG	PNCT UNIT2 status register	0x0058	RO
PCNT_U3_STATUS_REG	PNCT UNIT3 status register	0x005C	RO
Interrupt Register			
PCNT_INT_RAW_REG	Interrupt raw status register	0x0040	RO
PCNT_INT_ST_REG	Interrupt status register	0x0044	RO
PCNT_INT_ENA_REG	Interrupt enable register	0x0048	R/W
PCNT_INT_CLR_REG	Interrupt clear register	0x004C	WO
Version Register			
PCNT_DATE_REG	PCNT version control register	0x00FC	R/W

19.5 Registers

The addresses in this section are relative to Pulse Count Controller base address provided in Table 1-4 in Chapter 1 *System and Memory*.

Register 19.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xCn*)**



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)

Continued on the next page...

Register 19.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC*n*)

Continued from the previous page...

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)

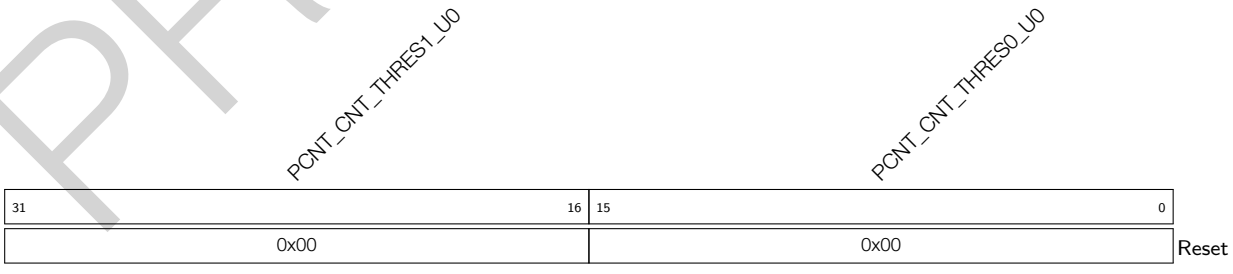
PCNT_CH1_HCTRL_MODE_U*n* This register configures how the CH*n*_POS_MODE/CH*n*_NEG_MODE settings will be modified when the control signal is high.

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)

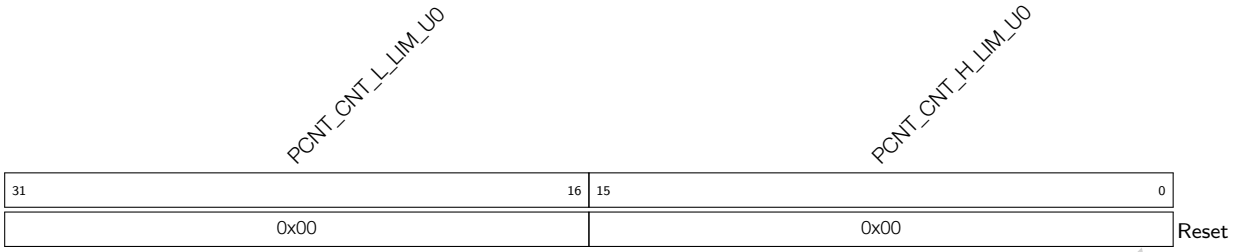
Register 19.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xC*n*)



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)

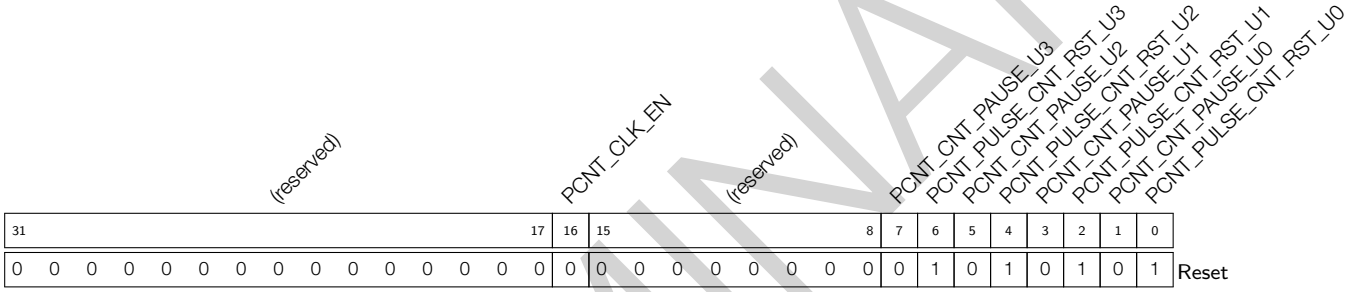
Register 19.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xCn*)**



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 19.4. PCNT_CTRL_REG (0x0060)

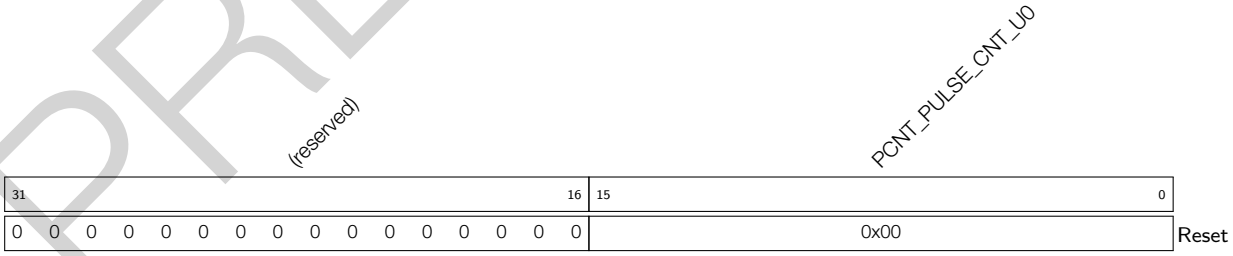


PCNT_PULSE_CNT_RST_U*n* Set this bit to clear unit *n*'s counter. (R/W)

PCNT_CNT_PAUSE_U*n* Set this bit to freeze unit *n*'s counter. (R/W)

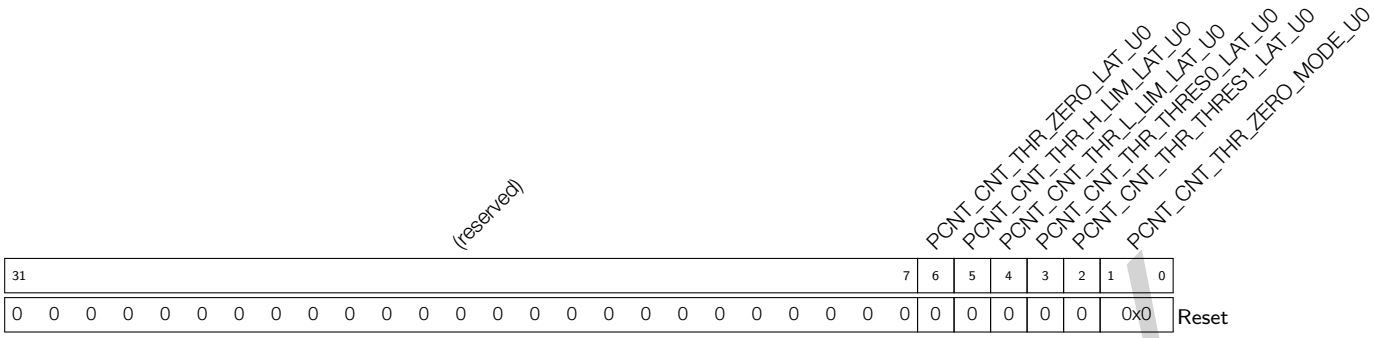
PCNT_CLK_EN The registers clock gate enable signal of PCNT module. 1: the registers can be read and written by application. 0: the registers can not be read or written by application (R/W)

Register 19.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4n*)**



PCNT_PULSE_CNT_U*n* This register stores the current pulse count value for unit *n*. (RO)

Register 19.6. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4n*)**



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)

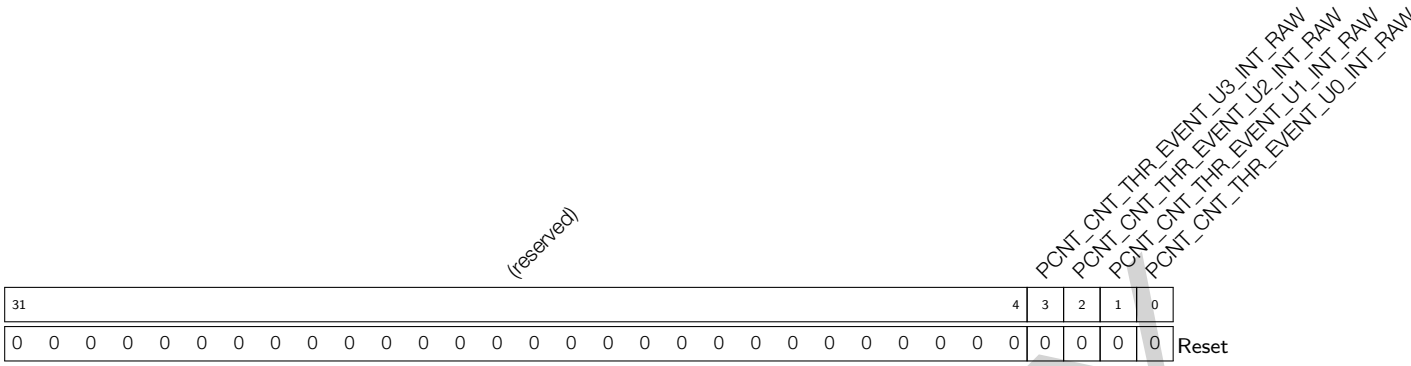
PCNT_CNT_THR_THRES0_LAT_U*n* The latched value of thres0 event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to thres0 and thres0 event is valid. 0: others (RO)

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)

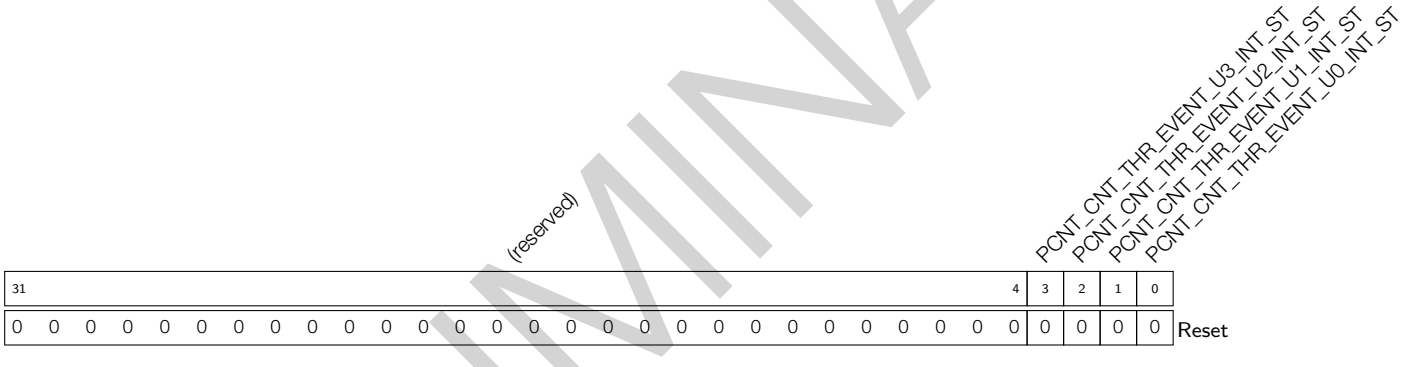
PCNT_CNT_THR_ZERO_LAT_U*n* The latched value of zero threshold event of PCNT_U*n* when threshold event interrupt is valid. 1: the current pulse counter equals to 0 and zero threshold event is valid. 0: others (RO)

Register 19.7. PCNT_INT_RAW_REG (0x0040)



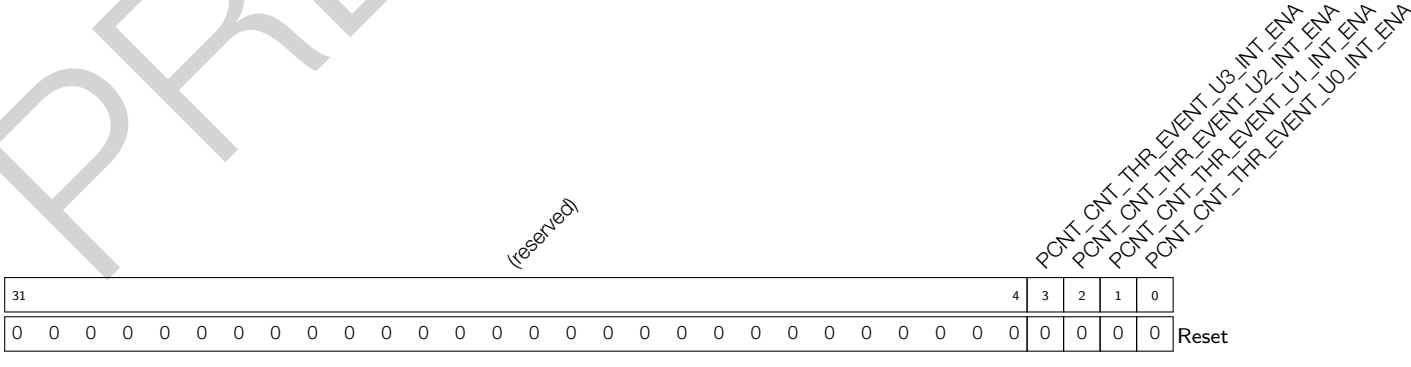
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 19.8. PCNT_INT_ST_REG (0x0044)



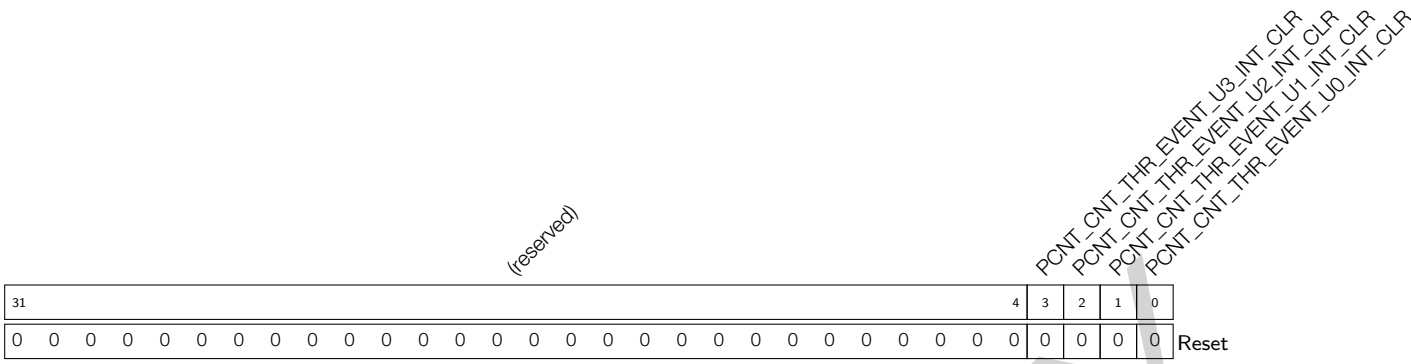
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 19.9. PCNT_INT_ENA_REG (0x0048)



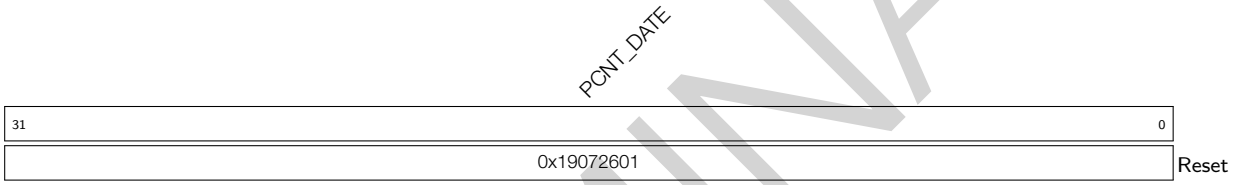
PCNT_CNT_THR_EVENT_U*n*_INT_ENA The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 19.10. 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)

Register 19.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE This is the PCNT version control register. (R/W)

Glossary

Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED Control PWM (Pulse Width Modulation)
MCPWM	Motor Control PWM (Pulse Width Modulation)
PCNT	Pulse Count Controller
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

Abbreviations for Registers

ISO	Isolation. When a module is power down, its output pins will be stuck in unknown state (some middle voltage). "ISO" registers will control to isolate its output pins to be a determined value, so it will not affect the status of other working modules which are not power down.
NMI	Non-maskable interrupt.
REG	Register.
R/W	Read/write. Software can read and write to these bits.
RO	Read-only. Software can only read these bits.
SYSREG	System Registers
WO	Write-only. Software can only write to these bits.

Revision History

Date	Version	Release notes
2021-07-09	V0.1	Preliminary release

www.espressif.com

Disclaimer and Copyright Notice